# A Multiagent Semantics for the Game Description Language

Stephan Schiffel and Michael Thielscher

Technical University of Dresden
Dresden, Germany
{stephan.schiffel,mit}@inf.tu-dresden.de

**Abstract.** The Game Description Language (GDL) has been developed for the purpose of formalizing game rules. It serves as the input language for general game players, which are systems that learn to play previously unknown games without human intervention. In this paper, we show how GDL descriptions can be intepreted as multiagent domains and, conversely, how a large class of multiagent environments can be specified in GDL. The resulting specifications are declarative, compact, and easy to understand and maintain. At the same time they can be fully automatically understood and used by autonomous agents who intend to participate in these environments. Our main result is a formal characterization of the class of multiagent domains that serve as formal semantics for—and can be described in—the Game Description Language.

## 1 Introduction

A novel and challenging research problem for Artificial Intelligence, General Game Playing is concerned with the development of systems that learn to play a previously unknown game solely on the basis of the rules. The Game Description Language (GDL) [1] has been developed to formalize the rules of any finite, information-symmetric $n$-player game in such a way that the description can be automatically processed by a general game player [2]. As a declarative language, GDL supports specifications that are modular and easy to develop, understand, and maintain. While the basic semantics for GDL is grounded in standard logic, the language uses several pre-defined predicates as keywords, whose intended meaning is only informally described in [1].

In this paper, we show that GDL can be understood as a specification language for a large class of multiagent environments. This allows for formalizing the physics and laws that govern an arbitrary domain in such a way that agents can automatically understand the rules and thus know how to participate in this environment. There is a variety of potential applications for machine processable descriptions of multiagent environments: the rules of an e-marketplace can be made accessible to agents, the interface of interactive Internet platforms for software agents can be formally described, and agent competitions can be run without revealing detailed problem specifications in advance. In each of these cases, an autonomous agent—or a team of agents—can learn how to participate in a new or modified environment without the need to be (re-)programmed for each specific case. Because GDL uses a decidable subset of logic programming,

J. Filipe, A. Fred, and B. Sharp (Eds.): ICAART 2009, CCIS 67, pp. 44–55, 2010.

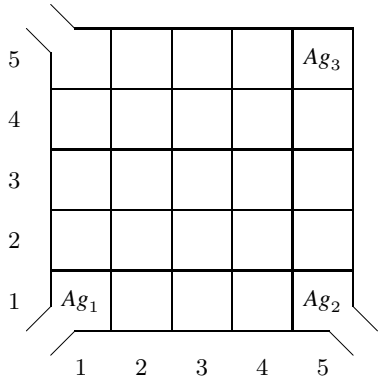


**Fig. 1.** A simple multiagent domain: two "guard" agents $Ag_1$ and $Ag_2$ shall cooperatively try to catch $Ag_3$, whose goal in turn is to escape via one of the three exits at locations $(1,1)$, $(5,1)$, and $(1,5)$. All agents act synchronously and can move horizontally or vertically to an adjacent position. $Ag_3$ is caught when it ends up in the same location as $Ag_1$ or $Ag_2$, or when it crosses path with one of them in a simultaneous move.

autonomous agents require just a simple, standard reasoning module to be able to understand and effectively process a given set of rules. Moreover, if an agent environment is specified in GDL, successful general game playing systems such as [3,4,5,6] can be readily employed as intelligent agents for these environments.

The main result in this paper is the definition of a formal class of multiagent environments which can be expressed in GDL and, conversely, which can be used to provide a semantics for any GDL game description. As a by-product we thus obtain a formal semantics for the special, pre-defined keywords in GDL.

The rest of the paper is organized as follows. In Section 2, we formally define the class of deterministic, synchronous multiagent environments. In Section 3, we show how these can be axiomatically described in GDL, and in the section that follows, we present the converse result by showing how all GDL games can be interpreted as a deterministic, asynchronous multiagent environment. We conclude in Section 5.

## 2 Multiagent Environments

As the running example in this paper, we will consider the multiagent domain depicted in Figure 1. As a discrete environment it can be formally described as a finite state transition system. However, even though it is obviously just a toy-size example, it has a considerably large state space, rendering an explicit encoding difficult—and practically impossible for even slightly larger environments. Fortunately it is possible to exploit the fact that any natural and realistic multiagent environment has an internal structure, which allows one to describe its dynamics with the help of symbols that represent individual components. Our example domain, for instance, can be formally described using the following symbolic expressions: $Ag_1, Ag_2, Ag_3$, representing the three agents; $At(r,x,y)$, where $r \in \{Ag_1, Ag_2, Ag_3\}$ and $x, y \in \{1,\ldots,5\}$, representing

the position of each agent; and $\mathit{Move}(d)$, where $d \in \{\mathit{North}, \mathit{South}, \mathit{East}, \mathit{West}\}$, along with *Stay* and *Exit*, representing the possible actions.

Based on a suitable collection of symbols, multiagent domains can be formally described as follows.

**Definition 1.** *Let $\Sigma$ be a countable set of ground (i.e., variable-free) symbolic expressions. A (discrete, synchronous, deterministic)* multiagent environment *is a structure*

$$(R, s_1, t, l, u, g)$$

*where*

- $R \subseteq \Sigma$ *finite (the agents, or* roles*);*
- $s_1 \subseteq \Sigma$ *finite (the* initial state*);*
- $t \subseteq 2^\Sigma$ *(the* terminal states*);*
- $l \subseteq R \times \Sigma \times 2^\Sigma$ *(the action preconditions, or* legality relation*);*
- $u : (R \mapsto \Sigma) \times 2^\Sigma \mapsto 2^\Sigma$ *(the transition function, or* update function*);*
- $g \subseteq R \times \mathbb{N} \times 2^\Sigma$ *(the utility, or* goal relation*).*

*Here, $2^\Sigma$ denotes the set of all* finite *subsets of $\Sigma$, and for any $r \in R$ and $S \in 2^\Sigma$, $l(r, a, S)$ holds for finitely many $a \in \Sigma$.*

This definition deserves some explanation. For the sake of simplicity, the symbolic expressions are not categorized—there is no formal distinction between symbols for objects, state components, actions, etc. For practical purposes, it is important that states are finitely representable; hence, while possibly infinitely many symbols give rise to infinitely many states, a state itself is an element of the set of all *finite* subsets of the given symbols. The legality relation $l(r, a, S)$ defines $a$ to be a legal action for agent $r$ in state $S$. Again for the sake of practical usability, it is assumed that every agent in every state has only finitely many possible actions. The update function takes an action for each agent and (synchronously) applies the joint actions to a current state, resulting in the updated state. For the sake of simplicity, we take natural numbers $n \in \mathbb{N}$ as the utility of a state $S$ for agent $r$ in the goal relation $g(r, n, S)$.

For illustration, consider a formalization of the multiagent environment of Figure 1 using the symbols introduced above.

- $R = \{\mathit{Ag}_1, \mathit{Ag}_2, \mathit{Ag}_3\}$;
- $s_1 = \{\mathit{At}(\mathit{Ag}_1, 1, 1), \mathit{At}(\mathit{Ag}_2, 5, 1), \mathit{At}(\mathit{Ag}_3, 5, 5)\}$ ;
- $t$ contains all states

  $$\{\mathit{At}(\mathit{Ag}_1, x_1, y_1), \mathit{At}(\mathit{Ag}_2, x_2, y_2)\}$$

  (that is, where $\mathit{Ag}_3$ has escaped) along with all states

  $$\{\mathit{At}(\mathit{Ag}_1, x_1, y_1), \mathit{At}(\mathit{Ag}_2, x_2, y_2), \mathit{At}(\mathit{Ag}_3, x_3, y_3)\}$$

  in which $x_1 = x_3 \wedge y_1 = y_3$ or $x_2 = x_3 \wedge y_2 = y_3$ (that is, where $\mathit{Ag}_3$ has been caught);

- $l$ is defined as follows: each agent can always *Stay*; in every non-terminal state each agent can *Move*$(d)$ in any direction $d$ unless this would lead outside the physical environment; $Ag_3$ can *Exit* from any of the locations $(1,1)$, $(5,1)$, or $(1,5)$, provided it has not been caught.
- $u$ is defined as follows: actions *Stay*, *Exit*, and *Move*$(d)$ have the expected effects on the individual locations of the agents, with the exception that when the paths of $Ag_3$ and either of $Ag_1$ or $Ag_2$ (or both) cross in a simultaneous move, then $Ag_3$ ends up (caught) in the same location as $Ag_1$ or $Ag_2$, respectively. For illegal actions or states that are not reachable, $u$ may be arbitrarily defined.
- $g$ shall be defined as true for $n = 100$ and $r = Ag_3$ in terminal states in which this agent has escaped; conversely, $g$ holds for $n = 100$ and both $Ag_1$ and $Ag_2$ in terminal states in which $Ag_3$ got caught. In all other states the goal relation gives value $0$ for all three agents.

We have thus obtained a formal, symbolic description of the example multiagent environment. However, this specification is not yet amenable to automatic processing by an autonomous agent, because it uses natural language to describe some of the components. If this were to be translated into an explicit enumeration of the transition function, this would again yield too large a description to be of any practical use. In the following section, we show how the Game Description Language can be readily used to provide a fully axiomatic, compact description of arbitrary multiagent environments; a description that on the one hand is declarative and easy to understand and maintain by humans, and on the other hand can be fully automatically processed by artificial, autonomous agent systems.

## 3 Axiomatizing Multiagent Environments as Game Descriptions

The Game Description Language (GDL) has been developed to formalize the rules of any finite game with complete information in such a way that the description can be automatically processed by a general game player. In this section, we first recapitulate the GDL syntax from [1] and then show how the multiagent environments defined in the preceding section can be formally described in this language.

### 3.1 General GDL Syntax

GDL is based on the standard syntax of logic programs, including negation. A logic program is a set of clauses according to the following definition (see, for example, [7]).

**Definition 2.**

- *A* term *is either a variable, or a function symbol applied to terms as arguments (a* constant *is a function symbol with no argument);*
- *An* atom *is a predicate symbol applied to terms as arguments;*
- *A* literal *is an atom or its negation;*
- *A* clause *is an implication* $h \Leftarrow b_1 \wedge \ldots \wedge b_n$ *where* head $h$ *is an atom and* body $b_1 \wedge \ldots \wedge b_n$ *a conjunction of literals (*$n \geq 0$*).*

**Table 1.** GDL keywords

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

We adopt the Prolog convention according to which variables are denoted by uppercase letters and predicate and function symbols start with a lowercase letter. (The interested reader may take a peek at Figure 2 at this point to see some example clauses, which in fact constitute a complete GDL axiomatization of our running example domain.) GDL imposes some general restrictions on a set of clauses, with the intention to ensure finite derivability.

**Definition 3.** *The* dependency graph *for a set $G$ of clauses is a directed, labeled graph whose nodes are the predicate symbols that occur in $G$ and where there is a* positive *edge $p \xrightarrow{+} q$ if $G$ contains a clause $p(\overline{s}) \Leftarrow \ldots \wedge q(\overline{t}) \wedge \ldots$, and a* negative *edge $p \xrightarrow{-} q$ if $G$ contains a clause $p(\overline{s}) \Leftarrow \ldots \wedge \neg q(\overline{t}) \wedge \ldots$.*

*To constitute a* valid GDL specification*, a set of clauses $G$ and its dependency graph $\Gamma$ must satisfy the following.*

1. *There are no cycles involving a negative edge in $\Gamma$ (this is also known as being* stratified *[8,9]);*
2. *Each variable in a clause occurs in at least one positive atom in the body (this is also known as being* allowed *[10]);*
3. *If $p$ and $q$ occur in a cycle in $\Gamma$ and $G$ contains a clause*

$$p(s_1, \ldots, s_m) \Leftarrow b_1(\overline{t}_1) \wedge \ldots \wedge q(v_1, \ldots, v_k) \wedge \ldots \wedge b_n(\overline{t}_n)$$

*then for every $i \in \{1, \ldots, k\}$,*
   - *$v_i$ is variable-free, or*
   - *$v_i$ is one of $s_1, \ldots, s_m$, or*
   - *$v_i$ occurs in some $\overline{t}_j$ ($1 \leq j \leq n$) such that $b_j$ does not occur in a cycle with $p$ in $\Gamma$.*

Stratified logic programs are known to admit a specific *standard model*; we refer to [8] for details and just mention the following properties.

1. To obtain the standard model, clauses with variables are replaced by their (possibly infinitely many) ground instances.
2. Clauses are interpreted as reverse implications.
3. The standard model is minimal while interpreting negation as non-derivability (the "negation-as-failure" principle [11]);

The second and third restriction in Definition 3 essentially guarantee that a logic program entails a *finite* number of ground atoms via its standard model. This is necessary to enable agents to make effective use of a set of game rules.

### 3.2 GDL Keywords

As a tailor-made specification language, GDL uses a few pre-defined predicate symbols. These are shown in Table 1 together with their informal meaning. A further, standard predicate is $\texttt{distinct}(\texttt{X}, \texttt{Y})$ to express (syntactic) inequality of two terms.[1]

GDL imposes additional restrictions on the use of these keywords.

**Definition 4.** *A* valid GDL specification *is a set of clauses $G$ that, in addition to the restrictions in Definition 3, satisfies the following conditions.*

- `role` *only appears in the head of clauses that have an empty body;*
- `init` *only appears as head of clauses and is not connected, in the dependency graph for $G$, to any of* `true`, `legal`, `does`, `next`, `terminal`, `goal`*;*
- `true` *only appears in the body of clauses;*
- `does` *only appears in the body of clauses and is not connected, in the dependency graph for $G$, to any of* `legal`, `terminal`, `goal`*;*
- `next` *only appears as head of clauses.*

According to the informal semantics given in [1], a GDL specification $G$ is to be understood as follows. The derivable instances of $\texttt{role}(\texttt{R})$ define the players. The initial state is composed of the derivable instances of $\texttt{init}(\texttt{P})$. 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`. More precisely, let $S = \{p_1, \ldots, p_n\}$ be a state (e.g., the derivable instances of $\texttt{init}(\texttt{P})$ at the beginning), then $G$ is extended by the clauses

$$\begin{array}{l} \texttt{true}(p_1) \Leftarrow \\ \ldots \\ \texttt{true}(p_n) \Leftarrow \end{array} \tag{1}$$

Those instances of $\texttt{legal}(\texttt{R}, \texttt{A})$ which are derivable from this extended program define all legal actions `A` for player `R` in state $S$. In the same way, the clauses for `terminal` and $\texttt{goal}(\texttt{R}, \texttt{N})$ define termination and goalhood (of value `N` for player `R`) *relative* to the encoding of a given state. Determining a state transition, finally, requires the encoding of the current state along with clauses representing a joint move. Specifically, if players $r_1, \ldots, r_n$ make moves $a_1, \ldots, a_n$, then

$$\begin{array}{l} \texttt{does}(r_1, a_1) \Leftarrow \\ \ldots \\ \texttt{does}(r_n, a_n) \Leftarrow \end{array} \tag{2}$$

must be added to $G$, and then the derivable instances of $\texttt{next}(\texttt{P})$ compose the updated state.

[1] The semantics of this predicate is given by tacitly assuming the addition of the clause

$$\texttt{distinct}(s, t) \Leftarrow$$

for every pair $s, t$ of syntactically different ground terms.

### 3.3 Multiagent Environments in GDL

GDL provides all necessary features for declarative, formal descriptions of arbitrary multiagent environments as defined in Section 2. Of course there are many possible ways in which any specific environment can be axiomatized. We therefore define two sets of GDL clauses as *logically equivalent* if for any finite set of ground clauses (1) and (2) added, the two standard models of the two resulting logic programs agree on the interpretation of all GDL keywords. Before we can show how to formalize multiagent environments in GDL, we need the following syntactic definitions.

For any finite subset $S = \{p_1, \ldots, p_n\} \subseteq \Sigma$ of a set of ground terms, the following conjunction axiomatizes $S$ as the current state:

$$S^{\texttt{true}} \stackrel{\text{def}}{=} \texttt{true}(p_1) \wedge \ldots \wedge \texttt{true}(p_n) \wedge \neg p_S \tag{3}$$

Here, $p_S$ is an auxiliary predicate, one for every finite $S \subseteq \Sigma$, whose purpose is to ensure that the conjunction does not hold for states that are strict supersets of $S$:

$$\begin{aligned} p_S \Leftarrow\ & \texttt{true}(X) \wedge \\ & \texttt{distinct}(X, p_1) \wedge \ldots \wedge \texttt{distinct}(X, p_n) \end{aligned} \tag{4}$$

Hence, $p_S$ is true for any state in which at least one state component $X$ is true that differs syntactically from any of $p_1, \ldots, p_n$, that is, the elements of $S$. This ensures that the conjunction defined in (3) is an exact axiomatization of state $S$.

Furthermore, for any function $A : \{r_1, \ldots, r_n\} \mapsto \Sigma$, where $r_1, \ldots, r_n \in \Sigma$, the following conjunction axiomatizes $A$ as a joint action:

$$A^{\texttt{does}} \stackrel{\text{def}}{=} \texttt{does}(r_1, A(r_1)) \wedge \ldots \wedge \texttt{does}(r_n, A(r_n)) \tag{5}$$

We are now ready to show how GDL can be used to axiomatize multiagent domains.

**Definition 5.** *Let $E = (R, s_1, t, l, u, g)$ be a multiagent environment based on ground symbolic expressions $\Sigma$, then any valid set of GDL clauses is an* axiomatic description *of $E$ if it is logically equivalent to the following.*

- $\texttt{role}(r) \Leftarrow$ *for each* $r \in R$;
- $\texttt{init}(p) \Leftarrow$ *for each* $p \in s_1$;
- $\texttt{terminal} \Leftarrow S^{\texttt{true}}$ *for each* $S \in t$;
- $\texttt{legal}(r, a) \Leftarrow S^{\texttt{true}}$ *for each* $(r, a, S) \in l$;
- $\texttt{next}(p) \Leftarrow A^{\texttt{does}} \wedge S^{\texttt{true}}$ *for each* $p \in u(A, S)$ *and* $A : R \mapsto \Sigma,\ S \subseteq \Sigma$;
- $\texttt{goal}(r, n) \Leftarrow S^{\texttt{true}}$ *for each* $(r, n, S) \in g$.

It is important to realize that this direct axiomatization, where all relations and functions are encoded explicitly, is used solely to define the intended semantics. In practice, of course, a domain can be described in a much more compact manner, using variables, logical equivalence, and possibly auxiliary predicates. As an example, Figure 2 depicts a complete GDL specification of the multiagent environment introduced in Section 2. It is not too difficult to verify that this is a valid set of clauses according to Definition 3 and 4 and that it is indeed a correct axiomatic description of this domain according to Definition 5.

$$
\begin{array}{l}
\mathtt{role}(\mathtt{ag}_1) \Leftarrow \\
\mathtt{role}(\mathtt{ag}_2) \Leftarrow \\
\mathtt{role}(\mathtt{ag}_3) \Leftarrow
\end{array}
$$

$$
\begin{array}{l}
\mathtt{init}(\mathtt{at}(\mathtt{ag}_1, 1, 1)) \Leftarrow \\
\mathtt{init}(\mathtt{at}(\mathtt{ag}_2, 5, 1)) \Leftarrow \\
\mathtt{init}(\mathtt{at}(\mathtt{ag}_3, 1, 5)) \Leftarrow
\end{array}
$$

$$
\begin{array}{l}
\mathtt{terminal} \Leftarrow \mathtt{true}(\mathtt{at}(\mathtt{ag}_1,\mathtt{X},\mathtt{Y})) \wedge \mathtt{true}(\mathtt{at}(\mathtt{ag}_3,\mathtt{X},\mathtt{Y})) \\
\mathtt{terminal} \Leftarrow \mathtt{true}(\mathtt{at}(\mathtt{ag}_2,\mathtt{X},\mathtt{Y})) \wedge \mathtt{true}(\mathtt{at}(\mathtt{ag}_3,\mathtt{X},\mathtt{Y})) \\
\mathtt{terminal} \Leftarrow \neg\mathtt{remain}
\end{array}
$$

$$
\mathtt{remain} \Leftarrow \mathtt{true}(\mathtt{at}(\mathtt{ag}_3,\mathtt{X},\mathtt{Y}))
$$

$$
\begin{array}{l}
\mathtt{legal}(\mathtt{R}, \mathtt{stay}) \Leftarrow \mathtt{true}(\mathtt{at}(\mathtt{R}, \mathtt{X}, \mathtt{Y})) \\
\mathtt{legal}(\mathtt{ag}_3, \mathtt{exit}) \Leftarrow \neg\mathtt{terminal} \wedge \mathtt{true}(\mathtt{at}(\mathtt{ag}_3, 1, 1)) \\
\mathtt{legal}(\mathtt{ag}_3, \mathtt{exit}) \Leftarrow \neg\mathtt{terminal} \wedge \mathtt{true}(\mathtt{at}(\mathtt{ag}_3, 5, 1)) \\
\mathtt{legal}(\mathtt{ag}_3, \mathtt{exit}) \Leftarrow \neg\mathtt{terminal} \wedge \mathtt{true}(\mathtt{at}(\mathtt{ag}_3, 1, 5)) \\
\mathtt{legal}(\mathtt{R}, \mathtt{move}(\mathtt{D})) \Leftarrow \neg\mathtt{terminal} \wedge \mathtt{true}(\mathtt{at}(\mathtt{R}, \mathtt{U}, \mathtt{V})) \wedge \mathtt{adjacent}(\mathtt{U}, \mathtt{V}, \mathtt{D}, \mathtt{X}, \mathtt{Y})
\end{array}
$$

$$
\begin{array}{l}
\mathtt{adjacent}(\mathtt{X}, \mathtt{Y}_1, \mathtt{north}, \mathtt{X}, \mathtt{Y}_2) \Leftarrow \mathtt{co}(\mathtt{X}) \wedge \mathtt{succ}(\mathtt{Y}_1, \mathtt{Y}_2) \\
\mathtt{adjacent}(\mathtt{X}, \mathtt{Y}_1, \mathtt{south}, \mathtt{X}, \mathtt{Y}_2) \Leftarrow \mathtt{co}(\mathtt{X}) \wedge \mathtt{succ}(\mathtt{Y}_2, \mathtt{Y}_1) \\
\mathtt{adjacent}(\mathtt{X}_1, \mathtt{Y}, \ \mathtt{east}, \mathtt{X}_2, \mathtt{Y}) \Leftarrow \mathtt{co}(\mathtt{Y}) \wedge \mathtt{succ}(\mathtt{X}_1, \mathtt{X}_2) \\
\mathtt{adjacent}(\mathtt{X}_1, \mathtt{Y}, \ \mathtt{west}, \mathtt{X}_2, \mathtt{Y}) \Leftarrow \mathtt{co}(\mathtt{Y}) \wedge \mathtt{succ}(\mathtt{X}_2, \mathtt{X}_1)
\end{array}
$$

$$
\begin{array}{llllll}
\mathtt{co}(1) \Leftarrow & \mathtt{co}(2) \Leftarrow & \mathtt{co}(3) \Leftarrow & \mathtt{co}(4) \Leftarrow & \mathtt{co}(5) \Leftarrow \\
\mathtt{succ}(1,2) \Leftarrow & \mathtt{succ}(2,3) \Leftarrow & \mathtt{succ}(3,4) \Leftarrow & \mathtt{succ}(4,5) \Leftarrow
\end{array}
$$

$$
\begin{array}{l}
\mathtt{next}(\mathtt{at}(\mathtt{R}, \mathtt{X}, \mathtt{Y})) \Leftarrow \mathtt{does}(\mathtt{R}, \mathtt{stay}) \wedge \mathtt{true}(\mathtt{at}(\mathtt{R}, \mathtt{X}, \mathtt{Y})) \\
\mathtt{next}(\mathtt{at}(\mathtt{R}, \mathtt{X}, \mathtt{Y})) \Leftarrow \mathtt{does}(\mathtt{R}, \mathtt{move}(\mathtt{D})) \wedge \mathtt{true}(\mathtt{at}(\mathtt{R}, \mathtt{U}, \mathtt{V})) \wedge \mathtt{adjacent}(\mathtt{U}, \mathtt{V}, \mathtt{D}, \mathtt{X}, \mathtt{Y}) \wedge \\
\qquad\qquad \neg\mathtt{capture}(\mathtt{R}) \\
\mathtt{next}(\mathtt{at}(\mathtt{ag}_3, \mathtt{X}, \mathtt{Y})) \Leftarrow \mathtt{true}(\mathtt{at}(\mathtt{ag}_3, \mathtt{X}, \mathtt{Y})) \wedge \mathtt{capture}(\mathtt{ag}_3)
\end{array}
$$

$$
\begin{array}{l}
\mathtt{capture}(\mathtt{ag}_3) \Leftarrow \mathtt{true}(\mathtt{at}(\mathtt{ag}_3, \mathtt{X}, \mathtt{Y})) \wedge \mathtt{true}(\mathtt{at}(\mathtt{R}, \mathtt{U}, \mathtt{V})) \wedge \mathtt{does}(\mathtt{ag}_3, \mathtt{move}(\mathtt{D}_1)) \wedge \\
\qquad\qquad \mathtt{does}(\mathtt{R}, \mathtt{move}(\mathtt{D}_2)) \wedge \mathtt{adjacent}(\mathtt{X}, \mathtt{Y}, \mathtt{D}_1, \mathtt{U}, \mathtt{V}) \wedge \mathtt{adjacent}(\mathtt{U}, \mathtt{V}, \mathtt{D}_2, \mathtt{X}, \mathtt{Y})
\end{array}
$$

$$
\begin{array}{l}
\mathtt{goal}(\mathtt{R}, 0) \Leftarrow \mathtt{role}(\mathtt{R}) \wedge \neg\mathtt{terminal} \\
\mathtt{goal}(\mathtt{R}, 0) \Leftarrow \mathtt{role}(\mathtt{R}) \wedge \mathtt{distinct}(\mathtt{R}, \mathtt{ag}_3) \wedge \mathtt{terminal} \wedge \neg\mathtt{remain} \\
\mathtt{goal}(\mathtt{R}, 100) \Leftarrow \mathtt{role}(\mathtt{R}) \wedge \mathtt{distinct}(\mathtt{R}, \mathtt{ag}_3) \wedge \mathtt{terminal} \wedge \mathtt{true}(\mathtt{at}(\mathtt{ag}_3, \mathtt{X}, \mathtt{Y})) \\
\mathtt{goal}(\mathtt{ag}_3, 0) \Leftarrow \mathtt{terminal} \wedge \mathtt{true}(\mathtt{at}(\mathtt{ag}_3, \mathtt{X}, \mathtt{Y})) \\
\mathtt{goal}(\mathtt{ag}_3, 100) \Leftarrow \mathtt{terminal} \wedge \neg\mathtt{remain}
\end{array}
$$

**Fig. 2.** A complete, formal description of the multiagent environment of Figure 1

The specification of the Game Description Language in [1] lacks a fully formal definition of the intended meaning of a specification. This is why there are no formal grounds on which it could actually be proved that Definition 5 yields a correct description of a multiagent environment. In fact, we can and will use our formal concept of a

multiagent domain to provide just this precise semantics for GDL in terms of a transition system.

## 4 A Multiagent Semantics for GDL

In the preceding section, we have shown how GDL provides a declarative, compact language to formally describe a large class of multiagent environments in a machine processable fashion. In this section, we show how the abstract model of a multiagent environment can in turn be used to provide a formal semantics for GDL in terms of a transition system. In this way we make precise what is only informally described in [1].

Any valid game description $G$ in GDL contains a finite set of function symbols, including constants, which implicitly determines a (usually infinite) set of ground terms. This set constitutes the symbol base $\Sigma$ in the transition-based semantics for $G$. The syntactic restrictions in GDL ensure finite derivability, so that each state, the set of roles, etc. are all finite subsets of $\Sigma$. The following definition of the semantics of a GDL description is straightforwardly obtained by reversing the mapping from a multiagent environment into GDL (cf. Definition 5). To this end, we redefine the abbreviations $S^{\mathtt{true}}$ and $A^{\mathtt{does}}$ as logic program facts (rather than conjunctions as in (3) and (5)). This allows to add them to $G$ in order to determine terminal states, legal moves, updates, and goalhood:

$$\begin{aligned} S^{\mathtt{true}} &\stackrel{\text{def}}{=} \{\, \mathtt{true}(p_1) \Leftarrow \\ &\qquad \ldots \\ &\qquad \mathtt{true}(p_n) \Leftarrow \,\} \\ A^{\mathtt{does}} &\stackrel{\text{def}}{=} \{\, \mathtt{does}(r_1, A(r_1)) \Leftarrow \\ &\qquad \ldots \\ &\qquad \mathtt{does}(r_n, A(r_n)) \Leftarrow \,\} \end{aligned}$$

It is worth mentioning that auxiliary predicate $p_S$ (cf. (4)) is not needed in the axiomatization of a state as a set of facts, because the principle of negation-as-failure and $S^{\mathtt{true}}$ imply $\neg\mathtt{true}(p)$ for any $p \notin S$.

**Definition 6.** *Let $G$ be a valid GDL specification, whose signature determines the set of ground terms $\Sigma$. The* semantics *of $G$ is the multiagent environment $(R, s_1, t, l, u, g)$ where*[2]

- $R = \{r \in \Sigma : G \models \mathtt{role}(r)\}$;
- $s_1 = \{p \in \Sigma : G \models \mathtt{init}(p)\}$;
- $t = \{S \in 2^\Sigma : G \cup S^{\mathtt{true}} \models \mathtt{terminal}\}$;
- $l = \{(r, a, S) : G \cup S^{\mathtt{true}} \models \mathtt{legal}(r, a)\}$, *where* $r \in R$, $a \in \Sigma$, *and* $S \in 2^\Sigma$;
- $u(A, S) = \{p \in \Sigma : G \cup A^{\mathtt{does}} \cup S^{\mathtt{true}} \models \mathtt{next}(p)\}$, *for all* $A : (R \mapsto \Sigma)$ *and* $S \in 2^\Sigma$;
- $g = \{(r, n, S) : G \cup S^{\mathtt{true}} \models \mathtt{goal}(r, n)\}$, *where* $r \in R$, $n \in \mathbb{N}$, *and* $S \in 2^\Sigma$.

This definition provides a formal semantics for GDL in terms of abstract multiagent environments. Finite derivability in valid GDL specifications implies that the

[2] Below, entailment ($\models$) is via the standard model of a set of clauses.

entailment relation is decidable, which in turn ensures that the definition of the semantics is effective.

In the preceding section we have seen that one and the same multiagent environment can be axiomatically described in many different ways. With the help of Definition 6 it is now easy to verify that two logically equivalent GDL descriptions (as defined in Section 3.3) describe exactly the same environment.

**Proposition 1.** *The semantics of two logically equivalent, valid GDL descriptions coincide.*

*Proof.* By definition, two logically equivalent GDL descriptions agree on the interpretation of all GDL keywords for all finite additions of clauses (1) and (2). It is easy to see, then, that the various components of their semantics according to Definition 6 must be identical.

Based on this result it is also straightforward to prove that Definition 6 indeed provides the complement to the encoding of a multiagent environment in GDL.

**Proposition 2.** *Let $E$ be a multiagent environment and $G$ any axiomatic description thereof, then the semantics of $G$ is $E$.*

*Proof.* Consider the generic encoding of $E$ given in Definition 5. It is easy to verify that the standard model for this set of clauses, augmented by any finite set of facts about relations `true` and `does` (cf. clauses (1) and (2), respectively, in Section 3.2), determines a semantics $(R, s_1, t, l, u, g)$ via Definition 6 which equals $E$. The claim follows from Proposition 1 and the fact that any GDL encoding for $E$ is logically equivalent to the generic clauses given in Definition 5.

## 5 Discussion

We have shown how the Game Description Language, developed in the context of General Game Playing, can be understood as a declarative language to provide compact and machine processable specifications of a large class of multiagent environments. This can be applied to formalize the rules, for example, of an e-marketplace, of publicly accessible agent platforms on the Internet, of problem domains used in agent competitions, etc. By automatically processing these specifications, autonomous agents can fully automatically learn how to participate in a new or modified environment without the need to be (re-)programmed. Moreover, successful off-the-shelf general game playing systems can be readily employed as intelligent agents for these environments.

It is interesting to note that GDL has been originally developed as problem specification language for a competition [2], much like the Planning Domain Description Language (PDDL) [12], which today is a quasi standard for the specification of planning domains. GDL can be viewed as a generalization of PDDL to domains with multiple agents, because solving a planning problem can be understood as playing a single-player game. Indeed, most features of current versions of PDDL can be expressed in GDL, though with one notable exception: sensing actions are not included in the current version of GDL. Although a GDL specification leaves agents with uncertainty about

how the world evolves (an agent can decide on its own actions but not on those of all other agents), the language has been written for games without information asymmetry. An important research issue for the near future is to extend the Game Description Language so as to support descriptions of games with asymmetric information and sensing actions, which is a typical feature of card games, for instance. This would then provide a suitable formalization language for an even larger class of multiagent environments than considered in this paper.

In the second part of the paper, we have used the concept of a multiagent environment to provide a formal, transition-based semantics for GDL. With this we have made precise what is only informally described in [1]. Our semantics for GDL in terms of multiagent environments is related to an existing formal characterization of GDL by a game structure [13]. The main difference of the latter in comparison to our work are:

- It is restricted to propositional GDL;
- It puts further restrictions on GDL, such as not allowing predicate `init` to occur in clause with non-empty bodies;
- It uses an inductive definition of the set of all states in order to obtain only those which are reachable from the initial state. Since it is possible to give valid GDL specifications of games that do not terminate, this definition would be undecidable in the general setting.[3]

These restrictions have been imposed because the focus in [13] lies on the use of Temporal Logic for the purpose of verifying properties of games, such as termination or winnability. In contrast to this, the semantics given in the present paper covers full GDL.

**Acknowledgements.** This research was partially supported by *Deutsche Forschungsgemeinschaft* under Contract TH 541/16-1.

## References

1. Love, N., Hinrichs, T., Haley, D., Schkufza, E., Genesereth, M.: 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 (2006), `games.stanford.edu`
2. Genesereth, M., Love, N., Pell, B.: General game playing: Overview of the AAAI competition. AI Magazine 26, 62–72 (2005)

[3] It is worth noting that this does not contradict the finite derivability property of valid GDL specifications, which just implies that all *local* reasoning problems are decidable. More specifically, given a particular state it is decidable whether an action is possible, and given a joint action it is also decidable what properties hold in the updated state, etc. On the other hand, GDL is expressive enough to describe any Turing machine as a "game" using clauses like

$$\begin{aligned} &\texttt{init}(\texttt{head}(0)) \Leftarrow \\ &\texttt{next}(\texttt{head}(\texttt{succ}(\texttt{X}))) \Leftarrow \texttt{true}(\texttt{head}(\texttt{X})) \wedge \\ &\qquad\qquad\qquad\qquad\quad \texttt{does}(\texttt{tm}, \texttt{move\_forward}) \end{aligned}$$

Hence, reachability of states is generally undecidable in GDL.

3. Kuhlmann, G., Dresner, K., Stone, P.: Automatic heuristic construction in a complete general game player. In: Proceedings of the AAAI National Conference on Artificial Intelligence, pp. 1457–1462. AAAI Press, Boston (2006)
4. Clune, J.: Heuristic evaluation functions for general game playing. In: Proceedings of the AAAI National Conference on Artificial Intelligence, Vancouver, pp. 1134–1139. AAAI Press, Menlo Park (2007)
5. Schiffel, S., Thielscher, M.: Fluxplayer: A successful general game player. In: Proceedings of the AAAI National Conference on Artificial Intelligence, Vancouver, pp. 1191–1196. AAAI Press, Menlo Park (2007)
6. Finnsson, H., Björnsson, Y.: Simulation-based approach to general game playing. In: Proceedings of the AAAI National Conference on Artificial Intelligence, Chicago, pp. 259–264. AAAI Press, Menlo Park (2008)
7. Lloyd, J.: Foundations of Logic Programming, 2nd extended edn. Series Symbolic Computation. Springer, Heidelberg (1987)
8. Apt, K., Blair, H.A., Walker, A.: Towards a theory of declarative knowledge. In: Minker, J. (ed.) Foundations of Deductive Databases and Logic Programming, pp. 89–148. Morgan Kaufmann, San Francisco (1987)
9. van Gelder, A.: The alternating fixpoint of logic programs with negation. In: Proceedings of the 8th Symposium on Principles of Database Systems, ACM SIGACT-SIGMOD, pp. 1–10 (1989)
10. Lloyd, J., Topor, R.: A basis for deductive database systems II. Journal of Logic Programming 3, 55–67 (1986)
11. Clark, K.: Negation as failure. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, pp. 293–322. Plenum Press (1978)
12. McDermott, D.: The 1998 AI planning systems competition. AI Magazine 21, 35–55 (2000)
13. van der Hoek, W., Ruan, J., Wooldridge, M.: Strategy logics and the game description language. In: Proceedings of the Workshop on Logic, Rationality and Interaction, Bejing, China (2007)