Liste-Backtracking

ESEMPI CON LE LISTE
PREVENIRE BACKTRACKING USANDO IL CUT,

Liste , operatori aritmetici

Create une lista con dei elementi numerici e una lista vuota?

```
?- L=[1,2,3,4].   ← Richiesta
L = [1, 2, 3, 4]. ← Risposta
```

```
?- L=[].   ← Richiesta
L = [].    ← Risposta
```

Cercate di trovare se il valore 2 e un member della lista L?

```
?- member(2,[1,2,3,4]).   ← Richiesta
true ;                    ← Risposta
false.
```

```
?- member(2,[1,2,3,2]).   ← Richiesta
true ;                    ← Risposta
true.
```

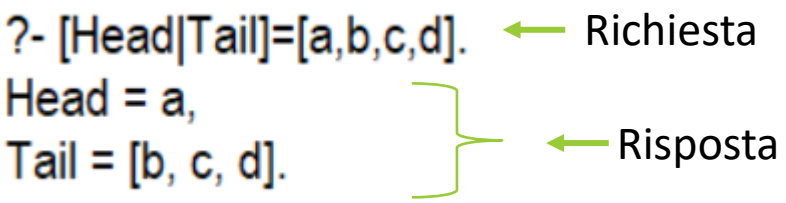
Liste , operatori aritmetici

Quanti elementi contengono le liste definite di sotto?

L1=[link(a,b),link(a,c),link(c,d)]

L2=[a,[b,c],d,[],f(X,Y)]

Che cosa avete capito dalla richiesta scritta sotto questa domanda?



Predicato di Sistema (member)

Member(X,L) %X e member della lista L

Non ha bisogno di scriverlo prima (si riconosce dal Sistema)

```
member( X, [ X | _ ]).          % X appears as head of list

member( X, [ _ | L])  :-
   member( X, L).               % X in tail of list
```

Predicato di Sistema (member)

Che risultati verrano dalle richieste?

member(X,[test,punto(1,2),valore]).

member(valore,[test,valore,punto(1,2),valore]).

member(valore,[test,valore,punto(1,2),val]).

member(2,L).

Predicato di Sistema (concatenate-append)

append(L1,L2,L3). % Concatenate lista L1 con lista L2 creando lista L3.

append([],L,L). %appendere lista vuota con lista L crea di nuovo lista L

append([X|L1],L2,[X|L3]):- append(L1,L2,L3)

?- append([a,b,c],[d,f],L).
L = [a, b, c, d, f].

?- append([a,b,c],[d,e],[a,b,c,d]).
false.

?- append([a,b,c],[d,e],[a,b,c,d,e]).
true.

Predicato Append

Descrivete le situazioni

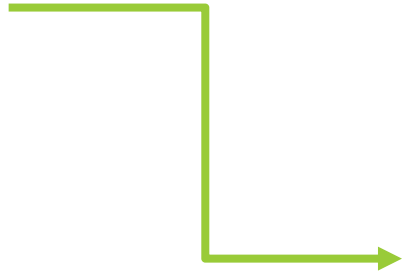
```
?- append(L1,L2,[1,2,3,4]).   ← Richiesta
L1 = [],
L2 = [1, 2, 3, 4] ;
L1 = [1],
L2 = [2, 3, 4] ;
L1 = [1, 2],
L2 = [3, 4] ;
L1 = [1, 2, 3],
L2 = [4] ;
L1 = [1, 2, 3, 4],
L2 = [] ;
false.
```

```
?- append(L1,L2,L3).   ← Richiesta
L1 = [],
L2 = L3 ;
L1 = [_A],
L3 = [_A|L2] ;
L1 = [_A, _B],
L3 = [_A, _B|L2] ;
L1 = [_A, _B, _C],
L3 = [_A, _B, _C|L2] ;
L1 = [_A, _B, _C, _D],
L3 = [_A, _B, _C, _D|L2] ;
L1 = [_A, _B, _C, _D, _E],
L3 = [_A, _B, _C, _D, _E|L2]
```

Predicato Append

Che cosa succedera se scriviamo la richiesta?

```
?- Month=[jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec],
|    append(Before,[may|After],Month).
```



```
Month = [jan, feb, mar, apr, may, jun, jul, aug, sep|...],
Before = [jan, feb, mar, apr],
After = [jun, jul, aug, sep, oct, nov, dec] ;
```

Che risultato verra dalla richiesta?

```
?- Valori=[1,2,3,4],
|    append([],Valori,L3).
```

Predicato Append

Rimuovere elementi di una lista

```
?- L1=[a,b,c,z,d,z,z,z,d,e],append(L2,[z,z,z|_],L1).
L1 = [a, b, c, z, d, z, z, z, d|...],
L2 = [a, b, c, z, d] ;
false.
```

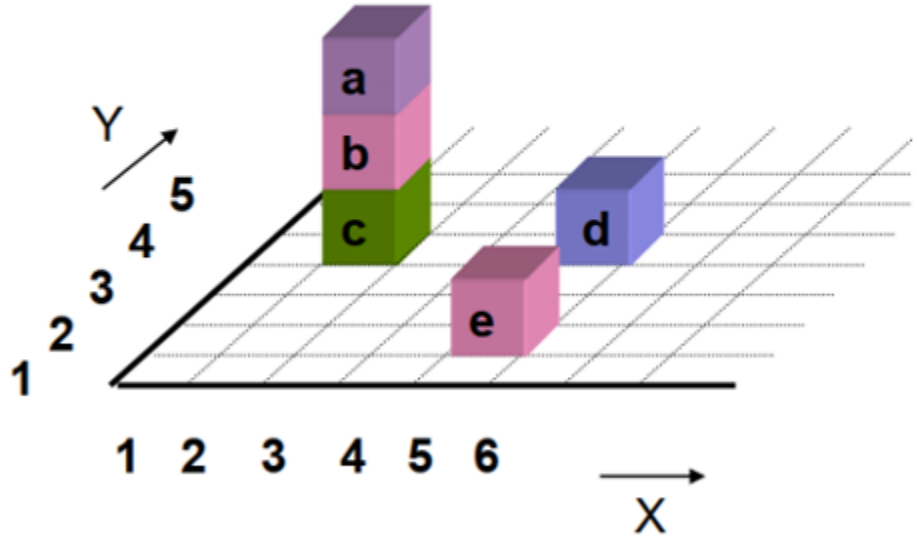
Che risultato avremo dopo
la richiesta ?

```
?- L1=[a,b,c,z,d,z,z,z,d,e],append(L2,[z|X],L1).
```

```
?- L1=[a,b,c,z,d,z,z,z,d,e],append(L2,[z|_],L1).
L1 = [a, b, c, z, d, z, z, z, d|...],
L2 = [a, b, c] ;
L1 = [a, b, c, z, d, z, z, z, d|...],
L2 = [a, b, c, z, d] ;
L1 = [a, b, c, z, d, z, z, z, d|...],
L2 = [a, b, c, z, d, z] ;
L1 = [a, b, c, z, d, z, z, z, d|...],
L2 = [a, b, c, z, d, z, z] ;
false.
```

Progetto (prossima settimana)

Create le clause necessarie
per **identificare il numero dei blocchi**
che sonno sopra un blocco esistente?



Soluzione

```
sopranumero(B,0):-not(on(_,B)).
sopranumero(B,X):-sopra(B1,B),
                  sopranumero(B1,Sop),
                  X is Sop+1.
```

Instantiation & Backtracking

backtracking.pl [modified]

```
male(ergi).
male(erion).
male(fatjon).
male(skerbi).
female(ilma).
female(ezgi).
female(suida).


parent(ilma,ergi).
parent(erion,ergi).
parent(ilma,skerb).
parent(erion,skerb).
parent(ilma,ezgi).
parent(erion,ezgi).
parent(fatjon,ezra).
parent(suida,ezra).

father(X,Y):-parent(X,Y),male(X).
```

Goal | [trace] ?- female(X).

False
False
False
False
True → X=ilma
True → X=ezgi
True → X=suida

X instantiated

Ilma;
Ezgi;
Suida.

output

Instantiation & Backtracking

backtracking.pl [modified]

```
male(ergi).
male(erion).
male(fatjon).
male(skerbi).
female(ilma).
female(ezgi).
female(suida).


parent(ilma,ergi).
parent(erion,ergi).
parent(ilma,skerb).
parent(erion,skerb).
parent(ilma,ezgi).
parent(erion,ezgi).
parent(fatjon,ezra).
parent(suida,ezra).

father(X,Y):-parent(X,Y),male(X).
```

Goal : [trace] ?- father(X,skerbi).

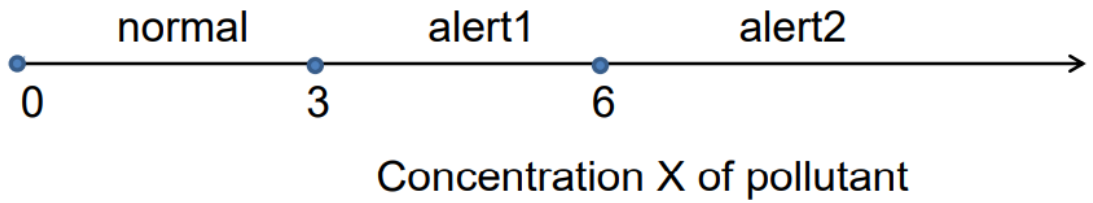
Parent(X,skerbi)-> call
parent(ilma,skerbi)->true
male(ilma)->false
Backtracking last moment true
Parent(erion,skerbi).
Male(erion)->true
Tutti I due predicate sono true allora
Si vede il risultato finale
X=erion.

True → Y=skerbi

X=erion.

- Prolog in modo automatic fara backtrack se e necessario per soddisfare una goal.
- I backtracking non controllati possono essere non efficienti in un program
- ‘Cut’ si puo usare come controllo o prevenzione del backtracking

Esempio-Allarme dell'aria contaminata



- *Rule 1*: if $X < 3$ then Y = normal
- *Rule 2*: if $3 \leq X$ and $X < 6$ then Y = alert1
- *Rule 3*: if $6 \leq X$ then Y = alert2

f(Concentration, State_of_alert) –Versione 1

```
f( X, normal)   :-   X < 3.                % Rule 1
f( X, alert1)   :-   3 =< X, X < 6.        % Rule 2
f( X, alert2)   :-   6 =< X.               % Rule 3
```

- *Rule 1*: if X < 3 then Y = normal
- *Rule 2*: if 3 ≤ X and X < 6 then Y = alert1
- *Rule 3*: if 6 ≤ X then Y = alert2

Esperimento Nr 1

?- f(2, Y), Y = alert1.

No

- Analizzate la trace dove noterete che il backtracking accade quando non ce piu bisogno (non fa senso che accadi)?

Versione 2

f(X, normal) :- X < 3, !.

f(X, alert1) :- 3 =< X, X < 6, !.

f(X, alert2) :- 6 =< X.

- "!" si legge come"cut" perche interrompe le alternative
- Il Cut previene I backtracking che sono inutili
- La seconda versione e molto piu efficace dalla prima versione
- Il cuts non affettano la loggica.

Esperimento Nr 2

?- f(7, Y).

Y = alert2

Analizate l’execuzione della trace, Prolog di nuovo fa dei lavori non necessari

Versione 3

f(X, normal) :- X < 3, !.

f(X, alert1) :- X < 6, !.

f(X, alert2).

- Questa e la versione migliore
- Ma la loggica in questo caso e cambiata.

Prova questo:

?- f(2, alert1).

yes % Non e quello che volevamo!

- Perche Prolog risponde con “yes”?

- Una formulazione piu correta della richiesta sarebbe:

?- f(2, Y), Y = alert1.

no

Esempio

Dobbiamo trovare il Max tra due valori, X e Y

```
max( X, Y, X) :- X >= Y.

max( X, Y, Y) :- X < Y.

                        % More efficient with cut

max( X, Y, X) :- X >= Y, !.

max( X, Y, Y).

                        % But note again!!!

?- max( 3, 1, 1).

yes                     % Not as intended!
```

Formulazione piu correta

```
max( X, Y, Max) :- X >= Y, !, Max = X;

Max = Y.

?- max( 3, 1, 1).

no                                        % As intended
```

Esercizio 1(prossima settimana)

Se abbiamo il programma

p(1).
p(2)-:!.
p(3).

Indovinate le risposte per le richieste successive.

a) ?-p(X).

b)?-p(X),p(Y).

c)?-p(X),!,p(Y).

Esercizio 2(prossima settimana)

La relazione classifica i numeri a tre classi:positive,zero e negative.

class(Number,positive):-Number>0.

class(0,zero).

class(Number,negative):-Number<0.

Create una forma piu efficiente usando il cut.

Esercizio 3 (prossima settimana)

Definite la procedura (usando Cut)

split(Numbers,Positive,Negative)

che divide una lista di numeri in due liste: I valori positive (includendo zero) e I valori negative.

Per esempio:

split([3,-1,0,5,-2],[3,0,5],[-1,-2]).

Domande?