Filtering With Logic Programs and Its Application to General Game Playing

Michael Thielscher
University of New South Wales
Australia
mit@cse.unsw.edu.au

Abstract

Motivated by the problem of building a basic reasoner for general game playing with imperfect information, we address the problem of filtering with logic programs, whereby an agent updates its incomplete knowledge of a program by observations. We develop a filtering method by adapting an existing backward-chaining and abduction method for so-called open logic programs. Experimental results show that this provides a basic effective and efficient "legal" player for general imperfect-information games.

Introduction

A general game-playing (GGP) system is one that can understand the rules of arbitrary games and use these rules to play effectively. The annual GGP competition at AAAI has been established in 2005 to foster research in this area (Genesereth, Love, and Pell 2005). While the competition in the past has focused on games in which players always know the complete game state, a recent extension (Thielscher 2011) of the formal game description language GDL also allows the description of general randomized games with imperfect and asymmetric information (Quenault and Cazenave 2007).

GDL uses normal logic program clauses to describe the rules of a game (Genesereth, Love, and Pell 2005). For games with perfect information, standard resolution techniques can be used to build a basic, so-called *legal player* that throughout a game always knows its allowed moves (Love et al. 2006). Efficient variations exist that use tailored data structures and algorithms for computing moves in classic GDL (Schkufza, Love, and Genesereth 2008; Waugh 2009; Kissmann and Edelkamp 2010; Saffidine and Cazenave 2011). But the generalization to imperfect-information games raises a fundamentally new reasoning challenge even for such a basic player. Computing with all possible states is practically infeasible except for very simple games (Parker, Nau, and Subrahmanian 2005). This is why the only two existing GGP systems described in the literature for imperfect-information GDL (Edelkamp, Federholzner, and Kissmann 2012; Schofield, Cerexhe, and Thielscher 2012) use the more practical alternative of randomly sampling states (Richards and Amir 2009; Silver and Veness 2010). But in so doing these players reason with a mere subset of all models, which is logically incorrect.

In this paper we address the problem of building a logically sound and efficient basic reasoning system for general imperfect-information games by first isolating and addressing the problem of *filtering with logic programs*: Suppose given a logic program with some hidden facts of which we have only partial knowledge. Suppose further that some consequences of this incomplete program can be observed. The question then is, what other conclusions can we derive from our limited knowledge plus the observations? This can be seen as an instance of the general notion of *filtering* as any process by which an agent updates its belief according to observations (Amir and Russell 2003).

We develop a method for filtering with logic programs under the assumption that incomplete knowledge is represented by two sets containing, respectively, known and unknown atoms, in the sense of 3-valued logic (Kleene 1952). Adapting an inference method for abduction in so-called open logic programs (Bonatti 2001a; 2001b), we show how a method for filtering can be obtained by augmenting standard backward-chaining with the computation of support.

We apply this method for filtering with logic programs to build a legal player for general game playing with imperfect information that, just like its counterpart for perfect-information games, is based on backward-chaining. We prove that the reasoner thus obtained is sound. We also show it to be complete if, as in perfect-information games, the player can observe all other players' moves. Experimental results with all imperfect-information games used at past GGP competitions demonstrate the effectiveness and efficiency of our method for a legal player to always know its allowed moves in almost all games. This in fact supports an argument that can be made for requiring that all games for competitions such as at AAAI be written so that basic backward-chaining is all that is needed to derive a player's legal moves. Interestingly, the experiments also revealed that in many existing game descriptions players are not given enough information to know the outcome after termination.

After the following brief summary of GDL, we define the problem of filtering with logic programs. We then develop a filtering method based on backward-chaining and abduction of support. We apply this to build a basic and logically sound legal player, present our experimental results, and conclude.

```
1 role(monty).  role(candidate).
2
3 init(closed(1)).  init(closed(2)).  init(closed(3)).
4 init(step(1)).
5
6 legal(monty,hide_car(D))    :- true(step(1)),
7                                true(closed(D)).
8 legal(monty,open_door(D))   :- true(step(2)),
9                                true(closed(D)),
10                               not true(car(D)),
11                               not true(chosen(D)).
12 legal(monty,noop)          :- true(step(3)).
13 legal(candidate,choose(D)) :- true(step(1)),
14                               true(closed(D)).
15 legal(candidate,noop)      :- true(step(2)).
16 legal(candidate,noop)      :- true(step(3)).
17 legal(candidate,switch)    :- true(step(3)).
18
19 sees(candidate,D) :- does(monty,open_door(D)).
20 sees(candidate,D) :- true(step(3)), true(car(D)).
21 sees(monty,move(R,M)) :- does(R,M).
22 next(car(D))    :- does(monty,hide_car(D)).
23 next(car(D))    :- true(car(D)).
24 next(closed(D)) :- true(closed(D)),
25                    not does(monty,open_door(D)).
26 next(chosen(D)) :- does(candidate,choose(D)).
27 next(chosen(D)) :- true(chosen(D)),
28                    not does(candidate,switch).
29 next(chosen(D)) :- does(candidate,switch),
30                    true(closed(D)),
31                    not true(chosen(D)).
32
33 next(step(2)) :- true(step(1)).
34 next(step(3)) :- true(step(2)).
35 next(step(4)) :- true(step(3)).
36
37 terminal :- true(step(4)).
38
39 goal(candidate,100) :- true(chosen(D)), true(car(D)).
40 goal(candidate,  0) :- true(chosen(D)), not true(car(D)).
41 goal(monty,    100) :- true(chosen(D)), not true(car(D)).
42 goal(monty,      0) :- true(chosen(D)), true(card(D)).
```

Figure 1: A description of the Monty Hall game (Rosenhouse 2009) adapted from (Schofield, Cerexhe, and Thielscher 2012).

Background: GDL-II

The science of general game playing requires a formal language for specifying arbitrary games by a complete set of rules. The declarative Game Description Language (GDL) serves this purpose (Genesereth, Love, and Pell 2005). It uses the syntax of normal logic programs (Lloyd 1987) and is characterized by these special keywords:

`role(R)`	`R` is a player
`init(F)`	feature `F` holds in the initial position
`true(F)`	feature `F` holds in the current position
`legal(R,M)`	`R` has move `M` in the current position
`does(R,M)`	player `R` does move `M`
`next(F)`	feature `F` holds in the next position
`terminal`	the current position is terminal
`goal(R,V)`	player `R` gets payoff `V`
`distinct(X,Y)`	terms `X`, `Y` are syntactically different
`sees(R,P)`	player `R` is told `P` in the next position
`random`	the random player (aka. Nature)

Originally designed for games with complete information (Genesereth, Love, and Pell 2005), GDL has recently been extended to GDL-II (for: *GDL with incomplete/imperfect information*) by the last two keywords (`sees`, `random`) to describe arbitrary (finite) games with randomized moves and imperfect information (Thielscher 2010).

Example 1 (Monty Hall) *The GDL-II rules in Fig. 1 formalize a game based on a popular problem where a car prize is hidden behind one of three doors, a goat behind the others, and where a candidate is given two chances to pick a door. The intuition behind the rules is as follows.*[1] *Line 1 introduces the players' names. Lines 3–4 define the features of the initial game state. The allowed moves are specified by the rules for* `legal`*: In step 1, Monty Hall decides where to place the car (lines 6–7) and, simultaneously, the candidate chooses a door (lines 13–14); in step 2, Monty Hall opens one of the other doors (lines 8–11) but not the one with a car behind it; finally, the candidate can either stick to the earlier choice (*`noop`*) or switch (lines 16–17). The candidate's only percepts are: the door opened by the host (line 19) and the location of the car at the end of the game (line 20). Monty Hall, on the other hand, sees all moves by the candidate (line 21). The remaining rules specify the state update (*`next`*), the conditions for the game to end (*`terminal`*), and the payoff for the players depending on whether the candidate picked the right door (*`goal`*).*

[1]For the sake of readability, we write GDL in standard Prolog syntax instead of the prefix notation used at competitions.

Formal Syntax and Semantics In order to admit an unambiguous interpretation, GDL-II 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). Stratified logic programs are known to admit a specific *standard model* (Apt, Blair, and Walker 1987), which equals its *unique stable model* (Gelfond and Lifschitz 1988). 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. Under these restrictions, any valid GDL-II game description determines a state transition system as follows.

To begin with, the derivable instances of `role(R)` define the players, and the initial state consists in the derivable instances of `init(F)`. In order to determine the legal moves of a player in any given state, this state has to be encoded first, using the keyword `true`: Let $S = \{f_1, \ldots, f_n\}$ be a state (i.e., a finite set of ground terms over the signature of G), then G is extended by the n facts

$$S^{\texttt{true}} \stackrel{\text{def}}{=} \{\texttt{true}(f_1). \quad \ldots \quad \texttt{true}(f_n).\} \qquad (1)$$

Those instances of `legal(R,M)` that follow from $G \cup S^{\texttt{true}}$ define all legal moves `M` for player `R` in position S.

In the same way, the clauses with `terminal` and `goal(R,N)` in the head define, respectively, termination and goal values *relative* to the encoding of a given position.

Determining a position update and the percepts of the players requires the encoding of both the current position and a *joint move*. Specifically, let M denote that players $r_1, \ldots, r_k$ take moves $m_1, \ldots, m_k$, then

$$M^{\texttt{does}} \stackrel{\text{def}}{=} \{\, \texttt{does}(r_1, m_1). \quad \ldots \quad \texttt{does}(r_k, m_k).\, \} \quad (2)$$

All instances of `next(F)` that follow from $G \cup M^{\texttt{does}} \cup S^{\texttt{true}}$ compose the updated position; likewise, the derivable instances of `sees(R,P)` describe what a player perceives when the given joint move is done in the given position. All this is summarized below, where "$\models$" denotes entailment wrt. the unique stable model of a stratified set of clauses.

Definition 1 *The* semantics *of a valid GDL-II game description G is the state transition system given by*

- $R = \{r\colon\ G \models \texttt{role}(r)\}$ *(player names);*
- $s_1 = \{f\colon\ G \models \texttt{init}(f)\}$ *(initial state);*
- $t = \{S\colon\ G \cup S^{\texttt{true}} \models \texttt{terminal}\}$ *(terminal states);*
- $l = \{(r, m, S)\colon\ G \cup S^{\texttt{true}} \models \texttt{legal}(r, m)\}$ *(legal moves);*
- $u(M, S) = \{f\colon\ G \cup M^{\texttt{does}} \cup S^{\texttt{true}} \models \texttt{next}(f)\}$ *(update);*
- $\mathcal{I} = \{(r, M, S, p) :\ G \cup M^{\texttt{does}} \cup S^{\texttt{true}} \models \texttt{sees}(r, p)\}$ *(players' percepts);*
- $g = \{(r, v, S)\colon\ G \cup S^{\texttt{true}} \models \texttt{goal}(r, v)\}$ *(goal values).*

GDL-II games are played using the following protocol.

1. All players receive the complete game description G.
2. Starting with s_1, in each state S each player $r \in R$ selects a legal move from $\{m\colon l(r, m, S)\}$. (The predefined role `random`, if present, chooses a legal move with uniform probability.)
3. The update function (synchronously) applies the joint move M to the current position, resulting in the new position $S' = u(M, S)$. Furthermore, the roles r receive their individual percepts $\{p\colon \mathcal{I}(r, M, S, p)\}$.
4. This continues until a terminal state is reached, and then the goal relation determines the result for all players.

Filtering with Logic Programs

The original game protocol for GDL (Love et al. 2006) differs from the above in that players are automatically informed about each other's moves in every round. Since they start off with complete knowledge of the initial state, knowing all moves implies that players have complete state knowledge throughout a game because there never is uncertainty about the facts $S^{\texttt{true}} \cup M^{\texttt{does}}$ (c.f. (1), (2)) that together with the game rules determine everything a player needs to know about the current state (such as the allowed moves as the derivable instances of `legal(R,M)`) and the next one (as the set of derivable instances of `next(F)`). The syntactic restrictions for valid game descriptions ensure that all necessary derivations are finite, so that a basic reasoner for GDL can be built based on standard backward chaining (Genesereth, Love, and Pell 2005).

In case of GDL-II, however, the situation is very different. Although players also start off with complete knowledge of the initial state, they are not automatically informed about each other's moves. But with only partial knowledge of the set of facts $M^{\texttt{does}}$, players can no longer fully determine the derivable instances of `next(F)` through standard backward chaining. This in turn means that players also lack complete knowledge of the facts $S^{\texttt{true}}$ in later states, which are needed to determine the legal moves and other crucial properties such as termination and goal values.

Rather than getting to see each other's moves, after every round players receive percepts according to the rules for `sees(R,P)`. In other words, they are informed about certain consequences that follow from the game rules and the incompletely known facts $S^{\texttt{true}} \cup M^{\texttt{does}}$. Building a basic player for GDL-II that is logically sound therefore requires a method for reasoning about the consequences of a partially known logic program and for updating this incomplete knowledge according to observations being made. Hence, we first isolate and address the more general problem of *filtering with logic programs*.

Definition 2 *Consider a normal logic program P and two sets, $\mathcal{B}$ and $\mathcal{O}$, of ground atoms called* base relations *and* observation relations, *respectively. A* filter *is a function that maps any given $\Phi \subseteq 2^{\mathcal{B}}$ and $O \subseteq \mathcal{O}$ into a set Filter$[O](\Phi) \subseteq \Phi$. A* correct *filter is one that satisfies*[2]

$$\textit{Filter}[O](\Phi) \supseteq \{B \in \Phi :\ P \cup B \models o \textit{ for all } o \in O \textit{ and } P \cup B \not\models o \textit{ for all } o \in \mathcal{O} \setminus O\}$$

A filter is complete *if these two sets are equal.*

In this definition, incomplete knowledge about the base relations is represented by a set of possible models Φ. A correct filter retains all models in Φ that entail all observations.

Example 2 *Consider the logic program below, with base relations $\mathcal{B} = \{b(1), b(2)\}$ and $\mathcal{O} = \{\text{obs}\}$.*

```
a   :- b(X).
obs :- not a.
p   :- not a.
q   :- a.
```

Suppose $\Phi = 2^{\{b(1), b(2)\}}$, that is, nothing is known about the base relations. If complete, Filter$[\{\text{obs}\}](\Phi)$ equals $\{\emptyset\}$. It follows that if obs *is observed then, under the only model left after filtering,* p *is derivable and* q *is not.*

Example 3 *Consider the GDL in Fig. 1 with the instances of* **true**`(F)` *and* **does**`(R,M)` *as base relations. Let Φ be such that all of* **true**`(closed(1))`, **true**`(closed(2))`, **true**`(closed(3))`, **true**`(chosen(1))`, **true**`(step(1))`, **does**`(candidate,noop)` *are true in all models in Φ, and let $\mathcal{O} = \{$***sees***`(candidate,2)`, **sees**`(candidate,3)`$\}$.*

*Suppose that we observe $O = \{$***sees***`(candidate,3)`$\}$, then* **does**`(monty,open(3))` *is true in all models resulting from a complete filter (cf. line 19 in Fig. 1), while* **does**`(monty,open(2))` *is false in each of them. It follows, for instance, that in all models remaining after filtering,* **next**`(closed(1))` *and* **next**`(closed(2))` *are derivable but not* **next**`(closed(3))` *(cf. line 24–25).*

[2]The definition applies to any chosen entailment relation "$\models$."

A Basic Legal Player for GDL-II

In this section we present a method for constructing a reasoner for GDL-II based on a method for filtering that operates on a compact representation of incomplete information.

Representing Incomplete Information About Facts

Since explicitly maintaining the set of possible states is practically infeasible in most games, we base our approach to filtering on a coarser but practically feasible encoding using two sets of ground atoms, $\mathcal{B}^+ \subseteq \mathcal{B}$ and $\mathcal{B}^0 \subseteq \mathcal{B}$, which respectively contain the base relations that are *known* to be true and those that *may* be true. Any such pair that satisfies $\mathcal{B}^+ \cap \mathcal{B}^0 = \emptyset$ implicitly represents the set of models

$$\Phi_{\mathcal{B}^+,\mathcal{B}^0} \overset{\text{def}}{=} \{\mathcal{B}^+ \cup B : B \subseteq \mathcal{B}^0\} \tag{3}$$

This representation allows us to base our filtering method on a derivation mechanism that has been developed in the context of so-called open logic programs (Bonatti 2001a).

Reasoning with Open Logic Programs

In the following we adapt some basic definitions and results from (Bonatti 2001a; 2001b) to our setting. Our *open logic programs* are triples $\Omega = \langle P, \mathcal{B}^+, \mathcal{B}^0 \rangle$ where P is a normal logic program and $\mathcal{B}^+, \mathcal{B}^0$ are as above. A program P' is called an *extension*[3] of Ω if $P' = P \cup \mathcal{B}^+ \cup B$ for some $B \subseteq \mathcal{B}^0$. This gives rise to two modes of reasoning:

1. *Skeptical inference*: $\Omega \models^s \varphi$ iff all stable models of all extensions P' of Ω entail φ.
2. *Credulous inference*: $\Omega \models^c \varphi$ iff some stable model of some extension P' of Ω entails φ.

We also make use of the following concepts (Bonatti 2001b):

1. A *support* for a ground atom A is a query Q obtained by unfolding A in $P \cup \mathcal{B}^+$ until all the literals in Q either occur in $\mathcal{B}^0$ or are negative.
2. A *countersupport* for a ground atom A is a set of ground literals S such that each $L \in S$ is the complement of some literal belonging to a support of A; and conversely, each support of A contains a literal whose complement is in S.

In the following, for a set S of literals we denote by S^+ the set of positive atoms in S and by S^- the set of atoms that occur negated in S. A support is *consistent* iff $S^+ \cap S^- = \emptyset$.

A Backward-Chaining Proof Method

The definitions from above form the basis of a backward-chaining derivation procedure for computing answer substitutions θ along with supports for literals L wrt. an open program $\Omega = \langle P, \mathcal{B}^+, \mathcal{B}^0 \rangle$ using the following derivation rules.

1. If $L\theta \in \mathcal{B}^+$ return θ along with support $\emptyset$.
2. If $L\theta \in \mathcal{B}^0$ return θ along with support $\{L\theta\}$.
3. If $L = \neg A$ is a negative ground literal and $\mathcal{S}$ the set of computed supports for A, return the empty substitution along with a consistent set containing the complement of some literal from each element in $\mathcal{S}$.
4. If $L = A$ is positive and unifiable with the head of a clause from P, unfold A and return the union, if consistent, of supports for all literals in the resulting query along with the combined answer substitutions.

[3]This is called a *completion* in (Bonatti 2001a), which however clashes with another concept so named earlier (Shepherdson 1984).

Recall, for instance, the short program from Example 2 and suppose $\mathcal{B}^+ = \emptyset$ and $\mathcal{B}^0 = \{b(1), b(2)\}$. Query $b(X)$ admits two computed supports: $S = \{b(1)\}$ with $\theta = \{X \setminus 1\}$, and $S = \{b(2)\}$ with $\theta = \{X \setminus 2\}$. Hence, the computed countersupport for query a is $\{\neg b(1), \neg b(2)\}$, which in turn is the (only) support for *obs* under the given sets $\mathcal{B}^+, \mathcal{B}^0$.

The above derivation rules are a subset of the calculus defined in (Bonatti 2001a; 2001b) but constitute a complete and decidable derivation procedure if the underlying logic program is syntactically restricted.

Proposition 1 *Let $\Omega = \langle P, \mathcal{B}^+, \mathcal{B}^0 \rangle$ be an open logic program with a finite Herbrand base and stratified program P.*

1. *Every computed support θ, S for a query Q satisfies $\langle P, \mathcal{B}^+ \cup S^+, \mathcal{B}^0 \setminus S^- \rangle \models^s Q\theta$.*
2. *If $\langle P, \mathcal{B}^+, \mathcal{B}^0 \rangle \models^c Q\tau$ for some τ, then there exists a computed support θ, S for Q with θ more general than τ.*

In the following, by $\Omega \vdash_{\theta,S} A$ we denote that substitution θ together with some S is a computed support for atom A wrt. open logic program Ω. In particular, $\Omega \vdash_{\theta,\emptyset} A$ means that $A\theta$ follows without additional support, i.e., is necessarily true in all extensions and hence skeptically entailed by Ω.

Filtering Based on Backward Chaining

Next, we use the backward chaining-based method for open logic programs to define a basic method for filtering with logic programs. In the following, by $\mathtt{Supp}(Q)$ we denote the set of all computed supports for query Q. Consider a normal logic program P; sets $\mathcal{B}^+, \mathcal{B}^0$; and a set $O \subseteq \mathcal{O}$ of observations. We compute $\mathit{Filter}[O](\Phi_{\mathcal{B}^+,\mathcal{B}^0})$ as two sets $\mathcal{B}^+_{\mathit{new}}$ and $\mathcal{B}^0_{\mathit{new}}$ as follows.

$$\mathcal{B}^+_{\mathit{new}} = \mathcal{B}^+ \cup \left(\bigcup_{o \in O} \bigcap_{S \in \mathtt{Supp}(o)} S^+ \cup \bigcup_{o \in \mathcal{O} \setminus O} \bigcap_{S \in \mathtt{Supp}(\neg o)} S^+ \right)$$

$$\mathcal{B}^0_{\mathit{new}} = (\mathcal{B}^0 \setminus \mathcal{B}^+_{\mathit{new}}) \setminus \left(\bigcup_{o \in O} \bigcap_{S \in \mathtt{Supp}(o)} S^- \cup \bigcup_{o \in \mathcal{O} \setminus O} \bigcap_{S \in \mathtt{Supp}(\neg o)} S^- \right)$$

Put in words, for each observation o made (resp. not made) we compute all supports (resp. all supports for $\neg o$) and then "strengthen" $\mathcal{B}^+, \mathcal{B}^0$ by every literal that is contained in all supports. More precisely, if a literal occurs positively in every support for some o (resp. $\neg o$), then it is added to $\mathcal{B}^+$ and removed from $\mathcal{B}^0$. Also removed from $\mathcal{B}^0$ are the literals that occur negatively in every support for some o (resp. $\neg o$).

Example 4 *Recall again the program from Example 2. As we have seen, when $\mathcal{B}^+ = \emptyset$ and $\mathcal{B}^0 = \{b(1), b(2)\}$ then the query* obs *has one support, namely, $\{\neg b(1), \neg b(2)\}$. This yields $\mathcal{B}^+_{\mathit{new}} = \emptyset$ and $\mathcal{B}^0_{\mathit{new}} = \emptyset$. On the other hand, consider the query $\neg$*obs. *It has two supports, $\{b(1)\}$ and $\{b(2)\}$. Their intersection being empty implies $\mathcal{B}^+_{\mathit{new}} = \mathcal{B}^+$ and $\mathcal{B}^0_{\mathit{new}} = \mathcal{B}^0$, i.e., nothing new is learned by* not *seeing* obs.

Proposition 2 *Under the assumptions of Proposition 1, the filter defined above is correct.*

Proof: By Definition 2 we need to show that $B \in \Phi_{\mathcal{B}^+_{new},\mathcal{B}^0_{new}}$ if $B \in \Phi_{\mathcal{B}^+,\mathcal{B}^0}$ and $P \cup B \models o$ for $o \in O$ while $P \cup B \not\models o$ for $o \in \mathcal{O} \setminus O$. So suppose the latter are all true, then for each $o \in O$ (and each $o \in \mathcal{O} \setminus O$, resp.) there must be a computed support $S \in \mathtt{Supp}(o)$ (resp., $S \in \mathtt{Supp}(\neg o)$) such that $S^+ \subseteq B$ and $S^- \cap B = \emptyset$. By construction of $\mathcal{B}^+_{new}, \mathcal{B}^0_{new}$ this implies $\mathcal{B}^+_{new} \subseteq B \subseteq \mathcal{B}^+_{new} \cup \mathcal{B}^0_{new}$. Hence, $B \in \Phi_{\mathcal{B}^+_{new},\mathcal{B}^0_{new}}$ according to (3). $\square$

Since the compact representation of incomplete knowledge via (3) does not support reasoning by disjunction, the filter is necessarily incomplete. Recall, for instance, the second case in Example 4. Not observing *obs* means that $b(1)$ or $b(2)$ must be true. Hence, model $\emptyset$ could be filtered out but is not because no two sets $\mathcal{B}^+, \mathcal{B}^0$ can encode $\Phi = \{\{b(1)\}, \{b(2)\}, \{b(1), b(2)\}\}$ via (3).

A Basic Update Method

The method for filtering with logic programs forms the core of our approach to building a basic reasoner for GDL-II. The syntactic restrictions for GDL-II ensure that the underlying open logic program satisfies the conditions Propositions 1 and 2. This will guarantee that the knowledge the player keeps in $\mathcal{B}^+, \mathcal{B}^0$ is always correct.

The procedure for maintaining the player's incomplete knowledge about a state is as follows, where G denotes the GDL-II description of a game whose semantics is given as per Definition 1; and where *my_role* $\in R$ is the role assigned to the player.

1. $\mathcal{B}^+ := \{\mathtt{true(F)}\theta : \langle G, \emptyset, \emptyset\rangle \vdash_{\theta,\emptyset} \mathtt{init(F)}\}$; $\mathcal{B}^0 := \emptyset$
2. In every round,

 2.1 Compute the possible legal moves of all other roles:

$$\mathcal{L} := \{\,\mathtt{(R,M)}\,\theta : \langle G, \mathcal{B}^+, \mathcal{B}^0\rangle \vdash_{\theta,S} \mathtt{legal(R,M)}, \mathtt{R} \neq \textit{my\_role}\}$$

 2.2 Let *my_move* be the selected move of the basic player and *my_percepts* the player's percepts.

 – Let $\mathcal{B}^+ := \mathcal{B}^+ \cup \{\mathtt{does}(\textit{my\_role}, \textit{my\_move})\}$
 $\mathcal{B}^0 := \mathcal{B}^0 \cup \{\mathtt{does(R,M)} : \mathtt{(R,M)} \in \mathcal{L}\}$

 – Now, let

$$\mathcal{O} := \{\mathtt{sees}(\textit{my\_role},\mathtt{P})\,\theta : \langle G, \mathcal{B}^+, \mathcal{B}^0\rangle \vdash_{\theta,S} \mathtt{sees}(\textit{my\_role},\mathtt{P})\}$$
$$O := \{\mathtt{sees}(\textit{my\_role}, p) : p \in \textit{my\_percepts}\}$$

 and compute $\mathcal{B}^+_{new}, \mathcal{B}^0_{new}$ as the result of filtering $\mathcal{B}^+, \mathcal{B}^0$ by O wrt. G and $\mathcal{O}$.

 – The knowledge about the next state is obtained as

$$\mathcal{B}^+ := \{\mathtt{true(F)}\,\theta : \langle G, \mathcal{B}^+_{new}, \mathcal{B}^0_{new}\rangle \vdash_{\theta,\emptyset} \mathtt{next(F)}\}$$
$$\mathcal{B}^0 := \{\mathtt{true(F)}\,\theta : \langle G, \mathcal{B}^+_{new}, \mathcal{B}^0_{new}\rangle \vdash_{\theta,S} \mathtt{next(F)}\} \setminus \mathcal{B}^+$$

3. The player knows that the game has terminated in case $\langle G, \mathcal{B}^+, \mathcal{B}^0\rangle \vdash_{\varepsilon,\emptyset} \mathtt{terminal}$.

Put in words, the player starts with complete information about the initial state (step 1). In every round, the player's knowledge is characterized by the skeptical consequences of the open logic program consisting of the game rules plus the incomplete knowledge $\mathcal{B}^+, \mathcal{B}^0$; specifically, this allows us to determine the player's own known legal moves as

$$\{\mathtt{M}\theta : \langle G, \mathcal{B}^+, \mathcal{B}^0\rangle \vdash_{\theta,\emptyset} \mathtt{legal}(\textit{my\_role},\mathtt{M})\}^4$$

The incomplete knowledge also allows us to compute credulous consequences, in particular the possible legal moves of all other players (step 2.1). For the update of $\mathcal{B}^+, \mathcal{B}^0$ (step 2.2), we first add to $\mathcal{B}^+$ the knowledge of the player's own move and to $\mathcal{B}^0$ the possible moves by the opponents. This allows us to determine the range of possible observations, $\mathcal{O}$, in order then to filter the player's knowledge by actual observations O. Finally, the player's knowledge of the updated state is determined as the skeptically (for $\mathcal{B}^+$) and credulously (for $\mathcal{B}^0$) entailed instances of `next(F)`.

The incompleteness of the filtering implies that the reasoner for GDL-II thus defined is incomplete in general. It is, however, easy to show that it is complete in case the only `sees`-rule for a player is

```
sees(player,move(R,M)) :- does(R,M).
```

This is so because under this rule the only support for an instance `sees(player,move(r,m))` is `does(r,m)`, and the only countersupport in case the observation is not made is $\neg$`does(r,m)`. Hence, the filter will add the former to $\mathcal{B}^+$ and remove all of the latter from $\mathcal{B}^0$. The update procedure will then result in complete knowledge whenever the player starts with complete knowledge.

Experimental Results

Because the representation of incomplete knowledge and the backward chaining-based filtering are in themselves incomplete, we have run experiments to test both the effectiveness and the efficiency of our method. We have used a simple implementation in the form of a vanilla meta-interpreter in Prolog. We have run experiments with all games that were played at past general game playing competitions with imperfect-information track.[5] We ran 1000 simulated random matches each to test whether the legal player always knew its legal moves, and also—in case it did—whether it had sufficient knowledge to know at the end that the game must have terminated and to derive its own goal value.

The results are summarized in Table 1. For games with two or more non-`random` roles that are not symmetric, we have run the basic player for each of them as shown. The times given are the average time, in seconds (cpu time), that the player took for 1000 rounds of updating its incomplete state knowledge on a desktop computer with a 2.66 GHz CPU and 8GB memory running Eclipse Prolog. Overall, the results demonstrate both the effectiveness and the efficiency of our basic backward-chaining method.

[4]The player knows that it doesn't know all its legal moves if some instance `legal(`*my_role*`,M)` can be derived only with non-empty support, i.e., is credulously but not skeptically entailed.

[5]The 1st German Open 2011, see `http://fai.cs.uni-saarland.de/kissmann/ggp/go-ggp`; and the 1st Australian Open 2012, see `https://wiki.cse.unsw.edu.au/ai2012/GGP`

Game	Legal	Terminal	Goal	Time
Backgammon	✓	✓	✓	8.84
Banker/Thief (role 1)	✓	✓	no	0.42
Banker/Thief (role 2)	✓	✓	no	0.69
Battleships in Fog	no	–	–	–
Battleships in Fog*	✓	no	no	930.04
Blackjack	no	–	–	–
Hidden Connect	✓	✓	no	4.08
Hold your Course II	✓	✓	✓	2.05
Krieg-Tictactoe 5x5	no	–	–	–
Mastermind448	✓	✓	no	0.58
Minesweeper (role 1)	✓	✓	✓	1.56
Minesweeper (role 2)	✓	no	no	199.82
Numberguessing	✓	✓	no	1.53
Monty Hall (role 1)	✓	✓	✓	0.21
Monty Hall (role 2)	✓	✓	✓	0.21
Small Dominion 2	✓	✓	✓	12376.75
Transit (role 1)	✓	no	no	4.18
Transit (role 2)	✓	no	no	5.76
vis_Pacman (role 1,2)	✓	no	no	(706.45)
vis_Pacman (role 3)	✓	no	✓	32.12

Table 1: Experimental results testing the basic player's ability to always know its legal moves, whether a game has terminated, and what its own goal value is in the end.

Knowledge of Legal Moves The experiments showed that the basic player always knows its legal moves in almost all of the games. An exception is Krieg-Tictactoe, where the uncertainty about the legal moves is due to this rule:

```
legal(P,mark(M,N)) :-
  role(P), true(cell(M,N,C)), distinct(C,P).
```

While a player `P` knows all cells occupied with its own symbol, that is, for which `true(cell(M,N,P))` holds, it does not know whether other cells are blank or have been marked by the opponent. Lacking the ability to reason disjunctively means that in this case there is no ground instance of `true(cell(M,N,C))` that is known to satisfy the body of the rule from above.

For a similar reason, the basic player fails to determine its legal moves in Blackjack. In the original version of Battleships in Fog, the reason why the player is uncertain about its legal moves lies in these (slightly simplified) rules:

```
sees(admiral,position(admiral,A,B)) :-
    does(random,setup(A,B,C,D)).
sees(commodore,position(commodore,C,D)) :-
    does(random,setup(A,B,C,D)).

next(position(admiral,A,B)) :-
    does(random,setup(A,B,C,D)).
next(position(commodore,C,D)) :-
    does(random,setup(A,B,C,D)).
```

Here, in a single random move two ships get positioned, one for each player. Despite the given information, however, the legal player is unable to determine the location of its own ship because the observation of some arguments of `setup(A,B,C,D)` does not entail a fully known instance of this move, and hence nothing can be learnt from filtering through an observation like, for example, `sees(admiral,position(admiral,1,2))`. For the sake of experimentation, we have defined a variant of the original game (marked by * in Table 1) where the random move is broken into two moves. With this simple modification, the basic player is able to determine its legal moves throughout that game.

Knowledge of Termination and Goal Values Somehow surprisingly, in a number of games the legal player was not able to derive that a game has terminated and what its goal value was. An inspection of the game rules showed that this is due to the fact that the game rules as such provide players with insufficient information in this regard. Hence, there is an argument to be made for requiring that games in competitions should always be defined so that the percepts suffice for every player to determine their outcome at the end.

Times The runtimes depicted in Table 1 show that basic backward chaining in general is an efficient approach for filtering observations and inferring the updated incomplete state knowledge in a basic player for GDL-II. A notable exception was the 3-person, imperfect-information version of the Pacman game when taking the role of either of the two "ghosts." In this game, the reasoner always slowed down significantly after around move 50 (where the maximal length of that game is 100 moves), and we had to interrupt the experiments a few moves later. We re-ran the experiments with a version of the basic player that only filters through the observations actually made instead of also filtering through all non-percepts. The results given in Table 1 for "vis_Pacman (role 1,2)" were obtained for this simplified legal player.

Conclusion

We have developed a method for filtering with logic programs and applied it to build a basic legal player for GDL-II based on backward-chaining. Our notion of filtering is similar to (Amir and Russell 2003; Shirazi and Amir 2011); in their case a dynamic system is not described by logic program rules but in the Situation Calculus. For our backward-chaining filtering method we have adapted results for open logic program from (Bonatti 2001a; 2001b). Our experiments showed that the method is sufficiently efficient in almost all games from previous GGP competitions. It is worth stressing that even in games where the reasoner is not fast enough to be used at every node of a search tree, it can and in fact should be applied at least at the beginning in order to guarantee that the player submits a legal move. Our method also proved effective in almost all games, which supports an argument that can be made for it to be generally desirable that all GDL-II games for competitions are written so that backward chaining augmented by support computation suffices to always determine a player's legal moves, as in our reformulation of Battleships in Fog.

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

References

Amir, E., and Russell, S. 2003. Logical filtering. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 75–82. Acapulco, Mexico: Morgan Kaufmann.

Apt, K.; Blair, H.; 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.

Bonatti, P. 2001a. Reasoning with open logic programs. In Eiter, T.; Faber, W.; and Trusczynski, M., eds., *Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR)*, volume 2173 of *LNCS*, 147–159. Vienna, Austria: Springer.

Bonatti, P. 2001b. Resolution with skeptical stable model semantics. *Journal of Automated Reasoning* 156(1):391–421.

Edelkamp, S.; Federholzner, T.; and Kissmann, P. 2012. Searching with partial belief states in general games with incomplete information. In Glimm, B., and Krüger, A., eds., *Proceedings of the German Annual Conference on Artificial Intelligence (KI)*, volume 7526 of *LNCS*, 25–36. Saarbrücken, Germany: Springer.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In Kowalski, R., and Bowen, K., eds., *Proceedings of the International Joint Conference and Symposium on Logic Programming (IJCSLP)*, 1070–1080. Seattle, OR: MIT Press.

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

Kissmann, P., and Edelkamp, S. 2010. Instantiating general games using Prolog or dependency graphs. In Dillmann, R.; Beyerer, J.; Hanebeck, U.; and Schultz, T., eds., *Proceedings of the German Annual Conference on Artificial Intelligence (KI)*, volume 6359 of *LNCS*, 255–262. Karlsruhe, Germany: Springer.

Kleene, S. 1952. *Introduction to Metamathematics*. Van Nostrand, New York.

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

Lloyd, J. 1987. *Foundations of Logic Programming*. Series Symbolic Computation. Springer, second, extended edition.

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. Available at: `games.stanford.edu`.

Parker, A.; Nau, D.; and Subrahmanian, V. S. 2005. Game-tree search with combinatorially large belief states. In Kaelbling, L., and Saffiotti, A., eds., *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 254–259.

Quenault, M., and Cazenave, T. 2007. Extended general gaming model. In *Computers Games Workshop*, 195–2004.

Richards, M., and Amir, E. 2009. Information set sampling in general imperfect information positional games. In *Proceedings of the IJCAI Workshop on General Intelligence in Game-Playing Agents (GIGA)*, 59–66.

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

Saffidine, A., and Cazenave, T. 2011. A forward chaining based game description language compiler. In *Proceedings of the IJCAI Workshop on General Intelligence in Game-Playing Agents (GIGA)*, 69–75.

Schkufza, E.; Love, N.; and Genesereth, M. 2008. Propositional automata and cell automata: Representational frameworks for discrete dynamic systems. In *Proceedings of the Australasian Joint Conference on Artificial Intelligence*, volume 5360 of *LNCS*, 56–66. Auckland: Springer.

Schofield, M.; Cerexhe, T.; and Thielscher, M. 2012. HyperPlay: A solution to general game playing with imperfect information. In *Proceedings of the AAAI Conference on Artificial Intelligence*. Toronto: AAAI Press.

Shepherdson, J. C. 1984. Negation as failure: A comparison of Clark's completed data base and Reiter's closed world assumption. *Journal of Logic Programming* 1:51–79.

Shirazi, A., and Amir, E. 2011. First-order logical filtering. *Artificial Intelligence* 175(1):193–219.

Silver, D., and Veness, J. 2010. Monte-Carlo planning in large POMDPs. In Lafferty, J.; Williams, C.; Shawe-Taylor, J.; Zemel, R.; and Culotta, A., eds., *Advances in Neural Information Processing (NIPS)*, 2164–2172.

Thielscher, M. 2010. A general game description language for incomplete information games. In Fox, M., and Poole, D., eds., *Proceedings of the AAAI Conference on Artificial Intelligence*, 994–999.

Thielscher, M. 2011. The general game playing description language is universal. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1107–1112.

Waugh, K. 2009. Faster state manipulation in general games using generated code. In *Proceedings of the IJCAI Workshop on General Intelligence in Game-Playing Agents (GIGA)*, 91–97.