ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Prolog – Meta-predicati

Federico Chesani

DISI

Department of Informatics – Science and Engineering

Disclaimer & Further Reading

- These slides are largely based on previous work by Prof. Paola Mello

Meta-predicati

- In Prolog predicati (programmi) e termini (dati) hanno la stessa struttura e possono essere utilizzati in modo interscambiabile

```
sum(0,X,X).
sum(s(X),Y,s(Z)):- sum(X,Y,Z).
```

- Operatore (non associativo) `:-`

```
:-(sum(0,X,X),true).
:-(sum(s(X),Y,s(Z)), (sum(X,Y,Z))).
```

Tutto è un termine (meta-programmazione)

Il meta-predicato `call`

- Molti meta-predicati hanno come argomenti termini che rappresentano parti di programma

- Un primo predicato predefinito che può essere utilizzato per trattare i dati come programmi è il predicato **`call`**

- **`call(T)`**: il termine **`T`** viene trattato come un atomo predicativo e viene richiesta la valutazione del goal **`T`** all'interprete Prolog
  - Al momento della valutazione **`T`** deve essere istanziato ad un termine non numerico (eventualmente contenente variabili)

Il meta-predicato `call`

- Il predicato **`call`** può essere considerato come un predicato di meta-livello in quanto consente l'invocazione dell'interprete Prolog all'interno dell'interprete stesso

- Il predicato **`call`** ha come argomento un predicato (termine non variabile e non numerico)

```
p(a).
q(X):-  p(X).

:- call(q(Y)).
yes Y = a.
```

Il predicato **`call`** richiede all'interprete la dimostrazione di **`q(Y)`**

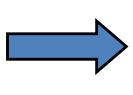
Il meta-predicato `call`

- Il predicato **`call`** può essere utilizzato all'interno di programmi

```
p(X):-  call(X).
q(a).

:- p(q(Y)).
yes Y = a.
```

- Una notazione consentita da alcuni interpreti (tra cui SWI Prolog, e SWIsh) è la seguente

```
p(X):- X.
```



`X` variabile meta-logica

Il meta-predicato `call` – Esempio

- Si supponga di voler realizzare un costrutto condizionale del tipo **`if_then_else`**

`if_then_else(Cond,Goal1,Goal2)`

Se **`Cond`** è vera viene valutato **`Goal1`**, altrimenti **`Goal2`**

```
if_then_else(Cond,Goal1,Goal2):-
   call(Cond), !,
   call(Goal1).
if_then_else(Cond,Goal1,Goal2):-
   call(Goal2).
```

Il meta-predicato `call` – Esempio

```
p(1).
p(2).
q(1).
q(2).
r(1).
k(3).

if_then_else(Cond,Goal1,_):-
    call(Cond), !,
    call(Goal1).
if_then_else(_,_,Goal2):-
    call(Goal2).

?-if_then_else((q(X),p(X)),r(X),k(X)).
Yes X=1
```

Il meta-predicato `fail`

- **`fail`** è un predicato predefinito senza argomenti

- La valutazione del predicato **`fail` fallisce sempre** e quindi forza l'attivazione del meccanismo di backtracking

- Vedremo alcuni esempi di uso del predicato fail:
  - Per ottenere forme di iterazione sui dati;
  - Per implementare la negazione per fallimento;
  - Per realizzare una implicazione logica.

Il meta-predicato `fail` – Esempio: iterazione

- Si consideri il caso in cui la base di dati contiene una lista di fatti del tipo **`p/1`** e si voglia chiamare la procedura **`q`** su ogni elemento **`x`** che soddisfa il goal **`p(X)`**

- Una possibile realizzazione è la seguente:

```
itera :- call(p(X)),
         verifica(q(X)),
         write(X),nl,
         fail.
itera.

verifica(q(X)):- call(q(X)), !.
```

Nota: il **`fail`** innesca il meccanismo di backtracking quindi tutte le operazioni effettuate da q(X) vengono **perse**, tranne quelle che hanno effetti non backtrackabili

Il meta-predicato `fail` – Esempio: iterazione

```
itera :- call(p(X)),
         verifica(q(X)),
         write(X),nl,
         fail.
itera.

verifica(q(X)):- call(q(X)), !.

p(1).
p(2).
p(3).
q(1).
q(3).
```

Stampa:

1

3

Il meta-predicato `fail` – Esempio: iterazione

- Esempio applicativo:

```
itera :- call(votoStud(X,Y)),
         verifica(promosso(Y)),write(X),nl,
         fail.
itera.
verifica(Z):- call(Z), !.

% Base di conoscenza:
promosso(Y) :- Y>= 18.

votoStud(mario, 20).
votoStud(giovanni, 12).
votoStud(maria, 30).
```

Il meta-predicato `fail` – Esempio: negazione

- Si supponga di voler realizzare il meccanismo della negazione per fallimento

  `not(P)` **`(not è \+ in SICStusProlog)`**

  – Vero se P non è derivabile dal programma

- Una possibile realizzazione è la seguente:

```
not(P) :- call(P), !,
          fail.
not(P).
```

Il meta-predicato `fail` – Esempio: negazione

```
p(1).
p(2).
p(3).
q(1).
q(3).

my_not(P) :- call(P), !,
             fail.
my_not(_).

%GOAL
?- my_not(q(2))
Yes.
```

Combinazione cut e fail

- La combinazione **`!,fail`** è interessante ogni qual volta si voglia, all'interno di una delle clausole per una relazione **`p`**, generare un fallimento globale per **`p`** (e non soltanto un backtracking verso altre clausole per **`p`**).

Il meta-predicato `fail` – Combinazione cut e fail

- Consideriamo il problema di voler definire una proprietà **`p`** che vale per tutti gli individui di una data classe tranne alcune eccezioni (**"ragionamento a default"**)

- Esempio: la proprietà "volare" vale per ogni individuo della classe degli uccelli tranne alcune eccezioni (ad esempio, i pinguini o gli struzzi):

```
vola(X)   :-  pinguino(X), !, fail.
vola(X)   :-  struzzo(X), !, fail.
....
vola(X)   :-  uccello(X).
```

I meta-predicati `setof` e `bagof`

- Ogni query `:- p(X).` è interpretata dal Prolog in modo esistenziale; viene cioè proposta una istanza per le variabili di `p` che soddisfa la query

- In alcuni casi può essere interessante poter rispondere a query del secondo ordine, ossia a query del tipo quale è l'insieme S di elementi `X` che soddisfano la query `p(X)` ?

- Molte versioni del Prolog forniscono alcuni predicati predefiniti per query del secondo ordine

I meta-predicati `setof` e `bagof`

- I predicati predefiniti per questo scopo sono

  `setof(X,P,S).`

  S è l'insieme delle istanze **X** che soddisfano il goal **P**

  `bagof(X,P,L).`

  L è la lista delle istanze **X** che soddisfano il goal **P**

  – In entrambi i casi, se non esistono **X** che soddisfano **P** i predicati falliscono

- **bagof** produce una lista in cui possono essere contenute ripetizioni, **setof** produce una lista corrispondente ad un insieme in cui non ci sono ripetizioni…
- … spesso non implementato negli interpreti, per motivi di efficienza

I meta-predicati `setof` e `bagof` – Esempio

- Supponiamo di avere un data base del tipo

```
p(1).
p(2).
p(0).
p(1).
q(2).
r(7).
```

```
:- setof(X,p(X),S).
 yes S = [0,1,2]
        X = X
```

```
:- bagof(X,p(X),S).
 yes S = [1,2,0,1]
        X = X
```

NOTA: la variabile X alla fine della valutazione non è legata a nessun valore. Provate il goal:
?- bagof(X,p(X),S),write(X), nl.

I meta-predicati `setof` e `bagof` – Esempio

- Supponiamo di avere un data base del tipo

```
p(1).
p(2).
p(0).
p(1).
q(2).
r(7).
```

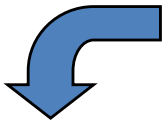
NOTA: Anche il terzo argomento può essere dato in ingresso a un goal

```
:- setof(X,p(X),[0,1,2]).
 yes  X = X

:- bagof(X,p(X),[1,2,0,1]).
 yes X = X

:- bagof(X,p(X),[1,0,2,1]).
false
```

I meta-predicati `setof` e `bagof` – Esempio

- Supponiamo di avere un data base del tipo

```
p(1).
p(2).
p(0).
p(1).
q(2).
r(7).
```



NOTA: Il secondo argomento può essere un goal più complesso (congiunzione)

```
:- setof(X,(p(X),q(X)),S).
 yes S = [2]
      X = X
:- bagof(X,(p(X),q(X)), S).
 yes S = [2]
      X = X
```

I meta-predicati `setof` e `bagof` – Esempio

- Supponiamo di avere un data base del tipo

```
p(1).
p(2).
p(0).
p(1).
q(2).
r(7).
```

```
:- setof(X,(p(X),r(X)),S).
 no                          %false in SWIsh Prolog

:- bagof(X,(p(X),r(X)), S).
 no
```

I meta-predicati `setof` e `bagof` – Esempio

- Supponiamo di avere un data base del tipo

```
p(1).
p(2).
p(0).
p(1).
q(2).
r(7).
```

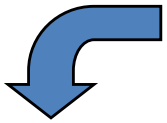
```
:- setof(X, s(X),S).
 no

:- bagof(X, s(X), S).
 no
```

NOTA: questo è il comportamento atteso. In realtà molti interpreti (ad esempio SWI Prolog) danno un errore del tipo:
`calling an undefined procedure s(X)`

I meta-predicati `setof` e `bagof` – Esempio

- Supponiamo di avere un data base del tipo

```
p(1).
p(2).
p(0).
p(1).
q(2).
r(7).
```



NOTA: Il primo argomento può essere un termine complesso (non solo una variabile).

```
:- setof(f(X), p(X), S).
 yes S=[f(0),f(1),f(2)]
     X=X

:- bagof(p(X), p(X), S).
 yes S=[p(1),p(2),p(0),p(1)]
     X=X
```

I meta-predicati `setof` e `bagof` – Esempio

- Supponiamo di avere un database del tipo:

```
padre(giovanni,mario).
padre(giovanni,giuseppe).
padre(mario, paola).
padre(mario,aldo).
padre(giuseppe,maria).
```

NOTA: non fornisce tutti gli **`X`** *per cui* **`padre(X,Y)`** *è vera, ma tutti gli* **`X`** *per cui, per lo stesso valore di* **`Y`**, **`padre(X,Y)`** *è vera.*

```
:-  setof(X, padre(X,Y), S).
  yes      X=X     Y= aldo        S=[mario];
           X=X     Y= giuseppe    S=[giovanni];
           X=X     Y= maria       S=[giuseppe];
           X=X     Y= mario       S=[giovanni];
           X=X     Y= paola       S=[mario];
  no
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esempio – la lista di tutti i padri

- Supponiamo di avere un data base del tipo:

```
padre(giovanni,mario).
padre(giovanni,giuseppe).
padre(mario, paola).
padre(mario,aldo).
padre(giuseppe,maria).
```

```
:-  setof(X, Y^padre(X,Y), S).
  yes       [giovanni, giuseppe, mario]
    X=X
    Y=Y
```

NOTA: per quantificare esistenzialmente Y si può usare questa sintassi

Altro esempio

- Supponiamo di avere un data base del tipo:

```
padre(giovanni,mario).
padre(giovanni,giuseppe).
padre(mario, paola).
padre(mario,aldo).
padre(giuseppe,maria).

:-  setof((X,Y), padre(X,Y), S).
  yes      S=[(giovanni,mario),(giovanni,giuseppe),
        (mario, paola), (mario,aldo),
        (giuseppe,maria)]
    X=X
    Y=Y
```

Il meta-predicato `findall`

- Per ottenere la stessa semantica di **`setof`** e **`bagof`** con quantificazione esistenziale per la variabile non usata nel primo argomento esiste un predicato predefinito

  `findall(X,P,S)`

  vero se **`S`** è la lista delle istanze **`X`** per cui la proprietà **`P`** è vera.

- Se non esiste alcun **`X`** per cui **`P`** è vera **`findall`** non fallisce, ma restituisce una lista vuota (errore in SICStusProlog e SWI Prolog se non esiste il predicato chiamato; lista vuota se esiste, ma non ci sono soluzioni)

Il meta-predicato `findall` – Esempio

- Supponiamo di avere un data base del tipo:

```
padre(giovanni,mario).
padre(giovanni,giuseppe).
padre(mario, paola).
padre(mario,aldo).
padre(giuseppe,maria).

:-  findall(X, padre(X,Y), S).
  yes      S=[giovanni,giovanni,mario,mario,giuseppe]
    X=X
    Y=Y
```

- Equivale a

```
:-  bagof(X, Y^padre(X,Y), S).
```

Non solo fatti: anche le regole con `setof`, `bagof`, `findall`

- I predicati **`setof`**, **`bagof`** e **`findall`** funzionano anche se le proprietà che vanno a controllare non sono definite da fatti, ma da regole.

```
p(X,Y):- q(X),r(X),c(Y).
q(0).
q(1).
r(0).
r(2).
c(1).
c(2).
```

- **Esercizio:** raccogliere le soluzioni X per la chiamata p(X,Y) (per qualsiasi Y)
- **Esercizio:** raccogliere le soluzioni (X,Y) per la chiamata p(X,Y).

Non solo fatti: anche le regole con `setof`, `bagof`, `findall`

```
p(X,Y):- q(X),r(X),c(Y).
q(0).
q(1).
r(0).
r(1).
c(1).
c(2).

:- findall(X, p(X,Y), S).
:- bagof(X, Y^p(X,Y), S).
%soluz. [0,0,1,1]

:- findall((X,Y), p(X,Y), S).
:- bagof((X,Y), p(X,Y), S).
%soluz. [(0,1),(0,2), (1,1), (1,2)]
```

Esempio di implicazione tramite `setof`

In questo esempio **`setof`** viene usato per realizzare un'implicazione.

- Supponiamo di avere predicati del tipo:
  `padre(X,Y)` e **`impiegato(Y)`**
- Si vuole verificare se per ogni **`Y`**:

  `padre(p,Y)` $\Rightarrow$ **`impiegato(Y)`**

  `% tutti i figli di p sono impiegati`

```
implica(Y):- setof(X, padre(Y,X),L),
             verifica(L).
verifica([]).
verifica([H|T]):- impiegato(H),
                  verifica(T).
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esempio di iterazione tramite `setof`

In questo esempio in cui **`setof`** viene usato per realizzare un'iterazione.

- Vogliamo chiamare la procedura **`q`** per ogni elemento per cui vale **`p`**.

```
itera:- setof(X, p(X),L),
        scorri(L).
scorri([]).
scorri([H|T]):- call(q(H)),
                scorri(T).
```

NOTA: nell'iterazione realizzata tramite il fail la procedura ***q*** *doveva produrre effetti non rimovibili dal backtracking. In questo caso invece non è necessario*

Esercizio

Dato un insieme di studenti e i voti ottenuti negli esami, rappresentati come fatti del tipo:

`studente(Nome, Voto).`

si definisca il predicato **`studMedia(Nome,Media)`** che dato il nome di uno studente (**`Nome`**) ne determina la media in **`Media`**.

```
studente(lucia,31).
studente(carlo,18).
studente(carlo,22).
studente(lucia,30).

?-studMedia(carlo,M).
M=20
```

Esercizio – Soluzione

```
studMedia(Nome,Media):-
        findall(Voto,studente(Nome,Voto),List),
        media(List, Media).


%calcola la media su una lista di numeri supposta non vuota
media(Lista,Media) :-
                        length(Lista,LunghezzaLista),
                        sum(Lista,Somma),
                        Media is Somma / LunghezzaLista.


sum([],0).
sum([A|B],N):- sum(B,N1),N is N1+A.
```

Esercizio

- Si definisca un predicato

  `prodotto_cartesiano(L1,L2,L3)`

che, date due liste qualsiasi L1 e L2 restituisca la lista L3 delle coppie ordinate che si possono formare con elementi di L1 e L2.

Per esempio:

```
?-prodotto_cartesiano([a,b,c],[a,d],L3)
L3=[(a,a),(a,d),(b,a),(b,d),(c,a),(c,d)]
```

- Oppure come lista di liste:

```
L3=[[a,a],[a,d],[b,a,],[b,d],[c,a],[c,d]]
```

Esercizio – Soluzione

```
prodotto_cartesiano(L1,L2,L3):-
   findall([A,B], (member(A,L1),member(B,L2)), L3).
```

Esercizio

Data una matrice **M** (come lista di liste) si scriva un un predicato **constant(M,C)** che è vero se **C** è un numero che compare in ciascuna riga della matrice **M.** Esempio:

```
?- constant (
   [[4, 9, 23, 55, 63, 107, 239],
    [5, 9, 31, 55, 60, 73, 82, 99, 107],
    [9, 23, 55, 107, 128, 512],
    [6, 9, 13, 17, 22, 55, 63, 107 ]], 9).
Yes

?- constant (
   [[4, 9, 23, 55, 63, 107, 239],
    [5, 9, 31, 55, 60, 73, 82, 99, 107],
    [9, 23, 55, 107, 128, 512],
    [6, 9, 13, 17, 22, 55, 63, 107 ]], 60).
No
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esercizio – Soluzione

```
constant([], _).
constant([R|Rs], C) :-
                member(C, R),
                constant(Rs, C).

member(H,[H|_]).
member(H,[_|T]) :-
                member(H,T).
```

Esercizio – Soluzione

- La soluzione proposta è invertibile?

```
?-constant(
   [[4, 9, 23, 55, 63, 107, 239],
    [5, 9, 31, 55, 60, 73, 82, 99, 107],
    [9, 23, 55, 107, 128, 512],
    [6, 9, 13, 17, 22, 55, 63, 107 ]], X).
```

Soluzioni:

X = 9;

X= 55;

X = 107;

false

Esercizio

Dato il predicato **`constant/2`**, definire ora un predicato **`constant1(M,L)`** che data la matrice **`M`** restituisce la lista **`L`** dei numeri che compaiono in ciascuna riga della matrice. Esempio:

```
?-constant1(
   [[4, 9, 23, 55, 63, 107, 239],
    [5, 9, 31, 55, 60, 73, 82, 99, 107],
    [9, 23, 55, 107, 128, 512],
    [6, 9, 13, 17, 22, 55, 63, 107 ]], CC).

Yes CC = [ 9, 55, 107 ]
```

Esercizio – Soluzione

```
constant1(M,L):-
    findall(Col, constant(M,Col), L).
```