Automated Theorem Proving for General Game Playing

Stephan Schiffel and **Michael Thielscher**
Department of Computer Science
Dresden University of Technology
{stephan.schiffel,mit}@inf.tu-dresden.de

Abstract

A general game player is a system that understands the rules of an unknown game and learns to play this game well without human intervention. To succeed in this endeavor, systems need to be able to extract and prove game-specific knowledge from the mere game rules. We present a practical approach to this challenge with the help of Answer Set Programming. The key idea is to reduce the automated theorem proving task to a simple proof of an induction step and its base case. We prove correctness of this method and report on experiments with an off-the-shelf Answer Set Programming system in combination with a successful general game player.

1 Introduction

General Game Playing is concerned with the development of systems that understand the rules of previously unknown games and learn to play these games well without human intervention. Identified as a Grand Challenge for Artificial Intelligence, this endeavor requires to combine methods from a variety of a sub-disciplines, including reasoning, search, game playing, and learning [Pell, 1993; Genesereth *et al.*, 2005]. Recent research in this area has led to two successful approaches to General Game Playing: simulation-based systems which use Monte Carlo game tree search [Finnsson and Björnsson, 2008]; and knowledge-based systems [Kuhlmann *et al.*, 2006; Clune, 2007; Schiffel and Thielscher, 2007], which rely on the ability to automatically extract game-specific knowledge from the rules of a game. This knowledge serves a variety of purposes that are crucial for good play:

- Games need to be classified in order to choose the right search method—e.g., Minimax with α/β is only suitable for two-player, zero-sum, and turn-taking games.
- Recognition of structures like boards, pieces, and mobility of pieces is needed to automatically construct good evaluation functions for the assessment of intermediate positions.
- Game-specific knowledge can be used to cut off the search in positions that are provably lost for the player.

While existing systems like [Clune, 2007] extract this kind of knowledge, they do not actually attempt to prove it; rather they generate random sample matches to test whether a property is violated at some point, and then rely on the correctness of this informed guess. The first method of automatically proving properties for general games is presented in [van der Hoek *et al.*, 2007]. But this requires to systematically search the entire set of reachable positions in a game and therefore is not suitable for practical play. Finding a practical method of rigorously proving game-specific knowledge from the mere rules of a game is an open and challenging problem.

In this paper, we present a first solution to this problem in the form of a method which allows systems to automatically prove properties that hold across all legal positions. For a general game player this is arguably the most important type of game-specific knowledge. We show how the focus on these properties allows us to reduce the automated theorem proving task to a simple proof of an induction step and its base case. Specifically, we will use the paradigm of Answer Set Programming (ASP) (see, e.g., [Gelfond, 2008]) to validate properties of games specified in the general Game Description Language (GDL) [Genesereth *et al.*, 2005]. This opens up possibilities for deploying off-the-shelf ASP systems in general game players to prove game-specific knowledge.

The rest of the paper is organized as follows. In the next section, we will recapitulate the basic syntax and semantics of GDL. In the section that follows, we will show how the concept of answer sets can be applied in the context of GDL in order to systematically prove game-specific properties. The correctness of this method will be formally proved, and we will report on experiments with an off-the-shelf ASP system [Potassco, 2008] in combination with a successful general game player [Schiffel and Thielscher, 2007]. Before we start, however, let us stress that in this paper we are only concerned with automatically *proving* properties, not with automatically *finding* suitable properties worth proving. We refer to [Clune, 2007; Schiffel and Thielscher, 2007] for an extensive discussion on various types of game-specific knowledge that helps a general game player find good heuristics.

2 Game Description Language

The Game Description Language (GDL) has been developed to formalize the rules of any finite game with complete information in such a way that the description can be automatically processed by a general game player. Due to lack of space, we

```
role(xplayer).  role(oplayer).
init(control(xplayer)).  init(cell(1,1,b)).  ...  init(cell(3,3,b)).

legal(P,mark(X,Y))  :- true(control(P)), true(cell(X,Y,b)).
legal(xplayer,noop) :- true(control(oplayer)).
legal(oplayer,noop) :- true(control(xplayer)).

next(cell(M,N,x)) :- does(xplayer,mark(M,N)).
next(cell(M,N,o)) :- does(oplayer,mark(M,N)).
next(cell(M,N,W)) :- true(cell(M,N,W)), does(P,mark(I,J)), distinct(M,I).
next(cell(M,N,W)) :- true(cell(M,N,W)), does(P,mark(I,J)), distinct(N,J).
next(control(oplayer)) :- true(control(xplayer)).
next(control(xplayer)) :- true(control(oplayer)).
```

Figure 1: A GDL description of Tic-Tac-Toe (without the definition of termination and goalhood). A position in this game is encoded using the features *control*(P), where $P \in \{xplayer, oplayer\}$, and *cell*$(X,Y,C)$, where $X, Y \in \{1,2,3\}$ and $C \in \{x, o, b\}$ (the last symbol stands for "blank"). `distinct(X,Y)` is an auxiliary standard predicate in GDL which means syntactic inequality of the two arguments.

can give just a very brief introduction to GDL and have to refer to [Love *et al.*, 2006] for details.

GDL is based on the standard syntax of logic programs, including negation. We assume familiarity with the basic notions of logic programming as can be found, e.g., in [Lloyd, 1987]. We adopt the Prolog convention according to which variables are denoted by uppercase letters and predicate and function symbols start with a lowercase letter. As a tailor-made specification language, GDL uses a few pre-defined predicate symbols shown in the table below.

`role(R)`	`R` is a player
`init(F)`	`F` holds in the initial position
`true(F)`	`F` holds in the current position
`legal(R,M)`	player `R` has legal move `M`
`does(R,M)`	player `R` does move `M`
`next(F)`	`F` holds in the next position
`terminal`	the current position is terminal
`goal(R,N)`	player `R` gets goal value `N`

GDL imposes some restrictions on the use of these keywords:

- `role` only appears in facts;
- `init` and `next` only appear as head of clauses, and `init` is not connected to any of `true`, `legal`, `does`, `next`, `terminal`, or `goal`;
- `true` and `does` only appear in clause bodies with `does` not connected to `legal`, `terminal`, or `goal`.

As an example, Figure 1 shows an excerpt of a GDL description for the simple game of Tic-Tac-Toe. GDL imposes some further, general restrictions on a set of clauses with the intention to ensure finiteness of the set of derivable predicate instances. Specifically, the program must be *stratified* [Apt *et al.*, 1987] and *allowed* [Lloyd and Topor, 1986]. Stratified logic programs are known to admit a specific *standard model* [Apt *et al.*, 1987].

Based on the concept of the standard model, a GDL description can be understood as a state transition system as follows (see [Schiffel and Thielscher, 2009] for details). To begin with, any valid game description G in GDL contains a finite set of function symbols, including constants, which implicitly determines a set of ground terms Σ. This set constitutes the symbol base Σ in the formal semantics for G.

The players and the initial position of a game can be directly determined from the clauses for, respectively, `role` and `init` in G. In order to determine the legal moves, update, termination, and goalhood for any given position, this position has to be encoded first, using the keyword `true`. To this end, for any *finite* subset $S = \{f_1, \ldots, f_n\} \subseteq \Sigma$ of a set of ground terms, the following set of logic program facts encodes S as the current position:

$$S^{\mathtt{true}} \stackrel{\text{def}}{=} \{\mathtt{true}(f_1)\mathtt{.}, \ldots, \mathtt{true}(f_n)\mathtt{.}\}$$

Furthermore, for any function $A : (\{r_1, \ldots, r_n\} \mapsto \Sigma)$ that assigns a move to each player $r_1, \ldots, r_n \in \Sigma$, the following set of facts encodes A as a joint move:

$$A^{\mathtt{does}} \stackrel{\text{def}}{=} \{\mathtt{does}(r_1, A(r_1))\mathtt{.}, \ldots, \mathtt{does}(r_n, A(r_n))\mathtt{.}\}$$

Definition 1 *Let G be a GDL specification whose signature determines the set of ground terms Σ. Let 2^Σ be the set of* finite *subsets of Σ. The* semantics *of G is the state transition system (R, S_1, T, l, u, g) where*[1]

- $R = \{r \in \Sigma : G \models \mathtt{role}(r)\}$ *(the players);*
- $S_1 = \{f \in \Sigma : G \models \mathtt{init}(f)\}$ *(the initial position);*
- $T = \{S \in 2^\Sigma : G \cup S^{\mathtt{true}} \models \mathtt{terminal}\}$ *(the terminal positions);*
- $l = \{(r, a, S) : G \cup S^{\mathtt{true}} \models \mathtt{legal}(r, a)\}$*, where $r \in R$, $a \in \Sigma$, and $S \in 2^\Sigma$ (the legality relation);*
- $u(A, S) = \{f \in \Sigma : G \cup S^{\mathtt{true}} \cup A^{\mathtt{does}} \models \mathtt{next}(f)\}$*, for all $A : (R \mapsto \Sigma)$ and $S \in 2^\Sigma$ (the update function);*
- $g = \{(r, n, S) : G \cup S^{\mathtt{true}} \models \mathtt{goal}(r, n)\}$*, where $r \in R$, $n \in \mathbb{N}$, and $S \in 2^\Sigma$ (the goal relation).*

This definition provides a formal semantics by which a GDL description is interpreted as an abstract n-player game: in every position S, starting with S_1, each player r chooses a

[1]Below, entailment $\models$ is via the abovementioned standard model for a set of clauses.

move a that satisfies $l(r,a,S)$. As a consequence the game state changes to $u(A,S)$, where A is the joint move. The game ends if a position in T is reached, and then g determines the outcome. The restrictions in GDL ensure that entailment wrt. the standard model is decidable and that only finitely many instances of each predicate are entailed. This guarantees that the definition of the semantics is effective.

3 Proving Properties of General Games Using Answer Set Programming

We are now ready to define the challenge addressed in this paper: given a GDL description of an unknown game, how can a general game player fully automatically prove game-specific knowledge in form of properties that hold across all finitely reachable positions?

As an example, recall the formal description of Tic-Tac-Toe given in Figure 1. These rules and their semantics according to Definition 1 imply that the argument of the feature *control*(P) is unique in every legal position. The ability to derive this fact is essential for a general game player to be able to identify Tic-Tac-Toe as a turn-taking game. A similar but less obvious consequence of the given description is the uniqueness of the third argument of *cell*(X,Y,C) in every legal position. This knowledge may help a general game player to identify this feature as representing a two-dimensional "board" with "markers" C.

As long as a game is finite, properties of this kind can in principle be determined by a complete search through the state transition diagram for a game [van der Hoek *et al.*, 2007]. However, for games that are simple enough to make this practically feasible, a general game player does not actually need game-specific knowledge because it can solve the game by exhaustive search anyway. For this reason, the challenge for the practice of General Game Playing (GGP) is to develop a local proof method. In case of game-specific properties that hold across all reachable states, the key idea is to reduce the automated theorem proving task to a simple proof of an induction step and its base case.

In the specific setting of GGP, proving a property φ by induction means to show that (1) φ holds in the initial position, and (2) if φ holds in a position and all players choose legal moves, then φ holds in the next position, too. Because game descriptions in GDL are logic programs with negation, this general proof principle can be put into practice with the help of Answer Set Programming (ASP). Answer sets provide models of logic programs with negation according to the following definition (for details, see e.g. [Gelfond, 2008]).

Definition 2 *Let* Γ *be a logic program with negation over a given signature, and let* $\text{ground}(\Gamma)$ *be the set of all ground (i.e., variable-free) instances of rules in* Γ*. For a set* M *of ground atoms (i.e., predicates with variable-free arguments), the* reduct of $\text{ground}(\Gamma)$ wrt. M *is obtained by deleting*

1. *all rules with some* $\neg p$ *in the body such that* $p \in M$*, and*
2. *all negated atoms in the bodies of the remaining rules.*

Then M *is an* answer set *for* Γ *if* M *is the least Herbrand model of the reduct of* $\text{ground}(\Gamma)$ *wrt.* M*.*

In the following, we use two common additions that have been defined for ASP [Niemelä *et al.*, 1999]: a *weight atom*

$$m \,\{\, p : d(\vec{x}) \,\}\, n$$

means that for atom p an answer set has at least m and at most n different instances that satisfy $d(\vec{x})$. Both m and n can be omitted, in which case there is no lower (respectively, upper) bound. A *constraint* is a rule `:-` $b_1, \ldots, b_k$, which excludes any answer set that satisfies $b_1, \ldots, b_k$.

As an example, consider the program

```
cdom(xplayer).
cdom(oplayer).
init(control(xplayer)).
init(cell(1,1,b)). ... init(cell(3,3,b)).
t0 :- 1{init(control(X)):cdom(X)}1.
:- t0.
```

where *cdom* defines the domain of the *control* feature. This program has *no* answer set, because the facts imply that there is exactly one instance of *cdom*(X) such that *init*$($*control*$(X))$ holds, and hence "theorem" t_0 must be true, which contradicts `:-t0`. This is a *proof* of the fact that exactly one instance of *control*(X) holds in the initial position according to the rules of Tic-Tac-Toe. In a similar fashion, we can prove the induction step for this uniqueness property by adding the following to the rules of Figure 1:

```
 1: cdom(xplayer).
 2: cdom(oplayer).
 3: fdom(control(X)) :- cdom(X).
 4: fdom(cell(X,Y,C)) :- ...
 5: mdom(mark(X,Y)) :- ...
 6: mdom(noop).
 7: {true(F):fdom(F)}.
 8: h0 :- 1{true(control(X)):cdom(X)}1.
 9: :- ¬h0.
10: 1{does(R,M):mdom(M)}1 :- role(R).
11: :- does(R,M), ¬legal(R,M).
12: t :- 1{next(control(X)):cdom(X)}1.
13: :- t.
```

Here, lines 3–6 are assumed to provide appropriate definitions of the domains for the features and the moves, respectively, as determined by the rules in Figure 1. Line 7 allows for arbitrary instances of the given features to hold in a current position, while lines 8–9 axiomatize the induction hypothesis that exactly one instance of *control*(X) is true in the current position. Line 10 requires every player to select exactly one move, and line 11 excludes any illegal move. Finally, lines 12 and 13 together encode the negation of the "theorem" that *control*(X) is unique in the next position, too. Again, the program admits no answer set, which proves the claim.

An interesting aspect of inductively proving properties of general games can be observed when trying to verify uniqueness of the third argument of *cell*(X,Y,C) in the same way. The straightforward attempt produces a counter-example to the induction step, namely, an answer set containing

```
true(cell(1,1,b)),
true(control(xplayer)),
true(control(oplayer)),
does(xplayer,mark(1,1)),
does(oplayer,mark(1,1)),
next(cell(1,1,x)), next(cell(1,1,o))
```

In this model, cell $(1,1)$ has a unique contents in the current position but then gets marked by both players simultaneously, which is a perfectly legal joint move under the assumption that each of the two players has the control! This shows that a proof may require to incorporate previously derived knowledge. The above answer set—and other, similar ones—disappear when one adds the assumption that in the current position exactly one instance of $\mathit{control}(X)$ holds. In the following section, we describe this proof method in general and show its correctness under the semantics of GDL as given in Definition 1.

4 The General Proof Method and its Correctness

When employing Answer Set Programming to automatically prove that all finitely reachable positions in a game satisfy a property φ, a general game player proceeds as follows.

Let G be a given GDL specification. The proof method requires additional, negation-free clauses $\mathcal{D}$ that define the domains of the features and moves according to G using predicates *fdom* and *mdom*, respectively.[2] Furthermore, for $p \in \{\texttt{init}, \texttt{true}, \texttt{next}\}$ let φ^p be an atom that, together with an associated set of clauses, encodes the fact that φ is satisfied in the state represented by keyword p. In other words, for every answer set M for $\varphi^{\texttt{init}}$, for example, the position $\{f : \texttt{init}(f) \in M\}$ satisfies φ according to the following definition.[3]

Definition 3 *Let G be a valid GDL specification whose signature determines the set of ground terms Σ. A* state property *is a first-order formula φ over a signature whose ground* atoms *are from Σ. Let $S \in 2^\Sigma$ be a state in the transition system for G (cf. Definition 1), then the notion of S* satisfying *φ (written: $S \models \varphi$) is defined on the basis of*

$$S \models f \text{ iff } f \in S \text{ (where } f \text{ atomic and ground)}$$

and with the usual definition for the logical connectives.

The automatic proof that φ holds in all reachable states in the game described by G is then obtained in two steps.

1. Show that there is no answer set for $G \cup \mathcal{D}$ augmented by

```
t0 :- φ^init.
:- t0.                                  (1)
```

2. Suppose that Φ is a (possibly empty) conjunction of state properties that have been proved earlier to hold across all legal game positions. Show that there is no answer set for $G \cup \mathcal{D}$ augmented by

```
{true(F) : fdom(F)}.
h :- Φ^true.
:- ¬h.
h0 :- φ^true.
:- ¬h0.                                 (2)
1{does(R,M) : mdom(M)}1 :- role(R).
:- does(R,M), ¬legal(R,M).
t :- φ^next.
:- t.
```

The correctness of this general proof method can be shown with the help of the following three theorems.

Theorem 1 Let (R, S_1, T, l, u, g) be a state transition system with symbol base Σ as in Definition 1. Let φ be a state property. If there is a finitely reachable state in 2^Σ which does not satisfy φ, then

1. $S_1 \not\models \varphi$, or
2. there is a finitely reachable state $S \in 2^\Sigma$ and a mapping $A : (R \mapsto \Sigma)$ such that
   - $S \models \varphi$;
   - $(r, A(r), S) \in l$ for all $r \in R$; and
   - $u(A, S) \not\models \varphi$.

Proof: Reachability means that successively, starting in state S_1, a joint move for the roles is chosen that is legal according to l, and then the current state is updated according to u. Given that a finitely reachable state exists that violates φ, the sequence leading to this state must contain a first one with this property. This state is either S_1 or has a predecessor that satisfies φ, which implies the claim. □

Theorem 2 Consider a valid GDL specification G whose semantics is (R, S_1, T, l, u, g). Let $\mathcal{D}$ be a negation-free program defining the domains for the features and moves according to G. For any state property φ for which $G \cup \mathcal{D} \cup \{(1)\}$ does not admit an answer set, we have that $S_1 \models \varphi$.

Proof: We prove that if $S_1 \not\models \varphi$ then $G \cup \mathcal{D} \cup \{(1)\}$ admits an answer set. Since $G \cup \mathcal{D}$ is stratified, it admits a unique answer set M that coincides with its standard model [Gelfond, 2008]. Hence, assumption $S_1 \not\models \varphi$ implies that $M \not\models \varphi^{\texttt{init}}$. This in turn implies that M is also an answer set for $G \cup \mathcal{D} \cup \{(1)\}$ (in which `t0` is false). □

Theorem 3 Consider a valid GDL specification G whose semantics is (R, S_1, T, l, u, g). Let $\mathcal{D}$ be a negation-free program defining the domains for the features and moves according to G. For any state properties Φ and φ for which $G \cup \mathcal{D} \cup \{(2)\}$ does not admit an answer set, there does not exist a state $S \in 2^\Sigma$ and a mapping $A : (R \mapsto \Sigma)$ such that

1. $S \models \Phi$ and $S \models \varphi$;
2. $(r, A(r), S) \in l$ for all $r \in R$; and
3. $u(A, S) \not\models \varphi$.

Proof: We prove that $G \cup \mathcal{D} \cup \{(2)\}$ admits an answer set whenever there exists a state S and a mapping A that satisfy the given conditions 1–3. Since $G \cup \mathcal{D} \cup S^{\texttt{true}} \cup A^{\texttt{does}}$

[2] A suitable $\mathcal{D}$ can be easily computed on the basis of the dependency graph for G; see Section 5.

[3] Note that in games with finitely many features such an encoding always exists: any φ can in principle be represented by an exhaustive propositional formula, although in practice a compact encoding is desirable (as in the example in the preceding section, where `1{ init(control(X)) : cdom(X) }1` is used to encode the fact that state property $(\exists! X)\, \mathit{control}(X)$ holds in the initial state).

is stratified, it admits a unique answer set M. According to conditions 1 and 2, M is also an answer set for $G \cup \mathcal{D}$ augmented by the first seven clauses in (2). Because condition 3 implies that $M \not\models \varphi^{\texttt{next}}$, model M is also an answer set for $G \cup \mathcal{D} \cup \{(2)\}$ (in which `t` is false). $\Box$

To summarize, if condition 1 in Theorem 1 is satisfied, then $G \cup \mathcal{D} \cup \{(1)\}$ must admit an answer set, and if condition 2 in Theorem 1 is satisfied, then $G \cup \mathcal{D} \cup \{(2)\}$ must admit an answer set under the assumption that all reachable states satisfy Φ. Hence, if neither is the case then the property in question must hold across all finitely reachable states. This completes the correctness proof.

5 An Automated Theorem Prover for GGP

The above method was implemented using Clingo [Potassco, 2008] as ASP solver in combination with the GGP system described in [Schiffel and Thielscher, 2007]. The answer set programs are automatically generated from the rules of a game. The domains, or more precisely supersets thereof, of all predicates and functions of a given game description are computed by generating a dependency graph from the clauses. The graph contains one node for every argument position of every function and predicate symbol, and one node for every function symbol itself (including each constant). An edge is added between an argument node and a function symbol node if the latter appears in the respective argument of a function or predicate in a rule of the game. An edge between two argument position nodes is added if there is a rule in the game in which the same variable appears in both arguments. Argument positions in each connected component of the graph share a domain. The constants and function symbols in the connected components are the domain elements. Specifically, we take as the domain of the moves that of the second argument of standard predicate `legal`, and as the domain of the features the union of the domains of `init` and `next`.

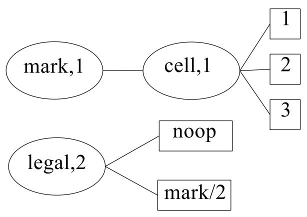

Figure 2: A dependency graph for calculating domains of functions and predicates. (Ellipses denote argument positions of functions or predicates, and rectangles denote function symbols themselves, including constants.)

Figure 2 shows a small part of the dependency graph for the game from Figure 1. The first argument of *mark* and the first argument of *cell* are connected because they share variable `M` in the game rule `next(cell(M,N,x)) :- does(xplayer,mark(M,N))`. The constants $1, 2, 3$ occur in the first argument of *cell* in several of the `init` rules. Thus the first arguments of *mark* and *cell* have the same domain $\{1, 2, 3\}$. Similarly, the constant *noop* and the binary function symbol *mark* occur in the second argument of `legal` in several rules. Therefore, the moves of the game consist of the constant *noop* and every possible instance of *mark*(X, Y), which in turn are determined by the domains for the arguments of *mark*.

After computing the domains, a general game player can use ASP to prove any game property it may find useful. For example, the player can conduct a systematic search for the uniqueness of arguments of features in the following way. Starting with $\Phi = \emptyset$, for every combination of arguments of each feature the system checks if these arguments are unique. Whenever some property is proved it is added to Φ and search is restarted with the properties that have not yet been proved. This simple method can be improved significantly if a property φ that may depend on some other property φ' is not tested until after testing φ'. A property of the arguments of a feature f may depend on a feature f' if f' occurs in the body of a rule matching `next`(f) or in a `legal` rule for a move that occurs in a rule matching `next`(f).

6 Experimental Results

We conducted experiments with our system using a wide range of games from previous GGP competitions [Genesereth *et al.*, 2005], 12 of which are summarized in Figure 3. Five different experiments were run on each game:

- proving that the argument of the *control*(P) feature is unique in every state (provided the feature exists);
- proving that the contents of a board cell is unique (to which end boards were identified manually);
- proving that the contents of a board cell is unique given the information that the argument of *control*(P) is;
- proving that the game is zero-sum;
- systematic search for unique arguments as described at the end of the preceding section.

As can be seen from the results, proving some properties (e.g., control and zero-sum) is very fast and successful for most of the games while proving other properties (e.g., board) is usually expensive and only possible in a few games. The main influence on the time and space required is the number and size of the rules of the answer set program. Since only those game rules were included in the answer set program which are potentially relevant for proving the respective property, the size of the answer set program depends on the connectedness between the features and moves of a game. For example, the relevant rules for feature *control*(P) typically do not depend on other features and in most cases not even on which moves are made. Therefore, the more complex rules for `next(F)` with `F` $\neq$ *control*(P) and `legal(R,M)` can be omitted when proving uniqueness of control. The rules for the board feature, on the other hand, are usually more complex, so that in complex games answer sets cannot be computed under suitable memory limitations.[4]

[4]Abstracting from certain features of the game may help in these cases; e.g., for the *piece_count* feature in Checkers the actual locations of the moved pieces do not matter—it is only relevant how many jumps were made.

	control	board	board given control	zero-sum	all unique arguments
3pttc	(yes,0.00)	(no,0.24)	(no,0.28)	(no,0.00)	1.99
amazons	(yes,0.00)	(-,70.18)	(-,70.08)	(yes,0.38)	491.36
blocker	n/a	(yes,0.00)	n/a	(yes,0.00)	0.11
checkers	(yes,0.02)	(-,25.96)	(-,25.94)	(yes,1.14)	256.07
connectfour	(yes,0.00)	(no,0.01)	(yes,0.01)	(yes,0.01)	0.14
endgame	(yes,0.02)	(-,61.50)	(-,61.37)	(yes,0.12)	433.83
knightthrough	(yes,0.00)	(no,8.34)	(yes,51.36)	(yes,0.00)	56.02
othello	(yes,0.12)	(-,60.01)	(-,61.13)	(-,57.03)	422.0
pacman3p	(yes,0.01)	(no,0.33)	(no,0.27)	(no,0.09)	3.05
quarto	n/a	(no,2.36)	n/a	(yes,3.14)	16.52
tictactoe	(yes,0.00)	(no,0.00)	(yes,0.00)	(yes,0.00)	0.13
tttcc4	(yes,0.01)	(-,64.00)	(-,64.00)	(no,0.01)	447.65

Figure 3: Results of proving properties and times in seconds it took for a selection of games. Experiments were run on an Intel Core 2 Duo cpu with 3.16GHz. "-" means the prover was aborted (after the given time) because it used more than 1 GB RAM. (We enforce this rather strict limit in our general game player in view of practical play, because proving properties is only one of many aspects of analyzing a game description.)

Another reason some properties were not proved by our system is that they can only be proved simultaneously with other properties. Changing the algorithm to accommodate for interdependent properties should be straightforward. However, in the worst case all combinations of properties would have to be considered.

7 Conclusion

The ability to prove properties of hitherto unknown games is a core ability of a successful general game playing system. We have shown that Answer Set Programming provides a theoretically grounded and practically feasible approach to this challenge, which not only is more reliable but often even faster than making informed guesses based on random sample matches. On the other hand, our experiments have also shown that state-of-the-art ASP systems cannot always be applied to prove properties of complex games in time reasonable for practical play. A promising alternative approach to tackle these games is given by the very recently developed method of first-order Answer Set Programming [Lee and Meng, 2008], by which grounding is avoided. A major challenge for future work is to develop implementation techniques for first-order ASP systems and apply it to GGP.

References

[Apt *et al.*, 1987] K. Apt, H. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, ed., *Found. of Deductive Databases and Log. Prog.*, 89–148, 1987.

[Clune, 2007] J. Clune. Heuristic evaluation functions for general game playing. In *Proc. of AAAI*, 1134–1139, 2007.

[Finnsson and Björnsson, 2008] H. Finnsson and Y. Björnsson. Simulation-based approach to general game playing. In *Proc. of AAAI*, 259–264, 2008.

[Gelfond, 2008] M. Gelfond. Answer sets. In *Handbook of Knowledge Representation*, 285–316. Elsevier, 2008.

[Genesereth *et al.*, 2005] M. Genesereth, N. Love, and B. Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62–72, 2005.

[Kuhlmann *et al.*, 2006] G. Kuhlmann, K. Dresner, and P. Stone. Automatic heuristic construction in a complete general game player. In *Proc. of AAAI*, 1457–1462, 2006.

[Lee and Meng, 2008] J. Lee and Y. Meng. On loop formulas with variables. In G. Brewka, P. Doherty, and J. Lang, editors, *Proc. of the Int.'l Conf. on Principles of Knowledge Representation and Reasoning*, 444–453, 2008.

[Lloyd and Topor, 1986] J. Lloyd and R. Topor. A basis for deductive database systems II. *J. of Logic Programming*, 3(1):55–67, 1986.

[Lloyd, 1987] J. Lloyd. *Foundations of Logic Programming*. Springer, 2nd edition, 1987.

[Love *et al.*, 2006] N. Love, T. Hinrichs, D. Haley, E. Schkufza, and M. Genesereth. General Game Playing: Game Description Language Specification. Technical Report LG–2006–01, Stanford University, 2006. Available at: `games.stanford.edu`.

[Niemelä *et al.*, 1999] I. Niemelä, P. Simons, and T. Soininen. Stable model semantics of weight constraint rules. In *Proc. of the Int.'l Conf. on Log. Prog. and Nonmonotonic Reasoning*, vol. 1730 of *LNCS*, 317–331, 1999. Springer.

[Pell, 1993] B. Pell. *Strategy Generation and Evaluation for Meta-Game Playing*. PhD thesis, Trinity College, University of Cambridge, 1993.

[Potassco, 2008] Potsdam Answer Set Solving Collection, 2008. `http://potassco.sourceforge.net/`.

[Schiffel and Thielscher, 2007] S. Schiffel and M. Thielscher. Fluxplayer: A successful general game player. In *Proc. of AAAI*, 1191–1196, 2007.

[Schiffel and Thielscher, 2009] S. Schiffel and M. Thielscher. A multiagent semantics for the game description language. In *Proc. of the Int.'l Conf. on Agents and Artificial Intelligence*, Porto 2009. Springer LNCS.

[van der Hoek *et al.*, 2007] W. van der Hoek, J. Ruan, and M. Wooldridge. Strategy logics and the game description language. In *Proc. of the Workshop on Logic, Rationality and Interaction*, Bejing 2007.