UNIVERSITÀ DEGLI STUDI DI NAPOLI

FEDERICO II

PROGETTO INTELLIGENZA ARTIFICIALE:

Sviluppo di una base di conoscenza in SWI-Prolog.

Prof.ssa:
AMATO Flora

Studenti:

COLANTUONO Antonio	***N46003371***
COPPOLA Vincenzo	***N46003356***
DELLA TORCA Salvatore	***N46003208***
IZZO Alberto	***N46003425***

INDICE

Prolog (Programming in Logic)

Il *Prolog* è un linguaggio ***logico*** (o dichiarativo), cioè è un insieme di *regole* e *fatti* che costituiscono la *Knowledge Base* (base di conoscenza).
Essendo un linguaggio dichiarativo il programmatore non specifica in alcun modo come il problema debba essere risolto.
I fatti rappresentano la verità: tutto ciò che è vero si trova nella *base di conoscenza*.
Esistono due teorie riguardanti i fatti:

- ***TEORIA DEL MONDO APERTO***: *tutto ciò che non si trova nella base di conoscenza è sconosciuto;*
- ***TEORIA DEL MONDO CHIUSO****: tutto ciò che non si trova nella base di conoscenza è falso*.

Le regole servono per determinare la verità o la falsità di un obiettivo, che viene anche indicato come *goal*.
Sono dei teoremi: se l'ipotesi è vera allora sarà vera anche la tesi.
L'ipotesi è vera se si trova nella *KB, afferita* se è stata dimostrata vera in precedenza con un'altra regola.
Il prolog è un linguaggio formale (senza ambiguità) del 1° ordine per cui i predicati fanno riferimento solo a variabili.
È basato sulle clausole di Horn e ha due diverse modalità:

- ***Editing****, in cui viene scritto il programma;*
- ***Shell****, per verificare il goal*.

Una regola è scritta nella forma TESI <- IPOTESI: le variabili che compaiono sia nella tesi che nell'ipotesi devono essere quantificate *universalmente* mentre le variabili che compaiono solamente nell'ipotesi devono essere quantificate *esistenzialmente*.
Dato un predicato che presenta almeno una variabile si definisce *SOSTITUZIONE* l'operazione che sostituisce alla variabile una costante.
Il predicato che si ottiene è un'*istanza* del predicato di partenza.
L'operazione più importante del Prolog è l'*UNIFICAZIONE*, il cui obiettivo è sostituire una costante ad una variabile per far si che due termini siano uguali tra loro.

- Una costante ed una variabile unificano sempre e la variabile assume il valore della costante;
- Due costanti unificano se sono uguali;
- Due variabili unificano sempre.

Gli assiomi sono forniti al programma logico da un esperto del dominio d'interesse e ad essi possono essere applicate delle regole, dette *regole di inferenza*, il cui risultato è generare nuovi fatti o teoremi.
Quando viene generato un teorema questo viene incluso nella base della conoscenza.
Il programma logico serve per dimostrare la verità o la falsità di un teorema, solitamente non presente tra gli assiomi iniziali.
Il motore inferenziale del Prolog tenta inizialmente di verificare il goal con uno degli assiomi della KB e, se non ci riesce, applica la prima regola di inferenza che incontra nella KB al goal.
A questo punto il goal è stato modificato e si può tentare di verificare se tale nuovo goal verifichi uno degli assiomi della KB.
Se non è possibile verificare il goal, si può ancora una volta applicare ad esso una regola della KB e ritentare la verifica: se il goal viene raggiunto, il procedimento di sostituzione termina, con l'asserzione della verità del goal, altrimenti, a valle di tutte le possibili applicazioni di tutte le regole si ha la falsità del goal.
Un sistema di questo tipo è detto ***sistema goal driven***.

Descrizione progetto

N.B: Le informazioni inserite nella base di conoscenza fanno riferimento al gioco *Monster Hunter*.

La base di conoscenza è stata realizzata in un documento di testo salvato con estensione *.pl* ed è stata caricata in SWI-Prolog utilizzando il comando

consult(contest.pl)

I *fatti* inseriti nella base di conoscenza hanno come obiettivo definire dei *mostri,* ognuno caratterizzato da un *tipo* e da un *livello di difficoltà*:

mostro(rathian, fuoco, 1).
mostro(rathalos, fuoco, 2).
mostro(teostra, fuoco, 3).
mostro(daimyo, acqua, 1).
mostro(plesioth, acqua, 2).
mostro(kirin, tuono, 2).
mostro(rajang, tuono, 3).
mostro(zamtrios, ghiaccio, 1).
mostro(kushala, ghiaccio, 3).

Per ogni mostro è stata inserita una debolezza:

debolezza(rathian, acqua).
debolezza(rathalos, acqua).
debolezza(teostra, acqua).
debolezza(daimyo, tuono).
debolezza(plesioth, tuono).
debolezza(kirin, terra).
debolezza(rajang, terra).
debolezza(zamtrios, fuoco).
debolezza(kushala, fuoco).

Sono state poi inserite le tipologie di armi disponibili per l'uccisione di un mostro:

- *arma* indica una generica arma, mentre *spada* e *arco* indicano l'arma specifica;

```
arma(fuoco).
arma(tuono).
arma(ghiaccio).
arma(acqua).

spada(fuoco).
spada(tuono).
spada(ghiaccio).
spada(acqua).

arco(fuoco).
arco(tuono).
arco(ghiaccio).
arco(acqua).
```

Per usare il comando ***not*** è stato inserito anche un predicato che indica se il mostro inserito come parametro è volante:

```
volante(teostra).
volante(rathalos).
volante(kushala).
```

Successivamente è stato realizzato un predicato che, sfruttando la ricorsione, ci consente, data una *lista* contenente le armi acquistabili, di verificare che un'arma *X* appartenga alla lista.

```
acquistabile(X,[X|_C]).
acquistabile(X,[_T|C]):- acquistabile(X,C).
```

Osserviamo che il parametro C è preceduto da un underscore. Questo perché si tratta di una *variabile singleton*, cioè una variabile che non viene realmente istanziata e che occorre soltanto in un predicato di una clausola: assume un valore dopodiché tale valore viene buttato via.

Se l'arma X è acquistabile, cioè la risposta alla query è *true,* allora è possibile aggiungere dinamicamente l'arma alla base di conoscenza. Affinchè ciò sia possibile è stato necessario aggiungere in cima alla base di conoscenza il predicato *:-dynamic()* che necessita come argomento dell'assioma che si vuole inserire a tempo di esecuzione:

```
:- dynamic (arma/1)
:- dynamic (spada/1).
:- dynamic (arco/1).
```

In questo modo è possibile anche sconfiggere i mostri che hanno come debolezza il tipo *terra*.
Nella base di conoscenza non sono infatti presenti armi di tipo *terra* e quindi, sfruttando il comando *assert,* è possibile, a tempo di esecuzione, caricare nella KB una spada o un arco di tipo terra:

```
assert(spada(terra)).
assert(arco(terra)).
```

La regola successiva sfrutta l'algoritmo di Dijkstra (*commesso viaggiatore)* per calcolare quanto sia difficile uccidere un mostro:

```
difficolta(X,Y,C):- arma(Y), debolezza(X,Y), mostro(X,_Z,C).

difficolta(X,Y,C):- arma(Y), debolezza(Z,Y), mostro(Z,W,C2),
                    difficolta(X,W,C3), C is C3 + C2.
```

X indica il mostro che si vuole uccidere con l'arma Y, mentre C è il costo.
Il passo base viene usato nel caso in cui il tipo dell'arma Y coincide con la debolezza del mostro X.
Se però l'arma Y è di un tipo diverso rispetto alla debolezza del mostro X, si utilizza il passo ricorsivo: il giocatore può uccidere un mostro debole al tipo di arma Y che egli possiede e, all'atto dell'uccisione, il mostro fornirà l'arma del suo tipo al giocatore.
La ricorsione si ripete fin quando il giocatore non riceverà l'arma col quale può uccidere il mostro X.
A questo punto saranno sommate tutte le difficoltà per determinare la difficoltà complessiva di uccisione del mostro.

Infine sono state inserite due regole che determinano se è possibile uccidere un mostro con una spada o un arco di un certo tipo:

uccisioneConSpada(X,Y):- not(volante(X)), spada(Y),
debolezza(X,Y).

uccisioneConSpada(X,Y):- not(volante(X)), spada(Y),
debolezza(Z,Y), mostro(Z,W,_D),
uccisioneConSpada(X,W).

uccisioneConArco(X,Y):- arco(Y), debolezza(X,Y).

uccisioneConArco(X,Y):- arco(Y), debolezza(Z,Y), mostro(Z,W,_D),
uccisioneConArco(X,W).

Anche in questo caso è stata utilizzata la ricorsione per far si che un giocatore possa uccidere altri mostri fin quando non riceverà l'arma con la quale può uccidere il mostro X.
In particolare, nel caso della regola *uccisioneConSpada,* è stato utilizzato il comando *not* per verificare che il mostro che si voglia uccidere con la spada non sia un mostro *volante*.
-PASSO BASE:
Il mostro X è ucciso con la spada di tipo Y se e solo se il mostro X non è volante, è disponibile la spada di tipo Y e la debolezza del mostro è l'elemento Y.
-PASSO RICORSIVO:
Il mostro X è ucciso con la spada di tipo Y se e solo se il mostro X non è volante, è disponibile la spada di tipo Y ed esiste un mostro Z di tipo W, la cui debolezza è l'elemento Y, tale che uccidendo questo mostro si ottenga l'arma di tipo W che verrà utilizzata nella chiamata ricorsiva.

N.B: *il discorso è analogo nel caso del predicato uccisioneConArco. L'unica differenza è che in tal caso non è necessario verificare che il mostro non sia volante.*

Knowledge Base

```
:- dynamic (arma/1).
:- dynamic (spada/1).
:- dynamic (arco/1).

mostro(rathian, fuoco, 1).
mostro(rathalos, fuoco, 2).
mostro(teostra, fuoco, 3).
mostro(daimyo, acqua, 1).
mostro(plesioth, acqua, 2).
mostro(kirin, tuono, 2).
mostro(rajang, tuono, 3).
mostro(zamtrios, ghiaccio, 1).
mostro(kushala, ghiaccio, 3).

volante(teostra).
volante(rathalos).
volante(kushala).

debolezza(rathian, acqua).
debolezza(rathalos, acqua).
debolezza(teostra, acqua).
debolezza(daimyo, tuono).
debolezza(plesioth, tuono).
debolezza(kirin, terra).
debolezza(rajang, terra).
debolezza(zamtrios, fuoco).
debolezza(kushala, fuoco).

arma(fuoco).
arma(tuono).
arma(ghiaccio).
arma(acqua).

spada(fuoco).
spada(tuono).
spada(ghiaccio).
spada(acqua).

arco(fuoco).
arco(tuono).
arco(ghiaccio).
```

```
arco(acqua).

acquistabile(X,[X|_C]).
acquistabile(X,[_T|C]):- acquistabile(X,C).

difficolta(X,Y,C):- arma(Y), debolezza(X,Y),
mostro(X,_Z,C).
difficolta(X,Y,C):- arma(Y), debolezza(Z,Y),
mostro(Z,W,C2), difficolta(X,W,C3), C is C3 + C2.

uccisioneConSpada(X,Y):- not(volante(X)), spada(Y),
debolezza(X,Y).
uccisioneConSpada(X,Y):- not(volante(X)), spada(Y),
debolezza(Z,Y), mostro(Z,W,_D), uccisioneConSpada(X,W).

uccisioneConArco(X,Y):- arco(Y), debolezza(X,Y).
uccisioneConArco(X,Y):- arco(Y), debolezza(Z,Y),
mostro(Z,W,_D), uccisioneConArco(X,W).
```

Esempi di Output

Per visualizzare come il motore inferenziale del prolog tenta di unificare il goal con uno degli assiomi della KB è stato utilizzato il comando *trace*.
Esistono tre fasi in cui può trovarsi il MI:

1. CALL: si tenta per la prima volta l'unificazione di un nuovo goal con una delle regole del sistema;
2. FAIL: se il goal corrente non viene unificato con la regola selezionata si ha un fail al quale segue un BACKTRACKING;
3. REDO: il MI tenta nuovamente di unificare un goal già provato almeno una volta con una regola diversa.

- ***Esiste un percorso di difficoltà 2 che permette di uccidere lo Zamtrios partendo da un'arma di elemento acqua?***

[trace] ?- difficolta(zamtrios, acqua, 2).

Call: (8) difficolta(zamtrios, acqua, 2) ? creep
Call: (9) arma(acqua) ? creep
Exit: (9) arma(acqua) ? creep
Call: (9) debolezza(zamtrios, acqua) ? creep
Fail: (9) debolezza(zamtrios, acqua) ? creep
Redo: (8) difficolta(zamtrios, acqua, 2) ? creep
Call: (9) arma(acqua) ? creep
Exit: (9) arma(acqua) ? creep
Call: (9) debolezza(_1932, acqua) ? creep
Exit: (9) debolezza(rathian, acqua) ? creep
Call: (9) mostro(rathian, _1934, _1936) ? creep
Exit: (9) mostro(rathian, fuoco, 1) ? creep
Call: (9) difficolta(zamtrios, fuoco, _1936) ? creep
Call: (10) arma(fuoco) ? creep
Exit: (10) arma(fuoco) ? creep
Call: (10) debolezza(zamtrios, fuoco) ? creep
Exit: (10) debolezza(zamtrios, fuoco) ? creep
Call: (10) mostro(zamtrios, _1934, _1936) ? creep
Exit: (10) mostro(zamtrios, ghiaccio, 1) ? creep
Exit: (9) difficolta(zamtrios, fuoco, 1) ? creep
Call: (9) 2 is 1+1 ? creep
Exit: (9) 2 is 1+1 ? creep
Exit: (8) difficolta(zamtrios, acqua, 2) ? creep

true .

- ***Data la lista di armi acquistabili, è possibile acquistare una spada di fuoco?***

[trace] ?- acquistabile(spada(fuoco), [arco(fuoco), arco(ghiaccio), arco(tuono), spada(fuoco), spada(acqua)]).

Call: (8) acquistabile(spada(fuoco), [arco(fuoco), arco(ghiaccio), arco(tuono), spada(fuoco), spada(acqua)]) ? creep
Call: (9) acquistabile(spada(fuoco), [arco(ghiaccio), arco(tuono), spada(fuoco), spada(acqua)]) ? creep
Call: (10) acquistabile(spada(fuoco), [arco(tuono), spada(fuoco), spada(acqua)]) ? creep
Call: (11) acquistabile(spada(fuoco), [spada(fuoco), spada(acqua)]) ? creep
Exit: (11) acquistabile(spada(fuoco), [spada(fuoco), spada(acqua)]) ? creep
Exit: (10) acquistabile(spada(fuoco), [arco(tuono), spada(fuoco), spada(acqua)]) ? creep
Exit: (9) acquistabile(spada(fuoco), [arco(ghiaccio), arco(tuono), spada(fuoco), spada(acqua)]) ? creep
Exit: (8) acquistabile(spada(fuoco), [arco(fuoco), arco(ghiaccio), arco(tuono), spada(fuoco), spada(acqua)]) ? creep

true .

- ***Esiste un'arma di tipo arco in grado di uccidere il Kushala?***

[trace] ?- uccisioneConArco(kushala,X).

Call: (8) uccisioneConArco(kushala, _2248) ? creep
Call: (9) arco(_2248) ? creep
Exit: (9) arco(fuoco) ? creep
Call: (9) debolezza(kushala, fuoco) ? creep
Exit: (9) debolezza(kushala, fuoco) ? creep
Exit: (8) uccisioneConArco(kushala, fuoco) ? creep

X = fuoco .

- ***Esista un'arma di tipo spada in grado di uccidere il Rathalos?***

[trace] ?- uccisioneConSpada(rathalos,X).

Call: (8) uccisioneConSpada(rathalos, _1618) ? creep
Call: (9) not(volante(rathalos)) ? creep
Call: (10) volante(rathalos) ? creep
Exit: (10) volante(rathalos) ? creep
Fail: (9) not(user:volante(rathalos)) ? creep
Redo: (8) uccisioneConSpada(rathalos, _1618) ? creep
Call: (9) not(volante(rathalos)) ? creep
Call: (10) volante(rathalos) ? creep
Exit: (10) volante(rathalos) ? creep
Fail: (9) not(user:volante(rathalos)) ? creep
Fail: (8) uccisioneConSpada(rathalos, _1618) ? creep

false.

- ***È possibile uccidere lo Zamtrios con una spada fuoco?***

[trace] ?- uccisioneConSpada(zamtrios,fuoco).

Call: (8) uccisioneConSpada(zamtrios, fuoco) ? creep
Call: (9) not(volante(zamtrios)) ? creep
Call: (10) volante(zamtrios) ? creep
Fail: (10) volante(zamtrios) ? creep
Exit: (9) not(user:volante(zamtrios)) ? creep
Call: (9) spada(fuoco) ? creep
Exit: (9) spada(fuoco) ? creep
Call: (9) debolezza(zamtrios, fuoco) ? creep
Exit: (9) debolezza(zamtrios, fuoco) ? creep
Exit: (8) uccisioneConSpada(zamtrios, fuoco) ? creep

true.