Intelligenza Artificiale

Jess – Esercitazione 1
Fox, Goat & Cabbage

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*
  - **`(deffact`** <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))
```

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`**")

Discussione

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