

Intelligenza Artificiale

Esercitazione con Jess

Marco Piastra

Esercitazione con Jess

1. Jess
2. Fox, Goat and Cabbage (esercitazione Jess)
3. Discussione

1

Sistema Jess (*Java Expert System Shell*)

Jess - Introduzione

- Un'implementazione dell'algoritmo Rete
- Incorporata in un ambiente di esecuzione Lisp (minimale)
- (il tutto implementato in Java)

- Realizzato da
Ernest J. Friedman-Hill
Distributed Computing Systems
Sandia National Laboratories
Livermore, CA - USA
- Disponibile presso:
<http://herzberg.ca.sandia.gov/jess>
- Licenza gratuita (incluso il sorgente)
per usi non-commerciali

Jess - Funzioni Lisp

- Il sistema a regole Jess è incorporato in un ambiente LISP
- La sintassi LISP è basata sul concetto di lista (*LISt Processing*)
(`<op> <arg> *`)
 - Le liste possono essere nidificate
 - La notazione è in generale prefissa (ma con eccezioni)
 - Si usano **simboli**, senza obbligo di preventiva dichiarazione
 - I simboli che identificano variabili iniziano per ?
 - Esempi:
 - (`+ 5 4`)
 - (`bind ?var (* 2 ?pi)`) (*assegnazione di valore a ?var*)

Jess - Sintassi

- I fatti vengono espressi come tuple

```
(maschio Mario)
(femmina Paola)
(padre Mario Alba)
(padre Mario Amelia)
(madre Paola Alba)
(madre Paola Amelia)
```

- Le regole vengono espresse con una sintassi particolare
(con notazione infissa)

```
(defrule sorella
  (padre ?x ?p)
  (padre ?y ?p)
  (madre ?x ?m)
  (madre ?y ?m)
  (femmina ?x)
=>
  (assert (sorella ?x ?y)))
```

Jess – Asserire e ritrattare

- In Jess è possibile chiamare funzioni LISP nelle *condizioni* e nelle *azioni*
 - nelle condizioni è necessario usare il costrutto (test <funct>)
- Due funzioni molto utili (tipicamente per le *azioni* di una regola)
 - (**assert** <fact>)
inserimento di <fact> nella *memoria di lavoro*
(con aggiornamento della struttura Rete)
 - (**retract** <fact>)
rimozione di <fact> dalla *memoria di lavoro*
(con aggiornamento della struttura Rete)

Jess – Funzioni di attivazione

- Il sistema Jess si attiva da linea di comando

```
$ java jess.Main  
Jess>
```

- La prima linea attiva il sistema,
nella seconda compare il prompt del *Lisp Listener*

- Tipicamente le regole sono scritte su un file
ma possono anche essere inserite direttamente

```
Jess> (batch "regole.clp")
```

- Caricamento del file `regole.clp`

- Attivazione

```
Jess> (reset)  
Jess> (run)
```

- inizializzazione della memoria di lavoro
con azzeramento dei fatti
- attivazione del ciclo principale
il sistema rimane attivo finchè vi sono regole da eseguire

Jess – Funzioni utili

- Ispezione
 - Jess> (agenda)**
 - mostra le regole istanziate presenti nell'agenda
 - Jess> (facts)**
 - mostra i fatti memorizzati nella memoria di lavoro
- Tracciamento
 - Jess> (watch all)**
 - tracciamento dell'esecuzione
 - Jess> (unwatch all)**
 - elimina il tracciamento dell'esecuzione
 - Jess> (run 1)**
 - attivazione di una sola regola
- Azzeramento
 - Jess> (clear)**
 - azzerare regole e fatti

2

Fox, Goat & Cabbage (*esercitazione con Jess*)

Un dilemma

- Partecipanti:
 - un agricoltore (*farmer*)
 - una volpe (*fox*)
 - una capra (*goat*)
 - un cavolo (*cabbage*)
- Scenario:
 - una barca
 - due rive (*shore-1*, *shore-2*)
- Obiettivo
 - tutti i partecipanti sono su una riva (*shore-1*)
 - l'agricoltore deve traghettare tutti sulla riva opposta (*shore-2*)
- Vincoli:
 - se lasciate sola con la capra, la volpe mangia la capra
 - se lasciata sola con il cavolo, la capra mangia il cavolo

Funzioni Lisp particolari

- Definizione contenuto iniziale della *memoria di lavoro*
 - `(defact <fact>)`
inserimento iniziale di `<fact>` nella *memoria di lavoro*
(e aggiornamento della struttura Rete)
al momento della esecuzione di `(reset)`
- Uso di templates per strutture dati composite
 - `(deftemplate <structure>)`
 - Esempio di definizione:

```
(deftemplate status
  (slot farmer-location)
  (slot fox-location)
  (slot goat-location)
  (slot cabbage-location))
```
 - Esempio d'uso (un *fatto specifico*):

```
(status (farmer-location shore-1)
        (fox-location shore-1)
        (goat-location shore-1)
        (cabbage-location shore-1))
```

Priorità tra regole

- Priorità tra regole regole (*salience*)
 - salvo diversa indicazione, ogni regola ha *salience* 0
 - indicazione esplicita della *salience*:

```
(defrule fox-eats-goat
  (declare (salience 100))
  ...)
```

- La priorità tra regole è relativa
(il valore assoluto della *salience* non conta)
- La priorità influenza la gestione dell'*agenda*
 - nella scelta per l'attivazione
 - le regole a priorità più alta vengono privilegiate
 - rispetto alle regole a priorità più bassa

Condizioni speciali

- Nella soluzione vengono usate alcune condizioni speciali
 - Identità (già vista nell'esempio precedente)

```
(defrule farmer-with-goat-and-fox
  (farmer-location ?x)
  (fox-location ?x)
  (goat-location ?x)
  ...
```
 - Differenza

```
(defrule fox-eats-goat
  (farmer-location ?x)
  (fox-location ?y&~?x)
  (goat-location ?y)
  ...
```
 - Confronto

```
(defrule y-larger-than-x
  (value ?x)
  (value ?y&:(< ?x ?y))
  ...
```

Azioni speciali

- Nella soluzione vengono usate alcune azioni speciali
 - Assegnazione di valori a variabili
`(bind ?x (+ ?x 1))`
 - Modifica di *fatti strutturati*
`(modify ?fact
 (farmer-location ?x)
 (fox-location ?y))`
 - Duplicazione e modifica di *fatti strutturati*
`(duplicate ?fact
 (farmer-location ?x)
 (fox-location ?y))`

Java: CLASSPATH

- La variabile di sistema `CLASSPATH` definisce un *path* nel *file system* che permette alla JVM di trovare il codice delle applicazioni e delle librerie specifiche
- Impostazione della variabile `CLASSPATH`
 - meglio se in `.login`

```
CLASSPATH=<value>
export CLASSPATH
```
 - per verificare

```
echo $CLASSPATH
```
- Nel nostro caso:

```
CLASSPATH=/home/doc/mpiastrea/Jess60
```

Rispettare il maiuscolo e minuscolo! (E` Linux, non Windows)
- Attivazione del sistema Jess

```
$ java jess.Main
Jess>
```
- Uscita dal sistema Jess

```
Jess> (exit)
$
```

Attivazione dell'esempio

- Copia del file delle regole
 - Copiare il file `examples/dilemma.clp` nella propria directory
`cp /home/doc/mpiastra/Jess60/examples/dilemma.clp .`
- Attivazione del sistema Jess

```
$ java jess.Main
Jess>
```
- Caricamento del file `dilemma.clp`

```
Jess> (batch "dilemma.clp")
```
- Attivazione dell'esempio

```
Jess> (reset)
Jess> (run)
```
- Documentazione su Jess
 - Attivare Netscape o altro browser
 - Aprire il file:
`/home/doc/mpiastra/Jess60/docs/index.html`

Domande:

- a) Qual'è l'algoritmo utilizzato per risolvere il dilemma?
 - 1) fornire una spiegazione informale
 - 2) spiegare il significato della struttura `status`

- b) Qual'è il ruolo della priorità tra regole?
 - 1) provare a togliere le indicazioni di *salience*
 - 2) spiegare la differenza di comportamento

- Trascurare invece le regole di presentazione del risultato
 - nella sezione `"FIND AND PRINT SOLUTION RULES"` del file
 - (ma date un'occhiata alla regola `"recognize-solution"`)

3

Discussione *(dell'esercitazione con Jess)*

Spazio degli stati

- Il dilemma viene risolto con una ricerca esaustiva nello **spazio degli stati**
 - ogni struttura **status** rappresenta una possibile condizione
- Le regole corrispondenti alle azioni (del dilemma) derivano nuovi **stati**
 - ogni azione parte da uno **status** e produce un nuovo **status**
 - ne risulta la costruzione di una struttura ad albero la cui radice è lo **status** iniziale
- Le regole corrispondenti ai **vincoli** eliminano gli stati non permessi
 - impedendo così lo sviluppo di rami inutili
 - la priorità più alta impedisce che una regola di azione intervenga *prima* che la regola di vincolo abbia rimosso lo stato

Strategie come percorsi

- Le foglie dell'albero sono **soluzioni** o 'vicoli ciechi'
- Una **strategia** (per risolvere il dilemma)
 - è un percorso che porta dal nodo radice (stato iniziale)
 - ad un nodo soluzione
 - possono essere più di una