

# CONTROLLO DI UN PROGRAMMA

---

- Predicati predefiniti che consentono di influenzare e controllare il processo di esecuzione (dimostrazione) di un goal.
- PREDICATO CUT (!)
  - E' denotato dal simbolo !
  - E' un predicato extra-logico, che risulta sempre un po' complesso da comprendere ...
- Per capire come funziona il predicato cut è necessario visualizzare l'albero SLD esplorato
- Nel seguito si fa vedere anche il modello *run time* di Prolog

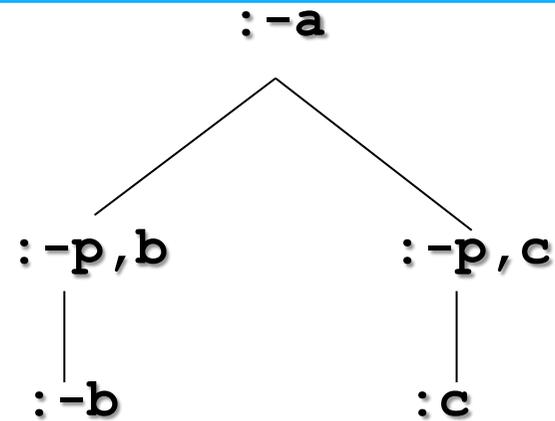
# CONTROLLO DI UN PROGRAMMA

---

(c11)  $a :- p, b.$

(c12)  $a :- p, c.$

(c13)  $p.$



# CONTROLLO DI UN PROGRAMMA

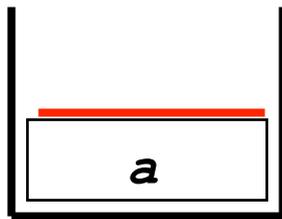
---

(c11)       $a :- p, b.$

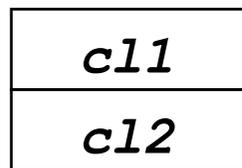
(c12)       $a :- p, c.$

(c13)       $p.$

- E la valutazione della query  $:-a.$



*Stack di esecuzione*



*Scelte per **a***



*Scelta corrente*

# CONTROLLO DI UN PROGRAMMA

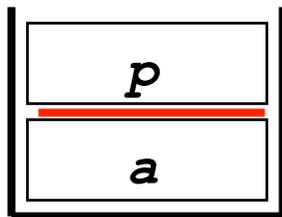
---

(c11)       $a :- p, b.$

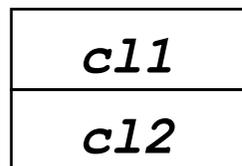
(c12)       $a :- p, c.$

(c13)       $p.$

- E la valutazione della query  $:-a.$



*Stack di esecuzione*



*Scelte per **a***

*Scelta corrente*

La valutazione di  $p$  ha successo

# CONTROLLO DI UN PROGRAMMA

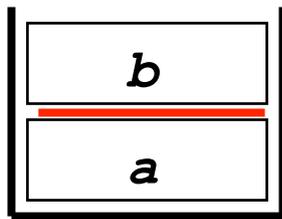
---

(c11)       $a :- p, b.$

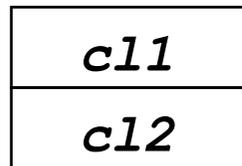
(c12)       $a :- p, c.$

(c13)       $p.$

- E la valutazione della query  $:-a.$



*Stack di esecuzione*



*Scelte per **a***

*Scelta corrente*

La valutazione di  $b$  fallisce  $\Rightarrow$  viene attivato il meccanismo di backtracking

# CONTROLLO DI UN PROGRAMMA

---

(c11)       $a :- p, b.$

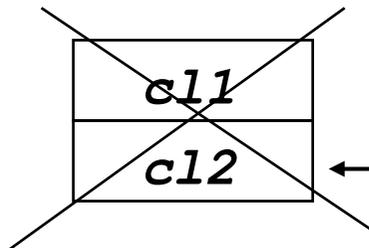
(c12)       $a :- p, c.$

(c13)       $p.$

- E la valutazione della query  $:-a.$



*Stack di esecuzione*



*Scelte per **a***

*Scelta corrente*

# CONTROLLO DI UN PROGRAMMA

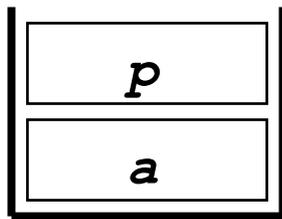
---

(c11)       $a :- p, b.$

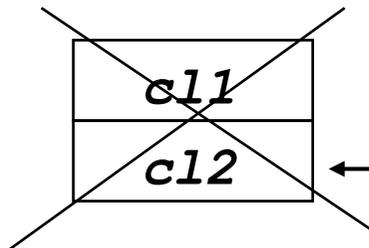
(c12)       $a :- p, c.$

(c13)       $p.$

- E la valutazione della query  $:-a.$



*Stack di esecuzione*



*Scelte per **a***

← *Scelta corrente*

La valutazione di  $p$  ha successo

# CONTROLLO DI UN PROGRAMMA

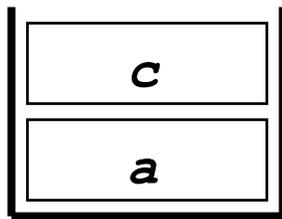
---

(c11)       $a :- p, b.$

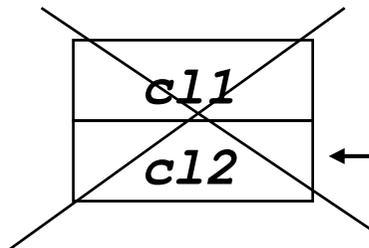
(c12)       $a :- p, c.$

(c13)       $p.$

- E la valutazione della query  $:-a.$



*Stack di esecuzione*



*Scelte per **a***

← *Scelta corrente*

La valutazione di  $c$  fallisce  $\Rightarrow$  viene attivato il meccanismo di backtracking ma non ci sono più punti di scelta. Quindi si ha il fallimento di  $a$

# MODELLO RUN-TIME PROLOG

---

- Due stack:
  - Stack di esecuzione che contiene i record di attivazione (***Environment***) delle varie procedure
  - Stack di backtracking che contiene l'insieme dei punti di scelta (***Choice-point***). Ad ogni fase della valutazione tale stack contiene puntatori alle scelte aperte nelle fasi precedenti della dimostrazione.

(In realtà è utilizzato un solo stack, con alternanza di *environment* e *choice point*; le linee rosse indicate rappresentano i *choice point*)

# CONTROLLO DI UN PROGRAMMA

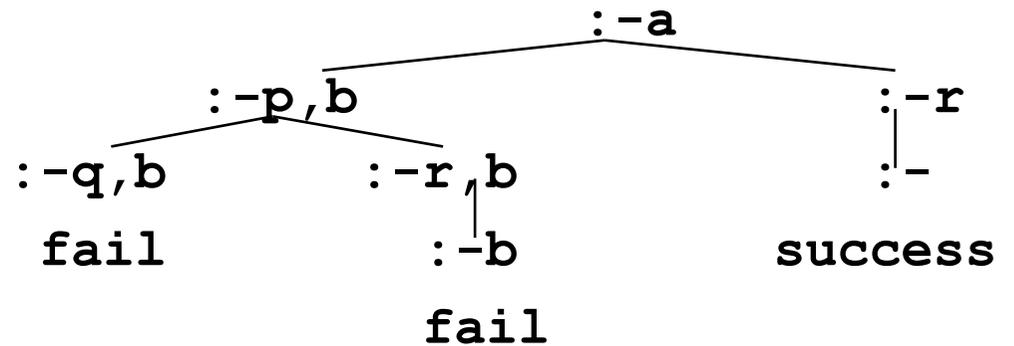
(c11) a :- p,b.

(c12) a :- r.

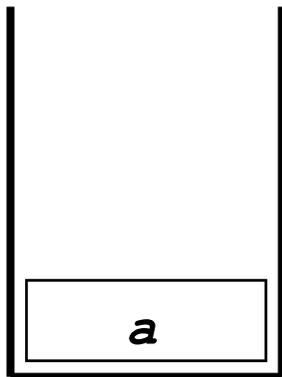
(c13) p :- q.

(c14) p :- r.

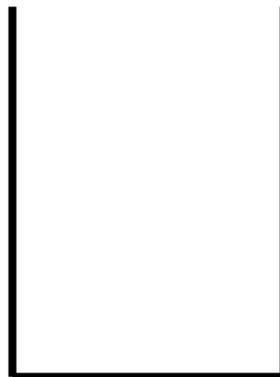
(c15) r.



- E la valutazione della query :-a.



*Stack di esecuzione*



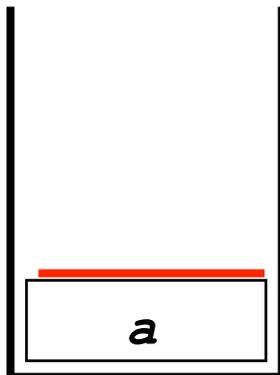
*Stack di backtracking*

# CONTROLLO DI UN PROGRAMMA

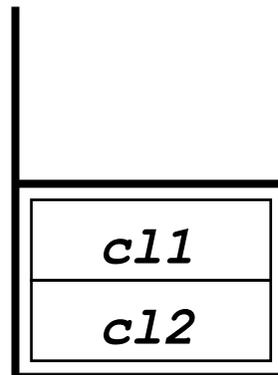
---

(c11)      a :- p,b.  
(c12)      a :- r.  
(c13)      p :- q.  
(c14)      p :- r.  
(c15)      r.

- E la valutazione della query :-a.



*Stack di esecuzione*



*Stack di backtracking*

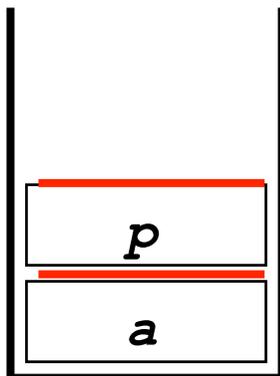
← *Scelta corrente*

# CONTROLLO DI UN PROGRAMMA

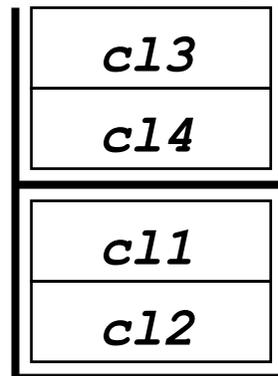
---

(c11)      a :- p,b.  
(c12)      a :- r.  
(c13)      p :- q.  
(c14)      p :- r.  
(c15)      r.

- E la valutazione della query :-a.



*Stack di esecuzione*



*Stack di backtracking*

← *Scelta corrente*

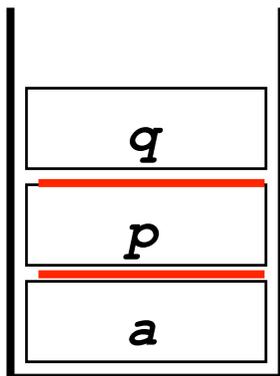
← *Scelta corrente*

# CONTROLLO DI UN PROGRAMMA

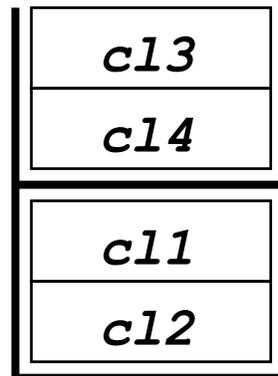
---

(c11)      a :- p, b.  
(c12)      a :- r.  
(c13)      p :- q.  
(c14)      p :- r.  
(c15)      r.

- E la valutazione della query :-a.



*Stack di esecuzione*



*Stack di backtracking*

← *Scelta corrente*

← *Scelta corrente*

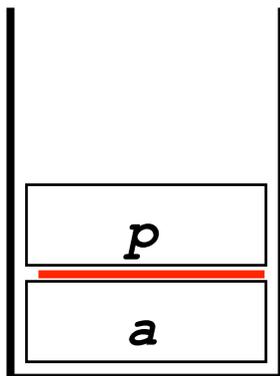
Fallimento

# CONTROLLO DI UN PROGRAMMA

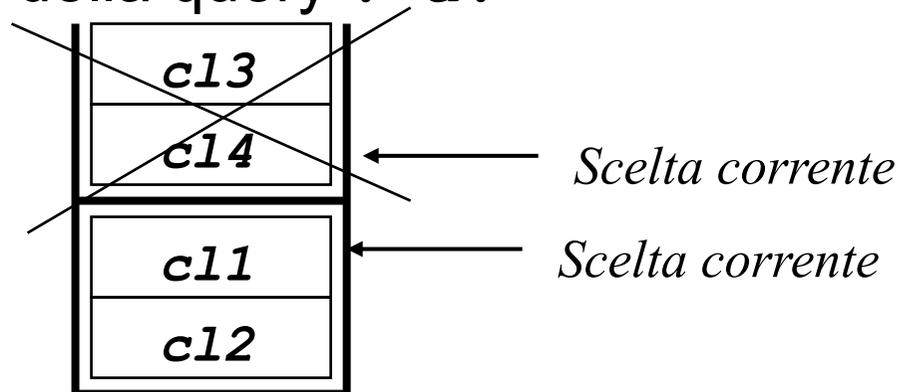
---

(c11)      a :- p, b.  
(c12)      a :- r.  
(c13)      p :- q.  
(c14)      p :- r.  
(c15)      r.

- E la valutazione della query :-a.



*Stack di esecuzione*



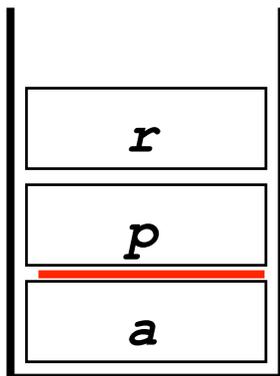
*Stack di backtracking*

# CONTROLLO DI UN PROGRAMMA

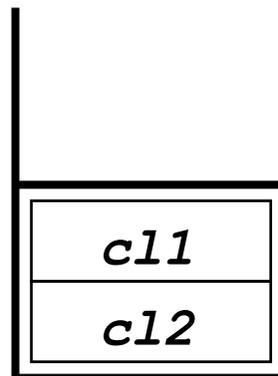
---

(c11)      **a** :- p,b.  
(c12)      **a** :- r.  
(c13)      **p** :- q.  
(c14)      **p** :- r.  
(c15)      **r**.

- E la valutazione della query :-a.



*Stack di esecuzione*



*Stack di backtracking*

*Scelta corrente*

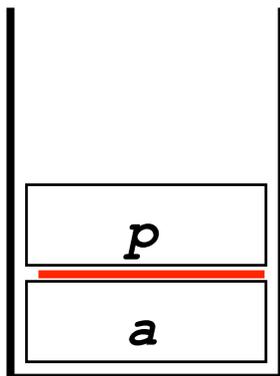
**r** ha successo

# CONTROLLO DI UN PROGRAMMA

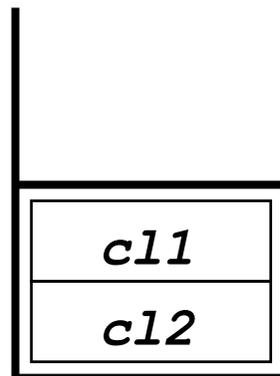
---

(c11)      a :- p, b.  
(c12)      a :- r.  
(c13)      p :- q.  
(c14)      p :- r.  
(c15)      r.

- E la valutazione della query :-a.



*Stack di esecuzione*



*Stack di backtracking*

*Scelta corrente*

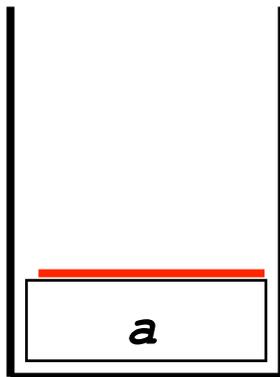
**p** ha successo

# CONTROLLO DI UN PROGRAMMA

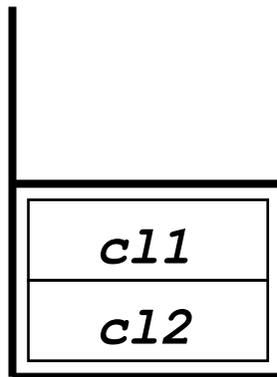
---

(c11)      a :- p, b.  
(c12)      a :- r.  
(c13)      p :- q.  
(c14)      p :- r.  
(c15)      r.

- E la valutazione della query :-a.



*Stack di esecuzione*



*Stack di backtracking*

*Scelta corrente*

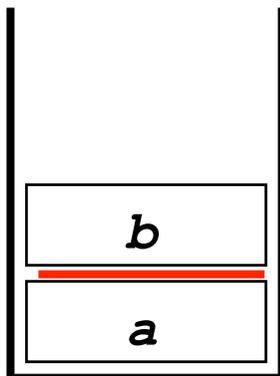
Si continua con **b**

# CONTROLLO DI UN PROGRAMMA

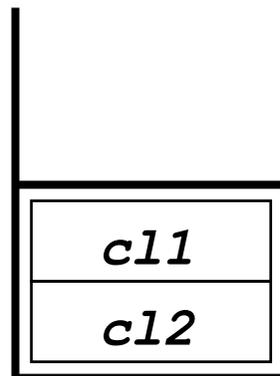
---

(c11)      a :- p, b.  
(c12)      a :- r.  
(c13)      p :- q.  
(c14)      p :- r.  
(c15)      r.

- E la valutazione della query :-a.



*Stack di esecuzione*



*Stack di backtracking*

*Scelta corrente*

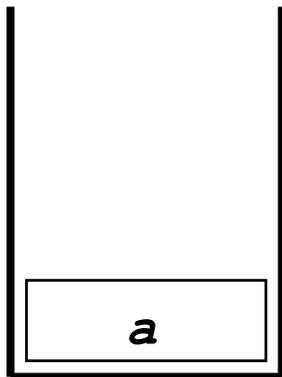
**b** fallisce

# CONTROLLO DI UN PROGRAMMA

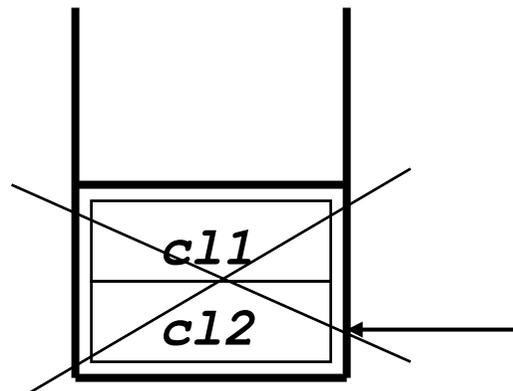
---

(c11)      a :- p, b.  
(c12)      a :- r.  
(c13)      p :- q.  
(c14)      p :- r.  
(c15)      r.

- E la valutazione della query :-a.



*Stack di esecuzione*



*Stack di backtracking*

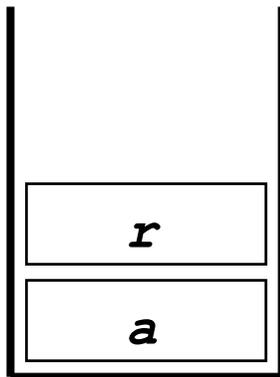
*Scelta corrente*

# CONTROLLO DI UN PROGRAMMA

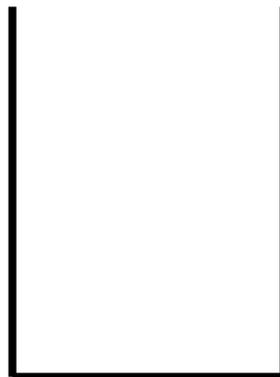
---

(c11)      **a** :- **p**,**b**.  
(c12)      **a** :- **r**.  
(c13)      **p** :- **q**.  
(c14)      **p** :- **r**.  
(c15)      **r**.

- E la valutazione della query :-**a**.



*Stack di esecuzione*



*Stack di backtracking*

**r** ha successo

# CONTROLLO DI UN PROGRAMMA

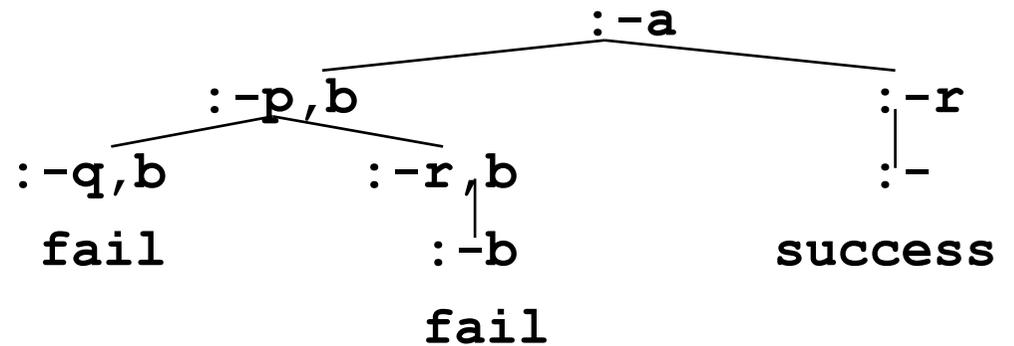
(c11) a :- p,b.

(c12) a :- r.

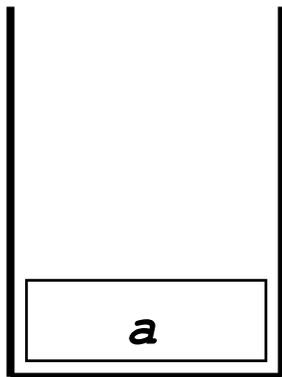
(c13) p :- q.

(c14) p :- r.

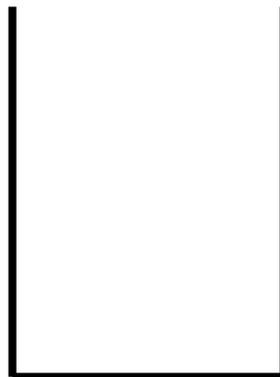
(c15) r.



- E la valutazione della query :-a.



*Stack di esecuzione*



*Stack di backtracking*

**Successo**

## IL PREDICATO CUT (!)

---

- L'effetto del cut è quello di rendere definitive alcune scelte fatte nel corso della valutazione dall'interprete Prolog ossia quello di **eliminare alcuni blocchi (choice point) dallo stack di backtracking**
- Il cut altera quindi il controllo del programma
- Effetto collaterale più importante: **perdita di dichiaratività**

## EFFETTO DEL CUT

---

- Si consideri la clausola:

$$p \text{ :- } q_1, q_2, \dots, q_i, !, q_{i+1}, q_{i+2}, \dots, q_n.$$

l'effetto della valutazione del goal ! (cut) durante la dimostrazione del goal "p" è il seguente:

- la valutazione di ! ha successo (come quasi tutti i predicati predefiniti) e ! viene ignorato in fase di backtracking;
- tutte le scelte fatte nella valutazione dei goal  $q_1, q_2, \dots, q_i$  e in quella del goal  $p$  vengono rese definitive; in altri termini, tutti i punti di scelta per tali goal (per le istanze di tali goal utilizzate) vengono rimossi dallo stack di backtracking.
- Le alternative riguardanti i goal seguenti al cut non vengono modificate

# EFFETTO DEL CUT

---

- Si consideri la clausola:

$$p \text{ :- } q_1, q_2, \dots, q_i, \text{ !, } q_{i+1}, q_{i+2}, \dots, q_n.$$

- Se la valutazione di  $q_{i+1}, q_{i+2}, \dots, q_n$  fallisce, fallisce tutta la valutazione di  $p$ . Infatti, anche se  $p$  o  $q_1, q_2, \dots, q_i$  avessero punti di scelta questi sarebbero eliminati dal cut;
- **Il cut taglia rami dell' albero SLD**

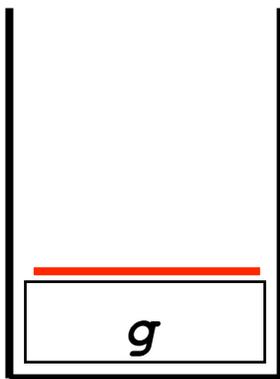
Pertanto il cut non può essere definito in modo dichiarativo (può tagliare eventuali soluzioni)

# EFFETTO DEL CUT

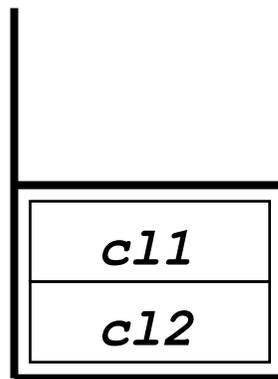
---

(c11)      g :- a.  
(c12)      g :- s.  
(c13)      a :- p,!,b.  
(c14)      a :- r.  
(c15)      p :- q.  
(c16)      p :- r.  
(c17)      r.

- E la valutazione della query :-g.



*Stack di esecuzione*



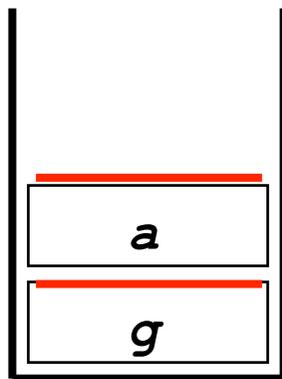
*Stack di backtracking*

*Scelta corrente*

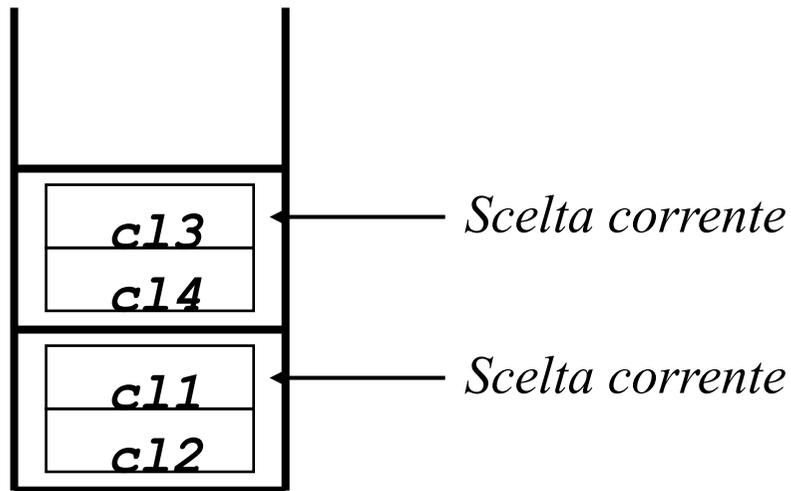
# EFFETTO DEL CUT

```
(c11)      g :- a.  
(c12)      g :- s.  
(c13)      a :- p,!,b.  
(c14)      a :- r.  
(c15)      p :- q.  
(c16)      p :- r.  
(c17)      r.
```

- E la valutazione della query :-g.



*Stack di esecuzione*

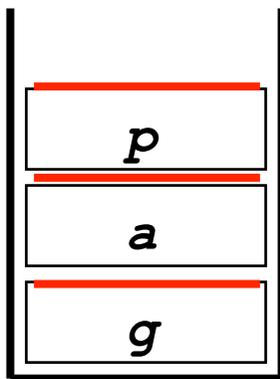


*Stack di backtracking*

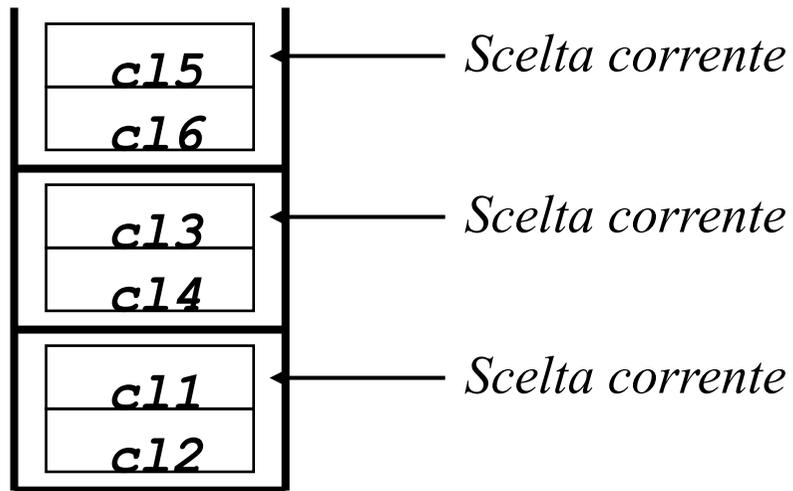
# EFFETTO DEL CUT

(c11)      g :- a.  
(c12)      g :- s.  
(c13)      a :- p,!,b.  
(c14)      a :- r.  
(c15)      p :- q.  
(c16)      p :- r.  
(c17)      r.

- E la valutazione della query :-g.



*Stack di esecuzione*

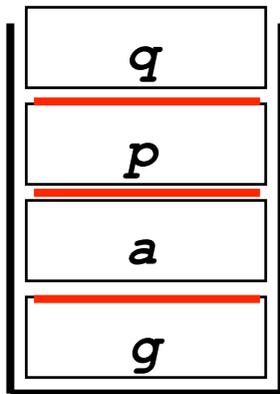


*Stack di backtracking*

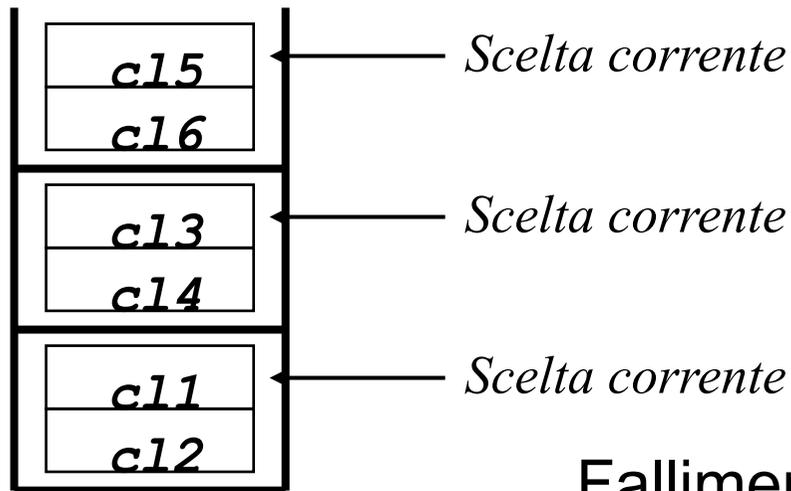
# EFFETTO DEL CUT

(c11)      g :- a.  
(c12)      g :- s.  
(c13)      a :- p,!,b.  
(c14)      a :- r.  
(c15)      p :- q.  
(c16)      p :- r.  
(c17)      r.

- E la valutazione della query :-g.



*Stack di esecuzione*



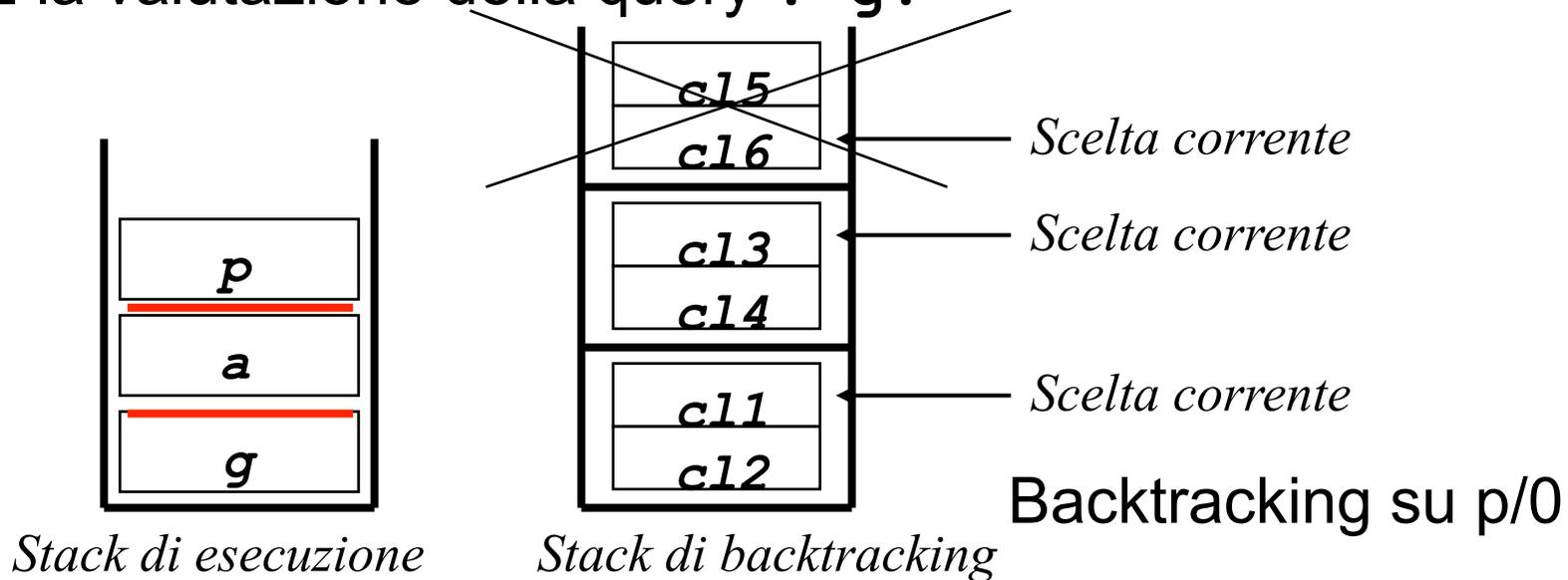
*Stack di backtracking*

Fallimento

# EFFETTO DEL CUT

(c11)      g :- a.  
(c12)      g :- s.  
(c13)      a :- p,!,b.  
(c14)      a :- r.  
(c15)      p :- q.  
(c16)      p :- r.  
(c17)      r.

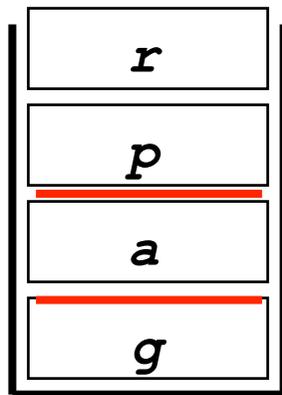
- E la valutazione della query :-g.



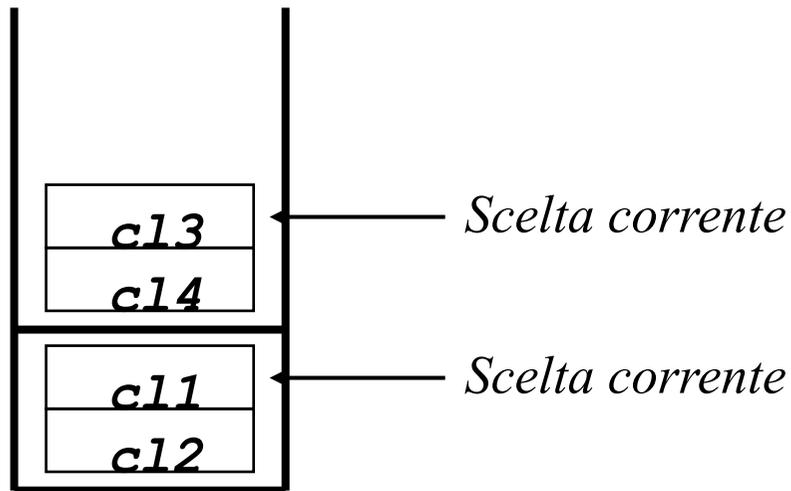
# EFFETTO DEL CUT

```
(c11)      g :- a.  
(c12)      g :- s.  
(c13)      a :- p,!,b.  
(c14)      a :- r.  
(c15)      p :- q.  
(c16)      p :- r.  
(c17)      r.
```

- E la valutazione della query `:-g.`



*Stack di esecuzione*

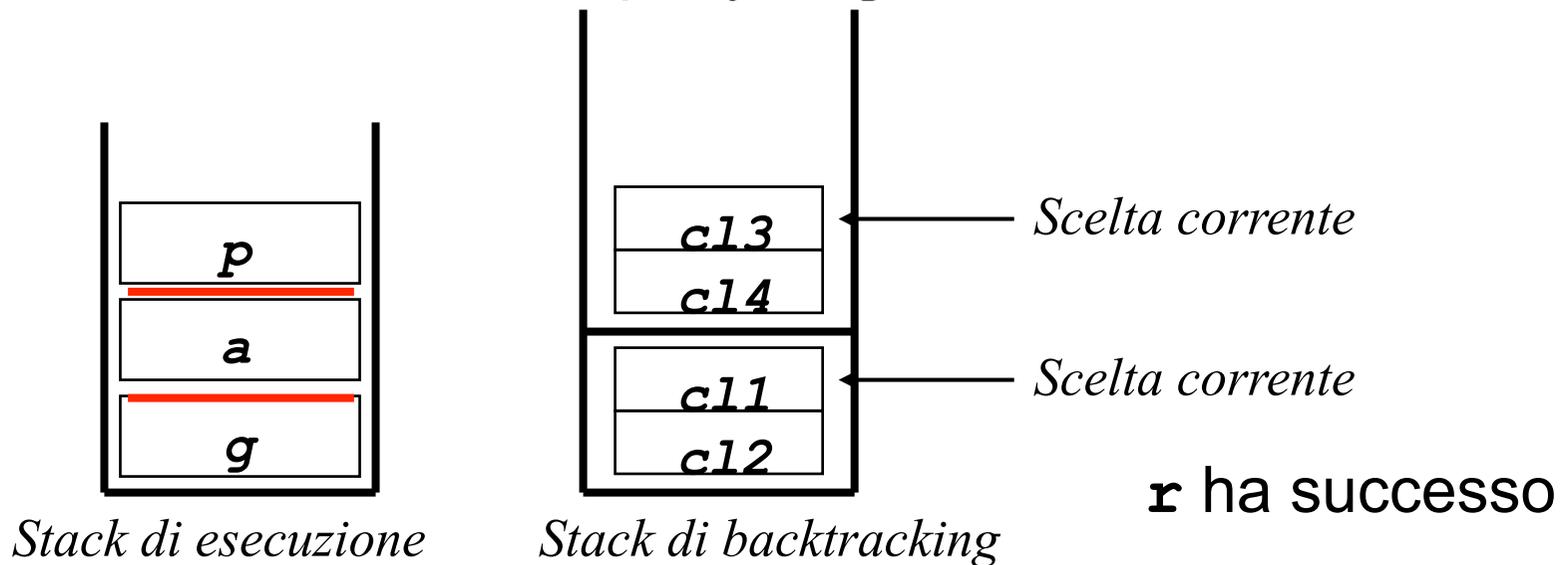


*Stack di backtracking*

# EFFETTO DEL CUT

```
(c11)    g :- a.  
(c12)    g :- s.  
(c13)    a :- p,!,b.  
(c14)    a :- r.  
(c15)    p :- q.  
(c16)    p :- r.  
(c17)    r.
```

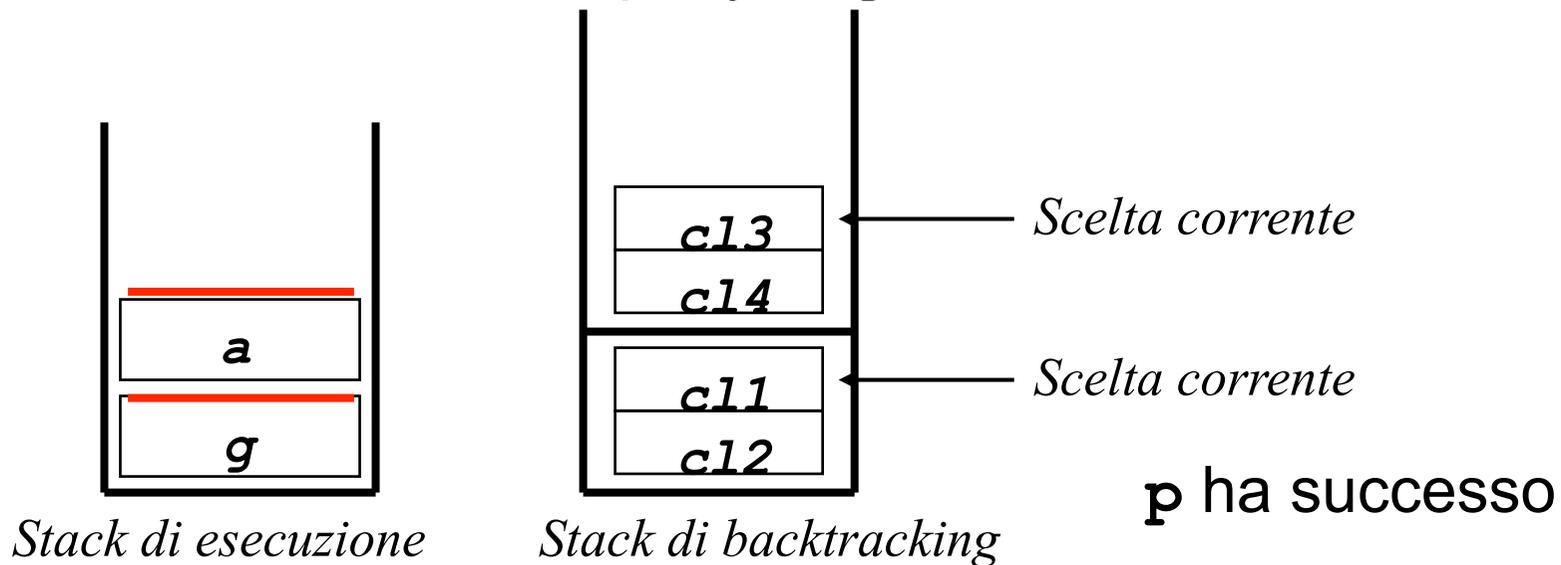
- E la valutazione della query `:-g.`



# EFFETTO DEL CUT

```
(c11)      g :- a.  
(c12)      g :- s.  
(c13)      a :- p,!,b.  
(c14)      a :- r.  
(c15)      p :- q.  
(c16)      p :- r.  
(c17)      r.
```

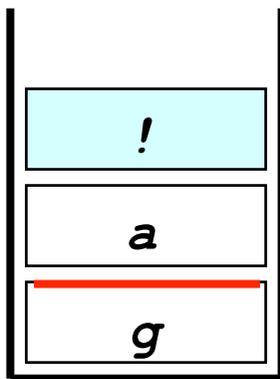
- E la valutazione della query `:-g.`



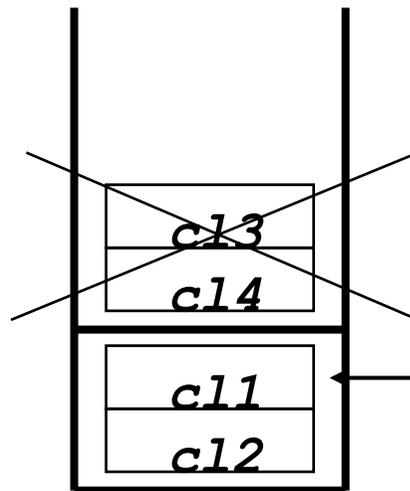
# EFFETTO DEL CUT

(c11)      g :- a.  
(c12)      g :- s.  
(c13)      a :- p,!,o.  
(c14)      a :- r.  
(c15)      p :- q.  
(c16)      p :- r.  
(c17)      r.

- E la valutazione della query :-g.



*Stack di esecuzione*



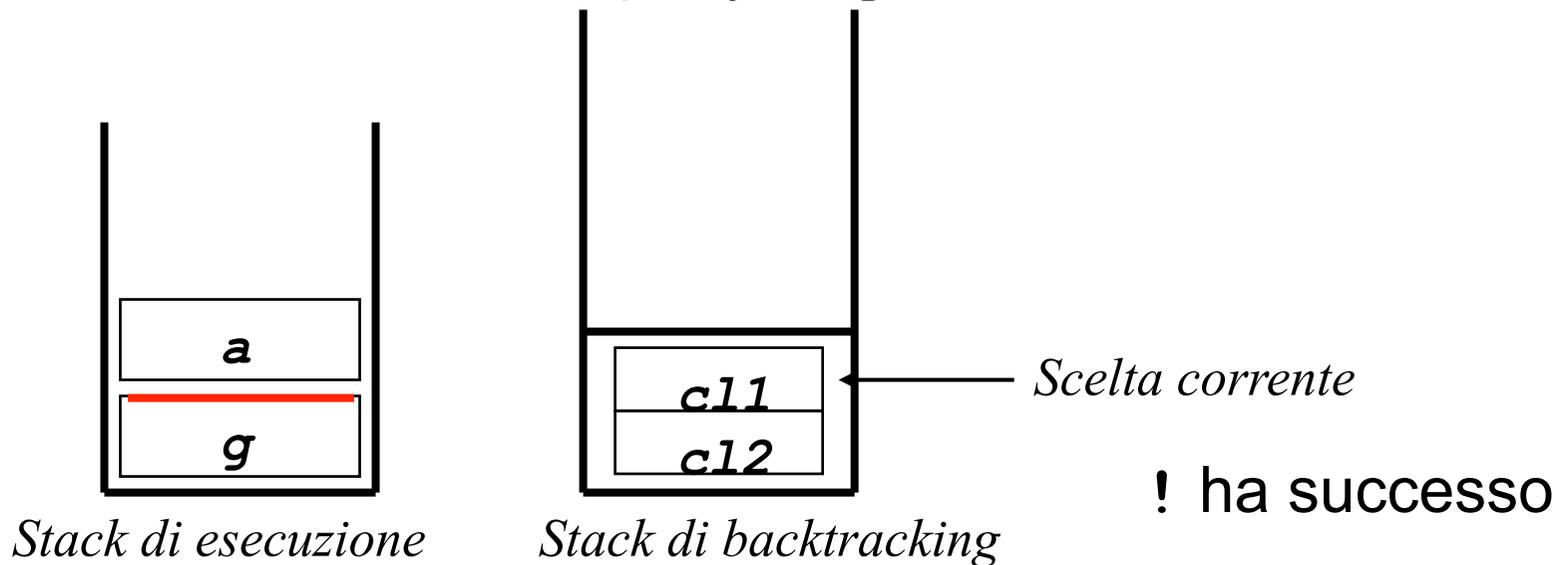
*Stack di backtracking*

**Effetto del !**  
Tutti i punti di scelta per **p** (se ce ne fossero ancora allocati) e per **a** sono rimossi dallo stack  
Il cut ha successo

# EFFETTO DEL CUT

```
(c11)      g :- a.  
(c12)      g :- s.  
(c13)      a :- p,!,b.  
(c14)      a :- r.  
(c15)      p :- q.  
(c16)      p :- r.  
(c17)      r.
```

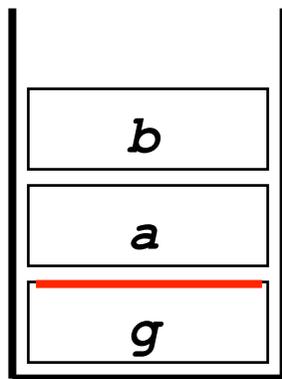
- E la valutazione della query `:-g.`



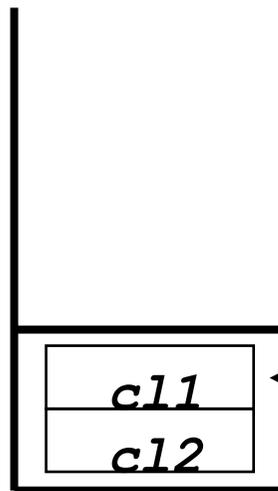
# EFFETTO DEL CUT

```
(c11)      g :- a.  
(c12)      g :- s.  
(c13)      a :- p,!,b.  
(c14)      a :- r.  
(c15)      p :- q.  
(c16)      p :- r.  
(c17)      r.
```

- E la valutazione della query `:-g.`



*Stack di esecuzione*



*Stack di backtracking*

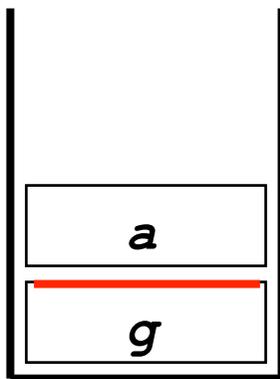
*Scelta corrente*

**b** fallisce e fallisce  
quindi anche **a**

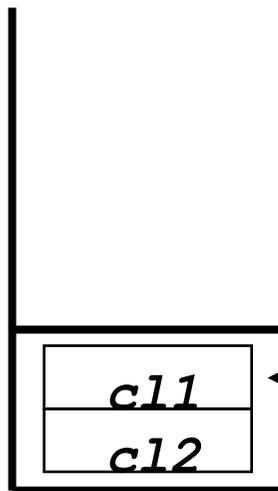
# EFFETTO DEL CUT

```
(c11)      g :- a.  
(c12)      g :- s.  
(c13)      a :- p,!,b.  
(c14)      a :- r.  
(c15)      p :- q.  
(c16)      p :- r.  
(c17)      r.
```

- E la valutazione della query `:-g`.



*Stack di esecuzione*



*Stack di backtracking*

**b** fallisce e fallisce  
quindi anche **a**

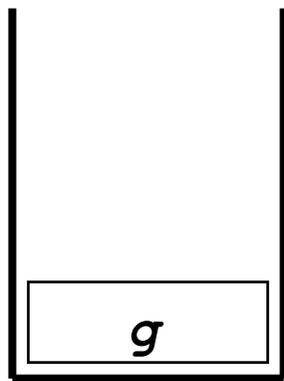
*Scelta corrente*

# EFFETTO DEL CUT

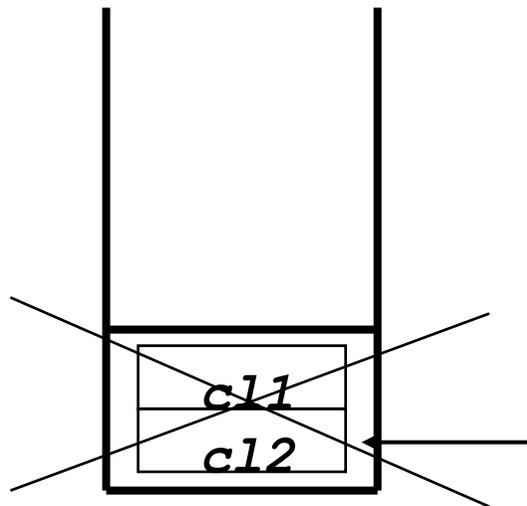
---

```
(c11)      g :- a.  
(c12)      g :- s.  
(c13)      a :- p,!,b.  
(c14)      a :- r.  
(c15)      p :- q.  
(c16)      p :- r.  
(c17)      r.
```

- E la valutazione della query `:-g.`



*Stack di esecuzione*



*Stack di backtracking*

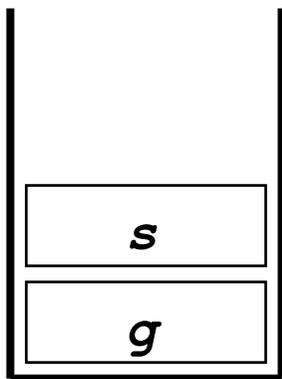
*Scelta corrente*

# EFFETTO DEL CUT

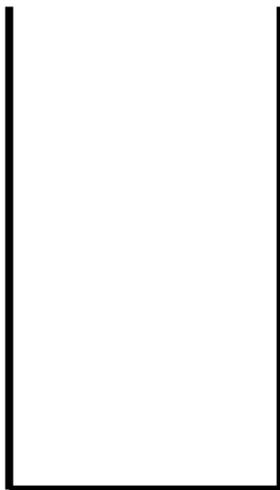
---

```
(c11)      g :- a.  
(c12)      g :- s.  
(c13)      a :- p,!,b.  
(c14)      a :- r.  
(c15)      p :- q.  
(c16)      p :- r.  
(c17)      r.
```

- E la valutazione della query `:-g.`



*Stack di esecuzione*



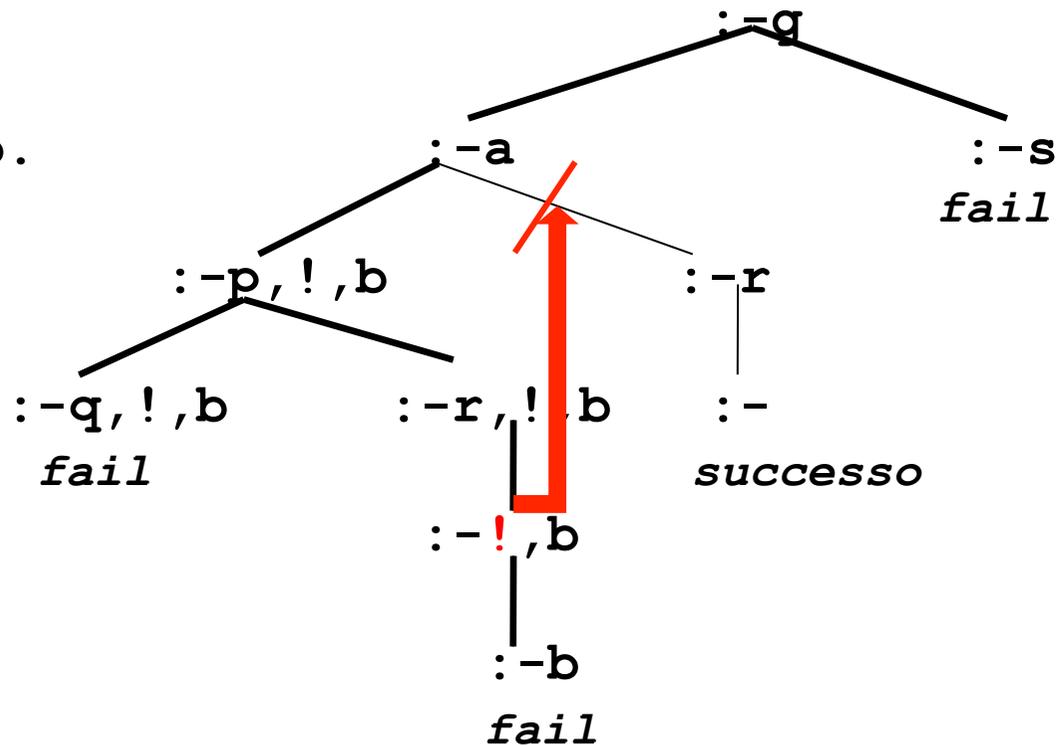
*Stack di backtracking*

Fallimento !!!

Senza il cut la query  
avrebbe avuto  
successo

# EFFETTO DEL CUT: ALBERO SLD

(c11) g :- a.  
(c12) g :- s.  
(c13) a :- p,!,b.  
(c14) a :- r.  
(c15) p :- q.  
(c16) p :- r.  
(c17) r.



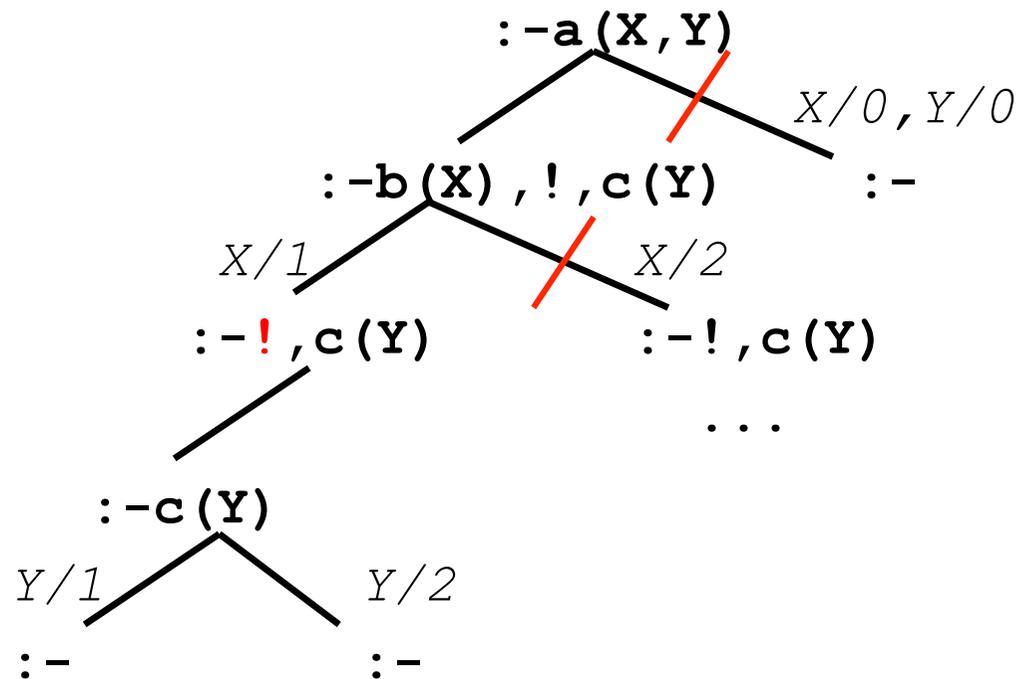
In backtracking, restano aperti solo i punti di scelta per g

Si noti che si perde la completezza (taglio un ramo di successo)

# ESEMPIO

$a(X, Y) :- b(X), !, c(Y) .$   
 $a(0, 0) .$   
 $b(1) .$   
 $b(2) .$   
 $c(1) .$   
 $c(2) .$

$:- a(X, Y) .$   
 yes     $X=1$      $Y=1 ;$   
            $X=1$      $Y=2 ;$   
 no



# EFFICIENZA

---

- Abbiamo visto il predicato **member**

```
member(E1, [E1|_]).
```

```
member(E1, [X|Tail):- E1\==X, member(E1,Tail).
```

- Se abbiamo bisogno di interpretare tale predicato solo per la verifica di appartenenza di un elemento a una lista, possiamo inserire un cut per migliorare l'efficienza

```
member(E1, [E1|_]) :- !.
```

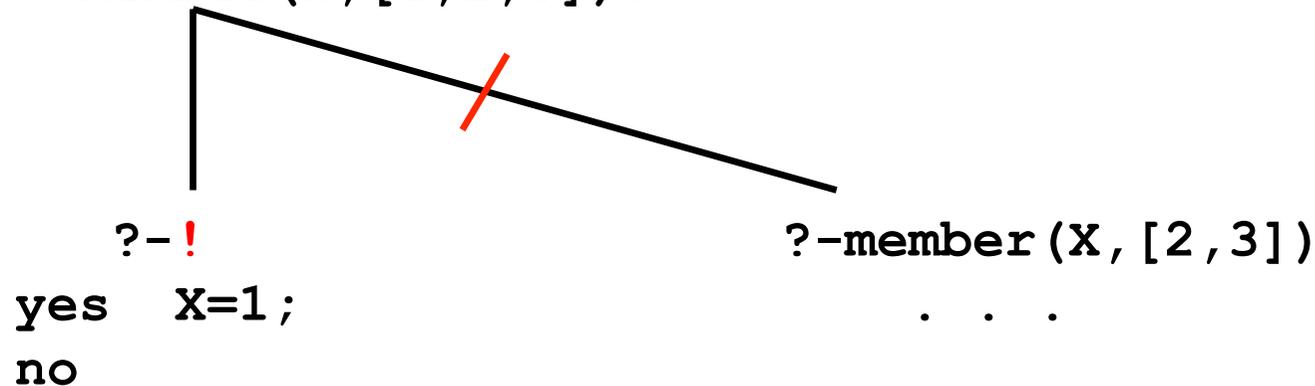
```
member(E1, [_|Tail):- member(E1,Tail).
```

- In questo caso però non è possibile usare il predicato **member** per
  - individuare tutti gli elementi di una lista
  - verificare l'appartenenza multipla di un elemento alla lista

# EFFICIENZA MEMBER

```
member(E1, [E1|_]) :- !.  
member(E1, [_|Tail]) :- member(E1, Tail) .
```

```
?-member(X, [1,2,3]) .
```



- Restituisce solo la prima soluzione (l'altro ramo alternativo nell'albero SLD è stato tagliato dall'applicazione del cut e con esso tutte le derivazioni di successo sottostanti)

# CUT

---

- La perdita della dichiaratività è il maggiore svantaggio derivante dall'uso del "cut".
- Tuttavia l'uso del "cut" è necessario per la correttezza di alcune classi di programmi ed è utile per l'efficienza di altre classi di programmi.

# MUTUA ESCLUSIONE TRA CLAUSOLE

---

- Il cut può essere utilizzato molto semplicemente per rendere deterministica la scelta tra due o più clausole alternative

`p(X) :- a(X), b.`

`p(X) :- c.`

- Si supponga che la condizione `a(X)` debba rendere le due clausole mutuamente esclusive per realizzare uno schema del tipo:

`if a(.) then b else c`

# MUTUA ESCLUSIONE TRA CLAUSOLE

---

- Si supponga che la condizione  $a(X)$  debba rendere le due clausole mutuamente esclusive per realizzare uno schema del tipo:

`if a(.) then b else c`

- Utilizzando il predicato predefinito "cut":

`p(X) :- a(X), !, b.`

`p(X) :- c.`

**ATTENZIONE:** la mancanza del cut rende il programma **SCORRETTO**

*Se  $a(X)$  e' vera, viene valutato il cut che toglie il punto di scelta per  $p(X)$ . Se invece  $a(X)$  fallisce, si innesca il backtracking prima che il cut venga eseguito.*

## MUTUA ESCLUSIONE TRA CLAUSOLE

- Predicato che, dati un termine T e una lista L, conta le “occorrenze” di T in L (*Esercizio 2.2 f*): ?-conta(1,[2,3,1]).

```
conta (_, [], 0) .
conta (T, [T|R], N) :- conta (T, R, N1) ,
                        N is N1+1 .
conta (T, [H|R], N) :- T \== H ,
                        conta (T, R, N) .
```

- ?-conta(a,[b,a],N).

```
conta (_, [], 0) .
conta (T, [T|R], N) :- ! , conta (T, R, N1) ,
                        N is N1+1 , .
conta (T, [_|R], N) :- conta (T, R, N) .
```

## MUTUA ESCLUSIONE TRA CLAUSOLE

- Predicato che, dati un termine T e una lista L, conta le “occorrenze” di T in L (*Esercizio 2.2 f*): ?-conta(1,[2,3,1]).

```
conta (_, [], 0) .
conta (T, [H|R], N) :- T=H, conta (T, R, N1) ,
                        N is N1+1.
conta (T, [H|R], N) :- T\==H,
                        conta (T, R, N) .
```

- ?-conta(a,[b,a],N).

```
conta (_, [], 0) .
conta (T, [T|R], N) :- !, conta (T, R, N1) ,
                        N is N1+1, .
conta (T, [_|R], N) :- conta (T, R, N) .
```

## RIMOZIONE DI ELEMENTI DA LISTE

---

- Cancellazione di un elemento uguale a T dalla lista

```
(c11) delete1(T, [], []).
```

```
(c12) delete1(T, [T|TAIL], TAIL).
```

```
(c13) delete1(T, [HEAD|TAIL], [HEAD|L]) :-  
      T \== HEAD, delete1(T, TAIL, L).
```

- Cancellazione di tutti gli elementi uguali a T dalla lista

```
(c14) delete(T, [], []).
```

```
(c15) delete(T, [T | TAIL], L) :-  
      delete(T, TAIL, L).
```

```
(c16) delete(T, [HEAD|TAIL], [HEAD|L]) :-  
      T \== HEAD, delete(T, TAIL, L).
```

## RIMOZIONE DI ELEMENTI DA LISTE

---

- Cancellazione di un elemento uguale a T dalla lista: le clausole (c12) e (c13) devono essere mutuamente esclusive.
- La condizione di mutua esclusione e' l' unificazione dell' elemento da cancellare con la testa della lista

```
(c11) delete1(T, [], []).
```

```
(c12') delete1(T, [T|TAIL], TAIL) :- !.
```

```
(c13) delete1(T, [HEAD|TAIL], [HEAD|L]) :-  
      delete1(T, TAIL, L).
```

## RIMOZIONE DI ELEMENTI DA LISTE

---

- Cancellazione di tutti gli elementi uguali a T dalla lista: le clausole (c15) e (c16) devono essere mutuamente esclusive.
- La condizione di mutua esclusione e' l' unificazione dell' elemento da cancellare con la testa della lista

```
(c14)    delete(T, [], []).
```

```
(c15)    delete(T, [T | TAIL], L) :- !,  
          delete(T, TAIL, L).
```

```
(c16)    delete(T, [HEAD | TAIL], [HEAD | L]) :-  
          delete(T, TAIL, L).
```

# EFFICIENZA

---

- La presenza del "cut" rende in molti casi un programma ricorsivo deterministico e consente l'applicazione dell'ottimizzazione della ricorsione tail.
- Merge di due liste ordinate: ?-merge([1,2,5],[1,3,4],L). Yes  
L=[1,1,2,3,4,5]

```
merge([], L2, L2).
merge(L1, [], L1).
merge([X|REST1], [Y|REST2], [X,Y|REST]) :-
    X=Y, merge(REST1, REST2, REST).
merge([X|REST1], [Y|REST2], [X|REST]) :-
    X < Y, merge(REST1, [Y|REST2], REST).
merge([X|REST1], [Y|REST2], [Y|REST]) :-
    X > Y, merge([X|REST1], REST2, REST).
```

- Sebbene merge sia definita in modo **tail ricorsivo**, la presenza dei punti di scelta rende l'ottimizzazione della tail recursion impossibile

## EFFICIENZA

---

?-merge([1,2],[1,3],L)      Yes L=[1,1,2,3]

- Inserendo il cut il programma cambia così:

```
merge([], L2, L2) :-!.
merge(L1, [], L1) :-!.
merge([X|REST1],[X|REST2],[X,X|REST]) :- !,
                                         merge(REST1,REST2,REST) .
merge([X|REST1],[Y|REST2],[X|REST]) :-
    X < Y, !, merge(REST1,[Y|REST2],REST) .
merge([X|REST1],[Y|REST2],[Y|REST]) :-
    merge([X|REST1],REST2,REST) .
```

## ESEMPIO: UNIONE DI INSIEMI

---

- Gli insiemi possono essere rappresentati come liste di oggetti (senza ripetizioni)
- Unione di due insiemi

`union(S1,S2,S3)` "l'insieme S3 contiene gli elementi appartenenti all'unione di S1 e S2"

```
union([], S2, S2).  
union([X|REST], S2, S) :- member(X, S2),  
                           union(REST, S2, S).  
union([X|REST], S2, Y) :- union(REST, S2, S), Y=[X|S].
```

- Il predicato `union` in backtracking ha un comportamento scorretto. Infatti, anche in questo caso non c'è mutua esclusione tra la seconda e la terza clausola.

## ESEMPIO: UNIONE DI INSIEMI (*cont.*)

---

```
union([], S2, S2).  
union([X|REST], S2, S) :- member(X, S2),  
                           union(REST, S2, S).  
union([X|REST], S2, [X|S]) :- union(REST, S2, S).
```

```
?- union([1,2], [1,3], Z).
```

```
Z=[2,1,3] ;
```

```
Z=[1,2,1,3] ;
```

```
no
```

- Il predicato `union` in backtracking ha un comportamento scorretto. Infatti, anche in questo caso non c'è mutua esclusione tra la seconda e la terza clausola.

## ESEMPIO: UNIONE DI INSIEMI (*cont.*)

---

- Unione di due insiemi

`union(S1,S2,S3)` "l'insieme S3 contiene gli elementi appartenenti all'unione di S1 e S2"

```
union([], S2, S2).  
union([X|REST], S2, S) :- member(X, S2), !,  
                           union(REST, S2, S).  
union([X|REST], S2, [X|S]) :- union(REST, S2, S).
```

?- `union([1,2], [1,3], Z)`.

`Z=[2,1,3]` ;

no

- Con il cut, il predicato `union` in backtracking ha un comportamento corretto. Infatti, in questo caso c'è mutua esclusione tra la seconda e la terza clausola.

# *HomeWork: TO DO!*

---

- *Esercizio compito 11 Sett 2008 – con !*
- *Predicato intersection*
  
- *Gioco Tic-Tac-Toe*
- *Cripto-aritmetica*

## *Esercizio compito 11 Sett 2008*

---

- Si scriva un programma Prolog `no_dupl (Xs, Ys)` che è vero se `Ys` è la lista (senza duplicazioni) degli elementi che compaiono nella lista `Xs`. Nella lista `Ys` gli elementi compaiono nello stesso ordine di `Xs` ed, in caso di elementi duplicati, si manterrà l'ultima occorrenza.

- Esempi:

```
?-no_dupl ([a,b,a,d] , [b,a,d] ) .
```

```
yes
```

```
?-no_dupl ([a,b,a,c,d,b,e] , L) .
```

```
yes  L=[a,c,d,b,e]
```

## *Esercizio compito 11 Sett 2008*

---

```
no_dupl([], []).
```

```
no_dupl([X|Xs], Ys):-  
    member(X, Xs),  
    no_dupl(Xs, Ys).
```

```
no_dupl([X|Xs], [X|Ys]):-  
    nonmember(X, Xs),  
    no_dupl(Xs, Ys).
```

```
nonmember(_, []).
```

```
nonmember(X, [Y|Ys]):-X \== Y,  
    nonmember(X, Ys).
```

## *Esercizio 11 Sett 2008 – vers.2*

---

```
no_dupl([], []) :- !.
```

```
no_dupl([X|Xs], Ys) :-  
    member(X, Xs), !,  
    no_dupl(Xs, Ys).
```

```
no_dupl([X|Xs], [X|Ys]) :-  
    no_dupl(Xs, Ys).
```

## ESEMPIO: INTERSEZIONE DI INSIEMI

---

- Consideriamo ora l'intersezione di due insiemi

`intersection(S1,S2,S3)` "l'insieme S3 contiene gli elementi appartenenti all'intersezione di S1 e S2"

```
intersection([],S2,[]).
intersection([H|T],S2,[H|T3]):- member(H,S2),
                                intersection(T,S2,T3).
intersection([H|T],S2,S3):- intersection(T,S2,S3).
```

- La seconda e la terza clausola **devono essere mutuamente esclusive**

```
:- intersection([1,2,3], [2,3,4], S).
```

```
yes    S=[2,3];
```

```
S=[2];
```

```
S=[3];
```

```
S=[]
```

*Risposte scorrette a causa della non mutua esclusione tra la seconda e la terza clausola*

## ESEMPIO: INTERSEZIONE DI INSIEMI

---

- La condizione che determina la mutua esclusione è `member(H, S2)` quindi il cut va inserito dopo tale condizione

```
intersection([], S2, []).
intersection([H|T], S2, [H|T3]) :- member(H, S2), !,
                                   intersection(T, S2, T3).
intersection([H|T], S2, S3) :- intersection(T, S2, S3).
```

- La seconda e la terza clausola sono **mutuamente esclusive**

```
:- intersection([1,2,3], [2,3,4], S).
   yes      S=[2,3];
```

# *Esercizi*

---

■ *Albero SLDNF per goal:*

`:-intersection([1,2,3], [2,3,4], S).`

*Nelle due versioni del programma*

`:-intersection([1,2,3],[2,3,4],S)`

---

`S=[1|S']`  
`:-member(1,[2,3,4,]),!`  
`intersection([2,3],[2,3,4],S')`  
`:-intersection([2,3],[2,3,4],S')`  
`S=S'`

*fail*

`S'=[2|S'']`  
`:-member(2,[2,3,4]),!`  
`intersection([3],[2,3,4],S'')`  
...

`:-!,intersection([3],[2,3,4],S'')`

`S''=[3|S''']`  
`:-member(3,[2,3,4]),!`  
`intersection([], [2,3,4],S''')`  
...

`:-!,intersection([], [2,3,4],S''')`

`:- S'''=[]`                      *yes* `S=[2,3]`

# *Esercizi*

---

## ■ *Albero SLDNF per goal:*

```
:-nodup1([1,2,1,2,1],Y)
```

```
nodup1([], []).
```

```
nodup1([X|Xs], Ys):-
```

```
    member(X, Xs),!,
```

```
    nodup1(Xs, Ys).
```

```
nodup1([X|Xs], [X|Ys]):-
```

```
    nodup1(Xs, Ys).
```