Automated Verification of Epistemic Properties for General Game Playing

Sebastian Haufe
Department of Computer Science
Dresden University of Technology, Germany
sebastian.haufe@tu-dresden.de

Michael Thielscher
School of Computer Science and Engineering
The University of New South Wales, Australia
mit@cse.unsw.edu.au

Abstract

Automatically deriving properties of new games is one of the fundamental challenges for *general* game-playing systems, whose task is to learn to play any previously unknown game solely by being given the rules of that game. A recently developed method uses Answer Set Programming for verifying finitely-bounded temporal invariance properties against a given game description by structural induction. Addressing the new challenge posed by the recent extension of the general Game Description Language to include games with imperfect information and randomness, we extend this method to *epistemic* properties about games. We formally prove this extension to be correct, and we report on experiments that show its practical applicability.

1 Introduction

General Game Playing is concerned with the development of intelligent systems that learn to play previously unknown games well without human intervention. Identified as a Grand Challenge for Artificial Intelligence, this endeavour requires to combine methods from a variety of subdisciplines, including reasoning, search, computer game playing, and learning (Pell 1993; Genesereth, Love, and Pell 2005).

Automated Theorem Proving is among the fundamental techniques in General Game Playing, where it serves two purposes (Haufe, Schiffel, and Thielscher 2012): In the first place, it can be used for the formal verification of desired properties during the game design phase. Secondly, general game-playing systems can use theorem proving to acquire knowledge of a previously unknown game, which is a prerequisite for the automatic generation of both search heuristics and an evaluation function for non-terminal positions (Kuhlmann, Dresner, and Stone 2006; Clune 2007; Schiffel and Thielscher 2007). We have shown recently how Answer Set Programming (Gelfond 2008) can be used to assist in this endeavour by the automated verification of *state sequence invariants*, that is, properties that may concern two or more successive game states but do not require to analyse the entire game tree (Haufe, Schiffel, and Thielscher 2012).

The recent extension of the general Game Description Language to imperfect information and randomness (Thielscher 2010) poses new challenges to both designer and player of a general game. Fundamental issues such as the knowledge of a player about its legal moves, game termination, or its opponents' viewpoints are no longer obvious. The designer of a game must endow players with sufficient information to guarantee that the game is playable, and players themselves require reliable information about the game state to maximise their utility. Gaining this knowledge is an intricate reasoning task over the game rules, and its automation is of major assistance for the game designer and essential for a successful general game-playing program.

In this paper, we provide such an automated reasoning method for *epistemic* properties by fundamentally extending our proof method for state sequence invariants. To this end, we specify syntax and semantics of a formula language for the expression of epistemic game knowledge; we develop an inductive proof theory that allows to verify "positive" epistemic formulas against a given game specification without analysis of the entire game tree; we prove the correctness of our proof theory; and we demonstrate its efficiency by reporting on experiments with a practical implementation.

2 The Game Description Language GDL

GDL is based on the standard syntax and semantics of logic programming (Genesereth, Love, and Pell 2005). We follow the Prolog convention of denoting variables by uppercase letters while predicate and function symbols start with a lowercase letter. GDL is characterised by the following special *keywords*, two of which (**random**, **sees**) were recently added for games with randomness and imperfect information (Thielscher 2010).

role (R)	R is a player
random	the random player
init (F)	F holds in the initial position
distinct(X,Y)	X and Y are syntactically different
true (F)	F holds in the current position
legal (R,M)	R can do move M in the current position
does (R,M)	player R does move M
next (F)	F holds in the next position
sees (R,P)	R perceives P in the next position
terminal	the current position is terminal
goal (R,N)	R gets N points in the current position

```
1  role(x).  role(o).
2  init(control(x)).  init(cell(1,1,b)).  ...  init(cell(3,3,b)).
3
4  legal(R,mark(M,N))  :- true(control(R)), true(cell(M,N,Z)), not true(tried(M,N)).
5  legal(x,noop)       :- true(control(o)).
6  legal(o,noop)       :- true(control(x)).
7
8  validmove :- does(R,mark(M,N)), true(cell(M,N,b)).
9
10 next(tried(M,N))  :- not validmove, does(P,mark(M,N)).
11 next(F)           :- not validmove, true(F).
12 next(cell(M,N,x)) :-     validmove, does(x,mark(M,N)).
13 next(cell(M,N,o)) :-     validmove, does(o,mark(M,N)).
14 next(cell(M,N,Z)) :-     validmove, true(cell(M,N,Z)), does(P,mark(I,J)), distinct(M,I).
15 next(cell(M,N,Z)) :-     validmove, true(cell(M,N,Z)), does(P,mark(I,J)), distinct(N,J).
16 next(control(o))  :-     validmove, true(control(x)).
17 next(control(x))  :-     validmove, true(control(o)).
18
19 sees(R,yourmove) :- not validmove, true(control(R)).
20 sees(x,yourmove) :-     validmove, true(control(o)).
21 sees(o,yourmove) :-     validmove, true(control(x)).
```

Figure 1: A GDL description of "Krieg-Tictactoe" (omitting the clauses for termination and goal values). The game positions are encoded using the three features *control*(R), *cell*(M, N, Z), and *tried*(M, N), where $R \in \{x, o\}$; $M, N \in \{1, 2, 3\}$; and $Z \in \{x, o, b\}$, with b meaning "blank."

As an example consider Fig. 1, which shows a GDL description of standard Tictactoe with the additional interesting twist that the players cannot see their opponent's moves (Schiffel and Thielscher 2011). The names of the players and the initial position are given in lines 1–2. The moves are specified by the rules with head **legal** (lines 4–6): The player whose turn it is can attempt to mark any cell that it has not tried before (line 4). The other player can only do *noop*, a move without effect (lines 5–6). This is the usual way of modelling turn-taking games in GDL, whose semantics and execution model assume all players to move simultaneously.

The position update is specified by the rules with head **next** (lines 10–17): If the submitted move *mark*(m, n) is invalid (line 8), then a special token *tried*(m, n) becomes true[1] (line 10), and every feature that is true in the current position continues to hold (line 11). If, on the other hand, the move is valid, then cell (m, n) receives the player's mark (lines 12–13) while all other cells retain their contents (lines 14–15). Moreover, move control goes to the other player (lines 16–17). The reader should also note that all instances of *tried*(m, n) cease to hold as there is no clause with head **next**(*tried*(m, n)) if *validmove* is true.

The clauses with head **sees** (lines 19–21), finally, say that the player whose turn it is will be informed about this (which is sufficient for a player to always know which squares contain its own marks). We have omitted the clauses for termination (a game state is terminal if there is a line of three equal markers or the board is completely filled) and goal values.

[1] It is important to note the difference between *legal* and *valid* moves in Krieg-Tictactoe: each attempt to mark a cell is considered legal, but only those moves that are actually possible in the current position are accepted as valid. Feature *tried*(m, n) is used to prevent a player from resubmitting a previously rejected move.

2.1 Formal Syntax and Semantics

In order to admit an unambiguous interpretation, GDL game descriptions must obey certain general syntactic restrictions. Specifically, a valid game description must be *stratified* (Apt, Blair, and Walker 1987) and *allowed* (Lloyd and Topor 1986). A further syntactic restriction ensures that only finitely many positive instances are true in this model; for details we must refer to Love et al. (2006) for space reasons. Finally, the special keywords are to be used as follows (Thielscher 2010):

- **role** only appears in facts (i.e., clauses with empty body) or in the body of clauses;
- **init** only appears as head of clauses and does not depend on any of **true**, **legal**, **does**, **next**, **sees**, **terminal**, **goal**;
- **true** only appears in the body of clauses;
- **does** only appears in the body of clauses, and none of **legal**, **terminal**, or **goal** depends on **does**;
- **next** and **sees** only appear as head of clauses.

These restrictions are imposed to ensure that a set of GDL rules can be effectively and unambiguously interpreted by a state transition system as follows. To begin with, any set of clauses determines an implicit domain-dependent set of ground symbolic expressions Σ, like *cell*$(1, 1, b)$, *mark*$(1, 3)$, *yourmove*, etc. Game positions (i.e., states) are represented by subsets of Σ since they are composed of individual features. Although Σ itself is usually infinite, the syntactic restrictions in GDL ensure that the set of roles, the

reachable states, and the set of legal moves are always finite subsets of Σ (Love et al. 2006).

In order to determine the legal moves of a player in any given state, this state is encoded using the keyword **true**. More precisely, let $S = \{f_1, \ldots, f_k\}$ be a finite state (e.g., the derivable instances of **init** (F) at the beginning), then game description G is extended by the k facts

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

Those instances of **legal** (R,M) that are derivable from $G \cup S^{\mathtt{true}}$ define all legal moves M for player R in position S. In addition to the encoding of the current position, determining a position update and the percepts of the players requires a *joint action*, which consists of an action for each of the players. Specifically, if joint action A is such that players $r_1, \ldots, r_k$ take moves $a_1, \ldots, a_k$, then let

$$A^{\mathtt{does}} \stackrel{\text{def}}{=} \{\, \mathbf{does}(r_1, a_1). \quad \ldots \quad \mathbf{does}(r_k, a_k).\,\}.$$

The instances of **next** (F) that are derivable from $G \cup A^{\mathtt{does}} \cup S^{\mathtt{true}}$ compose the updated position; likewise, the derivable instances of **sees** (R,P) describe what a player perceives when the given joint action is done in the given position. All this is summarised in the following definition.[2]

Definition 1 *(Thielscher 2010) Let G be a GDL specification whose signature determines the set of ground terms Σ. Let "$\vdash$" denote entailment by the* standard model *(Apt, Blair, and Walker 1987) of a stratified program, then the* semantics *of G is the state transition system $(R, S_{init}, T, l, u, \mathcal{I}, g)$ given by*

- $R = \{r : G \vdash \mathbf{role}(r)\}$ *(player names);*
- $S_{init} = \{f : G \vdash \mathbf{init}(f)\}$ *(initial state);*
- $T = \{S : G \cup S^{\mathtt{true}} \vdash \mathbf{terminal}\}$ *(terminal states);*
- $l = \{(r, a, S) : G \cup S^{\mathtt{true}} \vdash \mathbf{legal}(r, a)\}$ *(legal moves);*
- $u(A, S) = \{f : G \cup A^{\mathtt{does}} \cup S^{\mathtt{true}} \vdash \mathbf{next}(f)\}$*;*
- $\mathcal{I} = \{(r, A, S, p) : G \cup A^{\mathtt{does}} \cup S^{\mathtt{true}} \vdash \mathbf{sees}(r, p)\}$ *(players' percepts);*
- $g = \{(r, n, S) : G \cup S^{\mathtt{true}} \vdash \mathbf{goal}(r, n)\}$ *(goal values);*

for all finite subsets $S \subseteq \Sigma$ (called states) and functions $A : R \to \Sigma$ (called joint actions).

For states S, S' and a joint action A, we write $S \xrightarrow{A} S'$ if $(r, A(r), S) \in l$ for each role r; $S' = u(A, S)$; and $S \notin T$. We call $\sigma = (S_0 \xrightarrow{A_0} S_1 \ldots \xrightarrow{A_{m-1}} S_m)$ (where $m \geq 0$) a sequence *(of legal moves), sometimes abbreviated as $(S_0, S_1, \ldots, S_m)$. When $S_0 = S_{init}$, we also call σ a* development*, and we call a state S* reachable *if there is a development that ends in S.*

We will denote sequences with σ and developments with δ, possibly with super- or subscripts. For two sequences $\sigma_1 = (S_0, \ldots, S_m)$ and $\sigma_2 = (S_m, \ldots, S_{m+k})$, we also write (σ_1, σ_2) to denote their composition $(S_0, \ldots, S_m, \ldots, S_{m+k})$. The length of a given sequence $\sigma = (S_0, \ldots, S_m)$ is m, sometimes denoted by $|\sigma|$, and the last state S_m of σ is also referred to via the notion $last(\sigma)$.

[2] We omit the definition of the probability distribution over updated positions in case **random** $\in R$, as this is irrelevant for the present paper.

2.2 Indistinguishable Developments

We conclude this introduction by recalling what the execution model for GDL entails about what players know at any stage during a game (Thielscher 2010). This is formalised with the help of a binary equivalence relation $\sim_r$ over developments such that $\delta_1 \sim_r \delta_2$ if, and only if, δ_1 and δ_2 are indistinguishable for player r.

Proposition 1 Let G be a GDL description with semantics $(R, S_{init}, T, l, u, \mathcal{I}, g)$. For two developments of G,

$$\begin{aligned} \delta_1 &= (S_{init} \xrightarrow{A_0} S_1 \ldots \xrightarrow{A_{m-1}} S_m) \\ \delta_2 &= (S_{init} \xrightarrow{A'_0} S'_1 \ldots \xrightarrow{A'_{m-1}} S'_m) \end{aligned}$$

we have $\delta_1 \sim_r \delta_2$ iff for each $0 \leq i \leq m-1$:

- $\{p : (r, A_i, S_i, p) \in \mathcal{I}\} = \{p : (r, A'_i, S'_i, p) \in \mathcal{I}\}$ (that is, r's percepts are the same), and
- $A_i(r) = A'_i(r)$ (i.e., r always takes the same action). □

The proposition implies that each player r is aware of the initial game state, which results from the fact that r gets to know the complete game description prior to game play.

3 Formalising Epistemic Game Knowledge

We will now specify a language which allows to formulate properties that involve finitely many successive game states and the knowledge of different agents. It extends our previous language for game-specific properties (Haufe, Schiffel, and Thielscher 2012) by a unary knowledge operator.

3.1 Syntax

Definition 2 *Let G be a GDL specification. The set of* (epistemic) formulas (over G) *is given by the following Backus-Naur form:*

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \bigcirc\varphi \mid K_r\varphi,$$

where p is a ground atom over the signature of G which is different from **init** *and* **next** *and does not depend on* **does***, and r is any role of the game beside* **random***.*

Moreover, the degree *of a formula φ is the maximal "nesting" of the unary $\bigcirc$-operator in φ.*

Formulas that do not contain any knowledge operators K_r are also called *knowledge-free*. A formula $\bigcirc\psi$ states that ψ is true in the next game state, and a formula $K_r\psi$ states that agent r knows ψ. For example, in Krieg-Tictactoe the property "if the game has terminated, then the x-player knows that it has terminated" can be formulated via the following epistemic formula ρ:

$$\neg \mathit{terminal} \vee K_x \mathit{terminal}$$

3.2 Semantics

The following extends our existing semantics for knowledge-free formulas (Haufe, Schiffel, and Thielscher 2012), and we refer to this earlier work for a motivation to the non-standard treatment of state sequences and the temporal operator $\bigcirc$. The treatment of knowledge operators is motivated by a recently developed semantics for epistemic formulas over the GDL (Ruan and Thielscher 2011). Our semantics

employs the notion of an n-*max sequence*, which means a sequence that is either of length n, or shorter and ends in a terminal state.

Definition 3 *Let G be a GDL description, φ a formula over G with degree n, and $\delta = (S_{init}, S_1, \ldots, S_k)$ a development. We say that S_k* satisfies *φ wrt. δ (written $S_k \models_\delta \varphi$) if for all n-max sequences $\sigma = (S_k, \ldots, S_{k+m})$ ($m \leq n$) we have that $\sigma \models_\delta \varphi$ according to the following definition:*

$$
\begin{array}{lll}
\sigma \models_\delta p & \textit{iff} & G \cup S_k^{\texttt{true}} \vdash p \quad \textit{(p ground atom)} \\
\sigma \models_\delta \neg\varphi & \textit{iff} & \sigma \not\models_\delta \varphi \quad \textit{(likewise for } \wedge, \vee\textit{)} \\
\sigma \models_\delta \bigcirc\varphi & \textit{iff} & m = 0\textit{, or } \sigma' \models_{\delta'} \varphi \\
 & & \textit{for } \delta' = (S_{init}, S_1, \ldots, S_k, S_{k+1}) \\
 & & \textit{and } \sigma' = (S_{k+1}, \ldots, S_{k+m}) \\
\sigma \models_\delta K_r\varphi & \textit{iff} & last(\delta') \models_{\delta'} \varphi \textit{ for each } \delta' \textit{ s.t. } \delta \sim_r \delta'
\end{array}
$$

Since $\sim_r$ is an equivalence relation, our semantics can be proved to satisfy the S5 properties (Fagin et al. 1995). Furthermore, since a GDL specification contains a complete description of the initial state of the game, and since each player gets to know the complete specification prior to playing the game, each player knows all that is implied in the initial game state.

Proposition 2 Let φ be an epistemic formula, let $kf_0(\varphi)$ be the formula obtained from φ by removing each occurrence of a knowledge operator which is not in the scope of any $\bigcirc$, and let development δ be such that $\delta = (S_{init})$. Then $S_{init} \models_\delta \varphi$ iff $S_{init} \models_\delta kf_0(\varphi)$. □

4 Proof by Contradiction Using Linear Time

In our earlier work we have established an induction method to reliably show the validity[3] of arbitrary knowledge-free formulas φ with respect to a given game description (Haufe, Schiffel, and Thielscher 2012). The base case proof shows that $S_{init} \models_\delta \varphi$ holds for the initial development $\delta = (S_{init})$. The induction step shows that any development $\delta = (S_{init}, S_1, \ldots, S_k)$ s.t. $S_k \models_\delta \varphi$ implies $S_{k+1} \models_{\delta'} \varphi$ for each of its prolongations $\delta' = (S_{init}, S_1, \ldots, S_k, S_{k+1})$. We have constructed two *answer set programs*[4] (ASP), one for the base case and one for the induction step, which can be fed into any answer-set solver to prove the validity of φ by proving inconsistency of the two programs. This has been achieved by creating a correspondence between counter examples of φ (which amount to be finite state sequences) and answer sets of the programs, using a *linear* temporal extension of the GDL rules and a formula encoding which states that the formula is *violated*.

In this paper, we will extend the summarised method to (a subclass of) epistemic formulas. To this end, we first define the class of *positive-knowledge* formulas whose counter examples correspond to multiple state sequences and can hence succinctly be represented as answer sets using a linear time structure. As an intermediate step, we characterise potential counter examples of positive-knowledge formulas by a structure called *sequence mapping*, define an alternative formula semantics based on this structure and show its equivalence to the original formula semantics from Definition 3. In a later section, this equivalence will provide the link from developments that violate positive-knowledge formulas according to the original semantics to answer sets for the (later introduced) answer set programs.

4.1 Positive-Knowledge Formulas

Example formula $\rho = \neg\textit{terminal} \vee K_x\textit{terminal}$ is not valid with respect to the GDL description of Krieg-Tictactoe. If player o places a marker that completes a line of markers o (which yields a terminal state) while having another option that does not (which yields a non-terminal state), player x must consider both successor states possible afterwards and does hence not know about termination. More precisely, we have $last(\delta_\rho) \not\models_{\delta_\rho} \rho$ for a game development $\delta_\rho = (S_{init} \xrightarrow{A_0} S_1 \ldots \xrightarrow{A_5} S_6)$, say, that is such that $A_0(o) = A_1(x) = A_2(o) = \ldots = A_5(x) = \textit{noop}$ and

$$
\begin{array}{ll}
A_0(x) = \textit{mark}(1,1) & A_1(o) = \textit{mark}(3,1) \\
A_2(x) = \textit{mark}(1,2) & A_3(o) = \textit{mark}(3,2) \\
A_4(x) = \textit{mark}(2,1) & A_5(o) = \textit{mark}(3,3)
\end{array}
$$

State S_6 is terminal, hence the first disjunct $\neg\textit{terminal}$ of ρ is violated. The second disjunct $K_x\textit{terminal}$ is violated since there is a development δ'_ρ such that $\delta_\rho \sim_x \delta'_\rho$ which ends in a non-terminal state, e.g. the one resulting from δ_ρ by changing $A_5(o) = \textit{mark}(3,3)$ to $A'_5(o) = \textit{mark}(1,3)$. We see that the violation of ρ is uniquely characterised by the two developments δ_ρ and δ'_ρ. On the contrary, a counter example for formula $\neg\textit{terminal} \vee \neg K_x\textit{terminal}$ can *not* appropriately be characterised by two sequences, as it requires that $K_x\textit{terminal}$ be true and hence that *all* developments δ' such that $\delta_\rho \sim_x \delta'$ satisfy $last(\delta') \models_{\delta'} \textit{terminal}$. The following definition generalises this motivation to a class of formulas whose counter examples can compactly be represented by single state sequences. Intuitively, it comprises all formulas which do not include formulations of what players do *not* know.

Definition 4 *An epistemic formula φ is called* positive-knowledge *formula if no K_r (for any role r) in φ is within the scope of negation, $\neg$.*

It is also possible to allow K_r in the scope of even numbers of negations. At a later point, however, these double negations would have to be "compiled away," which is why we decided in favour of this more restrictive definition, for the sake of simplicity.

4.2 Sequence Mappings

We will now define a structure for the formal representation of single state sequences which amount to be potential counter examples for positive-knowledge formulas. This will be done by assigning one sequence to the formula itself,

[3] A formula is called *valid* if it is true in all reachable states.

[4] *Answer sets* are specific models of logic programs with negation (Gelfond 2008). We use two common additions (Niemelä, Simons, and Soininen 1999): a *weight atom* $m\,\{p_1, \ldots, p_k\}\,n$ means that an answer set satisfies at least m and at most n different p_i. 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 $b_1, \ldots, b_k$. Logic programs that contain these additions will be called *answer set programs*.

and by assigning one further sequence to each occurrence of a knowledge operator (handling multiple occurrences of the same knowledge operator separately). Sequences will be assigned using the following set of names.

Definition 5 *Let φ be a positive-knowledge formula with n occurrences of knowledge operators, uniquely characterised by their positions $\pi_1, \ldots, \pi_n$ in the syntax tree of φ. A set of* views[5] *of φ, denoted $\mathcal{V}_\varphi$, is a minimal set such that*

- *$v_0 \in \mathcal{V}_\varphi$, and*
- *there is a bijection from $\{v_1, \ldots, v_n\}$ to $\{\pi_1, \ldots, \pi_n\}$ and $\{v_1, \ldots, v_n\} \subseteq \mathcal{V}_\varphi$.*

For each positive-knowledge formula φ, we consider the set of views $\mathcal{V}_\varphi$ for φ as uniquely given. A *sequence mapping for φ wrt. a game development δ* is then given as a function $\mathcal{M}$ from views $v \in \mathcal{V}_\varphi$ to developments over a GDL specification that satisfies the following properties:

- Development $\mathcal{M}(v_0)$ is of the form (δ, σ), where σ is a $deg(\varphi)$-max sequence. That is to say, the given game development δ is prolongated by a sequence σ of appropriate length which potentially violates φ.
- For each occurrence of a knowledge operator K_r in φ and its corresponding view v_i ($v_i \neq v_0$), and for the view v_j corresponding to the knowledge operator nearest to K_r towards the root of the syntax tree of φ (in case it exists) or such that $v_j = v_0$ (otherwise):
  - development $\mathcal{M}(v_i)$ is of the form (δ', σ'), where $|\delta| = |\delta'|$ and σ' is a $deg(\varphi)$-max sequence, and
  - $\mathcal{M}(v_i)$ "relates appropriately" to $\mathcal{M}(v_j)$ according to $\sim_r$.

For the benefit of the reader, we refrain from a formal characterisation of the term "relates appropriately" and, rather than formalising all subsequent technical details, concentrate on showing the main ideas with the example formula $\rho = \neg terminal \vee K_x terminal$ in the remainder of the paper (a comprehensive presentation with all details is part of a doctoral dissertation (Haufe 2012)). The set of views for ρ can be given as $\{v_0, v_1\}$ (where v_0 corresponds to ρ and v_1 corresponds to K_x). Using the above-mentioned developments δ_ρ and δ'_ρ, one possible sequence mapping for ρ wrt. δ_ρ is the function $\mathcal{M}_\rho$ such that $\mathcal{M}_\rho(v_0) = \delta_\rho$ and $\mathcal{M}_\rho(v_1) = \delta'_\rho$. Both sequences "relate appropriately" according to $\sim_x$, e.g. we have $\mathcal{M}_\rho(v_0) \sim_x \mathcal{M}_\rho(v_1)$.

4.3 An Equivalent Semantics Over Sequence Mappings for Positive-Knowledge Formulas

The first step towards the establishment of our induction proof method for the validity of positive-knowledge formulas via answer set programs is the construction of an equivalent formula semantics over sequence mappings. A sequence mapping $\mathcal{M}$ for formula φ wrt. development δ can be used to interpret φ by interpreting each subformula of φ with respect to the development assigned to the nearest knowledge operator towards the root of the syntax tree of φ, or with respect to development δ in case no such knowledge operator exists. This is made precise with the following definition.

Definition 6 *Let G be a GDL specification, φ be a positive-knowledge formula, δ be a development, and $\mathcal{M}$ be a sequence mapping for φ wrt. δ. We say that $\mathcal{M}$* satisfies *φ, denoted $\mathcal{M} \Vdash \varphi$, if $(\mathcal{M}, |\delta|, v_0) \Vdash \varphi$ holds according to the following definition (where $0 \leq k \leq deg(\varphi)$):*

$(\mathcal{M}, k, v) \Vdash p$ *iff* $G \cup S_k^{\texttt{true}} \vdash p$ *(p ground atom), where* $\mathcal{M}(v) = (S_{init}, S_1, \ldots, S_k, \ldots, S_{k+m})$

$(\mathcal{M}, k, v) \Vdash \neg\psi$ *iff* $(\mathcal{M}, k, v) \nVdash \psi$ *(likewise for $\wedge, \vee$)*

$(\mathcal{M}, k, v) \Vdash \bigcirc\psi$ *iff* $m = 0$, *or* $(\mathcal{M}, k+1, v) \Vdash \psi$, *where* $\mathcal{M}(v) = (S_{init}, S_1, \ldots, S_k, \ldots, S_{k+m})$

$(\mathcal{M}, k, v) \Vdash K_r\psi$ *iff* $(\mathcal{M}, k, v') \Vdash \psi$ *where view v' corresponds to K_r.*

Note that all cases different from $K_r\psi$ exactly correspond to their counterparts in Definition 3. Case $K_r\psi$, on the contrary, does not explicitly incorporate $\sim_r$ anymore. Hence, $(\mathcal{M}, k, v) \Vdash K_r\psi$ for a sequence mapping $\mathcal{M}$ such that $\mathcal{M}(v) = (\delta, \sigma)$ does *not* generally imply the correspondent $\sigma \models_\delta K_r\psi$, as this would require $last(\delta') \models_{\delta'} \psi$ to hold for *all* developments δ' such that $\delta \sim_r \delta'$, whereas $(\mathcal{M}, k, v) \Vdash K_r\psi$ only provides that correspondence for *one* such development δ'. However, following the spirit of a sequence mapping as structure for potential *counter examples* of a positive-knowledge formula, there is a correspondence between the existence of a sequence mapping $\mathcal{M}$ such that $(\mathcal{M}, k, v) \nVdash K_r\psi$ and the existence of a sequence σ such that $\sigma \not\models_\delta K_r\psi$. This is more generally stated with the following theorem.

Theorem 1 Let φ be a positive-knowledge formula, δ be a development, and σ be a $deg(\varphi)$-max sequence starting at $last(\delta)$. Then the following are equivalent.

- $\sigma \not\models_\delta \varphi$
- There is a sequence mapping $\mathcal{M}$ for φ wrt. δ such that $\mathcal{M}(v_0) = (\delta, \sigma)$ and $\mathcal{M} \nVdash \varphi$.

Proof (sketch): Induction on the structure of φ. The base case considers φ arbitrarily knowledge-free (possibly containing negation). This allows to neglect negation in the induction step (as φ is a positive-knowledge formula), and *single* sequences can be related with *single* sequence-mappings in all remaining cases. □

5 Encoding Epistemic Game Knowledge

In this section we provide the second step towards the establishment of our induction proof method. It is concerned with the construction of a stratified logic program over a GDL specification which allows to infer whether a given formula is entailed with respect to a given sequence mapping solely via the unique standard model of the constructed program. To this end, we show how to extend the game rules to account for time and different views, how to encode a particular sequence mapping into these extended game rules,

[5] Also the notion *worlds* is conceivable here. However, since a *possible world* is widely understood to refer to a state which is considered possible, whereas we are assigning names to knowledge operators instead of particular worlds, we decide to introduce a different notion.

and how to further encode a particular formula. We prove that the resulting program admits a unique standard model containing a special formula name token if, and only if, the formula is true with respect to the encoded sequence mapping. This result will be necessary to relate arbitrary sequence mappings to answer sets of the (later introduced) answer set programs for the induction proof method.

5.1 Epistemic Temporal GDL Extension

A sequence mapping interprets each subformula of a formula φ with regard to the sequence associated with the nearest knowledge operator towards the root of the syntax tree of φ, or associated with φ in case this operator does not exist. Accordingly, an encoding into the game description will require one "copy" of the description for each view. Moreover, as we consider formulas with possible occurrences of operator $\bigcirc$ and hence with a finite lookahead in time, we also need a *temporal extension* (Thielscher 2009).

Definition 7 *For game G let $G_{\leq n}(v) \stackrel{\text{def}}{=} \bigcup_{0 \leq i \leq n} G_i(v)$, where each $G_i(v)$ is constructed by*

- *omitting all clauses from G with head* **init***;*
- *replacing each occurrence of*
  - **next**(f) *by* **true**$(f, i+1, v)$,
  - **sees**(r, s) *by* **sees**$(r, s, i+1, v)$, *and*
  - $p(t_1, \ldots, t_n)$ *by* $p(t_1, \ldots, t_n, i, v)$, *for each predicate p s.t. $p \neq$* **next** *and $p \neq$* **sees**.

Let φ be a formula with degree $n = deg(\varphi)$. The epistemic temporal extension of G wrt. φ*, denoted G_φ, is defined as*

$$G_\varphi \stackrel{\text{def}}{=} \bigcup_{v \in \mathcal{V}_\varphi} G_{\leq n}(v).$$

Put in words, an epistemic temporal extension of G wrt. φ is composed of a temporal extension of G up to depth $deg(\varphi)$ for each of the views of φ. The resulting program can be simplified by omitting the time and view argument in any atom over a predicate symbol that does not depend on any GDL keyword.

As an example for the epistemic temporal GDL extension, let G be the clauses in Figure 1, and reconsider example formula $\rho = \neg terminal \vee K_x terminal$ with the views v_0 and v_1. Then for clause 19, the following clause is in the epistemic temporal extension G_ρ of G for v_0 (and similarly for v_1):

```
sees(R,yourmove,1,v0) :-
        not validmove(0,v0),
        true(control(R),0,v0).
```

5.2 Sequence-Mapping and Formula Encoding

A sequence mapping $\mathcal{M}$ for a formula φ wrt. a development of length k is encoded to a program which solely consists of facts. For each view $v \in \mathcal{V}_\varphi$ and the corresponding development $\mathcal{M}(v) = (S_{init}, S_1, \ldots, S_k, \ldots, S_{k+m})$, it encodes the subsequence starting at S_k, i.e., all prefixes of length k are omitted. This allows to use the same finite lookahead $deg(\varphi)$ in the epistemic temporal GDL extension for *each* sequence mapping, and is especially important for the later construction of the induction-step program which will encode *multiple* sequence mappings "at once."

Definition 8 *Let $\mathcal{M}$ be a sequence mapping for a formula φ wrt. a development of length k. The* encoding of $\mathcal{M}$*, denoted Enc($\mathcal{M}$), is a program which consists of the following facts for each view $v \in \mathcal{V}_\varphi$ and its corresponding development $\mathcal{M}(v) = S_{init} \xrightarrow{A_0} S_1 \ldots \xrightarrow{A_{k-1}} S_k \ldots \xrightarrow{A_{k+m-1}} S_{k+m}$ (where $m \geq 0$):*

- *for all $f \in S_k$:* **true**$(f, 0, v)$ **.** $\in Enc(\mathcal{M})$*; and*
- *for all players r, and for all $i < m$: if r takes move a in joint action A_{k+i}, then* **does**(r, a, i, v) **.** $\in Enc(\mathcal{M})$.

We have shown recently (Haufe, Schiffel, and Thielscher 2012) how arbitrary knowledge-free formulas φ can be encoded into a temporally extended game description and a sequence encoding using a finite set of stratified clauses $Enc(\varphi)$. In case φ is a ground atom, $Enc(\varphi)$ solely contains a clause whose head is a unique name predicate of arity 0, denoted with $name_\varphi$, and whose clause body refers to a variant of this ground atom which is extended by a time argument. In case φ is a composed formula, $Enc(\varphi)$ contains a clause, again with unique head $name_\varphi$, whose body recursively resolves the toplevel connective of φ by referring to the unique name predicates of its respective subformulas, and $Enc(\varphi)$ additionally contains an encoding of these subformulas.

This process can easily be extended to the knowledge setting by adding the current view as further argument to the encoding of ground atoms, we omit the formal specification here. Example formula $\rho = \neg terminal \vee K_x terminal$ can be encoded as $Enc(\rho)$ such that $name_\rho =$ `rho` as follows (removing some indirections in the generated clauses):

```
rho :- not terminal(0,v0).
rho :-     terminal(0,v1).
```
(1)

5.3 Linking Sequence-Mapping Semantics and Logic-Program Encoding

The following theorem shows that a formula φ is violated by some sequence mapping if, and only if, the corresponding epistemic temporal GDL extension for φ together with an encoding of that sequence mapping and an encoding of φ does not contain the unique name token $name_\varphi$.

Theorem 2 Let G be a GDL description, φ be a positive-knowledge formula, $\delta = (S_{init}, S_1, \ldots, S_k)$ be a development and $\mathcal{M}$ be a sequence mapping for φ wrt. δ. Then the following are equivalent:

- $\mathcal{M} \not\Vdash \varphi$
- $G_\varphi \cup Enc(\mathcal{M}) \cup Enc(\varphi) \not\vdash name_\varphi$.

Proof (sketch): Induction on the structure of φ, using the corresponding Theorem 14 for knowledge-free formulas in our earlier work (Haufe, Schiffel, and Thielscher 2012) together with the observation that $G_\varphi \cup Enc(\mathcal{M}) \cup Enc(\varphi)$ is composed of independent subprograms which involve exactly one view $v \in \mathcal{V}_\varphi$. □

6 Proving Epistemic Game Knowledge

We are now ready to extend our recent induction technique (Haufe, Schiffel, and Thielscher 2012) to the general case of epistemic game knowledge. As mentioned before, for each formula which is to be proved, it relies on the construction of two answer set programs, one to establish the base case and one to establish the induction step. In case both answer set programs are inconsistent, the formula is known to be true in each reachable state of the game. In this section, we show how to specify these two programs, and prove the correctness of the extended method. We will again refrain from the technical details and show the ideas for program construction wrt. example formula $\rho = \neg \mathit{terminal} \vee K_x \mathit{terminal}$.

6.1 Base Case Program

The answer set program for the base case proof of ρ is given as

$$P^{bc}_{\rho}(G) = G_{\rho} \cup \mathit{Enc}(S_{\mathit{init}}) \cup \mathit{Enc}(\rho) \cup \{\texttt{ :- rho.}\}.$$

$G_{\rho} = G_{\leq 0}(v_0) \cup G_{\leq 0}(v_1)$ is the epistemic temporal extension of the clauses from Figure 1. $\mathit{Enc}(S_{\mathit{init}})$ is an encoding for the initial state in both views v_0 and v_1, i.e., $\mathit{Enc}(S_{\mathit{init}}) = \{\mathbf{true}(f, 0, v)\texttt{ .} : f \in S_{\mathit{init}},\ v \in \{v_1, v_2\}\}$. Program $G_{\rho} \cup \mathit{Enc}(S_{\mathit{init}})$ admits a unique answer set which corresponds to the (also unique) sequence mapping $\mathcal{M}$ for ρ which is wrt. the initial development (S_{init}). $\mathit{Enc}(\rho)$ is the encoding for ρ as specified in (1), enriched with the constraint `:- rho.`, which formulates that ρ should not be true. The addition of $\mathit{Enc}(\rho) \cup \{\texttt{ :- rho.}\}$ to $G_{\rho} \cup \mathit{Enc}(S_{\mathit{init}})$ (which yields $P^{bc}_{\rho}(G)$) ensures that the unique answer set is rejected in case $\mathcal{M}$ satisfies ρ and hence establishes the correspondence between answer set and counter example of ρ in the initial state. $P^{bc}_{\rho}(G)$ is inconsistent in this particular example, as the clauses $G_{\leq 0}(v_0)$ and $G_{\leq 0}(v_1)$ coincide up to renaming of the view argument but are forced to deviate regarding **terminal** by $\mathit{Enc}(\rho) \cup \{\texttt{ :- rho.}\}$.

6.2 Induction Step Program

The answer set program for the induction step proof of ρ is given as

$$\begin{aligned} P^{is}_{\rho}(G) \quad = \quad & G_{\bigcirc\rho} \cup S^{gen} \cup P^{legal}_{\bigcirc\rho} \cup \\ & \mathit{Enc}(\rho) \quad \cup \{\texttt{ :- } \mathbf{not}\ \texttt{rho.}\} \cup \\ & \mathit{Enc}(\bigcirc\rho) \cup \{\texttt{ :- nxt\_rho.}\}. \end{aligned}$$

The components of $P^{is}_{\rho}(G)$ are explained in the following.

Epistemic Temporal GDL Extension $G_{\bigcirc\rho}$: Compared to the base case program, the extension of the game rules involves a further time step: $G_{\bigcirc\rho} = G_{\leq 1}(v_0) \cup G_{\leq 1}(v_1)$.

State Generator S^{gen}: Instead of the initial state encoding $\mathit{Enc}(S_{\mathit{init}})$ in both views v_0 and v_1 from ρ, we need a "state generator" program whose answer sets correspond exactly to pairs of states (S_0, S_1) such that 1) S_0 is the last state of some development δ_0 for view v_0 and 2) S_1 is the last state of some development δ_1 for view v_1 such that $\delta_0 \sim_x \delta_1$. However, obtaining this information requires a full state space search in general, which is not feasible in interesting games. This motivates to apply easily obtainable overestimations of the reachable states and the relation $\sim_r$ which do not influence the (yet to be established) correctness of the method. The simplest approximation is the program

$$0\{\,\mathbf{true}(f\texttt{,}0\texttt{,}v)\texttt{ :}f \in \mathit{FDom},\ v \in \{v_0, v_1\}\},$$

which, for the finite domain of all fluents *FDom*[6], generates *all* possible combinations (S_0, S_1) of (not necessarily reachable) states S_0 for view v_0 and S_1 for v_1. Better approximations of the reachable states, e.g. the restriction to those which contain exactly one instance of $\mathit{cell}(x, y, c)$ for each coordinate pair (x, y), can be obtained automatically from the game description by preceding proofs of knowledge-free formulas (Haufe, Schiffel, and Thielscher 2012). Some restrictions on the estimation of $\sim_r$ are obtained by the encoding of the induction hypothesis mentioned below, and we give directions to further restrictions in Section 6.4.

State Sequence Encoding $P^{legal}_{\bigcirc\rho}$: Since the epistemic temporal GDL extension $G_{\bigcirc\rho}$ involves successive game states, we need an additional encoding which ensures that each player r performs one legal move out of the finite set *ADom* of moves in each non-terminal state wrt. both views $v \in \{v_0, v_1\}$ of ρ:

$$\begin{aligned} & 1\{\mathbf{does}\,(r, a, 0, v) : a \in \mathit{ADom}\}1. \\ & \texttt{ :- } \mathbf{does}\,(r, \texttt{A}, 0, v), \mathbf{not\ legal}\,(r, \texttt{A}, 0, v). \end{aligned}$$

Moreover, the clause sets for v_0 and v_1 need to be related according to $\sim_x$, that is,

- The actions of player x in both views coincide at time step 0:

```
:- does(x,A,0,v0) , not does(x,A,0,v1) .
:- does(x,A,0,v1) , not does(x,A,0,v0) .
```

- The percepts of player x in both views coincide at time step 1:

```
:- sees(x,S,1,v0) , not sees(x,S,1,v1) .
:- sees(x,S,1,v1) , not sees(x,S,1,v0) .
```

Each sequence mapping $\mathcal{M}$ of ρ wrt. a development of length k corresponds to an answer set of program $G_{\bigcirc\rho} \cup S^{gen} \cup P^{legal}_{\bigcirc\rho}$ where, similarly to Definition 8, the prefixes of length k are not represented. There are additional answer sets due to the overestimation of the reachable states and the relation $\sim_x$ in the state generator S^{gen}.

Formula Encoding $\mathit{Enc}(\rho) \cup \mathit{Enc}(\bigcirc\rho)$ and Constraints: To encode the induction hypothesis, we use the encoding $\mathit{Enc}(\rho)$ for ρ from (1) together with the constraint

[6] In our previous work (Haufe, Schiffel, and Thielscher 2012) we describe in detail a method for automatically computing finite domain sets like *FDom* and *ADom* (introduced later).

`:-` **not** `rho`. It restricts `rho` to be contained in each answer set and hence implies that ρ is satisfied by the sequence mapping corresponding to the answer set (if it exists). Additionally, we encode that ρ is *not* true in a direct successor state of S:

```
:- nxt_rho.
nxt_rho :- not terminal(1,v0).
nxt_rho :-     terminal(1,v1).
```

It differs from the encoding for ρ in the increased time step and in the constraint `:- nxt_rho.`, which forces ρ to be false at time step 1. $Enc(\rho)$ uses the same views as $Enc(\bigcirc\rho)$ and hence restricts the before-mentioned overestimation of $\sim_r$, in this example by removing all answer sets which correspond to state pairs (S_0, S_1) where S_0 is terminal and S_1 is non-terminal.

The addition of the formula encodings and the constraints `:-` **not** `rho` and `:- nxt_rho.` to program $G_{\bigcirc\rho} \cup S^{gen} \cup P^{legal}_{\bigcirc\rho}$ (which yields $P^{is}_{\rho}(G)$) ensures that each answer set is rejected which represents a sequence mapping that violates ρ in the current state or satisfies ρ in the next state. This assures that each remaining answer set either corresponds to a counter example of the induction step or occurs due to overestimation in the state generator. $P^{is}_{\rho}(G)$ admits an answer set in this particular example, which can be argued using the constructed sequence mapping $\mathcal{M}_\rho$ from the end of Section 4.2. As mentioned before, $\mathcal{M}_\rho$ violates ρ at depth $|\delta_\rho| = 6$. Additionally, $\mathcal{M}_\rho$ satisfies ρ when interpreted at depth 5 (as state S_5 is non-terminal). Hence, $P^{is}_{\rho}(G)$ admits an answer set which represents $\mathcal{M}_\rho$ with omitted prefixes of length 5.

6.3 Correctness

We believe that the general idea of the correctness proof for our method becomes clear without explicit representation of the two general answer set programs $P^{bc}_{\varphi}(G)$ and $P^{is}_{\varphi}(G)$ for arbitrary positive-knowledge formulas φ, which is why we now state the result.

Theorem 3 Consider a GDL specification G and a positive-knowledge formula φ over G. If both $P^{bc}_{\varphi}(G)$ and $P^{is}_{\varphi}(G)$ are inconsistent, then for all developments $\delta = (S_{init} \xrightarrow{A_0} S_1 \dots \xrightarrow{A_{k-1}} S_k)$ we have $S_k \vDash_\delta \varphi$.

Proof (sketch): Induction on k, similar to the proof of the corresponding Theorem 17 for knowledge-free formulas in our earlier work (Haufe, Schiffel, and Thielscher 2012).

Base case: Violation of φ by the development $\delta = (S_{init})$ implies violation of φ via some sequence mapping for φ wrt. δ (by Theorem 1), in turn implying satisfiability of $P^{bc}_{\varphi}(G)$ (by Theorem 2).

Induction step: Satisfiability of $P^{is}_{\varphi}(G)$ is obtained similarly to the base case, deriving existence of a sequence mapping $\mathcal{M}$ for formula $\bigcirc\varphi$ wrt. an *arbitrary* development δ using the following two arguments.

- The standard model of program $G_{\bigcirc\varphi} \cup Enc(\mathcal{M})$ is also an answer set of program $G_{\bigcirc\varphi} \cup S^{gen} \cup P^{legal}_{\bigcirc\varphi}$.
- The induction hypothesis $last(\delta) \vDash_\delta \varphi$ implies (by Theorem 1 and the coinciding view structures of φ and $\bigcirc\varphi$) that also $\mathcal{M}$ satisfies φ and hence that adding $Enc(\varphi) \cup \{$ `:-` **not** `rho.` $\}$ retains consistency. □

6.4 Towards Completeness

Assume a state generator S^{gen} for the induction step program $P^{is}_{\rho}(G)$ which does not rely on overestimation, i.e., which is such that each of its answer sets corresponds to a sequence mapping satisfying φ and violating $\bigcirc\varphi$. Calling such a state generator *accurate*, we get the following completeness result.

Theorem 4 Let G be a GDL specification, let φ be a positive-knowledge formula over G, and let $P^{is}_{\varphi}(G)$ be constructed over an accurate state generator. If for all developments $\delta = S_{init} \xrightarrow{A_0} S_1 \dots \xrightarrow{A_{k-1}} S_k$ we have $S_k \vDash_\delta \varphi$, then $P^{bc}_{\varphi}(G)$ and $P^{is}_{\varphi}(G)$ are inconsistent.

Proof (sketch): Consistency of one of the programs $P^{bc}_{\varphi}(G)$ and $P^{is}_{\varphi}(G)$ allows to conclude existence of a sequence mapping $\mathcal{M}$ which violates φ (by Theorem 2), in turn implying existence of a development δ which ends in some state S_k that violates φ (by Theorem 1). □

One of the main reasons for the practicability of our approach is the use of induction. It allows to verify formulas without having to traverse the usually vast state space of a game by taking advantage of the comparably compact structure of game rules formulated in the game description language. The information which is necessary to construct an accurate state generator, however, can in turn only be obtained by a full state space traversal in the worst case. Hence, exploiting the advantage of induction inherently requires to apply quickly obtainable overestimations of reachable states and the relation $\sim_r$, which possibly introduces counter examples for actually valid formulas and thus causes the loss of completeness in our approach.

Nevertheless, the accuracy of state generators can be increased by the generalisation of techniques where reachable states are approximated with increasing accuracy by adding encodings of previously proved formulas (together with constraints that require them to be true) to subsequent answer set programs (Haufe, Schiffel, and Thielscher 2012). Similarly, the estimation of the accessibility relation can be made more precise. For example, having proved that player x exactly knows his legal moves allows to encode this information in subsequent answer set programs by adding the clauses

```
:- legal(x,A,0,v0) , not legal(x,A,0,v1) .
:- legal(x,A,0,v1) , not legal(x,A,0,v0) .
```

As a result, each answer set violating this property is now dismissed, making the method "more complete".

7 Experimental Results

To show the practical applicability of our proof method, we have extended our implementation for proving knowledge-free properties (Haufe, Schiffel, and Thielscher 2012). It uses the general game-playing system FLUXPLAYER (Schiffel and Thielscher 2007) for program generation and tools

from POTASSCO (Gebser et al. 2011) to solve them. We have performed automated proofs with a selection of general epistemic properties in a variety of games, including Krieg-Tictactoe, a simple card game (Thielscher 2010), the famous Monty Hall problem (Rosenhouse 2009; Thielscher 2011), and all incomplete-information games[7] from the 1st German Open in General Game Playing[8], which recently featured the first ever track on incomplete-information games in a GGP competition.

7.1 Property Categories

To each of the aforementioned incomplete-information games, we have attempted proofs for all formulas from the following three categories, each referring to an arbitrarily chosen player r different from **random** (the other players yield similar results and are hence omitted here).

Knows Terminal We formulate that player r knows whether the game has terminated with the formula

$$K_r\mathit{terminal} \vee K_r\neg\mathit{terminal}.$$

Knows Legals For all non-random players r', we formulate that player r knows which of the moves of r' are legal with the formula

$$\bigwedge_{a\in\mathit{ADom}} (K_r\mathit{legal}(r',a) \vee K_r\neg\mathit{legal}(r',a)).$$

Knows Goals For all non-random players r' and the set GV of goal values that occur in the game description, we formulate that r knows the goal values of r' in a terminal state with the formula

$$\mathit{terminal} \supset (\bigwedge_{g\in GV} (K_r\mathit{goal}(r',g) \vee K_r\neg\mathit{goal}(r',g))).$$

7.2 Proof Procedure

Each newly considered game takes a few seconds for the initialisation of FLUXPLAYER (which includes the calculation of the domains *FDom* and *ADom*). Afterwards, we attempt to obtain a restrictive state generator for the induction step programs by proving automatically generated knowledge-free formulas for properties such as the uniqueness of cell content, this process takes time in the range of 0.1 seconds (Krieg-Tictactoe) to 4.6 seconds (Backgammon). We then attempt separate proofs of all formulas from the three introduced categories. Each attempt is done according to the following scheme, the results can be found in Figure 2. Let φ be any formula of the mentioned categories which is to be proved.

Phase 1 φ does not contain any $\bigcirc$, hence according to Proposition 2, omitting all knowledge operators from φ yields a formula $kf_0(\varphi)$ which is equivalent to φ with respect to entailment in the initial state of the game. Since $kf_0(\varphi)$ is a tautology, the base-case proof can completely be omitted (it will always prove φ in the initial state). Hence, we directly attempt the induction-step proof with program $P^{is}_{\varphi}(G)$. If it is successful (i.e., if $P^{is}_{\varphi}(G)$ is inconsistent), then the considered formula is valid, and we skip Phase 2. If, on the other hand, $P^{is}_{\varphi}(G)$ admits an answer set, the validity of φ is still unknown, as the answer set does not necessarily correspond to a sequence mapping due to the overestimation in the state generator S^{gen}. In that case we move to Phase 2.

Phase 2 We can also disprove φ by finding a consistent answer set program $P^{bc}_{\bigcirc^t\varphi}(G)$ for some $t \in \mathbb{N}$. Consistency allows to conclude that there is a development δ of length t such that $last(\delta) \not\models_\delta \varphi$ (by Theorem 4). Hence, φ is violated by the *reachable* state $last(\delta)$, implying that φ is invalid. We attempt successive Base-Case Proofs on $\bigcirc^t\varphi$ for $t = 1, 2, \ldots$, until one of the following cases arises:

- A time limit of 20 seconds is reached. We stop the process, the validity of the formula is still unknown. For some $t \geq 1$, the last proof attempt for the base case has been on $\bigcirc^t\varphi$, and we indicate this t in Figure 2.
- We obtain an answer set for $P^{bc}_{\bigcirc^t\varphi}(G)$ for some $t \geq 1$, in which case φ is invalid, and t is again indicated in Figure 2.

7.3 Interpretation of the Results

Formula $K_x\mathit{terminal} \vee K_x\neg\mathit{terminal}$ (category Knows Terminal) can be disproved in Krieg-Tictactoe in depth 6 of the game tree first. Intuitively, this is due to the passed turn information via **sees** in the GDL specification of Krieg-Tictactoe in Figure 1, which causes each player to know the amount of currently placed pieces after each development of the game. Hence, player x first considers both a non-terminal and a terminal state possible after at least 6 joint actions, when player o could have completed a line.

Each formula φ_t in category Knows Terminal has an associated set of views $\mathcal{V}_{\varphi_t}$ of size $|\mathcal{V}_{\varphi_t}| = 3$, whereas each formula φ_l in category Knows Legals is such that $|\mathcal{V}_{\varphi_l}| = 2 * |\mathit{ADom}| + 1$. This implies that the size of an answer set program which is generated for φ_l exceeds the size of the corresponding answer set program for φ_t by factor $|\mathit{ADom}|$. Accordingly, in Figure 2, times for proof attempts on formulas φ_l are generally higher than those for φ_t. The induction step proofs mostly take time in the range of a few seconds only and hence show the efficiency of our method. However, some invariants cannot be proved due to our rather uninformed state generator, and further methods to its restriction (e.g. by inclusion of previously-proved positive-knowledge formulas as mentioned in Section 6.4) are required.

8 Related Work

The first approach that addresses the automated verification of *epistemic* properties over GDL specifications (Ruan and Thielscher 2011) used standard epistemic logic over Kripke semantics (Fagin et al. 1995), and the main focus of that work is the introduction and equivalence proof of a new semantics that allows the direct verification of epistemic logic

[7] All game descriptions can be found at *ggpserver.general-game-playing.de/public/show_games.jsp*.

[8] *www.tzi.de/~kissmann/ggp/go-ggp*

game	Knows Terminal			Knows Legals			Knows Goals		
backgammon	r	20.2	(?,9)	r of r:	86.5	(?,1)	r of r:	21.7	(?,6)
				r of b:	86.8	(?,1)	r of b:	21.8	(?,6)
cardgame	j	0.0	(y)	j of j:	0.5	(y)	j of j:	19.6	(?,25)
				j of r:	0.4	(y)	j of r:	19.6	(?,25)
kriegtictactoe	x	1.5	(n,6)	x of x:	0.2	(y)	x of x:	2.7	(n,6)
				x of o:	1.6	(n,4)	x of o:	2.4	(n,6)
kriegTTT_5x5	x	1.4	(n,4)	x of x:	20.3	(?,5)	x of x:	2.2	(n,4)
				x of o:	1.4	(n,1)	x of o:	2.3	(n,4)
mastermind	p	19.7	(?,17)	p of p:	120.0	(?,1)	p of p:	21.3	(?,4)
meier	1	0.1	(y)	1 of 1:	24.4	(?,4)	1 of 1:	2.3	(n,6)
				1 of 2:	23.6	(?,4)	1 of 2:	2.3	(n,6)
montyhall	c	0.0	(y)	c of c:	0.1	(y)	c of c:	0.3	(n,3)
transit	t	19.5	(?,20)	t of t:	20.1	(?,15)	t of t:	19.5	(?,18)
				t of p:	1.0	(n,2)	t of p:	20.3	(?,19)
vis_pacman3p	p	20.8	(?,8)	p of p:	21.6	(?,7)	p of p:	9.7	(y)
				p of b:	7.2	(n,3)	p of b:	32.6	(?,4)
				p of i:	7.2	(n,3)	p of i:	32.7	(?,4)

Figure 2: Proof times in seconds for a selection of positive-knowledge formulas in several incomplete-information games, where "y" means valid, "n" means invalid (violated in some reachable state), and "?" means validity unknown. The names of players have been shortened to one letter, so that, e.g., "j of r" in Cardgame stands for "Jane knows the legal moves (goal values, respectively) of Rick." The time for result (y) indicates one induction-step proof (Phase 1). The time for results (n,t) and (?,t) indicates the overall time needed for one induction-step proof (Phase 1) and t base-case proofs (Phase 2).

formulas over GDL descriptions. This previous work also sketches an approach for model checking epistemic properties via answer set programming, which however requires to systematically search the entire set of reachable positions in a game and hence is not applicable in the time-restricted setting of the general game playing competition except for very simple games.

9 Summary and Discussion

Moving from complete to incomplete information poses a new challenge to both designer and player of a general game. While the former must endow each player with sufficient information to guarantee unobstructed gameplay, the latter needs to incorporate all opponents' viewpoints to maximise its utility. For this purpose, we have extended our recently developed automated verification method (Haufe, Schiffel, and Thielscher 2012) to incorporate a large class of *epistemic* properties. We have shown its effectiveness by proving several common epistemic properties in a variety of games with a practical implementation that uses a state-of-the-art answer set solver.

One of the two main reasons for the practicability of our approach is the use of induction together with an easily obtainable overestimation of the state space and the restriction to formulas that only contain finite time reference. The other main reason is the utilisation of a linear time structure. It allows to linearly bound the growth of game rules which is mandatory for the required temporal gdl extension, whereas an also possible branching time structure causes an exponential blowup which almost certainly drops performance below practical usability. Linear time in answer set programming, on the other hand, only allows the verification of formulas which have succinct counter examples that are representable using linear state sequences. Our introduced class of positive-knowledge formulas hence forms a necessary restriction to the utilisation of a linear time structure.

As only the *lack* of knowledge cannot be formulated, positive-knowledge formulas still cover an interesting property class. In our experiments, we have concentrated on three game-independent properties whose discovered validity assist a game designer by showing that each player is endowed with sufficient information to play a newly specified game. But also general game players can take advantage of our approach. For example, the provable fact that players always know which cells contain their own marks when playing the game Krieg-Tictactoe specified in Figure 1 allows to spare repeated reasoning regarding this fact in every reached game state. In exploration games such as the famous Wumpus World, a general game player which is able to find out what is known after certain actions in advance can prefer those that reveal as much information as possible. The development of imperfect information-game players is just at the beginning, and we expect that many of the existing methods using knowledge will require the addition of epistemic properties when being generalised to imperfect-information games.

Acknowledgements. We thank our anonymous reviewers for helpful suggestions on an earlier version of the paper.

This research was supported under Australian Research Council's (ARC) *Discovery Projects* funding scheme (project number DP 120102023). Michael Thielscher is the recipient of an ARC Future Fellowship (project number FT 0991348). He is also affiliated with the University of Western Sydney.

References

Apt, K.; Blair, H. A.; and Walker, A. 1987. Towards a theory of declarative knowledge. In Minker, J., ed., *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann. chapter 2, 89–148.

Clune, J. 2007. Heuristic evaluation functions for general game playing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1134–1139. Vancouver: AAAI Press.

Fagin, R.; Halpern, J. Y.; Moses, Y.; and Vardi, M. Y. 1995. *Reasoning About Knowledge*. MIT Press.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Ostrowski, M.; Schaub, T.; and Schneider, M. 2011. Potassco: The Potsdam answer set solving collection. *AI Communications* 24(2):105–124.

Gelfond, M. 2008. Answer sets. In van Harmelen, F.; Lifschitz, V.; and Porter, B., eds., *Handbook of Knowledge Representation*, 285–316. Elsevier.

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

Haufe, S.; Schiffel, S.; and Thielscher, M. 2012. Automated verification of state sequence invariants in general game playing. *Artificial Intelligence*. To appear.

Haufe, S. 2012. *Automated Theorem Proving for General Game Playing*. Ph.D. Dissertation, Technische Universität Dresden, Germany. Available at *www.general-game-playing.de/downloads/diss-sebastian.html*.

Kuhlmann, G.; Dresner, K.; and Stone, P. 2006. Automatic heuristic construction in a complete general game player. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1457–1462. Boston: AAAI Press.

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

Love, N.; Hinrichs, T.; Haley, D.; Schkufza, E.; and Genesereth, M. 2006. General Game Playing: Game Description Language Specification. Technical Report LG–2006–01, Stanford Logic Group, Computer Science Department, Stanford University, 353 Serra Mall, Stanford, CA 94305.

Niemelä, I.; Simons, P.; and Soininen, T. 1999. Stable model semantics of weight constraint rules. In *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99), volume 1730 of Lecture*, 317–331. Springer. LNAI.

Pell, B. 1993. *Strategy Generation and Evaluation for Meta-Game Playing*. Ph.D. Dissertation, Trinity College, University of Cambridge.

Rosenhouse, J. 2009. *The Monty Hall Problem*. Oxford University Press.

Ruan, J., and Thielscher, M. 2011. The epistemic logic behind the game description language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 840–845. San Francisco: AAAI Press.

Schiffel, S., and Thielscher, M. 2007. Fluxplayer: A successful general game player. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1191–1196. Vancouver: AAAI Press.

Schiffel, S., and Thielscher, M. 2011. Reasoning about general games described in GDL-II. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 846–851. San Francisco: AAAI Press.

Thielscher, M. 2009. Answer set programming for single-player games in general game playing. In Hill, P., and Warren, D., eds., *Proceedings of the International Conference on Logic Programming (ICLP)*, volume 5649 of *LNCS*, 327–341. Pasadena: Springer.

Thielscher, M. 2010. A general game description language for incomplete information games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 994–999. Atlanta: AAAI Press.

Thielscher, M. 2011. The general game playing description language is universal. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1107–1112. Barcelona: AAAI Press.