Automatic Synthesis of Generalized Winning Strategies of Impartial Combinatorial Games Using SMT Solvers

Kaisheng Wu[1], **Liangda Fang**[1,3*], **Liping Xiong**[2], **Zhao-Rong Lai**[1]
Yong Qiao[1], **Kaidong Chen**[1] and **Fei Rong**[1]
[1]Department of Computer Science, Jinan University, China
[2]Department of Computer Science, South China Normal University, China
[3]Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, China
mamduh0519@stu2016.jnu.edu.cn, fangld@jnu.edu.cn

Abstract

Strategy representation and reasoning has recently received much attention in artificial intelligence. Impartial combinatorial games (ICGs) are a type of elementary and fundamental games in game theory. One of the challenging problems of ICGs is to construct winning strategies, particularly, generalized winning strategies for possibly infinitely many instances of ICGs. In this paper, we investigate synthesizing generalized winning strategies for ICGs. To this end, we first propose a logical framework to formalize ICGs based on the linear integer arithmetic fragment of numeric part of PDDL. We then propose an approach to generating the winning formula that exactly captures the states in which the player can force to win. Furthermore, we compute winning strategies for ICGs based on the winning formula. Experimental results on several games demonstrate the effectiveness of our approach.

1 Introduction

Strategy representation and reasoning has recently received much attention in artificial intelligence, particularly, multi-agent systems and game theory [Chatterjee *et al.*, 2010; Zhang and Thielscher, 2015; Xiong and Liu, 2016]. In the area of game theory, one class of the elementary and fundamental games is impartial combinatorial games (ICGs) where two players alternate moving with perfect information [Ferguson, 2018]. Various ICGs (*e.g.*, Nim [Bouton, 1901], Wythoff [Wythoff, 1907], Chomp [Schuh, 1952] and Empty-and-Divide [Ferguson, 1998]), and their variants and extensions [Liu and Zhao, 2015; Ahn *et al.*, 2017] have been proposed and analyzed. The following is the 2-rowed Chomp game.

Example 1. Cookies are laid out on 2 rows each of which contains an arbitrary number of cells. The cookie in the leftmost-topmost position $(1, 1)$ is poisoned. At each move, a player has to eat a remaining cookie, together with all cookies to the right and above it. The loser is the player who has no choice but to eat the poisoned cookie.

[*]Corresponding author

One of the challenging problems of ICGs is to synthesize the winning strategy for one player. Given an instance, that is, an ICG together with an initial state, one winning strategy can be computed via the backward search algorithm [de Jonge and Zhang, 2016]. However, this strategy holds for only some instances but not (infinitely) many instances. This is illustrated in the 2-rowed Chomp game as follows. Assume that the number v_1 and v_2 of cookies at the first and second row are both initially 2. In the first round, the first player eats the cookie at $(2, 2)$. If the other player eats the cookie at $(1, 2)$ (resp. $(2, 1)$), then she chooses to eat the cookie at $(2, 1)$ (resp. $(1, 2)$) in the next round. Following the above strategy, the first player finally wins no matter how the other player does. But for another instance where the initial state is $v_1 = 3 \wedge v_2 = 1$, it is impossible that the first player eats the cookie at $(2, 2)$ since no cookie is at such position. It has been proved that the first player can force to win in all instances where $v_1 \neq v_2 + 1$ initially [Zeilberger, 2001]. Hence, the above strategy does not work for some of these instances.

Synthesizing a generalized winning strategy that works for possibly infinitely many instances is of utmost importance. However, this problem is notoriously difficult even for simple games, and undecidable in general [Luo and Liu, 2019].

In this paper, we concentrate on ICGs, and investigate synthesizing generalized winning strategies. The contributions of this paper are as follows: (1) We propose a logical framework to formalize ICGs via the linear integer arithmetic fragment of Planning Domain Definition Language (PDDL) [Fox and Long, 2003]. (2) We propose an approach to synthesizing the arithmetic formula, called the winning formula, which exactly captures the states in which the player forces to win. To do this, we propose three constraints for the winning formula, then the synthesis of the winning formula is reduced to searching an arithmetic formula that is consistent with these constraints. We implement the searching process via adapting the enumerative approach proposed in [Udupa *et al.*, 2013]. (3) We give a method to synthesize generalized winning strategies for impartial combinatorial games. To this end, we divide the winning formula into several arithmetic formulas through a series of syntactic operations. For each formula, we then choose a suitable action together with its parameters via the enumerative approach. (4) Finally, we evaluate our approach on several games, and experimental results demonstrate the effectiveness of the proposed approach.

2 Impartial Combinatorial Games

In this section, we briefly introduce *impartial combinatorial games (ICGs)*. A game that satisfies the following conditions is called an ICG [Ferguson, 2018]: (1) There are two players and many states such that the player can move from one state to another one. (2) Two players alternate moving and have the same choice of moving. (3) The game ends when it moves to an *ending state* in which no player has a possible move. (4) The game always ends in a finite number of moves, and the last player to move wins. An *instance* of an ICG is the ICG together with an initial state. In general, an ICG may contain infinite states, and hence it may have infinite instances.

Each instance of an ICG is a finite two-player game with perfect information and no-chance move where the players take turns moving. In addition, this game does not end in a draw. By Zermelo's Theorem [Zermelo, 1913], there always exists a winning strategy for one player in an ICG.

It can be easily observed that the player cannot guarantee to win in all states. To distinguish states in which the player can forces to win from those in which she cannot, we classify states into two types: *winning* and *losing states*.

Definition 1 ([Ferguson, 2018])**.** In an ICG, winning and losing states are recursively defined as follows:

1. All ending states are losing states.
2. All states such that there is at least one move to a losing state are winning states.
3. All states such that the only possible moves are to winning states are losing states.

We hereafter introduce the concept of *strategy*. ICGs are a special case of Gale-Stewart games, and so are determined by memoryless strategies [Gale and Stewart, 1953]. In addition, it is impossible to obtain a winning strategy for the player in a losing state, there is no need to consider how to choose a move in this case. We hence define a *strategy* as a function from winning states to moves.

3 A Logical Framework

In this section, we present a logical framework for describing ICGs. The framework we adopt is a slight modification of the linear integer arithmetic fragment of the PDDL 2.1 language defined in [Fox and Long, 2003].

Traditionally, arithmetic formulas allow the occurrence of multiplication. In this paper, we use the decidable fragment: linear integer arithmetic (LIA). An arithmetic formula is *LIA-definable* if it is defined by LIA. The syntax of LIA is defined as follows. Let $\mathcal{N}$ be the set of integers, $\mathcal{V}$ a set of variables and $\mathcal{X} \subseteq \mathcal{V}$ a finite set of state variables. The sets of *arithmetic expressions* (Exp), *literals* (Lit) and *formulas* (Form) is defined by the following grammar:

$$e, e' \in \texttt{Exp} :: c \mid v \mid e + e' \mid e - e'$$

$$l \in \texttt{Lit} :: e = e' \mid e \neq e' \mid e < e' \mid e > e' \mid e \leq e' \mid e \geq e' \mid e \equiv_{c'} c \mid e \not\equiv_{c'} c$$

$$\phi, \phi' \in \texttt{Form} :: l \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid \forall v \phi \mid \exists v \phi$$

where $c, c' \in \mathcal{N}$ and $v \in \mathcal{V}$.

We remark that formulas defined by the above syntax are negation-free. For the sake of simplification, when we use the negation symbol $\neg\phi$, we refer to the negation-free formula equivalent to $\neg\phi$, obtained in the obvious way. The literal $e \equiv_{c'} c$ denotes that $e - c$ are divisible by c', and its negation is $e \not\equiv_{c} c'$. We use $|\phi|$ (resp. $|e|$) for the size of an arithmetic formula ϕ (resp. an arithmetic expression e). It is well-known that LIA admits quantifier elimination, *i.e.*, any arithmetic formula can be equivalently transformed into a quantifier-free one [Cooper, 1972; Monniaux, 2010]. We use $\texttt{Form}^{\texttt{qf}}$ for the set of quantifier-free formulas (qf-formulas). We use $\texttt{Exp}_{\mathcal{X}}$ for the set of expressions over $\mathcal{X}$, and say e is *semi-ground* if $e \in \texttt{Exp}_{\mathcal{X}}$. We use $\texttt{Form}_{\mathcal{X}}$ for the set of formulas of which free variables are state variables, and say ϕ is *semi-ground* if $\phi \in \texttt{Form}_{\mathcal{X}}$. The notation $\texttt{Form}^{\texttt{qf}}_{\mathcal{X}}$ is similar. We say an arithmetic formula is an *arithmetic term*, if it is a conjunction of arithmetic literals.

A *state* s is a valuation function $\mathcal{X} \to \mathcal{N}$. Given a state s, we evaluate a semi-ground expression e into an integer $e(s)$ to which the expression simplifies when substituting every state variable v with their respective value $s(v)$. The Boolean value $\phi(s)$ of a semi-ground formula ϕ can be determined in a similar way. We say a state s *satisfies* a formula ϕ, denoted by $s \models \phi$, if $\phi(s) = \top$. We say a formula ϕ *entails* another one ϕ', denoted by $\phi \models \phi'$, if for all states s s.t. $s \models \phi$, we have $s \models \phi'$. A *numeric effect* is a triple $\langle \phi, v, e \rangle$ where $\phi \in \texttt{Form}$, $v \in \mathcal{X}$, and $e \in \texttt{Exp}$. Intuitively, it means that if ϕ holds in a state s, then the value of v becomes $e(s)$ after performing the action; otherwise, it remains unchanged.

Definition 2. An ICG is defined as a tuple $\Pi = \langle \mathcal{X}, \mathcal{A}, \mathcal{C}, \mathcal{E} \rangle$ where

- $\mathcal{X}$: a finite set of state variables.
- $\mathcal{A}$: a finite set of actions. Each action $a(\mathcal{Y}) \in \mathcal{A}$ can be parameterized by a vector $\mathcal{Y}$ of variables. An action $a(\mathcal{Y})$ is defined by a tuple $\langle \texttt{pre}, \texttt{eff} \rangle$ where $\texttt{pre} \in \texttt{Form}$ denotes the precondition, and eff is the effect represented by a set of numeric effects.
- $\mathcal{C}$: a formula of $\texttt{Form}_{\mathcal{X}}$ denoting all legal states.
- $\mathcal{E}$: a formula of $\texttt{Form}_{\mathcal{X}}$ denoting all ending states.

Let Σ be a vector of semi-ground expressions. A semi-ground action $a[\mathcal{Y}/\Sigma]$ is an action whose the i-th parameter y_i of $\mathcal{Y}$ is replaced by the i-th semi-ground expression e_i of Σ. For simplicity, we sometimes use a to denote semi-ground actions. Given a semi-ground action a, we say a is *applicable* over a state s, if $s \models \texttt{pre}(a)$. The result of applying an applicable action a over s is a state, written $do(a, s)$, which results from s by replacing $s(v)$ with $e(s)$ when $s \models \phi$ for all $\langle \phi, v, e \rangle \in \texttt{eff}(a)$.

Example 2. The formalization of the 2-rowed Chomp game is given as follows. Two state variables v_1 and v_2 denote the number of cookies in the first and second row respectively. Any legal state is such that (1) the first row contains at least as many cookies as the second one, (2) the first row contains at least one cookie, and (3) the number of the cookies in the second row is non-negative (*i.e.*, $\mathcal{C} = v_1 \geq v_2 \wedge v_1 \geq 1 \wedge v_2 \geq 0$). The ending condition $\mathcal{E}$ is $v_1 = 1 \wedge v_2 = 0$. The action $eat1(k)$ (resp. $eat2(k)$) means eating the cookie at $(1, k)$ (resp. $(2, k)$) together with all cookies to the right and/or above it. The following are descriptions of actions.

1. $\mathtt{pre}(eat1(k)) = v_1 \geq k \wedge k > 1$;
2. $\mathtt{pre}(eat2(k)) = v_2 \geq k \wedge k > 0$;
3. $\mathtt{eff}(eat1(k)) = \{\langle \top, v_1, k-1 \rangle, \langle v_2 \geq k, v_2, k-1 \rangle\}$;
4. $\mathtt{eff}(eat2(k)) = \{\langle \top, v_2, k-1 \rangle\}$. □

To obtain the finite representation of winning states, we use an arithmetic formula to exactly capture this notion, called *the winning formula*.

Definition 3. Let $\Pi = \langle \mathcal{X}, \mathcal{A}, \mathcal{C}, \mathcal{E} \rangle$ be an ICG and $\phi \in \mathtt{Form}_{\mathcal{X}}$. We call ϕ a winning formula of Π, if

1. For any ending state s, we have $s \models \neg\phi$;
2. For any winning state s, there is an action a s.t. a is applicable in s and $do(a, s) \models \neg\phi$;
3. For any losing state s and action a where a is applicable in s, we have $do(a, s) \models \phi$.

For an ICG, the winning formula is unique up to equivalence as the set of winning states is unique. Since the sets of winning states and losing states are disjoint, $\neg\phi$ is the symbolic representation of losing states. Each condition illustrated in the definition of winning formulas corresponds to each rule given in the definition of winning states (cf. Definition 1). In this paper, we restrict winning formulas to be LIA-definable. The winning formulas for some games are not LIA-definable. For example, the winning formula of the Wythoff game [Wythoff, 1907] is $v_1 \neq \lfloor g \cdot n \rfloor \vee v_2 \neq \lfloor g^2 \cdot n \rfloor$ for any positive integer n where g is the Golden ratio $\frac{1+\sqrt{5}}{2}$ and $\lfloor r \rfloor$ is the integer part of the real number r. We leave the investigation of synthesizing winning formulas for these games requiring more expressive language to future work.

We hereafter define winning strategies as a set of pairs of semi-ground formulas and actions.

Definition 4. Let $\Pi = \langle \mathcal{X}, \mathcal{A}, \mathcal{C}, \mathcal{E} \rangle$ be an ICG, and ϕ a winning formula of Π. A winning strategy δ is a set of pairs $\{(\psi_1, a_1), \cdots, (\psi_n, a_n)\}$ where each $\psi_i \in \mathtt{Form}_{\mathcal{X}}$ and a_i is a semi-ground action, if

1. $\bigvee_{i=1}^{n} \psi_i \equiv \phi$;
2. $\psi_i \not\models \psi_j$ for $i \neq j$.
3. For any winning state s and $a \in \delta(s)$, we have $do(a, s)$ is a losing state.

Here, $\delta(s) = \{a_i \mid s \models \psi_i\}$.

Conditions 1 and 2 together guarantee that the formulas ψ_i's form a cover of the winning formula ϕ. That is, (1) ϕ is covered by all of the ψ_i's; and (2) any formula ψ_i can not entail another one ψ_j. Condition 3 requires the successor state, resulting from a winning state by executing the action according to the strategy, be a losing state. Condition 3 together with the fact that $\mathcal{E} \models \neg\phi$ (Item 1 of Definition 3) implies that the strategy δ defined in Definition 4 is winning.

4 Synthesis of Winning Strategies

In this section, we provide a synthesis approach to winning strategies. Our approach consists of three steps: (1) *synthesizing the winning formula* (Sec. 4.2), (2) *refining the winning formula* (Sec. 4.3), and (3) *synthesizing the winning strategy* (Sec. 4.4). The first and third steps are based on the enumerative algorithm proposed in [Udupa *et al.*, 2013] (Sec. 4.1).

Algorithm 1: $\mathtt{Synthesize}(\Theta)$

```
Input: $\Theta : (\theta_1, \cdots, \theta_m)$: a set of specifications.
Output: $\Omega : (obj_1, \cdots, obj_n)$: a vector of expression and
        formula satisfying $\Theta$.
1  Initialize the set $E$ of examples
2  while $true$ do
3      $\Omega \leftarrow \mathtt{Enumerate}(E)$
4      $sat \leftarrow true$
5      $(in, sat) \leftarrow \mathtt{Verifiy}(\Omega, \Theta)$
6      if $sat = false$ then /* The specification is
        not satisfied */
7          Compute the output $(out_1, \cdots, out_n)$ according to
            the input $in$
8          $e \leftarrow (in, (out_1, \cdots, out_n))$
9          $E \leftarrow E \cup e$
10     else                      /* $\Omega$ is correct */
11         return $\Omega$
```

4.1 The enumerative algorithm for multiple objects

The task of the original enumerative algorithm is to synthesize a single object satisfying a set of specifications [Udupa *et al.*, 2013]. The object is an expression or a formula, and each specification is expressed by an formula. In this paper, we need to consider identifying a vector of expressions and formulas. We hereafter extend the original algorithm to be suitable for multiple objects.

As Algorithm 1 illustrates, the algorithm aims to synthesize n objects Ω satisfying the specifications Θ. It maintains a set E of examples. An example e consists of an input $in(e)$ and an output $out(e)$. The input is a state while the output is a vector of Boolean values and/or integers with the same length of the vector of objects. The i-th element of the output is a Boolean value (resp. integer) if the i-th object is a formula (resp. expression). The main idea of this algorithm is to iteratively enumerate candidates consistent with the set E of examples, and then return the correct ones via verifying them against the set Θ of specifications.

The enumeration process generates the candidate expressions (or formulas) by induction on size according to the grammar illustrated in Section 3. In the first step, only arithmetic expressions of size 1 (*i.e.*, state variables and integers) are considered. As the set of integers is infinite, the integers considered in the first step are 0, 1 together with the constants explicitly occurring in the formalization of games. Firstly, this restriction does not affect visiting all integers since any positive integer c can be represented by the addition of c 1's, and the negative integer $-c$ equals to $0 - c$. Furthermore, we observe that some integers occurring in the winning formula and winning strategy are close to some explicit constants of appearing in the formalization. Hence, supplementing these constants can accelerate the synthesis process. In the k-th step, we generate arithmetic expressions and formulas of size k. The enumeration process terminates when it finds candidates that are consistent with the set of examples. Formally, a vector of objects $(obj_1, \cdots, obj_n)$ is *consistent* with a set E of examples, if for any $1 \leq i \leq n$ and any example $e \in E$, we have $obj_i(in(e)) = out_i(e)$.

To improve the efficiency of the enumeration process, we remove expressions and formulas based on the notion of indistinguishability. Formally, two formulas ϕ_1 and ϕ_2 are *indistinguishable* w.r.t. a set S of states, if $\phi_1(s) = \phi_2(s)$ for every $s \in S$. The definition of indistinguishability between two expressions is similar. For example, suppose that $S = \{(v_1 : 1, v_2 : 2), (v_1 : 2, v_2 : 2)\}$. It is easily verified that the two formulas $v_1 \neq v_2$ and $v_1 + 1 = v_2$ hold in the first state and does not hold in the second one. Therefore, the two formulas are indistinguishable w.r.t. S. Suppose that we are generating expressions and formulas of size k. Let I be the collection of the inputs of the set of examples $e \in E$. If ϕ_1 and ϕ_2 are indistinguishable w.r.t. I, then it is sufficient to carry forward only one of ϕ_1 and ϕ_2 to the next iteration $k + 1$. Based on the notion of indistinguishability, we can prune the search space of candidates.

When desired candidates are generated by the enumeration process, they will be sent to the verification process for checking against the correctness of given specifications. The verification process can be executed by an SMT solver. If the candidates are correct, then we succeed in finding the correct expressions and formulas. Otherwise, the SMT solver returns a state that distinguishes the correct expressions and formulas and the candidates generated by the enumeration process. For each state s, we construct a new example e with the state and a suitable output, and enlarge the set E of example by e. Then, the enumeration process restarts. The whole enumerative algorithm repeats until it searches correct candidates.

The enumerative algorithm is a general approach to synthesizing multiple objects. To adopt this algorithm, three factors should be designed: (1) what is the type of each object and the constraints for objects, (2) how to initialize the set E of examples (Line 1), and (3) how to compute the output for the state generated by the SMT solver (Line 7). The details will be illustrated in Sections 4.2 and 4.4.

4.2 Synthesizing the winning formula

In this subsection, we introduce the idea of synthesizing the winning formula via adopting the enumerative algorithm.

Firstly, the search object is a semi-ground qf-formula. Each ouput of all examples of E contains a single Boolean value. Secondly, the set E initially contains only examples whose inputs are ending states. The output is $\bot$ as each ending state is a losing state. This can be done by finding states satisfying the ending condition $\mathcal{E}$ via the SMT solver. Thirdly, the Boolean value of a state s, denoting if s is a winning state is determined by a recursive way according to the definition of winning and losing states (cf. Definition 1).

In the following, we introduce the notion of *transition formulas* and *global transition formulas* that are ingredient components of the constraints. The following construction of transition formulas is a numerical generalization of the existing boolean encoding for the planning problem which focuses on transition systems over finite states [Hoffmann and Brafman, 2006; Rintanen, 2012]. For an action a, the transition formula of a reflects the relation between predecessor and successor states. In order to express these two notions, we consider two versions v and v' for all $v \in \mathcal{X}$. The unprimed version v denotes the state variable before performing an action, and the primed one v' denotes the one after.

Definition 5. Let a be an action. The transition formula $\mathcal{T}(a)$ of a is the conjunction of the following formulas:

1. $\texttt{pre}(a)$;
2. $\bigwedge_{\langle\phi,v,e\rangle\in\texttt{eff}(a)}(\phi \to v' = e)$;
3. $\bigwedge_{v\in\mathcal{V}}[(\bigvee_{\langle\phi,v,e\rangle\in\texttt{eff}(a)}\phi) \vee v' = v]$.

The first formula is the precondition of action a. The second one is the effect axiom meaning that the value of v becomes $e(s)$ if the condition ϕ holds in the state s. The third formulas is the frame axiom meaning that if no condition ϕ s.t. $\langle\phi, v, e\rangle \in \texttt{eff}(a)$ holds, then the value of v remains unchanged.

The global transition formula denotes the state transition system of the whole game. The definition is as follows:

Definition 6. Let $\mathcal{A}$ be a set of actions. The global transition formula $\mathcal{T}(\mathcal{A})$ of Π is $\bigvee_{a(\mathcal{Y})\in\mathcal{A}} \exists\mathcal{Y}[\mathcal{T}(a(\mathcal{Y}))]$.

The following example illustrates the construction of transition formulas from the definition of actions.

Example 3. Let us continue the 2-rowed Chomp game. We construct the transition formula $\mathcal{T}(eat1(k))$ as follows. The precondition is $v_1 \geq k \wedge k > 1$. The effects contain two elements: $\langle\top, v_1, k-1\rangle$ and $\langle v_2 \geq k, v_2, k-1\rangle$. Hence, Formula 2 of $\mathcal{T}(eat1(k))$ is $(\top \to v_1' = k-1) \wedge (v_2 \geq k \to v_2' = k-1)$. We now consider Formula 3 of $\mathcal{T}(eat1(k))$. For the variable v_1, the formula is $\top \vee v_1' = v_1$ as v_1 only occurs in the first element of $\texttt{eff}(eat1(k))$. Similarly, we obtain the formula $v_2 \geq k \vee v_2' = v_2$ for the variable v_2. Conjoining the above formulas and simplifying the conjunction lead to the transition formula $\mathcal{T}(eat1(k)) \equiv (v_1 \geq k \wedge k > 1) \wedge (v_1' = k-1) \wedge (v_2 \geq k \to v_2' = k-1) \wedge (v_2 \geq k \vee v_2' = v_2)$.

Similarly, we get that $\mathcal{T}(eat2(k)) \equiv v_2 \geq k \wedge k > 0 \wedge v_2' = k-1 \wedge v_1' = v_1$. By Definition 6, the global transition formula $\mathcal{T}(\mathcal{A}) \equiv \exists k[\mathcal{T}(eat1(k)) \vee (\mathcal{T}(eat2(k))]$. $\square$

With the global transition formula in hand, we are ready to give the constraints for the winning formula.

Definition 7. Let $\Pi = \langle\mathcal{X}, \mathcal{A}, \mathcal{C}, \mathcal{E}\rangle$ be an ICG. The constraints for the winning formula ϕ of Π are as follows:

1. $\mathcal{E} \to \neg\phi$;
2. $(\mathcal{C} \wedge \phi) \to \exists\mathcal{X}'[\mathcal{T}(\mathcal{A}) \wedge \neg\phi[\mathcal{X}/\mathcal{X}']]$;
3. $(\mathcal{C} \wedge \neg\phi) \to \forall\mathcal{X}'[\mathcal{T}(\mathcal{A}) \to \phi[\mathcal{X}/\mathcal{X}']]$.

where $\phi[\mathcal{X}/\mathcal{X}']$ is the formula obtained by replacing every occurrence of $v \in \mathcal{X}$ in ϕ with v'.

The three constraints correspond to Items 1 - 3 of Definition 3 respectively. When the enumeration process generates a candidate ϕ, we check if the constraint is valid via the SMT solver. If it is the case, then ϕ is the winning formula. Otherwise, the SMT solver returns a state s that can be used to construct a new example e. However, the state s may be not suitable to improve the candidate ϕ. Suppose that s is a winning state and $s \models \phi$. The example e we construct is $(s, \top)$. In this case, even if we add this example into E, the new candidate may be equivalent to the previous one. This phenomenon may also happen when s is a losing state and

Candidates of winning formulas	Examples	
	Inputs	Outputs
-	$(v_1:1, v_2:0)$	$\bot$
$v_1 \neq v_1$	$(v_1:1, v_2:1)$	$\top$
$v_2 \neq 0$	$(v_1:2, v_2:1)$	$\bot$
$v_1 = v_2$	$(v_1:2, v_2:2)$	$\top$
$v_1 = v_2$	$(v_1:3, v_2:1)$	$\top$
$v_1 \neq v_2 + 1$	-	-

Table 1: A run of synthesizing the winning formula

$s \models \neg\phi$. To repair this defect, we enumerate states in an increasing order of the sum of values of all state variables until we find a state s matches the two requirements: (1) the state s is not the input of an existing example (*i.e.*, $s \neq in(e)$ for all $e \in E$); (2) s is a winning state and $s \models \neg\phi$, or s is a losing state and $s \models \phi$. Finally, we add the example $(s, (out))$, where out denotes if s is a winning state, into E, and the enumeration process restarts.

We show the working of the process of synthesizing the winning formula on the 2-rowed Chomp game.

Example 4. Table 1 shows the candidates constructed by the enumeration process and the corresponding examples obtained from the verification process. Initially, the set E contains an example $((v_1 : 1, v_2 : 0), (\bot))$. The enumeration process constructs a candidate $v_1 \neq v_1$, and sends it to the verification process. The latter finds that $v_1 \neq v_1$ is not the winning formula, and enlarges the set E by $((v_1 : 1, v_2 : 1), (\top))$. The enumeration process then submits the formulas $v_2 \neq 0$ and $v_1 = v_2$ to the verification process. The SMT solver returns a losing state $(v_1 : 2, v_2 : 1)$ and a winning state $(v_1 : 2, v_2 : 2)$, respectively, as counterexamples to the above two formulas. Since the winning state $(v_1 : 2, v_2 : 2)$ satisfies the formula $v_1 = v_2$, it is a state that is not suitable to construct an example. At this iteration, we generate a winning state $(v_1 : 3, v_2 : 1)$ that falsifies the candidate $v_1 = v_2$. Finally, the enumeration process succeeds in finding the winning formula $v_1 \neq v_2 + 1$ which is certified by the verification process, and the whole algorithm terminates.

Next, we state two important properties of the enumerative algorithm to synthesize the winning formula.

Theorem 1. *Let* $\Pi = \langle \mathcal{X}, \mathcal{A}, \mathcal{C}, \mathcal{E} \rangle$ *be an ICG.*

Soundness *If Algorithm 1 synthesizes the formula ϕ satisfying the constraints illustrated in Definition 7, then ϕ is the winning formula of* Π.

Relatively Completeness *If the winning formula of Π is LIA-definable, then Algorithm 1 terminates with a winning formula.*

Proof. **Soundness** Suppose that ϕ is the formula that Algorithm 1 returns. It satisfies the constraints illustrated in Definition 7. It follows that ϕ satisfies the requirements illustrated in the definition of winning formulas.

Completeness Suppose that ϕ is the winning formula of Π with the smallest size. According to Algorithm 1, the enumeration process generates the formula ϕ, and the verification process guarantees the correctness of ϕ according to the constraints illustrated in Definition 7. $\square$

Although our synthesis algorithm of winning formulas is sound and relatively complete on LIA, this algorithm is non-terminating and incomplete. This is because some linear ICGs (*i.e.*, ICGs formalized in LIA) have no LIA-definable winning formula, *e.g.*, the Wythoff game.

4.3 Refining the winning formula

Although we obtain the winning formula from the above step, it is not sufficient to synthesize winning strategies. We illustrate this with the following example. The winning formula for the 2-rowed Chomp game is $v_1 \neq v_2 + 1$. For the first case $v_1 < v_2 + 1$, the player should perform the action $eat2(v_1)$ so as to reach a losing state. For the other case $v_1 > v_2 + 1$, she should execute $eat1(v_2 + 2)$. It can be observed that the player chooses two different actions $eat1$ and $eat2$ in different cases. Therefore, the winning formula should be divided into a set of formulas satisfying the meticulous condition.

Definition 8. Let $\Pi = \langle \mathcal{X}, \mathcal{A}, \mathcal{C}, \mathcal{E} \rangle$ be an ICG. Let ϕ be the winning formula of Π, and ψ a formula entailing ϕ. The formula ψ is *meticulous*, if there is a semi-ground action $a[\mathcal{Y}/\Sigma]$ s.t. the following condition holds

$$\mathcal{C} \wedge \psi \rightarrow \{\texttt{pre}(a[\mathcal{Y}/\Sigma]) \wedge \forall \mathcal{X}'[\mathcal{T}(a[\mathcal{Y}/\Sigma]) \rightarrow \neg\phi[\mathcal{X}/\mathcal{X}']]\}.$$

Intuitively, the above formula says that $a[\mathcal{Y}/\Sigma]$ is applicable over every legal state s satisfying ψ, and performing $a[\mathcal{Y}/\Sigma]$ from these states satisfying ψ leads to a losing state.

We say a cover Ψ of the winning formula is *meticulous*, if every formula $\psi \in \Psi$ is meticulous.

We hereafter present a syntactic method to refine the cover of the winning formula ϕ facilitating synthesis of winning strategies. The method involves two syntactic operations. We first obtain an equivalent formula ϕ' by replacing in ϕ every occurrence of numeric literals of the form $e_1 \neq e_2$, $e_1 \leq e_2$, $e_1 \geq e_2$ and $e_1 \not\equiv_{e_3} e_2$ by the disjunction of arithmetic literals as follows.

1. $e_1 \neq e_2 \Rightarrow e_1 < e_2 \vee e_1 > e_2$;
2. $e_1 \leq e_2 \Rightarrow e_1 < e_2 \vee e_1 = e_2$;
3. $e_1 \geq e_2 \Rightarrow e_1 > e_2 \vee e_1 = e_2$;
4. $e_1 \not\equiv_{e_3} e_2 \Rightarrow \bigvee_{0 \leq e' \leq e_3 \text{ and } e' \neq e_2} e_1 \equiv_{e_3} e'$.

We then transform ϕ' into the form $\bigvee_{i=1}^{n} \psi_i$ where ψ_i is a arithmetic term via the distributive law and removing contradictory arithmetic terms. It can be verified that $\{\psi_1, \cdots, \psi_n\}$ forms a cover of the winning formula.

We remark that this step may cause an exponential blowup in the size of the original winning formula. The experimental results in Section 5 show that the runtime of the refining step can be ignored. Another noteworthy point is that it is not guaranteed that the cover constructed by the method meet the meticulousness condition. Fortunately, our method does work on all benchmarks of games considered in this paper.

4.4 Synthesizing the winning strategy

Finally, we synthesize the winning strategy from the cover of the winning formula as follows.

Algorithm 2 gives the pseudo-code of the synthesis process. For each ψ of the cover, we need to find a semi-ground action $a[\mathcal{Y}/\Sigma]$ satisfying the constraint illustrated in

Algorithm 2: `SynthesizeWinningStrategy`(Π, Ψ)

```
Input: Π: an description of impartial combinatorial game;
       Ψ: a cover of the winning formula $\phi$ of the ICG Π.
Output: δ: a winning strategy for Π.
1  foreach $\psi \in \Psi$ do /* Traverses each formula ψ of the cover Ψ */
2      $found \leftarrow false$
3      foreach $a(\mathcal{Y}) \in \mathcal{A}$ do /* Find a vector Σ of semi-ground expressions s.t. $a[\mathcal{Y}/\Sigma]$ is winning for $\psi_i$ */
4          $\theta \leftarrow \mathcal{C} \wedge \psi \rightarrow \{\texttt{pre}(a[\mathcal{Y}/\Sigma]) \wedge$
5              $\forall \mathcal{X}'[\mathcal{T}(a[\mathcal{Y}/\Sigma]) \rightarrow \neg\phi[\mathcal{X}/\mathcal{X}']]\}$
6          $(\Sigma, found) \leftarrow \texttt{Synthesize}(\{\theta\})$
7          if $found = true$ then
8              $\delta \leftarrow \delta \cup (\psi, a[\mathcal{Y}/\Sigma])$
9              break
10     return $NULL$
```

Definition 8. To do this, we will identify the vector Σ of semi-ground expressions for each action $a(\mathcal{Y}) \in \mathcal{A}$ via the enumerative algorithm satisfying the constraints illustrated in Definition 8. Suppose that we are considering an action $a(\mathcal{Y})$. If the enumerative algorithm succeeds in finding a desired vector Σ, then $a[\mathcal{Y}/\Sigma]$ is the semi-ground action that is winning for ψ. Initially, the set E of examples is empty. Finally, the output of a state s is constructed as follows. The output is a vector I of integers such that the formula $\forall \mathcal{X}\{\texttt{Form}(s) \rightarrow [\texttt{pre}(a[\mathcal{Y}/I]) \wedge \forall \mathcal{X}'[\mathcal{T}(a[\mathcal{Y}/I]) \rightarrow \phi[\mathcal{X}/\mathcal{X}']]]\}$ is valid. The subformula $\texttt{Form}(s) = \bigwedge_{v \in \mathcal{X}}[v = s(v)]$ is a formula representing the state s. Intuitively, the above formula means that in the state s, the ground action $a(\mathcal{Y}/I)$ is applicable and executing it results in a losing state. The vector I can be computed via a SMT solver. To let the enumerative algorithm to be terminating, we restrict the maximal size of expressions to be m. If no such expression satisfies the constraints, then we consider that the action a is not appropriate, and continue to choose another action. If we succeeds in finding the semi-ground action $a[\mathcal{Y}/\Sigma]$ that is winning for ψ, we continue to handle another formula of the cover. If no action is suitable for the formula ψ, then we consider that ψ is not meticulous, and terminate the synthesis process.

Similarly to the step of synthesizing the winning formula, we obtain the soundness and completeness theorem for synthesizing the winning strategy.

Theorem 2. *Let $\Pi = \langle \mathcal{X}, \mathcal{A}, \mathcal{C}, \mathcal{E} \rangle$ be an ICG. Let ϕ be the winning formula of Π, Ψ the cover of ϕ, and m the maximal size of expressions generated in the enumeration process.*

Soundness *If Algorithm 2 synthesizes the strategy δ, then δ is the winning strategy of Π.*

Bounded Completeness *Suppose that for each $\psi_i \in \Psi$, there is a semi-ground action $a_i[\mathcal{Y}_i/\Sigma_i]$ s.t. each expression of Σ_i is of at most size m and $\{(\psi_1, a_1[\mathcal{Y}_1/\Sigma_1]), \cdots, (\psi_n, a_n[\mathcal{Y}_n/\Sigma_n])\}$ is the winning strategy of Π. Then, Algorithm 2 terminates with a winning strategy δ.*

Proof. **Soundness** Suppose that δ is the strategy that Algorithm 2 returns. For each pair $(\psi, a[\mathcal{Y}/\Sigma])$ of δ, we get that it satisfies the constraints illustrated in Definition 8. Hence, δ satisfies the requirements illustrated in the definition of winning stratigies.

Bounded Completeness Suppose that $\delta : \{(\psi_1, a_1[\mathcal{Y}_1/\Sigma_1]), \cdots, (\psi_n, a_n[\mathcal{Y}_n/\Sigma_n])\}$ is the winning strategy where for each $\psi_i \in \Psi$, there is a semi-ground action $a_i[\mathcal{Y}_i/\Sigma_i]$ s.t. each expression of Σ_i is of at most size m. According to Algorithm 2, the enumeration process in the $\texttt{Synthesize}(\{\theta\})$ process can generate the vector Σ_i of expressions of at most size m for each $\psi_i \in \Psi$. In addition, the verification process guarantees the correctness of Σ_i according to the constraints illustrated in Definition 8. Hence, Algorithm 2 returns a winning strategy s.t. each expression for the parameter of semi-ground actions is of at most size m. □

Now, we discuss the decidability of the problem of strategy synthesis for ICGs. Given a fixed ICG, we have (1) the problem of strategy synthesis is undecidable [Luo and Liu, 2019] since it requires the entailment of arithmetic theory containing multiplication, which is undecidable; (2) the problem of the existence of a LIA-definable strategy is decidable remains open; (3) although the problem that synthesis LIA-definable strategy is decidable if the given ICG has a LIA-definable strategy, Algorithm 2 only computes a LIA-definable strategy under the condition that the cover generated by refining winning formula satisfies the meticulous condition.

Finally, we end with an example of refining the winning formula and synthesizing the winning strategy.

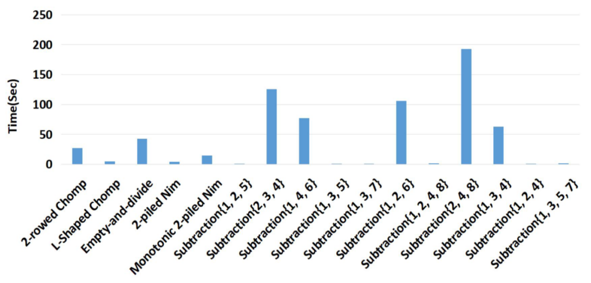
Example 5. We now synthesize the winning strategy of the 2-rowed Chomp game from the winning formula $v_1 \neq v_2 + 1$. We first refine the cover of the winning formula as $\{v_1 < v_2 + 1, v_1 > v_2 + 1\}$. Then, according to Algorithm 2, we choose the action $eat2(v_1)$ for the first case, and $eat1(v_2 + 2)$ for the second case. We combine all results, and obtain the winning strategy $\{(v_1 < v_2 + 1, eat2(v_1)), (v_1 > v_2 + 1, eat1(v_2 + 2))\}$.

5 Experimental Evaluation

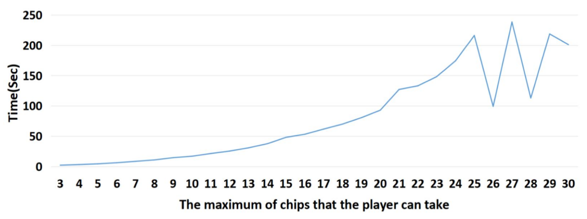
We have implemented our approach, proposed in the previous section, to a system by using Python and Z3 [de Moura and Bjørner, 2008]. We evaluate our system on the following games: 2-rowed and L-shaped Chomp [Schuh, 1952], Empty-and-Divide [Ferguson, 1998], 2-piled Nim [Bouton, 1901], the monotonic variation of 2-piled Nim [Ahn *et al.*, 2017], Take-away [Ferguson, 2018], and Subtraction [Yaglom, 2001]. All experiments were conducted on a machine with an Intel Core i5 2.50 GHz CPU and 8GB RAM under Windows 10. We let the maximal size m of expressions in the step of synthesizing winning strategies be 9, and use timeout of 300s for each synthesis task.

Although there are some related work about strategy synthesis [Beyene *et al.*, 2014; Farzan and Kincaid, 2018], they cannot solve the above games in a fully-automated way. This will be explained in Section 6. Hence, we do not make a comparison with their work.

2-rowed Chomp: The winning formula is $v_1 \neq v_2 + 1$, meaning that the number of cookies in the first row is not equal to the number of cookies in the second row plus 1. The



(a) The first 5 games and Subtraction game



(b) The Take-away game

Figure 1: Runtimes for impartial combinatorial games.

winning strategy is as follows: the player eats the cookie at $(2, v_1)$ (resp. $(1, v_2 + 2)$) if $v_1 < v_2 + 1$ (resp. $v_1 > v_2 + 1$).

L-shaped Chomp: The L-shaped Chomp game is the Chomp game with an L-shaped board with vertical and horizontal lines. The state variables v_1 and v_2 denote numbers of chips on the vertical and horizontal lines respectively. The winning formula is $v_1 \neq v_2$, and the winning strategy is that the player eats the cookie at $(2, v_1 + 1)$ (resp. $(1, v_2 + 1)$) if $v_1 < v_2$ (resp. $v_1 > v_2$).

Empty-and-divide: There are two piles containing v_1 and v_2 chips respectivley. Each player can empty one of the piles, and then divides the chips of the other into two piles such that at least one chip is in each pile. This game ends when both of two piles contains only one chip. The winning state is a state such that the number of chips in any pile is divided by 2 (*i.e.*, $v_1 \equiv_2 0 \vee v_2 \equiv_2 0$). The winning strategy for the case $v_1 \equiv_2 0$ is to empty the second pile, then to move $v_2 - 1$ chips into the first pile, and finally to keep only one chip in this pile if $v_1 \equiv_2 0$. The winning strategy for the case $v_2 \equiv_2 0$ is similar.

2-piled Nim: There are two piles of chips containing v_1 and v_2 chips respectivley. Each player can choose one from these two piles, and then take arbitrary chips from it. Each player cannot remove chips from both piles in one turn. If both piles are empty, then the game ends. The winning formula is $v_1 \neq v_2$. The winning strategy is to take $v_2 - v_1$ (resp. $v_1 - v_2$) chips from the second (resp. first) pile, if $v_1 < v_2$ (resp. $v_2 > v_1$).

Monotonic 2-piled Nim: This game is a monotonic variant of 2-piled Nim. The player can take arbitrary number of chips from one pile, given that the resulting sequence is non-decreasing. That is, the second pile always contains at least as many chips as the first pile ($v_1 \leq v_2$). The winning formula is $v_2 > v_1$. The winning strategy is to take $v_2 - v_1$ chips from the first pile at any winning state.

Take-away: There is a pile that contains v chips. Let n be the parameter of this game, denoting the maximum of chips that the player can take. At least one chip must be taken. This games ends when no chip is in the pile. For any parameter n, the winning formula is $v \equiv_{n+1} 0$, meaning that the number of the remaining chips is not divided by n, and the winning strategy is to take chips such that $v \equiv_{n+1} 0$ holds. We test 28 testcases for this game where $3 \leq n \leq 30$. As Figure 1(b) shows, the maximal time of computing winning strategy is 238s, and is able to scale up reasonably well.

Subtraction: The Subtraction game is a generalization of the Take-away game. Let S be a finite set of positive integers. In this game, each player can take k chips from the pile where $k \in S$. The ending state is that no player can execute the taking action. The winning formula and winning strategy varies for the games with different sets. Due to space reason, we only present the results our approach obtain for the game with $\{1, 4, 6\}$. The winning formula is $v \not\equiv_5 0 \wedge v \not\equiv_5 2$. The winning strategy is to take 1 (resp. 4) chip(s) if $v \equiv_5 1 \vee v \equiv_5 3$ (resp. $v \equiv_5 4$). We test 11 cases for this game. Our approach can solve all cases in 192s, and solves 6 cases in a very short time (< 1.5s).

Finally, we close this section by summarizing experimental results. The runtimes of our approach in the above games are reported in Figure 1. Our approach is able to solve all games in a reasonable amount of time (< 250s). This shows the effectiveness and scalability of our approach on a wide range of games. Hence, our approach provides an effective way to constructing winning formulas and winning strategies.

6 Related Work

Strategy Synthesis in LTL Games. A well-known class of games, called Linear Temporal Logic (LTL) Games, is proposed in [Pnueli and Rosner, 1989]. These games are represented by an automaton, and the goals are expressed by an LTL formula. The method proposed in [Pnueli and Rosner, 1989] is an automaton-based approach causing a doubly exponential time complexity. To lower the complexity, Asarin *et al.* [1998] restrict the goal to be some fragments of LTL formulas so as to obtain polynomial synthesis approaches, *e.g.*, the safety and reachability games whose goal is of the form $\Box\phi$ and $\Diamond\phi$ respectively where ϕ is propositional. The above methods only focus on games over finite states. In general, it is undecidable for the synthesis problem of infinite-state games [de Alfaro *et al.*, 2001]. Some decidable restricted classes of infinite-state games are identified, *e.g.*, games on automatic graphs [Neider and Topcu, 2016], pushdown graphs [Cachat, 2002; Chatterjee and Fijalkow, 2013], and prefix-recognizable graphs [Cachat, 2003].

Besides automatons, arithmetic formulas are an alternative to formalization for games. Compared to the former, the latter are easier to understand, and hence being more suitable for human beings to give a complete description. Recently, arithmetic formulas are used to represent infinite-state games [Beyene *et al.*, 2014; Farzan and Kincaid, 2018]. Our representative framework is also based on arithmetic formulas, and close to their work.

Beyene *et al.* [2014] devise a sound and relatively complete proof rules for computing winning strategies for safety, reachability and LTL games over infinite states. All impartial combinatorial games considered in this paper are a reachability game over infinite states. The rules for reachability games rely on winning formulas and well-founded relations that are used to guarantee the correctness and termination of the winning strategy respectively. The generation of winning formulas and well-founded relations is solved by the EHSF solver [Beyene *et al.*, 2013]. Our approach finds the winning formula via an enumerative approach, and does not generate the well-founded relation as an impartial combinatorial game always ends in a finite number of moves. One major drawback of Beyene *et al.*'s approach is that it requires users to provide hints in the form of templates to synthesize winning strategies, and hence it is not a full mechanization. By contrast, our approach succeeds in computing winning strategies for all games considered in this paper in a fully automated way.

Farzan and Kincaid [2018] propose a fully automated method, called SimSynth, for synthesizing winning strategies for reachability games over infinite states. SimSynth firstly unrolls games as a bounded variation. Then it reduces the problem of synthesizing the winning strategy for the bounded variation to the linear arithmetic satisfiability problem [Farzan and Kincaid, 2016]. Finally, it attempts to generalize the strategy so as to hold for all states. The insight of our approach is different from SimSynth. Instead, we first construct the winning formula, and then generate the winning strategy from the cover of the winning formula. SimSynth does not involve the winning formula, which precisely describes the set of winning states. In practice, SimSynth is only able to synthesizes the winning strategy of an ICG instance, and does not scale up well. For example, it fails to solve the 2-piled Nim game for the state $v_1 = 6 \wedge v_2 = 6$ within 10 minutes. By contrast, our approach succeeds in computing generalized winning strategies, which works for not only one state but also all winning states, for the 2-piled Nim game in 3.91 seconds.

Another closely related work is the verification of strategies proposed in [Luo and Liu, 2019]. They extend the situation calculus for describing combinatorial games, and use finite state automaton (FSA) to represent generalized winning strategy. The key insight of their approach is to find an invariant for the FSA strategy, which is similar to the notion of winning formulas. Hence, their approach has a few ideas in common with those proposed in this paper and [Beyene *et al.*, 2014]. Their approach only verifies if an FSA is a winning strategy, but cannot generate a winning strategy.

Generalized planning. A related problem is generalized planning that aims to construct a general plan that works for possibly infinitely many planning instances sharing the similar structure [Levesque, 2005; Srivastava *et al.*, 2011]. In this paper, we concentrate on ICGs that is more complex than the planning domain. The former involves two players while the latter only focuses on the single-agent case. Hence it is necessary to consider not only how the current player chooses actions, but also how the other player responds when we synthesize winning strategies. Recently, De Giacomo *et al.* [2016] study the translation from generalized planning under partial observability to two-player games with perfect information. In the case there are no additional fairness assumptions, the former corresponds to two-player games with imperfect information. By adapting the belief-state construction, De Giacomo *et al.* remove imperfect information from games, and hence obtain a two-player game with perfect information corresponding to the given generalized planning problem. However, they do not provide an approach to synthesizing winning strategies for the games with perfect information corresponding to the given generalized planning problem.

7 Conclusions and Future Work

We have devised an effective approach to computing generalized winning strategies that works for possibly infinite many instances of ICGs. The key insight of our approach is to synthesize the winning formula and winning strategy by extending the enumerative algorithm to learn multiple objects. To exploit the extended enumerative algorithm, we have provided constraints that are the specifications of the winning formula and winning strategy. In theory, our algorithm for synthesizing the winning formula (Alg. 1) is proven to be sound and relatively complete on LIA, and our algorithm for synthesizing the winning strategy (Alg. 2) is proven to be sound and bounded complete. We have implemented our approach and experimental results show the effectiveness and scalability of our proposed approach.

Our approach have some limitations, and hence leading to several avenues for future work. Firstly, the ICGs considered in this paper are formalized in LIA. For other ICGs, their formalization involves real numbers, existential quantifiers, multiple operator, rounding operator, exclusive-disjunction operator and so on. We would also like to consider the games whose formalizations require more expressive language. Secondly, the enumerative algorithm is a simple algorithm for the syntax-guided synthesis problem (SyGuS). It would like to design the solution to strategy synthesis based on more efficient algorithms for SyGuS [Alur *et al.*, 2017; Reynolds *et al.*, 2019]. Finally, we focus on ICGs under the normal rule. It is also interesting to apply our approach to other categories of games, *e.g.*, ICGs under the misère rule (*i.e.*, all ending states are losing states), and partizan combinatorial games (*i.e.*, some moves are available to one player and not to the other).

Acknowledgments

We are grateful to Yongmei Liu, Kailun Luo, Naiqi Li and Peiming Mo for their comments on the paper. This paper was supported by the National Natural Science Foundation of China (Nos. 61603152, 61602210, 61703182, 61772232, 61773179, 61877029, 61966009, U1711263, U1811263 and U1811264), the Science and Technology Planning Project of Guangdong (Nos. 2019KTSCX010, 2018KTSCX016 and 2019A050510024), the Science and Technology Project of Guangzhou (No. 201902010041), and the Project of Guangxi Key Laboratory of Trusted Software.

References

[Ahn *et al.*, 2017] June Ahn, Benjamin Chen, Richard Chen, Ezra Erives, Jeremy Fleming, Michael Gerovitch, Tejas Gopalakrishna, Tanya Khovanova, Neil Malur, Nastia Polina, and Poonam Sahoo. On variations of nim and chomp. *arXiv preprint arXiv:1705.06774*, 2017.

[Alur *et al.*, 2017] Rajeev Alur, Arjun Radhakrishna, and Abhishek Udupa. Scaling Enumerative Program Synthesis via Divide and Conquer. In *TACAS*, pages 319–336, 2017.

[Asarin *et al.*, 1998] Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller Synthesis for Timed Automata. In *Proc. of the 5th IFAC Conference on System Structure and Control*, pages 447–452, 1998.

[Beyene *et al.*, 2013] Tewodros A. Beyene, Corneliu Popeea, and Andrey Rybalchenko. Solving Existentially Quantified Horn Clauses. In *CAV*, pages 869–882, 2013.

[Beyene *et al.*, 2014] Tewodros Beyene, Swarat Chaudhuri, Corneliu Popeea, and Andrey Rybalchenko. A Constraint-based Approach to Solving Games on Infinite Graphs. In *POPL*, pages 221–233, 2014.

[Bouton, 1901] Charles L. Bouton. Nim, a game with a complete mathematical theory. *Ann. Math.*, 3(1/4):35–39, 1901.

[Cachat, 2002] Thierry Cachat. Symbolic Strategy Synthesis for Games on Pushdown Graphs. In *ICALP*, pages 704–715, 2002.

[Cachat, 2003] Thierry Cachat. Uniform Solution of Parity Games on Prefix-Recognizable Graphs. *Electronic Notes in Theoretical Computer Science*, 68(6):71–84, 2003.

[Chatterjee and Fijalkow, 2013] Krishnendu Chatterjee and Nathanaël Fijalkow. Infinite-state games with finitary conditions. In *CSL*, pages 181–196, 2013.

[Chatterjee *et al.*, 2010] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Strategy logic. *Information and Computation*, 208:677–693, 2010.

[Cooper, 1972] David C. Cooper. Theorem proving in arithmetic without multiplication. *Machine Intelligence*, 7(91-99):300, 1972.

[de Alfaro *et al.*, 2001] Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. Symbolic Algorithms for Infinite-State Games. In *CONCUR*, pages 536–550, 2001.

[De Giacomo *et al.*, 2016] Giuseppe De Giacomo, Aniello Murano, Sasha Rubin, and Antonio Di Stasio. Imperfect-Information Games and Generalized Planning. In *IJCAI*, pages 1037–1043, 2016.

[de Jonge and Zhang, 2016] Dave de Jonge and Dongmo Zhang. Lifted Backward Search for General Game Playing. In *Proc. of the 29th Australasian Joint Conference on Advances in Artificial Intelligence*, pages 3–16, 2016.

[de Moura and Bjørner, 2008] Leonardo de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *TACAS*, pages 337–340, 2008.

[Farzan and Kincaid, 2016] Azadeh Farzan and Zachary Kincaid. Linear arithmetic satisfiability via strategy improvement. In *IJCAI*, pages 735–743, 2016.

[Farzan and Kincaid, 2018] Azadeh Farzan and Zachary Kincaid. Strategy Synthesis for Linear Arithmetic Games. *Proceedings of the ACM on Programming Languages*, 2:61:1–61:30, 2018.

[Ferguson, 1998] Thomas S. Ferguson. Some chip transfer games. *Theoretical Computer Science*, 191(1-2):157–171, 1998.

[Ferguson, 2018] Thomas S. Ferguson. *Game Theory*. World Scientific, 2018.

[Fox and Long, 2003] Maria Fox and Derek Long. PDDL 2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[Gale and Stewart, 1953] David Gale and Frank M. Stewart. Infinite games with perfect information. *Contributions to the Theory of Games*, 2:245–266, 1953.

[Hoffmann and Brafman, 2006] Jörg Hoffmann and Ronen I. Brafman. Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence*, 170(6-7):507–541, 2006.

[Levesque, 2005] Hector J. Levesque. Planning with Loops. In *IJCAI*, pages 509–515, 2005.

[Liu and Zhao, 2015] Wen An Liu and Meile Zhao. General restrictions of Wythoff-like games. *Theoretical Computer Science*, 602:80 – 88, 2015.

[Luo and Liu, 2019] Kailun Luo and Yongmei Liu. Automatic Verification of FSA Strategies via Counterexample-Guided Local Search for Invariants. In *IJCAI*, pages 1814–1821, 2019.

[Monniaux, 2010] David Monniaux. Quantifier elimination by lazy model enumeration. In *CAV*, pages 585–599, 2010.

[Neider and Topcu, 2016] Daniel Neider and Ufuk Topcu. An Automaton Learning Approach to Solving Safety Games over Infinite Graphs. In *TACAS*, pages 204–221, 2016.

[Pnueli and Rosner, 1989] Amir Pnueli and Roni Rosner. On the Synthesis of a Reactive Module. In *POPL*, pages 179–190, 1989.

[Reynolds *et al.*, 2019] Andrew Reynolds, Haniel Barbosa, Andres Nötzli, Clark W. Barrett, and Cesare Tinelli. cvc4sy: Smart and Fast Term Enumeration for Syntax-Guided Synthesis. In *CAV*, pages 74–83, 2019.

[Rintanen, 2012] Jussi Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012.

[Schuh, 1952] Frederik Schuh. The game of divisions. 1952.

[Srivastava *et al.*, 2011] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. A new representation and associated algorithms for generalized planning. *Artificial Intelligence*, 175(2):615–647, 2011.

[Udupa *et al.*, 2013] Abhishek Udupa, Arun Raghavan, Jyotirmoy V. Deshmukh, Sela Mador-Haim, Milo M. K. Martin, and Rajeev Alur. TRANSIT: Specifying Protocols with Concolic Snippets. In *PLDI*, pages 287–296, 2013.

[Wythoff, 1907] W. A. Wythoff. A modification of the game of nim. *Nieuw Archief voor Wiskunde*, 7:199–202, 1907.

[Xiong and Liu, 2016] Liping Xiong and Yongmei Liu. Strategy Representation and Reasoning for Incomplete Information Concurrent Games in the Situation Calculus. In *IJCAI*, pages 1322–1329, 2016.

[Yaglom, 2001] IM Yaglom. Two games with matchsticks. *Kvant Selecta: Combinatorics, I*, 1:1–8, 2001.

[Zeilberger, 2001] Doron Zeilberger. Three-Rowed Chomp. *Advances in Applied Mathematics*, 26(2):168–179, 2001.

[Zermelo, 1913] Ernst Zermelo. Über eine anwendung der mengenlehre auf die theorie des schachspiels. In *Proc. of the 5th International Congress of Mathematicians*, pages 501–504, 1913.

[Zhang and Thielscher, 2015] Dongmo Zhang and Michael Thielscher. Representing and Reasoning about Game Strategies. *Journal of Philosophical Logic*, 44:203–236, 2015.