EPiC Series in Computing

Volume 41, 2016, Pages 227–239

GCAI 2016. 2nd Global Conference on Artificial Intelligence

Invariant Projections in Games

Abhijeet Mohapatra, Bertrand Decoster, Sudhir Agarwal, and Michael Genesereth

Stanford University, California, USA
{abhijeet, decoster, agarwal3, genesereth}@stanford.edu

Abstract

Identification of implicit structures in dynamic systems is a fundamental problem in Artificial Intelligence. In this paper, we focus on General Game Playing where games are modeled as finite state machines. We define a new property of game states called *invariant projections* which strongly corresponds to humans' intuition of game boards and may be applied in General Game Playing to support powerful heuristics, and to automate the design of game visualizations. We prove that the computation of invariant projections is Π_2^P-complete in the size of the game description. We also show that invariant projections form a lattice, and the lattice ordering may be used to reduce the time to compute invariant projections potentially by a factor that is exponential in the schema size of game states. To enable competitive general game players to efficiently identify boards, we propose a sound (but incomplete) heuristic for computing invariant projections and evaluate its performance.

1 Introduction

One of the classical problems in Artificial Intelligence is to discover hidden or implicit structures in dynamic systems. In this paper, we focus on the domain of General Game Playing (GGP) [6] since it provides a general theoretical framework for modeling discrete dynamic systems.

In GGP, games are modeled as finite state machines, with one distinguished initial state and one or more terminal states. The games in GGP are defined in a formal language called the Game Description Language (GDL) [8]. Each game has a finite number of players; each player has finitely many possible actions in a game state, and each state has an associated reward for each player.

In GDL, game states are conceptualized as databases, and the notions of legality, reward, termination, and state transitions are defined using logical rules. GDL allows games to be more compactly represented than explicitly representing game states and actions. The compact GDL representation of a game is enabled by representing a large number of game states and actions using fundamental structures such as game boards (e.g. 8x8 grid in Chess), pieces (e.g. pieces in Checkers), and patterns (e.g. lines and diagonals in Tic-Tac-Toe). However, the characterization of these structures is *implicit* in the game's GDL description.

Prior works in GGP [11, 14] have studied the problem of identifying game boards and pieces. However, the proposed solutions rely on specific signatures of game states, and are not applicable in a general setting. The applicability of these solutions is further limited by the

assumption of a one-to-one relationship between positions on a game board and pieces, which does not hold for games like Parchessi[1].

In this paper, we present a novel approach towards identifying game structures. We introduce a new property of game states, called *invariant projections*. Intuitively, invariant projections are components of game states that are identical across all game states. We formally define invariant projections in Section 2.

In Section 3, we prove that computation of invariant projections of a game is Π_2^P-complete in the size of the game description. We show that invariant projections form a lattice, and propose an algorithm called *Prune* that uses the lattice ordering to reduce the time to compute invariant projections of game, often by a factor that is exponential in the game's signature i.e. schema size of the game.

Typically, general game players have limited time to reason about supplied games and to compute their moves. To enable competitive general game players to feasibly compute invariant projections, we propose a heuristic called *VerifyIP* that verifies whether or not a supplied projection is invariant. *VerifyIP* is *sound* but *not complete*.

In Section 4, we evaluate the performance of *VerifyIP* and the effectiveness of the *Prune* algorithm in reducing the search space of invariant projections. We empirically validate our claim that invariant projections serve as an effective characterization of game boards, and discuss potential applications of this characterization.

We discuss related work on identifying game structures in Section 5 and present future directions of our work in Section 6.

2 Invariant Projections

In GGP, games are described using the Game Description Language (GDL). In the following, we present an overview of GDL[2]. Then, we formally define invariant projections. We refer the reader to [7, 8] for details regarding GDL.

Game Description Language. In GDL, game states are modeled as databases, and state transitions as database updates. The GDL model of a game starts with *objects* that are assumed to exist in the game and *relations* that describe the properties of objects or the relationships amongst objects.

Objects and relations are referred to using strings containing letters, digits, and a few non-alphanumeric characters (e.g. '_'). Examples of object and relation constants include `x`, `king`, and `cell`. *Variables* have the same spelling as object and relation constants. However, variables are distinguished from constants by prefixing variables with the character '`?`' e.g. `?x`.

The set of all objects and relations constitute a game's *schema*. For a supplied schema, a *proposition* is defined to be a structure consisting of an n-ary relation and n objects. The *propositional base* for a game consists of all propositions that can be formed from the relations and the objects in the schema.

In GDL, propositions are either *base propositions* which represent conditions that are true in the state of a game, or *effectory propositions* which represent actions performed by game players. A *state* of a game is an arbitrary subset of the game's base propositions.

A game starts in the initial state. On each time step, each player has one or more legal actions it can perform. As an action is performed, some base propositions become true and

[1] http://www.hasbro.com/common/instruct/Parchessi.pdf

[2] The version of GDL presented here is the prefix version that is used in the annual GGP competitions.

others become false, leading to a new set of true propositions and, consequently, a new state and possibly a new set of legal actions. This process repeats until the game enters a terminal state, at which point the game stops and the players are awarded the number of points associated with the terminal state.

(`role` a) : a is a role
(`base` p) : p is a base proposition
(`input` r a) : a is a feasible action for role r
(`init` p) : proposition p is true in the initial state
(`true` p) : proposition p is true in the current state
(`does` r a) : role r performs action a in the current state
(`next` p) : proposition p is true in the next state
(`legal` r a) : action a is legal for role r in the current state
(`goal` r n) : current state has utility n for player r
`terminal` : current state is a terminal state

Figure 1: Game independent constants in GDL

GDL consists of certain *reserved keywords* to denote game-independent relations. These keywords are listed in Figure 1.

We present a partial GDL encoding of Tic-Tac-Toe in Figure 2. We note that negation is handled in GDL rules using negation-as-failure (NAF) semantics.

In Figure 2, the object constants `xplayer` and `oplayer` denote the roles in Tic-Tac-Toe. Two relations are used to model the game states: (a) ternary relation `cell` that relates a row number and a column number in the Tic-Tac-Toe grid with the marker on the grid position, and (b) unary relation `control` to say whose turn it is to mark a cell. The base propositions are characterized by enumerating all possible combinations of grid positions i.e. row and column numbers, and grid contents i.e. marks `x` and `o`, and `b` (which denotes a blank space).

Initially, all grid positions are blank, and it is `xplayer`'s turn to move. The legality rule in Figure 2 says that a player can mark a cell if the cell is blank and if it is the player's turn to mark the Tic-Tac-Toe grid. The update rule in Figure 2 encodes the following behavior: if the `xplayer` player marks a blank cell, then this cell is marked with an `x` in the next state. The `xplayer` player gets 100 points if there is a line of `x` marks and no lines of `o` marks. The game terminates whenever either player has a line of marks, or if the board is not `open`, i.e. there are no cells containing blanks.

Identification of Game Structures. Certain game structures are explicitly stated in a game's GDL description and can, therefore, be easily extracted. For example, identifying a game's players is extremely straightforward, since the players are enumerated using the relation `role`. In addition to the game-independent relations in GDL, other relations and constants may be used to model the game states and to characterize game structures such as game boards and pieces (or markers). Such structures, which are modeled by game specific relations and constants, are *implicit* in the GDL description. For example, in the GDL encoding of Tic-Tac-Toe, the relation `cell` is used to characterize the relationship between the game board and the markers placed by the players. However, the relation constant `cell` *does not* have a special meaning within the game. The constant `cell` could be replaced consistently throughout the game by some other constant without changing the semantics of the game.

```
(role xplayer)
(role oplayer)
(<= (base (cell ?m ?n x)) (index ?m) (index ?n))
...
(base (control xplayer))
(base (control oplayer))

(<= (input ?r (mark ?m ?n)) (role ?r) (index ?m) (index ?n))
...
(index 1)
(index 2)
(index 3)
(init (cell 1 1 b))
...
(init (cell 3 3 b))
(init (control xplayer))
(<= (legal ?w (mark ?x ?y)) (true (cell ?x ?y b)) (true (control ?w)))
...
(<= (next (cell ?m ?n x)) (does xplayer (mark ?m ?n)) (true (cell ?m ?n b)))
...
(<= open (true (cell ?m ?n b)))
(<= (goal xplayer 100) (line x) (not (line o)))
...
(<= terminal (line x))
(<= terminal (line o))
(<= terminal (not open))
```

Figure 2: Partial GDL Description of Tic-Tac-Toe

In the remainder of this section, we define a new property of game states called *invariant projections*. As we will show (in Section 4), invariant projections strongly correspond to game boards and may be applied to construct effective game playing heuristics.

Projections. In GGP, a game state is any subset of the base propositions. Formally, a game state $\subseteq \{p \mid (\texttt{base}\, p)\}$.

Definition 1. The structure $\langle r, S\rangle$ is a *projection* of a game if r is a relation, there exists a base proposition that consists of r, and S is a non-empty subset of $\{1, 2, \ldots, k\}$ where k denotes the arity of r.

Consider the game of Tic-Tac-Toe whose GDL description is presented in Figure 2. Base propositions in the game description contain the ternary relation `cell` and the unary relation `control`. Therefore, $\langle \texttt{cell}, \{1\}\rangle$, $\langle \texttt{cell}, \{1, 3\}\rangle$ and $\langle \texttt{control}, \{1\}\rangle$ are examples of projections.

Suppose that $\langle r, S\rangle$ is a projection, where r is a k-ary relation and $S = \{A_1, A_2, \ldots, A_m\}$. *Evaluating* $\langle r, S\rangle$ on any game state produces a set of propositions that is identical to the answers obtained by evaluating the following query q in the game state.

$$(\Leftarrow (q\, ?X_{A_1}\, ?X_{A_2}\, \ldots ?X_{A_m})\, (r\, ?X_1\, ?X_2\, \ldots ?X_k))$$

Example 1. Consider the projection $\langle \texttt{cell}, \{1, 3\}\rangle$ for Tic-Tac-Toe. Evaluating $\langle \texttt{cell}, \{1, 3\}\rangle$ on the game's initial state results in the following propositions.

```
(q 1 b)
(q 2 b)
(q 3 b)
```

The above propositions can be characterized using the following query.

```
(<= (q ?X ?Z) (cell ?X ?Y ?Z))
```

We denote the result of evaluating a projection t on a state s as $\pi(t, s)$, and on the set of all base propositions as $\pi_b(t)$.

Definition 2. An *invariant projection* of a game is a projection t such that all of the following conditions hold.

- **Property 1.** If s_i denotes the initial state of the game, then $\pi(t, s_i) = \pi_b(t)$.
- **Property 2.** For every state s that is reachable from s_i, if s' is a next state that can be reached by executing a legal action on s, then $\pi(t, s) = \pi(t, s')$.

In the following example we present an invariant projection of Tic-Tac-Toe.

Example 2. Consider the projection $t_1 = \langle \texttt{cell}, \{1\} \rangle$ in the game of Tic-Tac-Toe. For this projection, $\pi_b(t_1) = \{\texttt{(q 1)}, \texttt{(q 2)}, \texttt{(q 3)}\}$. It is fairly straightforward to prove either by enumerating all states of Tic-Tac-Toe that are reachable from the initial state, or by using induction on the game rules that $\pi(t_1, s) = \{\texttt{(q 1)}, \texttt{(q 2)}, \texttt{(q 3)}\}$ for every game state s in Tic-Tac-Toe. Therefore, t_1 is an invariant projection.

An invariant projection $\langle \mathtt{r}, S \rangle$ is *maximal* if there does not exist a set T such that $S \subset T$ and $\langle \mathtt{r}, T \rangle$ is invariant.

3 Computing Invariant Projections

In this section, we discuss the computation of invariant projections. First, we show that the problem of deciding whether or not a projection is invariant is computationally hard.

Theorem 1. *The problem of deciding invariant projections is Π_2^P-complete in the size of the game description.*

Proof. Our proof proceeds in two parts. In the first part, we show that the problem of deciding whether a supplied projection is invariant or not is $\in \Pi_2^P$. In the second part, we prove that our decision problem is Π_2^P-hard.

Part 1. For the first part, it suffices to show that with access to an oracle that decides NP-complete problems, counter-examples for our decision problem can be verified in polynomial time.

Suppose we want to decide whether or not a supplied projection t is invariant. Let the initial state of the game be s_i. If $\pi_b(t) \neq \pi(t, s)$, then t is not invariant. This condition can be verified in time that is polynomial in the size of the game description because (i) $\pi_b(t)$ and $\pi(t, s_i)$ can be expressed as GDL queries (see Section 2), and (ii) query evaluation in GDL $\in$ PTIME. In the following, we assume that $\pi_b(t) = \pi(t, s)$.

To decide whether or not t is invariant, we guess a state s. We use an oracle[3] to decide whether or not s is reachable from s_i. From Definition 2, it follows that s is a counter-example

[3]The problem of deciding reachability in constraint graphs in NP-complete [10].

for our decision problem if s is reachable from s_i, and $\pi(t,s) \neq \pi_b(t)$. Since $\pi(t,s) \neq \pi_b(t)$ can be verified in time that is polynomial in the game description, the problem of deciding whether or not a supplied projection is invariant is in Π_2^P.

Part 2. To establish Π_2^P-hardness, we present a polynomial time reduction from the problem of deciding whether or not a 2-QBF (quantified boolean formula) is satisfiable to the problem of deciding whether or not a projection is invariant. The former decision problem is well known to be Π_2^P-complete.

We assume that our input is a 2-QBF of the form $\forall x_1, x_2, \ldots x_k \; \exists y_1, y_2, \ldots, y_m \; \phi$. In this formula, $x_1, x_2, \ldots x_k, y_1, y_2, \ldots,$ and y_ms are boolean variables, and ϕ is a 3-CNF expression over the boolean variables. Let the number of clauses in ϕ be w.

We transform the above 2-QBF into the GDL description of the following single player game. Our game consists of $k+1$ base propositions: `(base x1)`, `(base x2)`, ..., `(base xk)` and `(base (cell 1))`.

A game state that contains the proposition `xi` in a game state corresponds to a truth assignment where the boolean variable x_i is assigned the truth value *true*, and a game state that does not contain `xi` corresponds to an assignment where x_i is assigned the truth value *false*.

The game player is denoted using the object constant `p`. In each state, `p` has k legal moves i.e. to toggle `xi` for $1 \leq i \leq k$.

```
(<= (legal p (toggle x1)))
...
(<= (legal p (toggle xk)))
```

Toggling ***xi*** *flips its truth value in the next state. However, this action does not affect the truth value of* ***xj*** *if* $i \neq j$*. We encode this behavior in our game as follows.*

```
(<= (next ?x) (does p (toggle ?x)) (not (true ?x)))
(<= (next ?x) (does p (toggle ?y)) (true ?x) (distinct ?x ?y))
```

The initial state of game consists of only one proposition: `(cell 1)`. We use a view called `all` to characterize all possible truth assignments to the existentially quantified variables in ϕ. The view `all` is defined as follows.

```
(<= (all ?A1 ?A2 ... ?Am)  (y1 ?A1) (y2 ?A2) ... (ym ?Am))
(y1 t) (y1 f)
...
(ym t) (ym f)
```

For every clause c_i in ϕ, we characterize the satisfying assignments of c_i as a view called `ci` in our game. Suppose the i^{th} clause c_i in ϕ is $x_j \vee y_k \vee \neg y_l$. Then, the view `ci` is defined as follows.

```
(<= (ci ?A1 ?A2 ... ?Am)
    (all ?A1 ?A2 ... ?Am) (not (di ?A1 ?A2 ... ?Am)))
(<= (di ?A1 ... ?Ak-1 f ... ?Al-1 t ... ?Am)
    (all ?A1 ... ?Ak-1 f ... ?Al-1 t ... ?Am) (not (true xj)))
```

For a supplied assignment of universally quantified variables, we indicate the existence of a satisfiable assignment for ϕ using a view called `phi` which is defined as follows.

```
(<= phi  (c1 ?A1 ?A2 ... ?Am) ... (cw ?A1 ?A2 ... ?Am))
```

Finally, we add the following update rule which states that if `(cell 1)` and `phi` are true in the current state, then `(cell 1)` is true in the next state.

```
(<= (next cell 1) (true (cell 1)) phi)
```

The *generated instance* in the above translation is to decide whether or not $\langle \texttt{cell}, \{1\}\rangle$ is an invariant projection. We show that the above translation is indeed a reduction i.e. the supplied 2-QBF instance is satisfiable *if and only if* $\langle \texttt{cell}, \{1\}\rangle$ is an invariant projection of the above game.

Suppose that the supplied 2-QBF instance is satisfiable i.e. ϕ is satisfiable for every truth assignment to the universally quantified variables i.e. $x_i\,(1 \leq i \leq k)$. This means that proposition `phi` is true in every game state, and as a consequence, for *any* game state s and a state t that is reachable from s using a legal move, `(cell 1)` $\in s \implies$ `(cell 1)` $\in t$. Therefore, $\pi(\langle \texttt{cell}, \{1\}\rangle, s) = \pi(\langle \texttt{cell}, \{1\}\rangle, t)$. Let s_i denote the initial state of the game. Since `(cell 1)` $\in s_i$, we have $\pi(\langle \texttt{cell}, \{1\}\rangle, s_i) = \pi_b(\langle \texttt{cell}, \{1\}\rangle)$. Therefore, $\langle \texttt{cell}, \{1\}\rangle$ is invariant.

Suppose that $\langle \texttt{cell}, \{1\}\rangle$ is an invariant projection. Then, for each of the 2^k game states which contain `(cell 1)`, the view proposition `phi` is true. Since these 2^k states correspond to different truth assignments to the universally quantified variables, and `phi` being true corresponds to ϕ being satisfiable, the supplied 2-QBF instance is satisfiable. Therefore, the problem of deciding whether or not a projection is invariant is Π_2^P-complete. □

Computing all Invariant Projections. Suppose *Verify* is a procedure that takes as input the GDL description of a game and a projection t, and outputs whether or not t is invariant. A naive method for computing *all invariant projections* of a supplied game G is as follows. Let the constant r denote a k-ary relation that appears in the set of base propositions, and let P_r be the set of projections containing r. Then, the set of all invariant projections containing $r = \{t \mid t \in P_r \wedge \mathit{Verify}(G, t)\}$.

Note that in the above computation, $2^k - 1$ projections are verified. However, the above approach may perform verifications that are *redundant*. We can prune the set of candidate projections that need to be verified using the following result.

Theorem 2. *If $\langle \mathtt{r}, S\rangle$ is an invariant projection, then for every non-empty subset $T \subset S$, $\langle \mathtt{r}, T\rangle$ is also an invariant projection.*

Proof. Suppose that r is a k-ary relation, and $\langle \mathtt{r}, S\rangle$ is an invariant projection. Suppose that $S = \{A_1, \ldots, A_m\}$ and $T = \{B_1, \ldots, B_l\}$ is a subset of S.

From Section 2 we note that, $\pi(\langle \mathtt{r}, T\rangle, s)$ is equivalent to evaluating the following query q on the game state.

$$(\Leftarrow (q\,?X_{B_1}\,?X_{B_2}\,\ldots ?X_{B_l})\,(r\,?X_1\,?X_2\,\ldots ?X_k))$$

The query q may be reformulated as follows.

$$(\Leftarrow (q\,?X_{B_1}\,?X_{B_2}\,\ldots ?X_{B_l})\,(q_1\,?X_{A_1}\,?X_{A_2}\,\ldots ?X_{A_m}))$$
$$(\Leftarrow (q_1\,?X_{A_1}\,?X_{A_2}\,\ldots ?X_{A_m})\,(r\,?X_1\,?X_2\,\ldots ?X_k))$$

In the above reformulation, the set of answers obtained by evaluating q_1 in state s is $\pi(\langle \mathtt{r}, S\rangle, s)$. Since $\langle \mathtt{r}, S\rangle$ is invariant, q_1 produces identical answers for the following pairs of states: (a) initial state and the set of all base propositions, and (b) any state s that is reachable from the initial state and its next reachable state s'. As a consequence of the above reformulation, q also produces identical answers in the above pairs of states. Therefore, $\langle \mathtt{r}, T\rangle$ is also an invariant projection. □

Input: GDL description of a game G, a k-ary relation r
Initialization: $S \leftarrow \{1, 2, \ldots, k\}, i \leftarrow 1$.
Set of candidate projections $C_1 = \{\langle \mathtt{r}, T\rangle$ such that T is a singleton subset of $S\}$
for projection $t \in C_1$ **do**
  $IP_1 = \{t$ such that $t \in C_1$ and *Verify*(G, t) is true$\}$
end for
repeat
  $i \leftarrow i + 1$
  $C_i =$ cartesian product of C_1 and C_{i-1}
  Prune from C_i projections of the form $\langle \mathtt{r}, T\rangle$ if $\exists S$ such that $S \subset T$, cardinality of $S = i-1$, and $\langle \mathtt{r}, T\rangle \notin IP_{i-1}$
  $IP_i = \{t$ such that $t \in C_i$ and *Verify*(G, t) is true$\}$
until $IP_i = \varnothing$ or $i > k$
Output: $\{IP_j \mid 1 \leq j \leq i\}$

Figure 3: *Prune* procedure for computing invariant projections

Pruning Candidate Projections. According to Theorem 2, invariant projections form a lattice. We can use this lattice structure to prune the search space of invariant projections as follows. Suppose $\langle \mathtt{r}, T\rangle$ is a projection. From Theorem 2, it follows that $\langle \mathtt{r}, T\rangle$ is not invariant if there exists a set S such that $S \subset T$ and $\langle \mathtt{r}, S\rangle$ is *not* invariant. This pruning strategy is presented as the *Prune* procedure in Figure 3. The set of candidate and invariant projections in the i^{th} step of *Prune* are denoted as C_i and IP_i respectively.

We note that the strategy used in *Prune* for pruning candidate projections is identical to the strategy used for pruning candidate item-sets in the Apriori Algorithm [1]. In the worst case, no candidate projections may be pruned at all. However, as has been noted in [4], and validated by our evaluation in Section 4, for k-ary relations, pruning candidate projections may potentially decrease the number of calls to the verification procedure by a factor that is exponential in k.

Practical Considerations. Even though *Prune* may efficiently limit the search space for invariant projections, an efficient implementation of *Verify* is unlikely because of Theorem 1. In competitive game playing [6], game players have limited time to reason about supplied games (denoted by start clock) and to compute their moves (denoted by play clock). To enable the use of invariant projections in competitive game playing, we propose a heuristic called *VerifyIP* for computing invariant projections. *VerifyIP* outputs that a supplied projection is invariant if it satisfies Definition 2 with the following relaxation to Property 2.

Property 2 (new). For *every state* s, if s' is a next state that can be reached by executing a legal action on s, then $\pi(t, s) = \pi_b(t) \implies \pi(t, s') = \pi_b(t)$.

Theorem 3. *(Soundness) If a projection is identified by VerifyIP to be invariant, then it also satisfies Definition 2.*

Proof. Proof of the above theorem is immediate since Property 2 (new) holds for all game states including the set of states reachable from the initial state. □

We present an implementation of *VerifyIP* by representing the above characterization as an ASP program in Figure 4. In *VerifyIP*, the assertions in Step 1 restrict the models of G_a. The

Input: GDL description of a game G, k-ary relation r, a projection $t = \langle \mathtt{r}, \{i_1, i_2, \ldots, i_j\}\rangle$.
Let $G_a \leftarrow$ the encoding of G in ASP.
Step 1: Add to G_a the following assertions:

```
{true(X): base(X)}.
{does(R,M) : input(R, M)} 1 :- role(R).
:- does(R,M), not legal(R,M).
```

Step 2: Add to G_a the following rules:

```
pi_b(Xi1,Xi2,...,Xij) :- base(r(X1,X2,...,Xk)).
pi_init(Xi1,Xi2,...,Xij) :- init(r(X1,X2,...,Xk)).
pi_s(Xi1,Xi2,...,Xij) :- true(r(X1,X2,...,Xk)).
pi_next(Xi1,Xi2,...,Xij) :- next(r(X1,X2,...,Xk)).
initial :- pi_b(Xi1,Xi2,...,Xij), not pi_init(Xi1,Xi2,...,Xij).
initial :- pi_init(Xi1,Xi2,...,Xij), not pi_b(Xi1,Xi2,...,Xij).
current :- pi_b(Xi1,Xi2,...,Xij), not pi_s(Xi1,Xi2,...,Xij).
current :- pi_s(Xi1,Xi2,...,Xij), not pi_b(Xi1,Xi2,...,Xij).
nxt :- pi_b(Xi1,Xi2,...,Xij), not pi_next(Xi1,Xi2,...,Xij).
nxt :- pi_next(Xi1,Xi2,...,Xij), not pi_b(Xi1,Xi2,...,Xij).
variant :- not current, nxt.
variant :- initial.
:- not variant.
```

Output: *True* if the modified ASP program G_a is unsatisfiable i.e. t is invariant. Otherwise, *False*.

Figure 4: *VerifyIP* heuristic for identifying invariant projections.

first assertion ensures that a game state is a subset of the set of all base propositions. The second rule encodes the constraint that exactly one action is performed by a player, which is true for all GGP games. The last assertion enforces every action to be legal.

In Step 2 of *VerifyIP*, the relations `pi_b`, `pi_i`, `pi_s`, and `pi_next` characterize the computation of $\pi_b(t), \pi(t, s_i), \pi(t, s)$, and $\pi(t, next(s))$ respectively, where s_i denotes the initial state and s denotes a state in the game. The correctness of this characterization follows from the definition of projections presented in Section 2.

Correctness of Implementation. In *VerifyIP*, if a supplied projection does not satisfy Property 1 (from Definition 2), then there exists a model of G_a that satisfies `initial`. Analogously, if the projection does not satisfy Property 2 (new), then some model of G_a falsifies `current => nxt`. If there exists a model of G_a that contains `variant` then at least one of Property 1 or Property 2 (new) is violated. Therefore, if G_a is *unsatisfiable*, then t is an invariant projection.

We note that deciding invariant projections using *VerifyIP* is computationally cheaper than using Definition 2. In the former case, the decision problem is NP-hard; where as, in the latter case, the decision problem is Π_2^P-hard. In the worst case, the running-time of *VerifyIP* may be exponential in the size of the propositional base. However, our evaluation of *VerifyIP* in Section 4 indicates that the time taken to verify whether or not a supplied projection is invariant is *reasonable* e.g. within the values of the start and play clocks used in past GGP computations.

The invariant projections that are decided by *VerifyIP* also satisfy the lattice structure as stated in Theorem 2. This can be proved by extending the proof of Theorem 2 such that the

query q_1 produces identical answers for *every* pair of states s and s', such that s' is reachable from s through a legal move. Thus, we can use the *Prune* procedure as presented in Figure 3 to compute all invariant projections that satisfy Property 2 (new), by replacing calls to $Verify(G, t)$ in *Prune* with calls to $VerifyIP(G, t)$.

4 Evaluation

In this section, first we evaluate the performance of *VerifyIP*, and the effectiveness of the *Prune* procedure. Then, we validate our claim that invariant projections strongly correspond to humans' intuition of game boards. At the end of the section, we discuss applications of invariant projections.

In our evaluation, we used GDL descriptions and stylesheets of 13 games that were selected from the General Game Playing MOOC `https://www.coursera.org/course/ggp`, and the AAAI-13 and the AAAI-15 General Game Playing competitions. A game's stylesheet maps a game state to a visualization. The game descriptions and stylesheets that are used in our evaluation are accessible at `http://gamemaster.stanford.edu`. We performed our evaluation on a 2.4 GHz Intel Core 2 Duo Processor with 4 GB RAM.

We evaluated the performance of our proposed heuristic for deciding invariant projections and our pruning strategy as follows. For every game in our test suite, we computed all invariant projections using a version of the *Prune* procedure that decides invariant projections using *VerifyIP*. We used the Clingo ASP Solver [5] to check for unsatisfiability of the ASP programs generated by *VerifyIP*. We recorded the number of calls to *VerifyIP*, the total time to compute all invariant projections, the maximum time to verify a whether or not a projection is invariant, and the maximal invariant projections. These results are recorded in Table 1.

Game	**Relation (Arity)**	**Calls**	**Time**$_{\text{total}}$	**Time**$_{\text{max}}$	**Invariant Projection**
alquerque	`cell` (3)	4	3.76	1.13	$\langle$`cell`, $\{1,2\}\rangle$
chinesecheckers	`cell` (2)	2	32.64	21.41	$\langle$`cell`, $\{1\}\rangle$
chinook	`oddcell` (3), `evencell` (3)	8	24.45	5.73	$\langle$`oddcell`, $\{1,2\}\rangle$ $\langle$`evencell`, $\{1,2\}\rangle$
eightpuzzle	`cell` (3)	6	0.5	0.1	$\langle$`cell`, $\{1,2\}\rangle$ $\langle$`cell`, $\{3\}\rangle$
horseshoe	`cell` (2)	2	0.02	0.01	$\langle$`cell`, $\{1\}\rangle$
kono	`cell` (3)	4	0.42	0.14	$\langle$`cell`, $\{1,2\}\rangle$
multiplehunter	`cell` (4)	4	16.61	12.72	$\langle$`cell`, $\{1\}\rangle$
multiplesukoshi	`cell` (5)	16	1.06	0.13	$\langle$`cell`, $\{1,2,3,4\}\rangle$
multipletictactoe	`cell` (4)	8	0.24	0.04	$\langle$`cell`, $\{1,2,3\}\rangle$
reflectconnect	`cell` (3)	4	5.59	1.8	$\langle$`cell`, $\{1,2\}\rangle$
sukoshi	`cell` (3)	4	0.04	0.01	$\langle$`cell`, $\{1,2\}\rangle$
tictactoe	`cell` (3)	4	0.04	0.01	$\langle$`cell`, $\{1,2\}\rangle$
yinsh	`cell` (3)	4	1.5	0.81	$\langle$`cell`, $\{1,2\}\rangle$

Table 1: Evaluation of *VerifyIP* and *Prune*. The computation times reported are in seconds. **Calls** represents the number of calls to *VerifyIP*.

Typically, past GGP competitions set the value of the start clock between 40-60 seconds and

the value of play clock at 10 seconds. From Table 1, it is evident that the invariant projections for every game in our test suite were computed well within the value of the start clock (= 40 seconds). For 10 games, the computation finished within the value of the play clock. These results indicate that the *VerifyIP* heuristic may be feasibly incorporated in competitive game players to decide invariant projections.

Furthermore, the results in Table 1 indicate that for a majority of the games, the space of candidate projections is pruned by half using the *Prune* algorithm. For the game of *multiplehunter*, the *Prune* procedure reduced the number of calls to *Verify* by a factor that is *exponential* in the schema size of the game. In this case, only 4 projections of the relation *cell* were checked for invariance. Since, the arity of *cell* in *multiplehunter* is 4, a naive computation would have verified 15 projections.

Since invariant projections are components of a game that do not change with gameplay, they intuitively correspond to what humans think of as *game boards*. To validate this correspondence, we computed invariant projections for all the games. We inspected each game's stylesheet to determine the projections that correspond to the game board, and checked whether or not these projections are reported by our heuristic. For all the games, except *multiplehunter*, the invariant projection reported by our heuristic was identical to the characterization of the game board as presented in the game's stylesheet. This suggests that invariant projections provide an effective characterization of game boards.

For *multiplehunter*, $\langle \texttt{cell}, \{1\} \rangle$ was the reported invariant projection; however, the game's stylesheet uses the projection $\langle \texttt{cell}, \{1, 2, 3\} \rangle$ to characterize the game board. In the GDL description of *multiplehunter*, the state of every board position is not explicitly represented. The game state consists of only those board positions that have a piece or marker placed on them. In such cases, the intended board positions may, alternatively, be conceptualized as pieces that are placed by players either directly, or indirectly as a result of placing other conventional pieces.

Applications of Invariant Projections. It has been shown in [3, 11, 14], that the identification of implicit game structures such as boards and pieces may be used to enable effective game playing heuristics. The characterization of game boards in [3, 11, 14] is based on specific signatures of the game states, and is therefore not applicable in a general setting. We discuss this topic further in Section 5. Invariant projections may potentially broaden the applicability of such game playing heuristics since invariant projections effectively characterize game boards and the definition of invariant projections is game-independent.

The problem of verifying whether or not a projection is invariant is computationally hard. The *VerifyIP* heuristic, which is sound but not complete, may be used to efficiently verify whether or not a supplied projection is invariant. Another computationally cheap heuristic for deciding invariant projections is to verify Definition 2 over a *reasonably large* subset of all game states. However, such a heuristic generates *complete* but unsound invariant projections. According to [6], a large number of competitive general game players use Monte-Carlo Tree Search (MCTS) algorithm. Since MCTS explores a large number of game states - *Prune* procedure coupled with this heuristic may be applied in conjunction with the tree search to incrementally compute and maintain invariant projections.

Invariant projections may also be used to automate the design of a game's visualization by automatically generating stylesheets for games. A game's stylesheets can be automatically generated from its GDL description by identifying implicit game structures such as boards and pieces. Since, invariant projections strongly correspond to humans' characterization of game boards, invariant projections may be used to characterize game boards in a game's visualization. The Merlin system presented in [12] uses invariant projections to discover game boards

and pieces. Merlin leverages the lattice property of invariant projections to decompose a supplied game into different sets of boards and pieces, and subsequently generate different, potentially new visualizations for games e.g. Trifecta http://stanford.edu/~abhijeet/trifecta/, which is a card-based visualization of Tic-Tac-Toe.

5 Related Work

Prior works in GGP have studied the problem of identifying implicit game structures such as boards, pieces, and latches. The proposal presented in [11] characterizes game boards by focusing on a ternary relation called `cell` in game states. In this proposal, two arguments in the `cell` relation correspond to the board positions, and the third argument corresponds to an object placed on the board. A similar definition of game boards has been presented in [14]. However, the proposal presented in [14] allows `cell` to be an n-ary ($n > 2$) relation.

Our characterization of boards as invariant projection differs from the characterization of boards presented in [11, 14] as follows. First, the characterizations presented in [11, 14] assume that each board location contains at most one *piece*. However, this is not the case with our characterization. For games like Parchessi, where multiple pieces of players may co-occupy board positions, the prior definitions fail to capture the notion of a board. In fact it has been noted in [11] that their proposed characterization may reject valid boards.

Second, each board location appears in all game states in our proposal. However, this is not the case in prior definitions. Third, invariant projections allow multiple arguments in a relation to characterize the contents of a board location (or *pieces*). This is, however, not the case in prior definitions where only one argument may represent the contents of a board [11]. Consider a game whose states are captured in a 4-ary relation, where the first and the last two arguments characterize the board locations and pieces respectively. The game boards for this game prior *cannot* be identified using prior definitions of boards. Finally, in prior proposals, special meaning is attached to the relation constant `cell` which is *not* a GDL keyword. This is not the case in our approach.

Prior works [9, 13] in GGP have also addressed the problem of automatically proving properties of games. The technique presented in [13] translates the properties of games as Alternating-time Temporal Logic (ATL) formulae. Verification of these properties is shown to be EXPTIME-complete, and is done by interpreting the corresponding ATL formulate over all reachable states. The technique presented in [9] encodes properties of games as *sequence state invariants*, and proves these properties using Answer Set Programming similar to our approach in *VerifyIP*.

6 Conclusion and Future Work

In this paper, we define a new property of game states called invariant projections that strongly corresponds to humans' intuition of game boards. Invariant projections may be applied to identify game boards in enable effective game playing heuristics and in designing game visualizations. One potential direction in which our work may be expanded is to model other game structures and concepts such as *pieces* and their capture, and the type of game play. For example, configuration games e.g. 8-puzzle may be characterized as games that contain invariant projections of the form $\langle \mathtt{r}, S\rangle$ and $\langle \mathtt{r}, T\rangle$ where r is a k-ary relation, $S \cap T = \emptyset$ and $\mid S \cup T \mid = k$. Another interesting future topic for investigation is whether identification of invariant projections may be leveraged optimize game execution, analogous to loop invariant code motion [2] in compilers.

References

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499. Morgan Kaufmann Publishers Inc., 1994.

[2] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[3] Steve Draper and Andrew Rose. Sancho GGP Player: Heuristics. http://sanchoggp.blogspot.com/2014/06/heuristics.html, June 2014. Accessed: 2016-06-20.

[4] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall Press, 2 edition, 2008.

[5] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider. Potassco: The Potsdam Answer Set Solving Collection. *AI Communications*, 24(2):107–124, 2011.

[6] Michael R. Genesereth and Yngvi Björnsson. The International General Game Playing Competition. *AI Magazine*, 34:107–111, 2013.

[7] Michael R. Genesereth, Nathaniel Love, and Barney Pell. General Game Playing: Overview of the AAAI Competition. *AI Magazine*, 26:62–72, 2005.

[8] Michael R. Genesereth and Michael Thielscher. *General Game Playing*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2014.

[9] Sebastian Haufe, Stephan Schiffel, and Michael Thielscher. Automated Verification of State Sequence Invariants in General Game Playing. *Artificial Intelligence*, 187:1–30, 2012.

[10] Miki Hermann. Constrained Reachability is NP-complete. http://www.lix.polytechnique.fr/~hermann/publications/const-reach.pdf, 1998. Accessed: 2016-06-20.

[11] Gregory Kuhlmann, Kurt Dresner, and Peter Stone. Automatic Heuristic Construction in a Complete General Game Player. In *Proceedings of the 21st AAAI Conference on Artificial Intelligence*, pages 1457–1462. AAAI Press, 2006.

[12] Abhijeet Mohapatra and Michael Genesereth. Automating the Design of Game Visualizations. Technical Report LG-2015-01, Stanford University, Stanford, CA, January 2015. http://logic.stanford.edu/reports/LG-2015-01.pdf.

[13] Ji Ruan, Wiebe van der Hoek, and Michael Wooldridge. Verification of Games in the Game Description Language. *Journal of Logic and Computation*, 19:1127–1156, 2009.

[14] Stephan Schiffel and Michael Thielscher. Fluxplayer: A Successful General Game Player. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, pages 1191–1196. AAAI Press, 2007.