GDL-III: A Proposal to Extend the Game Description Language to General Epistemic Games

Michael Thielscher[1]

Abstract. We propose an extension of the standard game description language for general game playing to include *epistemic games*, which are characterised by rules that depend on the knowledge of players. A single additional keyword suffices to define *GDL-III*, a general description language for games with *imperfect information* and *introspection*. We present an Answer Set Program for automatically reasoning about GDL-III games. Our extended language along with a suitable basic reasoning system can also be used to formalise and solve general epistemic puzzles.

Introduction. A general game player is a system that can understand the rules of new strategy games at runtime and learn to play these games effectively without human intervention [4]. The game description language GDL has become the standard for describing the rules of games to general game-playing systems [3].

The extension GDL-II has been developed with the aim to include general *imperfect information* games [5]. While these games require general game players to reason about their knowledge of the state of a game, GDL-II does not support the specification of games with *epistemic* goals or, more generally, with rules that depend on the epistemic state of players. As an example, consider the game NUMBER-GUESSING from the GDL-II track at the AI'12 general game playing competition,[2] in which the goal for a single player is to repeatedly ask yes/no questions to determine an initially unknown number. However, the player can win merely by guessing correctly; the game description language does not provide means to specify, as a necessary winning condition, that the player must actually *know* the number. Another example of games beyond the expressiveness of GDL-II are so-called Russian Card problems [2], where the goal of two cooperating players is to inform each other about their hands through public announcements without a third player being able to learn anything from their communication.

The purpose of this paper is to propose a formal language suitable for general game playing that supports the encoding of game rules which depend on the epistemic states of the players. We will show that a single additional keyword suffices to define *GDL-III*, a general description language for games with *Imperfect Information* and *Introspection*. The new keyword can be used to express individual knowledge, which can also be nested (e.g. player A knows that her cards are known to player B) as well as common knowledge (e.g. player C does not know of any card held by another player and everyone knows this—and also everyone knows that everyone knows etc). While the main purpose of our language extension is to allow for the description of epistemic games for the purpose of general game playing, we will furthermore demonstrate how GDL-III can be used to encode, and automatically solve, epistemic puzzles like Cheryl's Birthday, which recently acquired public fame [1].

Game Descriptions. The declarative Game Description Language (GDL) uses a prefix-variant of the syntax of normal logic programs along with the following special keywords, the last two of which have been added in GDL-II for imperfect-information games.

`(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`
`(sees R P)`	player `R` is told `P` in the next position
`random`	the random player (aka. Nature)

GDL-III = GDL-II + Introspection. We define GDL-III as an extension of GDL-II by a new keyword for *introspection*,

`(knows R P)`	player `R` knows `P` in the current position
`(knows P)`	`P` is common knowledge

along with the following additional restrictions on syntactically valid game descriptions G:

1. `knows` only occurs in the body of clauses, and neither `role` nor `init` depend on `knows`.
2. There is a total ordering $>$ on all predicate symbols `P` that occur as argument of `knows` in G such that `P` $>$ `Q` whenever `P` itself depends on `(knows R Q)` or `(knows Q)` in G.
3. If `P` occurs as argument of `knows` in G then `P` does not depend on `does` in G.

Note the use of *reification*, whereby a defined predicate, `P`, occurs as an argument of another predicate. *Nested* knowledge can be expressed with the help of auxiliary predicates, for example, `(knows a kbp)` along with `(<= kbp (knows b p))`. The syntactical restrictions then ensure that nested knowledge is hierarchical (condition 2) and confined to state-dependent properties (condition 3). The former simply disallows circular definitions, as in `(<= p (knows q))`, `(<= q (knows p))`, while the latter restriction ensures that knowledge only refers to the current state and not to future actions.

[1] UNSW Australia, email: mit@unsw.edu.au. This research was supported by the Australian Research Council under grant no. DP150101351. The author is also affiliated with the University of Western Sydney.

[2] see `ai2012.web.cse.unsw.edu.au/ggp.html`

```
1  (role player) (role random)
2
3  (number 1)  ...  (number 32)
4  (succ 0 1)  ...  (succ 31 32)
5  (<= (less ?m ?n) (or (succ ?m ?n)
6                       ((succ ?m ?l) (less ?l ?n))))
7  (init (step 0))
8
9  (<= (legal random (choose ?n))
10     (number ?n) (true (step 0)))
11 (<= (legal random noop) (not (true (step 0))))
12
13 (<= (legal player noop) (true (step 0)))
14 (<= (legal player (ask_if_less ?n))
15     (number ?n) (not (true (step 0))))
16 (<= (sees player yes)
17     (does player (ask_if_less ?n))
18     (true (secret ?m)) (less ?m ?n))
19
20 (<= (next (secret ?n)) (does random (choose ?n)))
21 (<= (next (secret ?n)) (true (secret ?n)))
22 (<= (next (step ?n))   (true (step ?m)) (succ ?m ?n))
23
24 (<= (num ?n) (true (secret ?n)))
25 (<= (kows_the_number ?r) (role ?r) (knows ?r (num ?n)))
26
27 (<= (terminal) (or (knows_the_number player)
28                    (true (step 12))))
29 (<= (goal player 100) (knows_the_number player))
30 (<= (goal player   0) (not (knows_the_number player)))
```

Example 1 *The rules on the top of this page formalise a GDL-III variant of* NUMBERGUESSING. *Line 1 introduces the roles. Lines 3–5 define some auxiliary predicates. Line 7 provides the initial game state. The moves are specified by the rules for* `legal`*: In the first round, a number between 1 and 32 is randomly chosen (lines 9–10). The player can then repeatedly ask yes/no questions (lines 14–15). The player's percepts are truthful replies to these questions (lines 16–18). The rules for* `next` *specify the state update (lines 20–22). The objective of the game is formulated as a* knowledge goal *(lines 24–30): the game ends (*`terminal`*) and the player wins (*`goal` *value 100) upon* knowing *the secret number.*

Automated Reasoning for GDL-III. In order to be able to play games specified in the extended game description language, any general game-playing system needs an automated reasoning component for evaluating the rules to determine legal moves and compute state updates. In this section, we build on previous uses of Answer Set Programming (ASP) to develop the foundations for automated reasoners for GDL-III. The game description language and ASP have essentially the same basic semantics given by the unique stable model (aka. answer set) of a stratified set of logic program rules. Since the evaluation of knowledge conditions depends on previous moves and percepts, all state-dependent predicates in a game description are augmented by two additional arguments so that a single ASP can be used to reason about different legal play sequences, `seq(S)`, and different time points, `time(T)`. For example, the ASP-encoding of rule 25 in NUMBERGUESSING is

```
knows_the_number(R,S,T) :-
  seq(S), time(T),
  role(R), knows(R,num(N),S,T).
```

We follow the convention of using natural numbers for time, so that `(init F)` is replaced by `true(F,S,0)` and `(next F)` by `true(F,S,T+1)`. Knowledge conditions are evaluated on the basis of the (in-)distinguishability of legal play sequences, according to which players can distinguish any two sequences in which at least one of their preceding moves or percepts differ. Otherwise, the two sequences are indistinguishable (predicate `ind`):

```
distinguishable(R,S1,S2,N) :- time(N), T<N,
  does(R,M1,S1,T), does(R,M2,S2,T), M1!=M2.
distinguishable(R,S1,S2,N) :- time(T), T<=N,
  sees(R,P,S1,T), not sees(R,P,S2,T).

ind(R,S1,S2,N) :- role(R), seq(S1), seq(S2),
  time(N), not distinguishable(R,S1,S2,N).

indtrans(S1,S1,N) :- seq(S1), time(N).
indtrans(S1,S3,N) :- ind(R,S1,S2,N),
                     indtrans(S2,S3,N).
```

The last two clauses above encode the transitive closure of the indistinguishability relation over all roles. On this basis, conditions `(knows R` $p(\vec{x})$`)` can be evaluated according to the schema

```
knows(R,$p(\vec{x})$,S,T) :- $p(\vec{x}$,S,T), not np(R,$\vec{x}$,S,T).
np(R,$\vec{x}$,S,T) :- ind(R,S,S1,T), not $p(\vec{x}$,S1,T).
```

Put in words, if `S` is the actual play sequence at time `T`, then player `R` knows that $p(\vec{x})$ just in case $p(\vec{x})$ actually holds and it is *not* the case that (predicate `np`) there is a sequence `S1` that `R` cannot distinguish from `S` and in which $p(\vec{x})$ does not hold. A property $p(\vec{x})$ is *common* knowledge if $p(\vec{x})$ holds in all sequences that are in the transitive closure of the indistinguishability relation across all players:

```
knows($p(\vec{x})$,S,T) :- $p(\vec{x}$,S,T), not np($\vec{x}$,S,T).
np($\vec{x}$,S,T) :- indtrans(S,S1,T), not $p(\vec{x}$,S1,T).
```

Solving Epistemic Puzzles with GDL-III. Epistemic puzzles are characterised by multiple agents starting off with imperfect, and in many cases asymmetric, knowledge. They draw further conclusions by logical reasoning about each other's (lack of) knowledge in the course of a short sequence of actions, which often merely consists in repeated public announcements of an agent's ignorance until everyone has perfect knowledge. An example is the following puzzle, which recently acquired public fame [1].

Example 2 *Albert and Bernard want to know Cheryl's birthday. She draws a list with possible dates and then tells them separately the correct month and day, respectively. A dialogue follows in which Albert first says that he doesn't know the birthday* and *that he knows that Bernard doesn't know either, then Bernard says that he now knows the date, and after that Albert announces that he does so too.*

Puzzles like this that centre around knowledge of the knowledge of other agents can be formalised using GDL-III in such a way that *every* legal playout corresponds to a solution and vice versa. These puzzles can then be automatically solved by a mere GDL-III legal reasoner like the ASP-based reasoning system from above.

REFERENCES

[1] K. Chang, 'A math problem from Singapore goes viral: When is Cheryl's birthday?', *The New York Times*, (14 Apr 2015).

[2] A. Cordón-Franco, H. van Ditmarsch, D. Duque, and F. Soler-Toscano, 'A colouring protocol for the generalized Russian cards problem', *Journal of Theoretical Computer Science*, **495**, 81–95, (2013).

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

[4] M. Genesereth and M. Thielscher, *General Game Playing*, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool, 2014.

[5] S. Schiffel and M. Thielscher, 'Representing and reasoning about the rules of general games with imperfect information', *Journal of Artificial Intelligence Research*, **49**, 171–206, (2014).