Generalized Strategy Synthesis of Infinite-State Impartial Combinatorial Games via Exact Binary Classification

Liangda Fang
Jinan University
Guangzhou, China
Pazhou Lab
Guangzhou, China
fangld@jnu.edu.cn

Meihong Yang
Jinan University
Guangzhou, China
ymh67@stu2022.jnu.edu.cn

Dingliang Cheng
Jinan University
Guangzhou, China
dingliang123@stu2020.jnu.edu.cn

Yunlai Hao
Jinan University
Guangzhou, China
haoyunlai@stu2020.jnu.edu.cn

Quanlong Guan*
Jinan University
Guangzhou, China
gql@jnu.edu.cn

Liping Xiong
Wuyi University
Jiangmen, China
xionglp@wyu.edu.cn

ABSTRACT

In game theory, a fundamental class of games is impartial combinatorial games (ICGs). One of the challenging and long-standing problems of ICGs is to compute generalized winning strategies for possibly infinite number of legal states. Recently, Wu et al. proposed an automated method to synthesize generalized winning strategies of infinite-state ICGs. Their method has two major drawbacks: (1) it fails to generate winning formula with large size; and (2) it cannot usually construct the winning strategy even the winning formula is obtained. To tackle the above two drawbacks, in this paper, we propose the problem of exact binary classification and design a partial MaxSAT-based method to this problem. Then, we reduce the synthesis problem of generalized winning strategies of infinite-state ICGs to exact binary classification. The experimental results show that our method is more scalable and effective than Wu et al.'s approach.

KEYWORDS

Impartial Combinatorial Games, MaxSAT, Winning Strategies

ACM Reference Format:
Liangda Fang, Meihong Yang, Dingliang Cheng, Yunlai Hao, Quanlong Guan, and Liping Xiong. 2024. Generalized Strategy Synthesis of Infinite-State Impartial Combinatorial Games via Exact Binary Classification. In *Proc. of the 23rd International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – May 10, 2024*, IFAAMAS, 9 pages.

1 INTRODUCTION

Game theory is the study of mathematical models of strategic interactions among rational agents and has numerous applications in several areas of artificial intelligence, including multi-agent systems [2, 37], imitation and reinforcement learning [36, 38] and adversary training in generative adversarial networks [10, 16]. Impartial combinatorial games (ICGs) are a fundamental branch of game theory that studies sequential games with perfect information. An ICG satisfies the following conditions [15]: (1) there are two players and possibly infinitely 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. The states in an ICG are classified into two categories: winning and losing. One player has the ability to win the game in a winning state while it is impossible in a losing state.

In general, an ICG contains infinite number of states, and so winning and losing states are infinite. Identifying the exact characteristic of the set of winning states has always been a long-standing topic of ICGs [5, 20, 24, 30]. But the above works are manually accomplished by mathematicians and computer scientists. Moreover, they do not provide an explicit strategy that defines which action to be executed so as to reach a losing state from a winning state. The player has to compute an appropriate action repeatedly in every winning state when playing an ICG.

Recently, Wu et al. [33] proposed an automated synthesis approach called Enum to computing winning strategies. The Enum approach represents the set of winning states in an arithmetic formula, namely the winning formula. To generate the winning formula, Enum adopts a simple enumerative algorithm proposed in [31], generating candidate solutions iteratively by induction on the size until it finds a correct solution. After obtaining the winning formula, Enum splits it into a set of arithmetic terms such that their disjunction is equivalent to the winning formula according to a set of simple division rules. For each term, Enum also uses the enumerative algorithm to get the desired action such that the player reaches a losing state from any state satisfying the term via performing this action. However, Wu et al.'s approach has two major drawbacks: (1) It does not scale well since exhaustive generation of all candidate formulas becomes time consuming when the size of the winning formula is relatively large; (2) It fails to compute a winning strategy as the simple division rules cannot make sure that each term has the proper action.

*Correspoonding Author.

Proc. of the 23rd International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – May 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

The above two drawbacks of the Enum approach drive us to propose a more scalable method for synthesizing winning strategies. For the first drawback, we observe that in the synthesis of winning formulas, the crucial step is to generate a candidate formula that is consistent with two generated sets of positive states and negative states. To accelerate this step, we first formulate it as exact binary classification problem with all atoms as classification attributes and encode each classification problem as an instance of the partial MaxSAT problem. We can acquire the winning formula of large size with the aid of off-the-shelf partial MaxSAT solvers. To address the second drawback, our approach first produces a finite set of actions that transforms at least one winning state to a losing state, and then determines the condition formula that captures the set of winning states from which performing the action leads to a losing state for each action. The actions and their corresponding conditions formulas form a winning strategy. We evaluate our proposed approach on the publicly available benchmark composed of 5 categories of ICGs: Subtraction, Nim, Wythoff, Welter, and Chomp for a total of 3, 720 test cases. The experimental results show that our approach achieves a significant improvement over Wu et al.'s approach.

2 PRELIMINARIES

In this section, we introduce the syntax of linear integer arithmetic (LIA), and the partial maximum satisfiability (MaxSAT) problem.

2.1 Linear Integer Arithmetic

Let $\mathcal{N}$ be the set of integers and $\mathcal{V}$ the set of variables. *Linear integer arithmetic expressions* (Exp), *atoms* (Atom) and *formulas* (Form) are defined by the following grammar:

$$e, e_1, e_2 \in \texttt{Exp} :: c \mid v \mid e_1 + e_2 \mid e_1 - e_2$$

$$l \in \texttt{Atom} :: e_1 = e_2 \mid e_1 < e_2 \mid e_1 \leq e_2 \mid e \equiv_{c_1} c_2$$

$$\phi, \phi_1, \phi_2 \in \texttt{Form} :: \top \mid \bot \mid l \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \forall v\phi \mid \exists v\phi$$

where $c, c_1, c_2 \in \mathcal{N}$ and $v \in \mathcal{V}$.

The atom $e \equiv_{c_1} c_2$ means that e and c_2 are congruence modulo c_1, that is, $e - c_2$ is divisible by c_1. We use $e_1 \neq e_2$ for $\neg(e_1 = e_2)$, $e_1 > e_2$ for $\neg(e_1 \leq e_2)$, $e_1 \geq e_2$ for $\neg(e_1 < e_2)$, $e \not\equiv_{c_1} c_2$ for $\neg(e \equiv_{c_1} c_2)$, and $\phi_1 \rightarrow \phi_2$ for $\neg\phi_1 \vee \phi_2$. We use $|e|$ (resp. $|\phi|$) to represent the size of an LIA expression e (resp. LIA formula ϕ), that is, the number of occurrences of integers, variables and connectives in e (resp. ϕ). For example, the formula $v_1 = 3$ has size 3. Given a set Ψ of atoms, we say a formula ϕ is based on Ψ, iff it is built from the set Ψ, the connectives $\neg, \vee, \wedge, \rightarrow$ and two Boolean constants $\top$ and $\bot$. For example, suppose that $\Psi = \{x = 1, x > 2\}$. The formula $\neg(x = 1 \vee x > 2)$ is based on Ψ, but $x \leq 3 \wedge x > 2$ is not as the atom $x \leq 3 \notin \Psi$.

It is well-known that LIA allows for quantifier elimination, that is, any LIA formula can be transformed into an equivalent quantifier-free formula (qf-formula) [12, 27]. We use $\texttt{Form}^{\texttt{qf}}$ for the set of qf-formulas. Let $\mathcal{U}$ be a subset $\{u_1, \cdots, u_n\}$ of variables. We use $\texttt{Exp}_{\mathcal{U}}$ for the set of expressions over $\mathcal{U}$. We use $\texttt{Form}_{\mathcal{U}}$ for the set of formulas of which free variables are $\mathcal{U}$. The notation $\texttt{Form}^{\texttt{qf}}_{\mathcal{U}}$ is similar. We use $\forall\mathcal{U}\phi$ for $\forall u_1 \cdots \forall u_n\phi$ and $\exists\mathcal{U}\phi$ for $\exists u_1 \cdots \exists u_n\phi$.

2.2 Partial Maximum Satisfiability

Given a propositional formula in conjunctive normal form, the *satisfiability (SAT)* problem aims to determine if there is an assignment that satisfies all of the clauses in the formula. The *maximum satisfiability (MaxSAT)* problem, the optimization variation of the SAT problem, searches a truth assignment that satisfies the maximum number of clauses. One of the significant generalizations of the MaxSAT problem is the *partial MaxSAT* problem. It divides the clauses into two types: *hard* and *soft*, and searches an assignment that satisfies all of the hard clauses and maximizes the number of satisfied soft clauses.

3 IMPARTIAL COMBINATORIAL GAMES

In this section, we first introduce the concepts of numeric states and actions that serve as the theoretical basis of ICGs. Then, we present the formalization of ICGs and essential concepts of ICGs, including winning states, winning formulas and winning strategies. Finally, we present several constraints for correctness verification of winning formulas and strategies.

3.1 Numeric States and Actions

Throughout this paper, we use the notion $\mathcal{X}$ for the set of state variables that are used to define numeric states. An expression e (resp. a formula ϕ) is *semi-ground*, if $e \in \texttt{Exp}_{\mathcal{X}}$ (resp. $\phi \in \texttt{Form}_{\mathcal{X}}$).

A *numeric state* s is an interpretation that maps every state variable v to an integer $v(s)$. For a state s, we can evaluate a semi-ground expression e into an integer $e(s)$ that the expression simplifies to when replacing each state variable v with its corresponding value $v(s)$. The Boolean value $\phi(s)$ of a semi-ground formula ϕ can be defined in a similar way. A numeric state s *satisfies* a formula ϕ, denoted by $s \models \phi$, iff $\phi(s) = \top$. A formula ϕ *implies* another one ψ, denoted by $\phi \models \psi$, iff $s \models \psi$ for every state s satisfying ϕ. A formula ϕ is *valid*, iff $s \models \phi$ for every numeric state s; otherwise, it is *invalid*.

Each action $a(\mathcal{K})$ is parameterized by a vector $\mathcal{K}$ of variables, and is defined by a pair $\langle \texttt{pre}, \texttt{eff} \rangle$ where $\texttt{pre} \in \texttt{Form}$ is the precondition and $\texttt{eff}$ is the effect represented by a set of numeric effects. A *numeric effect* is a triple $\langle \phi, v, e \rangle$ where $\phi \in \texttt{Form}$, $v \in \mathcal{X}$, and $e \in \texttt{Exp}$. It implies that if ϕ holds in the state s, then the value of v becomes $e(s)$ after executing the action; otherwise, it remains unchanged.

Let $a(\mathcal{K})$ be an action and Σ a vector of semi-ground expressions where $\mathcal{K}$ and Σ have the same length n. A *semi-ground action* $a[\Sigma]$ is obtained from the action $a(\mathcal{K})$ by replacing the i-th parameter k_i of $\mathcal{K}$ with the i-th semi-ground expression e_i of Σ for $1 \leq i \leq n$. If every expression e_i of Σ is an integer, we say the action $a[\Sigma]$ is *ground*. For brevity, we sometimes use a for semi-ground (even ground) actions. A semi-ground action a is *applicable* in a state s, iff $s \models \texttt{pre}(a)$. The result of applying an applicable action a in a state s is a successor state, written $\tau(a, s)$, where every variable v is associated to an integer $v(\tau(s, a))$ defined as:

$$v(\tau(s,a)) = \begin{cases} e(s), & \text{if there is } \langle \phi, v, e \rangle \in \texttt{eff}(a) \text{ and } \phi(s) = \top; \\ v(s), & \text{otherwise.} \end{cases}$$

3.2 Formalization

We hereafter present the formalization of ICGs proposed in [33].

Definition 1. An ICG is defined as a tuple $\Pi = \langle \mathcal{X}, \mathcal{A}, C, \mathcal{E} \rangle$ where

- $\mathcal{X}$: a finite set of state variables.
- $\mathcal{A}$: a finite set of numeric actions.

- $\mathcal{C}$: a formula of $\mathtt{Form}_{\mathcal{X}}$ denoting all legal states.
- $\mathcal{E}$: a formula of $\mathtt{Form}_{\mathcal{X}}$ denoting all ending states.

The player who executes the final action wins. There are two categories of legal states: *winning* and *losing*. The player can force a win in the winning state whereas it is impossible in the losing state.

Definition 2 ([15]). Let $\Pi = \langle \mathcal{X}, \mathcal{A}, \mathcal{C}, \mathcal{E} \rangle$ be an ICG. The sets $\mathcal{W}$ of winning states and $\mathcal{L}$ of losing states are recursively defined as

(1) $\{s \mid s \models \mathcal{C} \wedge \mathcal{E}\} \subset \mathcal{L}$;
(2) $s \in \mathcal{W}$, if there is an applicable ground action a over s s.t. $\tau(s, a) \in \mathcal{L}$;
(3) $s \in \mathcal{L}$, if for every applicable ground action a over s, we have $\tau(s, a) \in \mathcal{W}$.

Definition 2 provides the exact characteristics of winning and losing states in a recursive way. In the base case, Condition 1 requires all legal ending states to be losing states. Both Conditions 2 and 3 are induction steps. Condition 2 says that every state such that at least one applicable action results in a losing state is a winning state. Condition 3 means that every state such that all possible successor states are winning states is a losing state. The sets $\mathcal{W}$ of winning states and $\mathcal{L}$ of losing states are disjoint.

Since the set $\mathcal{W}$ is in general infinite, we give the symbolic representation of $\mathcal{W}$, namely *winning formula*.

Definition 3 ([33]). Let $\Pi = \langle \mathcal{X}, \mathcal{A}, \mathcal{C}, \mathcal{E} \rangle$ be an ICG and $\mathcal{W}$ the set of winning states of Π. A formula $\phi \in \mathtt{Form}_{\mathcal{X}}$ is a *winning formula* of Π, iff $\{s \mid s \models \phi \wedge \mathcal{C}\} = \mathcal{W}$.

In general, there are many logical different winning formulas for an ICG. But they are logically equivalent under the background theory $\mathcal{C}$, that is, $\mathcal{C} \models \phi \equiv \phi'$. Hence, we sometimes call a winning formula the winning one. A *rule* (ψ, a) is a pair of semi-ground formulas ψ and actions a. Given a rule (ψ, a), we call ψ the condition formula of (ψ, a). A *strategy* is a set of rules.

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 rule (ψ, a) is *winning*, iff for every state s satisfying $\mathcal{C} \wedge \phi \wedge \psi$, we have a is applicable in s and $\tau(s, a) \in \mathcal{L}$.
- A rule (ψ, a) is *complete*, iff for every state s s.t. $\tau(s, a) \in \mathcal{L}$, we have $\psi(s) = \top$.
- A strategy $\{(\psi_1, a_1), \cdots, (\psi_n, a_n)\}$ is *winning*, iff every rule (ψ_i, a_i) is winning.
- A strategy $\{(\psi_1, a_1), \cdots, (\psi_n, a_n)\}$ is *complete*, iff $\bigvee_{i=1}^{n} \psi_i$ is the winning formula.

The winning property of the rule (ψ, a) requires that executing the action a in every state satisfying $\mathcal{C} \wedge \phi \wedge \psi$ yields a losing state. The completeness property of the rule says that the formula ψ covers all of the states s s.t. its successor state $\tau(s, a)$ is a losing state. The winning property of the strategy means that each of its rule is a winning rule. The completeness property of the strategy stipulates that the disjunction of the condition formula ψ_i of each rule is the winning formula. We illustrate the above concepts with the following ICG: Take-away game [19].

Example 1. The state variable v denotes the number of the remaining chips in this game. For simplicity, we use an integer c to denote the numeric state where v is assigned to c. A move consists of taking any number (from 1 to m) of chips. Here we consider the case where $m = 3$, which is called Take-away-3 game. All legal states can be represented by the formula $\mathcal{C} : v \geq 0$. The ending condition $\mathcal{E}$ is $v = 0$, meaning that this game ends when no chip is available. The following are preconditions and effects of actions.

(1) $\mathtt{pre}(take(k)) : v \geq k \wedge k > 0 \wedge k \leq 3$;
(2) $\mathtt{eff}(take(k)) : \{\langle \top, v, v - k \rangle\}$;

The two numeric states 0 and 4 are losing states. The former is a legal ending state. The latter has three applicable semi-ground actions: $take(1)$, $take(2)$, and $take(3)$. Executing each of them yields three different winning states $\tau(4, take(1)) = 3$, $\tau(4, take(2)) = 2$, and $\tau(4, take(3)) = 1$.

The winning formula for this game is $v \not\equiv_4 0$. It means that any legal state in which the number of the remaining chips is not a multiple of 4 is a winning state. The three rules $(v \equiv_4 1, take(1))$, $(v \equiv_4 2, take(2))$ and $(v \equiv_4 3, take(3))$ are winning and complete rules, and form a winning and complete strategy. But $(v = 2, take(2))$ is a winning but incomplete rule as there is a state 6 from which taking 2 chips leads to a losing state, which does not satisfy $v = 2$. □

3.3 Correctness Verification

The main idea of the constraint-based verification approach is to first translate the candidate winning formula and strategy to a set of constraints represented by LIA formulas and then use the state-of-the-art SMT solver to determine the validity of these constraints.

We first give the definition of transition formulas and global transition formulas. The transition formula $\mathcal{T}(a(\mathcal{K}))$ for an action $a(\mathcal{K})$ reflects the mapping from predecessor states to successor states due to the effect of $a(\mathcal{K})$. The global transition formula $\mathcal{T}(\mathcal{A})$ represents the union of the mapping due to the effect of every action $a(\mathcal{K}) \in \mathcal{A}$. To distinguish the predecessor state variable and the successor one, we use the unprimed variable v for the former and the primed one v' for the latter.

Definition 5.
- The transition formula $\mathcal{T}(a(\mathcal{K}))$ is $\mathtt{pre}(a(\mathcal{K})) \wedge \bigwedge_{\langle \phi, v, e \rangle \in \mathtt{eff}(a(\mathcal{K}))} (\phi \rightarrow v' = e) \wedge \bigwedge_{v \in \mathcal{V}} [(\bigvee_{\langle \phi, v, e \rangle \in \mathtt{eff}(a(\mathcal{K}))} \phi) \vee v' = v]$.
- The global transition formula $\mathcal{T}(\mathcal{A})$ is $\bigvee_{a(\mathcal{K}) \in \mathcal{A}} \exists \mathcal{K} [\mathcal{T}(a(\mathcal{K}))]$.

Definition 6. 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) $\mathcal{C} \wedge \mathcal{E} \rightarrow \neg\phi$;
(2) $(\mathcal{C} \wedge \phi) \rightarrow \exists \mathcal{X}' [\mathcal{T}(\mathcal{A}) \wedge (\mathcal{C} \wedge \neg\phi)[\mathcal{X}/\mathcal{X}']]$;
(3) $(\mathcal{C} \wedge \neg\phi) \rightarrow \forall \mathcal{X}' [\mathcal{T}(\mathcal{A}) \rightarrow (\mathcal{C} \wedge \phi)[\mathcal{X}/\mathcal{X}']]$.

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

Each of the above constraints corresponds to each condition of winning and losing states (cf. Definition 2). As a result, we obtain the soundness theorem of the constraints for the winning formula.

Theorem 1. *Let $\Pi = \langle \mathcal{X}, \mathcal{A}, \mathcal{C}, \mathcal{E} \rangle$ be an ICG. A formula ϕ is the winning formula for Π iff all of the constraints illustrated in Definition 6 are valid.*

The constraints for winning and complete rules and the soundness theorems are as follows:

Definition 7. Let $\Pi = \langle \mathcal{X}, \mathcal{A}, C, \mathcal{E} \rangle$ be an ICG and ϕ a winning formula of Π.

- The constraint for the winning rule (ψ, a) is $(C \wedge \phi \wedge \psi) \rightarrow \{\mathtt{pre}(a) \wedge \forall \mathcal{X}'[\mathcal{T}(a) \rightarrow \neg\phi[\mathcal{X}/\mathcal{X}']]\}$.
- The constraint for the complete rule (ψ, a) is $\exists \mathcal{X}'[\mathcal{T}(a) \wedge \neg\phi[\mathcal{X}/\mathcal{X}']] \rightarrow \psi$.

Theorem 2. *Let $\Pi = \langle \mathcal{X}, \mathcal{A}, C, \mathcal{E} \rangle$ be an ICG. Then, a rule (ψ, a) is winning (resp. complete) iff the constraint for the winning (resp. complete) rule illustrated in Definition 7 is valid.*

The correctness of a strategy $\{(\psi_1, a_1), \cdots, (\psi_n, a_n)\}$ can be verified according to Definition 7. We first check the correctness of each rule, and then verify if $\bigvee_i \psi_i$ is equivalent to the winning formula.

4 EXACT BINARY CLASSIFICATION

In this section, we introduce the exact binary classification problem and propose an approach that reduces this problem to a partial MaxSAT problem.

Definition 8. Given two sets $\mathcal{P}$ of positive states and $\mathcal{N}$ of negative states and a set Ψ of arithmetic atoms, the task of the exact binary classification problem is to generate a formula ϕ based on Ψ s.t. $\phi(s) = \top$ for every state $s \in \mathcal{P}$ and $\phi(s) = \bot$ for every state $s \in \mathcal{N}$.

The exact binary classification problem may not have a solution in some cases. The sufficient and necessary condition for the existence of a solution is given by the following theorem.

Theorem 3. *Let $\mathcal{P}$ and $\mathcal{N}$ be a set of positive and negative states, respectively, and Ψ a set of atoms. The following two statements are equivalent:*

(1) *For each pair of positive state s and negative state t, there is an atom $\psi \in \Psi$ s.t. $\psi(s) \neq \psi(t)$.*
(2) *There exists a formula ϕ based on Ψ that discriminates $\mathcal{P}$ from $\mathcal{N}$.*

To prove the above theorem, we need the following two definitions that are used to synthesize a desired formula for the exact binary classification problem if the solution exists.

Definition 9. Let $\mathcal{P}$ and $\mathcal{N}$ be two sets of positive and negative states, respectively, and Ψ a set of atoms. The *filter set* $\Psi_{\mathcal{P},\mathcal{N}}$ of Ψ w.r.t. $\mathcal{P}$ and $\mathcal{N}$ is $\{\psi \mid \psi \in \Psi \text{ and } \psi(s) \neq \psi(t) \text{ for some } s \in \mathcal{P} \text{ and } t \in \mathcal{N}\}$.

The filter set is the subset of Ψ in which each atom classifies at least one positive state and one negative state.

Definition 10. Let S be a set of states and Ψ a set of atoms. The *characteristic formula* of S based on Ψ is $\bigvee_{s \in S}[(\bigwedge_{\psi \in \Psi \text{ and } \psi(s)=\top} \psi) \wedge (\bigwedge_{\psi \in \Psi \text{ and } \psi(s)=\bot} \neg\psi)]$.

The conjunct $(\bigwedge_{\psi \in \Psi \text{ and } \psi(s)=\top} \psi) \wedge (\bigwedge_{\psi \in \Psi \text{ and } \psi(s)=\bot} \neg\psi)$ can be considered as a truth assignment of the state s on the set Ψ of atoms. The characteristic formula collects the assignments of every state $s \in S$ on Ψ.

Proof of Theorem 3: ($\Leftarrow$): Let ϕ be a formula based on Ψ s.t. ϕ discriminates $\mathcal{P}$ from $\mathcal{N}$. We prove the first statement by contradiction. Let $s \in \mathcal{P}$ and $t \in \mathcal{N}$ s.t. no atom $\psi \in \Psi$ satisfies the property $\psi(s) \neq \psi(t)$. It follows that $\phi(s) = \phi(t)$. This contradicts the assumption.

States		$v = 0$	$v \equiv_2 0$	$v \equiv_3 0$	$v \equiv_4 0$
$\mathcal{P}$	1	$\bot$	$\bot$	$\bot$	$\bot$
	2	$\bot$	$\top$	$\bot$	$\bot$
	3	$\bot$	$\bot$	$\top$	$\bot$
$\mathcal{N}$	0	$\top$	$\top$	$\top$	$\top$
	4	$\bot$	$\top$	$\bot$	$\top$

Table 1: The example of learning the characteristic formula

($\Rightarrow$): Let ϕ be the characteristic formula of $\mathcal{P}$ based on $\Psi_{\mathcal{P},\mathcal{N}}$. Clearly, ϕ is also based on Ψ. By Definition 10, for every positive state s, ϕ contains $(\bigwedge_{\psi \in \Psi \text{ and } \psi(s)=\top} \psi) \wedge (\bigwedge_{\psi \in \Psi \text{ and } \psi(s)=\bot} \neg\psi)$. Hence, $s \models \phi$. By the assumption and Definition 9, for every negative state t, there is an atom $\psi \in \Psi_{\mathcal{P},\mathcal{N}}$ and a positive state s s.t. $\psi(t) \neq \psi(s)$. None of the conjuncts of ϕ is satisfied by t. Hence, $t \not\models \phi$. In summary, ϕ discriminates $\mathcal{P}$ from $\mathcal{N}$. □

The proof of Theorem 3 offers a direct method to obtain a formula that discriminates positive states from negative ones. In general, the given set Ψ includes a large number of arithmetic atoms. So is the filter set $\Psi_{\mathcal{P},\mathcal{N}}$. Therefore, the characteristic formula of $\mathcal{P}$ based on $\Psi_{\mathcal{P},\mathcal{N}}$ has large size and is difficult to be applied in strategy synthesis.

To decrease the amount of atoms in the solution, we devise a practical method to the exact binary classification problem based on partial MaxSAT. The propositional encoding $T(\mathcal{P}, \mathcal{N}, \Psi)$ takes as inputs two sets $\mathcal{P}$ of positive states and $\mathcal{N}$ of negative states and the set Ψ of arithmetic atoms. The propositions in $T(\mathcal{P}, \mathcal{N}, \Psi)$ are

- $Sel(\psi)$: the atom $\psi \in \Psi$ appears in the formula ϕ.

The hard constraints of $T(\mathcal{P}, \mathcal{N}, \Psi)$ are

- $\bigvee_{\psi(s) \neq \psi(t)} Sel(\psi)$ for every $s \in \mathcal{P}$ and $t \in \mathcal{N}$.

It means that for every positive state s and every negative state t, at least one of the atoms that classify s and t is chosen, corresponding to the first statement of Theorem 3.

The clauses in the soft constraints of $T(\mathcal{P}, \mathcal{N}, \Psi)$ are the negative literal $\neg Sel(\psi)$ for each atom ψ. The partial MaxSAT solver looks for a solution with the fewest possible arithmetic atoms.

Example 2. We use Take-away-3 game to illustrate the method to learn the characteristic formula. Suppose that we are given the sets of positive states $\mathcal{P} : \{1, 2, 3\}$ and of negative states $\mathcal{N} : \{0, 4\}$, and the set of arithmetic atoms $\Psi : \{v = 0, v \equiv_2 0, v \equiv_3 0, v \equiv_4 0\}$. The Boolean value of every atom on every state is shown in Table 1. The propositional encoding $T(\mathcal{P}, \mathcal{N}, \Psi)$ include four propositions: $Selected(v = 0), Selected(v \equiv_2 0), Selected(v \equiv_3 0), Selected(v \equiv_4 0)$. The hard and soft constraints are constructed as mentioned before. The partial MaxSAT solver returns the following assignment:

$Sel(v = 0) = \bot, Sel(v \equiv_2 0) = \bot, Sel(v \equiv_3 0) = \bot, Sel(v \equiv_4 0) = \top.$

Only the atom $v \equiv_4 0$ is picked from the assignment. As the chosen atom does not hold in every state of $\mathcal{P}$, its negation $v \not\equiv_4 0$ is the characteristic formula of $\mathcal{P}$ based on Ψ. In addition, it exactly classifies the given states of positive states and of negative states. □

5 SYNTHESIZING WINNING FORMULAS AND STRATEGIES

In this section, based on exact binary classification, we solve two key synthesis problems of ICGs: winning formulas and strategies.

Algorithm 1: `SynWinForm`

```
1  Initialize the set $S$ of legal states
2  $\mathcal{P}, \mathcal{N} \leftarrow$ ClassifyStates($S$)
3  while true do
4      $\Psi \leftarrow$ EnumerateAtoms($\mathcal{P}, \mathcal{N}$)
5      $\phi \leftarrow$ ExactBinaryClassify($\mathcal{P}, \mathcal{N}, \Psi$)
6      $S' \leftarrow$ VerifyWinForm($\phi$)
7      if $S' = \emptyset$ then
8          return $\phi$
9      else
10         $\mathcal{P}', \mathcal{N}' \leftarrow$ ClassifyStates($S'$)
11         $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}'$ and $\mathcal{N} \leftarrow \mathcal{N} \cup \mathcal{N}'$
```

5.1 Synthesizing Winning Formulas

In this subsection, we propose a counterexample-guided inductive synthesis (CEGIS) approach to synthesizing winning formulas for ICGs over infinite states, illustrated in Algorithm 1. A counterexample in this approach in fact is a state in ICGs. The main idea of Algorithm 1 is as follows: It first attempts to generate a candidate formula ϕ via a set of counterexamples, and then verifies the correctness of ϕ. If ϕ passes the verification, then the algorithm terminates with the winning formula ϕ. Otherwise, a set of new counterexamples will be gathered and used to create a new candidate formula. The above steps repeat until the algorithm discovers the winning formula of the ICG.

We hereafter elaborate Algorithm 1 in details. It initializes the set S as a finite set of legal states, which is accomplished by an SMT solver (Line 1). Each legal state s is the state satisfying the formula C denoting all legal states (cf. Definition 1). The SMT solver takes the formula C as input, and returns a state satisfying C. Since the SMT solver works in a deterministic manner, it always returns the same state for the formula C. We can easily fix this defect and generate many different legal states via adding an extra subformula on C. Suppose that we are given a set S' of legal states. The formula $C_{S'}$ is $C \wedge \bigwedge_{s \in S'} \neg[\bigwedge_{v \in \mathcal{X}}(v = v(s))]$, and hence any state satisfying $C_{S'}$ is a legal state not in S'.

The set S of states will be classified into two sets $\mathcal{P}$ of positive states and $\mathcal{N}$ of negative states (Line 2). In ICGs, each positive state is a winning state while each negative state is a losing state. The classification can be accomplished via backward induction according to the definition of winning and losing states (Definition 2).

Algorithm 1 then enters a loop until a winning formula is discovered (Lines 3 - 11). It generates a set Ψ of arithmetic atoms of increasing sizes until for every pair of positive state and negative state, there is an atom $\psi \in \Psi$ classifying them. By Theorem 3, this property guarantees that we are able to obtain a formula that precisely distinguishes $\mathcal{P}$ from $\mathcal{N}$. In order to prune the huge size of Ψ, we eliminate some arithmetic atoms in Ψ. For example, if two atoms ψ and ψ' agrees on every state of S (*i.e.*, $\psi(s) = \psi'(s)$ for $s \in S$), then only one is kept and the other is removed.

With the two sets of positive states and of negative states and the set of atoms in hand, we can produce a candidate winning formula ϕ via exact binary classification mentioned in Section 4 (Line 5). Then, using an SMT solver we check whether the candidate formula ϕ is a winning formula according to the constraints illustrated in

$\mathcal{P}$	$\mathcal{N}$	Candidate formulas	Counterexamples
1	0	$v = 1$	3
1, 3	0	$v > 0$	4
1, 3	0, 4	$v \not\equiv_2 0$	2
1, 2, 3	0, 4	$v \not\equiv_4 0$	$\emptyset$

Table 2: A run of synthesizing the winning formula

Definition 6 (Line 6). If ϕ is correct (that is, $S' = \emptyset$), then Algorithm 1 terminates with the solution ϕ (Line 8). Otherwise, the verification procedure yields some states of S' that are witnesses of its failure and incorporates them into $\mathcal{P}$ and $\mathcal{N}$, respectively (Lines 10 - 12).

By Theorem 1, the verification procedure confirms the soundness property of Algorithm 1 for synthesizing the winning formula.

Theorem 4. *Let $\Pi = \langle \mathcal{X}, \mathcal{A}, C, \mathcal{E} \rangle$ be an ICG. The formula returned by Algorithm 1 is the winning formula of* Π.

Example 3. Table 2 shows the run of synthesizing the winning formula of Take-away-3 game. At the beginning, only the state 1 is in the set $\mathcal{P}$ and the state 0 in $\mathcal{N}$. Our approach generates the formula $v = 1$ as the candidate solution that is consistent with $\mathcal{P}$ and $\mathcal{N}$. However, it is not the correct winning formula that holds for all the legal states and the SMT solver returns a winning state 3. Our approach takes into account the new state. Two candidate formulas $v > 0$ and $v \not\equiv_2 0$ are sequentially found. The above two formulas, however, both fail to verify and return a losing state 4 and a winning state 2, respectively, as counterexamples to the two formulas. Finally, the algorithm terminates with the correct winning formula $v \not\equiv_4 0$ that constructed from the above 5 states.

5.2 Synthesizing Winning Strategies

After the process of synthesizing winning formulas, we have the winning formula ϕ and the two sets of states $\mathcal{P}$ and $\mathcal{N}$ that are used to produce ϕ via exact binary classification. In this subsection, we also devise a CEGIS approach to synthesizing winning strategies, illustrated in Algorithm 2. It first generate a pool of candidate actions for the set of positive states. For each action, we synthesize its condition formula via the partial Max-SAT-based approach. This step will continue until we synthesize a winning strategy that consists of a set of pairs of action and its condition formula. The following gives a comprehensive explanation of the algorithm.

In the main loop of Algorithm 2, it first generates a pool $\mathcal{A}_p$ of semi-ground actions such that every state $s \in \mathcal{P}$ has at least one action $a \in \mathcal{A}_p$ that transforms s to a losing state $\tau(s, a)$ (Line 3). For every action a of $\mathcal{A}_p$, if it does not appear in the strategy δ, then the inner loop of Algorithm 2 attempts to generate a formula ψ_a and produces a rule (ψ_a, a) (Lines 4 - 17). We reduce the problem of generating the formula ϕ_a to exact binary classification problem. We then collect the set $\mathcal{P}_a$ of positive states and $\mathcal{N}_a$ of negative states for this problem (Lines 5 - 10). Every $s \in \mathcal{P}$ is a positive state for generating the formula ϕ_a, if the following conditions hold: (1) the action a is applicable over s; and (2) the successor state $\tau(s, a)$ is a losing state. In contrast, the set $\mathcal{N}_a$ includes the positive states of $\mathcal{P}$ not comply with the above two conditions and all negative states of $\mathcal{N}$. Similarly to Algorithm 1, it enumerates a set Ψ_a of arithmetic

Algorithm 2: SynWinStrat($\phi, \mathcal{P}, \mathcal{N}$)

```
   Input: $\phi$: the winnig formula of the ICG
          $\mathcal{P}$: the set of positive states
          $\mathcal{N}$: the set of negative states
   Output: $\delta$: a winnig strategy
1  $\delta \leftarrow \emptyset$
2  while true do
3      $\mathcal{A}_p \leftarrow$ EnumerateActions($\mathcal{P}$)
4      foreach $a \in \mathcal{A}_p$ s.t. $(\psi_a, a) \notin \delta$ do
5          $\mathcal{P}_a \leftarrow \emptyset$ and $\mathcal{N}_a \leftarrow \mathcal{N}$
6          foreach $s \in \mathcal{P}$ do
7              if $a$ is applicable over $s$ and $\tau(s, a) \in \mathcal{L}$ then
8                  $\mathcal{P}_a \leftarrow \mathcal{P}_a \cup \{s\}$
9              else
10                 $\mathcal{N}_a \leftarrow \mathcal{N}_a \cup \{s\}$
11         $\Psi_a \leftarrow$ EnumerateAtoms($\mathcal{P}_a, \mathcal{N}_a$)
12         $\psi_a \leftarrow$ ExactBinaryClassify($\mathcal{P}_a, \mathcal{N}_a, \Psi_a$)
13         $S' \leftarrow$ VerifyRule($\psi_a, a$)
14         if $S' = \emptyset$ then
15             $\delta \leftarrow \delta \cup \{(\psi_a, a)\}$
16         else
17             $\mathcal{P} \leftarrow \mathcal{P} \cup S'$
18     $S^* \leftarrow$ IsValid($(C \wedge \phi) \leftrightarrow [C \wedge (\bigvee_{(\psi_a,a)\in\delta} \psi_a)]$)
19     if $S^* = \emptyset$ then
20         return Simplify($\delta$)
21     else
22         $\mathcal{P} \leftarrow \mathcal{P} \cup S^*$
```

atoms of increasing sizes and then creates the candidate formula ψ_a that classifies $\mathcal{P}_a$ from $\mathcal{N}_a$.

The correctness of the rule (ψ_a, a) will be validated according to the constraint for the winning rule (cf. Definition 7) (Line 13). If the constraint is valid, then no counterexample is returned by the SMT solver. Otherwise, we are given a set S' of states certifying that ψ_a is not a desired formula for the action a. It is easily verified from the constraint for the winning rule that (1) the set S' contains only winning states; (2) the state $s \in S'$ is not an applicable state for a, or the successor state $\tau(s, a)$ is not a losing state. In addition, we also examine the completeness property of (ψ_a, a) according to the constraint for the complete rule (cf. Definition 7). This is because the action a is the unique action that changes some winning states to a losing state, but ψ_a may not involve these winning states. We therefore enlarge the set $\mathcal{P}$ by the above winning states (Line 17).

After obtaining a set δ of winning rules, we will check if δ forms a winning strategy, that is, verifying if the formula $\bigvee_{(\psi_a,a)\in\delta} \psi_a$ is the winning formula (Line 18). Since we have obtained the winning formula ϕ, it suffices to check if ϕ and $\bigvee_{(\psi_a,a)\in\delta} \psi_a$ are logically equivalent under the background theory C. If it is the case (that is, $S^* = \emptyset$), the algorithm yields the simplified winning strategy (Line 20). The simplification procedure eliminates some rules that are covered by the other rules. Given two rules (ψ_1, a_1) and (ψ_2, a_2), we say (ψ_1, a_1) covers (ψ_2, a_2), if $\psi_2 \models \psi_1$. In this case, the rule (ψ_2, a_2) can be eliminated since performing the action a_1 in every state satisfying ψ_2 leads to a losing state. If the formula $\bigvee_{(\psi_a,a)\in\delta} \psi_a$ is not the winning formula, then some winning states s do not have an semi-ground action a such that the successor state $\tau(s, a)$ is a losing state. The set $\mathcal{P}$ will be extended by S^* for the next iteration of the main loop (Line 22).

Round	$\mathcal{A}_p$	$\mathcal{P}_a$	$\mathcal{N}_a$	ψ_a	S'	S^*
1	$take(1)$	1, 5	0, 2, 3, 4	$v \equiv_4 1$	$\emptyset$	9
	$take(2)$	2	0, 1, 3, 4, 5	$v = 2$	6	
	$take(3)$	3	0, 1, 2, 4, 5	$v = 3$	7	
	$take(v)$	1, 2, 3	0, 4, 5	$v \geq 1 \wedge v < 4$	$\emptyset$	
2	$take(2)$	2, 6	0, 1, 3, 4, 5, 7, 9	$v \equiv_4 2$	$\emptyset$	$\emptyset$
	$take(3)$	3, 7	0, 1, 2, 4, 5, 6, 9	$v \equiv_4 3$	$\emptyset$	
	$take(v-4)$	5, 6, 7	0, 1, 2, 3, 4, 9	$v \geq 5 \wedge v < 8$	$\emptyset$	

Table 3: A run of synthesizing the winning strategy

Each rule generated in Algorithm 2 is guaranteed to be complete and winning. In addition, the formula $\bigvee_{(\psi_a,a)\in\delta} \psi_a$ is the winning formula. We therefore obtain soundness theorem of Algorithm 2 for synthesizing winning strategies.

Theorem 5. *Let $\Pi = \langle \mathcal{X}, \mathcal{A}, C, \mathcal{E} \rangle$ be an ICG and ϕ the winning formula of Π. The set of rules returned by Algorithm 2 is a complete and winning strategy of Π.*

Example 4. Table 3 shows the synthesis process of the winning strategy of Take-away-3 game. In the first iteration, four actions $take(1)$, $take(2)$, $take(3)$ and $take(v)$ as well as their corresponding condition ψ_a are generated. The two rules $(v = 2, take(2))$ and $(v = 3, take(3))$ are winning but incomplete. The two states 6 and 7 are the evidences of incompleteness of the above two rules, respectively. The other two rules $(v \equiv_4 1, take(1))$ and $(v \geq 1 \wedge v < 4, take(v))$ are both winning and complete and hence constitute a strategy. However, this strategy is incomplete since the formula $(v \equiv_4 1) \vee (v \geq 1 \wedge v < 4)$ ignores the winning state 9 that is a counterexample provided by the SMT solver. In the second iteration, a new action $take(v-4)$ is added into the set $\mathcal{A}_p$. The condition of this action together with $take(2)$ and $take(3)$ which fail in the first iteration are generated. The three rules $(v \equiv_4 2, take(2))$, $(v \equiv_4 3, take(3))$ and $(v \geq 5 \wedge v < 8, take(v-4))$ are winning and complete. We put them together and obtain the final winning strategy $\{(v \equiv_4 1, take(1)), (v \equiv_4 2, take(2)), (v \equiv_4 3, take(3)), (v \geq 1 \wedge v < 4, take(v)), (v \geq 5 \wedge v < 8, take(v-4))\}$.

6 EXPERIMENTAL EVALUATION

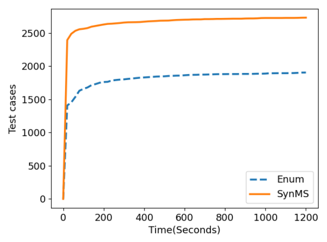
This section first presents implementation of our proposed algorithms and the benchmarks we establish in the experiments, and then compares our approach with the enumerative approach, namely Enum [33], which shows the clear advantage of our approach over Enum on the number of solved test cases.

6.1 Implementation and Benchmarks

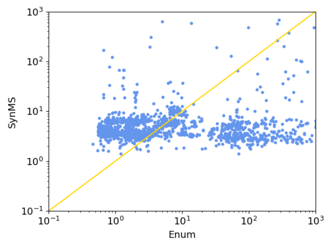
We implemented the partial MaxSAT encoding for exact binary classification illustrated in Section 4 and Algorithms 1 as well as 2 illustrated in Sections 5.1 and 5.2 in Python 3.7, respectively, called SynMS. Our implementation reads a PDDL-like file that describes the formalization of an ICG and outputs the winning formula and strategy for this ICG. We employ Z3 [28] as the SMT solver to verify the correctness of the winning formula and strategy as well as to determine if a state is winning. We adopt NuWLS [11] as the partial

Category	Test cases	Formula						Strategy					
		Solved		Time		Size		Solved		Time		Size	
		Enum	SynMS	Enum	SynMS	Enum	SynMS	Enum	SynMS	Enum	SynMS	Enum	SynMS
Sub	479	124(1)	**363**(240)	138.50(136.93)	61.16(**14.26**)	10.89(**10.80**)	61.00(26.03)	84(5)	**274**(195)	39.67(31.63)	11.50(**7.06**)	77.92(**77.40**)	216.84(110.55)
Nim	302	59(29)	**73**(43)	131.26(**22.37**)	51.82(24.73)	9.98(**8.20**)	96.90(37.33)	22(9)	**43**(30)	8.36(**0.91**)	294.57(266.84)	15.45(**8.46**)	300.21(168.31)
Welter	7	3(1)	**4**(2)	3.07(**2.71**)	7.03(6.13)	8.67(**8.50**)	76.50(88.00)	0(0)	**1**(1)	-(-)	29.22(-)	-(-)	67.00(-)
Wythoff	2924	1715(89)	**2287**(661)	47.63(44.61)	24.52(**8.11**)	8.31(**8.29**)	75.76(32.40)	308(102)	**1266**(1060)	0.24(**0.33**)	11.26(10.02)	4.16(**5.75**)	197.64(97.69)
Chomp	8	**2**(0)	**2**(0)	1.53(**1.53**)	8.18(8.18)	6.50(**6.50**)	6.50(**6.50**)	**2**(2)	0(0)	0.52(-)	-(-)	13.00(-)	-(-)
Total	3720	1903(120)	**2729**(946)	64.40(41.63)	30.54(**12.28**)	8.87(**8.46**)	63.33(38.05)	416(118)	**1584**(1286)	12.20(**10.96**)	86.64(94.64)	27.63(**30.53**)	195.42(125.51)

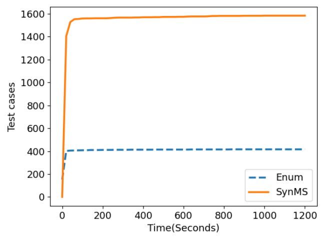
Table 4: Results of synthesizing winning formulas and strategies



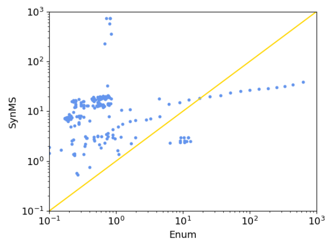
(a) Comparison on synthesizing winning formulas on the whole benchmark



(b) Comparison on synthesizing winning formulas on the commonly solved benchmark



(c) Comparison on synthesizing winning strategies on the whole benchmark



(d) Comparison on synthesizing winning strategies on the commonly solved benchmark

Figure 1: Comparison of the running time of synthesizing winning formula and strategy

MaxSAT solver in order to choose the small set of generated arithmetic atoms. NuWLS is an incomplete partial MaxSAT solver and may return a sub-optimal solution. This however has no influence on the soundness of the chosen set as the solution satisfies the hard constraint. By Theorem 3, we can always construct a formula from the chosen set that distinguishes the set of positive states from that of negative states. The time-out bounds for Z3 and NuWLS are 10 and 3 seconds, respectively. The experiments were conducted on a PC with an Intel Core i7-7700 3.60 GHz CPU and 16GB RAM running under Windows 10, with a runtime cutoff of 1, 200 seconds. Source codes are provided in this link[1].

We evaluated the performance of our approach SynMS and Enum [33] on a large number of ICG domains that consist of the following 5 categories: Subtraction [35], Nim [8], Wythoff [34], Welter [32] and Chomp [17]. Each category contains multiple classes of ICGs, including some extensions or variants of the original game. Due to space limits, the description of each test cases are given in this link[2]. Since the winning formula and strategy are not LIA-definable or unknown in some classes of ICGs, we impose certain limitations on these classes in order to guarantee that they have LIA-definable solution. For example, the winning formula of 3-rowed Chomp game is unknown. We restrict the maximum number of cookies on the 3rd row to be a fixed integer and obtain a restricted version of 3-rowed Chomp which has a LIA-definable solution. In summary, the benchmark consists of a total of 32 classes with 3, 720 test cases, including the 46 test cases in [33].

[1]https://github.com/YMH0607/SynMS/tree/master
[2]https://github.com/YMH0607/domain/tree/master

6.2 Experimental Results

We summarize the results on synthesis of winning formulas in Table 4. The column "Category" and "Test cases" denotes the name and the number of test cases of each category, respectively. The columns "Solved" denotes the number of test cases that solved by each approach, and the figure in parentheses denotes the number of test cases that solved by the corresponding approach but not by the other approach. The columns "Time" and "Size" denote the average time and size of solved test cases for each approach, and the figure in parentheses denotes the average time and size of the test cases solved by both approaches, respectively. The optimal data of each group are in bold.

Comparison on synthesizing winning formulas. Out of 3, 720 test cases, the number of test cases solved by Enum is 1, 903 test cases while SynMS is able to solve 2, 729 test cases including 946 test cases that Enum cannot solve, demonstrating a substantial improvement over Enum. For a deeper analysis of runtime behavior, we show the number of solved test cases after a certain amount of time in Figure 1(a). In only 46 seconds, SynMS is able to solve approximately 2, 500 test cases. In comparison, Enum only completes 1, 903 test cases within the timeout 1, 200 seconds. For the commonly solved 1, 783 test cases, we can see that the running time of SynMS is 12.28 seconds on average, while Enum runs in 41.63 seconds, resulting in nearly an approximate 3.39× speedup. In addition, Figure 1(b) displays the results on common benchmarks solved by both approaches, where x-axis and y-axis are the running time of Enum and SynMS, respectively. The results indicate that SynMS outperforms Enum in most cases when computing winning formulas on commonly solved benchmarks. Enum performs better only when the size of the winning formula is small, which does not exceed 25.

We observe that Enum solves 120 test cases that SynMS fails to solve. The winning formulas of these test cases are an arithmetic literal (that is, an arithmetic atom or its negation) of large size. Enum is more effective than SynMS at solving these test cases since it can enumerate formulas up to size 25. SynMS takes some time to choose arithmetic atoms and combine the chosen atoms into an arithmetic formula and hence generates arithmetic atoms of at most size 7. As for the size, SynMS generates larger formulas than Enum, with an average size 38.05 on commonly solved test cases, while Enum has an average size 8.46. In theory, Enum is guaranteed to find the smallest-size formula via exhaust enumeration [1]. SynMS constructs the winning formula in disjunctive normal form based on the selected atoms. In addition, any formula is exponentially smaller than its equivalent one in disjunctive normal form. It is noticeable that SynMS achieves a better performance in computing winning formulas compared to Enum although it sacrifices solution size.

Comparison on synthesizing winning strategies. The experimental results about synthesizing winning strategies are shown in the last 6 columns of Table 4 and Figures 1(c) and 1(d). We can make two observations. First, Enum only computes the winning strategy on 416 (21.9%) out of 1, 903 test cases of which winning formulas are successfully obtained by this approach. This is because Enum fails to refine the winning formula into several arithmetic term so that each term has a proper action. In contrast, SynMS is able to synthesize the winning strategy on 1, 584 test cases and to achieve 58.0% success rate, both dominating Enum. Algorithm 2 firstly creates a set $\mathcal{A}_p$ of actions, and then synthesizes the corresponding condition formula ψ_a for each action $a \in \mathcal{A}_p$ so that (ψ_a, a) is a winning and complete rule. So it avoids the weakness of Enum when refining the winning formula. As Figure 1(c) shows, SynMS solves more than 1, 500 test cases within only 36 seconds. In contrast, Enum only solves 416 test cases within 1, 200 seconds. Second, for the commonly solved 298 test cases, compared to SynMS, Enum is more effective and produces the winning strategy with a smaller size. This is due to the simplicity of these test cases. However, SynMS is an efficient approach to the 1, 286 test cases that Enum cannot solve.

7 RELATED WORK

Automatic methods to solve ICGs. Two classical methods for solving the winning strategy of finite-state ICGs are backward induction [22] and alpha-beta pruning [21]. Backward induction reasons backward from the ending state to the given initial state. Alpha-beta pruning reduces the number of nodes in the search tree that captures the plays from the given initial state by two players. Recently, Beling and Rogalski [6] proposed a novel method to prune the search tree based on the node values calculated by the Sprague-Grundy function. But the above methods do not work for infinite-state ICGs. Luo and Liu [25] extended the situation calculus for formalizing ICGs, and proposed a verification method for the correctness of winning strategies that are defined in finite state automaton. Their approach however does not address the strategy synthesis problem. Later, they [26] developed a CEGIS approach for synthesizing invariant strategies. An invariant strategy is of the form $\phi?; \pi a.a; \varphi?$, meaning whenever ϕ holds, the player can execute an action to enforce φ, and whenever φ holds, executing any action makes φ true regardless of what the action chooses. In fact, the formulas ϕ and φ are the winning and the losing formulas, respectively. Strictly speaking, Luo and Liu [26]'s approach only constructs winning and losing formulas but not winning strategies that specifies which action to do in different partitions of winning states (cf. Definition 4). In addition, as reported in [26], their approach is much less efficient than Enum. Hence, we do not compare with their approach in experimental evaluation.

LTL synthesis. Pnueli and Rosner [29] first proposed a well-known class of games, namely *linear temporal logic (LTL) games*, where the game is represented by an automaton, and the goal is expressed by a propositional LTL formula. The objective of LTL synthesis is to synthesize a strategy that ensures goal achievement. The LTL synthesis problem has been addressed in numerous literature, such as [4, 9, 18, 23, 39]. However, the above works only deal with the synthesis of winning strategies for finite-state games and cannot be applied to infinite-state games. The synthesis of winning strategies has been extended to infinite-state games by a number of ways [3, 7, 13, 14], but these approaches are not suitable for ICGs. The reasons are the following. Firstly, these approaches only generates a winning strategy for a specific initial state, and the generated strategy cannot be generalized to all legal states. Additionally, the formalization of these approaches are based on linear real arithmetic (LRA) rather than LIA. The differences between LIA and LRA are the following: (1) variables in LIA have an integer domain rather than real one; and (2) LIA contains the congruence modulo relation while LRA does not. The majority of ICGs contain only integer variables and require congruence modulo relation in their winning formula.

8 CONCLUSIONS

In this paper, we have introduced a novel method for synthesizing the winning formula and strategy of ICGs over infinite states. To this end, we have put forward the exact binary classification problem that produces an arithmetic formula to separate the given set of positive states from that of negative states, and devised a partial MaxSAT-based encoding to solve this problem. Then, by integrating the partial MaxSAT-based encoding and the SMT-based verification procedure, we have developed a CEGIS approach to synthesizing the winning formula and strategy of ICGs, and evaluated the performance of our approach in comparison to the Enum approach on an ICG benchmark with a significant number of test cases. Experimental results show that our method has better performance and efficiency.

ACKNOWLEDGMENTS

We are grateful to Shaowei Cai and Yi Chu for sharing the source code of the partial MaxSAT solver NuWLS. We also appreciate the help of Yalin Luo and Siyue Zheng in collecting ICG test cases. This paper was supported by the Science and Technology Planning Project of Guangdong (2022A0505030025), National Natural Science Foundation of China (62077028, 62276114), Guangdong Basic and Applied Basic Research Foundation (2021A1515011873), Guangdong-Macao Advanced Intelligent Computing Joint Laboratory (2020B1212030003), the Fundamental Research Funds for the Central Universities, JNU (21623202).

REFERENCES

[1] Rajeev Alur, Arjun Radhakrishna, and Abhishek Udupa. 2017. Scaling Enumerative Program Synthesis via Divide and Conquer. In *Proceedings of Twenty-Third International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS-2017)*. Lecture Notes in Computer Science, Vol. 10205. 319–336.

[2] Nicolas Anastassacos, Stephen Hailes, and Mirco Musolesi. 2020. Partner Selection for the Emergence of Cooperation in Multi-agent Systems Using Reinforcement Learning. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-2020)*, Vol. 34. 7047–7054.

[3] Christel Baier, Norine Coenen, Bernd Finkbeiner, Florian Funke, Simon Jantsch, and Julian Siber. 2021. Causality-Based Game Solving. In *Proceedings of the Thirty-Third International Conference on Computer Aided Verification (CAV-2021)*. Lecture Notes in Computer Science, Vol. 12759. 894–917.

[4] Mrudula Balachander, Emmanuel Filiot, and Jean-François Raskin. 2023. LTL Reactive Synthesis with a Few Hints. In *Proceedings of the Twenty-Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS-2023)*. Lecture Notes in Computer Science, Vol. 13994. 309–328.

[5] Gabriel Beaulieu, Kyle Burke, and Eric Duchêne. 2013. Impartial Coloring Games. *Theoretical Computer Science* 485 (2013), 49–60.

[6] Piotr Beling and Marek Rogalski. 2020. On Pruning Search Trees of Impartial Games. *Artificial Intelligence* 283 (2020), 103262.

[7] Tewodros Beyene, Swarat Chaudhuri, Corneliu Popeea, and Andrey Rybalchenko. 2014. A Constraint-based Approach to Solving Games on Infinite Graphs. In *Proceedings of the Forty-First ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL-2014)*. 221–233.

[8] Charles L. Bouton. 1901. Nim, a Game with a Complete Mathematical Theory. *Annals of Mathematics* 3, 1/4 (1901), 35–39.

[9] Alberto Camacho, Christian Muise, Jorge A. Baier, and Sheila A. McIlraith. 2018. LTL Realizability via Safety and Reachability Games. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-2018)*. 4683–4691.

[10] Tatjana Chavdarova and Francois Fleuret. 2018. Sgan: an Alternative Training of Generative Adversarial Networks. In *Proceedings of 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR-2018)*. 9407–9415.

[11] Yi Chu, Shaowei Cai, and Chuan Luo. 2023. NuWLS: Improving Local Search for (Weighted) Partial MaxSAT by New Weighting Techniques. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI-2023)*. 3915–3923.

[12] David C. Cooper. 1972. Theorem Proving in Arithmetic without Multiplication. *Machine Intelligence* 7, 91-99 (1972), 300.

[13] Marco Faella and Gennaro Parlato. 2023. Reachability Games Modulo Theories with a Bounded Safety Player. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI-2023)*. 6330–6337.

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

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

[16] Barbara Franci and Sergio Grammatico. 2020. A Game–theoretic Approach for Generative Adversarial Networks. In *Proceedings of the Fifty-Ninth IEEE Conference on Decision and Control (CDC-2020)*. 1646–1651.

[17] Schuh Frederik. 1952. the Game of Divisions. *Nieuw Tijdschrift voor Wiskunde* 39 (1952), 299–304.

[18] Giuseppe De Giacomo, Marco Favorito, Jianwen Li, Moshe Y. Vardi, Shengping Xiao, and Shufang Zhu. 2022. LTL$_f$ Synthesis as AND-OR Graph Search: Knowledge Compilation at Work. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI-2022)*. 2591–2598.

[19] Solomon W. Golomb. 1966. A Mathematical Investigation of Games of "take-away". *Journal of Combinatorial Theory* 1, 4 (1966), 443–458.

[20] Vladimir Gurvich and Nhan Bao Ho. 2018. On Tame, Pet, Domestic, and Miserable Impartial Games. *Discrete Applied Mathematics* 243 (2018), 54–72.

[21] George T. Heineman, Gary Pollice, and Stanley Selkow. 2008. *Algorithms in a Nutshell*. O'Reilly Media.

[22] Watson Joel. 2002. *Strategy: an Introduction to Game Theory*. Vol. 139. WW Norton New York.

[23] Jan Kretínský, Tobias Meggendorfer, Maximilian Prokop, and Sabine Rieder. 2023. Guessing Winning Policies in LTL Synthesis by Semantic Learning. In *Proceedings of the Thirty-Fifth International Conference on Computer Aided Verification (CAV-2023)*. Lecture Notes in Computer Science, Vol. 13964. 390–414.

[24] Wenan Liu and Meile Zhao. 2015. General Restrictions of Wythoff-like Games. *Theoretical Computer Science* 602 (2015), 80–88.

[25] Kailun Luo and Yongmei Liu. 2019. Automatic Verification of FSA Strategies via Counterexample-Guided Local Search for Invariants. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-2019)*. 1814–1821.

[26] Kailun Luo and Yongmei Liu. 2022. Automated Synthesis of Generalized Invariant Strategies via Counterexample-Guided Strategy Refinement. In *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI-2022)*. 5800–5808.

[27] David Monniaux. 2010. Quantifier Elimination by Lazy Model Enumeration. In *Proceedings of the Twenty-Second International Conference on Computer Aided Verification (CAV-2010)*. Lecture Notes in Computer Science, Vol. 6174. 585–599.

[28] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: an Efficient SMT Solver. In *Proceedings of the Fourteenth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS-2008)*. Lecture Notes in Computer Science, Vol. 4963. 337–340.

[29] Amir Pnueli and Roni Rosner. 1989. On the Synthesis of a Reactive Module. In *Proceedings of the Sixteenth ACM SIGPLAN-SIGACT Symposium Conference on Principles of programming languages (POPL-1989)*. 179–190.

[30] Eric Sopena. 2016. *i*-Mark: A new subtraction division game. *Theoretical Computer Science* 627 (2016), 90–101.

[31] Abhishek Udupa, Arun Raghavan, Jyotirmoy V. Deshmukh, Sela Mador-Haim, Milo M. K. Martin, and Rajeev Alur. 2013. TRANSIT: Specifying Protocols with Concolic Snippets. In *Proceedings of the Thirty-Fourth Annual ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI-2013)*. 287–296.

[32] C. P. Welter. 1952. the Advancing Operation in a Special Abelian Group. *Indagationes Mathematicae* 55 (1952), 304–314.

[33] Kaisheng Wu, Liangda Fang, Liping Xiong, Zhao-Rong Lai, Yong Qiao, Kaidong Chen, and Fei Rong. 2020. Automatic Synthesis of Generalized Winning Strategies of Impartial Combinatorial Games Using SMT Solvers. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-2020)*. 1703–1711.

[34] Willem A. Wythoff. 1907. A Modification of the Game of Nim. *Nieuw Archief voor Wiskunde* 7 (1907), 199–202.

[35] I. M. Yaglom. 2001. Two Games with Matchsticks. *Kvant Selecta: Combinatorics, I* 1 (2001), 1–8.

[36] Mingfeng Yuan, Jinjun Shan, and Kevin Mi. 2021. Deep Reinforcement Learning Based Game-theoretic Decision-making for Autonomous Vehicles. *IEEE Robotics and Automation Letters* 7, 2 (2021), 818–825.

[37] Jie Zhang, Gang Wang, Shaohua Yue, Yafei Song, Jiayi Liu, and Xiaoqiang Yao. 2020. Multi-agent System Application in Accordance with Game Theory in Bi-directional Coordination Network Model. *Journal of Systems Engineering and Electronics* 31, 2 (2020), 279–289.

[38] Liyuan Zheng, Tanner Fiez, Zane Alumbaugh, Benjamin Chasnov, and Lillian J Ratliff. 2022. Stackelberg Actor-critic: Game-theoretic Reinforcement Learning Algorithms. In *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI-2022)*. 9217–9224.

[39] Shufang Zhu, Lucas M. Tabajara, Jianwen Li, Geguang Pu, and Moshe Y. Vardi. 2017. Symbolic LTL$_f$ Synthesis. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-2017)*. 1362–1369.