Game Description Language for Real-time Games

Jakub Kowalski, Andrzej Kisielewicz

GIGA

27 July 2015
Evolution of Game Description Language
GDL Evolution

GDL (Genesereth, Love, Pell, 2005)

*n*-player, turn-based, finite, deterministic, full information games
GDL Evolution

GDL (Genesereth, Love, Pell, 2005)
$n$-player, turn-based, finite, deterministic, full information games

GDL-II (Thielscher, 2010)
$n$-player, turn-based, finite, deterministic, full information games
GDL Evolution

GDL (Genesereth, Love, Pell, 2005)
n-player, turn-based, finite, deterministic, full information games

GDL-II (Thielscher, 2010)
n-player, turn-based, finite, deterministic, full information games

rtGDL (2015)
n-player, turn-based, finite, deterministic, full information games
GDL Evolution

GDL (Genesereth, Love, Pell, 2005)
$n$-player, turn-based, finite, deterministic, full information games

GDL-II (Thielscher, 2010)
$n$-player, turn-based, finite, deterministic, full information games

rtGDL (2015)
$n$-player, turn-based, finite, deterministic, full information games

rtGDL-II (2015)
$n$-player, turn-based, finite, deterministic, full information games
What is real-time in games?
Real-time games

Players can take actions at any moment.

Events can have duration time.
Real-time games

Players can take actions at any moment.

Events can have duration time.
Real-time games

Players can take actions at any moment.
Events can have duration time.

Jakub Kowalski, Andrzej Kisielewicz
Game Description Language for Real-time Games
Real-time games

Players can take actions at any moment.

Events can have duration time.
Real-time games

Players can take actions at any moment.

Events can have duration time.
Real-time games

Players can take actions at any moment.

Events can have duration time.
Real-time games
Real-time games

- Players can take actions at any moment.
- Events can have duration time.
Real-time in GDL

Goal
- Preserve purely declarative style
- No arithmetic in rule engine
Real-time in GDL

**Goal**

- Preserve purely declarative style
- No arithmetic in rule engine
- **Reasoners should remain unchanged**
Real-time in GDL

Goal
- Preserve purely declarative style
- No arithmetic in rule engine
- Reasoners should remain unchanged

Problem
There have to be real numbers somewhere inside the GDL code.
Real-time in GDL

Goal
- Preserve purely declarative style
- No arithmetic in rule engine
- Reasoners should remain unchanged

Problem
There have to be real numbers somewhere inside the GDL code.

But we already have natural numbers (in goal relation)!
Real-time in GDL

Goal
- Preserve purely declarative style
- No arithmetic in rule engine
- Reasoners should remain unchanged

Problem
There have to be real numbers somewhere inside the GDL code.

But we already have natural numbers (in goal relation)!

Solution
- Treat time as immutable values, only copying is allowed.
- Arithmetic operations are allowed only by the search engine (between state updates).
REAL-TIME GAMES
DESCRIPTION LANGUAGE
(rtGDL)
# Keywords

<table>
<thead>
<tr>
<th>GDL</th>
<th>rtGDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>role(R)</td>
<td></td>
</tr>
<tr>
<td>init(F)</td>
<td></td>
</tr>
<tr>
<td>true(F)</td>
<td></td>
</tr>
<tr>
<td>legal(R,M)</td>
<td></td>
</tr>
<tr>
<td>does(R,M)</td>
<td></td>
</tr>
<tr>
<td>next(F)</td>
<td></td>
</tr>
<tr>
<td>terminal</td>
<td></td>
</tr>
<tr>
<td>goal(R,N)</td>
<td></td>
</tr>
</tbody>
</table>

\[
T \in \mathbb{R}^+ \cup \{\infty\} \quad \text{(lifetime)}
\]
## Keywords

<table>
<thead>
<tr>
<th>GDL</th>
<th>rtGDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>role(R)</td>
<td>role(R)</td>
</tr>
<tr>
<td>init(F)</td>
<td>init(T,F)</td>
</tr>
<tr>
<td>true(F)</td>
<td>true(T,F)</td>
</tr>
<tr>
<td>legal(R,M)</td>
<td>legal(R,M)</td>
</tr>
<tr>
<td>does(R,M)</td>
<td>does(R,M)</td>
</tr>
<tr>
<td>next(F)</td>
<td>next(T,F)</td>
</tr>
<tr>
<td>terminal</td>
<td>terminal</td>
</tr>
<tr>
<td>goal(R,N)</td>
<td>goal(R,N)</td>
</tr>
<tr>
<td></td>
<td>infinity</td>
</tr>
<tr>
<td></td>
<td>expired(F)</td>
</tr>
</tbody>
</table>

\[ T \in \mathbb{R}^+ \cup \{\infty\} \]  
(lifetime)
Sketch of semantics

**Event**

An *event* occurs when at least one fact becomes obsolete or at least one player performs a move.

**Time update**

Update of state $S$ after time $\Delta t$:

- **Holding facts**:
  \[
  \{ \text{true}(t_i - \Delta t, f_i) : \text{true}(t_i, f_i) \in S \land t_i > \Delta t \}\]

- **Expired facts**:
  \[
  \{ \text{expired}(f_i) : \text{true}(t_i, f_i) \in S \land t_i \leq \Delta t \}\]

**State update**

State update is performed with the first occurrence of an event.
Simple example

Self appearing and disappearing object

1. `init(1.0, appear(cheshireCat))`
2. `next(3.0, disappear(C)) ⇐ expired(appear(C)).`
3. `next(T, disappear(C)) ⇐ true(T, disappear(C)).`
4. `next(1.0, appear(C)) ⇐ expired(disappear(C)).`
5. `next(T, appear(C)) ⇐ true(T, appear(C)).`
Communication Protocol

Game Manager

Players

(START id ...)

P1
P2

(PLAY id nil $t_0$)

P1
P2

expired($f_1$)

$P_1$
$P_2$

$P_1$
$P_2$

(PLAY id ($m_1$ nil) $t_2$)

expired($f_2$)

$P_1$
$P_2$
Execution model

Game Manager

1. Send START messages.
   Set $S := S_0$. Turn the timer on.
2. Send initial PLAY messages.
3. Restart the timer.
   Calculate $t_u = \text{the minimal time until something from } S \text{ expires}$.
4. Wait until move received or the timer equal to $t_u$.
   Set $M := \text{‘players moves’}, \Delta t := \text{‘timer indications’}$.
5. If $M \neq \emptyset$ send PLAY$(M, \Delta t)$.
6. Update state $S := u(\Delta t, M, S)$.
7. If $S$ is not terminal go to 3.
   Otherwise, send STOP.

Clocks

STARTCLOCK – unchanged
PLAYCLOCK – is a multiplier for every $T$
Examples
Turn-based games

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)).

...
Chess clock

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

5. `next(infinity, timer(R,T))`  
   `⇐ does(R,M)`  
   `∧ true(T, clock).`
6. `next(infinity, timer(S,T))`  
   `⇐ true(infinity, timer(S,T))`  
   `∧ does(R,M)`  
   `∧ distinct(R,S).`
7. `next(T, clock)`  
   `⇐ true(infinity, timer(S,T))`  
   `∧ does(R,M)`  
   `∧ distinct(R,S).`
8. `terminal ⇐ expired(clock).`
9. `goal(R, 0)`  
   `⇐ expired(clock)`  
   `∧ true(infinity, control(R)).`
Respawning objects

1 \texttt{init}(\textit{infinity}, \texttt{onMap}(2,3, yellowArmor, 5.5))
2 \texttt{init}(\textit{infinity}, \texttt{onMap}(11,11, redArmor, 10.0))
3
4 \texttt{next}(T, \texttt{toRespawn}(X,Y,A,T))
5 \quad \leftarrow \texttt{true}(\textit{infinity}, \texttt{onMap}(X,Y,A,T))
6 \quad \wedge \texttt{playerAtPosition}(X,Y).
7 \texttt{next}(T, \texttt{toRespawn}(X,Y,A,S))
8 \quad \leftarrow \texttt{true}(T, \texttt{toRespawn}(X,Y,A,S)).
9
10 \texttt{next}(\textit{infinity}, \texttt{onMap}(X,Y,A,T))
11 \quad \leftarrow \texttt{true}(\textit{infinity}, \texttt{onMap}(X,Y,A,T))
12 \quad \wedge \neg \texttt{playerAtPosition}(X,Y).
13 \texttt{next}(\textit{infinity}, \texttt{onMap}(X,Y,A,T))
14 \quad \leftarrow \texttt{expired}(\texttt{toRespawn}(X,Y,A,T))
Self-moving objects

1. `speed(xwing, 2, 2, 1)`
2. `init(infinity, space(35, 86, 40, xwing))`
3. `init(0.5, refresh)`
4. `next(0.5, refresh) ⇐ expired(refresh)`
5. `next(T, refresh) ⇐ true(T, refresh)`
6. `next(infinity, space(F, G, H, S))`  
   ⇐ `true(infinity, space(X, Y, Z, S))`  
   ∧ `expired(refresh)`  
   ∧ `speed(S, A, B, C)`  
   ∧ `plus(X, A, F)`  
   ∧ `plus(Y, B, G)`  
   ∧ `plus(Z, C, H)`.
7. `next(infinity, space(X, Y, Z, S))`  
   ⇐ `true(infinity, space(X, Y, Z, S))`  
   ∧ ¬ `expired(refresh)`.
Player ordering time-taking vents

1  cost(\textit{barracks},160,6.0).
2  cost(\textit{blacksmith},140,7.0).
3  \textit{init}(\textit{infinity},\textit{gold}(500)).
4
5  \textit{legal}(\textit{player},\textit{build}(B))
6    \leftarrow \textit{true}(\textit{infinity},\textit{gold}(G))
7        \land \textit{cost}(B,F,T)
8        \land \textit{greaterEqual}(G,F).
9
10 \textit{next}(T,\textit{underConstruction}(B))
11     \leftarrow \textit{does}(\textit{player},\textit{build}(B))
12      \land \textit{cost}(B,G,T).
13 \textit{next}(T,\textit{underConstruction}(B))
14      \leftarrow \textit{true}(T,\textit{underConstruction}(B)).
15
16 \textit{next}(\textit{infinity},\textit{constructed}(B))
17     \leftarrow \textit{expired}(\textit{underConstruction}(B)).
18 \textit{next}(\textit{infinity},\textit{constructed}(B))
19      \leftarrow \textit{true}(\textit{infinity},\textit{constructed}(B)).
Real-time GDL with Incomplete Information (rtGDL-II)
Nondeterminism

What happens? (following GDL-II)

Using a non-empty set of random’s actions: perform an action drawn with uniform probability and update the state.
Nondeterminism

What happens? (following GDL-II)
Using a non-empty set of random’s actions:
perform an action drawn with uniform probability and update the state.

When it happens?
If random’s action arity > 1 and the first argument is a real number:
perform state update after time $t'$ drawn using the $\mathcal{U}(0, t)$ distribution.

Example
- wait,
- move(right),
- shot(0,pistol),
- shot(2,blaster).
Incomplete Information

Basics (following GDL-II)

- sees as a predicate for players’ percepts.
- Game Manager sends percepts instead of the joint move.

We should not give a clue that someone made a move. Solution

Compute two sets of percepts:
- after state update,
- as if update never happened.
Send message only if the sets differ.

Leads to silent updates problem (which can be solved by making them verbose).
Incomplete Information

Basics (following GDL-II)

- *sees* as a predicate for players’ percepts.
- Game Manager sends percepts instead of the joint move.
- We should not give a clue that someone made a move.
## Incomplete Information

### Basics (following GDL-II)
- `sees` as a predicate for players’ percepts.
- Game Manager sends percepts instead of the joint move.
- We should not give a clue that someone made a move.

### Solution
Compute two sets of percepts:
- after state update,
- as if update never happened.
Send message only if the sets differ.

Leads to silent updates problem
(which can be solved by making them verbose)
SUMMARY AND FUTURE WORK
We have presented a real-time extension of GDL, which:
- remains in the spirit of GGP,
- preserve pure logical reasoning,
- is the largest GDL family language so far,
- opens a new area of GGP research,
- can be further extended by merging with GDL-II.
Future work

- Investigate limitations of the language and provide more examples.
- Investigate relations with other formalisms (e.g., Extensive form games).
- Implement the Game Manager.
- Implement a reasonable player.
- Formally describe rtGDL-II.
Future work

- Investigate limitations of the language and provide more examples.
- Investigate relations with other formalisms (e.g., Extensive form games).
- Implement the Game Manager.
- Implement a reasonable player.
- Formally describe rtGDL-II.

If anyone is interested, I am very willing to cooperate 😊.
Future work

- Investigate limitations of the language and provide more examples.
- Investigate relations with other formalisms (e.g., Extensive form games).
- Implement the Game Manager.
- Implement a reasonable player.
- Formally describe rtGDL-II.

If anyone is interested, I am very willing to cooperate 😊.

THANK YOU