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

Motivation

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

Logic Programming

- Atoms: $P, Q, on(a, b)$ Literals: $P, \text{not } P, \neg P$
- 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

$$
\begin{aligned}
P_1 &= \{a \leftarrow ; \quad b \leftarrow a ; \quad c \leftarrow d, a\} \\
P_2 &= \{d \leftarrow \text{not } c ; \quad c \leftarrow \text{not } d\} \\
P_3 &= \{\text{not } b \leftarrow a\}
\end{aligned}
$$

Semantics of Logic Programs

- Definite LPs
- Normal LPs
- Generalized LPs

Example

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

- Minimal models
- $least(P_1) = \{a, b\}$

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 $M = least(P^M)$
- P^M - remove non-modelled rules, then default literals.
- $P_2^{\{c\}} = \{c \leftarrow\}$ $\qquad Sem(P_2) = \{\{c\}, \{d\}\}$

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 $M' = least((P \cup M^-)')$
- X' = replace $\text{not } A$ with not_A in X
- $M^- = M \cap \{\text{not } A | A \text{ is an atom}\}$

Semantics of Dynamic LPs

Example

$$P_1 = \{\mathbf{a} \leftarrow\} \qquad P_2 = \{\text{not } a \leftarrow\} \qquad P_3 = \{a \leftarrow a\}$$

Definition

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

$$M = least(Res(\mathcal{P}, M) \cup Def(\mathcal{P}, M))$$

Semantics of Dynamic LPs

Example

$$P_1 = \{\mathbf{a} \leftarrow\ \} \qquad P_2 = \{\text{not } a \leftarrow\} \qquad P_3 = \{a \leftarrow a\}$$

Definition

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

$$M = least(Res(\mathcal{P}, M) \cup Def(\mathcal{P}, M))$$

Rejected rules: cannot reject can reject

Semantics of Dynamic LPs

Example

$$P_1 = \{\mathbf{a} \leftarrow \} \qquad P_2 = \{\text{not } a \leftarrow\} \qquad P_3 = \{a \leftarrow a\}$$

Definition

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

$$M = least(Res(\mathcal{P}, M) \cup Def(\mathcal{P}, M))$$

Rejected rules: cannot reject can reject

Semantics of Dynamic LPs

Example

$P_1 = \{\mathbf{a} \leftarrow \quad ; \quad \text{not } a \leftarrow\} \qquad P_3 = \{a \leftarrow a\}$

Definition

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

$$M = least(Res(\mathcal{P}, M) \cup Def(\mathcal{P}, M))$$

Rejected rules:	cannot reject	can reject
Incomparable	cannot	can

Semantics of Dynamic LPs

Example

$P_1 = \{\mathbf{a} \leftarrow \quad ; \quad \text{not } a \leftarrow\}$ $\quad P_3 = \{a \leftarrow a\}$

Definition

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

$$M = least(Res(\mathcal{P}, M) \cup Def(\mathcal{P}, M))$$

Rejected rules:	cannot reject	can reject
Incomparable	cannot	can

Semantics of Dynamic LPs

Example

$P_1 = \{\mathbf{a} \leftarrow \quad ; \quad \text{not } a \leftarrow\} \qquad P_3 = \{a \leftarrow a\}$

Definition

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

$$M = least(Res(\mathcal{P}, M) \cup Def(\mathcal{P}, M))$$

Rejected rules:	cannot reject	can reject
Incomparable	cannot	can
$Def(\mathcal{P}, M)$:	M^-	$\left\{ \text{not } A \;\middle|\; \begin{array}{l} A \text{ is not supported} \\ \text{in } \mathcal{P} \text{ wrt } M \end{array} \right\}$

World of a Computer Game as a Multiagent System

World

- Background knowledge (world mechanics)
- Environment (state of the world)
- Relatively mall
- 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 $\Longrightarrow$ 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 . . .

Player1

Keyboard input, . . .

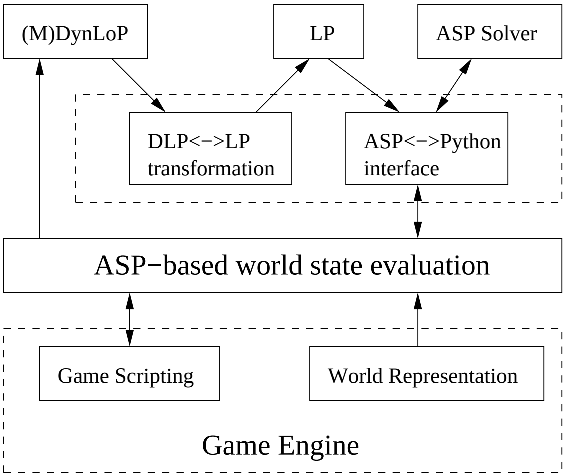
Player2

Experimental planning application

Player3

QSModels

World State Evaluation



Quest Evaluation – Example

$$P_0 = \{\ alive(King);\ alive(General);\ speak(King);\ speak(General);\ \}$$

$$P_1 = \{\ killed(Player, King);\ \}$$

$$P_g = \left\{ \begin{array}{c} \text{not } alive(X) \leftarrow killed(Y, X); \quad alive(X) \leftarrow ressurect(Y, X); \\ \text{not } speak(X) \leftarrow spell(Silence, X); \ldots \end{array} \right\}$$

$$P_q = \left\{ \begin{array}{c} war \leftarrow alive(King), influenced(King, X); \\ influenced(King, X) \leftarrow alive(X), wants\_war(X); \\ wants\_war(General); \end{array} \right\}$$

Quest Evaluation – Example

$$P_0 = \{ \; alive(King); \; alive(General); \; speak(King); \; speak(General); \}$$

$$P_1 = \{ \; killed(Player, King); \; \}$$

$$P_g = \left\{ \begin{array}{c} \text{not } alive(X) \leftarrow killed(Y, X); \quad alive(X) \leftarrow ressurect(Y, X); \\ \text{not } speak(X) \leftarrow spell(Silence, X); \ldots \end{array} \right\}$$

$$P_q = \left\{ \begin{array}{c} war \leftarrow alive(King), influenced(King, X); \\ influenced(King, X) \leftarrow alive(X), wants\_war(X), speak(X); \\ wants\_war(General); \end{array} \right\}$$

Quest Evaluation – Example

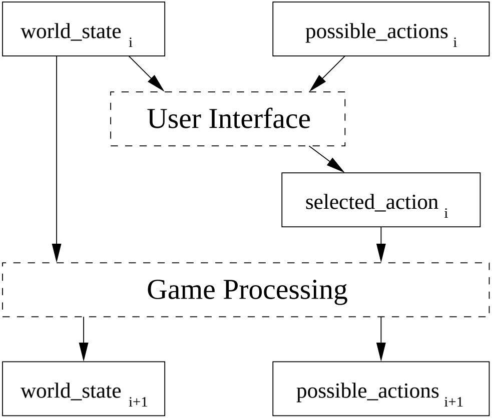
$$P_0 = \{\ alive(King);\ alive(General);\ speak(King);\ speak(General);\ \}$$

$$P_1 = \{\ cast\_spell(Player, Silence, General);\ \}$$

$$P_g = \left\{ \begin{array}{c} \text{not } alive(X) \leftarrow killed(Y, X); \quad alive(X) \leftarrow ressurect(Y, X); \\ \text{not } speak(X) \leftarrow spell(Silence, X); \ldots \end{array} \right\}$$

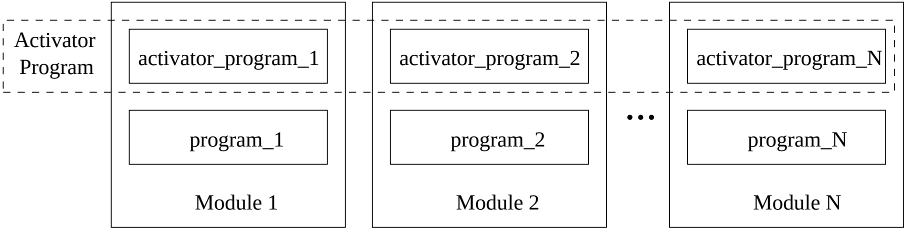
$$P_q = \left\{ \begin{array}{c} war \leftarrow alive(King), influenced(King, X); \\ influenced(King, X) \leftarrow alive(X), wants\_war(X), speak(X); \\ wants\_war(General); \end{array} \right\}$$

EDDOM – LP Based Game

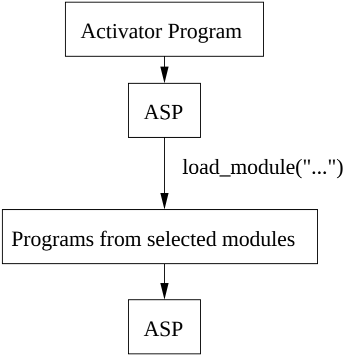


Modularization

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



Modularization cont.



Dynamic LP and Modularization

eddom.sm

```
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

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

quest_leave.dlpm

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

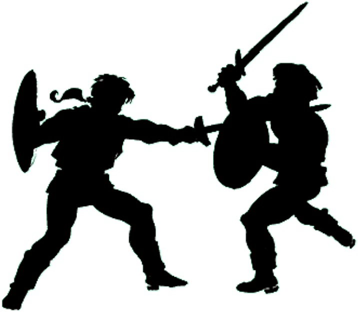
Future

- 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