Translating General Game Descriptions into an Action Language

Michael Thielscher

School of Computer Science and Engineering
The University of New South Wales
`mit@cse.unsw.edu.au`

Abstract. The game description language (GDL), which is the basis for the grand AI challenge of *general* game playing, can be viewed as yet another action language. However, due to its unique way of addressing the frame problem, GDL turned out to be surprisingly difficult to relate to any of the classical action formalisms. In this paper, we present the first complete embedding of GDL into an existing member, known as, $\mathcal{C}+$, of the family of action languages. Our provably correct translation paves the way for applying known results from reasoning about actions, including competitive implementations such as the Causal Calculator, to the new and ambitious challenge of general game playing.

1 Introduction

General game playing is concerned with the development of systems that understand the rules of previously unknown games and learn to play these games well without human intervention. Identified as one of the contemporary grand AI challenges, this endeavour requires to build intelligent agents that are capable of high-level reasoning and learning. An annual competition has been established in 2005 to foster research in this area [10]. This has lead to a number of successful approaches and systems [12, 5, 18, 6].

Representing and reasoning about actions is a core technique in general game playing. A game description language (GDL) has been developed to formalise the rules of arbitrary n-player games ($n \geq 1$) in such a way that they can be automatically processed by a general game player [14]. The emphasis is on high-level, declarative descriptions. This allows successful players to reason about the rules of an unknown game in order to extract game-specific knowledge [19, 23] and to automatically design evaluation functions [12, 5, 18].

The game description language shares principles with action formalisms, a central aspect of logic-based knowledge representation ever since the introduction of the classical Situation Calculus [15]. For instance, a game description must entail the conditions under which a move is legal. This corresponds to action preconditions. Furthermore, the rules of a game must include how the various moves change the game state. This corresponds to effect specifications.

Although GDL can thus be viewed as a special-purpose action language, formally relating GDL relates to any existing action formalism proved to be surprisingly difficult. The main reason seems to be that GDL is based on a rather unique solution to the fundamental frame problem [16] where positive frame axioms are combined with the principle of negation-by-failure [4] to encode negative effects. This makes it notoriously difficult to infer the explicit positive and negative effects of an individual move from its GDL description [21].

In this paper, we present the first formal result that links the general game description language to an existing action formalism. Specifically, we develop an embedding of GDL into a successor of the Action Description Language invented by Michael Gelfond and his colleague Vladimir Lifschitz [9]. The target action language we chose, known as $\mathcal{C}+$ [11], allows to circumvent the problem of identifying the positive and negative effects of individual moves. As the main result we formally prove that our embedding is correct and provides an easily automatable translation of full GDL into an action language.

Our result paves the way for applying known methods from reasoning about actions in the area of general game playing. For example, we can immediately deploy an existing implementation for $\mathcal{C}+$, the Causal Calculator [11, 1], to do automated reasoning with, and about, game descriptions in GDL. In this way the relatively new field of General Game Playing can profit from many years of research in reasoning about actions. Conversely, this grand AI challenge may provide a new and interesting testbed for existing action formalisms.

The rest of the paper is organised as follows. In Section 2, we recapitulate the basic syntax and semantics of the general game description language, followed by a brief introduction to action languages. In Section 3, we present a complete embedding of this language into the action language $\mathcal{C}+$. In the section that follows, we prove that this translation is correct. We conclude in Section 5.

2 Preliminaries

2.1 Describing Games in GDL

The Game Description Language (GDL) has been developed to formalise 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 [10, 14]. GDL is based on the standard syntax of logic programs, including negation. We assume familiarity with the basic notions of normal logic programs, as can be found in [3]. We adopt the Prolog convention according to which variables are denoted by uppercase letters and predicate and function symbols start with a lowercase letter. As a tailor-made specification language, GDL uses a few predefined predicate symbols shown in Table 1. The use of these keywords must obey the following syntactic restrictions [14].

Definition 1. *In a* valid *GDL game description,*

- `role` *only appears in facts;*
- `init` *and* `next` *only appear as head of clauses;*

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

Table 1. The pre-defined keywords of GDL.

– `init` *does not depend*[1] *on any of* `true`*,* `legal`*,* `does`*,* `next`*,* `terminal`*, or* `goal`*;*
– `true` *and* `does` *only appear in the body of clauses; and*
– *neither of* `legal`*,* `terminal`*, or* `goal` *depends on* `does`*.*

Example 1 Figure 1 depicts a game description of standard Tic-Tac-Toe, where two players, respectively called *xplayer* and *oplayer*, take turn in marking the cells of a 3×3-board. (For the sake of simplicity, the straightforward definitions of termination and winning criteria are omitted.) Of particular interest from the viewpoint of action logics is the encoding of position updates. The first clause with head `next` specifies the direct positive effect of marking a cell. The second clause serves as an action-independent frame axiom for state feature *cell*. The third and fourth clause for `next` describe how the argument of *control* alters between consecutive moves. It is especially noteworthy that the clauses together entail the implicit negative effect that $control(xplayer)$ will become false (after any joint move) unless $control(oplayer)$ holds in the current position, and vice versa. This is a consequence of the negation-as-failure principle built into the semantics of GDL, which we will briefly recapitulate below. The reader should also note the clausal definition of the predicate *taken*. This clause acts as a state constraint, giving rise to the indirect effect that a cell will be taken after a player has marked it. ■

GDL imposes some general restrictions on a set of clauses as a whole. Specifically, it must be *stratified* [2], *allowed* [13], and comply with a recursion restriction that guarantees that the entire set of clauses is equivalent to a finite set of ground clauses (we refer to [14] for details). Stratified logic programs are known to admit a specific standard model [17], which coincides with its unique stable model (also called its answer set) [8, 7].

Based on the concept of this standard model, a GDL description is understood as a state transition system as follows [20]. To begin with, any valid game description G contains a finite set of function symbols, including constants,

[1] A predicate p is said to *depend* on a predicate q if q occurs in the body of a clause for p, or if some predicate in a clause for p depends on q.

```
role(xplayer).
role(oplayer).

init(control(xplayer)).

legal(P,mark(M,N))   :- true(control(P)),
                        index(M), index(N), ¬taken(M,N)
legal(xplayer,noop) :- ¬true(control(xplayer))
legal(oplayer,noop) :- ¬true(control(oplayer))

next(cell(M,N,Z))       :- marking(M,N,Z)
next(cell(M,N,Z))       :- true(cell(M,N,Z))
next(control(oplayer)) :- true(control(xplayer))
next(control(xplayer)) :- true(control(oplayer))

marking(M,N,x) :- does(xplayer,mark(M,N))
marking(M,N,o) :- does(oplayer,mark(M,N))

taken(M,N) :- marker(Z), true(cell(M,N,Z))

index(1).
index(2).
index(3).
marker(x).
marker(o).

terminal :- ...
goal(xplayer,100) :- ...
```

Fig. 1. A GDL description of Tic-Tac-Toe. Game positions are encoded using two features: $\mathit{control}(P)$, where $P \in \{\mathit{xplayer}, \mathit{oplayer}\}$, and $\mathit{cell}(M,N,Z)$, where $M,N \in \{1,2,3\}$ and $Z \in \{x,o\}$ (the markers).

which implicitly determines a set of ground terms Σ. This set constitutes the symbol base Σ in the formal semantics for G.

The players $R \subseteq \Sigma$ and the initial position of a game can be directly determined from the clauses for `role` and `init`, respectively. In order to determine the legal moves, update, termination, and goal values (if any) for a given position, this position has to be encoded first, using the keyword `true`. To this end, for any *finite* subset $S = \{f_1, \dots, f_n\} \subseteq \Sigma$ of ground terms, the following set of logic program facts encodes S as the current position:

$$S^{\mathtt{true}} \overset{\text{def}}{=} \{\mathtt{true}(f_1)\mathtt{.}, \ \dots, \ \mathtt{true}(f_n)\mathtt{.}\}$$

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

$$A^{\mathtt{does}} \overset{\text{def}}{=} \{\mathtt{does}(r_1, A(r_1))\mathtt{.}, \ \dots, \ \mathtt{does}(r_k, A(r_k))\mathtt{.}\} \tag{1}$$

Adding the unary clauses $S^{\texttt{true}} \cup A^{\texttt{does}}$ allows to infer the position that results from taking moves A in state S. All this is summarised in the following definition.

Definition 2. *Let G be a GDL specification whose signature determines the set of ground terms Σ. Let 2^{Σ} be the set of* finite *subsets of Σ. The* semantics *of G is the state transition system (R, S_0, T, l, u, g) where*[2]

- $R = \{r \in \Sigma : G \models \texttt{role}(r)\}$ *(the players);*
- $S_0 = \{f \in \Sigma : G \models \texttt{init}(f)\}$ *(the initial position);*
- $T = \{S \in 2^{\Sigma} : G \cup S^{\texttt{true}} \models \texttt{terminal}\}$ *(the terminal positions);*
- $l = \{(r, a, S) : G \cup S^{\texttt{true}} \models \texttt{legal}(r, a)\}$*, where $r \in R$, $a \in \Sigma$, and $S \in 2^{\Sigma}$ (the legality relation);*
- $u(A, S) = \{f \in \Sigma : G \cup S^{\texttt{true}} \cup A^{\texttt{does}} \models \texttt{next}(f)\}$*, for all $A : (R \mapsto \Sigma)$ and $S \in 2^{\Sigma}$ (the update function);*
- $g = \{(r, v, S) : G \cup S^{\texttt{true}} \models \texttt{goal}(r, v)\}$*, where $r \in R$, $v \in \mathbb{N}$, and $S \in 2^{\Sigma}$ (the goal relation).*

For $S, S' \in 2^{\Sigma}$ we write $S \xrightarrow{A} S'$ if $A : (R \mapsto \Sigma)$ is such that $(r, A(r), S) \in l$ for each $r \in R$ and $S' = u(A, S)$ (and $S \notin T$). We call

$$S_0 \xrightarrow{A_0} S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{n-1}} S_n$$

a development *(where $n \geq 0$).*

Example 1 (Continued) The clauses in Figure 1 obviously entail the initial state $S_0 = \{\mathit{control}(\mathit{xplayer})\}$. Suppose, therefore, $S_0^{\texttt{true}}$ is added to the program:

```
true(control(xplayer)).
```

Since all instances of $\mathit{taken}(m, n)$ are false in the standard model of the program thus obtained, it follows that *xplayer* has nine legal moves, viz. $\mathit{mark}(m, n)$ for each pair of arguments $(m, n) \in \{1, 2, 3\} \times \{1, 2, 3\}$. The only derivable legal move for *oplayer* is the constant *noop*. In order to determine the outcome of a particular joint move, say $A = \{\mathit{xplayer} \mapsto \mathit{mark}(2, 2), \mathit{oplayer} \mapsto \mathit{noop}\}$, we have to further add $A^{\texttt{does}}$ to the program:

```
does(xplayer,mark(2,2)).
does(oplayer,noop).
```

The resulting position is determined as the derivable arguments of keyword `next`, which in this case are

$$\{\mathit{cell}(2, 2, x),\ \mathit{control}(\mathit{oplayer})\}$$

■

[2] Below, entailment $\models$ is via the aforementioned standard model for a stratified set of clauses.

Definition 2 provides the basis for interpreting a GDL description as an abstract k-player game as follows. In every position S, starting with S_0, each player r chooses a move a that satisfies $l(r, a, S)$. As a consequence the game state changes to $u(A, S)$, where A is the joint move. The game ends if a position in T is reached, and then the goal relation, g, determines the outcome. The restrictions in GDL ensure that entailment wrt. the standard model is decidable and that only finitely many instances of each predicate are entailed. This guarantees that the definition of the semantics is effective.

2.2 Action Languages

Michael Gelfond and Vladimir Lifschitz in the early 1990s developed a basic language called $\mathcal{A}$ to describe action domains in a simple and intuitive way but with a precise, formal semantics [9]. The purpose of this language was to facilitate the assessment and comparison of existing AI formalisms and implementations for reasoning about actions. Over the years, $\mathcal{A}$ has been extended in various ways, and today there exists a whole family of action languages, including one called $\mathcal{C}+$ [11], along with several implementations.

Unlike GDL, all action languages use a sorted signature which distinguishes between *actions* and *fluents*. In $\mathcal{C}+$, there is the further distinction between *simple* fluents and *statically determined* ones. For example, a description of Tic-Tac-Toe as action domain may use the simple fluents $\mathit{control}(P)$ and $\mathit{cell}(M, N, Z)$ (for all instances $P \in \{\mathit{xplayer}, \mathit{oplayer}\}$, $M, N \in \{1, 2, 3\}$, and $Z \in \{x, o\}$); the statically determined fluent $\mathit{legal}(\mathit{xplayer}, \mathit{noop})$; and, with the same domain for M, N, Z, the actions $\mathit{marking}(M, N, Z)$ and $\mathit{does}(\mathit{xplayer}, \mathit{mark}(M, N))$.

Given a sorted domain signature, a *fluent formula* in the action language $\mathcal{C}+$ is a formula with only fluents as atoms, while an *action formula* is a formula with at least one action as atom. A general formula can have both actions and fluents as atoms. For instance, with the fluents and actions just introduced, $\neg \mathit{control}(\mathit{xplayer})$ is a fluent formula while any instance of the (reverse) implication $\mathit{marking}(M, N, Z) \subset \mathit{does}(\mathit{xplayer}, \mathit{mark}(M, N))$ is an action formula.

A domain description in $\mathcal{C}+$ is composed of so-called causal laws, of which there are three types.

1. A *static law* is of the form **caused** F **if** G, where F and G are fluent formulas. Intuitively, it means that there is a cause for F to be true in every state in which G holds. An example is

$$\textbf{caused } \mathit{legal}(\mathit{xplayer}, \mathit{noop}) \textbf{ if } \neg \mathit{control}(\mathit{xplayer})$$

2. An *action dynamic law* is of the same form, **caused** F **if** G, but with F an action formula and G a general formula. An example is

$$\textbf{caused } \mathit{marking}(M, N, Z) \subset \mathit{does}(\mathit{xplayer}, \mathit{mark}(M, N)) \textbf{ if } \top$$

where symbol $\top$ denotes the unconditional truth.

3. A *fluent dynamic law* is of the form G **causes** F, where G is a general formula and F is a fluent formula without statically determined fluents. Intuitively, it means that there is a cause for F to be true in the next state if G is true in the current one.[3] An example is

$$\mathit{marking}(M,N,Z)\ \textbf{causes}\ \mathit{cell}(M,N,Z)$$

The formal semantics for causal laws will be introduced in Section 4 as part of the main correctness proof in this paper.

3 Translating GDL into Action Language $\mathcal{C}+$

In this section we construct, step by step, a complete embedding of GDL into the action language $\mathcal{C}+$. For this we assume given an arbitrary game description G. By $\mathit{grd}(G)$ we denote the (finite) set of ground instances of the clauses in G.

Since action languages use a sorted signature, the first step of translating GDL into action language $\mathcal{C}+$ is to assign a unique sort to the various syntactic elements of a given game description. This is easily done as follows.

Definition 3. *Let G be a GDL game description with roles R.*

1. The simple fluents *are the ground terms f that occur as arguments in* `init`(f)*,* `true`(f)*, or* `next`(f) *in some clause in $\mathit{grd}(G)$.*
2. The statically determined fluents *are*
   - *the instances of the three pre-defined predicates* `legal`*,* `terminal`*, and* `goal`*; and*
   - *the instances of the atoms other than the pre-defined GDL predicates (cf. Table 1) that occur in some clause in $\mathit{grd}(G)$ and do not depend on* `does`*.*
3. The actions *are*
   - *all terms of the form $does(r,a)$ such that $r \in R$ and "a" is a ground term with* `does`$(\_,a)$ *or* `legal`$(\_,a)$ *appearing in a clause in $\mathit{grd}(G)$; and*
   - *the instances of the atoms other than the pre-defined GDL predicates that occur as a head of some clause in $\mathit{grd}(G)$ and that depend on* `does`*.*

This assignment of sorts is straightforward with the exception of the domain-dependent predicates whose definition depends on the GDL keyword `does` (an example is the predicate $\mathit{marking}(M,N,Z)$ in Figure 1). These predicates need to be formally treated as actions for purely syntactic reasons, as will become clear in Section 4 when we prove the correctness of the translation.

[3] Full $\mathcal{C}+$ uses a slightly more general definition of fluent dynamic laws, but the restricted version suffices as a target language for translating GDL.

Example 1 (Continued) Recall the Tic-Tac-Toe clauses in Figure 1. They determine the simple fluents[4]

$$control(xplayer),\ control(oplayer),\ cell(1,1,x),\ \dots,\ cell(3,3,o)$$

along with the statically determined fluents

$$\begin{array}{l} legal(xplayer, mark(1,1)),\ \dots,\ legal(oplayer, noop), \\ terminal,\ goal(xplayer, 100),\ \dots, \\ taken(1,1),\ \dots,\ taken(3,3),\ index(1),\ \dots,\ marker(o) \end{array}$$

and the actions

$$\begin{array}{l} does(xplayer, noop),\ does(oplayer, noop),\ does(xplayer, mark(1,1)),\ \dots \\ marking(1,1,x),\ \dots,\ marking(3,3,o) \end{array}$$

■

Next we present the main result of this paper, which provides a fully automatic translation for any given GDL description into a set of causal laws in action language $\mathcal{C}+$.

Definition 4. *Consider a GDL description G in which each occurrence of* `true`(f) *has been substituted by f. The* translation *of G into $\mathcal{C}+$ is obtained as follows.*

1. *For every clause f `:-` B in $grd(G)$ where f is a statically determined fluent (or a user-defined action predicate, respectively) introduce the law*

$$\textbf{caused}\ \ (f \subset B^+)\ \ \textbf{if}\ \ B^- \tag{2}$$

where B^+ (respectively, B^-) is the conjunction of all atoms that occur positively (respectively, negatively) in B.

2. *For every statically determined fluent f (and every user-defined action predicate, respectively) add the law*

$$\textbf{caused}\ \ \neg f\ \ \textbf{if}\ \ \neg f \tag{3}$$

3. *For every clause* `next`(f) `:-` B *in $grd(G)$ introduce the fluent dynamic law*

$$B\ \ \textbf{causes}\ \ f \tag{4}$$

4. *For every simple fluent f add the fluent dynamic law*

$$\bigwedge_j \neg B_j\ \ \textbf{causes}\ \ \neg f \tag{5}$$

where the conjunction ranges over the bodies (taken as conjunctions) of all clauses `next`(f) `:-` $B_j \in grd(G)$.[5]

[4] For the sake of clarity, we only mention some of the instances of the fluents and actions in this domain; the actual sets are larger and include irrelevant instances such as $cell(xplayer, xplayer, xplayer)$. When deploying our translation in practice, such instances should be detected with the help of a pre-processor, which can be built using a method described in [18] to compute the domains of the predicates in a GDL description.

[5] As usual, an empty body of a clause is equivalent to $\top$, and an empty conjunction is equivalent to $\bot$, which stands for the unconditional falsity.

5. *For every action* $does(r,a)$ *introduce the action dynamic law*
$$\textbf{caused } does(r,a) \textbf{ if } does(r,a) \wedge legal(r,a) \tag{6}$$
6. *For every pair of actions* $does(r,a)$ *and* $does(r,a')$ *such that* $a \neq a'$ *add the fluent dynamic law*
$$does(r,a) \wedge does(r,a') \textbf{ causes } \bot \tag{7}$$
7. *Add the fluent dynamic law*
$$\bigvee_r \bigwedge_a \neg does(r,a) \textbf{ causes } \bot \tag{8}$$

This construction deserves some explanation. According to law (2), if there is a clause for a statically determined fluent or user-defined action f, then there is a cause for f being implied by the positive atoms in the body of this clause if the negative atoms in the body hold.[6] Law (3) says that no such cause is needed for f to be false (more precisely, $\neg f$ suffices as cause for itself); this corresponds to the negation-as-failure principle used in GDL when a user-defined predicate is false in a situation where none of the clauses for this predicate applies. Note that (2) and (3) are static laws if f is a fluent, but action dynamic laws if f is an action.

Laws (4) and (5) say that for a fluent f to hold as a result of a move, there must be an applicable clause with head `next`(f), otherwise there is a cause for this fluent to be false. This is analogous to the negation-as-failure principle in GDL for negative effects of moves.

Law (6) is the formal way to express that action occurrences do not require a cause as long as they are legal. This corresponds to the treatment of moves as exogenous when they are added to a GDL program via (1) as outlined in Section 2.1. Finally, laws (7) and (8) together say that each player has to choose exactly one move in each step.

It is easy to see that the translation is modular. If we assume that the number of players, fluents, and actions is small in comparison to the number of game rules, then the resulting action theory is linear in the size of the GDL description. As an example, the translation of our Tic-Tac-Toe game is shown in Figure 2.[7]

4 Correctness

4.1 Syntactic Correctness

We begin by showing that our translation always results in a syntactically correct $\mathcal{C}+$ domain theory.

[6] For a subtle reason, which will be revealed in Section 4, the more straightforward translation of the corresponding GDL clause into **caused** f **if** $B^+ \wedge B^-$ is, in general, incorrect.

[7] Again we refrain from depicting the specifications of termination and goal values, for the sake of simplicity. Also not shown are the generic executability laws according to items 5–7 of Definition 4.

$$\begin{array}{rll}
\mathit{marking}(M,N,Z) & \textbf{causes} & \mathit{cell}(M,N,Z) \\
\mathit{cell}(M,N,Z) & \textbf{causes} & \mathit{cell}(M,N,Z) \\
\neg\mathit{marking}(M,N,Z) \wedge \neg\mathit{cell}(M,N,Z) & \textbf{causes} & \neg\mathit{cell}(M,N,Z) \\
\mathit{control}(\mathit{xplayer}) & \textbf{causes} & \mathit{control}(\mathit{oplayer}) \\
\neg\mathit{control}(\mathit{xplayer}) & \textbf{causes} & \neg\mathit{control}(\mathit{oplayer}) \\
\mathit{control}(\mathit{oplayer}) & \textbf{causes} & \mathit{control}(\mathit{xplayer}) \\
\neg\mathit{control}(\mathit{oplayer}) & \textbf{causes} & \neg\mathit{control}(\mathit{xplayer})
\end{array}$$

$$\begin{array}{ll}
\textbf{caused } \mathit{legal}(P,\mathit{mark}(M,N)) \subset \mathit{control}(P) \wedge \mathit{index}(M) \wedge \mathit{index}(N) & \\
& \textbf{if } \neg\mathit{taken}(M,N) \\
\textbf{caused } \mathit{legal}(\mathit{xplayer},\mathit{noop}) & \textbf{if } \neg\mathit{control}(\mathit{xplayer}) \\
\textbf{caused } \mathit{legal}(\mathit{oplayer},\mathit{noop}) & \textbf{if } \neg\mathit{control}(\mathit{oplayer}) \\
\textbf{caused } \neg\mathit{legal}(P,A) & \textbf{if } \neg\mathit{legal}(P,A) \\
\\
\textbf{caused } \mathit{marking}(M,N,x) \subset \mathit{does}(\mathit{xplayer},\mathit{mark}(M,N)) & \\
& \textbf{if } \top \\
\textbf{caused } \mathit{marking}(M,N,o) \subset \mathit{does}(\mathit{oplayer},\mathit{mark}(M,N)) & \\
& \textbf{if } \top \\
\textbf{caused } \neg\mathit{marking}(M,N,Z) & \textbf{if } \neg\mathit{marking}(M,N,Z) \\
\\
\textbf{caused } \mathit{taken}(M,N) \subset \mathit{marker}(Z) \wedge \mathit{cell}(M,N,Z) & \\
& \textbf{if } \top \\
\textbf{caused } \neg\mathit{taken}(M,N) & \textbf{if } \neg\mathit{taken}(M,N) \\
\\
\textbf{caused } \mathit{index}(1) & \textbf{if } \top \\
\ldots & \\
\textbf{caused } \mathit{marker}(o) & \textbf{if } \top
\end{array}$$

Fig. 2. Beginning with the fluent dynamic laws, these are the $\mathcal{C}+$ laws that result from translating the game rules for Tic-Tac-Toe in Section 2.1. Laws with variables represent the collection of their ground instances.

Proposition 1. *Let G be a valid GDL description. Given the signature of Definition 3, all laws constructed from G by Definition 4 are syntactically correct.*

Proof:

– By Definition 1, neither of the pre-defined predicates `legal`, `terminal`, or `goal` depends on `does` in G. Also, no other statically determined fluent depends on `does` according to Definition 3. Hence, in case f is statically determined fluent, (2) and (3) are syntactically correct static laws since f, B, $\neg f$, and $\bigwedge_j B_j$ are all fluent formulas. In case f is an action formula, both (2) and (3) are syntactically correct action dynamic laws.
– By Definition 3, if f occurs as argument in `next`(f) in some clause then f is a simple fluent. Hence, (4) and (5) are syntactically correct fluent dynamic laws since both f and $\neg f$ are fluent formulas without statically determined fluents.
– By Definition 3, $\mathit{does}(r,a)$ is an action formula. Hence, (6) is a syntactically correct action dynamic law.

– Laws (7) and (8) are syntactically correct fluent dynamic laws since $\bot$ is a fluent formula without statically determined fluents.

$\square$

4.2 From $\mathcal{C}+$ to Causal Theories

In order to prove that our translation from GDL results in a correct $\mathcal{C}+$ domain description, we need to recapitulate the precise meaning of the resulting set of causal laws. Following [11], the semantics is obtained in two steps. First, any $\mathcal{C}+$ domain D along with any non-negative integer m, which indicates the number of time steps to be considered, defines a set D_m of so-called *causal rules* as follows.

1. The signature of D_m consists in the pairs $i\!:\!c$ for all $i \in \{0, \ldots, m\}$ and fluents c, and the pairs $i\!:\!c$ for all $i \in \{0, \ldots, m-1\}$ and actions c. The intuitive meaning of $i\!:\!c$ is that c holds at time step i.
2. For every static law **caused** F **if** G, D_m contains the causal rule
$$i\!:\!F \ \Leftarrow\ i\!:\!G$$
for every $i \in \{0, \ldots, m\}$.[8]
3. For every action dynamic law **caused** F **if** G, D_m contains the causal rule
$$i\!:\!F \ \Leftarrow\ i\!:\!G$$
for every $i \in \{0, \ldots, m-1\}$.
4. For every fluent dynamic law G **causes** F, D_m contains the causal rule
$$i+1\!:\!F \ \Leftarrow\ i\!:\!G$$
for every $i \in \{0, \ldots, m-1\}$.
5. For every simple fluent f, D_m contains the two causal rules
$$\begin{aligned} 0\!:\!f &\Leftarrow \ 0\!:\!f \\ \neg 0\!:\!f &\Leftarrow \neg 0\!:\!f \end{aligned}$$

Intuitively, a causal rule $p \Leftarrow q$ means that there is a cause for p if q is true. The purpose of these rules is to allow the application of the principle of universal causation, according to which everything that holds must have a cause. This is made precise in the second step of defining the semantics for $\mathcal{C}+$. Let I be an interpretation, that is, a set of atoms from the signature for D_m as given above. The *reduct* D_m^I is the set of all heads of causal rules in D_m whose bodies are satisfied by I. A *model* of D_m is an interpretation I which is the unique model of D_m^I.

[8] If F is a formula, then $i : F$ denotes the result of placing "i :" in front of every action or fluent occurrence in F.

Example 1 (Continued) Consider the following small—and simplified—extract of the causal theory corresponding to the $\mathcal{C}+$ translation of Tic-Tac-Toe (cf. Figure 2):

$$\begin{array}{rcl}
0\!:\!\mathit{legal}(\mathit{xplayer}, \mathit{mark}(1,1)) \subset 0\!:\!\mathit{control}(\mathit{xplayer}) & & \\
 & \Leftarrow & \neg 0\!:\!\mathit{taken}(1,1) \\
0\!:\!\mathit{legal}(\mathit{xplayer}, \mathit{noop}) & \Leftarrow & \neg 0\!:\!\mathit{control}(\mathit{xplayer}) \\
\neg 0\!:\!\mathit{legal}(\mathit{xplayer}, \mathit{mark}(1,1)) & \Leftarrow & \neg 0\!:\!\mathit{legal}(\mathit{xplayer}, \mathit{mark}(1,1)) \\
\neg 0\!:\!\mathit{legal}(\mathit{xplayer}, \mathit{noop}) & \Leftarrow & \neg 0\!:\!\mathit{legal}(\mathit{xplayer}, \mathit{noop}) \\
 & & \\
0\!:\!\mathit{taken}(1,1) \subset 0\!:\!\mathit{cell}(1,1,x) & \Leftarrow & \top \\
\neg 0\!:\!\mathit{taken}(1,1) & \Leftarrow & \neg 0\!:\!\mathit{taken}(1,1) \\
 & & \\
0\!:\!\mathit{control}(\mathit{xplayer}) & \Leftarrow & 0\!:\!\mathit{control}(\mathit{xplayer}) \\
\neg 0\!:\!\mathit{control}(\mathit{xplayer}) & \Leftarrow & \neg 0\!:\!\mathit{control}(\mathit{xplayer}) \\
0\!:\!\mathit{cell}(1,1,x) & \Leftarrow & 0\!:\!\mathit{cell}(1,1,x) \\
\neg 0\!:\!\mathit{cell}(1,1,x) & \Leftarrow & \neg 0\!:\!\mathit{cell}(1,1,x)
\end{array}$$

Let these rules be denoted by D_0. Consider, then, the interpretation

$$I = \{0\!:\!\mathit{control}(\mathit{xplayer}), 0\!:\!\mathit{legal}(\mathit{xplayer}, \mathit{mark}(1,1))\}$$

The reduct D_0^I is

$$\begin{array}{l}
0\!:\!\mathit{legal}(\mathit{xplayer}, \mathit{mark}(1,1)) \subset 0\!:\!\mathit{control}(\mathit{xplayer}) \\
\neg 0\!:\!\mathit{legal}(\mathit{xplayer}, \mathit{noop}) \\
0\!:\!\mathit{taken}(1,1) \subset 0\!:\!\mathit{cell}(1,1,x) \\
\neg 0\!:\!\mathit{taken}(1,1) \\
0\!:\!\mathit{control}(\mathit{xplayer}) \\
\neg 0\!:\!\mathit{cell}(1,1,x)
\end{array}$$

Obviously, I is the unique model for this reduct, hence it is a model for D_0. The reader may verify that the following two are also models for this causal theory:

$$\begin{array}{rl}
I' & = \{0\!:\!\mathit{legal}(\mathit{xplayer}, \mathit{noop})\} \\
I'' & = \{0\!:\!\mathit{cell}(1,1,x), 0\!:\!\mathit{taken}(1,1), 0\!:\!\mathit{legal}(\mathit{xplayer}, \mathit{noop})\}
\end{array}$$

■

4.3 Game Developments and Causal Models Coincide

For the following we adopt from [11] a layered representation of interpretations for causal theories D_m. To this end, let $i\!:\!s_i$ denote the set of all fluent atoms of the form $i : f$ that are true in a given interpretations, and let $i : e_i$ denote the set of all action atoms of the form $i\!:\!a$ that are true. Any interpretation can then be represented in the form

$$(0\!:\!s_0) \cup (0\!:\!e_0) \cup (1\!:\!s_1) \cup (1\!:\!e_1) \cup \ldots \cup (m-1\!:\!e_{m-1}) \cup (m\!:\!s_m) \qquad (9)$$

This enables us to define a formal correspondence between models of a causal theory and game developments.

Definition 5. *Consider a game with terminal positions T and goal relation g. A game development $S_0 \xrightarrow{A_0} S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{n-1}} S_n$* coincides *with a model (9) if, and only if,*

1. $m = n$;
2. *S_i and s_i agree on all simple fluents, for all $i = 0, \ldots, n$;*
3. *A_i and e_i agree on all actions $does(r, a)$, for all $i = 0, \ldots, n-1$;*
4. *$S_i \notin T$ and $i\!:\!terminal \notin s_i$, for all $i = 0, \ldots, n-1$;*
5. *$S_n \in T$ iff $i\!:\!terminal \in s_n$; and*
6. *$(r, v, S_i) \in g$ iff $i\!:\!goal(r, v) \in s_i$, for all $i = 0, \ldots, n$.*

We are now prepared to state—and prove—our main result on the correctness of the mapping developed in Section 3 from GDL to $\mathcal{C}+$.

Theorem 1. *Consider a valid GDL game G with initial state S_0. For a given non-negative number $n \geq 0$ let D_n be the causal theory determined by the translation of G into $\mathcal{C}+$ with horizon n. With the addition of causal rules for the initial game state,*

$$\begin{array}{rl} 0\!:\!f \Leftarrow & \textit{for all } f \in S_0 \\ \neg 0\!:\!f \Leftarrow & \textit{for all simple fluents } f \notin S_0 \end{array} \tag{10}$$

every game development $S_0 \xrightarrow{A_0} S_1 \xrightarrow{A_1} \ldots \xrightarrow{A_{n-1}} S_n$ for G coincides with some causal model for $D_n \cup (10)$ and vice versa.

Proof: By induction on n.

For the base case $n = 0$, items 2, 5, and 6 of Definition 5 are the only relevant ones. In a lemma used in the proof for their Proposition 3, Giunchiglia et al. [11] show that the answer sets for a logic program are equivalent to the models of the causal theory that is obtained by

– identifying each clause f :- B with the rule $(f \subset B^+) \Leftarrow B^-$ and
– adding $\neg f \Leftarrow \neg f$ for every atom.

This observation can be directly applied to prove our base case: the construction in (10) ensures that the initial game state, S_0, agrees with the (unique) causal model for $D_0 \cup (10)$ on all simple fluents; furthermore, the construction of the static laws (2) and (3) from the respective clauses in G ensures that the game model coincides with the causal model on all statically determined fluents, in particular *terminal* and *goal*.[9]

For the induction step, consider a game development $S_0 \xrightarrow{A_0} \ldots \xrightarrow{A_{n-1}} S_n$ and a causal model $(0 : s_0) \cup (0 : e_0) \cup \ldots \cup (n : s_n)$ for D_n so that the two coincide and both S_n and s_n are non-terminal states. From the aforementioned

[9] At this point it becomes clear why we have to map clauses f :-B onto causal laws **caused** $f \subset B^+$ **if** B^- (cf. Footnote 6). The aforementioned observation in [11] would not hold if a clause f :-B were identified with the causal rule $f \Leftarrow B$. For example, the standard model for the program $\{p$:- $p\}$ is $\{\}$, whereas the causal theory $\{p \Leftarrow p, \neg p \Leftarrow \neg p\}$ admits two causal models, viz. $\{\}$ and $\{p\}$.

lemma in [11] it follows that the game model and the causal model coincide on the interpretation of the statically determined fluent *legal*. With this, the construction of the dynamic laws (6), (7), and (8) ensures that for every joint legal action A_n in state S_n there is a model $(0\!:\!s_0) \cup (0\!:\!e_0) \cup \ldots \cup (n\!:\!s_n) \cup (n\!:\!e_n) \cup (n+1 : s_{n+1})$ for D_{n+1}—and vice versa—so that A_n and e_n agree on all actions $does(r, a)$. Moreover, the construction of the action dynamic laws (2) and (3) along with the fluent dynamic laws (4) and (5) ensure that the updated states S_{n+1} and s_{n+1} agree on all simple fluents. Items 5 and 6 of Definition 5 again follow from the aforementioned lemma in [11]. □

5 Conclusion

The game description language GDL has been developed to provide a basis for the new, grand AI challenge of general game playing [10]. Although GDL can be viewed as a special-purpose action language, as yet it had not been formally related to any of the existing formalisms for reasoning about actions. In this paper, we have presented the first formal result in this regard, by giving a complete embedding of GDL into the action language $\mathcal{C}+$.

Our result paves the way for applying results from reasoning about actions to various aspects of general game playing, and conversely to use this AI challenge as an interesting and ambitious testbed for existing methods in reasoning about actions. Specifically, the embedding of GDL into $\mathcal{C}+$ allows to directly deploy an existing implementation, the Causal Calculator [11, 1], to reason about game descriptions in GDL. Further interesting directions for future work are, first, to investigate the formal relation between GDL and other existing action formalisms, possibly with $\mathcal{C}+$ as intermediary language on the basis of our result; and second, to investigate how the extension of GDL to imperfect information games [22] can be related to an existing action language suitable for representing both incomplete knowledge and sensing actions.

Acknowledgement. The author is indebted to Marina De Vos for her valuable comments on an earlier version of this paper. The author is the recipient of an Australian Research Council Future Fellowship (project number FT 0991348).

References

1. Varol Akman, Selim Erdogan, Joohyung Lee, Vladimir Lifschitz, and Hudson Turner. Representing the Zoo world and the Traffic world in the language of the Causal Calculator. *Artificial Intelligence*, 153(1–2):105–140, 2004.
2. Krzysztof Apt, H. A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 2, 89–148. Morgan Kaufmann, 1987.
3. Krzysztof Apt and Roland Bol. Logic programming and negation: A survey. *Journal of Logic Programming*, 19/20:9–71, 1994.
4. Keith Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, 293–322. Plenum Press, 1978.

5. Jim Clune. Heuristic evaluation functions for general game playing. In *Proceedings of the AAAI Conference*, 1134–1139, 2007.
6. Hilmar Finnsson and Yngvi Björnsson. Simulation-based approach to general game playing. In *Proceedings of the AAAI Conference*, 259–264, 2008.
7. Michael Gelfond. Answer sets. In F. van Harmelen, V. Lifschitz, and B. Porter, editors, *Handbook of Knowledge Representation*, 285–316. Elsevier, 2008.
8. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of the International Joint Conference and Symposium on Logic Programming (IJCSLP)*, 1070–1080, 1988.
9. Michael Gelfond and Vladimir Lifschitz. Representing action and change by logic programs. *Journal of Logic Programming*, 17:301–321, 1993.
10. Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62–72, 2005.
11. Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner. Nonmonotonic causal theories. *Artificial Intelligence*, 153(1–2):49–104, 2004.
12. Gregory Kuhlmann, Kurt Dresner, and Peter Stone. Automatic heuristic construction in a complete general game player. In *Proceedings of the AAAI Conference*, 1457–1462, 2006.
13. John Lloyd and R. Topor. A basis for deductive database systems II. *Journal of Logic Programming*, 3(1):55–67, 1986.
14. Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General Game Playing: Game Description Language Specification. Technical Report LG–2006–01, Computer Science Department, Stanford University, 2006. Available at: `games.stanford.edu`.
15. John McCarthy. *Situations and Actions and Causal Laws.* Stanford Artificial Intelligence Project, Memo 2, Stanford University,1963.
16. John McCarthy and Patrick Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.
17. Teodor Przymusinski. On the declarative semantics of deductive databases and logic programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, 193–216. Morgan Kaufmann, 1988.
18. Stephan Schiffel and Michael Thielscher. Fluxplayer: A successful general game player. In *Proceedings of the AAAI Conference*, 1191–1196, 2007.
19. Stephan Schiffel and Michael Thielscher. Automated theorem proving for general game playing. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 911–916, 2009.
20. Stephan Schiffel and Michael Thielscher. A multiagent semantics for the Game Description Language. In J. Filipe, A. Fred, and B. Sharp, editors, *Agents and Artificial Intelligence: Proceedings of ICAART*, volume 67 of *Communications in Computer and Information Science*, 44–55, 2009. Springer.
21. Stephan Schiffel, Michael Thielscher, and Dengji Zhao. Decomposition of multiplayer games. In *Proceedings of the Australian Joint Conference on Artificial Intelligence*, volume 5866 of *LNCS*, 475–484, 2009. Springer.
22. Michael Thielscher. A general game description language for incomplete information games. In *Proceedings of the AAAI Conference*, 994–999, 2010.
23. Michael Thielscher and Sebastian Voigt. A temporal proof system for general game playing. In *Proceedings of the AAAI Conference*, 1000–1005, 2010.