Logik für Informatiker: PROLOG
Part 6: Cuts & Negation

Andreas Karwath

&

Wolfram Burgard

(original slides by Peter Flach)

Cut

- **Definition:**
  „`!` succeeds immediately and commits Prolog to all the choices since the parent goal was unified with the head of the clause the cut occurs in.“ (Sterling/Shapiro: The Art of Prolog, p.158)

- ```
  p(X) :- q(X, Y), !, r(X), s(Y, Z).
  p(X) :- q(Y, X).
  ```

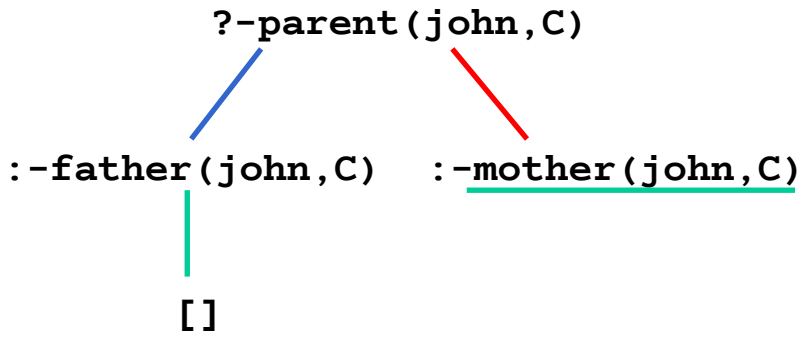
- All clauses of a predicate following a clause with a cut in its body are not considered in a proof, once the cut succeeds (is proven)

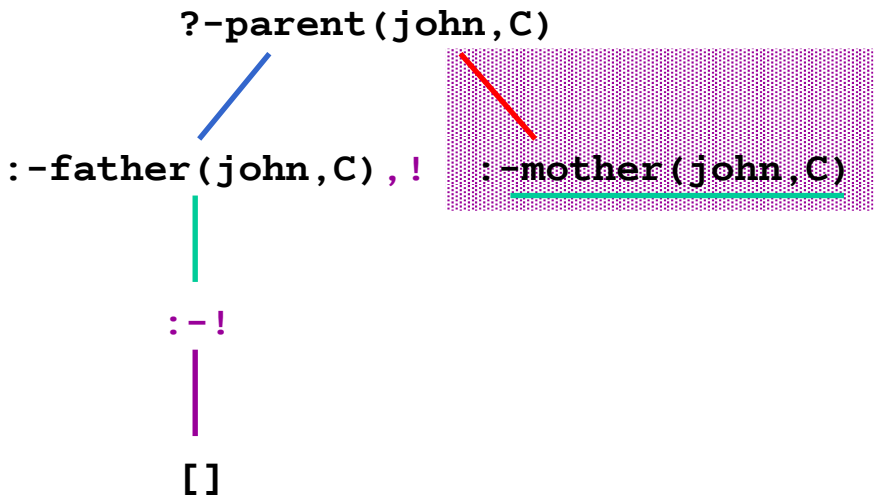
- Alternative solutions for the subgoals preceding the cut cannot be found anymore.

- The cut does not influence the subgoals within the same clause after the cut.

Cut
  ```

```
parent(X,Y):-father(X,Y).
parent(X,Y):-mother(X,Y).
father(john,paul).
mother(mary,paul).
```






```
parent(X,Y):-father(X,Y),!.
parent(X,Y):-mother(X,Y).
father(john,paul).
mother(mary,paul).
```


Pruning by means of cut

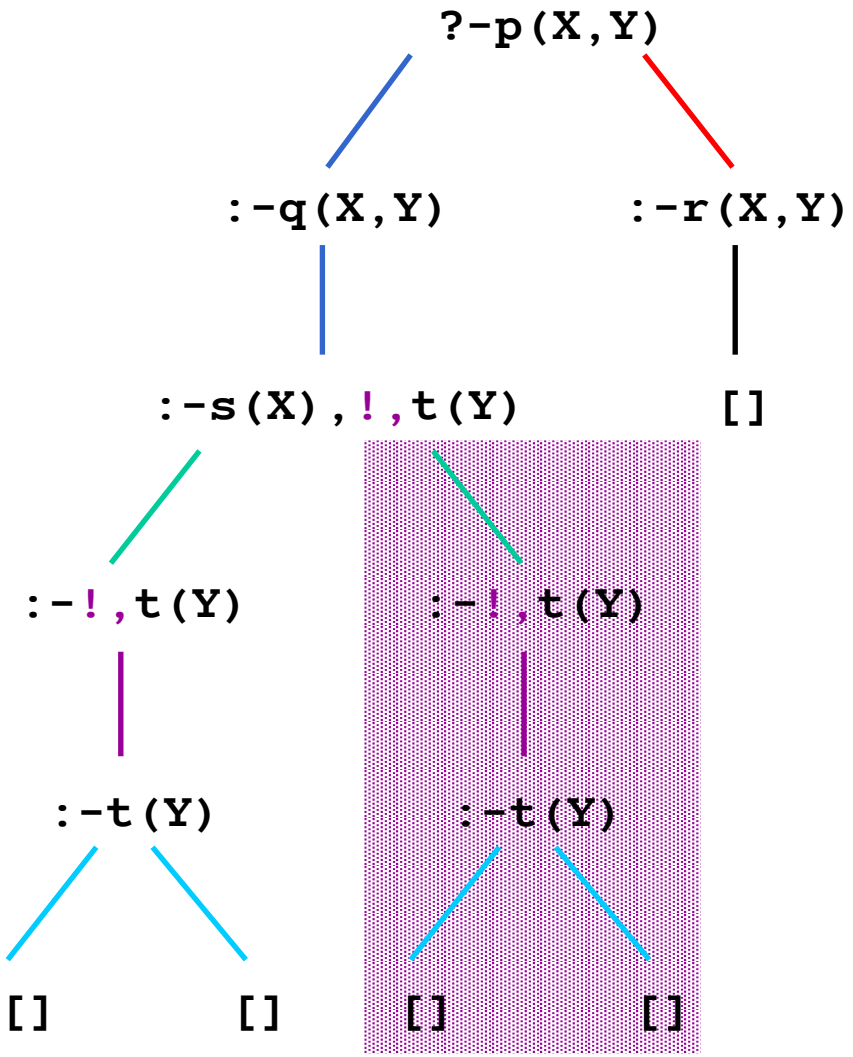
```
p(X,Y):-q(X,Y).
p(X,Y):-r(X,Y).

q(X,Y):-s(X),!,t(Y).

r(c,d).

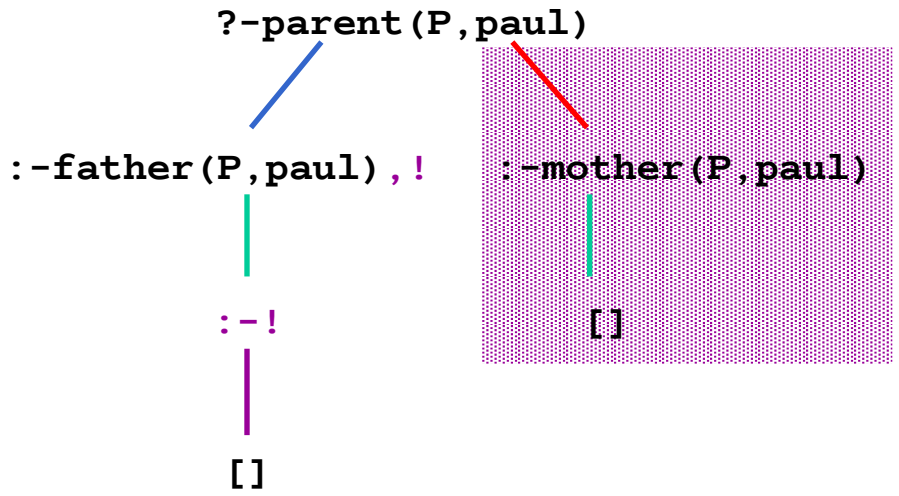
s(a).
s(b).

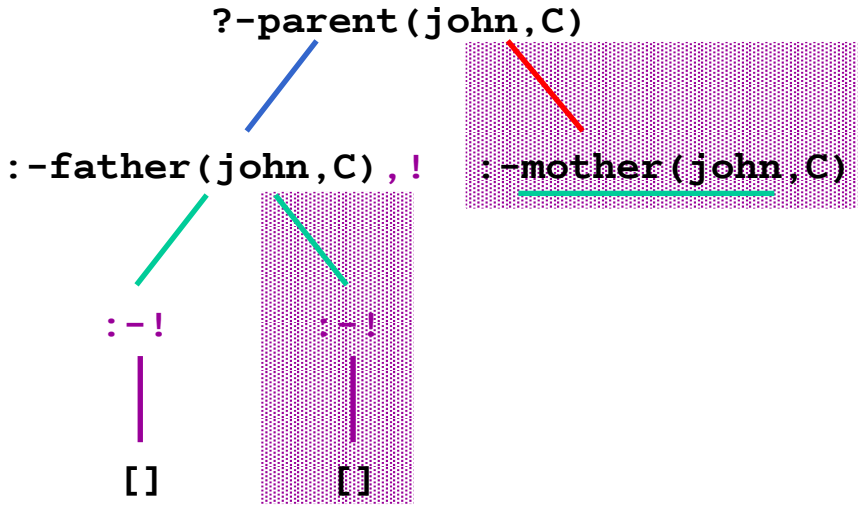
t(a).
t(b).
```



The effect of cut

```
parent(X,Y):-father(X,Y),!.
parent(X,Y):-mother(X,Y).
father(john,paul).
mother(mary,paul).
```






```
parent(X,Y):-father(X,Y),!.
parent(X,Y):-mother(X,Y).
father(john,paul).
father(john,peter).
mother(mary,paul).
mother(mary,peter).
```


Pruning away success branches

- Reduces the search space by dynamic pruning of the search tree
- Can be used to prevent the Prolog system from searching parts of the search space that are known to contain no solutions.
- Green cuts do not change the meaning of the program, red cuts do (to be avoided!)

Cut

Example for reasonable use of green cuts: Definition of relation `polynomial(Term, X)` for recognizing whether a term is a polynomial in `X`.

Useful for distinguishing cases that exclude each other

Reasonable use of green cuts

```
polynomial(X,X) :- !.
polynomial(Term,X) :- constant(Term),!.
polynomial(Term1+Term2,X) :-
          !,polynomial(Term1,X),
          polynomial(Term2,X).
polynomial(Term1-Term2,X) :-
          !,polynomial(Term1,X),
          polynomial(Term2,X).
polynomial(Term1*Term2,X) :-
          !,polynomial(Term1,X),
          polynomial(Term2,X).
polynomial(Term1/Term2,X) :-
          !,polynomial(Term1,X),
          constant(Term2).
polynomial(Term^N,X) :-
          !,natural_number(N),
          polynomial(Term,X).
```


Reasonable use of green cuts

- Red cuts: explicitly omitting conditions = changing the meaning of a program
- Thus, problematic; also most common source of bugs in Prolog programs
- An (almost) reasonable use: definition of relation

```
if_then_else(P,Q,R)
```


An (almost) reasonable use of red cuts

Negation as Failure

- Consider:

```
p :- q, !, r.
p:- s.
```

  means roughly:

```
p :- q, r.
p :- not_q, s.
```

  Define `not_q`:

```
not_q :- q, !, fail.
not_q.
```

- Not very practical. Better:

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

- `not/1, call/1` are meta-predicates.

Negation as failure

Draw an SLD tree for the query:

```
?- likes(A, B)
```

given the following program:

```
likes(peter, Y) :- friendly(Y).
likes(T, S) :- student_of(S, T).
student_of(maria, peter).
student_of(paul, peter).
friendly(maria).
```

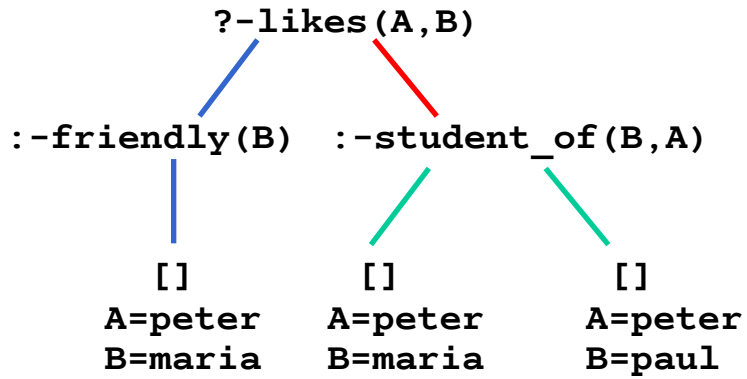
Add a *cut* , in order to „remove“ one of the answers

```
A->peter, B->maria
```

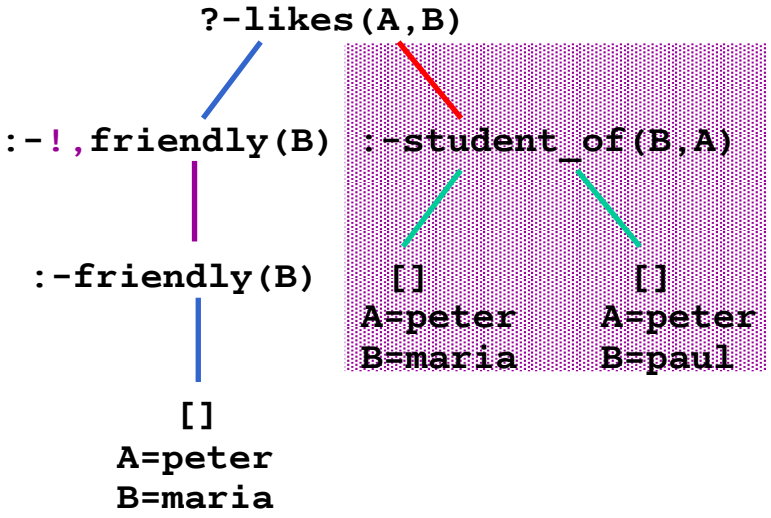
and show the result in the SLD tree. Can this be done without removing the third answer?

Exercise

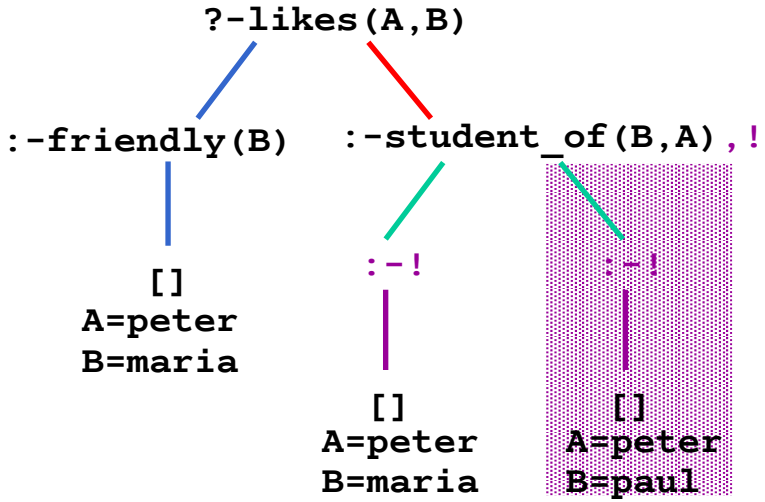
```
likes(peter,Y):-friendly(Y).
likes(T,S):-student_of(S,T).
student_of(maria,peter).
student_of(paul,peter).
friendly(maria).
```




```
likes(peter,Y):-!,friendly(Y).
```



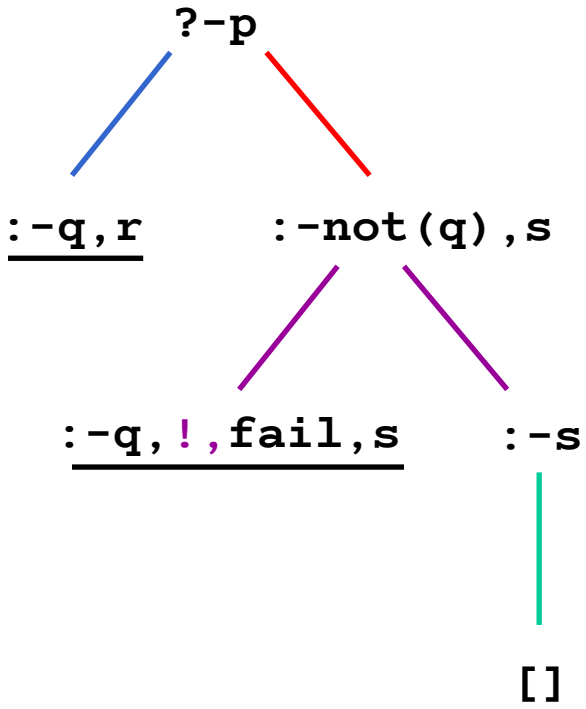

```
likes(T,S):-student_of(S,T),!.
```

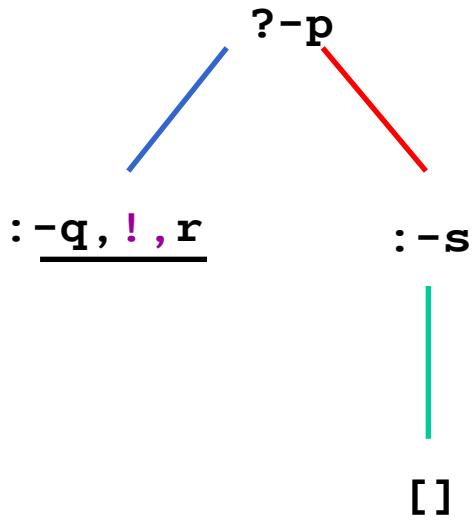


Exercise

```
p:-q,r.
p:-not(q),s.
s.

not(Goal):-Goal,!,fail.
not(Goal).
```



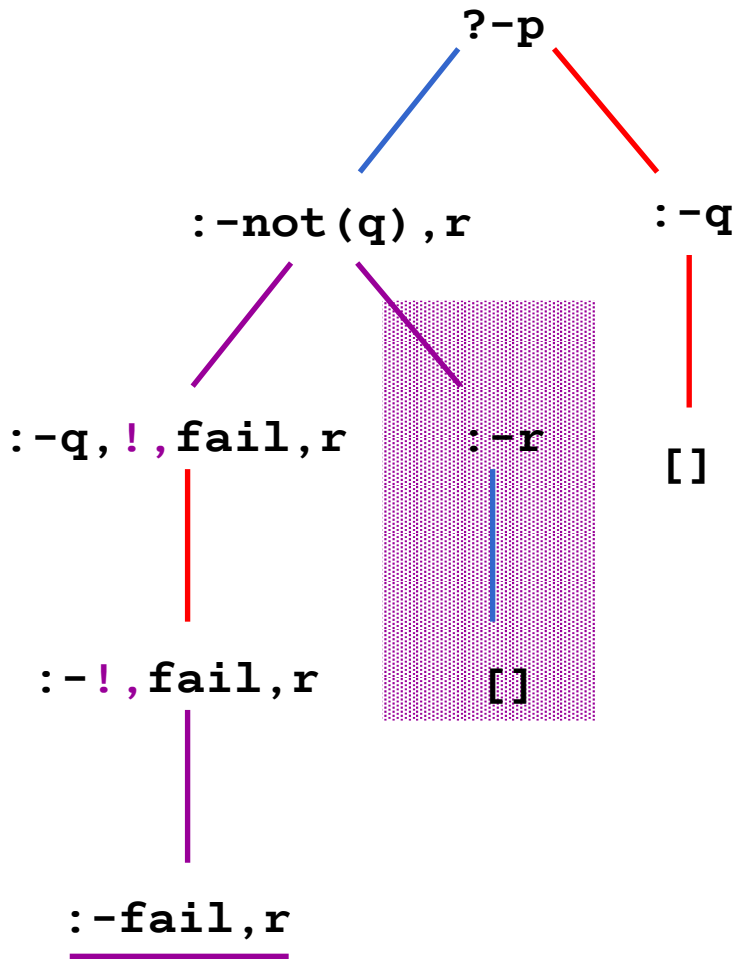



```
p:-q,!,r.
p:-s.
s.
```


not vs. cut

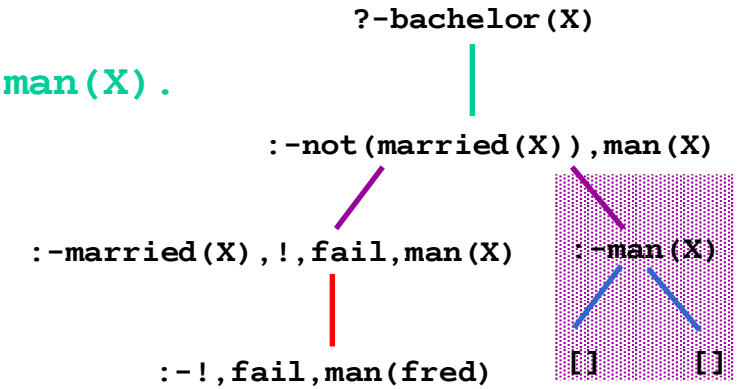
```
p:-not(q),r.
p:-q.
q.
r.

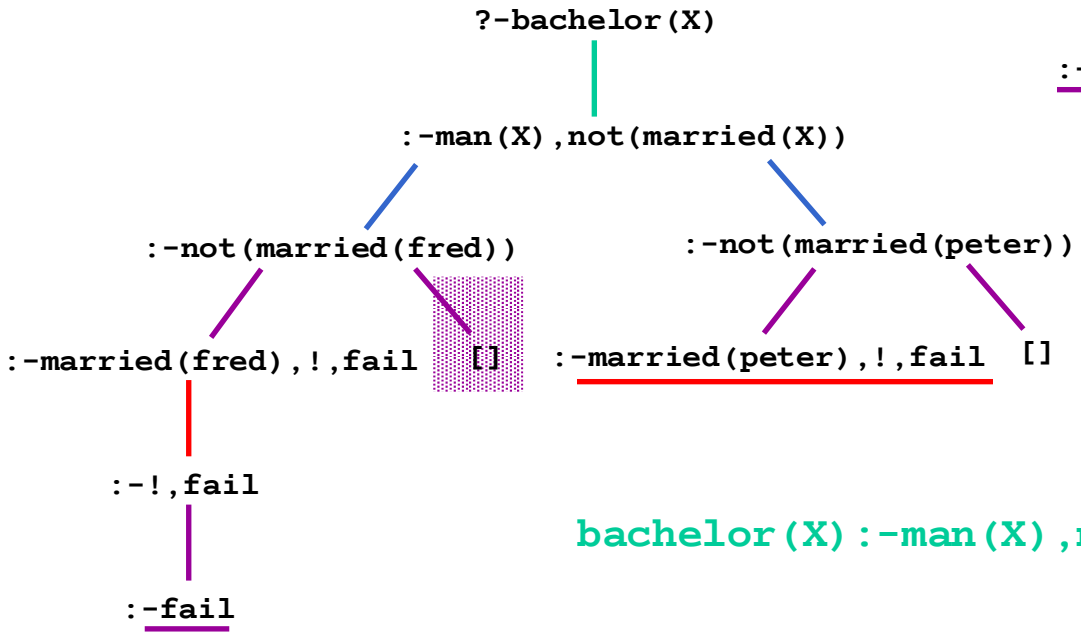
not(Goal):-Goal,!,fail.
not(Goal).
```



:-not(q) fails

```
bachelor(X):-not(married(X)),man(X).
man(fred).
man(peter).
married(fred).
```






```
bachelor(X):-man(X),not(married(X)).
```


Prolog's `not` is unsound

- Red cuts: explicitly omitting conditions = changing the meaning of a program
- Thus, problematic; also most common source of bugs in Prolog programs
- An (almost) reasonable use: definition of relation

```
if_then_else(P,Q,R)
```


An (almost) reasonable use of red cuts

- Declarative meaning: relation is true if `P` and `Q` are true, or if not `P` and `Q` are true.
  Operational meaning: if `P` is shown, then `Q`, else `R`.

```
% if_then_else(P,Q,R) :-
% either P and Q or not P and R
if_then_else(P,Q,R) :- call(P),!,call(Q).
if_then_else(P,Q,R) :- call(R).
```

- Using the cut, it is implicitly known in the second clause, that `P` has failed in the first clause.

An (almost) reasonable use of red cuts

- Syntax: (*if-condition* -> *then-branch*; *else-branch*)
- Only the first solution of *if-condition* is explored.

```
max(Xs, X) :- max(Xs, -inf, X).


max([], Max, Max).
max([X|Xs], MaxSofar, Max) :-
    (X > MaxSofar ->
        max(Xs, X, Max)
    ;
        max(Xs, MaxSofar, Max)
    ).
```


Useful programming device: if-then-else

```
p :- q, r, s, !, t.
p :- q, r, u.
q.
r.
u.
```

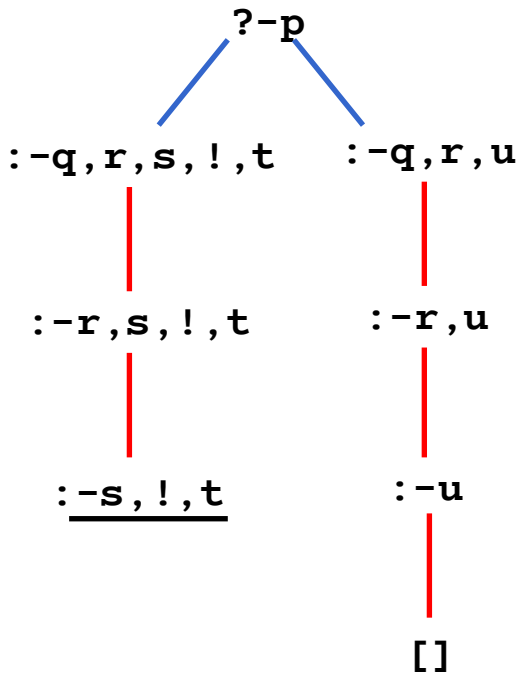
Show that `?- p` succeeds, but that `q` und `r` are used twice.

```
p :- q, r, if_s_then_t_else_u.
if_s_then_t_else_u :- s, !, t.
if_s_then_t_else_u :- u.
```

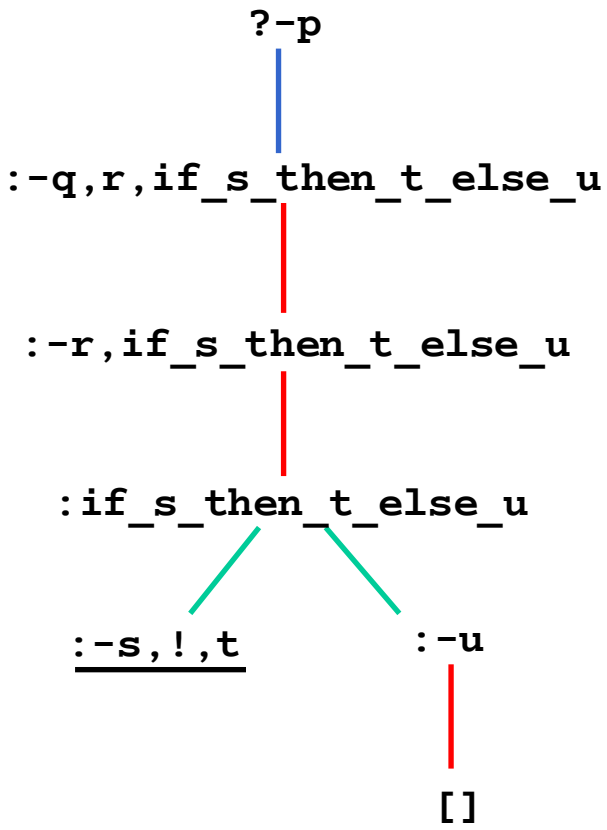
Show that `q` and `r` are used exactly once.

Exercise

```
p:-q,r,s,!,t.
p:-q,r,u.
q.
r.
u.
```




```
p:-q,r,if_s_then_t_else_u.
if_s_then_t_else_u:-s,!,t.
if_s_then_t_else_u:-u.
q.
r.
u.
```



Exercise