Pushing the Envelope: General Game Players Prove Theorems

Sebastian Haufe [1] and Michael Thielscher [2]

[1] Department of Computer Science
Dresden University of Technology
sebastian.haufe@mailbox.tu-dresden.de
[2] School of Computer Science and Engineering
The University of New South Wales
mit@cse.unsw.edu.au

Abstract. A *general* game player is a system that can play previously unknown games given nothing but their rules. A key to success in this endeavour is the ability to automatically gain knowledge about new games that follows from the rules without being explicitly given. In this paper, we show how a recently developed, theoretical method for automated theorem proving in general game playing can be put into practice. To this end, we extend the method so as to allow a general game player to systematically search and verify multiple temporal game properties at once. We formally prove this extension to be correct, and we report on extensive experiments that show how this improvement helps to significantly enhance the ability of a successful general game player to infer new properties about a previously unknown game.

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 AI, this endeavour requires to combine methods from a variety of sub-disciplines, including automated reasoning, search, game playing, and learning [9, 5, 3, 11, 2].

A key capability is to automatically gain knowledge about games that follows from the rules without being explicitly given. In [12, 13] we have laid the foundations for the use of Answer Set Programming [4] to automatically prove properties of a game from its mere rules. While initial experiments had shown that this provides a viable method for a general game player to establish the truth of a specific property, the practice of general game playing requires a player to systematically search large sets of potentially valid and useful properties in order to find those that actually hold [5, 3, 11]. Proving each candidate formula individually constitutes a considerable computational burden [13].

In this paper, we extend the method in [13] so as to allow a general game player to systematically search through and verify multiple temporal game properties at once. The correctness of our extended approach is formally proved, and

we report on extensive experiments that show how our improvement significantly enhances the ability of a successful general game player to infer new properties about a previously unknown game.

2 Background

2.1 Game Description Language

The Game Description Language (GDL) has been developed to formalise the rules of any finite $n \geq 1$-player 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 can give just a very brief introduction to GDL and have to refer to [7] for details.

GDL is based on the standard syntax of logic programs, including negation. We assume familiarity with the basic notions of logic programming. 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:

`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`

A further standard predicate is `distinct(X,Y)`, which means syntactic inequality of the two arguments. GDL imposes restrictions on the use of these keywords:

- `role` only appears in facts (i.e., clauses with empty body);
- `init` and `next` only appear as head of clauses, and `init` does not depend on any of `true`, `legal`, `does`, `next`, `terminal`, or `goal`;
- `true` and `does` only appear in clause bodies with `does` not depending on any of `legal`, `terminal`, or `goal`.

Additional general restrictions are placed on a set of rules with the intention to ensure finiteness of the set of derivable predicate instances. Specifically, the set of rules must be *stratified* [1] and *allowed* [6]. Stratified logic programs are known to admit a unique *standard model* [1]. As an example, Figure 1 shows an excerpt of a GDL description for a game called "Quarto."

Based on the concept of the standard model, a GDL description can be understood as a state transition system as follows [7]. 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.

```
role(r1).  role(r2).    init(cell(1,1,b)).  ...  init(cell(4,4,b)).
init(sctrl(r1)).        init(pool(p0000)).  ...  init(pool(p1111)).

legal(R,select(P))    :- true(sctrl(R)),true(pool(P)).
legal(R,place(P,M,N)) :- true(pctrl(R)),true(slctd(P)),true(cell(M,N,b)).
legal(R,noop)         :- role(R),not true(sctrl(R)),not true(pctrl(R)).

next(pool(P))     :- true(pool(P)),not does(r1,select(P)),
                                   not does(r2,select(P)).
next(slctd(P))    :- does(R,select(P)).
next(cell(M,N,P)) :- does(R,place(P,M,N)).
next(cell(M,N,P)) :- true(cell(M,N,P)),does(R,select(P)).
next(cell(M,N,P)) :- true(cell(M,N,P)),does(R,place(P,S,T)),!=(M,N,S,T).

next(sctrl(R))  :- true(pctrl(R)).
next(pctrl(r1)) :- true(sctrl(r2)).
next(pctrl(r2)) :- true(sctrl(r1)).
```

Fig. 1. A GDL description of "Quarto" (without definitions for termination, goal values, and `!=/4`). Two players take turns selecting one of 16 jointly used 4-attributed pieces $p0000, p0001, \ldots, p1111$ which the other player must place on a 4×4 board. The player wins who completes a line of 4 pieces with a common attribute.

The players R and the initial position of a game can be directly determined from the clauses for `role` and `init`, respectively. In order to determine the legal moves, update, termination, and outcome (i.e., goal values) for a 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:

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

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

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

Definition 1. *Let G be a GDL specification whose signature determines ground terms Σ. The* semantics *of G is the state transition system $(R, S_{init}, T, l, u, g)$ where*[3]

- $R = \{r : G \models \texttt{role}(r)\}$ *(the players);*
- $S_{init} = \{f : G \models \texttt{init}(f)\}$ *(the initial position);*
- $T = \{S : G \cup S^{\texttt{true}} \models \texttt{terminal}\}$ *(the terminal positions);*
- $l = \{(r, a, S) : G \cup S^{\texttt{true}} \models \texttt{legal}(r, a)\}$ *(the legality relation);*

[3] Below, entailment $\models$ is via the aforementioned standard model for stratified clause sets.

– $u(A,S) = \{f : G \cup S^{\texttt{true}} \cup A^{\texttt{does}} \models \texttt{next}(f)\}$ *(the update function);*
– $g = \{(r,v,S) : G \cup S^{\texttt{true}} \models \texttt{goal}(r,v)\}$ *(the goal relation).*

We write $S \xrightarrow{A} S'$ *if* $A : (R \mapsto \Sigma)$ *is such that* $(r, A(r), S) \in l$ *for each* $r \in R$ *and* $S' = u(A,S)$ *(and* $S \notin T$*). We call* $S_0 \xrightarrow{A_0} S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{m-1}} S_m$ *(where* $m \geq 0$*) a* sequence *(of legal moves), sometimes abbreviated as* $(S_0, S_1, \ldots, S_m)$*. A state* S *is called* reachable *iff there is a sequence which starts in the initial state* S_{init} *and ends in* S*.*

This definition provides a formal semantics by which a GDL description is interpreted as an abstract k-player game: in every position S, starting with S_{init}, each player r chooses a move a that is legal, i.e., 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 w.r.t. 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 [7].

2.2 Formalising and Encoding Temporal Game Properties

Next, we briefly summarise syntax and semantics of a language for formulating individual game properties. We also recapitulate from [13] the so-called *temporal GDL extension*, which is needed for proving properties given in this language.

Definition 2. *The set of* formulas *is (1) based on all ground atoms over the signature of a GDL description which are different from* `init` *and* `next` *and not dependent on* `does`*, and (2) closed under* $\neg, \wedge, \vee, \supset, \bigcirc$*. The* degree *of a formula* φ *is the maximal "nesting" of the unary* $\bigcirc$*-operator in* φ*.*

Modality $\bigcirc\varphi$ states that φ holds in all positions that are a direct, legal successor of the current game state. An example property in the Quarto game is the periodic return of "select control" to player *r1* every four moves, which can be formulated via the formula $true(sctrl(r1)) \supset \bigcirc^4 true(sctrl(r1))$ with degree 4.

A formula with degree n follows from a GDL description if it holds w.r.t. all sequences of length n and all shorter sequences that end in a terminal state [13].

Definition 3. *A sequence is called* n-max *iff it is of length* n*, or shorter and ending in a terminal state. Let* G *be a GDL description and* φ *a formula with degree* n*. We say that* S_0 satisfies φ *(written* $S_0 \models_t \varphi$*) if for all* n*-max sequences* $(S_0, \ldots, S_m)$ $(m \leq n)$ *we have that* $(S_0, \ldots, S_m) \models_t \varphi$ *according to the following definition:*

$(S_i, \ldots, S_m) \models_t p$ *iff* $G \cup S_i^{\texttt{true}} \models p$ *(*p *ground atom)*
$(S_i, \ldots, S_m) \models_t \neg\varphi$ *iff* $(S_i, \ldots, S_m) \not\models_t \varphi$ *(likewise for* $\wedge, \vee, \supset$*)*
$(S_i, \ldots, S_m) \models_t \bigcirc\varphi$ *iff* $i = m$ *or* $(S_{i+1}, \ldots, S_m) \models_t \varphi$

Automatically verifying properties over sequences of successive game states against a given GDL specification G requires to build the *temporal extension*

of G (with some horizon n), denoted G_n. It is obtained by joining timed variants of G (which enrich predicates with a time argument) for each time level $0 \leq i \leq n$, omitting `does`-dependent rules for level n. We refer to [13] for a formal definition and just give an example: Consider the fourth rule with head `next` in the GDL description G of Figure 1. It depends on `does`, hence the following timed variant is contained in G_n for every $0 \leq i \leq n-1$:

$$\texttt{true(cell(M,N,P),}i+1\texttt{)} \texttt{ :- true(cell(M,N,P),}i\texttt{),}\ \texttt{does(R,select(P),}i\texttt{).}$$

The definitions of $S^{\texttt{true}}$ and $A^{\texttt{does}}$ (cf. Section 2.1) are similarly extended to $S^{\texttt{true}}(0)$ and $A^{\texttt{does}}(i)$, respectively. The encoding of a formula φ can now be related to a temporally extended GDL description G_n in a way that corresponds to formula entailment w.r.t. G.

Definition 4. *Let $\eta(\varphi)$ be a 0-ary atom which represents a unique name for formula φ with degree n. An* encoding of φ, *denoted $Enc(\varphi)$, is a stratified set of rules whose heads include $\eta(\varphi)$ and do not occur elsewhere, and such that for all n-max sequences $S_0 \stackrel{A_0}{\rightarrow} S_1 \ldots \stackrel{A_{m-1}}{\rightarrow} S_m$ of a GDL description G:*

$$(S_0, \ldots, S_m) \vDash_t \varphi \quad \textit{iff} \quad S_0^{\texttt{true}}(0) \cup G_n \cup \bigcup_{i=0}^{m-1} A_i^{\texttt{does}}(i) \cup Enc(\varphi) \vDash \eta(\varphi)$$

In the following we assume *Enc* to be given, whose construction can be easily automated. Recall, e.g., $\varphi = true(sctrl(r1)) \supset \bigcirc^4 true(sctrl(r1))$ from above and let $\eta(\varphi) = \texttt{a}$, then the following set of rules encode φ:

```
a :- not true(sctrl(r1),0).  a :- terminal(0).  a :- terminal(1).
a :-     true(sctrl(r1),4).  a :- terminal(2).  a :- terminal(3).
```

3 Proving Multiple Temporal Game Properties At Once

In [13] we have shown how the encoding of a game property (i.e., a temporal formula), together with the temporal extension of a given set of game rules, can be fed into a system for Answer Set Programming (ASP) in order to establish whether the rules entail the property.[4] Even though being the currently fastest approach for calculating models of logic programs, requiring a general game player to evoke an ASP system individually for each formula in a large set of candidate properties is not feasible for the practice of general game playing with a limited amount of time to analyse the rules of a hitherto unknown game.

In the following we therefore develop a crucial extension of our method that enables a general game player to evoke an ASP system only once in order to determine precisely which of a whole set Φ of formulas is valid w.r.t. a given game description. For this purpose, we construct two answer set programs for

[4] *Answer sets* are specific models of logic programs with negation; see e.g. [4].

Φ, one to establish base case proofs and one for the induction steps. For any $\varphi \in \Phi$, then, if all answer sets for the base case program satisfy φ, then φ is entailed in the initial state. If additionally all answer sets of the induction step program satisfy $\varphi \supset \bigcirc\varphi$, we can conclude that φ is entailed in all reachable states. The encoding of each player performing a legal move in each nonterminal state is given by a set of ASP clauses P_n^{legal}, consisting of a set of negation-free clauses which defines the domains of actions (`adom`) and the following clauses for each $0 \leq i \leq n$:[5]

(c_1) `terminated(`i`) :- terminal(`i`).`
(c_2) `terminated(`i`) :- terminated(`$i-1$`).`
(c_3) `1{does(R,A,`i`):adom(R,A)}1 :- role(R), not terminated(`i`).`
(c_4) `:- does(R,A,`i`), not legal(R,A,`i`).`

For a GDL description G and a finite set of formulas Φ with maximal degree $\widehat{n}$, the answer set program for the *base case* is defined as follows:

$$P_\Phi^{bc}(G) = S_{init}^{\texttt{true}}(0) \cup G_{\widehat{n}} \cup P_{\widehat{n}-1}^{legal} \cup \bigcup_{\varphi \in \Phi} Enc(\varphi)$$

Put in words, $P_\Phi^{bc}(G)$ consists of an encoding for the initial state, $S_{init}^{\texttt{true}}(0)$; a temporal GDL description up to time step $\widehat{n}$, $G_{\widehat{n}}$; the necessary requirements concerning legal moves, $P_{\widehat{n}-1}^{legal}$; and an encoding for each of the formulas in Φ, $\bigcup_{\varphi \in \Phi} Enc(\varphi)$. Encoding $Enc(\varphi)$ ensures that if $\eta(\varphi)$ occurs in each answer set for $P_\Phi^{bc}(G)$, then every state sequence starting at S_{init} makes φ true—which means that $S_{init} \vDash_t \varphi$.

For the *induction step* answer set program, instead of the state encoding $S_{init}^{\texttt{true}}(0)$ we need a "state generator" program whose answer sets correspond exactly to the reachable states of a GDL description. These, however, cannot be calculated efficiently in most cases, motivating an easily obtainable approximation which comprises some non-reachable states as well. The simplest approximation is the program `0{true(F,0) : fdom(F)}.`, which, together with stratified clauses defining the domain of features (`fdom`), generates *all* states. Assuming a (probably more informed) state generator S^{gen}, the *induction step* answer set program is

$$P_\Phi^{is}(G) = S^{gen} \cup G_{\widehat{n}+1} \cup P_{\widehat{n}}^{legal} \cup \bigcup_{\varphi \in \Phi} Enc(\varphi \supset \bigcirc\varphi)$$

Besides the state generator instead of the initial state, $P_\Phi^{is}(G)$ deviates from $P_\Phi^{bc}(G)$ in that the maximal time step $\widehat{n}$ is increased by one. Moreover, encoding $Enc(\varphi \supset \bigcirc\varphi)$ ensures that if $\eta(\varphi \supset \bigcirc\varphi)$ occurs in each answer set for $P_\Phi^{is}(G)$, then φ is entailed by each direct successor of a state that itself entails φ.

[5] In the following we use two common additions to ASP [8]: a *weight atom* $m\ \{\ p : d(\overline{x})\ \}\ n$ means that for atom p an answer set has at least m and at most n different instances that satisfy $d(\overline{x})$. If n is omitted, there is no upper bound. A *constraint* is a rule `:-` $b_1, \ldots, b_k$, which excludes any answer set that satisfies $b_1, \ldots, b_k$.

4 Expressiveness and Correctness of the Proof Method

We will now show that our generalisation is correct and that it is at least as strong as the original Temporal Proof System [13]. We require two results, the first of which concerns sequences that are longer than the degree of the formula to be proved. The result refers to the standard restriction to *playable* GDL games, meaning that every role has at least one legal move in every non-terminal reachable state [7].

Lemma 1. *Let φ be a formula with degree n and G be a GDL description, then for all $\widehat{n} \geq n$:*

(A) Every $\widehat{n}$-max sequence which does not satisfy φ can be reduced to an n-max sequence which does not satisfy φ.

(B) Let G be playable and S reachable. Then every n-max sequence starting in S which does not satisfy φ can be extended to an $\widehat{n}$-max sequence starting in S which does not satisfy φ.

Note that item (B) is not true for non-reachable states S_0. Consider, e.g., formula $\varphi = \texttt{true(f)}$ in a single-player game where `f` is true initially, where the only action `a` is legal if `f` holds, and where `a` makes `f` true in the direct successor state. Assume the (non-reachable) empty state $\{\}$ to be non-terminal. Then sequence $(\{\})$ of length 0 does not satisfy φ but cannot be extended to any 1-max sequence, as the only action `a` is not legal in $\{\}$.

Our second lemma relates answer set programs to sequence-encoding stratified programs (which in turn relate to formula entailment via Definition 4).

Lemma 2. *For a GDL description G, let $P = S_0^{\texttt{true}}(0) \cup G_n \cup P_{n-1}^{legal}$. Then P has an answer set $\mathcal{A}_n$ iff there is an n-max sequence $(S_0 \stackrel{A_0}{\rightarrow} \ldots \stackrel{A_{m-1}}{\rightarrow} S_m)$ such that replacement of all rules of the form (c_3) and (c_4) in P (occurring in P_{n-1}^{legal}) with $\bigcup_{i=0}^{m-1} A_i^{\texttt{does}}(i)$ yields a program with unique standard model $\mathcal{A}_n$.*

Correctness can now be established as follows.

Theorem 1. *Let $\varphi \in \Phi$ and G be a playable GDL description with initial state S_{init}. If every answer set for $P_{\Phi}^{bc}(G)$ contains $\eta(\varphi)$ and every answer set for $P_{\Phi}^{is}(G)$ contains $\eta(\varphi \supset \bigcirc\varphi)$, then for all finite sequences $S_{init} \stackrel{A_0}{\rightarrow} S_1 \ldots \stackrel{A_{k-1}}{\rightarrow} S_k$ we have $S_k \vDash_t \varphi$.*

Proof. (Sketch) Induction on k, using Lemma 1 (B) and Lemma 2. Base case $k = 0$: $S_{init} \nvDash_t \varphi$ implies the existence of an answer set for $P_{\Phi}^{bc}(G)$ that does not contain $\eta(\varphi)$. Induction step: $S_k \vDash_t \varphi$, $S_k \stackrel{A_k}{\rightarrow} S_{k+1}$, and $S_{k+1} \nvDash_t \varphi$ imply the existence of an answer set for $P_{\Phi}^{is}(G)$ which does not contain $\eta(\varphi \supset \bigcirc\varphi)$.

To show that our proof method is a generalisation of the original approach, we need to restate the programs $P_{\varphi}^{bc}(G)$ and $P_{\varphi}^{is}(G)$ [13], where φ has degree n:

$$P_{\varphi}^{bc}(G) = S_{init}^{\texttt{true}}(0) \cup G_n \cup P_{n-1}^{legal} \cup Enc(\varphi) \cup \{\texttt{:-}\ \eta(\varphi)\texttt{.}\}$$

$$P_{\varphi}^{is}(G) = S^{gen} \cup G_{n+1} \cup P_n^{legal} \cup Enc(\varphi) \cup Enc(\bigcirc\varphi) \cup \{\texttt{:- not}\ \eta(\varphi)\texttt{.}, \texttt{:-}\ \eta(\bigcirc\varphi)\texttt{.}\}$$

The main difference is the reduced maximal time level $n \leq \widehat{n}$. Moreover the encoding for φ in $P^{bc}_{\varphi}(G)$ is constrained such as to only allow answer sets that represent φ-violating sequences. Similarly, answer sets for $P^{is}_{\varphi}(G)$ represent sequences $(S_0, \ldots, S_m)$ where φ holds in S_0 but not in S_1. Both $P^{bc}_{\varphi}(G)$ and $P^{is}_{\varphi}(G)$ being inconsistent yields $S \models_t \varphi$ for all reachable states.

Theorem 2. *Let $\varphi \in \Phi$ and G be a GDL description.*

- *If $P^{bc}_{\varphi}(G)$ is inconsistent then $\eta(\varphi)$ is in all answer sets of $P^{bc}_{\Phi}(G)$.*
- *If $P^{is}_{\varphi}(G)$ is inconsistent then $\eta(\varphi \supset \bigcirc\varphi)$ is in all answer sets of $P^{is}_{\Phi}(G)$.*

Proof. (Sketch) If there is an answer set for $P^{bc}_{\Phi}(G)$ ($P^{is}_{\Phi}(G)$) that does not contain $\eta(\varphi)$ ($\eta(\varphi \supset \bigcirc\varphi)$) then program transformations using Lemma 1 (A) and Lemma 2 imply that there is an answer set for $P^{bc}_{\varphi}(G)$ ($P^{is}_{\varphi}(G)$).

It should be stressed that the converse of Theorem 2, however, does not hold: An answer set for $P^{is}_{\varphi}(G)$ represents an established n-max sequence *Seq* (cf. Lemma 2) which violates $\varphi \supset \bigcirc\varphi$. *Seq* however might not be extendable to an $\widehat{n}$-max sequence (cf. the remark following Lemma 1 (B)) which could serve as counter example for $\varphi \supset \bigcirc\varphi$ in $P^{is}_{\Phi}(G)$. Hence our generalisation strengthens the result, depending on the maximal degree $\widehat{n}$ of the given formula set Φ.

5 Experimental Results

We have implemented our proof method using Fluxplayer [11] for the generation of the ASP program, which is then processed by grounder Bingo and ASP solver Clasp from a state-of-the-art answer set solving collection [10]. We use option "cautious reasoning" for Clasp to compute the intersection of all answer sets. In the following we sketch the formula sets we had the player try to prove. The resulting proof times for a variety of games can be seen in Figure 2.

- *Persistence (Φ^p)*: Ground features $f(\bar{t})$ which stay true [false] once they become true [false] are proved using the set Φ^p of *all* formulas of the form $[\neg]true(f(\bar{t})) \supset \bigcirc[\neg]true(f(\bar{t}))$. In the game Quarto, say, $\neg true(pool(X)) \supset \bigcirc\neg true(pool(X))$ can be proved for all instances $X \in \{p0000, \ldots, p1111\}$, stating that once a piece is not available for selection anymore, it will not be available throughout the remainder of the game.
- *Existence (Φ^{ex})*: We prove [non]existence of ground instances for each feature f_i/k_i and its interaction with ground instance existence of different features f_j/k_j (fixing $i < j$ in an arbitrary total feature order). The set Φ^{ex} of existential formulas contains all formulas of the form $[\neg]\varphi_{f_i/k_i}$ and all formulas of the form $\varphi_{f_i/k_i} \vee [\neg]\varphi_{f_j/k_j}$ and $\neg\varphi_{f_i/k_i} \vee [\neg]\varphi_{f_j/k_j}$, where $\varphi_{f/k} := \bigvee_{\bar{t} \in (D_1 \times \ldots \times D_k)} true(f(\bar{t}))$ (the *finite* sets $D_i \subseteq \Sigma$ being calculated automatically). For Quarto, the prover successfully shows $\neg\varphi_{slctd/1} \vee \varphi_{pctrl/1}$, hence a selected piece always implies a player to have place control. Formulas $\varphi_{pctrl/1} \vee \varphi_{sctrl/1}$ and $\neg\varphi_{pctrl/1} \vee \neg\varphi_{sctrl/1}$ prove mutual exclusion of the two control features and, together with $\neg\varphi_{pctrl/1} \vee \varphi_{cell/3}$ and $\neg\varphi_{sctrl/1} \vee \varphi_{cell/3}$, imply existence of a cell instance in each reachable state.

Game	Φ^p	Φ^{ex}	Φ^{ctrl}	$\Phi^p \cup \Phi^{ex} \cup \Phi^{ctrl}$
3pttc	0.78 (77/362)	0.45 (10/18)	0.39 (3/9)	1.55 (90/389)
bidding-tictactoe	0.18 (9/108)	0.31 (13/50)	0.23 (0/12)	0.51 (29/170)
breakthrough	1.02 (32/260)	0.78 (5/8)	1.17 (4/6)	1.69 (41/274)
capture_the_king	33.01 (7/1744)	9.65 (5/32)	29.98 (10/12)	85.05 (22/1788)
catcha_mouse	1.34 (359/998)	1.05 (8/18)	0.20 (4/6)	2.50 (371/1022)
checkers	50.47 (41/1098)	10.16 (13/32)	56.79 (4/6)	98.26 (58/1136)
chomp	0.09 (58/120)	0.14 (6/18)	0.12 (10/12)	0.20 (75/150)
connect4	0.30 (294/508)	0.32 (5/8)	0.19 (4/6)	0.73 (303/522)
endgame	453.48 (2/546)	4.54 (12/18)	33.21 (4/6)	520.80 (18/570)
knightfight	3.91 (0/608)	1.07 (2/18)	3.18 (4/12)	12.35 (6/638)
othello	3.89 (8/260)	1.41 (5/8)	4.00 (4/6)	10.34 (17/274)
pawn_whopping	0.45 (32/260)	0.20 (5/8)	0.22 (4/6)	0.74 (41/274)
quarto	38.74 (32/616)	34.48 (6/50)	33.19 (4/12)	147.02 (42/678)
tictactoe	0.09 (27/58)	0.10 (5/8)	0.13 (4/6)	0.14 (36/72)
tttcc4	15.66 (311/1244)	2.64 (7/18)	3.90 (3/9)	42.48 (321/1271)

Fig. 2. Property proof times, in seconds (average over 10 runs), for a variety of games taken from *www.general-game-playing.de*. Each time indicates one proof attempt (one ASP proof for the base case and one ASP proof for the induction step) of the respective formula set. The numbers in parantheses mean: (number of proved properties/size of the formula set). Experiments were run on an Intel Core 2 Duo CPU with 3.16 GHz.

– *Control* (Φ^{ctrl}): The periodic return of control features is proved via the set Φ^{ctrl} of all formulas $true(f(r)) \supset \bigcirc^n true(f(r))$, where r is a role and $2 \leq n \leq 4$. In Quarto we obtain successful proofs for $n = 4$ and $f(r) \in \{pctrl(white), pctrl(black), sctrl(white), sctrl(black)\}$, indicating the return of the same game phase every 4 steps.

In general, our timings for *Control* and *Persistence* are of the same order as the runtimes for games and property instances we obtained for the original method [13], since attempting proofs for all instances in one run spares the solver to repeat similar processes multiple times. This amounts to a significant speedup, which e.g. for *Persistence* means to check several hundred instances. Properties like *Existence* and *Persistence* together with initially true [false] features provide valuable information about reachable states, due to the fast timings their proofs qualify as basis for further state generator restriction, thus obtaining better timings and more accurate results for increasingly sophisticated properties. Joint proof attempts for multiple classes of properties (cf. column $\Phi^p \cup \Phi^{ex} \cup \Phi^{ctrl}$ in Figure 2), however, sometimes decrease performance (e.g. for *tttcc4*) due to less viable program rule optimisations, which suggests to divide properties in classes of "similar" form. Note that sometimes more formulas are proved (e.g. for *bidding-tictactoe*) with the joint approach thanks to the effect mentioned at the end of Section 4.

6 Summary

A key to success in general game playing is the ability to automatically infer properties of a new game that follow from the rules without being explicitly given. By extending a recently developed, basic approach to automated theorem proving for this purpose, we have developed a method that enables a general game player to systematically and simultaneously search large sets of candidate formulas in order to identify those whose validity can be established. We have formally proved the correctness of this extended method, and we have conducted systematic experiments with a variety of games that have been used by the scientific community in the past. As the experimental data show, our extended method allows to search through large sets of formulas of a similar form in times comparable to proving just a single one of these properties using the original method.

Acknowledgement. Michael Thielscher is the recipient of an Australian Research Council Future Fellowship (project number FT 0991348).

References

1. Apt, K., Blair, H. A., Walker, A.: Towards a Theory of Declarative Knowledge. In: Foundations of Deductive Databases and Logic Programming, 89–148 (1987)
2. Björnsson, Y., Finnsson, H.: CADIAPLAYER: A Simulation-Based General Game Player. IEEE Transactions on Computational Intelligence and AI in Games, 1(1):4–15 (2009)
3. Clune, J.: Heuristic Evaluation Functions for General Game Playing. In: AAAI, 1134–1139 (2007)
4. Gelfond, M.: Answer Sets. In: Handbook of Knowledge Representation, 285–316. Elsevier (2008)
5. Kuhlmann, G.; Dresner, K.; and Stone, P.: Automatic Heuristic Construction in a Complete General Game Player. In: AAAI, 1457–1462 (2006)
6. Lloyd, J., and Topor, R.: A Basis for Deductive database Systems II. J. of Logic Programming, 3(1):55–67 (1986)
7. Love, N.; Hinrichs, T.; Haley, D.; Schkufza, E.; and Genesereth, M.: General Game Playing: Game Description Language Specification. Technical Report, LG–2006–01, Stanford University. Available at *games.stanford.edu* (2006)
8. Niemelä, I.; Simons, P.; and Soininen, T.: Stable Model Semantics of Weight Constraint Rules. In: Proceedings of LPNMR, vol. 1730 of *LNCS*, 317–331 (1999)
9. Pell, B.: Strategy Generation and Evaluation for Meta-Game Playing. Ph.D., Cambridge (1993)
10. Potassco, Potsdam Answer Set Solving Collection. Available at *potassco.sourceforge.net* (2008)
11. Schiffel, S., and Thielscher, M.: Fluxplayer: A Successful General Game Player. In: AAAI, 1191–119 (2007)
12. Schiffel, S., and Thielscher, M.: Automated Theorem Proving for General Game Playing. In: IJCAI, 911–916 (2009)
13. Thielscher, M., Voigt, S.: A Temporal Proof System for General Game Playing. In: AAAI, 1000–1005 (2010)