

Lezioni 13 e 14

PROLOG FINALE

Elenco caratteristiche prolog viste e non viste, con completamento di alcuni aspetti relativi al lessico

- **Costanti** sono anche '...', ad esempio 'X'
- **Commenti**: /* */
- **Direttive**: :- Body.
- Direttiva per includere un programma: :- include(nomefile)
- **Operatori**
op(precedenza, associazione, simbolo)

Uso delle direttive.

Esempio tipico 1. Caricare programmi.

```
:- consult(impianto).  
:- consult(elettricista).  
esecuzione
```

Esempio tipico 2: introdurre operatori

```
:- op(200, fx, s)  
posso scrivere s s s 0 anziché s(s(s(0)))
```

Associazioni:

- fx : unario, non associativo
esempio: op(200, fx, il):
posso scrivere: il cane ma non: il il cane
- fy : unario, associativo
esempio: op(200, fy, s):
posso scrivere: s s s 0 al posto di s(s(s(0)))
- xfx : binario, non associativo
esempio: op(200, fx, '<-'):
posso scrivere: A <- B, ma non A <- B <- C
- xfy : binario, associativo a destra
esempio: op(200, xfy, &):
posso scrivere: A & B & C al posto di A & (B & C)
- yfx : binario, associativo a sinistra
esempio: op(200, yfx, '^'):
posso scrivere: 1^0^1^1 al posto di ((1^0)^1)^1

```
%ESEMPIO:  
%tab(C1,C2,C,R^°S) tabellina binaria: R^°S = C1+C2+R
```

```
:- op(100, yfx, ^°).
```

```
tab(C1,C2,C,R^°S) :- C1 \== C2, !, (C = 0, R^°S = 0^° 1;  
C = 1, R^°S = 1^° 0).
```

```
tab(C1,_C,R^°S) :- (C1=0, R^°S = 0^° C;  
C1 = 1, C = 0, R^°S = 1^° 0;  
C1 = 1, C = 1, R^°S = 1^° 1).
```

Aritmetica:

- Costanti intere 53, -5, ...
- Costanti reali: 2.5, ...
- Operazioni: +, *, -, // (intera), / (reale), mod
- Predicati: ==, ==, <, >, >=, <=
- Quando un'espressione (termine o predicato) E è valutata con sostituzione corrente σ , E σ dev'essere ground
- **Assegnamento**: X is E; in questo caso:
 - E σ dev'essere ground;
 - X σ dev'essere una costante numerica o una variabile

ESEMPIO.

$\text{pow}(X,Y,Z)$: significato inteso: $Z = X$ elevato ad Y .

vedere programmi fatti la volta scorsa

- **Controllo:** !
- Esempio: la negazione come fallimento con il cut:
 $\text{not}(A) :- A, !, \text{fail}.$
 $\text{not}(_).$
- **Disgiunzione**, indicata con ; :
 $A :- \text{Body1}.$
 $A :- \text{Body2}.$
equivale logicamente a
 $A :- \text{Body1} ; \text{Body2}.$

Controllo: if C then A else B si scrive $C \rightarrow A ; B$

$H :- (C \rightarrow A ; B).$

equivale a

$H :- C, !, A.$

$H :- B.$

Si ha una *differenza nel backtracking*:

il cut influisce su tutte le clausole che definiscono il predicato di H,
mentre $C \rightarrow A ; B$ influisce solo localmente.

ESEMPIO:

```
tab(C1,C2,C,R ° S) :-  
  C1 =\= C2 -> (C = 0 -> R ° S = 0 ° 1;  
                R ° S = 1 ° 0);  
  (C1=0 -> R ° S = 0 ° C;  
   (C = 0 -> R ° S = 1 ° 0;  
    R ° S = 1 ° 1)).
```

Esempio della somma

- Concludiamo l'esempio della somma di numeri binari con un programma aperto;
- Se istanziate i predicati tab e cifra con un sistema a k cifre, ottenete un sommatore in quella base.

```
sum(X,Y,R) :- cifra(X), !,  
               addCifra(Y,X,R).  
sum(X,Y,R) :- cifra(Y), !,  
               addCifra(X,Y,R).  
sum(X ° C1, Y ° C2, U) :- sumR(X ° C1, Y ° C2, 0, U).  
  
sumR(X ° C1, Y ° C2, R, V ° S) :- !, tab(C1,C2,R, C ° S),  
                                   sumR(X,Y,C,V).  
sumR(X,Y,R,U) :- sum(X,Y,V),  
                  addCifra(V,R,U).  
  
addCifra(X,0,X) :- !.  
addCifra(X ° C1, C2, V ° S) :- !, tab(C1, C2, 0, C ° S),  
                                addCifra(X, C, V) .  
addCifra(C1, C2, R) :- tab(C1, C2, 0, R).
```

Esercizio 1: completare per ottenere il sommatore binario.

Esercizio 2: completare per ottenere un sommatore in base 1000, dove le cifre sono numeri fra 0 e 999; ad esempio 1 ° 54 ° 321 significa 1.054.321.

Assert e retract

- Vi sono istruzioni per operare su basi dati, ovvero su insiemi di fatti e regole che possono essere modificati dinamicamente
 - Simili a updates in basi dati
- `assert(C)` aggiunge una clausola alla base dati
- `asserta(C)` la aggiunge all'inizio
- `assertz(C)` la aggiunge alla fine
- `retract(C)` toglie la prima della base dati che unifica con C
- `retractall(C)` le toglie tutte (se disponibile)

ESEMPIO: generazione di indici nuovi

```
:- dynamic(ind/1).
```

```
newInd(I) :- retract(ind(J)), !,  
             I is J+1,  
             assert(ind(I)).  
newInd(0) :- assert(ind(0)).
```

Studiate il libro a pag. 486, 487; è molto interessante; per capire alcune parti guardare le difference lists, pag. 85

Famiglie di programmi (schemi)

- Prolog è adatto per costruire famiglie di programmi, derivanti da schemi generali.
- ***Vediamo lo schema del divide et impera***

- `p(X,Y)` predicato per un problema I/U
 - ingresso: lista X di argomenti,
 - uscita: lista Y di argomenti, eventualmente vuota
- `base(X,U)` caso base della ricorsione su X
- `div(X,Y,Z)` divide X in due parti Y,Z **minori** di X secondo un **ordinamento ben fondato nell'insieme dei valori non base**
- `compose(A,B,C)` ricomposizione dei risultati ricorsivi

Proprietà di correttezza:

```
base(X,U) → p(X,U)  
div(X,Y,Z), p(X,A), p(Y,B), compose(A,B,C) → p(X,C)
```

```
p(X,U) :- (base(X,U) ->  
           true;  
           divide(X,Y,Z), p(Y,A), p(Z,B), compose(A,B,U)).
```

Se si trova una misura *size* : ingressi → Naturali tale che:
div(X,Y,Z) → size(Y) ≤ size(X/2) e size(Z) ≤ size(X/2)
per X non base, si ha un numero logaritmico di chiamate ricorsive.

ESERCIZIO:

Per il sort delle liste istanziamo lo schema come segue:

```
base(X,U) :- (X=[]; X=[_]), U=X.  
div([H,K|L],Y,Z) può essere dettagliato in molti modi  
compose(A,B,U) è l'usuale operazione di merge
```

Completare il tutto per ottenere l'insertion sort e il merge sort, lasciando aperto l'ordinamento sugli elementi.

Esercizio

- Lo schema completato con il programma per le potenze è:

```
/* ***** DIVIDE ET IMPERA ***** */
p(X,U) :- (base(X,U) -> true;
          divide(X,Y,Z), p(Y,A), p(Z,B), compose(A,B,U)).

/* ***** ISTANZIAZIONE del principio per la potenza ***** */
pow(B,E,U) :- B>0, E>=0, p([B,E],U).
base([_,Y],U) :- Y=0, !, U=1.
base([X,Y],U) :- Y=1, !, U=X.
divide([B,X],[B,Y],[B,Z]) :- Y is X // 2, Z is X - Y.
compose(A,B,U) :- U is A * B.
```

- Trovare una versione che non usi direttamente questa implementazione, sostituendo le definizioni ai predicati nello schema. Tale procedimento è uno schema di trasformazione generale, detto di unfold-fold.

- Se si toglie la clausola
base([X,Y],U) :- Y=1, !, U=X.

l'ordinamento realizzato da

divide([B,X],[B,Y],[B,Z]) :- Y is X // 2, Z is X - Y.

è ancora ben fondato nell'insieme dei casi contemplati ora dalla clausola ricorsiva? Discutere la cosa e fare delle prove di testing, eventualmente con l'aiuto della trace.

Difference lists, difference nats,

sum(X-Y,Y-Z,X-Z).

Esecuzioni in aula.