Logic Programming and Knowledge Representation in Computer Games

Jozef Šiška  Michal Turček  Milutín Krištofič

KAI FMFI UK, Mlynská dolina, 842 48 Bratislava

Znalosti 2008
Outline

1 Motivation
2 Logic Programming
3 Computer Games
4 Logic Programming in Computer Games
Motivation

- Dynamic Logic Programming
- Evaluate DLP in a simple application area
  - world of a computer game
- “Useful” application
Logic Programming

- Atoms: $P, Q, \text{on}(a, b)$
- Literals: $P, \neg P$
- Logic program – set of rules
  
  $$L \leftarrow L_1, L_2, \ldots, L_n$$

Dynamic Logic Program

- Sequence of LPs
- Newer/more important knowledge overrides older

Multidimensional Dynamic Logic Programs

- Non-linear, directed acyclic graph, partial ordering
Semantics of Logic Programs

- Definite LPs
- Normal LPs
- Generalized LPs

Example

\[
P_1 = \{ a \leftarrow ; \quad b \leftarrow a ; \quad c \leftarrow d, a \}\n\]
\[
P_2 = \{ d \leftarrow \text{not } c ; \quad c \leftarrow \text{not } d \}\n\]
\[
P_3 = \{ \text{not } b \leftarrow a \}\n\]
Semantics of Logic Programs

- Definite LPs
- Normal LPs
- Generalized LPs

Example

\[
\begin{align*}
P_1 &= \{ a \leftarrow ; \ b \leftarrow a ; \ c \leftarrow d, a \} \\
P_2 &= \{ d \leftarrow \text{not} \ c ; \ c \leftarrow \text{not} \ d \} \\
P_3 &= \{ \text{not} \ b \leftarrow a \}
\end{align*}
\]

- Minimal models
- \( \text{least}(P_1) = \{ a, b \} \)
Semantics of Logic Programs

- Definite LPs
- Normal LPs
- Generalized LPs

**Example**

\[
P_1 = \{ a \leftarrow ; \ b \leftarrow a ; \ c \leftarrow d, a \} \]

\[
P_2 = \{ d \leftarrow \text{not} \ c \ ; \ c \leftarrow \text{not} \ d \} \]

\[
P_3 = \{ \text{not} \ b \leftarrow a \} \]

- \( M \) is a **stable model of** \( P \) iff \( M = \text{least}(P^M) \)
- \( P^M \) - remove non-modelled rules, then default literals.
- \( P_2^{\{c\}} = \{ c \leftarrow \} \quad \text{Sem}(P_2) = \{ \{c\}, \{d\} \} \)
Semantics of Logic Programs

- Definite LPs
- Normal LPs
- Generalized LPs

Example

\[ P_1 = \{ a \leftarrow ; \quad b \leftarrow a ; \quad c \leftarrow d, a \} \]
\[ P_2 = \{ d \leftarrow \text{not} \ c ; \quad c \leftarrow \text{not} \ d \} \]
\[ P_3 = \{ \text{not} \ b \leftarrow a \} \]

- \( M \) is a stable model of \( P \) iff \( M' = \text{least}((P \cup M^-)') \)
- \( X' = \text{replace not } A \text{ with not}_\text{A} \text{ in } X \)
- \( M^- = M \cap \{\text{not } A | A \text{ is an atom}\} \)
Semantics of Dynamic LPs

Example

\[ P_1 = \{ a \leftarrow \} \quad P_2 = \{ \text{not } a \leftarrow \} \quad P_3 = \{ a \leftarrow a \} \]

Definition

\( M \) is a stable model of \( \mathcal{P} \) iff

\[ M = \text{least}(\text{Res}(\mathcal{P}, M) \cup \text{Def}(\mathcal{P}, M)) \]
Semantics of Dynamic LPs

Example

\[ P_1 = \{ a \leftarrow \} \quad P_2 = \{ \text{not } a \leftarrow \} \quad P_3 = \{ a \leftarrow a \} \]

Definition

\[ M \text{ is a stable model of } \mathcal{P} \text{ iff } \]

\[ M = \text{least}(\text{Res}(\mathcal{P}, M) \cup \text{Def}(\mathcal{P}, M)) \]

Rejected rules:

- cannot reject
- can reject
Semantics of Dynamic LPs

Example

\[ P_1 = \{ a \leftarrow \} \quad P_2 = \{ \text{not } a \leftarrow \} \quad P_3 = \{ a \leftarrow a \} \]

Definition

\[ M \text{ is a stable model of } \mathcal{P} \text{ iff } \]

\[ M = \text{least}( \text{Res}(\mathcal{P}, M) \cup \text{Def}(\mathcal{P}, M)) \]

Rejected rules:

- cannot reject
- can reject
Semantics of Dynamic LPs

Example

\[ P_1 = \{ a \leftarrow \}; \quad \text{not } a \leftarrow \} \quad P_3 = \{ a \leftarrow a \} \]

Definition

\[ M \text{ is a stable model of } P \text{ iff } \]

\[ M = \text{least}(\text{Res}(P, M) \cup \text{Def}(P, M)) \]

Rejected rules: cannot reject \hspace{1cm} can reject

Incomparable: cannot \hspace{1cm} can
Semantics of Dynamic LPs

Example

\[ P_1 = \{ a \leftarrow ; \text{not } a \leftarrow \} \quad P_3 = \{ a \leftarrow a \} \]

Definition

\[ M \text{ is a stable model of } P \text{ iff } \]

\[ M = \text{least}(\text{Res}(P, M) \cup \text{Def}(P, M)) \]

Rejected rules:

- cannot reject
- can reject

Incomparable:

- cannot
- can
Semantics of Dynamic LPs

Example

\[ P_1 = \{ a \leftarrow ; \text{not } a \leftarrow \} \quad P_3 = \{ a \leftarrow a \} \]

Definition

\[ M \text{ is a stable model of } \mathcal{P} \text{ iff} \]

\[ M = \text{least}(\text{Res}(\mathcal{P}, M) \cup \text{Def}(\mathcal{P}, M)) \]

Rejected rules:

- cannot reject
- can reject

Incomparable:

- cannot
- can

\[ \text{Def}(\mathcal{P}, M) : \quad M^- \quad \text{\{ not } A \mid A \text{ is not supported in } \mathcal{P} \text{ wrt } M \} \]
World of a Computer Game as a Multiagent System

World
- Background knowledge (world mechanics)
- Environment (state of the world)
- Relatively small
- Exactly specified

Agents
- Player character
- NPCs (computer controlled characters)
- Active objects

Actions
- Highly specific
- Exact conditions and consequences
- Relatively small number of allowed actions
Game Engines

Game engines
- user interface
- world representation
- scripting, dialogues

Game AI
- part of the game responsible for the opponents / objects
- understaffed
- game cool factor: graphics physics AI

Story or environment interaction oriented games
- RPG, adventure
- More different ways to finish a quest → better the game
- All possible ways to achieve a quest have to be scripted
- "in-place" inference in scripts
LP in Computer Games

- World State Evaluation
- Game description in LP
- Modularization
- Planning, ...
Usual AI Research in Games

Game Engine
Network / IPC / Module interface . . .

Player 1
Keyboard input, . . .

Player 2
Experimental planning application

Player 3
QSModels
World State Evaluation

(M)DynLoP → DLP<→>LP transformation → ASP<→>Python interface → ASP-based world state evaluation → Game Scripting, World Representation

Game Engine
Quest Evaluation – Example

\[ P_0 = \{ \text{alive(King)}; \text{alive(General)}; \text{speak(King)}; \text{speak(General)}; \} \]

\[ P_1 = \{ \text{killed(Player, King)}; \} \]

\[ P_g = \{ \text{not alive(X)} \leftarrow \text{killed(Y, X)}; \quad \text{alive(X)} \leftarrow \text{ressurect(Y, X)}; \quad \text{not speak(X)} \leftarrow \text{spell(Silence, X)}; \ldots \} \]

\[ P_q = \{ \text{war} \leftarrow \text{alive(King)}, \text{influenced(King, X)}; \quad \text{influenced(King, X)} \leftarrow \text{alive(X)}, \text{wants_war(X)}; \quad \text{wants_war(General)}; \} \]
Quest Evaluation – Example

\[
P_0 = \{ \text{alive(King)}; \text{alive(General)}; \text{speak(King)}; \text{speak(General)}; \}
\]

\[
P_1 = \{ \text{killed(Player, King)}; \}
\]

\[
P_g = \{ \text{not alive(X)} \leftarrow \text{killed(Y, X)}; \quad \text{alive(X)} \leftarrow \text{ressurect(Y, X)}; \\
\quad \text{not speak(X)} \leftarrow \text{spell(Silence, X)}; \ldots \}
\]

\[
P_q = \{ \text{war} \leftarrow \text{alive(King)}; \text{influenced(King, X)}; \\
\quad \text{influenced(King, X)} \leftarrow \text{alive(X)}; \text{wants\_war(X)}; \text{speak(X)}; \\
\quad \text{wants\_war(General)}; \}
\]
**Quest Evaluation – Example**

\[P_0 = \{ \text{alive}(\text{King}); \text{alive}(\text{General}); \text{speak}(\text{King}); \text{speak}(\text{General}); \}\}\]

\[P_1 = \{ \text{cast\_spell}(\text{Player}, \text{Silence}, \text{General}); \}\}\]

\[P_g = \{ \text{not alive}(X) \leftarrow \text{killed}(Y, X); \quad \text{alive}(X) \leftarrow \text{ressurect}(Y, X); \}
  \begin{align*}
  &\quad \text{not speak}(X) \leftarrow \text{spell}(\text{Silence}, X); \\
  &\quad \ldots
  \end{align*}\]

\[P_q = \{ \text{war} \leftarrow \text{alive}(\text{King}), \text{influenced}(\text{King}, X); \quad \text{influenced}(\text{King}, X) \leftarrow \text{alive}(X), \text{wants\_war}(X), \text{speak}(X); \}
  \begin{align*}
  &\quad \text{wants\_war}(\text{General});
  \end{align*}\]
EDDOM – LP Based Game

- world_state_i
- possible_actions_i

User Interface
- selected_action_i

Game Processing
- world_state_{i+1}
- possible_actions_{i+1}
Modularization

- Modules that represent different locations in game
- "dynamic", "on-demand" loading
- Module = activator program + module program
Modularization cont.

Activator Program

ASP

load_module("...")

Programs from selected modules

ASP
**Dynamic LP and Modularization**

**eddom.sm**

```prolog
possible(moveto(X)) :-
    holds(at(Y)),
    accessible(X,Y),
    not ab_possible(moveto(X)).

ab_possible(moveto(X)) :-
    holds(at(home)),
    not holds(have(bag)).
```

**moving.dlpm**

```prolog
40:
possible(moveto(X)) :-
    holds(at(Y)),
    accessible(X,Y).
```

**quest_leave.dlpm**

```prolog
activator:
load_me :- at(home).
module:
80:
-possible(moveto(X)) :-
    holds(at(home)),
    not holds(have(bag)).
```
Future

- Planning
  - Actions of computer controlled characters
- Transformation of game world data to LP
- Interaction with NPCs
- Fuzzy logics?
  - Attitude of NPCs against the player...
Summary

- Logic Programming, Dynamic Logic Programming
- Computer Games
- Enhancing Games
  - Game World State Evaluation
  - Representation of Game World data – EDDOM
  - Modularization
Thank You...

Game Over