Prolog – Esercitazione 03

Meta-interpreti

Es. 1

Si costruisca un meta-interprete per Prolog che chieda all'utente i goal che non è in grado di dimostrare, solo se per essi NON compaiono fatti ground nel database con lo stesso funtore e arità.

Questo meta-interprete può essere utile nella realizzazione di sistemi esperti dove, spesso, il sistema chiede all'utente goal che non è in grado di dimostrare.

L'utente può rispondere in diversi modi:

1) può semplicemente asserire true o false.
2) può rispondere true istanziando eventuali variabili unbound.

Es. 1 – Soluzione (1)

```
meta1(true):-!.
meta1((X,Y)):- !, meta1(X), meta1(Y).
meta1(X):- clause(X,Body), meta1(Body) , !.

% Non ho dimostrato X: cerco fatti ground nel database.
% Se non li trovo chiedo all'utente con ask/1 se X è true
% o false
meta1(X):-
   \+(fatti(X)),
   ask(X,Risp), call(Risp).

fatti(X):-
   functor(X,P,A),
   functor(K,P,A),
   clause(K,true),
   K=..[_|Arg],
   gr(Arg).
```

Es. 1 – Soluzione (1)

```
ask(X,Risp):- write(X), write("  true or false ? "),
              read(Risp).

% gr controlla che gli argomenti siano ground.
gr([]).
gr([Argh|Argt]) :- controllo(Argh), gr(Argt).

% Caso di una costante come argomento
controllo(B) :- atomic(B), !.
% Caso di una lista come argomento.
controllo([H|T]):- !, controllo(H), controllo(T).
% Caso di un termine composto come argomento.
controllo(B):- compound(B), B=..[_|L], controllo(L).
```

Es. 1 – Soluzione (2)

```
ask(X,Risp):- write(X), write("  bindings ? "),
              read(Risp),
              asserta(Risp). % modifico il db...
% OPPURE???

% gr controlla che gli argomenti siano ground.
gr([]).
gr([Argh|Argt]) :- controllo(Argh), gr(Argt).

% Caso di una costante come argomento
controllo(B) :- atomic(B), !.
% Caso di una lista come argomento.
controllo([H|T]):- !, controllo(H), controllo(T).
% Caso di un termine composto come argomento.
controllo(B):- compound(B), B=..[_|L], controllo(L).
```

Es. 2

Si scriva un meta-interprete che riceva in ingresso, oltre al goal da dimostrare, il nome di un predicato e la sua arità e conti, nel corso della computazione (solo i rami di successo) il numero di chiamate al predicato.

Ad esempio, dato il programma:

```
p(X):- q(X), r(X,Y).
q(1).
q(2).
r(1,Y):- s(Y).
r(2,Y):- t(Y).
t(3).
```

Il meta-interprete risponde alla chiamata solve(p(Y), r,2,N) istanziando N a 1.

Es. 2 - Soluzione

```
solve(Goal,Pred,Arity,N):-
   solve(Goal,Pred,Arity,0,N).

solve(true,_,_,N,N).
solve((A,B), Pred, Arity, Acc, Accout):-
   solve(A,  Pred, Arity, Acc,     AccTemp),
   solve(B,  Pred, Arity, AccTemp, Accout).

% caso 1: il goal coincide col predicato da contare
solve(Goal, Pred, Arity, Accin, Accout):-
   functor(Goal,Pred,Arity), !,  % test se coincide... e poi cut
   AccTemp is Accin + 1,
   clause(Goal, Body),
   solve(Body,Pred,Arity,AccTemp,Accout).

%caso 2: il goal non e' il predicato da contare...
solve(Goal, Pred, Arity, Accin, Accout):-
   clause(Goal, Body),
   solve(Body, Pred, Arity, Accin, Accout).
```

Es. 3 – predicati speciali e sospensione

Si scriva un meta-interprete in grado di riconoscere i predicati is/2 e di valutare se il predicato stesso deve essere sospeso o eseguito. Si considerino solo espressioni (a destra di is) a due argomenti. Nel caso in cui i due argomenti a destra del predicato non siano sufficientemente istanziati, il predicato deve essere sospeso ponendolo in un'apposita lista; in caso contrario viene eseguito normalmente.

Ad esempio A is 2+3 viene sempre eseguito mentre A is B+C viene sospeso se una delle variabili B o C non è istanziata.

Al termine della dimostrazione, il meta-interprete deve controllare se nella lista dei predicati sospesi ne compaiono alcuni che possono essere eseguiti perché sufficientemente istanziati.

Se la valutazione del goal termina con successo, ma la lista dei predicati sospesi contiene ancora predicati non sufficientemente istanziati, il meta-interprete restituisce la lista dei goal sospesi.

Es. 3 - Soluzione

Hp: si hanno a disposizione fatti del tipo **`clausola(Head,Body)`** dove **`Body`** è una lista di sotto-goal.

Il meta-interprete **`interi/3`** ha come parametri la lista dei goal da eseguire, la lista dei goal sospesi (dovrebbero risultare esserci solo is/2) in ingresso e la lista dei goal sospesi in uscita:

Caso base:

```
interi([],[],_) :- !.
```

Es. 3 - Soluzione

Nel caso in cui si sia giunti al termine dell'esecuzione di un goal e la lista dei predicati sospesi non sia vuota, si deve controllare se tale lista contiene predicati che possono essere eseguiti -> a seconda, diversi casi possibili…

Caso: "sono rimasti degli is/2, e tutti non risolvibili"

Idea: verifico che siano tutti **`is/2`**, che siano non risolvibili, e allora li stampo a video. **`tutti_is_non_solvibili/1`** ha successo se sono tutti **`is/2`**, e sono non risolvibili…

```
interi([],L,_):-
        tutti_is_non_solvibili(L), !,
        write('Lista di goal sospesi:'),
        nl,
        stampa(L).
```

Es. 3 - Soluzione

Caso: "qualcuno di quelli rimasti è risolvibile"

Idea: si prova a risolvere tutta la lista di quelli rimasti sospesi…

```
interi([],L,_):-
        interi(L,[],_).
```

Caso di una lista generica di goal:

```
interi([Goal | Altri], L2, L1):- !,
        interi(Goal, L2, Ltemp),
        interi(Altri, Ltemp, L1).
```

Es. 3 - Soluzione

Rimangono da risolvere i casi in cui il meta-interprete è invocato con un termine (non lista) come goal. Tre casi ulteriori: il goal è **`is/2`** e può essere risolto; il goal è **`is/2`** e non può essere risolto; il goal non è **`is/2`**.

Caso in cui il predicato è is/2 ma gli argomenti sono sufficientemente istanziati (cioè può essere risolto):

```
interi(Goal, Lgoal, Lnuovi):-
        is_solvable(Goal), !,
        call(Goal),
        Lgoal=Lnuovi.
```

Caso in cui il predicato è is/2 ma gli argomenti non sono sufficientemente istanziati:

```
interi(Goal, Lgoal, Lnuovi):-
        is_not_solvable(Goal), !,
        append(Lgoal, [Goal], Lnuovi).
```

Es. 3 - Soluzione

Rimangono da risolvere i casi in cui il meta-interprete è invocato con un termine (non lista) come goal. Tre casi ulteriori: il goal è **`is/2`** e può essere risolto; il goal è **`is/2`** e non può essere risolto; il goal non è **`is/2`**.

Caso normale (il goal non è un is/2):

```
interi(Goal, Lgoal, Lgoal1):-
        clausola(Goal, Body),
        interi(Body, Lgoal, Lgoal1).
```

Es. 3 - Soluzione

Predicati di supporto:

```
is_solvable(Goal):-
        Goal=..[is, _, B],
        B=..[_, L, K],
        nonvar(L), nonvar(K), !.

is_not_solvable(Goal):-
        Goal=..[is, _, B],
        B=..[_, L, K],
        (var(L) ; var(K)).
```

Es. 3 - Soluzione

Predicati di supporto:

```
tutti_is_non_solvibili([]).
tutti_is_non_solvibili( [Lh | Lt] ):-
        is_not_solvable(Lh), !,
        tutti_is_non_solvibili( Lt ).
tutti_is_non_solvibili ([Lh | _]):-
        is_solvable(Lh), fail.


stampa([]).
stampa([H|T]):-
        write(H),nl,
        stampa(T).
```

Es. 3bis

Riscrivere il meta-interprete usando la clause/2 standard del prolog (no ipotesi semplificativa iniziale).

Variante: Riscrivere il meta-interprete rendendo il predicato da sospendere come "parametrico"…

Variante bis: specificare tramite un predicato apposito i criteri per cui la valutazione di un predicato debba essere sospesa o meno…

Variante tris: si estenda il meta-interprete ai casi in cui a destra di is ci siano espressioni complesse con più di due variabili…