Game Description Language for Real-time Games

Jakub Kowalski
Institute of Computer Science
University of Wrocław, Poland
jakub.kowalski@cs.uni.wroc.pl

Andrzej Kisielewicz
Institute of Mathematics
University of Wrocław, Poland
andrzej.kisielewicz@math.uni.wroc.pl

Abstract

We present a simple extension of the Game Description Language GDL that makes it possible to describe a large variety of games involving a real-time factor. Apart from the formal syntax and semantics of our extension (acronymed rtGDL), we provide also a series of illustrative examples. In addition, we show that by combining rtGDL with the GDL-II extension one may obtain a simple and elegant universal game description language that enables the formalization of the rules of arbitrary n-player games allowing randomness, imperfect information and time-dependent events.

1 Introduction

The aim of *General Game Playing* (GGP) is to develop a system that can play a variety of games with previously unknown rules. Unlike standard AI game playing, where designing an agent requires special knowledge about the game, in GGP the key is to create a universal algorithm performing well in different situations and environments.

The beginning of the GGP field dates from 1968 with the work of Pitrat [Pitrat, 1968] concerning the class of arbitrary chess-like board games. In 2005 General Game Playing was identified as a new Grand Challenge of Artificial Intelligence and from this year the annual International General Game Playing Competition (IGGPC) has taken place to foster and monitor progress in this research area [Genesereth *et al.*, 2005].

For the purpose of GGP, a new language called the *Game Description Language* (GDL) [Love *et al.*, 2006] was developed. GDL has enough power to describe all turn-based, finite and deterministic n-player games with full information.Playing a game given by such a description requires not only developing a move searching algorithm, but also implementing a reasoning approach to understand the game rules in the sense of computing legal moves, the state update function, and the goal function. In 2010, GDL-II, the extension to GDL removing deterministic and full information restrictions was proposed in [Thielscher, 2010].

Currently, the various aspects of General Game Playing are a popular research topic. A successful method of learning neural networks to play 29 from 49 tested Atari 2600 games at or above the human level was presented in [Mnih *et al.*, 2015]. The recently launched General Video Game AI Competition, focusing on learning how to play simple video games by simulation and experience (rather then learning knowledge from the rules), is summarized in [Perez *et al.*, 2015].

The classic, GDL-based branch of GGP is constantly aimed for improving performance of the players [Björnsson and Schiffel, 2013; Kowalski and Szykuła, 2013], and developing better playing algorithms in both perfect information [Finnsson, 2012; Tak *et al.*, 2012] and imperfect information domain [Geißer *et al.*, 2014]. For a survey of General Game Playing and International GGP Competition, its history, achievements and actual research challenges we refer to [Genesereth and Thielscher, 2014] and [Genesereth and Björnsson, 2013].

In this paper, we follow the general idea of [Thielscher, 2010] to extend substantially the class of described games with minimal modifications to the language and communication protocol. We extend GDL to a Real-time Game Description Language (rtGDL), which can represent games involving time-dependent events, where the exact moment of an action is relevant. This concerns most current video games (such as RTS, FPS and RPG) for which computers are programmed to play these games using a specially designed algorithm [Ontanón *et al.*, 2013; Hingston, 2010].

We add only two new keywords, and an additional argument for the other three standard GDL relations. What we have achieved is that no real number arithmetic is required inside the rule engine, which unlike some other proposed extensions, e.g. [Thielscher and Zhang, 2010], preserve the pure declarative character of GDL. We present the formal syntax and semantics of this language, and also sketch a construction of the union of rtGDL and GDL-II, which yields the most universal game description language in the GDL family.

We assume basic familiarity with GGP and GDL structure. The paper is organized as follows. In the next section we introduce rtGDL as an extension of GDL. Section 3 provides formal semantics of Real-time GDL, while Section 4 covers the subject of the execution model. In the next section, the extension is illustrated by a series of examples. Section 6 contains a sketch of joining rtGDL with GDL-II yielding a language capable of describing games with both imperfect information and real-time factor. Concluding remarks are given in Section 7.

2 From GDL to rtGDL

Game Description Language [Love *et al.*, 2006], developed for the purpose of the General Game Playing Competition [Genesereth *et al.*, 2005] is a formal language to describe rules of any turn-based, finite, deterministic, n-player game with simultaneous moves and perfect information. GDL is a high-level, strictly declarative language using logic programming-like syntax based on the Knowledge Interchange Format (KIF [Genesereth and Fikes, 1992]). Every game description contains declarations of player roles, the initial game state, legal moves, state transition function, terminating conditions and the goal function.

GDL does not provide any predefined functions: neither arithmetic expressions nor game-domain specific structures like board or card deck. Every function and declaration must be defined explicitly from scratch, and the only keywords used to define the game are presented in Table 1. This ensures that, given the game rules, any additional, background knowledge is not required for the player.

`role(R)`	`R` is a player
`init(F)`	fact `F` is true in the initial state
`true(F)`	fact `F` is true in the current state
`legal(R,M)`	in the current state `R` can perform move `M`
`does(R,M)`	`R` performed move `M` in the previous state
`next(F)`	`F` will be true in the next state
`terminal`	current state is terminal
`goal(R,N)`	player `R` score is `N`

Table 1: GDL keywords

A surprisingly simple, but very powerful extension of GDL, called GDL-II (for Game Description Language with Incomplete Information) proposed by Thielscher [Thielscher, 2010; 2011a; 2011b], removes the restrictions that games have to be deterministic and with full information. This extension adds two additional keywords, and it slightly changes the communication protocol.

The extension of GDL presented in this paper is intended to remove another important restriction of GDL games that they need to be turn-based. We preserve the purely declarative style, so that the state computing inference engine can remain unchanged. In general, dealing with real-time games may involve real number arithmetic, which cannot be encoded in pure GDL in a simple way. The proposed approach requires real arithmetic on the level of search engine reasoning only. On the level of the rule engine, numbers are treated like standard terms without additional semantics (in a manner similar to natural numbers in a `goal` relation).

`init(T,F)`	`F` is true in the initial state for the time `T`
`true(T,F)`	`F` is true in the current state for the time `T`
`next(T,F)`	`F` is true in the next state for the time `T`
`infinity`	stands for ∞ when used as a time value
`expired(F)`	holds when `F` becomes obsolete

Table 2: The rtGDL keywords: the top three are modified GDL keywords, while the last two are new keywords

Changes between rtGDL and GDL keywords are summarized in Table 2. Keywords that are not listed: `role`, `legal`, `does`, `terminal` and `goal` remain unchanged. Parameter `T`$\in \mathbb{R}^+ \cup \{\infty\}$ linked with the currently holding fact `F` encodes the *lifetime* of this fact, that is the time for which this fact will be true. After the time is over, the state update is performed and the additional relation `expired(F)` indicates that `F` have just expired and allows changes based on this knowledge.

The `expired` keyword can be seen as just a syntactic sugar for the expression `true(0,F)`, however its introduction simplifies games rules (there is no more need for conditioning time value) and provides unambiguous semantics for obsolete facts. Thus, syntactic restrictions guarantees that 0 time value never occurs in `true` relation and `expired` is used instead.

State updates are performed after a player makes a move or when some fact becomes obsolete. Before such an update is computed, all times assigned to the set of holding facts are properly updated according to the time since the last update. The detailed semantics of rtGDL language is provided in the next two sections. Before going into formal details, the reader may want to see first examples in Section 5.

2.1 Formal Syntax Restrictions

Descriptions of rtGDL games use the standard syntax of logic programs, including negation and inequality (denoted as `distinct`). For the sake of readability, we will use Prolog convention rather than standard GDL code convention, that is, clauses are written in infix form, variables are uppercase letters, and function names start with lowercase letters.

As in the case of the standard GDL, certain syntactic restrictions have to be fulfilled to ensure that the game specification has an unambiguous interpretation, and all derivations are finite. First, exactly the same restrictions as in GDL and GDL-II are imposed on the keywords (a new keyword `expired` is restricted in the same way as `true`).

- `role` only appears in the head of ground atomic sentences;
- `init` only appears in the head of clauses and does not depend on `true`, `legal`, `does`, `next`, `terminal`, `goal` or `expired`;
- `true` only appears in the body of clauses;
- `does` only appears in the body of clauses and does not depend on `legal`, `terminal` or `goal`;
- `next` only appears in the head of clauses;
- `expired` only appears in the body of clauses.

Also, we adopt the convention that to be considered as *valid*, rtGDL game description must be *stratified* [Apt *et al.*, 1988], *allowed* [Lloyd and Topor, 1986], and satisfy the general *recursion restriction* (see [Schiffel and Thielscher, 2014, Definition 3.]). These restrictions ensure that game rules can be effectively and unambiguously interpreted by a state transition system.

Despite introducing time values, which in principle can be arbitrary real numbers, all derivations remain finite and decidable. This is due to the fact, that in each step there are

only a finite number of time values, which appear in the form of ground constants. In the next section we describe a unique game model that is obtained from a valid rtGDL game description using the concept of a unique stable model as in [Schiffel and Thielscher, 2014].

3 Semantics: A Game model for rtGDL

We show that rtGDL may be seen as a straightforward extension of GDL. As in GDL, any game description defines a finite set of domain-dependent function symbols and constants, which determines a (usually infinite) set of ground symbolic expressions Σ. These ground terms can represent players, moves or parts of the game state. However, unlike in the standard GDL, the game states are not simply subsets of Σ. Every individual feature of the game state has assigned a *lifetime*, which is a real number value or the infinity symbol. Such feature holds for this period of time, and after that (unless other events occur) it disappears from the state. Moreover, initially declared real numbers which occur in Σ are not the only ones. Any real number, given by the outside environment as updated lifetime, can appear as a constant later during the game.

Therefore, we define an rtGDL game state to be a finite subset of $\mathbb{R}_+^\infty \times \Gamma$, where $\mathbb{R}_+^\infty = (0, \infty]$ (including ∞), and $\Gamma \subseteq \Sigma(\mathbb{R}_+^\infty)$ where $\Sigma(\mathbb{R}_+^\infty)$ is a set of ground terms of the game, with the set of constants extended by $\mathbb{R}_+^\infty$.

Now, in introducing the elements of the game model, we follow the constructions for GDL and GDL-II given in [Schiffel and Thielscher, 2014]

- $R \subseteq \Sigma$ (the *roles*);
- $s_0 \subseteq \mathbb{R}_+^\infty \times \Sigma$ (the *initial position*);
- $t \subseteq \mathcal{P}_{\text{fin}}(\mathbb{R}_+^\infty \times \Gamma)$ (the *terminal positions*);
- $l \subseteq R \times \Gamma \times \mathcal{P}_{\text{fin}}(\mathbb{R}_+^\infty \times \Gamma)$ (the *legality relation*);
- $u : \mathbb{R}_+^\infty \times (R \nrightarrow \Gamma) \times \mathcal{P}_{\text{fin}}(\mathbb{R}_+^\infty \times \Gamma) \mapsto \mathcal{P}_{\text{fin}}(\mathbb{R}_+^\infty \times \Gamma)$ (the *update function*);
- $g \subseteq R \times \mathbb{N} \times \mathcal{P}_{\text{fin}}(\mathbb{R}_+^\infty \times \Gamma)$ (the *goal relation*).

(where $\mathcal{P}_{\text{fin}}(A)$ denotes the finite subsets of A) The legality relation $l(r, m, S)$, states that in the position S, a player r can perform move m. Given a time Δt since the last state update, and a partial function of the joint moves performed by the players $M : (R \nrightarrow \Gamma)$, the state update function $u(t, M, S)$ determines the updated position. Finally, the relation $g(r, n, S)$ defines the payoff of player r in the game state S.

3.1 State transition system

Let $\mathcal{G}$ be a valid game description of an n-player rtGDL game. The players' roles are defined by the derivable instances of `role(R)`. A *game state* $S = \{(t_1, f_1), \ldots, (t_k, f_k)\}$ is encoded as a set of facts with assigned lifetimes. At the beginning of the game, the state is composed of the derivable instances of `init(T,F)`. We use the keyword `true` to extend the game description $\mathcal{G}$ by the facts resulting from S, which form the set

$$S^{\texttt{true}} \stackrel{\text{def}}{=} \{\texttt{true}(t_1, f_1), \ldots, \texttt{true}(t_k, f_k)\}.$$

Instances of `goal(R,N)` derivable from $\mathcal{G} \cup S^{\texttt{true}}$ assign to every player `R` its goal value `N` in this state. If `terminal` is derivable from $\mathcal{G} \cup S^{\texttt{true}}$, then the state S is a terminal position. Computing legal actions, requires derivable instances of `legal(R,M)`, for every role `R` and move `M`.

The state of the game remains unchanged until an *event* occurs. There are two types of events, one caused by players sending their moves, and the other resulting from expiration of some facts in $S^{\texttt{true}}$. Let Δt be the time to the first occurrence of an event.

When a subset of players $r_{i_1}, \ldots, r_{i_l}$, where $\forall j \; i_j \in \{1, \ldots, n\}$, perform moves $m_{i_1}, \ldots, m_{i_l}$, then

$$M^{\texttt{does}} \stackrel{\text{def}}{=} \{\texttt{does}(r_{i_1}, m_{i_1}), \ldots, \texttt{does}(r_{i_l}, m_{i_l})\}.$$

Let us notice that $M^{\texttt{does}} = \emptyset$ if no player sends a move.

Performing state update requires updating lifetime of every holding fact, and information about facts that expired. These two sets are defined in a following way:

$$\mu(S, \Delta t) = \{\texttt{true}(t_i - \Delta t, f_i) : \texttt{true}(t_i, f_i) \in S^{\texttt{true}} \wedge t_i > \Delta t\}$$

$$S^{\text{exp}}_{\Delta t} \stackrel{\text{def}}{=} \{\texttt{expired}(f_i) : \texttt{true}(t_i, f_i) \in S^{\texttt{true}} \wedge t_i \leq \Delta t\}$$

where $\mu(S, \Delta t)$ updates the facts lifetimes and $S^{\text{exp}}_{\Delta t}$ contains expired facts. The updated position is composed of instances of `next(T,F)` derivable from $\mathcal{G} \cup M^{\texttt{does}} \cup \mu(S, \Delta t) \cup S^{\text{exp}}_{\Delta t}$.

The semantics of rtGDL, arising from the above description is presented in the definition below. We follow [Schiffel and Thielscher, 2014] using the unique finite stable model $\text{SM}[\mathcal{G}]$ rather then the derivability relation as in [Thielscher, 2010]. For details concerning the existence of stable models for logic programming the reader is referred to [Gelfond and Lifschitz, 1988].

Definition 1 *Let $\mathcal{G}$ be a valid rtGDL specification, whose signature determines the set of ground terms Σ, and $\Gamma \subseteq \Sigma(\mathbb{R}_+^\infty)$. The semantics of $\mathcal{G}$ is the state transition system (R, s_0, t, l, u, g) given by*

- $R = \{r \in \Sigma : \texttt{role}(r) \in \text{SM}[\mathcal{G}]\}$;
- $s_0 = \{(t, f) \in \mathbb{R}_+^\infty \times \Sigma : \texttt{init}(t, f) \in \text{SM}[\mathcal{G}]\}$;
- $t = \{S \in 2^{\mathbb{R}_+^\infty \times \Gamma} : \texttt{terminal} \in \text{SM}[\mathcal{G} \cup S^{true}]\}$;
- $l = \{(r, m, S) : \texttt{legal}(r, m) \in \text{SM}[\mathcal{G} \cup S^{true}]\}$, *for all $r \in R$, $m \in \Gamma$ and $S \in \mathcal{P}_{\text{fin}}(\mathbb{R}_+^\infty \times \Gamma)$;*
- $u(\Delta t, M, S) = \{(t, f) : \texttt{next}(t, f) \in \text{SM}[\mathcal{G} \cup M^{does} \cup \mu(S, \Delta t) \cup S^{exp}_{\Delta t}]\}$, *for all $M : (R \nrightarrow \Gamma)$, $S \in \mathcal{P}_{\text{fin}}(\mathbb{R}_+^\infty \times \Gamma)$, and minimal Δt such that $M^{does} \cup S^{exp}_{\Delta t} \neq \emptyset$;*
- $g = \{(r, n, S) : \texttt{goal}(r, n) \in \text{SM}[\mathcal{G} \cup S^{true}]\}$, *for all $r \in R$, $n \in \mathbb{N}$ and $S \in \mathcal{P}_{\text{fin}}(\mathbb{R}_+^\infty \times \Gamma)$.*

This defines a formal semantics for an abstract game model of rtGDL. The calculation of function μ takes place outside the state computation and is a part of the execution model, which is the subject of the next section

4 Execution Model

An execution model of rtGDL is designed to handle real-time events, which makes it a bit more complex than the standard execution model of GDL. After the initial state s_0 is computed, the timer is turned on and the game starts. The initial state is treated as the current one, until some event occurs which stops the timer. This event can be triggered by players who sent their moves, or it can be scheduled, known in advance, expiration of some state features. At the moment of the event, the state update is performed according to the game model semantics with Δt equal to time indicate by the timer. The timer is then turned on again, and the whole process repeats until a game reaches a terminal position. The final scores of the players are defined by the goal relation.

It should be noticed, that scheduled updates caused by fact expirations are silent, i.e. there is no message to the players indicating that the update took place. Such messages are unnecessary due to the fact that players have all the informations to perform such updates by themselves, based on control times sent by the Game Manager.

The straightforward implementation of a Game Manager according to this model looks as follows:

1. Send to the players the rtGDL code containing game description with information about their roles and binding timelimits. Set $S := S_0$. Turn the timer on.
2. After the time for the players to familiarize with game has passed, inform the players of the game beginning.
3. Restart the timer. Calculate t_u which is the minimal time for some position feature from S to become obsolete.
4. Wait until some players send their moves or the timer is equal to t_u. Set $M :=$ 'players moves', $\Delta t :=$ 'timer indications'.
5. If $M \neq \emptyset$ inform the players about the performed moves M and the current game time (since the beginning of the game).
6. Perform state update by setting $S := u(\Delta t, M, S)$.
7. If S is not a terminal state go to step 3. Otherwise, compute the payoff for every player accordingly to g relation and finish the game.

The game flow is unequivocally dictated according to the Game Manager's timer. In a case when a move sent by a player is not legal in the current state, update is not performed. There are no restrictions on the number of moves the players can send, and all of them should be considered by the Game Manager accordingly to their arrival times.

4.1 Communication Protocol

There are slight differences between the GDL and rtGDL communicates, but the main idea and architecture remains the same. For the detailed description of the GDL communication protocol we refer to [Genesereth *et al.*, 2005].

In rtGDL protocol there are no new types of messages, and all player replies remain unchanged. In the Game Manager message of type `START`, the semantic of `PLAYCLOCK` parameter has changed. Now it is used as a multiplier of fact lifetimes, i.e. to get the number of seconds before a fact becomes obsolete, its current lifetime has to by multiply by `PLAYCLOCK`. With `PLAYCLOCK` 20, fact `true(1.5,control(white))` becomes expired after 30 seconds, while fact `true(0.5,cooldown)` after 10 seconds. This preserves a very desirable possibility of controlling the game pace without changes in the game code. The semantic of the `STARTCLOCK` parameter remains unchanged.

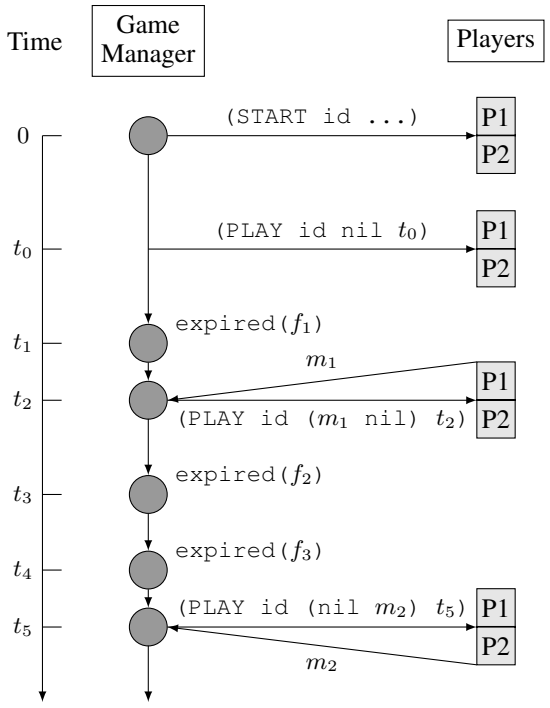

Figure 1: Visualization of communication during some rtGDL game. Circles in the Game Manager's column indicates a state update.

The `PLAY` and `STOP` messages are extended, and now their last, third, parameter is a game time as `PLAYCLOCK` multiplyer, i.e. the number of seconds (as a floating point number) from the beginning of the game to the moment when the message was sent by the Game Manager divided by the current `PLAYCLOCK` value.

The second argument of `PLAY`/`STOP` message is a list of actions taken by all players. This list can be set to `NIL` in two special cases: when this is the initial game message indicating the end of the preparation phase, or it is an answer to a player who sent an illegal move (other players should not receive this message). In a real-time gaming the Game Manager is receiving players' moves sequentially rather than simultaneously, which results that the lists of performed actions sent in `PLAY`/`STOP` messages are only partially filled. If a player on a position k made no move, his move is simply denoted as `NIL` which allows that the length of the list remains constant and equal to number of the players in the game. (In fact, the lists may be replaced by *role–move* pairs.)

An example of communication between Game Manager and players during exemplary rtGDL game with two players is presented in Figure 1.

5 Examples

To illustrate the expressiveness of Real-time Game Description Language, we provide a series of short examples describing real-time elements appearing in many games.

Example 1. Turn-based games

First, we demonstrate how to translate a turn-based GDL game into rtGDL. For this purpose we use the standard Tic-tac-toe game. The following code describes the most important parts of the suitable rtGDL description:

```
role(xplayer).
role(oplayer).
init(infinity,cell(1,1,blank)).
...
init(infinity,cell(3,3,blank)).
init(1.0,control(xplayer))

next(1.0,control(S))
    ⇐ does(R,M)
    ∧ role(S)
    ∧ distinct(R,S).

next(infinity,cell(M,N,R))
    ⇐ does(R,mark(M,N))
    ∧ true(infinity,cell(M,N,blank)).
next(infinity,cell(M,N,C))
    ⇐ true(infinity,cell(M,N,C)).
    ∧ distinct(C,blank).
next(infinity,cell(M,N,blank))
    ⇐ true(infinity,cell(M,N,blank))
    ∧ does(R,mark(J,K))
    ∧ (distinct(J,M) ∨ distinct(K,N)).

legal(R,mark(M,N))
    ⇐ true(infinity,cell(M,N,blank))
    ∧ true(T,control(R)).

terminal ⇐ expired(control(R)).

goal(R,0) ⇐ expired(control(R)).
goal(S,100)
    ⇐ expired(control(R))
    ∧ role(S)
    ∧ distinct(R,S).

terminal ⇐ ¬boardopen.
terminal ⇐ line(R).
goal(R,100) ⇐ line(R)).
...
```

Information about the board is stored as a set of forever holding facts, while the time for making a move is determined by the lifetime of the `control` fact. State update is performed only in two cases, after the current player performs a move – then the board should be accordingly updated and control switched to the other player; or after the current player exceeds timeout, which cause expiration of `control` and immediate end of the game.

Other terminal and goal conditions, omitted in the above listing, remain as in the standard Tic-Tac-toe, with `boardopen` holding when there is some blank space left on the board, and `line(R)` holding when player `R` composed a line of his symbols. It is worth to note that in the presented game translation (which is not the only one possible) there is no need for the special `NOOP` move by a player without control.

Example 2. Chess clock

Instead of allocating a constant time for every move, rtGDL makes it possible to implement a chess clock, and count accumulated move time for each player.

```
role(white).
role(black).
init(infinity,timer(R,120.0)) ⇐ role(R).
init(infinity,control(white)).
init(120.0,clock).

next(infinity,timer(R,T))
    ⇐ does(R,M)
    ∧ true(T,clock).
next(infinity,timer(S,T))
    ⇐ true(infinity,timer(S,T))
    ∧ does(R,M)
    ∧ distinct(R,S).

next(T,clock)
    ⇐ true(infinity,timer(S,T))
    ∧ does(R,M)
    ∧ distinct(R,S).

terminal ⇐ expired(clock).
goal(R,0)
    ⇐ expired(clock)
    ∧ true(infinity,control(R)).
```

The remaining times for players are stored in the `timer` relation, while the passing time is counted using the lifetime of the `clock`. After every turn, the remaining `clock` time is copied to the `timer` of the player with control, and it is set as the remaining time of the other player.

Example 3. Self-appearing and disappearing objects

The following rules specify an entity that appears and disappears in constant amounts of time.

```
init(1.0,appear(cheshireCat))

next(3.0,disappear(C)) ⇐ expired(appear(C)).
next(T,disappear(C)) ⇐ true(T,disappear(C)).

next(1.0,appear(C)) ⇐ expired(disappear(C)).
next(T,appear(C)) ⇐ true(T,appear(C)).
```

Example 4. Respawning objects

Slightly more complicated case describes respawning objects, e.g. items lying in the game area, reappearing some time after a player takes them, which is very popular concept in FPS games.

```
init(infinity,onMap(2,3,yellowArmor,5.5))
init(infinity,onMap(11,11,redArmor,10.0))
```

```
3
4 next(T,toRespawn(X,Y,A,T))
5     ⇐ true(infinity,onMap(X,Y,A,T))
6     ∧ playerAtPosition(X,Y).
7 next(T,toRespawn(X,Y,A,S))
8     ⇐ true(T,toRespawn(X,Y,A,S)).
9
10 next(infinity,onMap(X,Y,A,T))
11     ⇐ true(infinity,onMap(X,Y,A,T))
12     ∧ ¬playerAtPosition(X,Y).
13 next(infinity,onMap(X,Y,A,T))
14     ⇐ expired(toRespawn(X,Y,A,T))
```

Example 5. Self-moving objects

Another example contains an object traveling through the discrete space grid with fixed speed and refresh rate (`plus` is the $+$ relation defined on a finite subset of natural numbers).

```
1 speed(xwing,2,2,1)
2 init(infinity,space(35,86,40,xwing))
3 init(0.5,refresh)
4
5 next(0.5,refresh) ⇐ expired(refresh)
6 next(T,refresh) ⇐ true(T,refresh)
7
8 next(infinity,space(F,G,H,S))
9     ⇐ true(infinity,space(X,Y,Z,S))
10    ∧ expired(refresh)
11    ∧ speed(S,A,B,C)
12    ∧ plus(X,A,F)
13    ∧ plus(Y,B,G)
14    ∧ plus(Z,C,H).
15 next(infinity,space(X,Y,Z,S))
16    ⇐ true(infinity,space(X,Y,Z,S))
17    ∧ ¬expired(refresh).
```

Example 6. Player ordering time-taking events

In many games, after giving an order, the player is free to make other actions, while the execution of the order needs some time to complete. As an example of such situation we present a partial code of an RTS game where the player can order to build a barracks or a blacksmith (`greaterEqual` is the $\geq$ relation defined on a finite subset of natural numbers).

```
1 cost(barracks,160,6.0).
2 cost(blacksmith,140,7.0).
3 init(infinity,gold(500)).
4
5 legal(player,build(B))
6     ⇐ true(infinity,gold(G))
7     ∧ cost(B,F,T)
8     ∧ greaterEqual(G,F).
9 next(T,underConstruction(B))
10    ⇐ does(player,build(B))
11    ∧ cost(B,G,T).
12 next(T,underConstruction(B))
13    ⇐ true(T,underConstruction(B)).
14
15 next(infinity,constructed(B))
16    ⇐ expired(underConstruction(B)).
17 next(infinity,constructed(B))
18    ⇐ true(infinity,constructed(B)).
```

6 Real-time GDL with Imperfect Information

Following the extension from GDL to GDL-II proposed by Thielscher in [Thielscher, 2010], we can also drop the deterministic and perfect information restrictions. Below we sketch the rules of another GDL extension, called rtGDL-II, which is a combination of Real-time GDL and GDL with Incomplete Information, and in consequence, the most universal language in GDL family.

6.1 Nondeterminism

The nondeterminism introduced by GDL-II rules relies on the special `random` role operated by the Game Manager. Real-time GDL can be extended in a similar way, however in this case the randomness may be introduced at two dimensions. We may describe not only *which* action is to be drawn (like in GDL-II), but also *when* it should be drawn.

The first case can be solved in a straightforward way, by assuming that if in some state the `random` role has a non-empty set of legal moves, then immediately one of these moves should be made with uniform probability and the state update should be performed. Such usage of randomness is suitable for generating discrete random variables, e.g. rolling a dice or shuffling a deck of cards.

For the latter case we may adopt the following solution. If the `random` player's move drawn by the Game Manager is a relation with arity greater then 0, and the first argument of this relation is a non-negative real number t, then the state update should be performed after a random time t' obtained using the continuous uniform distribution $\mathcal{U}(0,t)$.

As an example consider four legal actions of `random`: `wait`, `move(right)`, `shot(0,pistol)`, `shot(2,blaster)`. Every action has a 25% chance to be chosen. If one of the first three actions will is to be made, then the Game Manager should perform state update immediately. In the remaining case, a random value t' generated with distribution $\mathcal{U}(0,2)$ should be drawn. After the time t', if the move is still legal, the state update should be performed. In the case when the move is no longer legal (due to the players' actions or some other events) no state update is performed, and the action is wasted.

6.2 Imperfect Information

In orded to introduce imperfect information we follow directly Thiescher's approach. The relation `sees` serves as a container for players' percepts, and the Game Manager as a feedback for the player sends his percepts instead of the joint move.

However, as the game is not turn-based any more, the Game Manager cannot simply send everyone messages after obtaining some player's move, because this will give everyone a clue that someone made a move. We may want to avoid this in games where just knowing that other player made an action is a crucial piece of information.

Therfore we may adopt the following solution. For every player two sets of percepts are computed. One containing percepts after updating a state according to the game rules, and the other with the percepts from the same moment, but with the assumption that the state update did not occur. The latter is possible, because apart from the players' actions the Game Manager can predict the state in any future moment. A message containing the updated percept is send to the player only in the case when this two sets differ.

7 Conclusion

With introduction of GDL-II, it was claimed that the GDL language can be considered complete [Schiffel and Thielscher, 2014], and additional elements can only serve for simplifying description or will be forcing setting extensions far beyond the concept of General Game Playing (e.g. open-world games or physical games like in General Video Game Playing [Perez *et al.*, 2015]).

The real-time extension presented in this paper preserves the core idea of GDL – a purely logical way of reasoning about the state and a Game Manager based execution model. At the same time it is a larger extension than GDL-II, allowing a game to have an infinite number of states and the players to have an infinite number of actions. By introducing time based events and by giving relevance to the move ordering it makes it possible to describe properly many real world situations. A good example is the game of chicken, which in continuous-time space takes a deeper meaning.

Extending general games with real-time events allows to the modeling of elements of the popular computer games, which are currently used as a test-bed for dedicated AI players, e.g. Unreal Tournament 2004 [Hingston, 2010], TORCS [Loiacono *et al.*, 2010] or Super Mario Bros [Togelius *et al.*, 2013]. In some cases like in Starcraft [Ontanón *et al.*, 2013], where real-world physics is minimized, it theoretically allows modeling the entire game.

Beyond the usage in General Game Playing, the correlations between the GDL and game theory [Thielscher, 2011b] and Multiagent Systems [Schiffel and Thielscher, 2010] are often pointed out. A goal for GDL is to become a universal description language which can describe as large class of game-like problems as possible, at the same time remaining compact, high-level and machine-processable. We presented the next step into such a generalization of problems description, which brings the class of games covered by the GDL family closer to real-time game theory (e.g. Differential Games [Isaacs, 1999], continuous time repeated games [Bergin and MacLeod, 1993], extensive games in continuous time [Simon and Stinchcombe, 1989]) and Real Time Multiagent Systems [Julian and Botti, 2004].

In this paper we have introduced a new language rtGDL with possible extension to rtGDL-II, and addressed a new ambitious challenge for General Game Playing systems. Taking into account response time of message arrival, creates for the players an "act-wait dilemma", i.e. given more computation time, the game tree could be explored better, however this may cause some promising paths to no longer be available due to the in-game events or the other players actions. This scenario requires developing new general-case solutions, like for example usage of real-time Monte Carlo Tree Search [Pepels *et al.*, 2014]. It makes real-time general gaming a very hard, but interesting area of further General Game Playing research.

Acknowledgements

We thank Steve Draper for reading the first draft of the paper and many valuable remarks.

This work was supported by Polish National Science Centre grants No 2014/13/N/ST6/01817 and No 2012/07/B/ST1/03318.

References

[Apt *et al.*, 1988] Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Foundations of deductive databases and logic programming. chapter Towards a Theory of Declarative Knowledge, pages 89–148. 1988.

[Bergin and MacLeod, 1993] James Bergin and W. Bentley MacLeod. Continuous time repeated games. *International Economic Review*, pages 21–37, 1993.

[Björnsson and Schiffel, 2013] Yngvi Björnsson and Stephan Schiffel. Comparison of GDL Reasoners. In *Proceedings of the IJCAI-13 Workshop on General Game Playing (GIGA'13)*, pages 55–62, 2013.

[Finnsson, 2012] Hilmar Finnsson. Generalized Monte-Carlo Tree Search Extensions for General Game Playing. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, pages 1550–1556, 2012.

[Geißer *et al.*, 2014] Florian Geißer, Thomas Keller, and Robert Mattmüller. Past, Present, and Future: An Optimal Online Algorithm for Single-Player GDL-II Games. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI)*, pages 357–362, 2014.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, volume 88, pages 1070–1080, 1988.

[Genesereth and Björnsson, 2013] Michael Genesereth and Yngvi Björnsson. The International General Game Playing Competition. *AI Magazine*, 34(2):107–111, 2013.

[Genesereth and Fikes, 1992] Michael Genesereth and Richard E. Fikes. Knowledge Interchange Format Version 3.0 Reference Manual. Technical Report LG-1992-01, Stanford Logic Group, 1992.

[Genesereth and Thielscher, 2014] Michael Genesereth and Michael Thielscher. *General Game Playing*. Morgan & Claypool, 2014.

[Genesereth *et al.*, 2005] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26:62–72, 2005.

[Hingston, 2010] Philip Hingston. A new design for a turing test for bots. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 345–350. IEEE, 2010.

[Isaacs, 1999] Rufus Isaacs. *Differential games: a mathematical theory with applications to warfare and pursuit, control and optimization*. Courier Corporation, 1999.

[Julian and Botti, 2004] Vicente Julian and Vicent Botti. Developing real-time multi-agent systems. *Integrated Computer-Aided Engineering*, 11(2):135–149, 2004.

[Kowalski and Szykuła, 2013] Jakub Kowalski and Marek Szykuła. Game Description Language Compiler Construction. In *AI 2013: Advances in Artificial Intelligence*, volume 8272 of *LNCS*, pages 234–245. 2013.

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

[Loiacono *et al.*, 2010] Daniele Loiacono, Pier Luca Lanzi, Julian Togelius, Enrique Onieva, David A Pelta, Martin V Butz, Thies D Lonneker, Luigi Cardamone, Diego Perez, Yago Sáez, et al. The 2009 simulated car racing championship. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2(2):131–147, 2010.

[Love *et al.*, 2006] Nathaniel Love, Timothy Hinrichs, David Haley, Eric Schkufza, and Michael Genesereth. General Game Playing: Game Description Language Specification. Technical Report LG-2006-01, Stanford Logic Group, 2006.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[Ontanón *et al.*, 2013] Santiago Ontanón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. A survey of real-time strategy game AI research and competition in Starcraft. *Computational Intelligence and AI in Games, IEEE Transactions on*, 5(4):293–311, 2013.

[Pepels *et al.*, 2014] Tom Pepels, Mark HM Winands, and Marc Lanctot. Real-time Monte Carlo Tree Search in Ms Pac-Man. *Computational Intelligence and AI in Games, IEEE Transactions on*, 6(3):245–257, 2014.

[Perez *et al.*, 2015] Diego Perez, Spyridon Samothrakis, Julian Togelius, Tom Schaul, S Lucas, Adrien Couëtoux, Jerry Lee, C Lim, and Tommy Thompson. The 2014 General Video Game Playing Competition. *Computational Intelligence and AI in Games, IEEE Transactions on*, 2015. To appear.

[Pitrat, 1968] Jacques Pitrat. Realization of a general game-playing program. In *IFIP Congress*, pages 1570–1574, 1968.

[Schiffel and Thielscher, 2010] Stephan Schiffel and Michael Thielscher. A Multiagent Semantics for the Game Description Language. In *Agents and Artificial Intelligence*, volume 67 of *Communications in Computer and Information Science*, pages 44–55. 2010.

[Schiffel and Thielscher, 2014] Stephan Schiffel and Michael Thielscher. Representing and Reasoning About the Rules of General Games With Imperfect Information. *Journal of Artificial Intelligence Research*, 49:171–206, 2014.

[Simon and Stinchcombe, 1989] Leo K. Simon and Maxwell B. Stinchcombe. Extensive Form Games in Continuous Time: Pure Strategies. *Econometrica*, 57(5):1171–1214, 1989.

[Tak *et al.*, 2012] Mandy J. W. Tak, Mark H. M. Winands, and Yngvi Björnsson. N-Grams and the Last-Good-Reply Policy Applied in General Game Playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):73–83, 2012.

[Thielscher and Zhang, 2010] Michael Thielscher and Dongmo Zhang. From general game descriptions to a market specification language for general trading agents. In *Agent-Mediated Electronic Commerce. Designing Trading Strategies and Mechanisms for Electronic Markets*, pages 259–274. 2010.

[Thielscher, 2010] Michael Thielscher. A General Game Description Language for Incomplete Information Games. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 994–999, 2010.

[Thielscher, 2011a] Michael Thielscher. GDL-II. *Künstliche Intelligenz*, 25:63–66, 2011.

[Thielscher, 2011b] Michael Thielscher. The General Game Playing Description Language is Universal. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1107–1112, 2011.

[Togelius *et al.*, 2013] Julian Togelius, Noor Shaker, Sergey Karakovskiy, and Georgios N. Yannakakis. The Mario AI Championship 2009-2012. *AI Magazine*, 34(3):89–92, 2013.