PROLOG

DALLA PROGRAMMAZIONE LOGICA A PROLOG

- regola di selezione: leftmost
- regola di ricerca: depth-first con backtracking basato sull'ordinamento delle clausole

- algoritmo di unificazione senza occur-check (a richiesta in alcune versioni più nuove)

  - motivo principale: l'efficienza dell'implementazione

- a parte queste differenze, per il resto PROLOG fornisce numerosi meccanismi, primitive, operatori molto utili in pratica anche se spesso poco coerenti con i suoi fondamenti

  - per esempio, il <u>cut</u> è molto utile per migliorare l'efficienza dei programmi, talora permette di scrivere programmi più concisi, ma in generale fa sì che non ci sia più relazione tra semantica dichiarativa e semantica procedurale.

LE PRIMITIVE

- alcuni tipi di dato primitivi (tipicamente l'aritmetica)
    - le operazioni primitive non sono le stesse che uno scriverebbe in PROLOG
        - per esempio, la somma funziona solo se gli vengono forniti 2 valori (completi) come input
- influenza la semantica dei predicati che le utilizzano

- input-output, ovviamente con le osservazioni fatte sopra
    - write(X) non è invertibile!

- alcune primitive sono in realtà primitive di meta-livello e permettono di analizzare la struttura interna di un termine, senza ricorrere all'unificazione
    - var(X), integer(X), functor(T, F, N), etc.

IL PREDICATO DI CONTROLLO CUT

• Prolog mette a disposizione il predicato di controllo "cut" (!), uno strumento molto potente per il controllo del comportamento procedurale dei programmi.

• Lo scopo principale del cut è dare la possibilità di controllare dinamicamente lo spazio di ricerca dei programmi Prolog, consentendo di tagliare nel corso della computazione rami "morti" o ritenuti senza importanza.

* Usare il cut nei programmi può modificare pesantemente il loro significato originale

* Il significato del cut può essere compreso solo tenendo conto della interpretazione procedurale dei programmi: si perde così molto dello stile dichiarativo di Prolog

* Per questi motivi, il cut va usato con molta cautela; d'altra parte però un suo uso controllato e adeguato migliora di molto l'efficienza dei programmi senza comprometterne la chiarezza e consente di usare tecniche avanzate di programmazione

PREDICATI DI CONTROLLO

- IL CUT

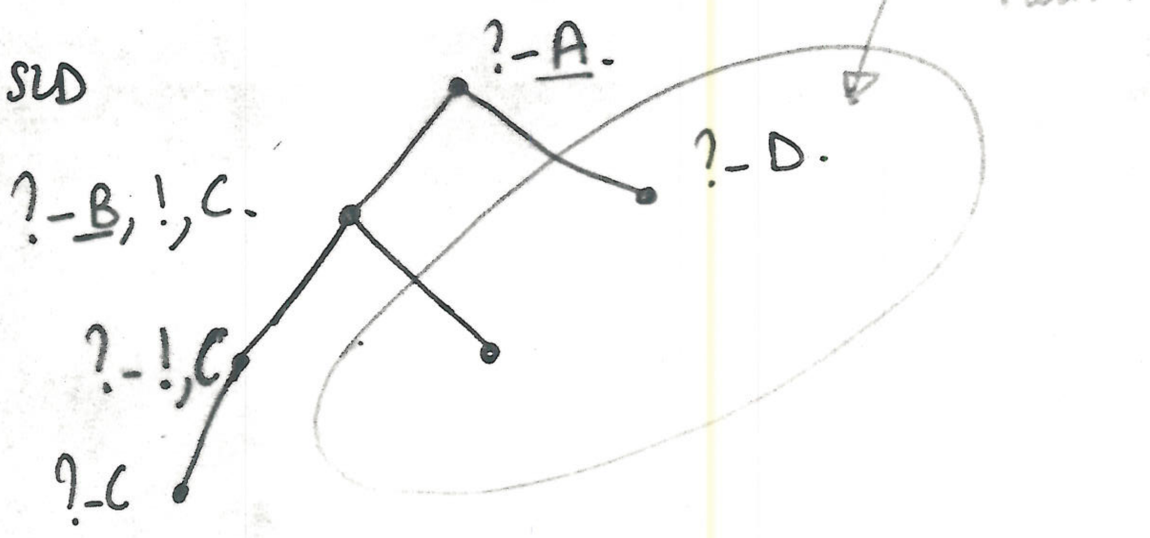
A :- B, !, C.
A :- D.

- goal ?- A. ⇒ ?- B, !, C.
  - viene trovata una soluzione per B ⇒ ?- !, C.
  - ! ha successo, ma, come effetto laterale, predispone quanto serve a realizzare ciò che vedremo

⇒ ?- C.

- se C fallisce, il goal fallisce senza fare altri tentativi con
  - altre clausole per A
  - altre clausole per B

- sull'albero SLD



- È chiaro che, in generale, si possono perdere soluzioni e quindi anche quel po' che resta di completezza.

IL CUT

• Un uso tipico del cut è il suo inserimento in un programma allo scopo di esplicitare che si tratta di un programma deterministico:

il caso più comune è dato dalla combinazione "test,!"

◦ Esempio

```
minimo(N,M,N):-  N<=M.
minimo(N,M,M):-  M<=N.
```

per N=M ci sono 2 soluzioni; perciò si modifica il programma
così:

```
minimo(N,M,N):-  N<=M,!.
minimo(N,M,M):-  N>M.
```

◦ Esempio

merge(Xs,Ys,Zs) Zs è una lista ordinata ottenuta fondendo 2 liste ordinate di interi Xs e Ys

```
merge(Xs,[ ],Xs):-!.

merge([ ],Ys,Ys):-!.

merge([X\Xs],[Y\Ys],[X\Zs]):-X<Y,!,merge(Xs,[Y\Ys],Zs).

merge([X\Xs],[Y\Ys],[X,Y\Zs]):-  X=Y,!,merge(Xs,Ys,Zs).

merge([X\Xs],[Y\Ys],[Y\Zs]):-  merge([X\Xs],Ys,Zs).
```

GLI OPERATORI LOGICI

- disgiunzione (;), che, in assenza di !, può essere visto solo come zucchero sintattico

- not implementazione scorretta della negazione come fallimento finito

- if then else basato sull'or e sul not sbagliato

- in qualche modo legati alla quantificazione, e, comunque, di metalivello, i predicati di tipo "all-solutions" : restituiscono la lista dei valori di una (o più) variabile nelle diverse soluzioni di un dato goal

IL DATA BASE DI CLAUSOLE

- equivalenza tra dati e programmi
  - le clausole: hanno una rappresentazione come dati, possono essere
    - accedute
    - asserite
    - eliminate

    Head :- Body

    - clause (Head, Body)
    - assert (clause)
    - retract (clause)
  - i dati possono essere trasformati in goal ed eseguiti
    - call (Term)
- queste operazioni stanno alla base delle tecniche di metaprogrammazione, che sono una delle più forti ragioni della fortuna di PROLOG
  - si può scrivere un interprete di PROLOG in PROLOG in 4 semplicissime clausole
- sono tutte operazioni di metalivello non previste dalla teoria della programmazione logica

Predicati extra-logici

Esempio:

```
P:  p(a) :- q(X),r.
    p(b).
Q:     ?-clause(p(X),B).
    X= a, B= q(X1),r
    X=b,  B=true
```

Aggiungere clausole:

assert(Clause)
: aggiunge Clause in fondo alla definizione corrispondente

asserta(Clause)
: la aggiunge in testa

In entrambi i casi Clause deve avere la testa istanziata.

Togliere clausole:

retract(C)
: toglie la prima clausola che unifica con C

Programmazione con side effects: problemi con la semantica, si deve usare solo in caso di necessità.

CAMBIARE PROLOG?

- non certo per i suoi "vizi" dal punto di vista teorico, ampiamente compensati da
  - flessibilità (facilità di estensione)
  - potere espressivo
  - efficienza
  - soprattutto dopo aver ottenuto risultati anche molto raffinati sulla compilazione (WAM), sull'ottimizzazione, e sulle implementazioni parallele.

- potrà eventualmente essere soppiantato solo da linguaggi (loro implementazioni) che offrono un paradigma più generale
  - un potenziale aspirante

    constraint logic programming

    (anche Prolog II e Prolog III)