Controllo backtracking e negazione in Prolog

Sistemi intelligenti distribuiti LS
2005/2006
Prof. Andrea Omicini

Prevenire il backtracking

- *Il backtracking automatico è fondamentale*
  - *risparmia lavoro al programmatore*
- *A volte può essere utile controllare il backtracking*
  - *per migliorare l'efficienza*
  - *per evitare rami infiniti*
- *Soluzione*
  - *abbandonare la programmazione logica "pura"*
  - *utilizzare il cut*

Il cut

- *Sintassi*
  - *!*
  - *si applica come un atomo del corpo di una clausola*
  - *A :- B1, B2, …, Bi, !, …, Bj*
  - *non può andare nella testa*
- *Semantica*
  - *se sono selezionati l'atomo G e la clausola A :- B1, …, Bi, !, …, Bj*
    - *con G che unifica con A*
  - *e se l'esecuzione arriva con successo fino al cut*
  - *allora non si fa backtracking per B1, …, Bi*
  - *né eventualmente per G stesso*
    - *ossia, non si usano altre clausole dopo quella*

Esempio: calcolare il massimo

```
max(X,Y,X) :- X >= Y.
max(X,Y,Y) :- X < Y.
```

- *Come faccio a rendere le due regole mutuamente esclusive?*

```
max(X,Y,X) :- X >= Y, !.
max(X,Y,Y).
```

- *Cosa ottengo?*
  - *risparmio un backtracking inutile*
    - *nel caso di molte scelte mutuamente esclusive, può essere importante*
- *Pericoli?*
  - *in questo caso no…*
  - *ma in generale potrei perdere soluzioni*
- *Commitment delle scelte dall'unificazione della testa al cut*

Cut & fail

- *Indurre il fallimento*
  - *predicato* `fail`
  - *induce fallimento sul ramo corrente*
    - *a che serve???*
- *Giovanni ama i linguaggi di programmazione, tranne quelli logici*

```
ama(giovanni,linguaggio(X)) :- logico(X), !, fail.
ama(giovanni,linguaggio(X)).
```

- *La combinazione cut & fail si usa spesso*
  - *è però un surrogato della negazione*
    - *c'è quindi un operatore di negazione apposta*

Not: *negation as failure*

- *Definizione operazionale*

```
not(P) :- P, !, fail ; true.
```

- *cioè, not(P) ha successo se la dimostrazione di P fallisce*
- *Esempio*

```
ama(giovanni,linguaggio(X)) :- not logico(X).
```

Problemi con il cut

- *Ci sono usi del cut che preservano il significato dichiarativo dei programmi logici*
  - *cut verdi*
  - *usati per aumentare l'efficienza o evitare rami infiniti*
- *Altri cut (rossi) invece fanno perdere il significato dichiarativo*
  - *può essere voluto*
    - *per eliminare "soluzioni"*
  - *può essere invece un errore*
- *Comunque, con il cut bisogna fare ancora maggiore attenzione all'ordine delle clausole*

Problemi con la negazione

```
?- not simpatico(luca).
no
```

- *Di solito, non significa che ho dimostrato che Luca* è *antipatico*
  - *adottando per default l'interpretazione ovvia*
- *Significa solo che "non ho prove che sia simpatico"*
  - *cioè che non esiste nella mia teoria un fatto* `simpatico(luca).`
  - *o non lo posso derivare da altri fatti*
- *Ciò che non so* è *assunto essere falso*
  - *Close World Assumption*
- *Questo chiaramente limita il significato della negazione*