Verification of General Games with QBF Solvers

Yifan He[1], Abdallah Saffidine[2] and Michael Thielscher[1]

[1]UNSW Sydney, Australia
[2]Potassco Solutions, Germany

Abstract

The Game Description Language (GDL) is a lightweight formalism for representing the rules of arbitrary finite perfect information games. Its purpose is to build general game-playing systems, that is, automated players that can understand the rules of games and learn how to play them without human intervention. Coalition Logic (CL) is a logical framework capable of expressing strategic behaviors of a set of players that involve finitely many successive game states, which can formulate important properties such as whether one player can enforce a win of the game within a certain number of steps, regardless of the actions of the other players. In this paper, we investigate how to define the CL model-checking problem in the context of GGP and how to reason about CL properties of general games using Quantified Boolean Formula (QBF) solvers. This work extends our earlier AAMAS 2024 paper [1]. We evaluate the efficiency of our approach through a case study involving two-player general games and show that it has the potential to assist general game-playing agents with endgame analysis.

Keywords

Logic Programming, Quantified Boolean Formula, Reasoning about actions, General Game Playing

1. Introduction

The Game Description Language (GDL) is a lightweight knowledge representation formalism for describing the rules of arbitrary games [2]. GDL describes game rules in normal logic program syntax similar to Prolog. It is used as the input language for general game-playing (GGP) systems, which can learn to play any new game from its rules alone and without human intervention, thereby demonstrating a form of general intelligence. As a lightweight specification language, GDL merely provides means for *representing* the rules of a game. At the same time, a crucial aspect of GGP is the ability to automatically *reason* about a given specification. In early years, successful GGP systems performed random simulations to test the validity of certain properties and rely on their informed guess in case no violation could be detected [3]. However, validating properties through random simulations can be unreliable, and the performance of the GGP system can be affected if incorrect assumptions are made about the game.

Due to the similarity between the semantics of GDL and interpreted systems, model-checking techniques have been introduced to support GGP systems in formally analyzing whether a GDL description satisfies certain logical properties. Notable work in this area include using the model-checker Mocha to reason about properties of GDL games specified in Alternating-time Temporal logic (ATL) [4] and using Answer Set Programming to reason about General Game Temporal Logic (GTL) properties–a logic similar to the Linear Temporal Logic with only the temporal operator "next"– of GDL games [5]. However, model-checking against ATL typically requires expanding all game states to construct the underlying Concurrent Game Structure, which is only practical for very small games. While model-checking against GTL has much lower computational complexity and does not require explicitly storing all game states in memory, GTL cannot express any strategic properties. Hence, it is interesting to investigate whether there is some logical framework that can express strategic properties of general games, and whose model-checker has the potential to be used by GGP systems in practice.

Coalition Logic (CL) [6], a logical fragment of ATL that includes only the temporal operator "next," can express strategic behaviors of groups of players over finitely many successive game states. For instance, it can capture important solution concepts such as *bounded-depth strong winnability*, which asks whether a player can enforce a win within a certain number of steps, regardless of the actions of

23rd International Workshop on Nonmonotonic Reasoning, November 11-13, 2025, Melbourne, Australia
yifan.he1@unsw.edu.au (Y. He); abdallah.saffidine@gmail.com (A. Saffidine); mit@unsw.edu.au (M. Thielscher)

the other players. Importantly, model checking for CL has significantly lower computational complexity than for ATL, making it a more practical choice for use in GGP systems.

In this paper, we establish a concrete link between GDL and CL and present a potentially practical approach to reasoning about CL properties of GDL games. To this end, we define the syntax and semantics of CL for general games (GCL). We present the complexity of GCL model-checking and show how some important game properties of GDL games can be formulated with GCL. We present a sound and complete method of GCL model-checking leveraging Quantified Boolean Formula (QBF) solvers [7]. We briefly report the efficiency of our method with a case study on solving two-player games.

Our work is related to the QBF-based approach of reasoning about bounded-depth strong winnability of *specific* perfect information games such as Generalized Tic-Tac-Toe and Hex [8]. Such an approach usually involves encoding the bounded-depth strong winnability of some specific strategic games with QBF based on human interpretation of the game rules and calling a QBF solver to evaluate the expression. It has been shown in Generalized Tic-Tac-Toe that QBF solvers can prove bounded-depth strong winnability faster than proof number search solvers [9]. Since bounded-depth strong winnability is just one property that can be expressed in GCL, and GDL can express *all* finite perfect information games, our work is a generalization of existing literature.

2. Preliminaries

We assume readers to be familiar with basic concepts of logic programming with negation, Answer Set Programming (ASP) [10], and Quantified Boolean Formula (QBF) [7].

2.1. Game Description Language

The Game Description Language (GDL) can be used to describe the rules of any finite game with concurrent moves. GDL uses a normal logic program syntax along with the following preserved keywords used to describe the different elements of a game [2]:

$role(P)$	P is a player
$base(F)$	F is a base proposition for game positions
$input(P, A)$	Action A is in the move domain of player P
$init(F)$	base proposition F holds in the initial position
$true(F)$	base proposition F holds in the current position
$legal(P, M)$	P can do move M in the current position
$does(P, M)$	player P does move M
$next(F)$	F holds in the next position
$terminal$	the current position is terminal
$goal(P, N)$	P gets N points in the current position

There are further restrictions for a set of GDL rules to be **valid** [2]: *role* can appear only in facts; *init* and *next* can only appear as heads of rules; *true* and *does* can only appear in rule bodies. Moreover, *init* cannot depend on *true*, *does*, *legal*, *next*, *terminal*, or *goal* while *legal*, *terminal*, and *goal* cannot depend on *does*. Finally, valid game descriptions must be *stratified* and *allowed*—such logic programs always admit a finite grounding and a unique stable model.

A valid GDL game description G over ground terms Σ can be interpreted as a multi-agent state transition system: Let $\beta = \{f \in \Sigma \mid G \models base(f)\}$ be the **base propositions** and $\gamma = \{(p, a) \in \Sigma \times \Sigma \mid G \models input(p, a)\}$ the **move domain** for the players. Suppose that $S = \{f_1, \dots, f_n\} \subseteq \beta$ is any given position and $A = \{p_1, \dots, p_k\} \to \Sigma$ any function that assigns to each of $k \geq 1$ players an action from their move domain. In order to use the game rules G to determine the state update, S needs to be encoded as a set of facts using keyword *true*: $S^{true} = \{true(f_1)., \dots, true(f_n).\}$ and the joint action A by a set of facts using keyword *does*: $A^{does} = \{does(p_1, A(p_1))., \dots, does(p_k, A(p_k)).\}$.

Definition 1 ([11]). *The* semantics *of a valid GDL description G is the transition system* (R, S_0, T, l, u, g)

- $R = \{p \in \Sigma \mid G \models role(p)\}$ *(player names)*

- $S_0 = \{f \in \beta \mid G \models init(f)\}$ *(initial state)*
- $T = \{S \subseteq \beta \mid G \cup S^{true} \models terminal\}$ *(terminal states)*
- $l = \{(p, a, S) \mid G \cup S^{true} \models legal(p, a)\}$ *(legal moves)*
- $u(A, S) = \{f \in \beta \mid G \cup S^{true} \cup A^{does} \models next(f)\}$ *(update)*
- $g = \{(p, v, S) \mid G \cup S^{true} \models goal(p, v)$ *and* $v \in \mathbb{N}$ *and* $0 \leq v \leq 100\}$ *(goal value)*

Based on the above definition, we represent a *valid game playing sequence* of n steps as follows.

$$S_0 \xrightarrow{A_1} S_1 \xrightarrow{A_2} \ldots S_{n-1} \xrightarrow{A_n} S_n$$

In the above game sequence, we write $S_i \xrightarrow{A_{i+1}} S_{i+1}$ if $S_i \notin T$ and all moves are legal in the state in which they are taken, that is, $(r, A_{i+1}(r), S_i) \in l$ for each $r \in R$. We say a valid game playing sequence *terminates in n steps* if $S_n \in T$ [5].

2.2. Quantified Answer Set Programming

Similar to the difference between QBF and SAT [12], allowing quantifiers in ASP programs can provide a more expressive language. The resulting language is called Quantified Answer Set Programming (QASP). Its semantics is defined as follows.

Definition 2 ([13])**.** *Let P be a logic program with ground atoms* **A**. *A QASP over* **A** *has the form*

$$Q_1\ X_1 \ldots Q_n\ X_n\ \ P$$

where X_i are pairwise disjoint subsets of **A**, *every Q_i is either* $\exists$ *or* $\forall$, *and P is a logic program over* **A**. *We define $fix(X, Y)$, where $Y \subseteq X \subseteq$* **A**, *as the logic program:* $\{:\text{- not } x. \mid x \in Y\} \cup \{:\text{- } x. \mid x \in X \setminus Y\}$.

A normal logic program P is satisfiable iff it has a stable model. Satisfiability of a QASP is recursively defined as follows.

1. *If the QASP has form* $\exists\ X\ P$ *(resp.* $\forall\ X\ P$*), the program is satisfiable iff there exists (resp. for all)* $Y \subseteq X$ *such that the program* $P \cup fix(X, Y)$ *is satisfiable.*
2. *If the QASP has form* $\exists\ X\ \mathbf{Q}\ P$ *(resp.* $\forall\ X\ \mathbf{Q}\ P$*), the program is satisfiable iff there exists (resp. for all)* $Y \subseteq X$ *such that the program* $\mathbf{Q}\ (P \cup fix(X, Y))$ *is satisfiable.*

QASP has the same expressive power as QBF. A QASP can be solved by converting it to an equal-satisfiable QBF expression using tools like *qasp2qbf* [13] and calling a QBF solver to evaluate the satisfiability of the converted expression.

3. Coalition Logic For General Games

We define the Coalition Logic for General Games (GCL) that can model the strategic behavior of different players that involve finitely many successive game states.

Definition 3 (GCL Syntax)**.** *The Coalition Logic for General Games is defined over a valid GDL description G that has the following grammar:*

$$\varphi ::= q \mid \varphi \wedge \varphi \mid \neg\varphi \mid \langle\langle C \rangle\rangle \mathbf{X}\varphi$$

where q is a ground atom of G that is not an instance of the predicate symbol ***init***, ***next*** *and does not depend on* ***does***. *C is a subset of players in G (i.e., $C \subseteq R$). Standard propositional operators* $\vee$ *and* $\rightarrow$ *are also allowed and defined as usual.*

GCL is based on the Coalition Logic [6] and similar to the Alternating Temporal Logic with finite traces [14], with only the primitive temporal operator $\langle\langle C\rangle\rangle \mathbf{X}\varphi$ (read "the current state is non-terminal and the players in C can enforce φ to hold next"). We introduce the abbreviation $[[C]]\,\widetilde{\mathbf{X}}\neg\varphi ::= \neg\langle\langle C\rangle\rangle \mathbf{X}\varphi$ (read "the current state is terminal or no matter what the players in C do, players in $R \setminus C$ have a set of legal joint moves to ensure φ does not hold next").

For a GDL description and any subset of players $C = \{p_1, \ldots, p_{|C|}\}$ ($C \subseteq R$), we define the notation $L(C,S) = \{\{does(p_1,a_1).,\ldots,does(p_{|C|},a_{|C|}).\} \mid \forall\ p_i \in C,\ G \cup S^{true} \models legal(p_i,a_i)\}$, which denotes the set of all possible legal joint actions of players in a coalition C at state S. The semantics of GCL is given as follows.

Definition 4 (GCL Semantics). *Let G be a valid GDL description, and φ is a GCL formula over G. Then, we say G satisfies φ at S (written $G,S \models_t \varphi$) as per the following inductive definition:*

$G,S \models_t q$	*iff*	$G \cup S^{true} \models q$ (q *ground atom*)
$G,S \models_t \neg\varphi$	*iff*	$G,S \not\models_t \varphi$
$G,S \models_t \varphi_1 \wedge \varphi_2$	*iff*	$G,S \models_t \varphi_1$ ***and*** $G,S \models_t \varphi_2$
$G,S \models_t \langle\langle C\rangle\rangle \mathbf{X}\varphi$	*iff*	$G \cup S^{true} \not\models terminal$ ***and*** $\exists A_e \in L(C,S)$. *such that* $\forall A_u \in L(R \setminus C,S)$. $G,S' \models_t \varphi$ *where* $S' = \{f \in \beta \mid G \cup A_e \cup A_u \cup S^{true} \models next(f)\}$

For a GDL description G and a formula φ, the GCL model-checking task decides if $G,S_0 \models_t \varphi$ holds, which is abbreviated as $G \models_t \varphi$.

We now show how some general game properties can be expressed as GCL formulas. First, similar to GTL, GCL can express non-strategic properties that involve finitely many successive game states. For example, *universal termination* within n steps can be expressed as follows.

- $\varphi_{term}(n) \equiv \varphi_{term}(0) \vee [[R]]\,\widetilde{\mathbf{X}}\,\varphi_{term}(n-1)$, where $\varphi_{term}(0) \equiv terminal$.

Turn-taking is another important non-strategic property, which requires that at every step of the game, exactly one player is allowed to take a turn—i.e., has more than one legal action. For games that are guaranteed to terminate within n steps, this property can be expressed using the following GCL formula.

- $\varphi_{turn}(n) \equiv \varphi_{turn}(0) \wedge [[R]]\,\widetilde{\mathbf{X}}\,\varphi_{turn}(n-1)$, where $\varphi_{turn}(0) \equiv \neg \bigvee_{r_i \in R \wedge r_j \in R \wedge r_i \neq r_j} (\varphi_{two}(r_i) \wedge \varphi_{two}(r_j))$, and
- $\varphi_{two}(r) \equiv \bigvee_{(r,a)\in\gamma \wedge (r,b)\in\gamma \wedge a\neq b} legal(r,a) \wedge legal(r,b)$ (recall that γ is the move domain of all players)

Besides non-strategic properties, GCL can also formulate strategic properties. A general game player might be concerned about whether it can win the game within n steps (aka. *bounded-depth strong winnability*) regardless of the actions of the remaining players. This can be expressed as follows:

- $\varphi_{win}(p,n) \equiv \varphi_{win}(p,0) \vee \langle\langle\{p\}\rangle\rangle \mathbf{X}\,\varphi_{win}(p,n-1)$, where $\varphi_{win}(p,0) \equiv goal(p,100) \wedge terminal$.

The existence of a *threatening move* is another interesting strategic property. In a two-player turn-taking game, if the players take alternating moves (i.e., one player might have more than 1 legal action in odd steps and the other player might have more than 1 legal action in even steps), then, we say that a player p that is taking turn has a threatening move if there exists a legal action for p in the current state such that, in the next state, the opponent has at most one legal response that prevents p from winning the game within k steps (for some small value of k). For example, in Tic-Tac-Toe, player x has a threatening move in step 1 by marking the corner-cell $(1,1)$. Because the only not losing move for o in the next state is to mark the center cell $(2,2)$. For all the other 7 possible moves for o, x can force a win within 5 additional steps. To express the existence of a *threatening move* for player p in GCL, we must augment the original GCL game description G with the following rule, which says that if a player R plays the action A in the current state, $true(done(R,A))$ holds in the next state.

- $next(done(R,A))$:- $does(R,A)$.

With this new rule, we can express the existence of a *threatening move* for player p as follows:

- $\varphi_{threat}(p,k) \equiv \langle\!\langle \{p\} \rangle\!\rangle \mathbf{X}(\varphi_{win}(p,0) \vee \varphi_{one}(q,k))$, where
- $\varphi_{one}(q,k) \equiv \bigwedge_{(q,a),(q,b) \in \gamma \wedge a \neq b} \varphi_{response}(q,a,k) \vee \varphi_{response}(q,b,k)$,
- $\varphi_{response}(q,a,k) \equiv [[q]]\,\widetilde{\mathbf{X}} \neg true(done(q,a)) \vee \varphi_{win}(p,k)$

Here, $\varphi_{response}(q,a,k)$ checks that for all legal moves of player q, either q does not perform action a or player p can force a win within k steps. In other words, $\varphi_{response}(q,a,k)$ does not hold if and only if player q can play action a to ensure that p cannot force a win within k steps. $\varphi_{one}(q,k)$ ensures that every pair of actions a and b in the move domain of q, either $\varphi_{response}(q,a,k)$ or $\varphi_{response}(q,b,k)$ hold. This means that q has at most 1 legal action in the next state to ensure that p cannot enforce a win within k steps. We end this section by stating the complexity result of GCL model-checking.

Theorem 1. *GCL model-checking over a ground GDL description G and a formula φ is PSPACE-complete.*

Proof. GCL model-checking is in PSPACE because one can perform model-checking by following the semantics definition 4, and the trivial recursive algorithm uses a polynomial amount of space. The hardness result is based on the fact that verifying bounded-depth strong winnability for two-player games when the depth is encoded in unary is PSPACE-hard [15]. □

Readers familiar with CL might wonder why the complexity of model-checking against a ground GDL description is PSPACE instead of PTIME when applied to Concurrent Game Structures. This is because GDL can represent game models that are exponentially larger than the size of the GDL description.

4. QASP Encoding for GCL Model-Checking

One can perform GCL model-checking naively by following its semantics definition; however, it is inefficient (see Section 5 for more details). Due to the PSPACE-completeness of GCL model-checking, using a QBF solver to perform the task is a viable option. We now discuss how to perform GCL model-checking with QBF solvers. GDL and QASP utilize stable models for their semantics, whereas QBF is based on classical models. Encoding the GCL model-checking task in QBF directly is therefore challenging as it would require some form of completion technique [16]. Thanks to work on converting from QASP to QBF [13], dealing with the completion task from scratch is unnecessary as long as we can encode GCL model-checking in QASP.

To perform GCL model-checking expressions with QASP, we need to convert the GCL formula into negation normal form (NNF), where the negation symbol only appears in front of ground atoms of G (i.e., we substitute $\neg\varphi$ in Definition 9 with $\neg q$). The **only** other allowed operators are $\vee$, $\wedge$, $[[C]]\,\widetilde{\mathbf{X}}$, and $\langle\!\langle C \rangle\!\rangle \mathbf{X}$. The procedure of rewriting GCL formulas to NNF is similar to rewriting other modal logics to NNF, which is based on the elimination of $\rightarrow$, the definition of $[[C]]\,\widetilde{\mathbf{X}}$, and the use of De Morgan's laws [17]. Similar to other modal logics, if the input GCL only contains the operators $\neg$, $\vee$, $\wedge$, $\rightarrow$, $\langle\!\langle C \rangle\!\rangle \mathbf{X}$, and $[[C]]\,\widetilde{\mathbf{X}}$, we can ensure that the converted NNF has at most a *linear growth* in the formula size [17].

Based on the semantics of GCL, we know that whether $G \models_t \varphi$ depends on the truth value of each of the subformulas of φ at different states of G. We now introduce the concept of subformula positions, which is a mapping from subformulas of φ to their unique operator prefix in the syntax tree.

Definition 5 (Subformula Position)**.** *Given a GCL formula φ, the **position** of each subformula ψ in φ (denoted as $pos(\psi)$) is recursively defined as follows:*

- *If $pos(\neg\psi) = \pi$, then $pos(\psi) = \pi\neg$ (similarly for $[[C]]\,\widetilde{\mathbf{X}}$ and $\langle\!\langle C \rangle\!\rangle \mathbf{X}$).*
- *If $pos(\psi_1 \wedge \psi_2) = \pi$, then $pos(\psi_1) = \pi\wedge_1$ and $pos(\psi_2) = \pi\wedge_2$ (similarly for $\psi_1 \vee \psi_2$).*

We define $pos(\varphi) = \epsilon$ and denote Pos_φ as the set of all positions of all subformulas of φ.

Although each subformula of a GCL expression φ has a different position, when verifying φ, the truth value of different subformulas in φ might be evaluated at the same state of G. We provide a name to each position of Pos_φ so that two subformulas have the same position naming if and only if they are within the same scope of $\langle\langle C\rangle\rangle \mathbf{X}$ and $[[C]]\,\widetilde{\mathbf{X}}$, and as a result, their truth value are always evaluated at the same state of G. Our notion of subformula positions and position naming is inspired by similar concepts used in epistemic GTL model-checking [18].

Definition 6 (Position Naming)**.** *Let G be a valid GDL description and φ a GCL formula. A position naming for φ is a function $\mathscr{V}_\varphi : Pos_\varphi \to \mathbb{N}$ such that $\mathscr{V}_\varphi(\epsilon) = 0$; and for all positions $\pi_1, \pi_2 \in Pos_\varphi$ and their longest prefixes π_1' and π_2' which ends in some $\langle\langle C\rangle\rangle \mathbf{X}$ or $[[C]]\,\widetilde{\mathbf{X}}$ we have that $\mathscr{V}_\varphi(\pi_1) = \mathscr{V}_\varphi(\pi_2)$ iff $\pi_1' = \pi_2'$ (where $\pi_i' = \epsilon$ if no such $\langle\langle C\rangle\rangle \mathbf{X}$ or $[[C]]\,\widetilde{\mathbf{X}}$ exists).*

We define the *depth* of a position name $\mathscr{V}_\varphi(\pi)$ (denoted as $depth(\mathscr{V}_\varphi(\pi))$) as the total number of $\langle\langle C\rangle\rangle \mathbf{X}$ or $[[C]]\,\widetilde{\mathbf{X}}$ operators in π. The *degree* of a GCL expression φ (denoted as $deg(\varphi)$) is the maximum possible depth of any position name $\mathscr{V}_\varphi(\pi)$ such that $\pi \in Pos_\varphi$, which is also equal to the maximum nesting of $\langle\langle C\rangle\rangle \mathbf{X}$ or $[[C]]\,\widetilde{\mathbf{X}}$ operators in the syntax tree of φ.

Example 1. *Consider the formula $\varphi_1 = (\langle\langle\{x\}\rangle\rangle \mathbf{X}\, goal(x, 100) \wedge terminal) \vee ([[\{o\}]]\,\widetilde{\mathbf{X}}\, goal(o, 100) \wedge terminal)$, which says that either x can force a win in the next state or no matter what o does, x can let o win in the next state. The positions of the subformulas of φ_1 and a possible valid position naming is as follows:*

- $pos(\varphi_1) = \epsilon$, *and* $\mathscr{V}_{\varphi_1}(\epsilon) = 0$
- $pos(\langle\langle\{x\}\rangle\rangle \mathbf{X}\, goal(x, 100) \wedge terminal^1) = \epsilon \vee_1$, *and* $\mathscr{V}_{\varphi_1}(\epsilon \vee_1) = 0$
- $pos(goal(x, 100) \wedge terminal^1) = \epsilon \vee_1 \langle\langle\{x\}\rangle\rangle \mathbf{X}$, *and* $\mathscr{V}_{\varphi_1}(\epsilon \vee_1 \langle\langle\{x\}\rangle\rangle \mathbf{X}) = 1$
- $pos(goal(x, 100)) = \epsilon \vee_1 \langle\langle\{x\}\rangle\rangle \mathbf{X} \wedge_1$, *and* $\mathscr{V}_{\varphi_1}(\epsilon \vee_1 \langle\langle\{x\}\rangle\rangle \mathbf{X} \wedge_1) = 1$
- $pos(terminal^1) = \epsilon \vee_1 \langle\langle\{x\}\rangle\rangle \mathbf{X} \wedge_2$, *and* $\mathscr{V}_{\varphi_1}(\epsilon \vee_1 \langle\langle\{x\}\rangle\rangle \mathbf{X} \wedge_2) = 1$
- $pos([[\{o\}]]\,\widetilde{\mathbf{X}}\, goal(o, 100) \wedge terminal^2) = \epsilon \vee_2$, *and* $\mathscr{V}_{\varphi_1}(\epsilon \vee_2) = 0$
- $pos(goal(o, 100) \wedge terminal^2) = \epsilon \vee_2 [[\{o\}]]\,\widetilde{\mathbf{X}}$, *and* $\mathscr{V}_{\varphi_1}(\epsilon \vee_2 [[\{o\}]]\,\widetilde{\mathbf{X}}) = 2$
- $pos(goal(o, 100)) = \epsilon \vee_2 [[\{o\}]]\,\widetilde{\mathbf{X}} \wedge_1$, *and* $\mathscr{V}_{\varphi_1}(\epsilon \vee_2 [[\{o\}]]\,\widetilde{\mathbf{X}} \wedge_1) = 2$
- $pos(terminal^2) = \epsilon \vee_2 [[\{o\}]]\,\widetilde{\mathbf{X}} \wedge_2$, *and* $\mathscr{V}_{\varphi_1}(\epsilon \vee_2 [[\{o\}]]\,\widetilde{\mathbf{X}} \wedge_2) = 2$

Although terminal appears twice in φ_1, they are different subformulas of φ_1. For clarity, we use $terminal^1$ and $terminal^2$ to distinguish them. The position naming above ensures that any subformulas within the same scope of the temporal operator have the same naming. For example, when performing GCL model-checking against φ_1, we know that $goal(o, 100)$ and $terminal^2$ should always be evaluated at the same state to decide the truth value of the subformula $goal(o, 100) \wedge terminal^2$, thus, they should have the same position naming. For the valid position naming provided in this example, we can ensure this requirement because the subformulas $goal(o, 100)$, $terminal^2$, and $goal(o, 100) \wedge terminal^2$ all have the same position naming 2.

The depth of the position names is given as: $depth(0) = 0$ and $depth(1) = depth(2) = 1$; and $deg(\varphi) = 1$.

Based on the semantics of GCL, we know that whether a GCL formula φ holds at a particular state S of G depends on the evaluation of all subformulas φ' of φ that have the same position name as φ at state S and the evaluation of all subformulas φ'' that have exactly 1 more nesting of temporal operators at some successor state S' of S. We now introduce the concept of successor position, which refers to all positions that have exactly 1 more nesting of the temporal operators than the current position.

Definition 7 (Successor Position Name). *Suppose that G is a valid GDL description, φ is a GCL formula, π_1 and π_2 are two positions in Pos_φ, and $\mathscr{V}_\varphi : Pos_\varphi \to \mathbb{N}$ is a valid position naming function. A position name $\mathscr{V}_\varphi(\pi_2)$ is a successor of the position name $\mathscr{V}_\varphi(\pi_1)$ (denoted as $succ(\mathscr{V}_\varphi(\pi_1), \mathscr{V}_\varphi(\pi_2))$) if and only if $depth(\mathscr{V}_\varphi(\pi_2)) = depth(\mathscr{V}_\varphi(\pi_1)) + 1$, and for the longest prefixes π_1' of π_1 and π_2' of π_2 which ends in some $\langle\!\langle C \rangle\!\rangle \mathbf{X}$ or $[[C]]\,\widetilde{\mathbf{X}}$ we have that π_1' is a proper prefix of π_2'.*

Example 2. *Consider the formula $\varphi_1 = (\langle\!\langle \{x\} \rangle\!\rangle \mathbf{X}\, goal(x, 100) \wedge terminal) \vee ([[\{o\}]]\,\widetilde{\mathbf{X}}\, goal(o, 100) \wedge terminal)$ with the same position naming as in Example 1. Then, both position names 1 and 2 are successor positions of the position name 0 (i.e., $succ(0, 1)$ and $succ(0, 2)$). This implies that the truth value of φ_1 with position name 0 may depend on the evaluation of the subformulas of φ_1 with position names 1 and 2.*

The overall QASP encoding consists of 4 parts: Firstly, an ASP representation of the GDL description G. Secondly, an ASP encoding of the GCL formula φ. Thirdly, an action generator in ASP that can generate a legal action for each player at a non-terminal state. Finally, a quantifier prefix **Q**. The ultimate goal is to ensure that $G \models_t \varphi$ if and only if the QASP is satisfiable.

We first map a GDL description G into an ASP representation called Position-Extended ASP, which is similar to the method discussed in GTL model-checking [5].

Definition 8 (Position Extension). *Suppose G is a valid GDL description, the Position-Extended ASP of G (denoted as $P_{ext}(G)$) is obtained from G as follows:*

- *Change all occurrences of $true(F)$ to $true(F, (T, 0))$ and all occurrences of $next(F)$ to $true(F, (Q, 0))$.*
- *For any atom of the form $p(\vec{t}\,)$ (i.e., predicate p with argument $\vec{t}\,$) such that the predicate symbol $p \notin \{true, next\}$; if p depends on **true** but does not depend on **does**, change it to $p(\vec{t}\,, (T, 0))$; if p depends on **does**, change it to $p(\vec{t}\,, (Q, 1))$.*
- *For each rule whose head has been extended by $(T, 0)$, add $tdom(T)$ to its body; and for each rule whose head has been extended by $(Q, 0)$ or $(Q, 1)$, add $succ(T, Q)$ to its body.*

We extend the definition of S^{true} to $S^{true}(i) = \{true(f, (i, 0)) \mid true(f) \in S^{true}\}$ and the definition of A^{does} to

$$A^{does}(i) = \{does(p_1, A(p_1), (i, 1)), ..., does(p_k, A(p_k), (i, 1))\}$$

Example 3. *Consider the following rule of Tic-Tac-Toe, which says that if player x marks a cell and the cell is currently blank, the cell will be marked as x in the next state.*

- $next(cell(X, Y, x)) \;:-\; true(cell(X, Y, b)), does(x, mark(X, Y))$.

In $P_{ext}(G)$, the rule is mapped to

- $true(cell(X, Y, x), (Q, 0)) \;:-\; true(cell(X, Y, b), (T, 0)), does(x, mark(X, Y), (Q, 1)), succ(T, Q)$.

Note that the program $P_{ext}(G)$ is stratified whenever G is. The following theorem shows the semantics equivalence of $P_{ext}(G)$ and valid game playing sequences in the original game description G.

Theorem 2. *Consider a GDL description G with semantics (R, S_0, T, l, u, g) and a valid play sequence*

$$S_0 \xrightarrow{A_1} S_1 \xrightarrow{A_2} ...S_{n-1} \xrightarrow{A_n} S_n.$$

Let $P = S_0^{true}(i_0) \cup P_{ext}(G) \cup A_1^{does}(i_1) \cup ... \cup A_n^{does}(i_n) \cup \bigcup_{k=0}^{n-1}\{succ(i_k, i_{k+1}).\} \cup \bigcup_{k=0}^{n}\{tdom(i_k).\}$, where i_k $(0 \le k \le n)$ are arbitrary pairwise distinct integers. Then, for any predicate symbol p in the game description G and for all $0 \le k \le n$, such that p is not **init** *or* **next** *and p does not depend on* **does**, *we have:*

- $S_k = \{f \mid P \models true(f, (i_k, 0))\}$, *and*
- $G \cup S_k^{true} \models p(\vec{t}\,)$ *iff* $P \models p(\vec{t}\,, (i_k, 0))$

In particular, since legal does not depend on does, $G \cup S_k^{true} \models legal(r,a)$ *iff* $P \models legal(r,a,(i_k,0))$.

The correctness of the theorem is a direct consequence of Theorem 1 in [5]. The only difference is that in [5], it assumes that $i_k = k$.

Next, we show how a GCL formula can be encoded as an ASP program. Similar to GTL model-checking [5], we assume that there is a function $\eta(\psi, i)$ that would give an atom of arity 0 for each GCL formula ψ with position naming i.

Definition 9. *Suppose that* φ *is a GCL formula in NNF, the ASP encoding of any of its subformula* ψ *with position* π *(denoted as* $Enc(\psi, \mathscr{V}_\varphi(\pi))$*) is recursively defined as follows:*

- $Enc(p(\vec{t}\,), i) = \{\eta(p(\vec{t}\,), i) \text{ :- } p(\vec{t}\,, (i,0)).\}$.
- $Enc(\neg p(\vec{t}\,), i) = \{\eta(\neg p(\vec{t}\,), i) \text{ :- not } p(\vec{t}\,, (i,0)).\}$.
- $Enc(\varphi_1 \wedge \varphi_2, \mathscr{V}_\varphi(\pi)) = Enc(\varphi_1, \mathscr{V}_\varphi(\pi\wedge_1)) \cup Enc(\varphi_2, \mathscr{V}_\varphi(\pi\wedge_2)) \cup \{\eta(\varphi_1 \wedge \varphi_2, \mathscr{V}_\varphi(\pi)) \text{ :- } \eta(\varphi_1, \mathscr{V}_\varphi(\pi\wedge_1)), \eta(\varphi_2, \mathscr{V}_\varphi(\pi\wedge_2))\}$.
- $Enc(\varphi_1 \vee \varphi_2, \mathscr{V}_\varphi(\pi)) = Enc(\varphi_1, \mathscr{V}_\varphi(\pi\vee_1)) \cup Enc(\varphi_2, \mathscr{V}_\varphi(\pi\vee_2)) \cup \{\eta(\varphi_1 \vee \varphi_2, \mathscr{V}_\varphi(\pi)) \text{ :- } \eta(\varphi_1, \mathscr{V}_\varphi(\pi\vee_1)).\} \cup \{\eta(\varphi_1 \vee \varphi_2, \mathscr{V}_\varphi(\pi)) \text{ :- } \eta(\varphi_2, \mathscr{V}_\varphi(\pi\vee_2))\}$.
- $Enc(\langle\langle C\rangle\rangle \mathbf{X}\phi, \mathscr{V}_\varphi(\pi)) = \{\eta(\langle\langle C\rangle\rangle \mathbf{X}\phi, \mathscr{V}_\varphi(\pi)) \text{ :- } \eta(\phi, \mathscr{V}_\varphi(\pi\langle\langle C\rangle\rangle \mathbf{X})), \textit{not terminal}((\mathscr{V}_\varphi(\pi), 0)).\} \cup P_{gen}(G, C, \mathscr{V}_\varphi(\pi), \mathscr{V}_\varphi(\pi\langle\langle C\rangle\rangle \mathbf{X})) \cup Enc(\phi, \mathscr{V}_\varphi(\pi\langle\langle C\rangle\rangle \mathbf{X})) \cup \{succ(\mathscr{V}_\varphi(\pi), \mathscr{V}_\varphi(\pi\langle\langle C\rangle\rangle \mathbf{X}))\} \cup \{tdom(\mathscr{V}_\varphi(\pi\langle\langle C\rangle\rangle \mathbf{X}))\} \cup \{\_exists(r, \mathscr{V}_\varphi(\pi\langle\langle C\rangle\rangle \mathbf{X}))). \mid r \in C\}$
- $Enc([[C]]\,\widetilde{\mathbf{X}}\phi, \mathscr{V}_\varphi(\pi)) = \{\eta([[C]]\,\widetilde{\mathbf{X}}\phi, \mathscr{V}_\varphi(\pi)) \text{ :- } \eta(\phi, \mathscr{V}_\varphi(\pi[[C]]\,\widetilde{\mathbf{X}}))\.\} \cup \{\eta([[C]]\,\widetilde{\mathbf{X}}\phi, \mathscr{V}_\varphi(\pi)) \text{ :- } terminal((\mathscr{V}_\varphi(\pi), 0))\.\} \cup P_{gen}(G, R \setminus C, \mathscr{V}_\varphi(\pi), \mathscr{V}_\varphi(\pi[[C]]\,\widetilde{\mathbf{X}})) \cup Enc(\phi, \mathscr{V}_\varphi(\pi[[C]]\,\widetilde{\mathbf{X}})) \cup \{succ(\mathscr{V}_\varphi(\pi), \mathscr{V}_\varphi(\pi[[C]]\,\widetilde{\mathbf{X}}))\.\} \cup \{tdom(\mathscr{V}_\varphi(\pi[[C]]\,\widetilde{\mathbf{X}}))\.\}$

The intuition of the above inductive encoding is as follows. Suppose that we want to evaluate the subformula ψ with position i at state S, the atom $\eta(\psi, i)$ holds if and only if $G, S \models_t \psi$. The case when ψ is a ground atom $p(\vec{t}\,)$ is converted to a rule which justifies $\eta(p(\vec{t}\,), i)$ if and only if $p(\vec{t}\,)$ with position i holds at state S. To show that ψ where $\psi = \varphi_1 \wedge \varphi_2$ holds (i.e., $\eta(\varphi_1 \wedge \varphi_2, i)$ is justified), the encoding states that the both subformulas φ_1 and φ_2 must be justified at state S. Remark that the positions $\pi\wedge_1$ and $\pi\wedge_2$ have the same name, because they are within the same scope of $\langle\langle C\rangle\rangle \mathbf{X}$ and $[[C]]\,\widetilde{\mathbf{X}}$.

ψ with connectivities $\neg$ or $\vee$ are defined the same way as in the GTL model-checking encoding [5].

The case when the subformula ψ is of the form $\langle\langle C\rangle\rangle \mathbf{X}\phi$ with position naming $\mathscr{V}_\varphi(\pi)$ is converted to rules which justify $\eta(\langle\langle C\rangle\rangle \mathbf{X}\phi, \mathscr{V}_\varphi(\pi))$ if and only if the current state S is not terminal and the players joint actions of the players perform at S would lead us to a successor state S' such that the formula ϕ with position $\mathscr{V}_\varphi(\pi\langle\langle C\rangle\rangle \mathbf{X})$ holds at S'. The case when the subformula ψ is of the form $[[C]]\,\widetilde{\mathbf{X}}\phi$ with position naming $\mathscr{V}_\varphi(\pi)$ is converted to rules which justify $\eta([[C]]\,\widetilde{\mathbf{X}}\phi, \mathscr{V}_\varphi(\pi))$ if and only if the current state S is terminal or the players joint actions of the players perform at S would lead us to a successor state S' such that the formula ϕ with position $\mathscr{V}_\varphi(\pi[[C]]\,\widetilde{\mathbf{X}})$ holds at S'.

In the last two cases, whether a subformula ψ at a state S holds is related to the joint actions of the players and whether ϕ holds in the successor state S' of S. Similar to GTL model-checking [5], we need an action generator program in ASP. Suppose that the current state is S and we want to reason about a subformula with position name t whose truth value is related to the truth value of the subformula at position name q at some successor state S' of S such that $succ(t, q)$ holds. The action generator program describes the state transition from S to S' (cf. Theorem 2), which generates 1 action per player that is legal at state S as long as *terminal* cannot be derived at position name t. Since GCL also involves existential and universal quantification of player actions at a particular state, we define the following action generator, which generates 1 action per player that is legal at state S such that one successor of position name t is position q and the actions of the players in C are quantified existentially (i.e., the subformula ψ is of the form $\langle\langle C\rangle\rangle \mathbf{X}\phi$), while the actions of the players in $R \setminus C$ are quantified universally.

Definition 10 (Action Generator)**.** *Suppose that G is a valid GDL description, $C \subseteq R$ is a coalition, t and q the names of two positions, the action generator $P_{gen}(G, C, t, q)$ contains the following clauses.*

1. $end((t,0))$ *:-* $terminal((t,0))$.
2. $end((q,0))$ *:-* $end((t,0))$.
3. $1\ \{does(R, A, (q,1)) : input(R, A)\}\ 1$ *:-* $not\ end((t,0)),\ role(R)$.
4. *:-* $not\ legal(R, A, (t,0)),\ does(R, A, (q,1))$.
5. *For each $r \in R \setminus C$, create the clause:* $\{moveL(r, L, (q,1)) : ldom(r, L)\}$.
6. *For each $r \in R \setminus C$ and each m_i in the move domain of r, create the clause:*
   $does(r, m_i, (q,1))$ *:-* $legal(r, m_i, (t,0)),\ not\ end((t,0)),\ moveL(r, \rho_1, (q,1)), \ldots,\ moveL(r, \rho_j, (q,1)),$
   $not\ moveL(r, \mu_1, (q,1)), \ldots,\ moveL(r, \mu_k, (q,1))$.

   where $\rho_1, \ldots, \rho_j$ are the 1 bits in the binary representation of $i-1$, and $\mu_1, \ldots, \mu_k$ are the 0 bits in the binary representation of $i-1$.

In the action generator, (1) checks that state S is not a terminal state. (2) ensures that if S is a terminal state, all of its successor states should be terminal. (3) states that one legal action should be generated for all players at state S that would lead to state S' as long as S is not a terminal state. The integrity constraint (4) ensures that all actions generated by (3) should be legal. One challenging part of the encoding is to generate all possible **legal** joint actions of players whose actions are quantified universally (i.e., players in $R \setminus C$) at a particular state. We need to ensure that every player in this set must make exactly one legal action. Note that we cannot simply use integrity constraints like (4) to achieve this goal because we do not want the "universal" players in $R \setminus C$ to "deliberately" pick illegal actions to unsatisfy the QASP. (5) and (6) constitute a logarithmic encoding of the actions of the "universal" players in $R \setminus C$ that uses the idea of the so-called corrective encoding of propositional games [8]. *ldom* is the logarithmic move domain, which represents the domain of the second parameter of moveL. Suppose the size of the move domain of some player $r \in R \setminus C$ is $|M|$, then *ldom* is defined over 1 to $|L| = \lceil \log_2 |M| \rceil$. We create $|M|$ rules of the form (6), one for each action m_i in the move domain. The logarithmic encoding ensures that every player in $R \setminus C$ will make exactly one legal move at state S. It is important to note that $|M|$ might be less than $2^{|L|}$, which means there might be some binary combinations of *moveL* that do not correspond to a legal action in the move domain. In this case, no action of some players in $R \setminus C$ can be generated by rule (6). However, (3) and (4) ensure that exactly one legal action of these players in $R \setminus C$ is generated in such a case.

Example 4. *Consider the formula $\varphi_1 = (\langle\!\langle \{x\} \rangle\!\rangle\, \mathbf{X}\, goal(x, 100) \wedge terminal) \vee ([[\{o\}]]\, \widetilde{\mathbf{X}}\, goal(o, 100) \wedge terminal)$ with the same position naming as in Example 1. Then, one possible definition of the mapping η of the subformulas in φ_1 is:*

- $\eta(\varphi_1, 0) = a_0$
- $\eta(goal(x, 100) \wedge terminal^1, 1) = a_2$
- $\eta(terminal^1, 1) = a_4$
- $\eta(goal(o, 100) \wedge terminal^2, 2) = a_6$
- $\eta(terminal^2, 2) = a_8$
- $\eta(\langle\!\langle \{x\} \rangle\!\rangle\, \mathbf{X}\, goal(x, 100) \wedge terminal^1, 0) = a_1$
- $\eta(goal(x, 100), 1) = a_3$
- $\eta([[\{o\}]]\, \widetilde{\mathbf{X}}\, goal(o, 100) \wedge terminal^2, 0) = a_5$
- $\eta(goal(o, 100), 2) = a_7$

And the ASP encoding of $Enc(\varphi_1, 0)$ the GCL formula φ_1 contains the following clauses:

- a_0 *:-* a_1.
- a_1 *:-* $a_2,\ not\ terminal((0,0))$.
- a_0 *:-* a_5.
- a_2 *:-* $a_3,\ a_4$.

- a_3 :- $goal(x, 100, (1, 0))$. a_4 :- $terminal((1, 0))$.
- a_5 :- a_6. a_5 :- $terminal((0, 0))$. a_6 :- a_7, a_8.
- a_7 :- $goal(o, 100, (2, 0))$. a_8 :- $terminal((2, 0))$.
- *The action generation programs:* $P_{gen}(G, \{x\}, 0, 1)$ *and* $P_{gen}(G, \{x\}, 0, 2)$
- *Facts:* $succ(0, 1)$. $succ(0, 2)$. $tdom(1)$. $tdom(2)$. $\_exists(x, 1)$

To ensure that the GCL formula φ holds, $\eta(\varphi, 0)$ must be justified. Hence, we add the integrity constraint $\{$:- *not* $\eta(\varphi, 0)\}$ to our encoding. Note that all encodings we have discussed so far do not involve any actual quantification. To verify a GCL formula, we need to define a quantifier prefix $\mathbf{Q}$ to the program $Enc(\varphi, 0) \cup \{tdom(0).\} \cup P_{ext}(G) \cup \{$:- *not* $\eta(\varphi, 0)\}$ so that the resulting QASP is satisfiable if and only if $G \models_t \varphi$. The idea is to quantify all ground atoms of the predicate *moveL* in the ground program universally, and all the other ground atoms existentially. One possible quantification method based on atom dependency is defined as follows, which is similar to the method we discussed in [1].

Definition 11 (Atom dependency)**.** *Suppose that Q is an ASP program with grounding $\mathbf{A}$ and P is the ground normal logic program of Q. For two atoms $p, q \in \mathbf{A}$ we say that q depends on p in Q iff the following recursive definition holds: Program P contains a rule such that the atom q appears in the head of the rule and the atom p appears in the body of the rule; or, there exists an atom $z \in \mathbf{A}$ such that q depends on z and z depends on p. We denote that q depends on p by $p \rightarrow q$.*

Definition 12 (Dependency Based Quantification Method)**.** *Suppose $\mathbf{A}$ is the set of ground atoms of the program $P = S_0^{true}(0) \cup Enc(\varphi, 0) \cup \{tdom(0).\} \cup P_{ext}(G) \cup \{$:- *not* $\eta(\varphi, 0)\}$ and the quantifier prefix $\mathbf{Q}_d$ of the program is of form where $T_{max} = deg(\varphi)$*

$$\mathbf{Q}_d = \exists E_0\ \forall U_1\ \exists E_1\ \forall U_2 \ldots\ \forall U_{T_{max}}\ \exists E_{T_{max}}$$

Then, for each $a \in \mathbf{A}$, the quantifier block it belongs to is determined by the following three rules:

1. *If $a = moveL(r, L, (t, 1))$ for some r, L, t then $a \in U_{depth(t)}$.*
2. *If $a = does(r, M, (t, 1))$ for some r, M, t and $\_exists(r, t) \in \mathbf{A}$ then $a \in E_{depth(t)-1}$*
3. *If $a = does(r, M, (t, 1))$ for some r, M, t and $\_exists(r, t) \notin \mathbf{A}$ then $a \in E_{depth(t)+1}$*
4. *Otherwise, $a \in E_t$ with t the maximum $1 \leq t \leq T_{max}$ such that $moveL(r, L, (p, 1)) \rightarrow a$ for some $moveL(r, L, (p, 1)) \in \mathbf{A}$ and $depth(p) = t$. If no such t exists then $a \in E_0$.*

To put it in words, we quantify the ground atoms of P that is an instance of the predicate *moveL* universally. Item 1 ensures that the quantification level of the universal variables follows the depth of the position names. There are two possible situations of the quantifier level of a ground atom that is an instance of *does* and is extended with position name t (e.g., $does(r, M, (t, 1))$). We should note that the action generator generates any instance of *does* in the program P. To generate the ground atom $does(r, M, (t, 1))$ in P, the program $P_{gen}(G, C, q, t)$ must be part of $Enc(\varphi, 0)$ for some $succ(q, t)$ and C. According to definition 9, $P_{gen}(G, C, q, t)$ is part of $Enc(\varphi, 0)$ if and only if either $Enc(\langle\!\langle C \rangle\!\rangle \widetilde{\mathbf{X}}\phi, q)$ or $Enc([[R \setminus C]]\, \mathbf{X}\phi, q)$ is part of $Enc(\varphi, 0)$. For the former case, we want to verify whether $\langle\!\langle C \rangle\!\rangle \widetilde{\mathbf{X}}\phi$ holds. The actions of players in C should be generated before the actions in $R \setminus C$. Therefore, in item 2, if $r \in C$, $does(r, M, (t, 1))$ should be quantified existentially in $E_{depth(t)-1}$. In all other cases, according to item 3, the ground atom $does(r, M, (t, 1))$ should be quantified existentially in $E_{depth(t)+1}$. Item 4 ensures that no ground atom in P that depends on an action is quantified before that action atom.

We now revisit proving $\langle\!\langle C \rangle\!\rangle \mathbf{X}\phi$ at state S. In the ASP encoding of GCL formulas (cf. definition 9), if we want to justify $\eta(\langle\!\langle C \rangle\!\rangle \mathbf{X}\phi, \mathcal{V}_\varphi(\pi))$ at state S, S must not be terminal and $\eta(\phi, \mathcal{V}_\varphi(\pi\langle\!\langle C \rangle\!\rangle \mathbf{X}))$ must be justified at the successor state. With our quantification method, we can ensure that $\eta(\langle\!\langle C \rangle\!\rangle \mathbf{X}\phi, \mathcal{V}_\varphi(\pi))$ is justified if and only if the S is not terminal and there exists some legal joint actions of the players

in C such that for all legal joint actions of players in $R \setminus C$, $\eta(\phi, \mathscr{V}_\varphi(\pi\langle\!\langle C \rangle\!\rangle \mathbf{X}))$ must be justified in the successor state and thereby shows that $G, S \models_t \langle\!\langle C \rangle\!\rangle \mathbf{X}\phi$. Generating all legal joint actions of players in $R \setminus C$ is achieved by the logarithmic encoding of the actions of the players in $R \setminus C$.

Example 5. *Consider the formula* $\varphi_1 = (\langle\!\langle \{x\} \rangle\!\rangle \mathbf{X}\, goal(x, 100) \wedge terminal) \vee ([[\{o\}]]\, \widetilde{\mathbf{X}}\, goal(o, 100) \wedge terminal)$ *with the same position naming as in Example 1. Let* $P = S_0^{true}(0) \cup Enc(\varphi, 0) \cup \{tdom(0).\} \cup P_{ext}(G) \cup \{:\text{- not } \eta(\varphi, 0)\}$ *and the quantifier prefix* $\mathbf{Q}_d$ *and the ground atoms of* P *is* $\mathbf{A}$*. Assume the move domain of players is* $\{(x, ac_1), (x, ac_2), (x, ac_3), (o, ac_1), (o, ac_2), (o, ac_3)\}$*. Then, the quantifier prefix of* P *has the form* $\exists E_0 \forall U_1 \exists E_1$*, where:*

- $E_0 = \{terminal((0,0)), tdom(0..2), succ(0,1), succ(0,2), does(x, A, (1,1)), \_exists(x,1) \ldots\}$
- $U_1 = \{moveL(o, 1, (1,1)), moveL(o, 2, (1,1)), moveL(x, 1, (2,1)), moveL(x, 2, (2,1))\}$
- $E_1 = \{does(o, A, (1,1)), does(x, A, (2,1)), does(o, A, (2,1)), a_0, \ldots, a_8, \ldots\}$*, where* $A \in \{ac_1, ac_2, ac_3\}$

When proving the subformula $\langle\!\langle \{x\} \rangle\!\rangle \mathbf{X}\, goal(x, 100) \wedge terminal$*, the action of player* x *must be made before player* o*. Hence, the actions of player* x *at position 1 are quantified in* E_0*, and all the actions of player* o *at position 1 are quantified later in* E_1*.* $moveL(o, 1, (1,1))$ *and* $moveL(o, 2, (1,1))$ *are quantified universally before the actions of player* o *ensuring that all 3 possible actions of player* o *at position 1 may be generated by some binary combination of* $moveL(o, 1..2, (1,1))$*. And if the binary combination of* $moveL(o, 1..2, (1,1))$ *does not correspond to a legal action at position 1, then player* o *can pick any legal action at position 1. The logarithmic encoding and quantification method ensures that the constraint "there exists a legal action of player* x *such that for all legal actions of player* o*, something holds next" can be implemented correctly.*

In QBF research, "quantification method" is also referred to as "prenexing strategy". For GCL model-checking, there are many correct prenexing strategies. For example, due to both $Enc(\varphi, 0)$ and $P_{ext}(G)$ are stratified, the quantifier prefix of P can also be defined as follows, where $E_0' = \{does(x, A, (1,1))\}$, $U_1' = U_1$ and E_1' contains all other ground atoms. Such a quantification method moves all atoms that were originally quantified in the first existential block E_0 and are not an instance of *does* to the existential block E_1. The quantification method is also correct because the truth value of all atoms that have been moved (e.g., $tdom(0).$) does not depend on *does* and can be uniquely determined at state S_0.

Different prenexing strategies can significantly affect the efficiency of GCL model-checking [1]. Systematically exploring how the efficiency of GCL model-checking can be affected by different prenexing strategies is beyond the scope of this paper. Interested readers can find out more about how prenexing strategies can affect the efficiency of QBF solving in [19].

To summarize, our encoding is similar to GTL model checking [5]; the only difference lies in the use of a logarithmic encoding to handle the existential and universal quantification over players' actions. The following proposition states the correctness of the encoding, which can be proved by induction on the structure of the GCL formula (cf. our explanation in Definition 9) and relies on Theorem 2 [1].

Proposition 1. *Suppose that* G *is a valid GDL description and* φ *is a GCL formula over* G*, then* $G \models_t \varphi$ *if and only if the following QASP is satisfiable.*

$$\mathbf{Q}_d \;\; S_0^{true}(0) \cup Enc(\varphi, 0) \cup \{tdom(0).\} \cup P_{ext}(G) \cup \{:\text{- not } \eta(\varphi, 0)\}$$

5. Experimental Results

One of the most important strategic properties of games is strong winnability, which is whether a player has a strategy to win the game regardless of the actions of the remaining players. In this section, we briefly report the efficiency of our QBF-based model-checking method on reasoning strong winnability on some popular turn-based two-player games: Generalized 4×4 Tic-Tac-Toe (GT-1-1, and GT-2-2) [9], Connect-3 (C-3), Connect-4 (C-4) [20], Breakthrough (BT) [21], and Dots and Boxes (D&B) [22]. For

[1] Detailed proof is available here: https://github.com/hharryyf/gdl2qbf-general

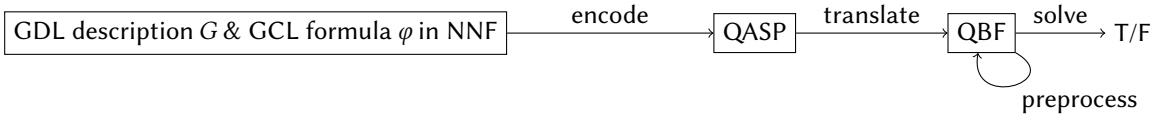

Figure 1: The logic flow of GCL Model-Checking with QBF solvers

simplicity, we assume the two players in these games are denoted as x and o, with x as the player taking the first turn. We only concern ourselves with whether the game is strongly winnable by player x. We test the efficiency of our model-checking method on various game configurations. For C-3, C-4, BT, and D&B, different configurations refer to different board sizes, whereas in GT-1-1 and GT-2-2, different configurations indicate different winning domino shapes [9]. The GDL rules of these games are created and modified based on the GGP Base repository [23].

Well-formed GDL games must be finite and terminating [2]. We employ iterative deepening to prove the strong winnability of any game G [8] by gradually increasing the search depth, T_{max}, and checking if $G \models_t \varphi_{win}(x, T_{max})$. For a game G, we denote by μ_G the length of the longest valid playing sequence with player x as the winner. Iterative deepening is complete and terminates as long as we are provided with μ_G: If $G \not\models_t \varphi_{win}(x, T_{max})$ for all $T_{max} \leq \mu_G$, then x cannot strongly win the game at all. Computing μ_G is beyond the scope of this paper, and so we used human domain knowledge [21] to determine it.

Our overall approach is given in Figure 1.[2] We use *qasp2qbf* [13] for the QASP to QBF translation. For the QBF preprocessing, we use *bloqqer* [24]. Finally, we solve the preprocessed formulas with one of the state-of-the-art QBF solvers *Caqe* [7]. Our approach is certainly not the only method of reasoning about the bounded-depth strong winnability of strategic games. Forward search algorithms are widely applied in solving such a problem. For comparison purposes, we implemented a naive Minimax Search-based solver (Minx) in C++ that uses Prolog as the GDL reasoner for legal actions [25] and an enhanced version with transposition tables (MinxTT). All the experiments were run on a Latitude 5430 laptop with a solving time limit of 900 seconds.

In Table 1, we record the value of μ_G and the smallest depth T_{max} of the game in **bold** if the game can be proved to be strongly winnable by player x within T_{max} steps by any of the 3 solvers within the solving time limit. If player x cannot force a win at all depths, and the game can be proved not to be strongly winnable by player x within μ_G steps by any solver under any encoding in the solving time limit, we let $T_{max} = \mu_G$ and record T_{max} in plain format. For games that can neither be proved to be strongly winnable by x at some depth nor be proved not to strongly winnable by x for all depth within the solving time limit, we record the maximum refuted depth T_{max} [21] (i.e., the maximum T_{max} such that the game is proved not to be strongly winnable by x within T_{max} steps by any solver under any encoding within the solving time limit) of the unsolved games in *italic* format.

In Table 1, we also record the preprocessing time of *bloqqer* (Bloq), the total run time of *bloqqer* and *Caqe* (Caqe+bloq) to verify $\varphi_{win}(x, T_{max})$, the run time of the baseline minimax solver (Minx), and the run time of the minimax with transposition table solver (MinxTT).

We can observe that for almost all of these small-sized games, *Caqe* can completely solve the game (i.e., prove or disprove whether x can strongly win the game) in a reasonable amount of time. For the relatively larger instances C-4-5x5 and C-4-6x6, although *Caqe* cannot solve the game completely, it can disprove the bounded-depth strong winnability property of these games to a relatively large depth. If we compare the QBF-based approach (*Caqe+bloq*) with the baseline minimax solver (*Minx*), we can observe that the efficiency of the QBF-based approach outperforms *Minx* in almost all instances except the easy ones, such as C-3, where the *bloqqer* preprocessing time dominates the solving procedure. The comparison between the QBF-based approach with the minimax and transposition table solver *MinxTT*, however, is mixed, and the result is highly domain dependent. Except for the easy domain such as C-3, the QBF-based approach outperforms *MinxTT* in most of the non-trivial games from C-4, GT-1-1, and GT-2-2. However, *MinxTT* outperforms *Caqe+bloq* by more than an order of magnitude in D&B.

[2]The interested reader can find the code used in our experiments here: https://github.com/hharryyf/gdl2qbf-general

Game	Config	μ_G	T_{max}	Bloq	Caqe+bloq	Minx	MinxTT
	2×5	21	21	6.61	6.88	1.64	**0.36**
	2×6	29	**15**	5.71	15.63	25.58	**3.13**
BT	3×4	19	19	11.91	12.12	5.49	**1.13**
	4×4	25	25	51.57	**65.97**	*	106.20
	4×4	15	**9**	0.64	0.72	0.33	**0.20**
C-3	5×5	25	**9**	0.98	1.22	1.28	**0.74**
	6×6	35	**9**	1.30	2.10	2.65	**1.54**
	4×4	15	15	1.06	**1.37**	9.06	1.54
C-4	5×5	25	*21*	2.55	**219.35**	*	628.17
	6×6	35	*19*	4.19	**649.13**	*	*
	elly	15	**7**	2.65	**3.01**	21.48	11.74
	fat.	15	15	7.40	**259.80**	*	307.38
GT-1-1	knob.	15	15	7.29	**587.17**	*	*
	skin.	15	15	7.15	*	*	**263.20**
	tip.	15	**9**	3.43	**4.79**	172.78	30.94
	elly	14	**6**	1.83	1.90	0.39	**0.23**
	fat.	14	14	5.63	**316.63**	*	*
GT-2-2	knob.	14	**6**	1.77	1.85	1.24	**0.88**
	skin.	14	14	5.44	**560.45**	*	662.32
	tip.	14	**6**	1.70	**1.75**	6.14	3.87
D&B	2×2	12	**12**	1.89	7.21	22.61	**0.64**
	2×3	17	17	5.89	592.51	*	**18.05**

Table 1
Time (in seconds) for the QBF-based approach (Caqe+bloq), the minimax baseline solver (Minx), and the minimax + transposition table solver to $\varphi_{win}(x, T_{max})$. * means the solver timed out after 900 seconds.

Note that GCL model-checking can also be performed by a naive depth-first search algorithm by directly following Definition 4, and for verifying the bounded-depth strong winnability, the algorithm is similar to the baseline minimax search, which performs drastically worse than our QBF-based approach. Although with a transposition table, the efficiency of the minimax search solver for verifying the bounded-depth strong winnability can be improved significantly, for general GCL model-checking, it is still unclear how a transposition table can be properly defined. Our QBF-based approach is more generic because the learning mechanism within the QBF solver (e.g., Counterexample Guided Abstract Refinement [7]) applies to any QBF expression. Although the transposition table is not the same as the learning mechanism in QBF solvers, both can prune unnecessary search space. With a generic QBF solver, we can leverage its pruning effect without the need to design a transposition table from scratch.

Even if bounded-depth strong winnability is just one important property that can be expressed in GCL, the experiment indicates that our QBF-based approach has the potential to support GGP systems to perform endgame search when the game property can be formulated as GCL formulas.

6. Conclusion and Future Work

In this paper, we investigate the connection between Coalition Logic and GDL and propose a QBF-based model-checking method. Although Coalition Logic is only a subset of ATL [4], it allows the formulation of both strategic and non-strategic properties up to a bounded depth. Our case study on solving two-player games shows that the QBF-based model-checking method has the potential to assist GGP systems in reasoning about endgame properties that can be formulated as GCL formulas. In the future, we plan to systematically evaluate our QBF-based approach on verifying other GCL properties and investigate how to integrate the QBF-based method into a GGP system. Also, we intend to investigate how to extend our approach to reason about GCL formulas of games with imperfect information [26].

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT in order to: Grammar and spelling check.

References

[1] Y. He, A. Saffidine, M. Thielscher, Solving Two-player games with QBF solvers in General Game Playing, in: Proceedings of AAMAS, 2024, pp. 807–815.

[2] M. Genesereth, M. Thielscher, General game playing, Synthesis Lectures on Artificial Intelligence and Machine Learning 8 (2014) 1–229.

[3] J. Clune, Heuristic evaluation functions for general game playing, in: Proceedings of AAAI, 2007, pp. 1134–1139.

[4] J. Ruan, W. Van Der Hoek, M. Wooldridge, Verification of games in the game description language, Journal of Logic and Computation 19 (2009) 1127–1156.

[5] M. Thielscher, S. Voigt, A Temporal Proof System for General Game Playing, in: Proceedings of AAAI, 2010, pp. 1000–1005.

[6] M. Pauly, A modal logic for coalitional power in games, Journal of logic and computation 12 (2002) 149–166.

[7] M. N. Rabe, L. Tentrup, CAQE: A Certifying QBF Solver, in: Proceedings of FMCAD, 2015, pp. 136–143.

[8] V. Mayer-Eichberger, A. Saffidine, Positional Games and QBF: The Corrective Encoding, in: Proceedings of SAT, 2020, pp. 447–463.

[9] Diptarama, R. Yoshinaka, A. Shinohara, QBF Encoding of Generalized Tic-Tac-Toe, in: F. Lonsing, M. Seidl (Eds.), Proceedings of the 4th International Workshop on Quantified Boolean Formulas (QBF 2016), 2016, pp. 14–26.

[10] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Answer set solving in practice, Synthesis lectures on artificial intelligence and machine learning 6 (2012) 1–238.

[11] S. Schiffel, M. Thielscher, A Multiagent Semantics for the Game Description Language, in: Agents and Artificial Intelligence, 2010, pp. 44–55.

[12] J. Marques-Silva, I. Lynce, S. Malik, Conflict-driven clause learning SAT solvers, in: Handbook of Satisfiability, IOS press, 2021, pp. 133–182.

[13] J. Fandinno, F. Laferrière, J. Romero, T. Schaub, T. C. Son, Planning with incomplete information in quantified answer set programming, Theory and Practice of Logic Programming 21 (2021) 663–679.

[14] F. Belardinelli, A. Lomuscio, A. Murano, S. Rubin, et al., Alternating-time Temporal Logic on Finite Traces., in: Proceedings of IJCAI, volume 18, 2018, pp. 77–83.

[15] E. Bonnet, A. Saffidine, On the complexity of general game playing, in: Workshop on Computer Games, Springer, 2014, pp. 90–104.

[16] T. Janhunen, I. Niemelä, Compact Translations of Non-disjunctive Answer Set Programs to Propositional Clauses, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 111–130. URL: https://doi.org/10.1007/978-3-642-20832-4_8. doi:10.1007/978-3-642-20832-4_8.

[17] A. J. Robinson, A. Voronkov, Handbook of automated reasoning, volume 1, Elsevier, 2001.

[18] S. Haufe, Automated Theorem Proving for General Game Playing, Ph.D. thesis, Dresden University of Technology, 2012. URL: https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa-89998.

[19] S. Heisinger, M. Heisinger, A. Rebola-Pardo, M. Seidl, Quantifier shifting for quantified boolean formulas revisited, in: Proceedings of IJCAR, 2024, pp. 325–343.

[20] I. P. Gent, A. G. Rowley, Encoding Connect-4 using quantified Boolean formulae, in: Proceedings of International Workshop on Modelling and Reformulating Constraint Satisfaction Problems, 2003, pp. 78–93.

[21] I. Shaik, J. van de Pol, Concise QBF Encodings for Games on a Grid (extended version), arXiv preprint 2303.16949, 2023. URL: http://arxiv.org/abs/2303.16949.

[22] E. R. Berlekamp, The Dots and Boxes game: sophisticated child's play, CRC Press, 2000.

[23] GGP, GGP Base Repository. http://games.ggp.org/base/, 2023.

[24] A. Biere, F. Lonsing, M. Seidl, Blocked Clause Elimination for QBF, in: Proceedings of CADE, 2011, pp. 101–115.

[25] S. Schiffel, Y. Björnsson, Efficiency of GDL reasoners, IEEE Transactions on Computational

Intelligence and AI in Games 6 (2014) 343–354.
[26] M. Thielscher, A general game description language for incomplete information games, in: Proceedings of AAAI, 2010, pp. 994–999.