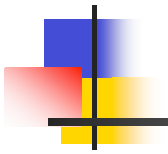
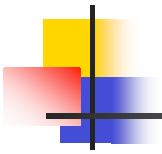


Il predicato ! (cut)

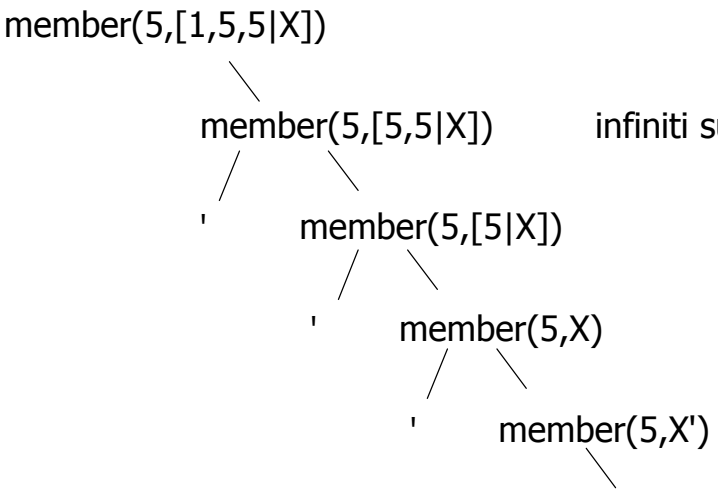


Problemi di efficienza

- Liste differenza e tecnica dell'accumulo ci permettono di evitare inefficienze generate dalla scomposizione e ricomposizione di liste, permettendo di "preparare" il risultato in una struttura ausiliaria (accumulatore) durante la fase di scomposizione ed utilizzando l'unificazione per il calcolo finale.
- Questa non è l'unica fonte di inefficienza, altre gravi inefficienze sono legate alla struttura dell'albero di ricerca (albero SLD) per un goal G. Questo può infatti contenere molti cammini ripetuti o vicoli ciechi.

esempio 1

```
member(X, [X| _ ]).
member(X, [ _ |L]) :- member(X,L).
?- member(5,[1,5,5|X]).
```



infiniti successi tutti uguali

esempio 2

```
confronta(X,Y,minore) :- X<Y.
confronta(X,Y,uguale) :- X=:=Y.
confronta(X,Y,maggiore) :- X>Y.
?- confronta(10,20,X), X=maggiore.
```

Notiamo che

- il goal fallisce (vedi diapositiva successiva)
- accorgendosi che le tre clausole sono mutamente esclusive si sarebbe potuta evitare la ricerca del successo dopo il primo fallimento.

esempio 2

confronta(10,20,X), X=:=maggiore

10<20, minore =:= maggiore
minore =:= maggiore
fail

10=:=20, uguale=:=maggiore
fail

10>20, maggiore=:=maggiore
fail

Il cut

- Il problema con gli esempi precedenti deriva dalla strategia di ricerca dell'interprete Prolog che cerca sempre di ispezionare TUTTO l'albero SLD.
- Per alterare tale strategia viene offerto al programmatore il predicato cut (!) che può essere inserito nel corpo di una qualsiasi clausola.
- L'esecuzione del predicato cut ha l'effetto di "potare" l'albero SLD, tagliando i rami ridondanti o morti secondo le regole seguenti.

esecuzione di ! (cut)

Sia !, G1 un goal che ha il cut (!) come atomo più a sinistra e sia c: H :- **A**, !, **B**. l'istanza della clausola che ha introdotto tale occorrenza del cut.

L'esecuzione del cut rende definitiva ogni scelta fatta nel risolvere l'atomo che unificava con H, ovvero la scelta della clausola c e le scelte fatte per risolvere **A**.

esempio

p(s1) :- **B1**.

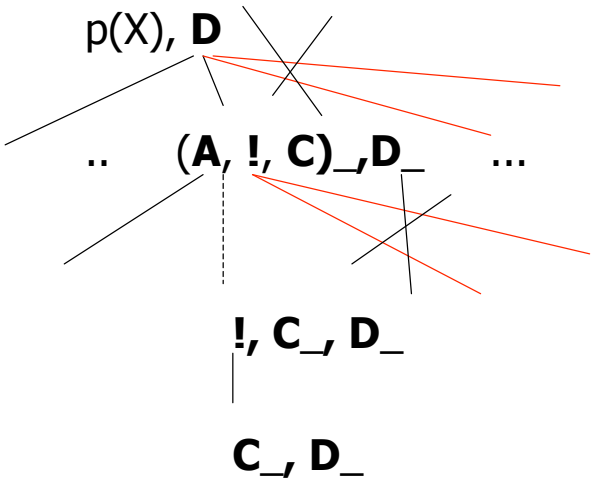
...

p(si):- **A**, !, **C**.

...

p(sn) :- **Bn**

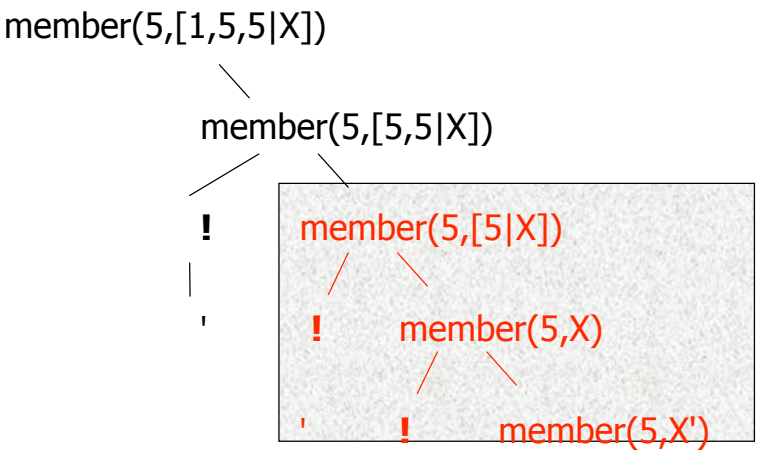
G = p(X), **D**



La parte di albero cancellata non verrà più ispezionata, anche se il goal C_, D_ fallisce.

esempio: member con cut

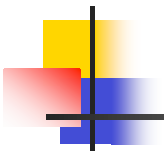
```
member(X, [X|_]) :- !.
member(X,[-|L]) :- member(X,L).
```



esempio: confronta con cut

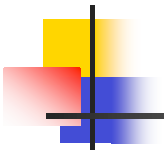
```
confronta(X,Y,minore) :- X<Y, !.
confronta(X,Y,uguale) :- X=:=Y, !.
confronta(X,Y,maggiore) :- X>Y.
```

```
confronta(10,20,X), X=maggiore.
confronta(10,20,X), !, X=maggiore.
!, minore=maggiore.
minore=maggiore.
fail
```



cut verdi e cut rossi

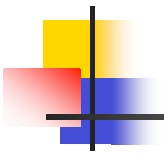
- CUT VERDE. Negli esempi precedenti l'introduzione del cut non ha alterato la corrispondenza tra semantica operazionale e semantica dichiarativa: stesso insieme delle soluzioni con e senza esecuzione del cut.
- CUT ROSSO. Questo non è sempre vero: l'esecuzione del cut può alterare il significato del programma cambiando l'insieme delle soluzioni.



cut rosso: esempio

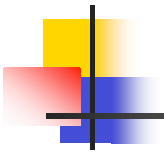
```
red_confronta(X,Y,minore) :- X<Y, !.
red_confronta(X,Y,uguale) :- X=:=Y, !.
red_confronta(X,Y,maggiore).
```

Questo programma produce lo stesso insieme di risposte di quello visto in precedenza (che aveva il controllo X>Y nel corpo dell'ultima clausola) ma la semantica dichiarativa dei due programmi NON è la stessa.



Il predicato fail

Il predicato fail è un altro predicato predefinito che, contrariamente al cut, quando viene eseguito genera un fallimento.

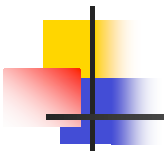


la sequenza !, fail

```
not_member(X, []).
not_member(X, [X|_]):- !,fail.
not_member(X,[-|L]) :- not_member(X,L).
```

Quando X unifica con la testa della lista [X|-] viene prima eseguito il cut che taglia tutte le scelte successive e poi il fail che fa abortire la computazione (effetto desiderato).

E' un modo per realizzare la negazione (… vedremo).

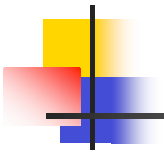


Esempio: set

Vogliamo definire un predicato

set(L,S) dove S ed L contengono gli stessi elementi ma S non ha duplicati.

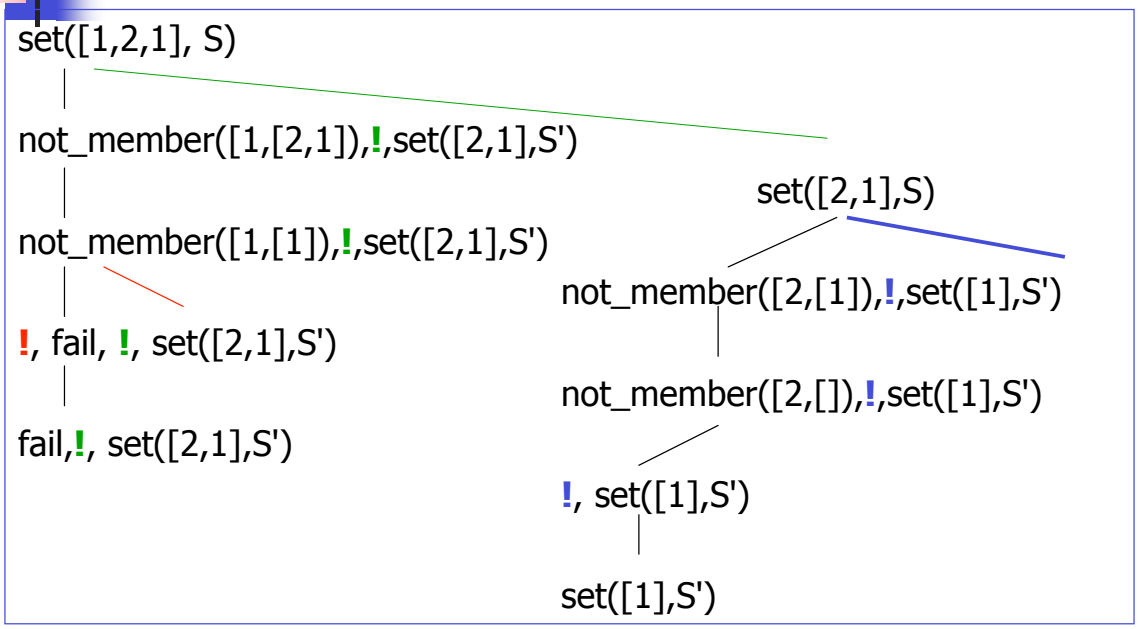
```
set([],[]).
set([X|L],[X|S]) :- not_member(X,L),!,set(L,S).
set([X|L], S) :- set(L,S).
```



associazione cut-taglio

- In un albero SLD possono occorrere più cut, ciascuno (se eseguito) può generare delle potature, è importante una giusta associazione.

esempio



Esercizio

- Definire un predicato flatten(L,L') dove L è una lista di liste ed L' è la sua versione "appiattita",
  es. L=[[a,b],[c],[[d,e],f]] L=[a,b,c,d,e]

```
flatten([X|L],F) :-  flatten(X,F1), flatten(L,F2),
                     append(F1,F2,F).
flatten(X,[X]) :- constant(X), X =!=[].
flatten([],[]).
```

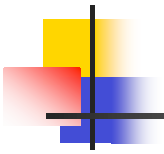
flatten_dl

Sostituiamo ogni lista risultato con una lista differenza

```
flatten_dl([X|L],F-Z) :-  flatten_dl(X,F1-Y),
                          flatten_dl(L,F2-W), append_dl(F1-Y,F2-W,F-Z).
flatten_dl(X,[X|Z]-Z) :- constant(X), X =!=[].
flatten_dl([],Z-Z).
```

Risolvendo append nella prima clausola Y=F2, W=Z e F1=F:

```
flatten_dl([X|L],F-Z) :- flatten_dl(X,F-F2), flatten_dl(L,F2-Z).
flatten_dl(X,[X|Z]-Z) :- constant(X), X =!=[].
flatten_dl([],Z-Z).
```



Esercizio

- Definire il tipo di dato coda con le operazioni: enqueue e dequeue.