Prolog – Esercitazione 02

- *Scopo:* Risolvere alcuni problemi mediante il linguaggio Prolog
  - Ricorsione e ricorsione tail
  - Liste
  - Cut
  - setof, bagof, findall
  - Meta-interpreti

Es. 1

Si definisca un predicato in PROLOG chiamato **`filtra`** che applicato ad una lista di elementi (tra cui alcuni interi) dia come risultato la lista dei soli interi, scartando tutti gli altri elementi. Si definisca direttamente la versione ricorsiva-tail.

A tal scopo si usi il predicato **`number/1`**, che è valutato come vero se e solo se l'argomento è un numero.

Esempio:

```
?-
?- filtra([7,[3,10,2], cdddc,[1,2], 13], X).
yes, X = [7, 13]
```

Es. 1 - Soluzione

```
filtra([], []).

filtra([H|T], [H|R1]):-
  number(H), !, filtra(T, R1).

filtra([_|T], R1) :-
  filtra(T, R1).
```

Es. 1bis

Si definisca un predicato in PROLOG chiamato **`filtra`** che applicato ad una lista di elementi (tra cui alcuni interi) dia come risultato la lista dei soli interi, scartando tutti gli altri elementi. Qualora un elemento sia una lista, si estraggano da questa gli elementi numerici e li si riportino nella soluzione (similmente al "*flattening*" di una lista).

A tal scopo si usi il predicato **`number/1`**, che è valutato come vero se e solo se l'argomento è un numero.

Esempio:

```
?-
?- filtra([7,[3,10,[2]], cdc,[1,2], 13], X).
yes, X = [7, 3, 10, 2, 1, 2, 13]
```

Es. 1bis - Soluzione

```
filtra([], []).

filtra([H|T], [H|R1]):-
   number(H), !, filtra(T, R1).

filtra([[H1|T1]|T], R) :-
   !,
   filtra([H1|T1], R1),
   filtra(T, R2),
   append(R1, R2, R).

filtra([_|T], R1) :-
   filtra(T, R1).
```

Es. 2

L'anagrafe di una banca rappresenta le informazioni riguardo i nuclei familiari con fatti prolog del tipo:

```
padre(emilio,franco).
padre(franco, federico).
padre(franco, paolo).
padre(federico, francesco).
padre(federico, chiara).
padre(paolo, tommy).
padre(paolo, mary).
```

Es. 2

Si definisca un predicato **`antenato(X,Y)`** che venga valutato come vero se X è un antenato di Y. Ad esempio, le seguenti query devono avere le seguenti risposte:

```
:- antenato(federico, paolo). no
:- antenato(emilio, francesco). yes
```

Si definisca un predicato **`successore(X,Y)`** che venga valutato come vero se X è un successore di Y. Ad esempio, le seguenti query devono avere le seguenti risposte:

```
:- successore(federico, paolo). no
:- successore(francesco, emilio). yes
```

Es. 2

Si definisca il predicato **`parente_diretto(X,L)`** che ricevuto un termine ground in X, restituisca in L la lista degli antenati e dei successori di X. Ad esempio, se invocato:

```
:- parente_diretto(federico, L).
   yes, L/[franco,emilio,francesco,chiara].
```

Ulteriore problema: come si può calcolare l'elenco dei "parenti indiretti"?

Es. 2 - Soluzione

```
padre(emilio, franco).
padre(franco, federico).
padre(franco, paolo).
padre(federico, francesco).
padre(federico, chiara).
padre(paolo, tommy).
padre(paolo, mary).

antenato(X,Y) :- padre(X,Y).
antenato(X,Y) :- padre(X,Z), antenato(Z,Y).

successore(X,Y):- padre(Y,X).
successore(X,Y):- padre(Y,Z), successore(X,Z).

parente_diretto(X,Y) :-
   findall(A, antenato(A, X), L1),
   findall(B, successore(B, X), L2),
   append(L1, L2, Y).
```

Es. 3 – Un semplice meta-interprete

Si scriva un meta-interprete prolog che stampi a video, prima e dopo l'invocazione di ogni sottogoal, tale sotto-goal. Per stampare a video si può usare la primitiva **`write/1`** che stampa l'argomento a video, ed **`nl/0`** che stampa un newline.

Esempio:

```
p(X) :- q(X).
q(1).
q(2).

?- solve(p(X)).
yes   X/1
'Solving: 'p(X_e0)
'Selected Rule: 'p(X_e0)':-'q(X_e0)
'Solving: 'q(X_e0)
'Selected Rule: 'q(1)':-'true
'Solved: 'true
'Solved: 'q(1)
```

Es. 3 - Soluzione

Il punto di patenza (quasi sempre) è il meta-interprete vanilla:

```
solve(true):-!.
solve((A,B)):-!,solve(B),solve(A).
solve(A) :- clause(A,B),solve(B).
```

La soluzione si può ottenere modificando l'interprete sopra:

```
solve(true):- !.
solve((A,B)):- !, solve(A), solve(B).
solve(A) :-
   write('Solving: '), write(A), nl,
   clause(A,B),
   write('Selected Rule: '), write(A), write(":-"), write(B), nl,
   solve(B),
   write('Solved: '), write(B), nl.
```

Es. 3bis – Un semplice meta-interprete

Modificare il meta-interprete precedente affinchè stampi i sottogoal con una indentazione variabile in dipendenza della profondità del sotto-goal nell'albero sld

Esempio:

```
p(X) :- q(X).
q(1).
q(2).

?- s (p(X)).
yes   X/1
'Solving: 'p(X_e0)
   'Selected Rule: 'p(X_e0)':-'q(X_e0)
   'Solving: 'q(X_e0)
      'Selected Rule: 'q(1)':-'true
      'Solved: 'true
   'Solved: 'q(1)
```

Es. 3bis - Soluzione

La soluzione si può ottenere modificando l'interprete sopra:

```
s(true, N):- !.
s((A,B), N):- !, s(A, N), s(B, N).
s(A, N) :-
   tt(N), write('Solving: '), write(A), nl,
   clause(A,B),
   N1 is N+1,
   tt(N1), write('Selected Rule: '), write(A), write(":-"),
   write(B), nl,
   s(B,N1),
   tt(N1), write('Solved: '), write(B), nl.

tt(0).
tt(N):-
   N>0,
   tab(3),
   N1 is N-1,
   tt(N1).
```

Es. 4 – tratto dal Compito del 31 gennaio 2013

Si scriva un meta-predicato:

```
predicati(ListaAtomi, ListOut)
```

che, dato in ingresso una ListaAtomi, restituisce in ListOut (lista di uscita), le coppie (simbolo di predicato, numero di argomenti) di ciascun atomo in ListaAtomi.

Ad esempio, l'invocazione seguente ha successo con esito come indicato:

```
?-
      predicati([p(X, Y), q(a), g(1, 2,d), p(X,S)],
                          ListOut).

yes,
ListOut=[(p,2), (q,1), (g,3), (p,2)]
```

Es. 4 – soluzione

```
predicati([], []).
predicati([Atom|Rest], [(PredName,PredN)|LOut]) :-
        Atom =.. [PredName|Lista],
        length(Lista,PredN),
        predicati(Rest,LOut).
```

Es. 5 – tratto dal Compito del 10 gennaio 2013

Si scriva un meta-predicato:

trova_tutte(Goal, ListOut)

che, dato in ingresso un Goal con predicato binario, restituisce in ListOut (lista di uscita), le soluzioni (come coppie di elementi) derivanti dalla sua invocazione.

Ad esempio, supponendo di avere anche le clausole:

```
p(1,2).
p(2,3).
p(a,b).
```

l'invocazione seguente ha successo con esito come indicato:

```
?- trova_tutte(p(X,Y), ListOut).
yes L=[(1,2), (2,3), (a,b)]
```

Es. 5 – soluzione

```
trova_tutte(Atom, Lout) :-
        Atom =.. [PredName, X, Y],
        findall((X,Y), call(Atom), Lout).
```