Corso di Linguaggi di Programmazione

Lezione 11

Alberto Ceselli
alberto.ceselli@unimi.it

Università degli Studi di Milano

16 Aprile 2013

Cut

- Predicato per influenzare il comportamento *procedurale* dei programmi
- Effetto: ridurre lo **spazio di ricerca** delle soluzioni **tagliando rami** dall'albero di ricerca
- **green cut:** migliorare l'efficienza, o rimuovere comportamenti inattesi, senza influenzare l'*esito* della computazione
- **red cut:** influenzano l'*esito* della computazione (se rimuovo il cut, ottengo un programma **diverso**).

Cut

- Cut è rappresentato dal simbolo **!**
- Effetto di un cut:
  - il cut **ha successo**, ed
  - indica all'interprete di **escludere** tutti i rami dell'albero di ricerca non esplorati
  - a partire dall'unificazione nel **goal padre** con la testa della clausola in cui è inserito il cut.
- la ricerca (anche nella stessa clausola) prosegue come prima.

Cut

Esempio: `?- csortedmerge([1,3,5],[2,4],R)`, con disegno dell'albero di ricerca (alla lavagna).

Cut

N.B. Il cut ha un effetto "istantaneo" (non ne rimane traccia nello stato dell'interprete). Ad esempio ...

- Un cut taglia tutte le clausole "sotto" di esso (nel db delle regole).
- Un cut taglia tutte le soluzioni alternative per i goals che appaiono alla sua sinistra nella stessa clausola
- Un cut non ha effetto sui goals che appaiono alla sua destra nella stessa clausola.

Cut e negazione

- Prolog assume di trattare con un "mondo chiuso" (closed world assumption)
- non è un sistema `true / false`, ma un sistema `true / fail`
- `not(X)` $\equiv$ `cannotprove(X).`
- come implementare `not(X)`?
- `not(X) :- X, !, fail.`
- `not(X).`
- Problemi: `negation.pl`
- stesso discorso per "diverso"

Esempio di utilizzo della negazione:

Predicato `alldifferent(Xs)`: ha successo se Xs è una lista di *tutti elementi diversi*.

Liste

- `append`, `delete`, `reverse`
- codice, discussione ed esempio di funzionamento dell'interprete su `delete` e variante `select`

Generate-and-test

- La programmazione logica è per natura *non-deterministica*.
- Intuitivamente, una macchina *non deterministica* è in grado di scegliere *automaticamente* e *correttamente* la computazione successiva tra varie alternative.
- Computazioni non-deterministiche sono *simulate* dall'interprete Prolog attraverso ricerche sequenziali e backtracking.

Generate-and-test

Esempio: permutation sort
(Oggi pomeriggio in Lab: the "n-queens" problem `queens_aop.pl`)

Accumulatori e tail recursion

- Esempio: `fact`
- Esempio: `ifact`

Difference-lists

- Le **difference lists** sono strutture dati *incomplete*
- Particolarmente utili come *accumulatori*
- `difference_lists.pl`
- (`sort.pl`)

Meta-programmazione Prolog

In Prolog è possibile modificare la base dati *inserendo fatti*:
`assert(X).`
o *rimuovendo fatti*:
`restract(X)`
oppure *collezionare elementi che soddisfano un certo predicato*:
`setof(X, p(X), Rs)` (senza duplicati)
`bagof(X, p(X), Rs)` (con duplicati)
`findall(X, p(X), Rs)` (tutte le variabili libere quantificate esistenzialmente)

Esercizi

- Esercizio "Dijkstra's dutch flag problem" : data una lista di elementi colorati 'rosso' 'bianco' 'blu', riordinare la lista in modo che tutti gli elementi rossi precedano gli elementi bianchi, e tutti gli elementi bianchi precedano gli elementi blu. Nell'ordinamento deve essere preservato l'ordine parziale degli elementi dello stesso colore nella lista di partenza. Realizzare due versioni del programma: una che utilizzi difference lists, ed una che non le utilizzi.

Struttura ed efficienza

`mapcolor.pl`