ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Prolog – Cut

Federico Chesani

DISI

Department of Informatics – Science and Engineering

Disclaimer & Further Reading

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

Controllo di un Programma

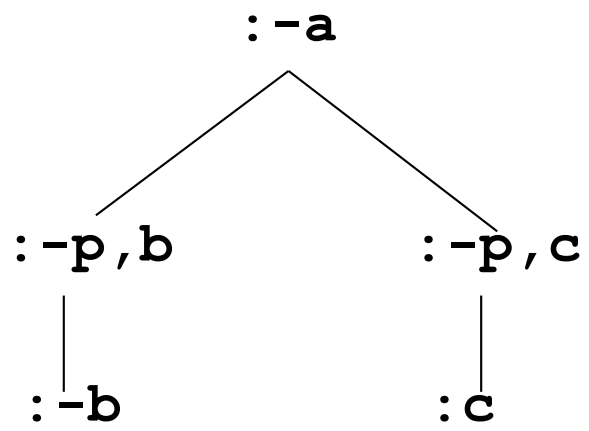
In Prolog sono stati aggiunti dei predicati predefiniti che consentono di influenzare e controllare il processo di esecuzione (dimostrazione) di un goal.

- predicato **CUT (!)**
  - E' denotato dal simbolo !
  - E' un predicato extra-logico, che può risultare difficile da comprendere …

Controllo di un Programma

Per capire meglio come funziona il predicato cut è utile sia rappresentare l'albero SLD esplorato dall'interprete Prolog:

```
(cl1)    a :- p,b.
(cl2)    a :- p,c.
(cl3)    p.
```



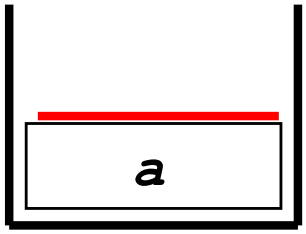
Sarà utile anche fare riferimento al modello *run-time* della macchina astratta Prolog

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

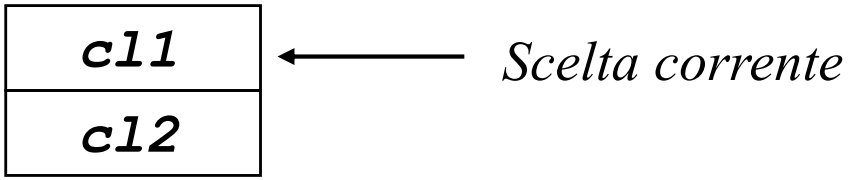
Controllo di un Programma

```
(cl1)      a :- p,b.
(cl2)      a :- p,c.
(cl3)      p.
```

– E la valutazione della query `:-a.`



Stack di esecuzione



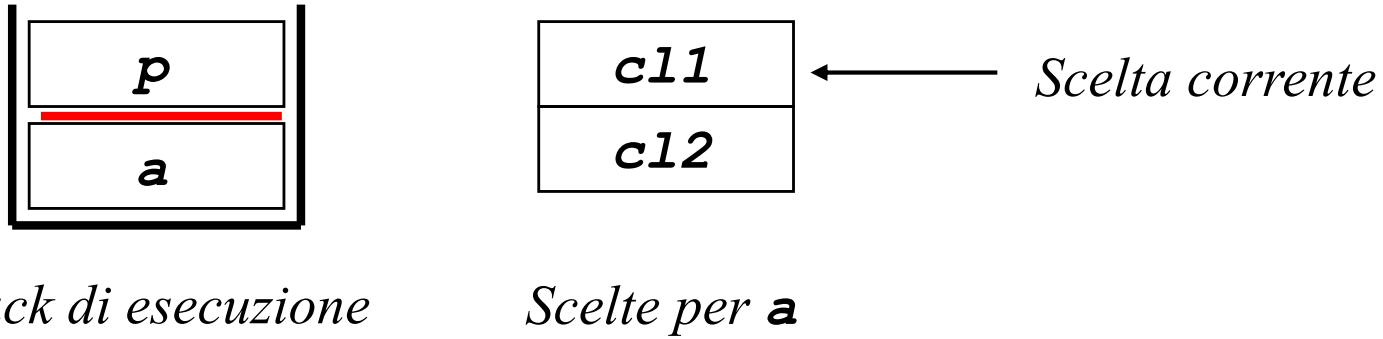
Stack di backtracking:
Scelte per **a**

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Controllo di un Programma

```
(cl1)      a :- p,b.
(cl2)      a :- p,c.
(cl3)      p.
```

– E la valutazione della query `:-a.`



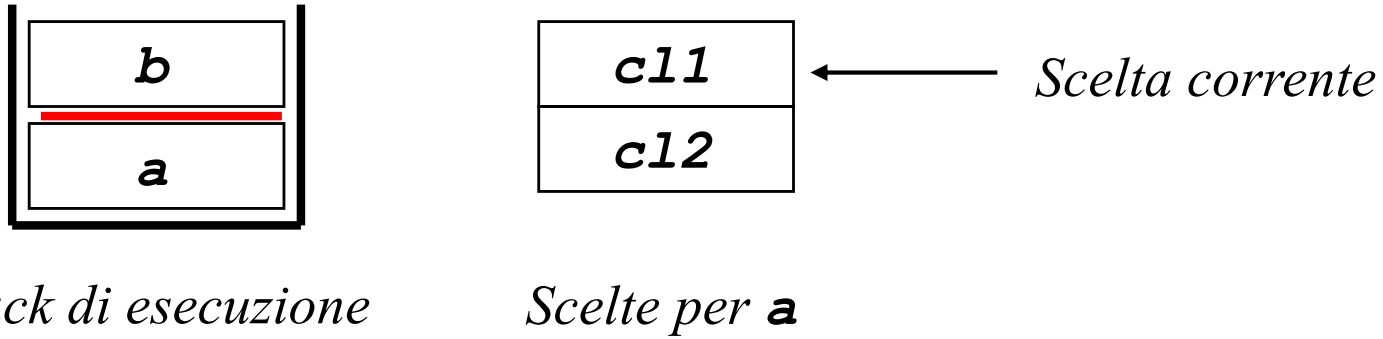
La valutazione di p ha successo!

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Controllo di un Programma

```
(cl1)      a :- p,b.
(cl2)      a :- p,c.
(cl3)      p.
```

– E la valutazione della query `:-a.`

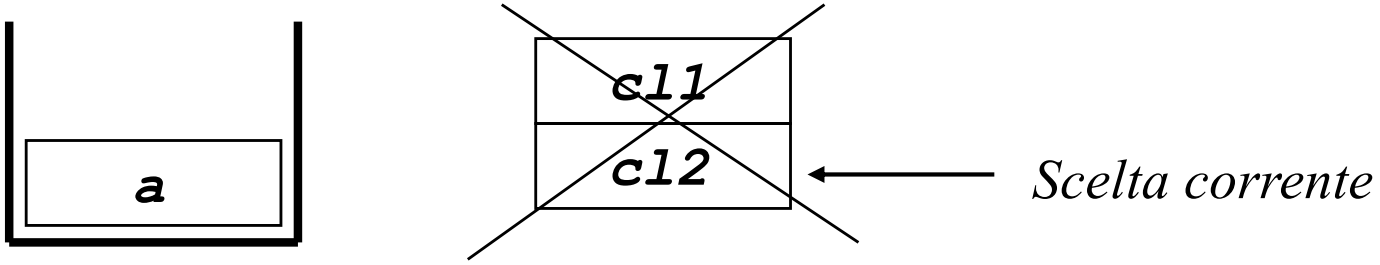


La valutazione di b fallisce ⇒ viene attivato il meccanismo di backtracking

Controllo di un Programma

```
(cl1)      a :- p,b.
(cl2)      a :- p,c.
(cl3)      p.
```

– E la valutazione della query `:-a.`



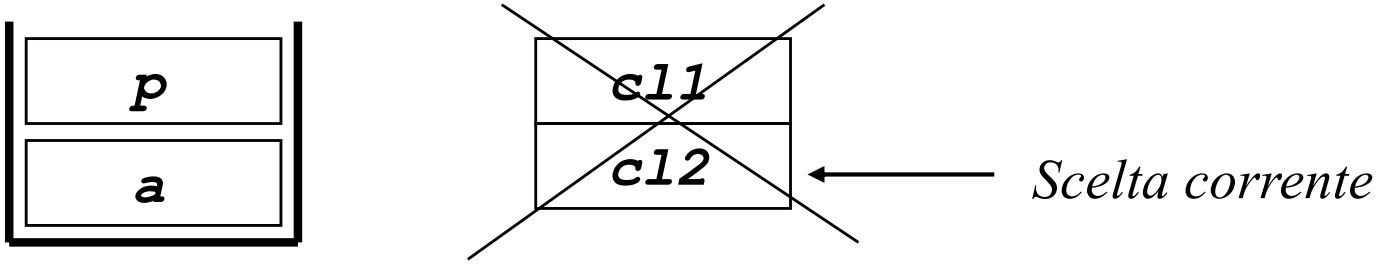
Stack di esecuzione *Scelte per* **a**

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Controllo di un Programma

```
(cl1)      a :- p,b.
(cl2)      a :- p,c.
(cl3)      p.
```

– E la valutazione della query `:-a.`



Stack di esecuzione *Scelte per* **a**

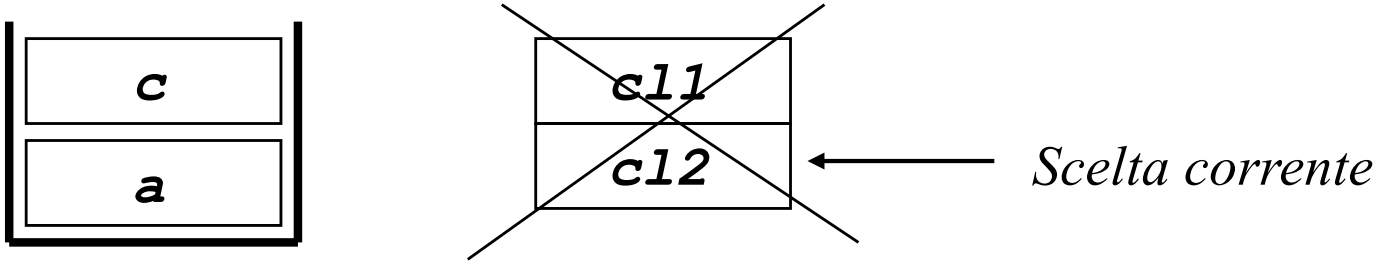
La valutazione di p ha successo

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Controllo di un Programma

```
(cl1)      a :- p,b.
(cl2)      a :- p,c.
(cl3)      p.
```

– E la valutazione della query `:-a.`



Stack di esecuzione *Scelte per* **a**

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

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Modello Run-time del Prolog

Due stack:

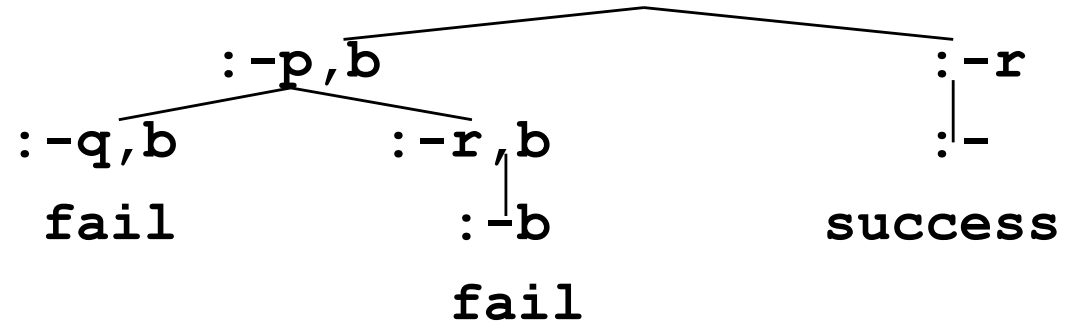
- **Stack di esecuzione** che contiene i record di attivazione (***Environment***) delle varie procedure

- **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.

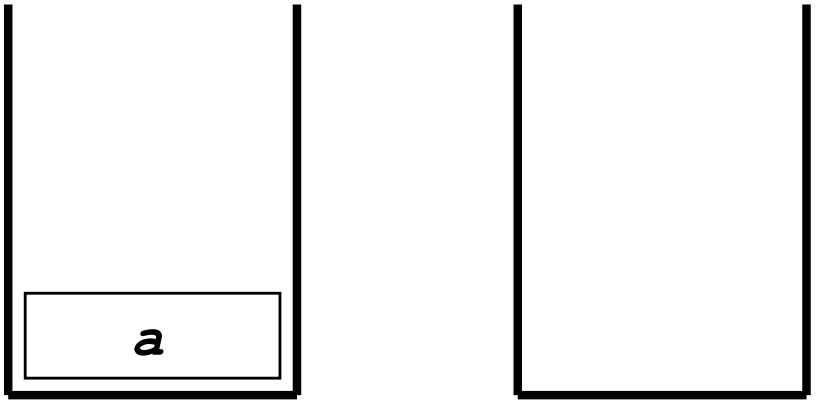
Osservazione: in realtà è utilizzato un solo stack, con alternanza di *environment* e *choice point; le linee rosse indicate rappresentano i choice point*

Controllo di un Programma

```
(cl1)    a :- p,b.
(cl2)    a :- r.
(cl3)    p :- q.
(cl4)    p :- r.
(cl5)    r.
```



– E la valutazione della query `:-a.`

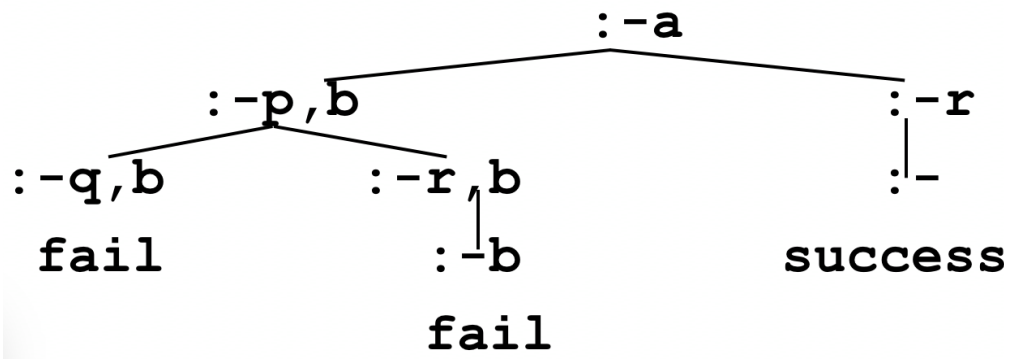


Stack di esecuzione *Stack di backtracking*

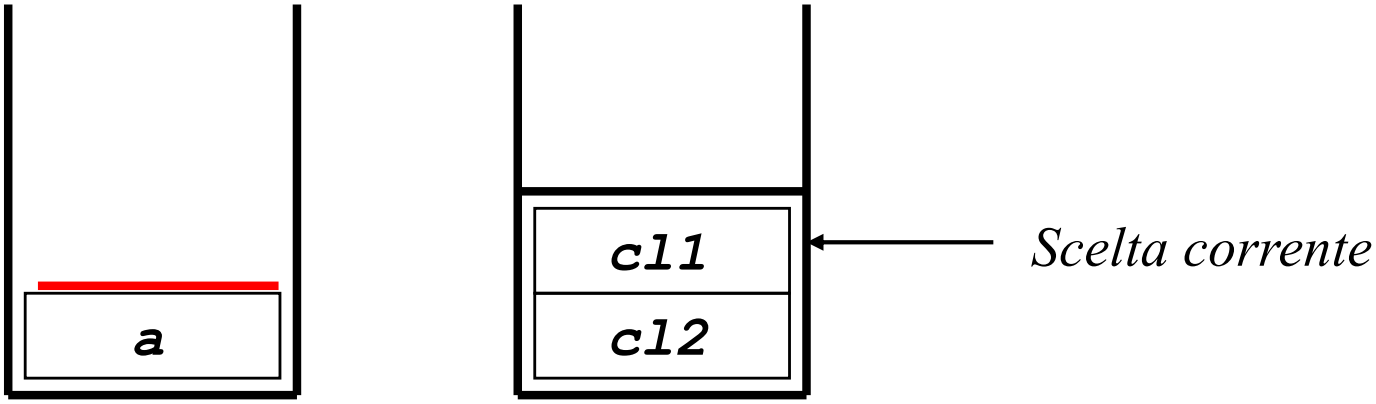
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Controllo di un Programma

```
(cl1)      a :- p,b.
(cl2)      a :- r.
(cl3)      p :- q.
(cl4)      p :- r.
(cl5)      r.
```



– E la valutazione della query `:-a.`

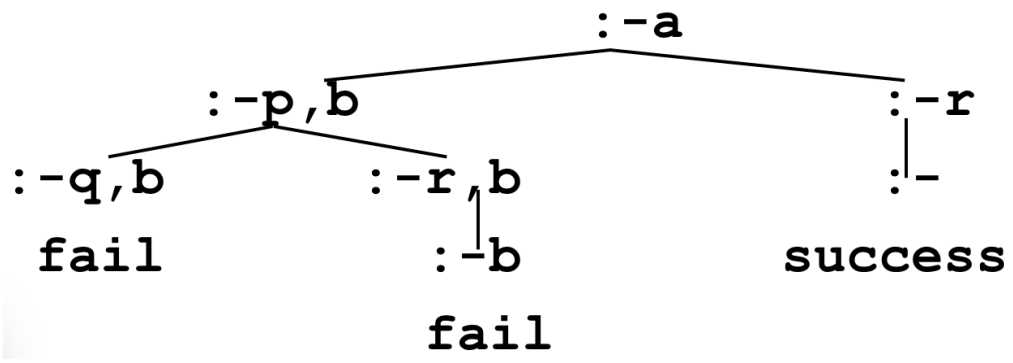


Stack di esecuzione *Stack di backtracking*

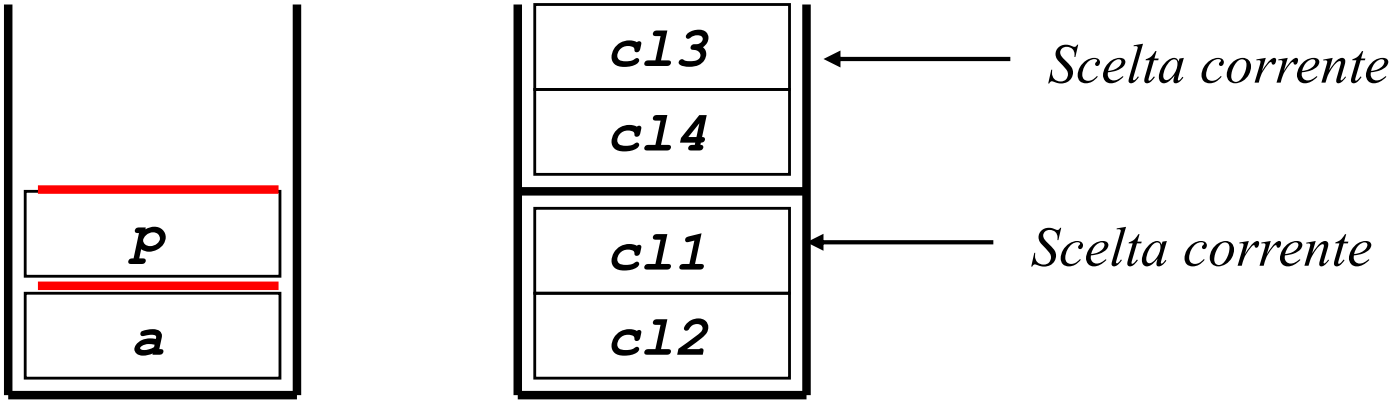
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Controllo di un Programma

```
(cl1)      a :- p,b.
(cl2)      a :- r.
(cl3)      p :- q.
(cl4)      p :- r.
(cl5)      r.
```



– E la valutazione della query `:-a.`

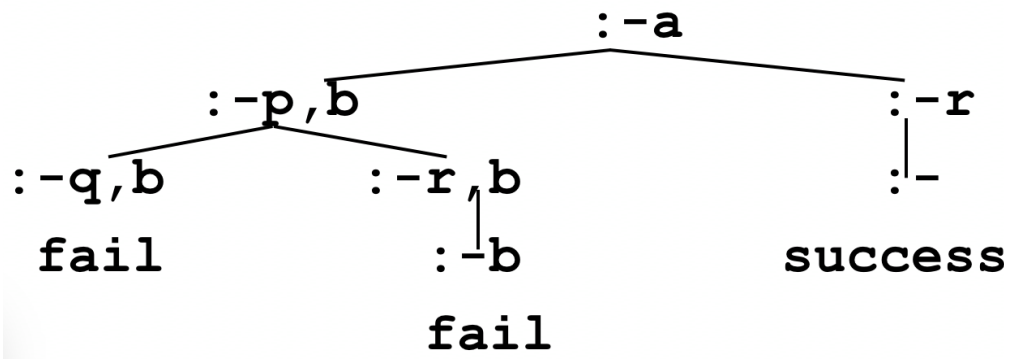


Stack di esecuzione *Stack di backtracking*

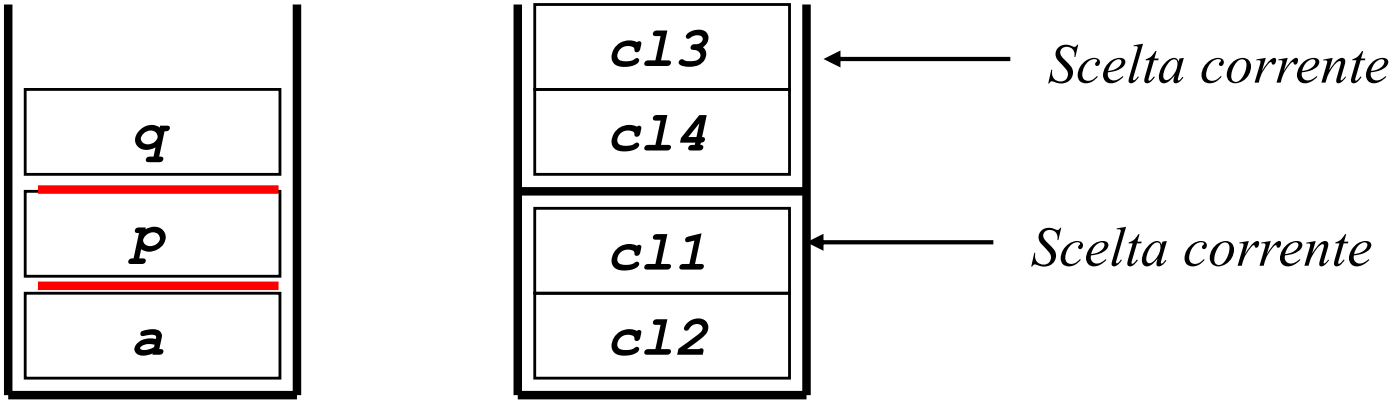
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Controllo di un Programma

```
(cl1)      a :- p,b.
(cl2)      a :- r.
(cl3)      p :- q.
(cl4)      p :- r.
(cl5)      r.
```



– E la valutazione della query `:-a.`



Stack di esecuzione

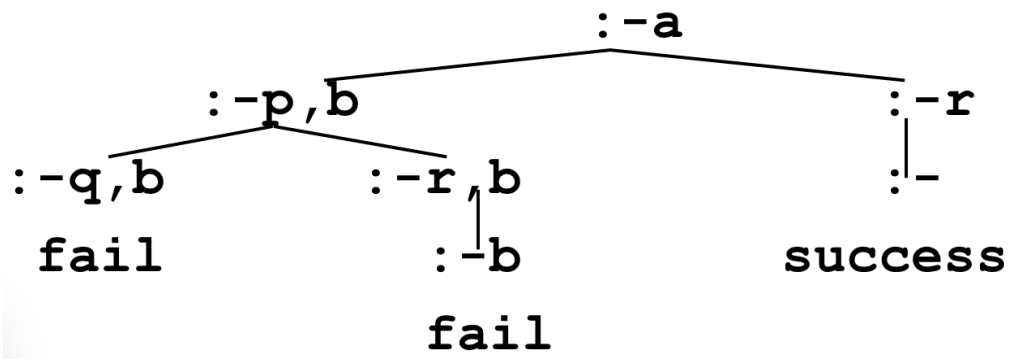
Stack di backtracking

Fallimento

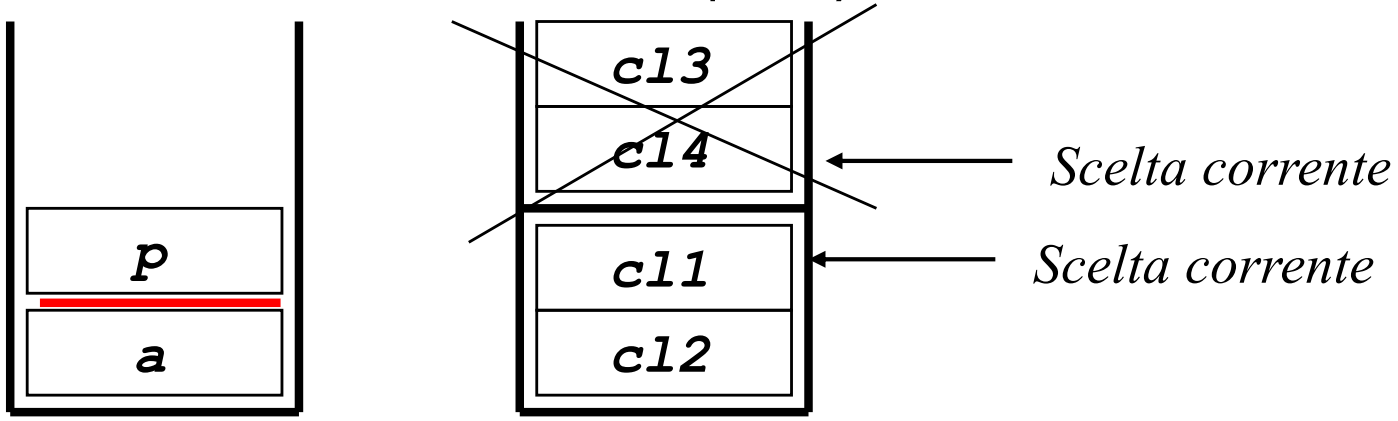
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Controllo di un Programma

```
(cl1)      a :- p,b.
(cl2)      a :- r.
(cl3)      p :- q.
(cl4)      p :- r.
(cl5)      r.
```



– E la valutazione della query `:-a.`

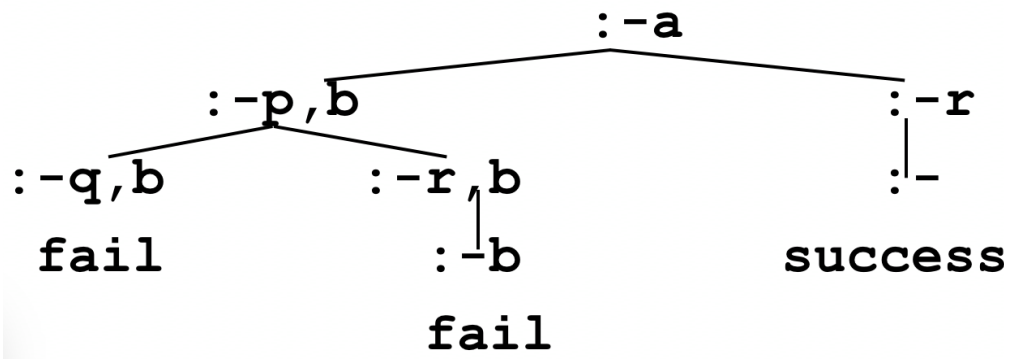


Stack di esecuzione *Stack di backtracking*

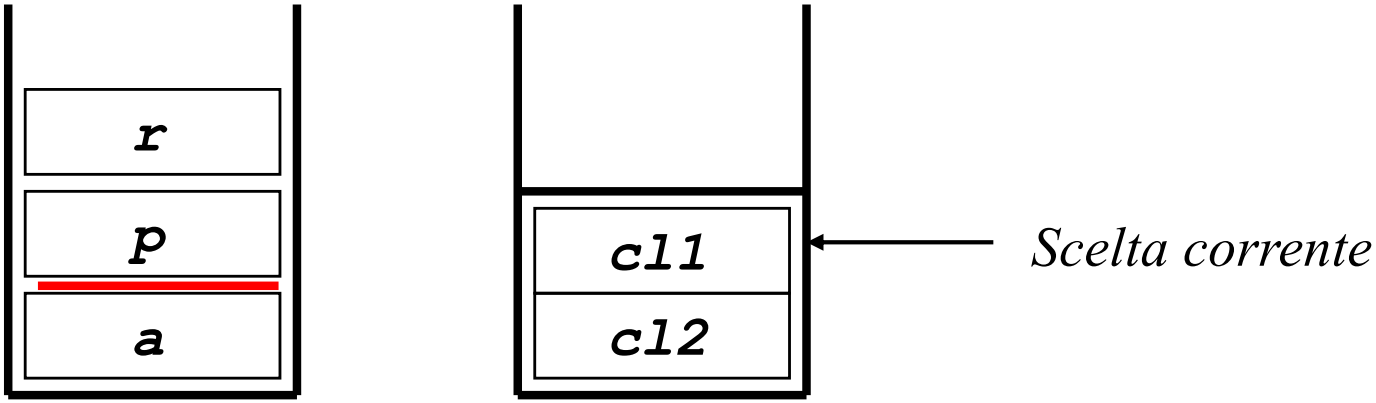
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Controllo di un Programma

```
(cl1)      a :- p,b.
(cl2)      a :- r.
(cl3)      p :- q.
(cl4)      p :- r.
(cl5)      r.
```



– E la valutazione della query `:-a.`



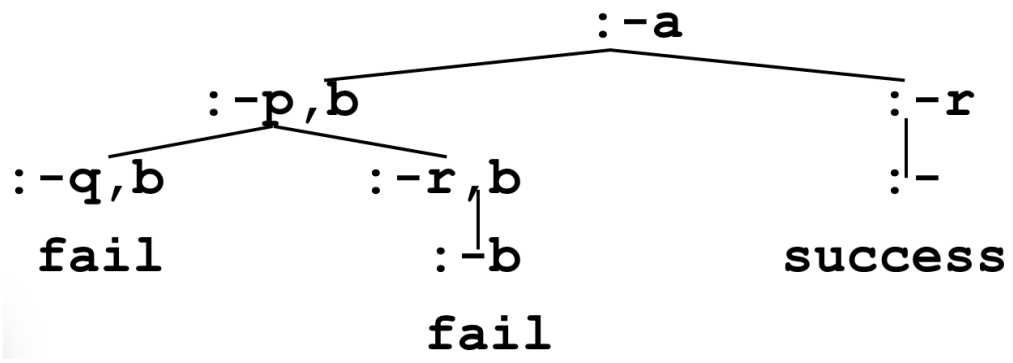
Stack di esecuzione *Stack di backtracking*

`r` ha successo

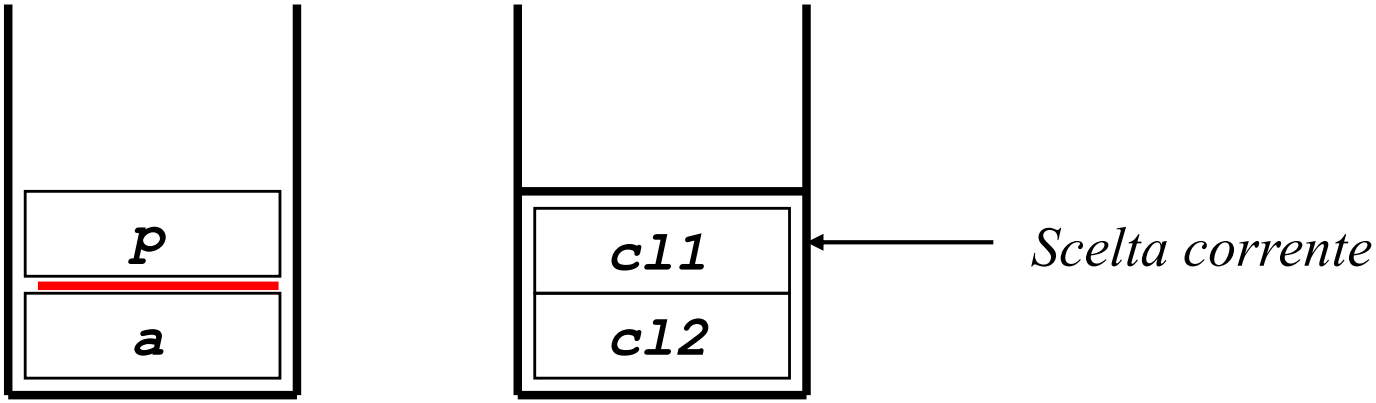
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Controllo di un Programma

```
(cl1)      a :- p,b.
(cl2)      a :- r.
(cl3)      p :- q.
(cl4)      p :- r.
(cl5)      r.
```



– E la valutazione della query `:-a.`



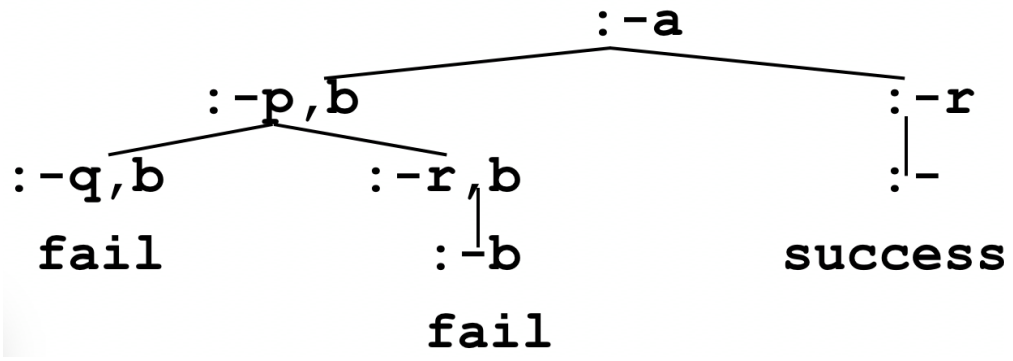
Stack di esecuzione *Stack di backtracking*

p ha successo

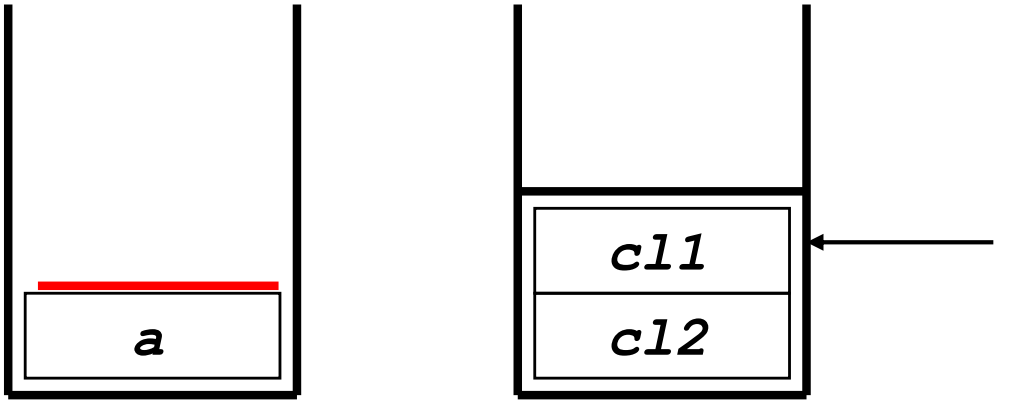
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Controllo di un Programma

```
(cl1)      a :- p,b.
(cl2)      a :- r.
(cl3)      p :- q.
(cl4)      p :- r.
(cl5)      r.
```



– E la valutazione della query `:-a.`

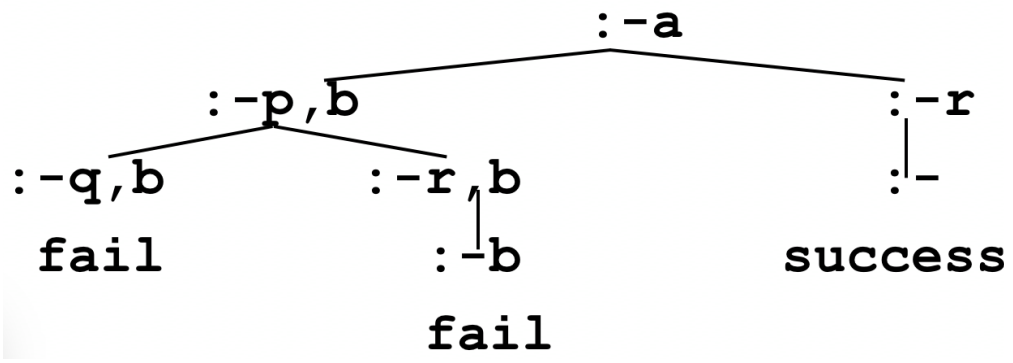


Stack di esecuzione *Stack di backtracking*

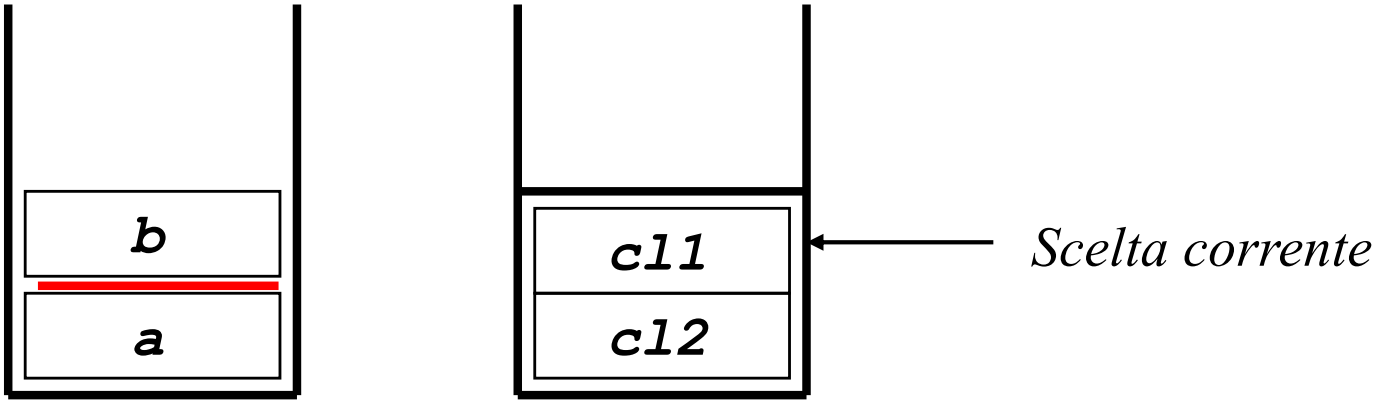
Si continua con **b**

Controllo di un Programma

```
(cl1)      a :- p,b.
(cl2)      a :- r.
(cl3)      p :- q.
(cl4)      p :- r.
(cl5)      r.
```



– E la valutazione della query `:-a.`



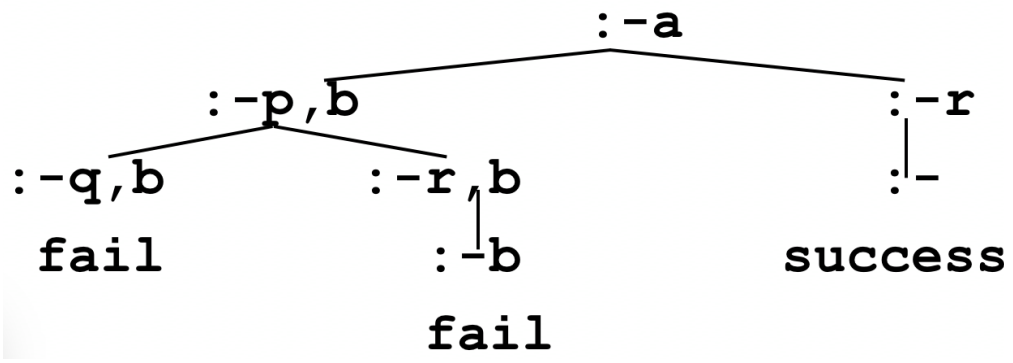
Stack di esecuzione *Stack di backtracking*

b fallisce

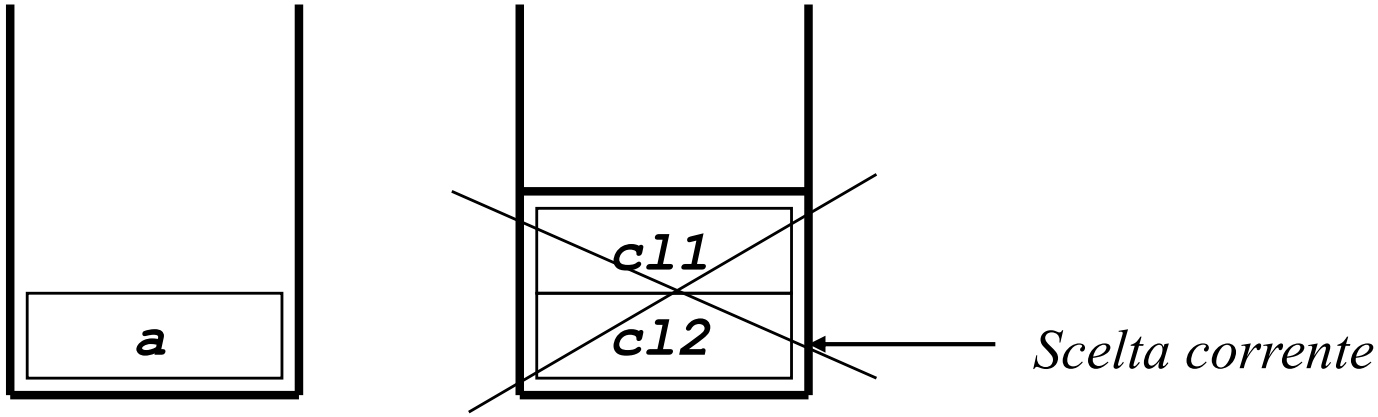
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Controllo di un Programma

```
(cl1)      a :- p,b.
(cl2)      a :- r.
(cl3)      p :- q.
(cl4)      p :- r.
(cl5)      r.
```



– E la valutazione della query `:-a.`

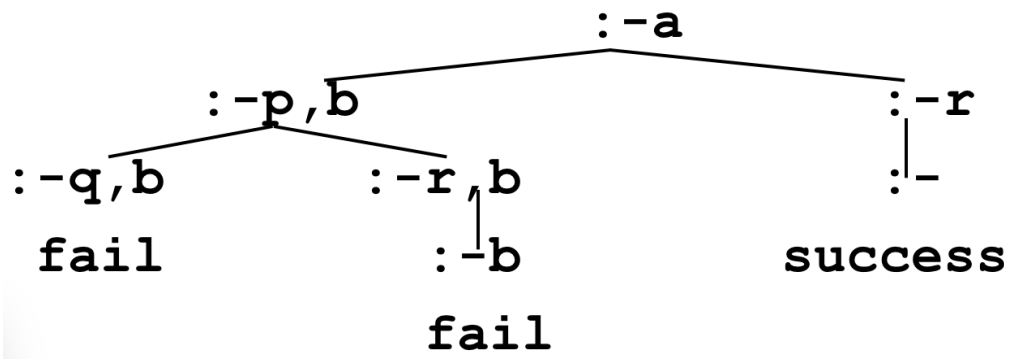


Stack di esecuzione *Stack di backtracking*

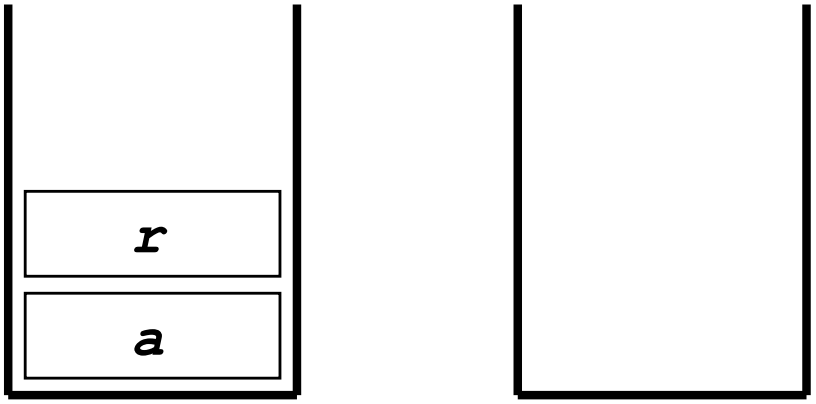
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Controllo di un Programma

```
(cl1)      a :- p,b.
(cl2)      a :- r.
(cl3)      p :- q.
(cl4)      p :- r.
(cl5)      r.
```



– E la valutazione della query `:-a.`



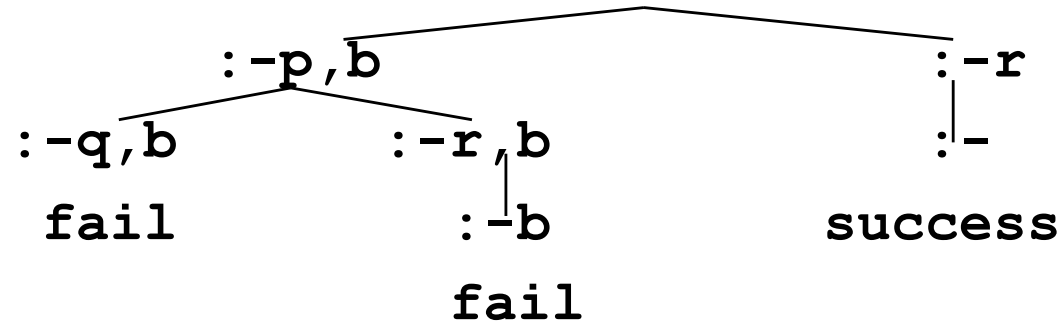
Stack di esecuzione *Stack di backtracking*

`r` ha successo

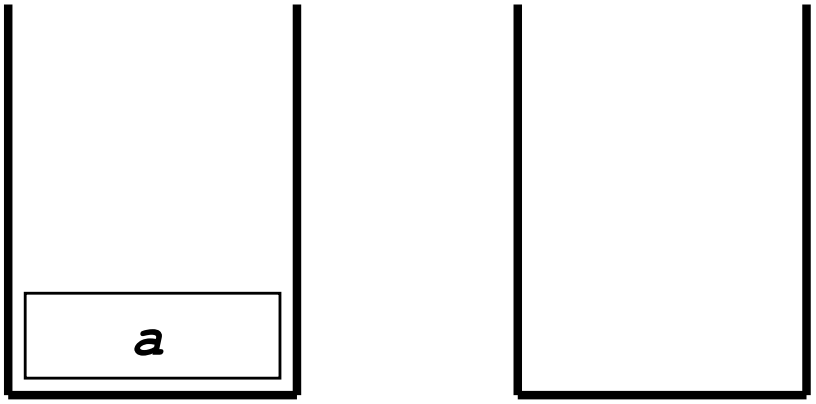
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Controllo di un Programma

```
(cl1)    a :- p,b.
(cl2)    a :- r.
(cl3)    p :- q.
(cl4)    p :- r.
(cl5)    r.
```



– E la valutazione della query `:-a.`



Stack di esecuzione *Stack di backtracking*

Successo

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

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:

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

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 $q_1, q_2, \ldots, q_i$ 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:

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

- Se la valutazione di $q_{i+1}, q_{i+2}, \ldots, q_n$ fallisce, fallisce tutta la valutazione di p. Infatti, anche se p o $q_1, q_2, \ldots, q_i$ avessero punti di scelta questi sarebbero eliminati dal cut;

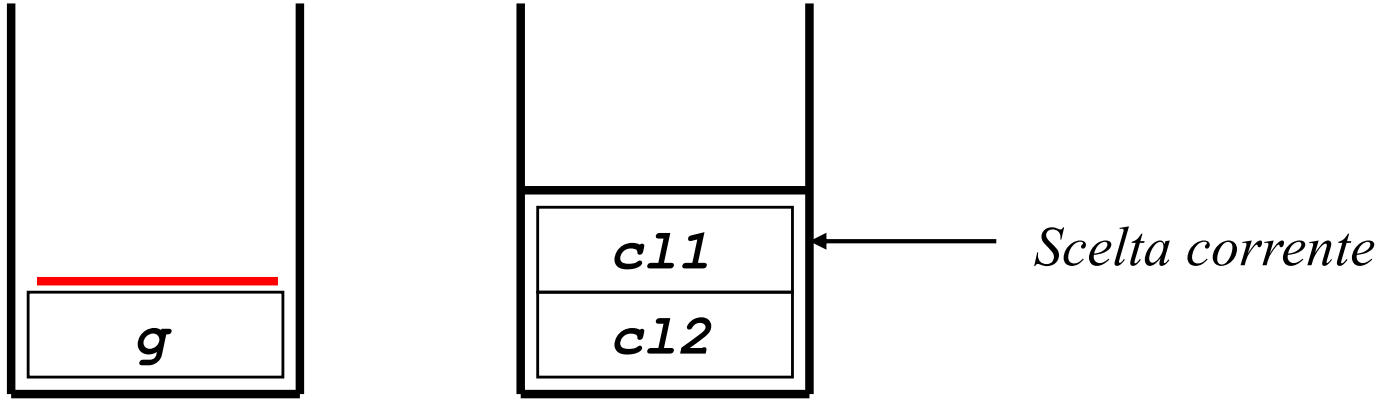
- **Il cut taglia rami dell'albero SLD !!!**

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

Effetto del CUT

```
(cl1)      g :- a.
(cl2)      g :- s.
(cl3)      a :- p,!,b.
(cl4)      a :- r.
(cl5)      p :- q.
(cl6)      p :- r.
(cl7)      r.
```

– E la valutazione della query `:-g.`

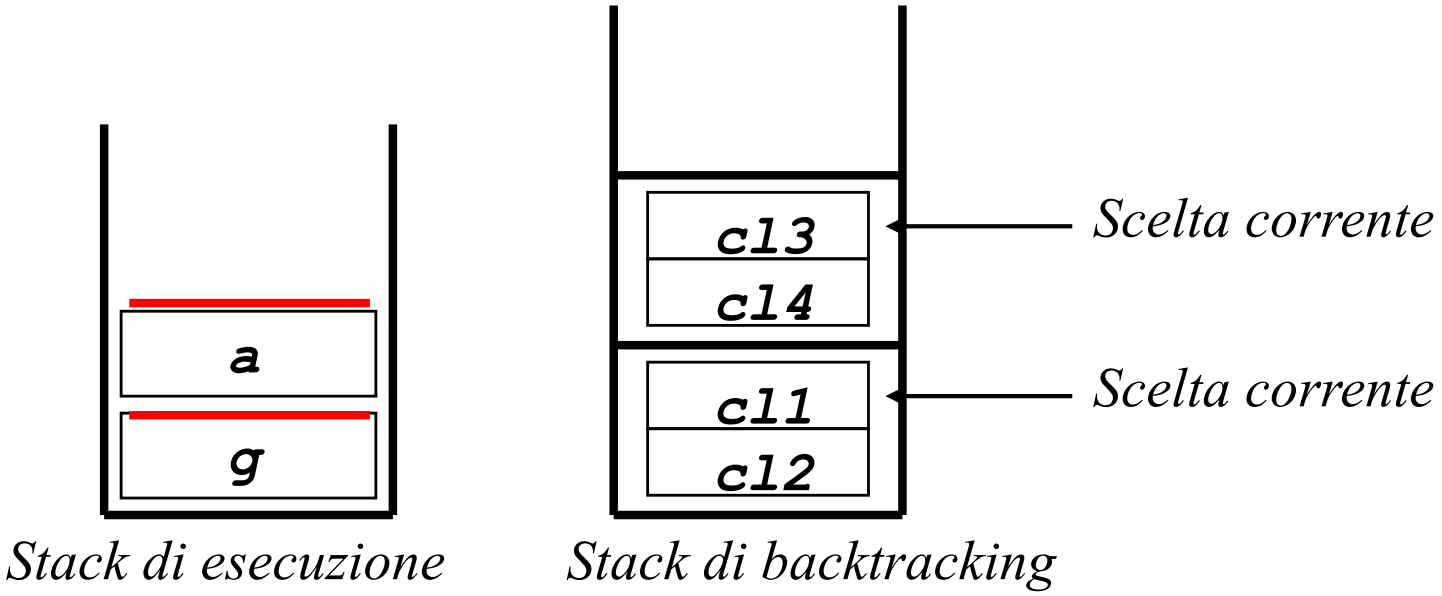


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Effetto del CUT

```
(cl1)      g :- a.
(cl2)      g :- s.
(cl3)      a :- p,!,b.
(cl4)      a :- r.
(cl5)      p :- q.
(cl6)      p :- r.
(cl7)      r.
```

– E la valutazione della query `:-g.`

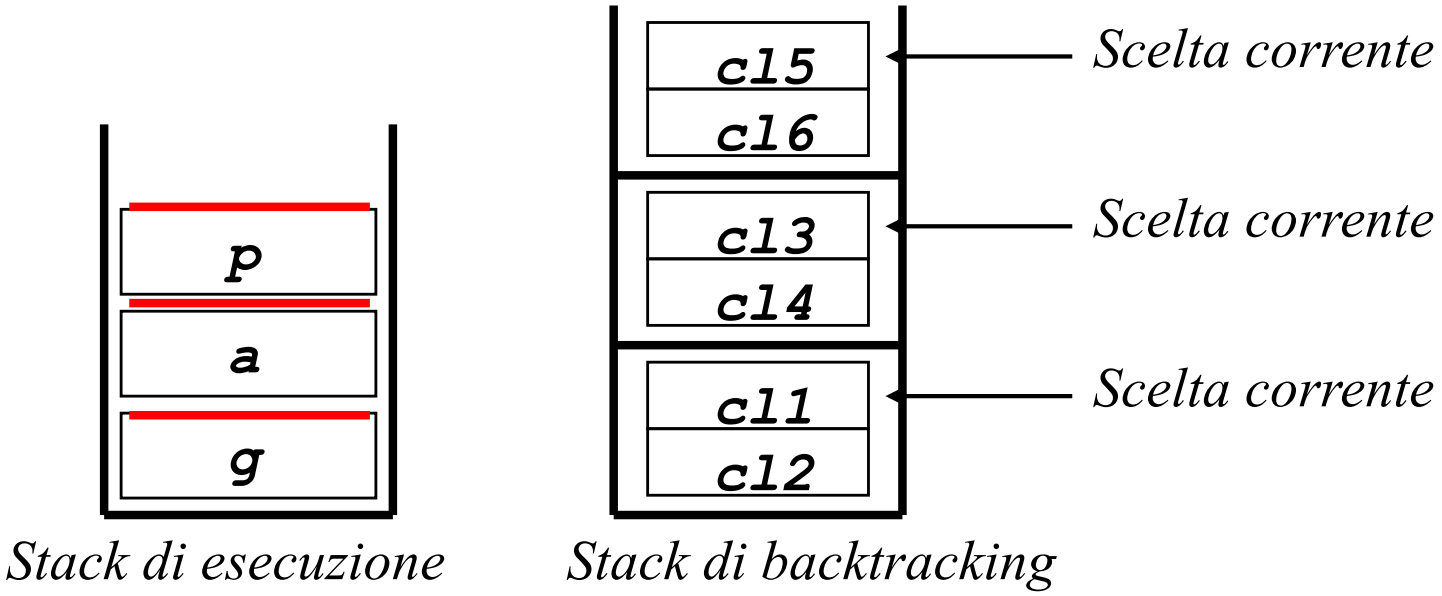


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Effetto del CUT

```
(cl1)      g :- a.
(cl2)      g :- s.
(cl3)      a :- p,!,b.
(cl4)      a :- r.
(cl5)      p :- q.
(cl6)      p :- r.
(cl7)      r.
```

– E la valutazione della query `:-g.`

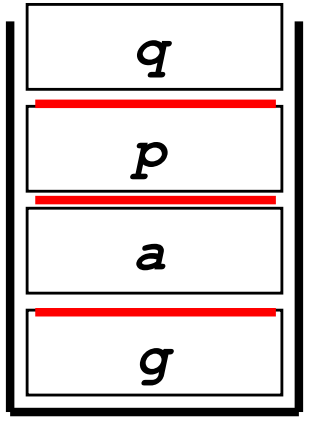


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

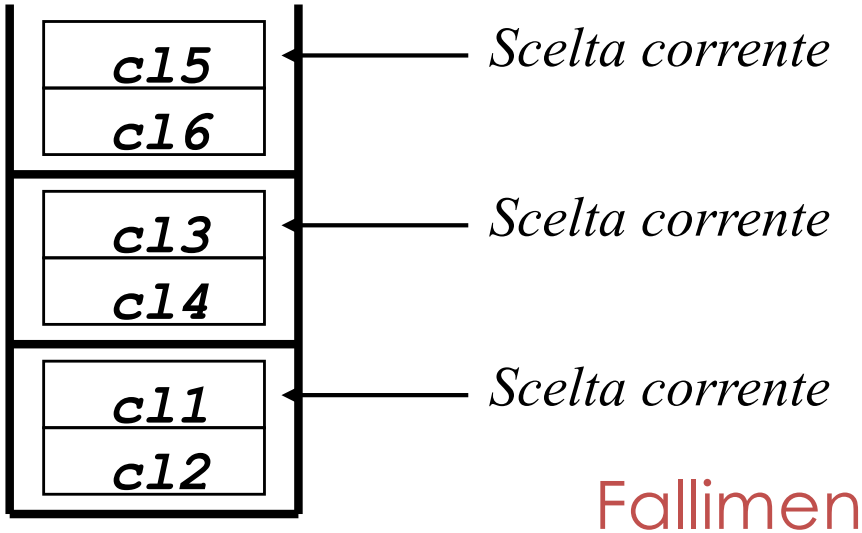
Effetto del CUT

```
(cl1)      g :- a.
(cl2)      g :- s.
(cl3)      a :- p,!,b.
(cl4)      a :- r.
(cl5)      p :- q.
(cl6)      p :- r.
(cl7)      r.
```

– E la valutazione della query `:-g.`



Stack di esecuzione



Stack di backtracking

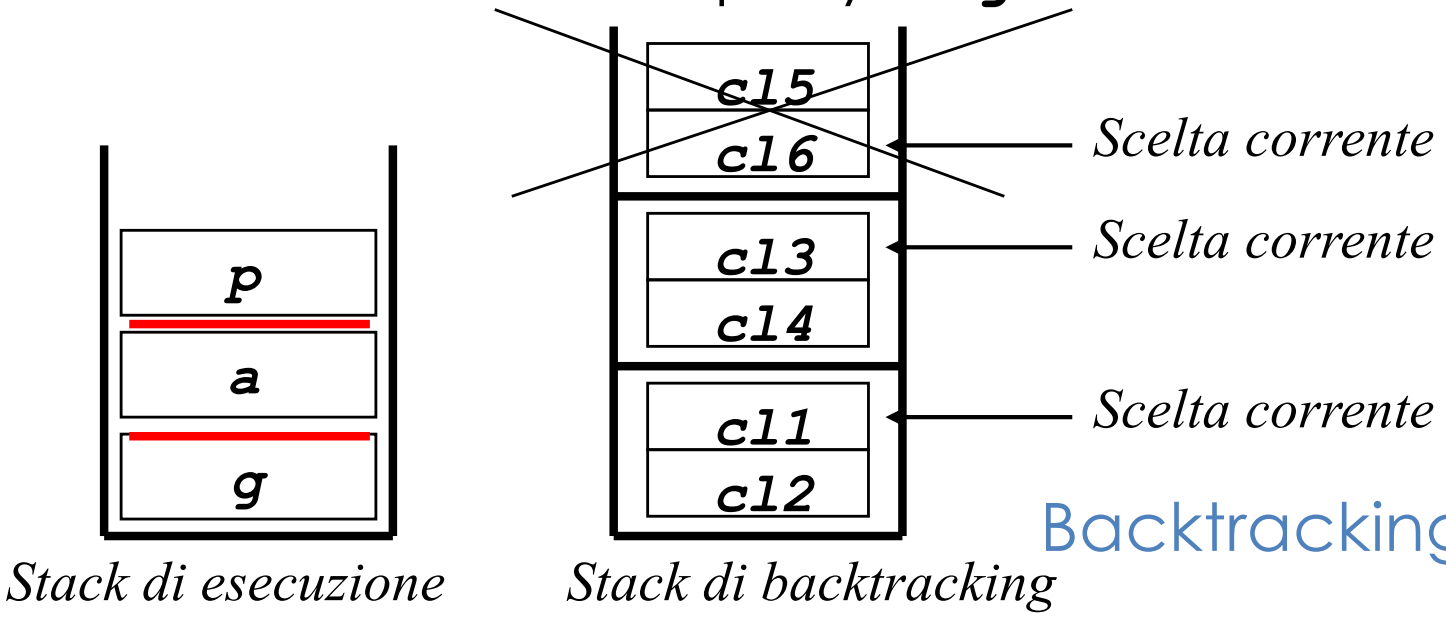
Fallimento

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Effetto del CUT

```
(cl1)      g :- a.
(cl2)      g :- s.
(cl3)      a :- p,!,b.
(cl4)      a :- r.
(cl5)      p :- q.
(cl6)      p :- r.
(cl7)      r.
```

– E la valutazione della query `:-g.`

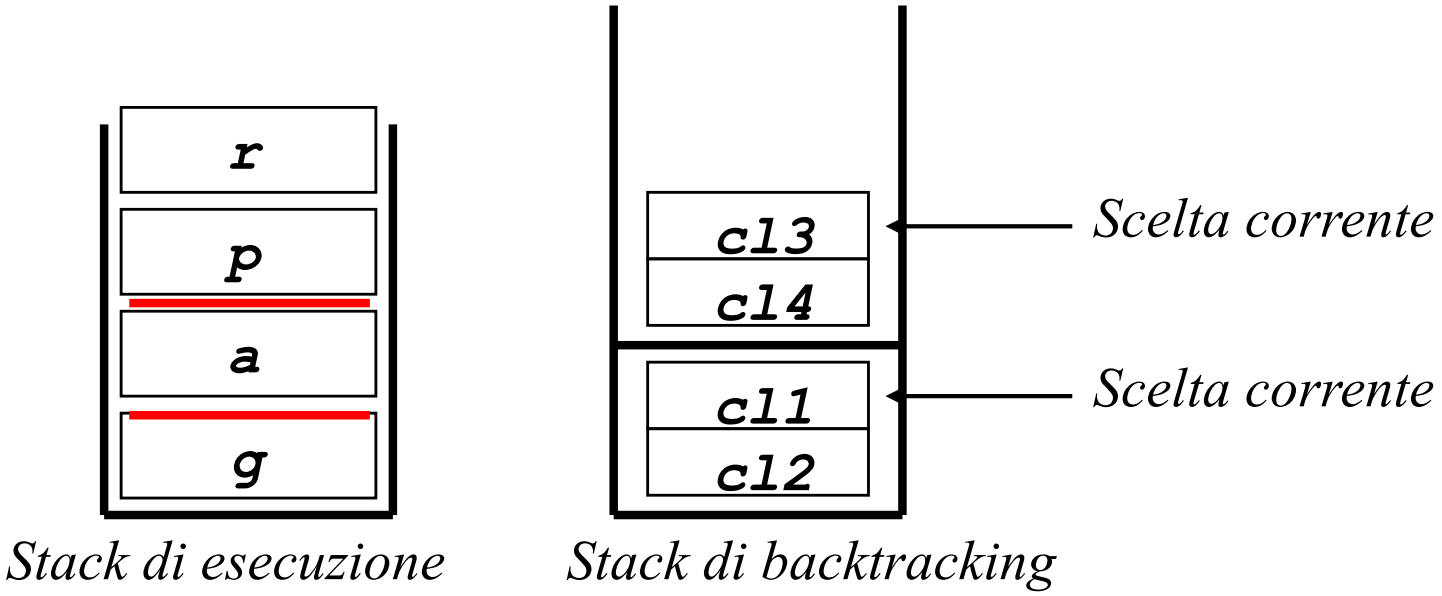


Backtracking su p/0

Effetto del CUT

```
(cl1)      g :- a.
(cl2)      g :- s.
(cl3)      a :- p,!,b.
(cl4)      a :- r.
(cl5)      p :- q.
(cl6)      p :- r.
(cl7)      r.
```

– E la valutazione della query `:-g.`

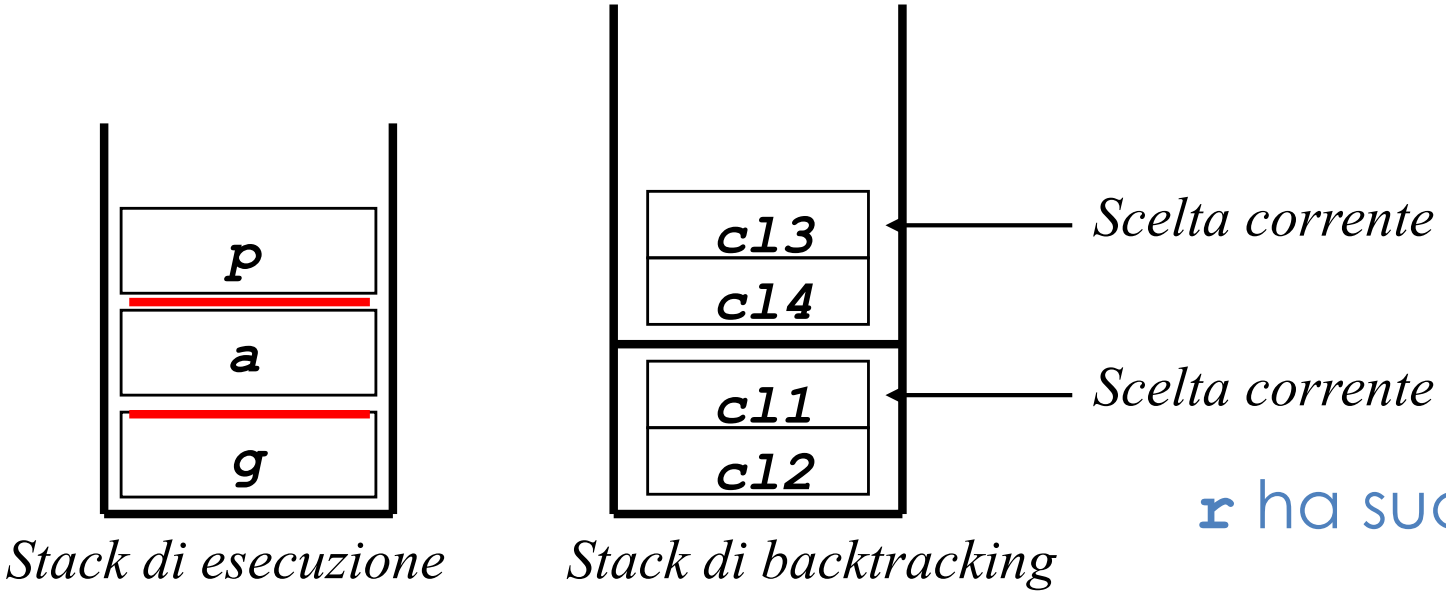


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Effetto del CUT

```
(cl1)      g :- a.
(cl2)      g :- s.
(cl3)      a :- p,!,b.
(cl4)      a :- r.
(cl5)      p :- q.
(cl6)      p :- r.
(cl7)      r.
```

– E la valutazione della query `:-g.`



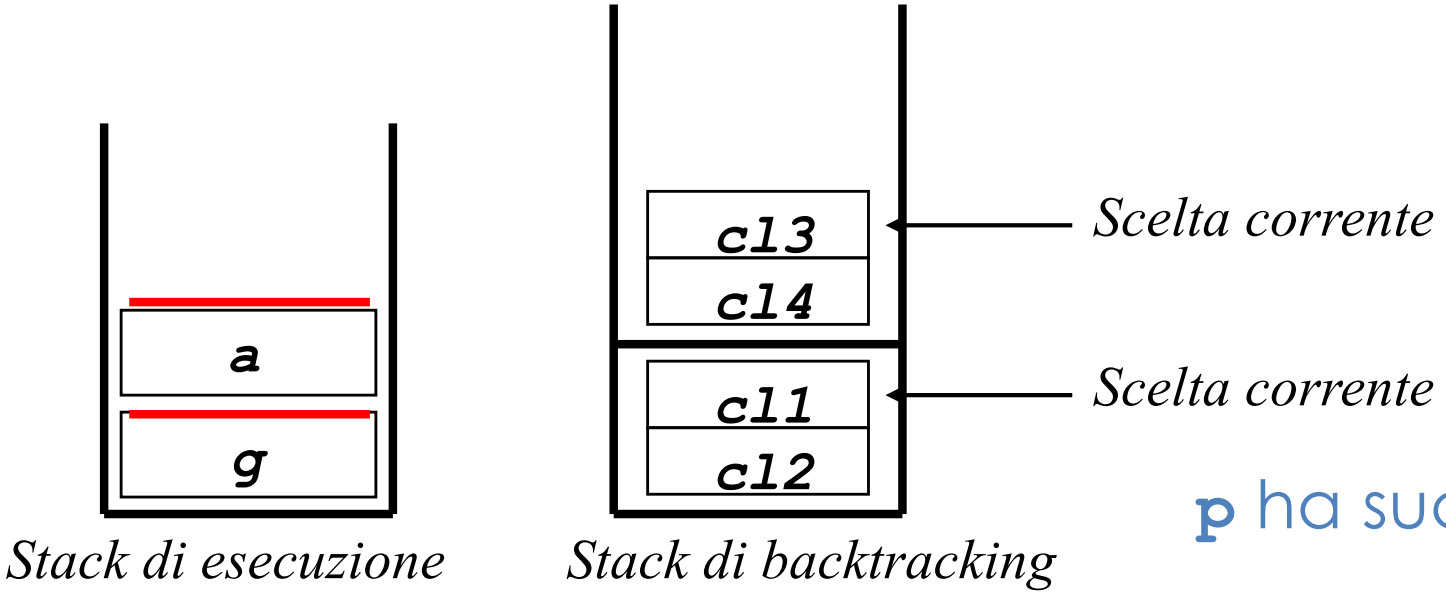
`r` ha successo

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Effetto del CUT

```
(cl1)      g :- a.
(cl2)      g :- s.
(cl3)      a :- p,!,b.
(cl4)      a :- r.
(cl5)      p :- q.
(cl6)      p :- r.
(cl7)      r.
```

– E la valutazione della query `:-g.`

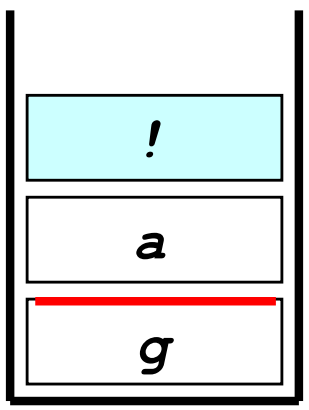


`p` ha successo

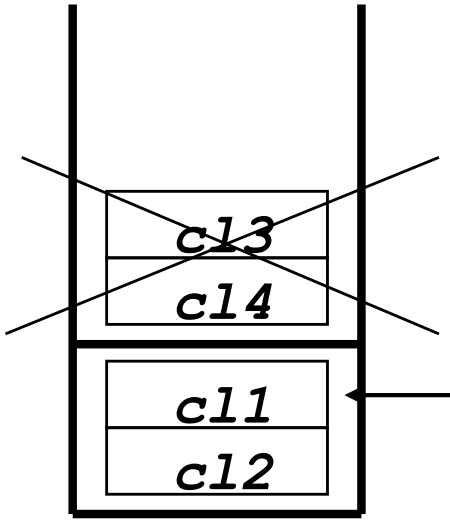
Effetto del CUT

```
(cl1)      g :- a.
(cl2)      g :- s.
(cl3)      a :- p,!,p.
(cl4)      a :- r.
(cl5)      p :- q.
(cl6)      p :- r.
(cl7)      r.
```

– E la valutazione della query `:-g.`



Stack di esecuzione



Stack di backtracking

Scelta corrente

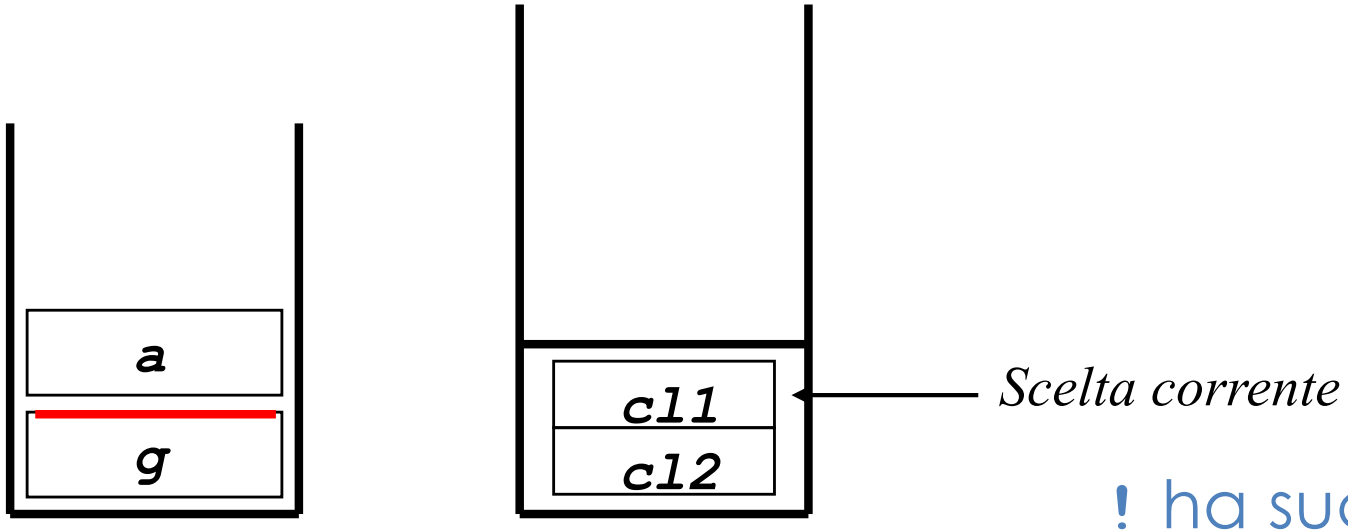
Effetto del !
Tutti i punti di scelta per p (se ce ne fossero ancora allocati) e per **a** sono rimossi dallo stack.
Il cut ha successo

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Effetto del CUT

```
(cl1)      g :- a.
(cl2)      g :- s.
(cl3)      a :- p,!,b.
(cl4)      a :- r.
(cl5)      p :- q.
(cl6)      p :- r.
(cl7)      r.
```

– E la valutazione della query `:-g.`

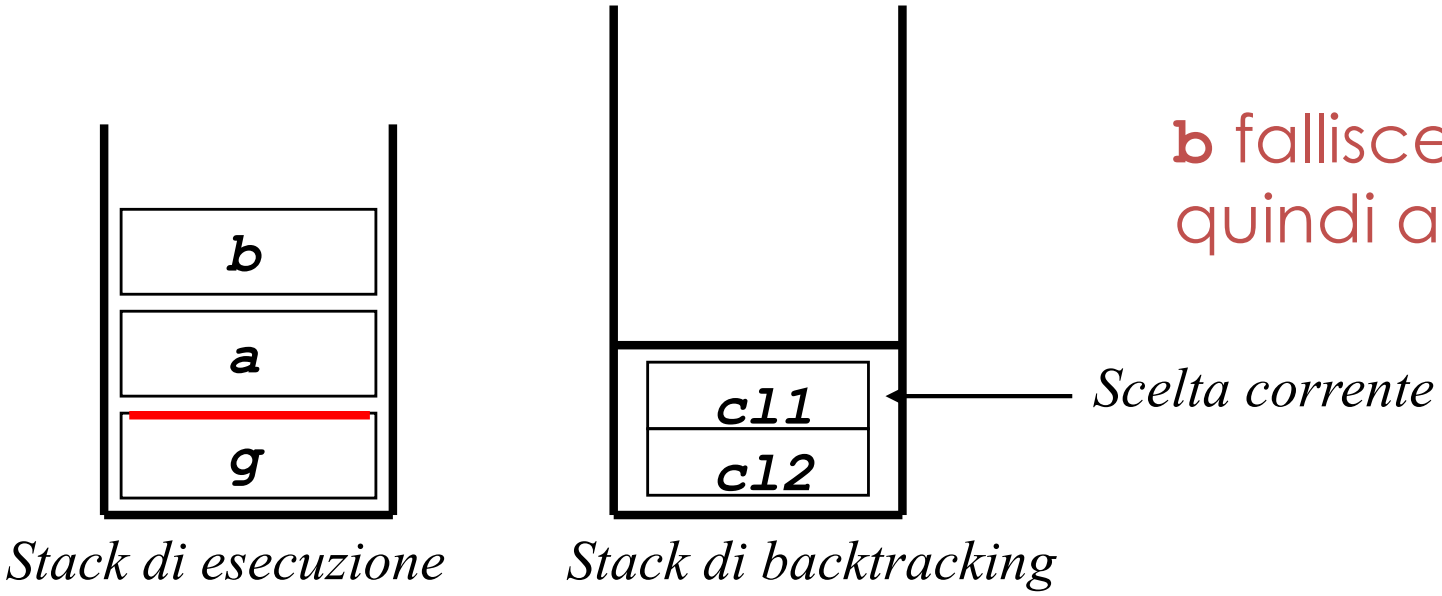


! ha successo

Effetto del CUT

```
(cl1)      g :- a.
(cl2)      g :- s.
(cl3)      a :- p,!,b.
(cl4)      a :- r.
(cl5)      p :- q.
(cl6)      p :- r.
(cl7)      r.
```

– E la valutazione della query `:-g.`



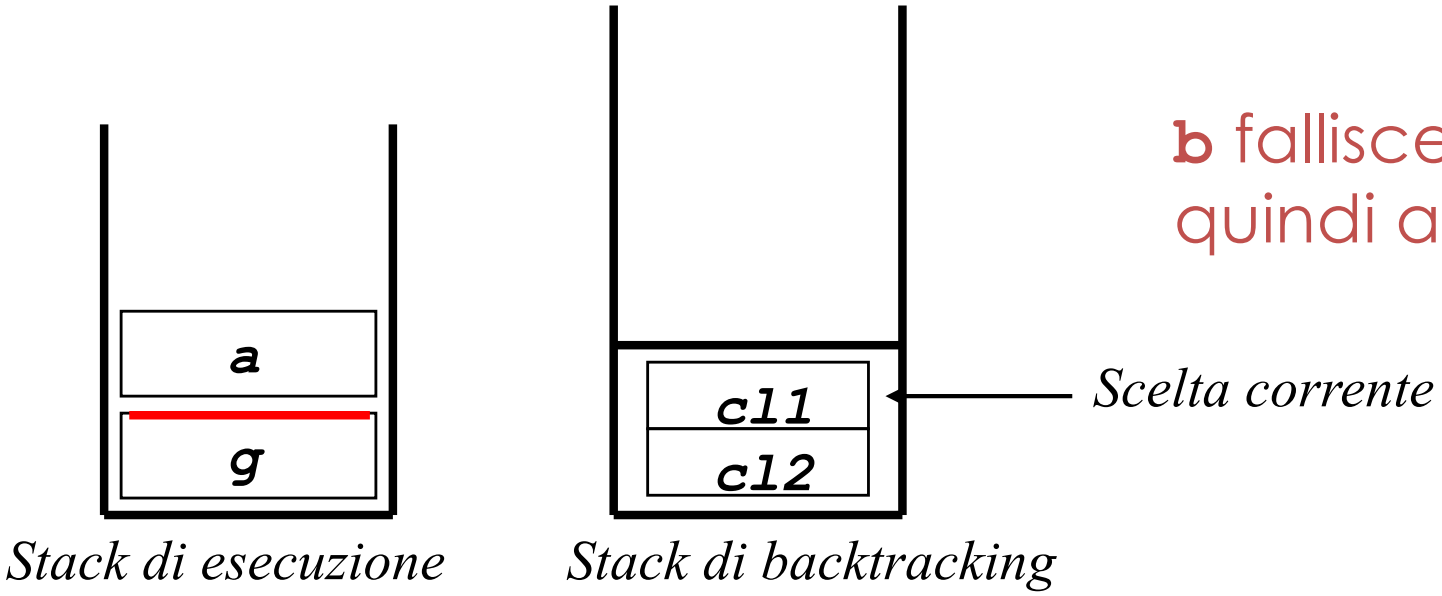
b fallisce e fallisce quindi anche **a**

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Effetto del CUT

```
(cl1)      g :- a.
(cl2)      g :- s.
(cl3)      a :- p,!,b.
(cl4)      a :- r.
(cl5)      p :- q.
(cl6)      p :- r.
(cl7)      r.
```

– E la valutazione della query `:-g.`



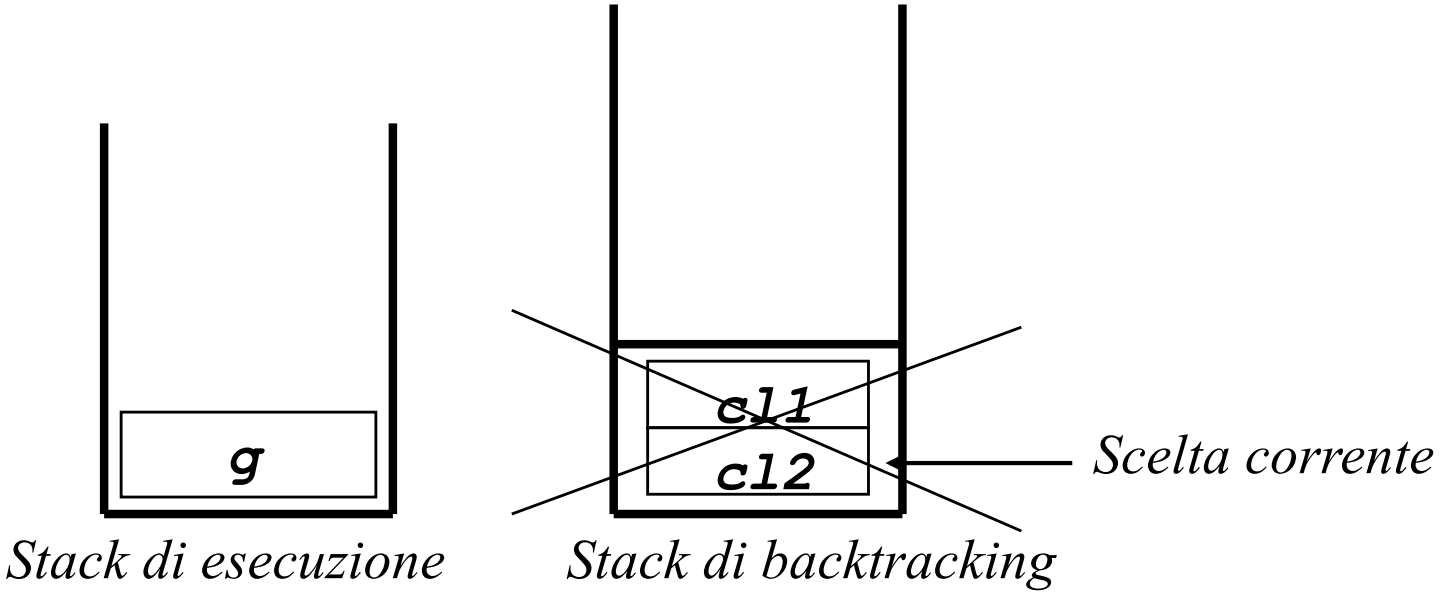
b fallisce e fallisce quindi anche **a**

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Effetto del CUT

```
(cl1)      g :- a.
(cl2)      g :- s.
(cl3)      a :- p,!,b.
(cl4)      a :- r.
(cl5)      p :- q.
(cl6)      p :- r.
(cl7)      r.
```

– E la valutazione della query `:-g.`

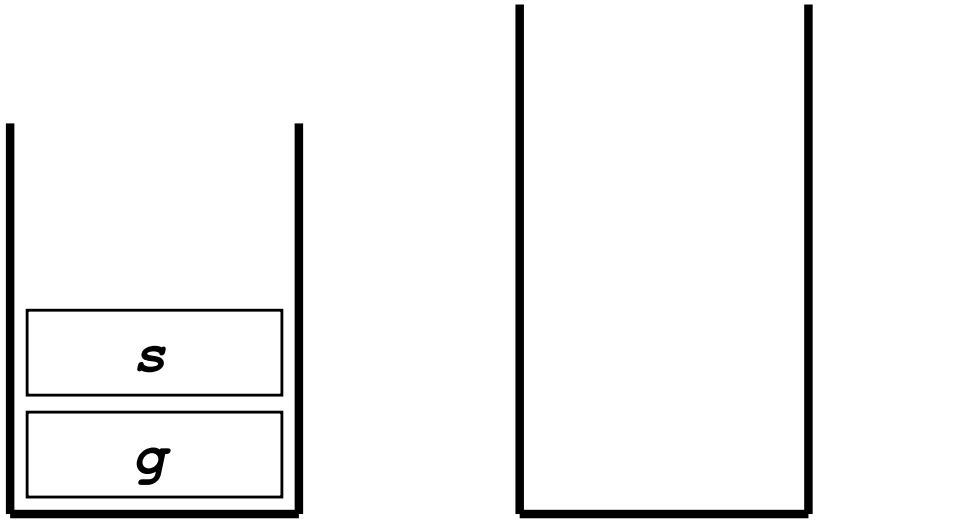


ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Effetto del CUT

```
(cl1)      g :- a.
(cl2)      g :- s.
(cl3)      a :- p,!,b.
(cl4)      a :- r.
(cl5)      p :- q.
(cl6)      p :- r.
(cl7)      r.
```

– E la valutazione della query `:-g.`



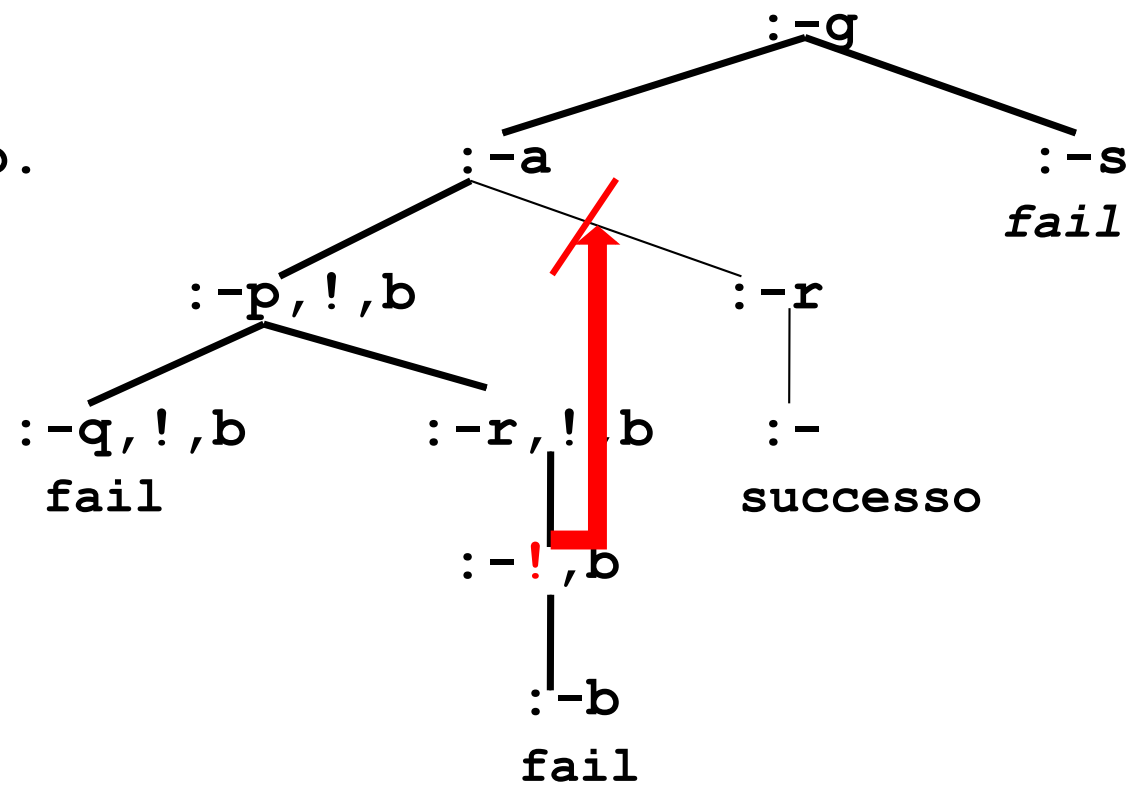
Stack di esecuzione *Stack di backtracking*

Fallimento !!!

Senza il cut la query avrebbe avuto successo

Effetto del CUT – albero SLD

```
(cl1) g :- a.
(cl2) g :- s.
(cl3) a :- p,!,b.
(cl4) a :- r.
(cl5) p :- q.
(cl6) p :- r.
(cl7) r.
```



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

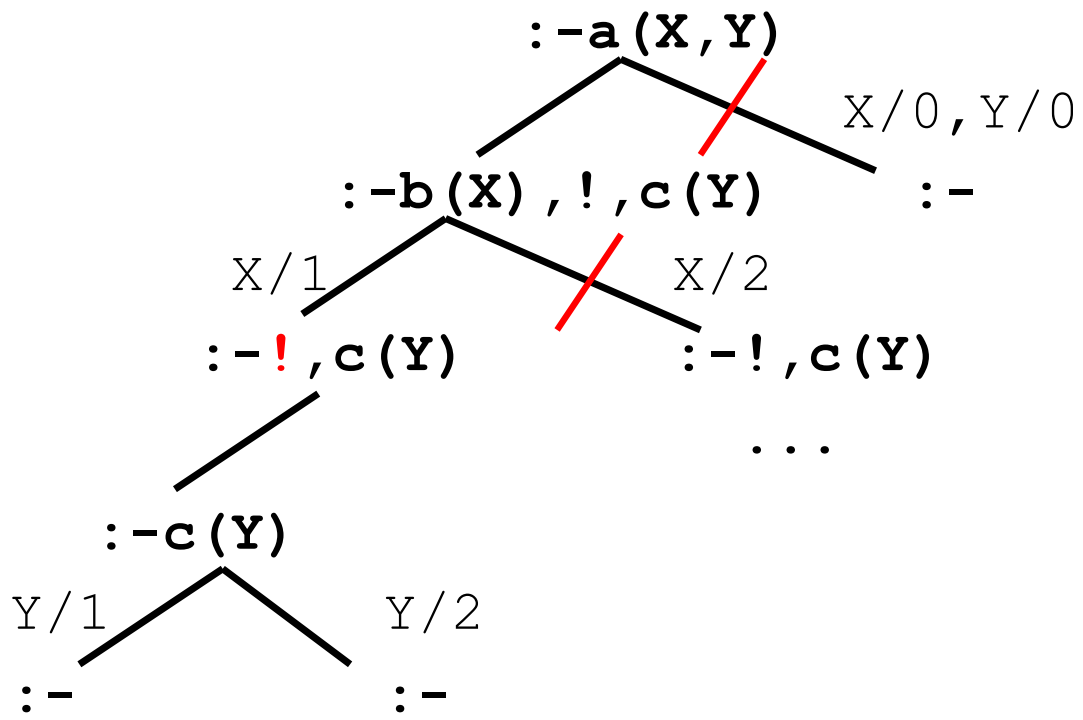
Si noti che si perde la completezza (taglio un ramo di successo)

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

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
```



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CUT: Esempio di non completezza

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

:- p(X).
yes   X=2
```

```
p(X):- q(X),!,r(X).
q(1).

q(2).
r(2).

:- p(X).
no
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

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.

Efficienza – esempio del predicato `member/2`

Abbiamo visto il predicato **`member`**:

```
member(El, [El|_]).
member(El, [X|Tail]):- member(El,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

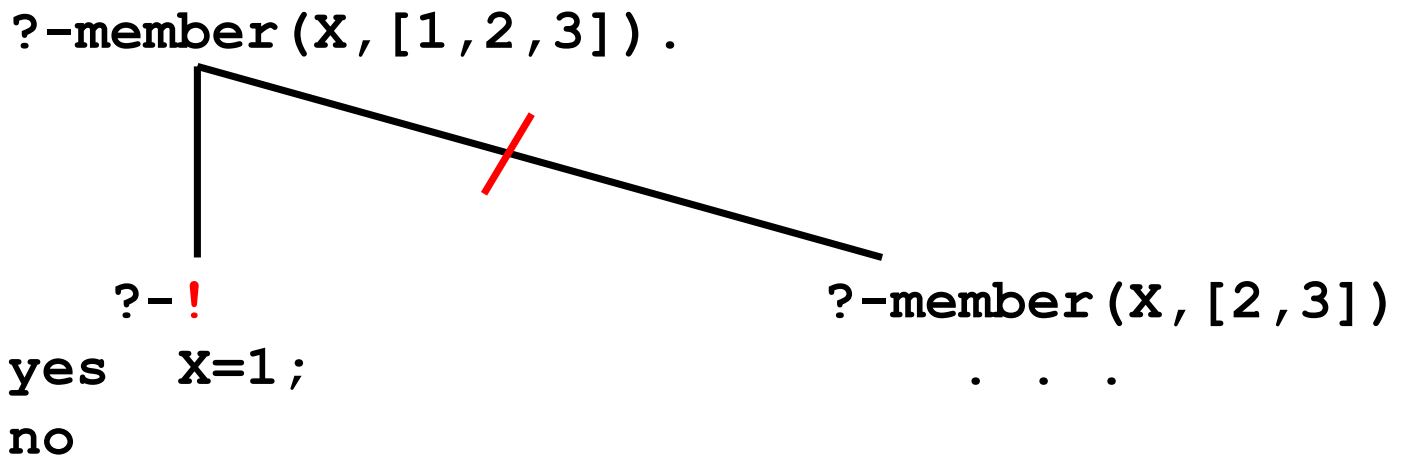
```
member(El, [El|_]):- !.
member(El, [_|Tail]):- member(El,Tail).
```

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 – esempio del predicato `member/2`

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



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)

Usi del CUT – Mutua Esclusione tra Clausole

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

Esempio:

```
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
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Usi del CUT – 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.
```

*Se **`a(X)`** è 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.*

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

Usi del CUT – Mutua Esclusione tra Clausole

Predicato che, dati un termine T e una lista L, conta le "occorrenze" di T in L:

```
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,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).
```

Usi del CUT – Mutua Esclusione tra Clausole

Predicato che, dati un termine T e una lista L, conta le "occorrenze" di T in L

```
?-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).
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Usi del CUT – Rimozione di Elementi da Liste

- Cancellazione di un elemento uguale a T dalla lista:

```
(cl1)    delete1(T, [], []).
(cl2)    delete1(T, [T|TAIL], TAIL).
(cl3)    delete1(T, [HEAD|TAIL], [HEAD|L]):-
                    T\==HEAD, delete1(T, TAIL, L).
```

- Cancellazione di tutti gli elementi uguali a T dalla lista:

```
(cl4)    delete(T, [], []).
(cl5)    delete(T, [T | TAIL], L):-
                    delete(T, TAIL, L).
(cl6)    delete(T,[HEAD|TAIL],[HEAD|L]):-
                    T\=HEAD, delete(T, TAIL, L).
```

Usi del CUT – Rimozione di Elementi da Liste

- Cancellazione di un elemento uguale a T dalla lista: le clausole **(cl2)** e **(cl3)** devono essere mutuamente esclusive.

- La condizione di mutua esclusione è l'unificazione dell'elemento da cancellare con la testa della lista

```
(cl1)    delete1(T, [], []).
(cl2')   delete1(T, [T|TAIL], TAIL):- !.
(cl3)    delete1(T, [HEAD|TAIL], [HEAD|L]):-
                                  delete1(T, TAIL, L).
```

Usi del CUT – Rimozione di Elementi da Liste

- Cancellazione di tutti gli elementi uguali a T dalla lista: le clausole **(cl5)** e **(cl6)** devono essere mutuamente esclusive.

- La condizione di mutua esclusione è l'unificazione dell'elemento da cancellare con la testa della lista

```
(cl4)    delete(T, [], []).
(cl5)    delete(T, [T | TAIL], L):- !,
                 delete(T, TAIL, L).
(cl6)    delete(T,[HEAD|TAIL],[HEAD|L]):-
                                 delete(T, TAIL, L).
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

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).
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esempio: Intersezione di Insiemi

- Riprendiamo l'esempio dell'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 delle non mutua esclusione tra la seconda e la terza clausola*

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

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];
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

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

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

```
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).
```

```
% Esempio:
?- union([1,2],[1,3],Z).
Z=[2,1,3] ;
Z=[1,2,1,3];
no
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esempio: Unione di Insiemi

- **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'e' mutua esclusione tra la seconda e la terza clausola.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Predicati built-in per la verifica del "tipo" di un termine

- Determinare, dato un termine T, se T è un atomo, una variabile o una struttura composta.
  - **`atom(T)`** "T è un atomo non numerico"
  - **`number(T)`** "T è un numero (intero o reale)"
  - **`integer(T)`** "T è un numero intero"
  - **`atomic(T)`** "T è un'atomo oppure un numero (ossia T non è una struttura composta)"
  - **`var(T)`** "T è una variabile non istanziata"
  - **`nonvar(T)`** "T non è una variabile“

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esercizio – SLD e CUT

Si consideri il seguente programma Prolog:

```
intersection([],Y,[]).                    (c1)
intersection([X|More],Y,[X|Z]):-
   member(X,Y),
   intersection(More,Y,Z).                (c2)
intersection([X|More],Y,Z):-intersection(More,Y,Z).  (c3)
```

Si rappresenti l'albero SLD relativo al goal

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

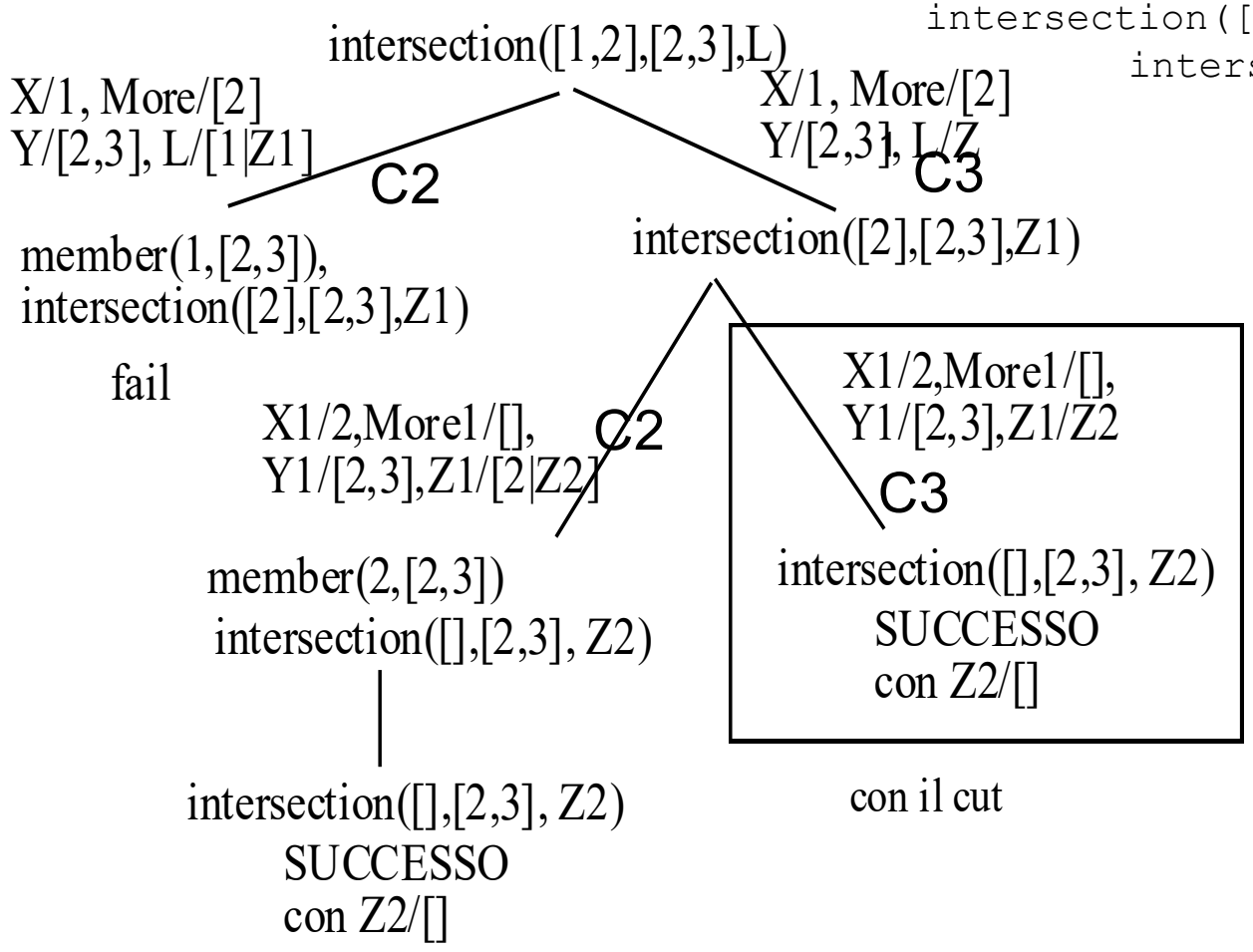
e si indichino i rami di successo. **Si indichi come l'utilizzo del cut (!) possa portare alla definizione corretta del predicato intersezione.**

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esercizio – SLD e CUT – Soluzione

```
intersection([],Y,[]).                    (c1)
intersection([X|More],Y,[X|Z]):-
        member(X,Y),
        intersection(More,Y,Z).  (c2)

intersection([X|More],Y,Z):-
        intersection(More,Y,Z).  (c3)
```



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esercizio – Esame 11 Settembre 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]
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esercizio – Esame 11 Settembre 2008 – Soluzione 1

```
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).
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esercizio – Esame 11 Settembre 2008 – Soluzione 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).
```

Esercizio – Albero SLD

Albero SLD per goal:

:-no_dupl([a,b,b],Y)

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

(non espandere le chiamate a member)

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Esercizio – Albero SLD – ?-no_dupl([a,b,b],Y).

Tree =