

# SLD Resolution, Cut, Negation as Failure, Arithmetic

Adapted from slides provided by Peter Flach  
for his book *Simply Logical*

Unification

# Recap... Substitutions

- **Example:**

```
likes(X,Y) :- dogPerson(X), dog(Y).
```

# Recap... Substitutions

- **Example:**

`likes(X,Y) :- dogPerson(X), dog(Y).`

`S = {X/alice, Y/barney}`

# Recap... Substitutions

- **Example:**

```
likes(X,Y) :- dogPerson(X), dog(Y).
```

```
S = {X/alice, Y/barney}
```

```
likes(alice, barney) :- dogPerson(alice), dog(barney)
```

# Unification

- (We need unification to perform resolution.)
- **Unifier:** Given two atoms or terms  $A$  and  $B$  with disjoint (!) sets of variables, their **unifier** is a substitution  $\theta$  such that  $A\theta = B\theta$ .

# Unification

- (We need unification to perform resolution.)
- **Unifier:** Given two atoms or terms  $A$  and  $B$  with disjoint (!) sets of variables, their **unifier** is a substitution  $\Theta$  such that  $A\Theta = B\Theta$ .
- A unifier  $\Theta$  is more general (*technically speaking, we should be saying more or equally general*) than a unifier  $\Theta'$  iff there exists a substitution  $\sigma$  such that  $\Theta' = \Theta \sigma$ .

# Unification - Example

- ...A unifier  $\Theta$  is more general (*technically speaking, we should be saying more or equally general*) than a unifier  $\Theta'$  iff there exists a substitution  $\sigma$  such that  $\Theta' = \Theta \sigma$ . If  $\Theta$  is more general than  $\Theta'$  and not vice versa, then  $\Theta$  is strictly more general than  $\Theta'$ .

## Example:

A = studies(X, Y)

B = studies(alice, Z)

$\Theta = \{X/\text{alice}, Y/Z\}$

$\Theta' = \{X/\text{alice}, Y/\text{bob}, Z/\text{bob}\}$

$\Theta' = \Theta \sigma$ , where  $\sigma = \{Z/\text{bob}\}$



# Unification – Most General Unifier

- **Most General Unifier:** Given two atoms or terms  $A$  and  $B$  with disjoint sets of variables, their **most general unifier (MGU)** is a unifier  $\theta$  such that there is no strictly more general unifier of  $A$  and  $B$ .

- **Example:**

$A = \text{studies}(X, Y)$

$B = \text{studies}(\text{alice}, Z)$

$\theta = \{X/\text{alice}, Y/Z\}$

Here,  $\theta$  is an MGU of  $A$  and  $B$ .

# The Hebrand Unification Algorithm

(Adapted from Pierre M. Nugues: Language Processing with Perl and Prolog)

- **Initialization:**  $\Theta = \{\}$ , `failure = false`, **push**  $A = B$  on the stack

- **Loop:**

- 1: **Pop**  $x = y$  from the stack

- 2: **If**  $x$  and  $y$  are constants and  $x == y$  **then continue.**

- 3: **Else if**  $x$  is a variable and  $x$  does not appear in  $y$  **then** /\* “Occurs check” \*/

- 4:           Replace  $x$  with  $y$  in the stack and in  $\Theta$ . Add the substitution  $x/y$  to  $\Theta$ .

- 5: **Else if**  $x$  is a variable and  $x == y$  **then continue.**

- 6: **Else if**  $y$  is a variable and  $x$  is not a variable **then push**  $y = x$  on the stack.

- 7: **Else if**  $x$  and  $y$  are compound and  $x$  is  $f(t_1, \dots, t_k)$  and  $y$  is  $f(u_1, \dots, u_k)$  **then**

- 8:           **Push** on the stack:  $t_1 = u_1$ ,  $t_2 = u_2$ , ...,  $t_k = u_k$ .

- 9: **Else set** `failure = true`,  $\Theta = \{\}$  and **return.**

**Until** stack is empty.

# Example

$$f(g(X, h(X, b)), Z) = f(g(a, Z), Y)$$

(Adapted from Pierre M. Nugues: Language Processing with Perl and Prolog)

**Stack:**

$f(g(X, h(X, b)), Z) = f(g(a, Z), Y)$  /\* In Prolog, "=" means "unify" \*/

$\Theta = \{\}$

# Example

$$f(g(X, h(X, b)), Z) = f(g(a, Z), Y)$$

(Adapted from Pierre M. Nugues: Language Processing with Perl and Prolog)

**Stack:**

$$g(X, h(X, b)) = g(a, Z)$$

$$Z = Y$$

$$\Theta = \{\}$$

# Example

$$f(g(X, h(X, b)), Z) = f(g(a, Z), Y)$$

(Adapted from Pierre M. Nugues: Language Processing with Perl and Prolog)

**Stack:**

$$X = a$$

$$h(X, b) = Z$$

$$Z = Y$$

$$\Theta = \{\}$$

# Example

$$f(g(X, h(X, b)), Z) = f(g(a, Z), Y)$$

(Adapted from Pierre M. Nugues: Language Processing with Perl and Prolog)

**Stack:**

$$h(a, b) = Z$$

$$Z = Y$$

$$\Theta = \{X/a\}$$

# Example

$$f(g(X, h(X, b)), Z) = f(g(a, Z), Y)$$

(Adapted from Pierre M. Nugues: Language Processing with Perl and Prolog)

**Stack:**

$$Z = h(a, b)$$

$$Z = Y$$

$$\Theta = \{X/a\}$$

# Example

$$f(g(X, h(X, b)), Z) = f(g(a, Z), Y)$$

(Adapted from Pierre M. Nugues: Language Processing with Perl and Prolog)

**Stack:**

$$h(a, b) = Y$$

$$\Theta = \{X/a, Z/h(a, b)\}$$



# Example

$$f(g(X, h(X, b)), Z) = f(g(a, Z), Y)$$

(Adapted from Pierre M. Nugues: Language Processing with Perl and Prolog)

**Stack:**

$$Y = h(a, b)$$

$$\Theta = \{X/a, Z/h(a, b)\}$$

# Example

$$f(g(X, h(X, b)), Z) = f(g(a, Z), Y)$$

(Adapted from Pierre M. Nugues: Language Processing with Perl and Prolog)

**Stack:**

EMPTY

$$\Theta = \{X/a, Z/h(a, b), Y/h(a, b)\}$$

**Result of unification - MGU:**

$$f(g(a, h(a, b)), h(a, b))$$

# Example

$$f(g(X, h(X, b)), Z) = f(g(a, Z), Y)$$

The screenshot shows the SWISH web interface. The browser address bar displays `swish.swi-prolog.org`. The page title is "SWISH -- SWI-Prolog for SHaring". The interface includes a menu bar with "File", "Edit", "Examples", and "Help", a search bar, and a notification bell showing "25". A "Program" tab is active, showing a list of line numbers (1-4) on the left. The main area displays a Prolog program and its solution, both highlighted with a red circle:

```
f(g(X, h(X, b)), Z) = f(g(a, Z), Y)
```

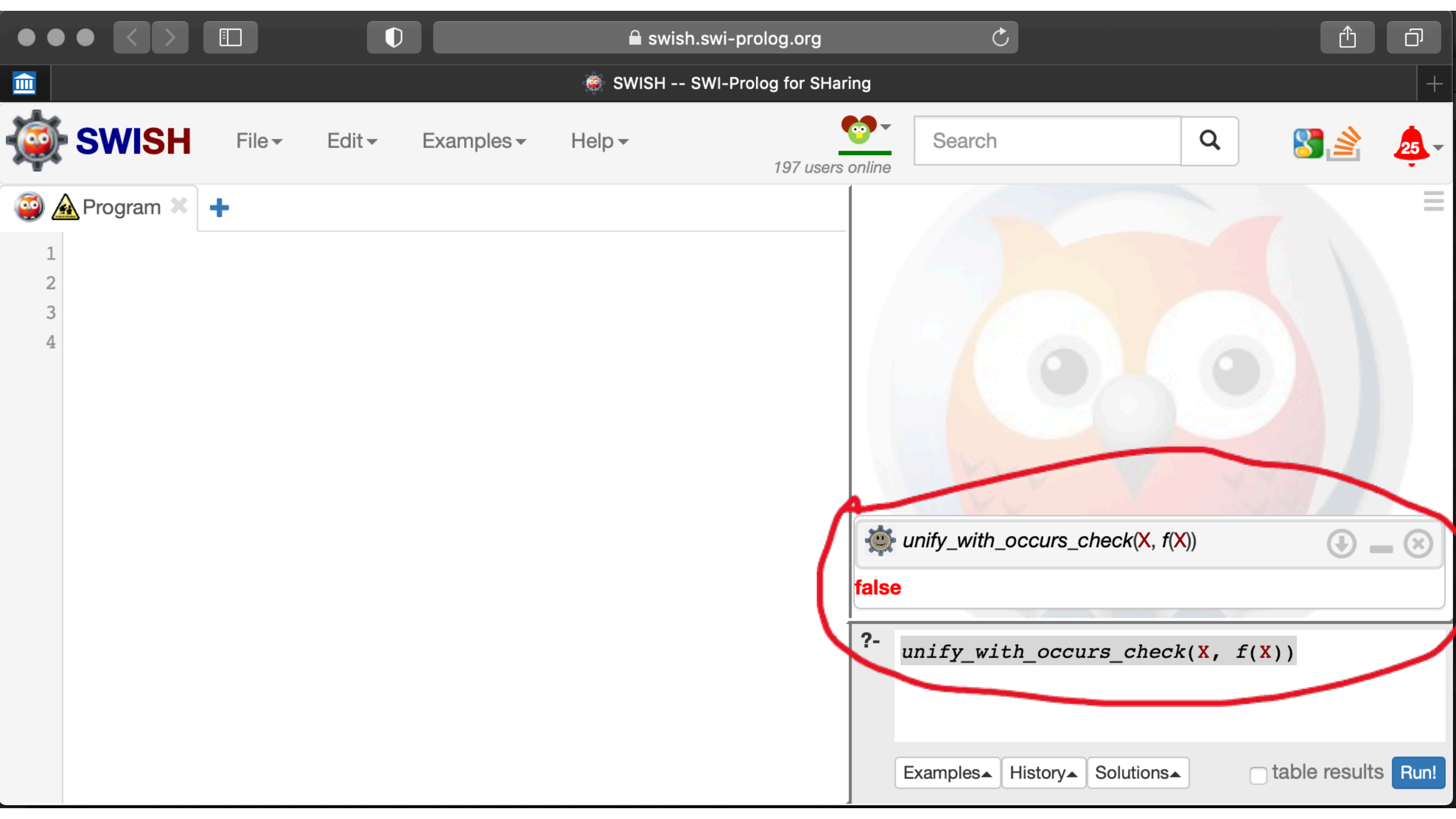
**X = a,**  
**Y = Z, Z = h(a, b)**

?- `f(g(X, h(X, b)), Z) = f(g(a, Z), Y)`




At the bottom, there are buttons for "Examples", "History", "Solutions", a checkbox for "table results", and a "Run!" button.

# The “Occurs Check”

- Line 3: **Else if**  $x$  is a variable and  $x$  does not appear in  $y$  **then** ...
- This is not implemented in many Prologs for efficiency reasons and programmers have to take care of it themselves. Without occurs check, Prolog inference is not really sound (this is a tradeoff for efficiency).
- If you want sound unification, use `unify_with_occurs_check/2`



1  
2  
3  
4

 `unify_with_occurs_check(X, f(X))`    
**false**

?- `unify_with_occurs_check(X, f(X))`



Program x +

1  
2  
3  
4

unify\_with\_occurs\_check(X, f(X))

**false**

X = f(X)

**X = f(X)**

?- X = f(X)

# SLD Resolution

# Order of Processing Goals

## Informally:

- Goals are processed from “left to right”:

`: - follows(S,C), teaches(peter,C)`  
          1                                  2

- ...and we try to resolve them with rules from the program going top down

```
follows(alice,prolog). 1
follows(bob,planning). 2
follows(bob,prolog). 3
```



# SLD-trees

```
student_of(X,T):-follows(X,C),teaches(T,C).  
follows(paul,computer_science).  
follows(paul,expert_systems).  
follows(maria,ai_techniques).  
teaches(adrian,expert_systems).  
teaches(peter,ai_techniques).  
teaches(peter,computer_science).
```

?-student\_of(S,peter)

# SLD-trees

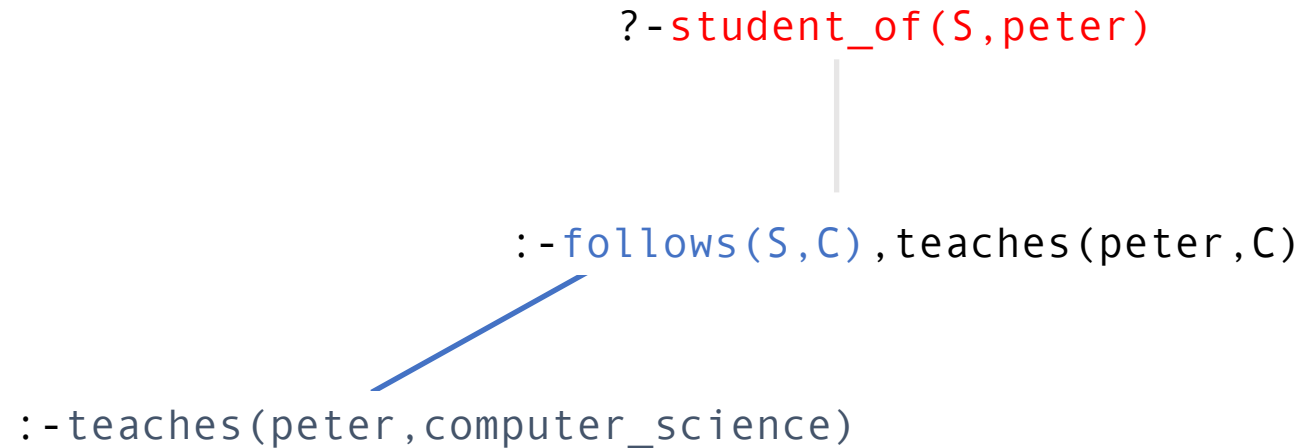
```
student_of(X,T):-follows(X,C),teaches(T,C).  
follows(paul,computer_science).  
follows(paul,expert_systems).  
follows(maria,ai_techniques).  
teaches(adrian,expert_systems).  
teaches(peter,ai_techniques).  
teaches(peter,computer_science).
```

?-student\_of(S,peter)

:-follows(S,C),teaches(peter,C)

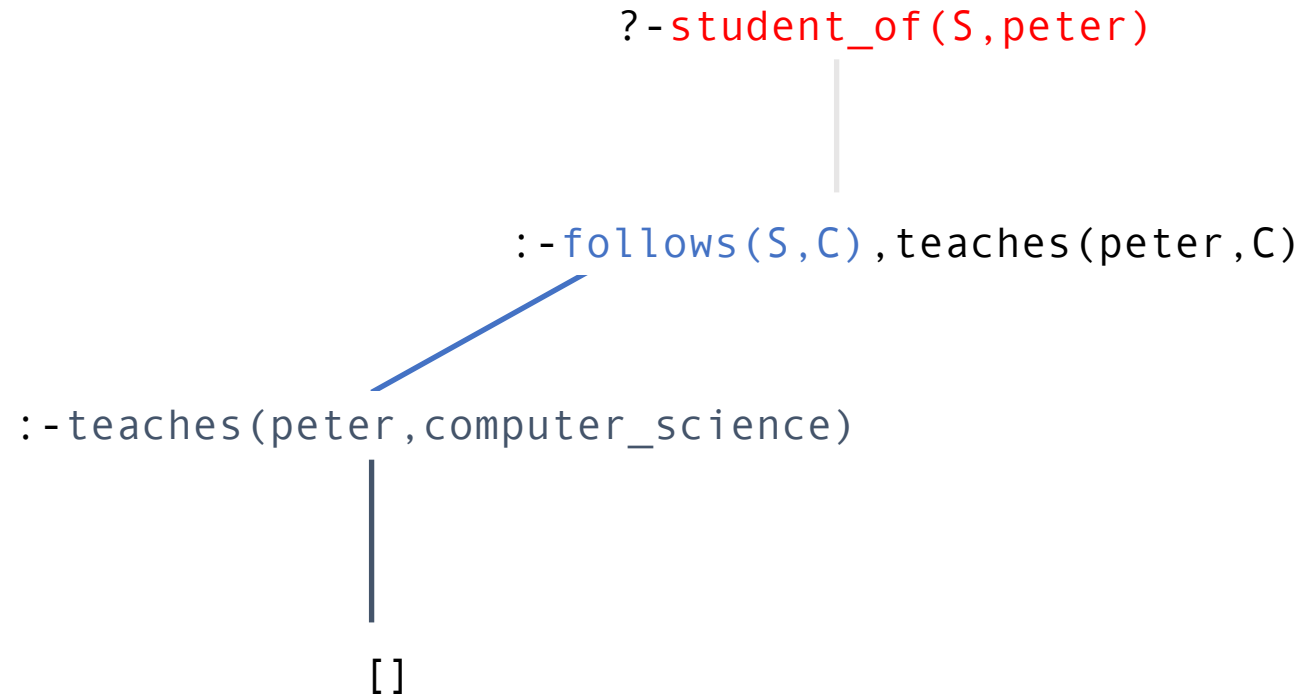
# SLD-trees

```
student_of(X,T):-follows(X,C),teaches(T,C).  
follows(paul,computer_science).  
follows(paul,expert_systems).  
follows(maria,ai_techniques).  
teaches(adrian,expert_systems).  
teaches(peter,ai_techniques).  
teaches(peter,computer_science).
```



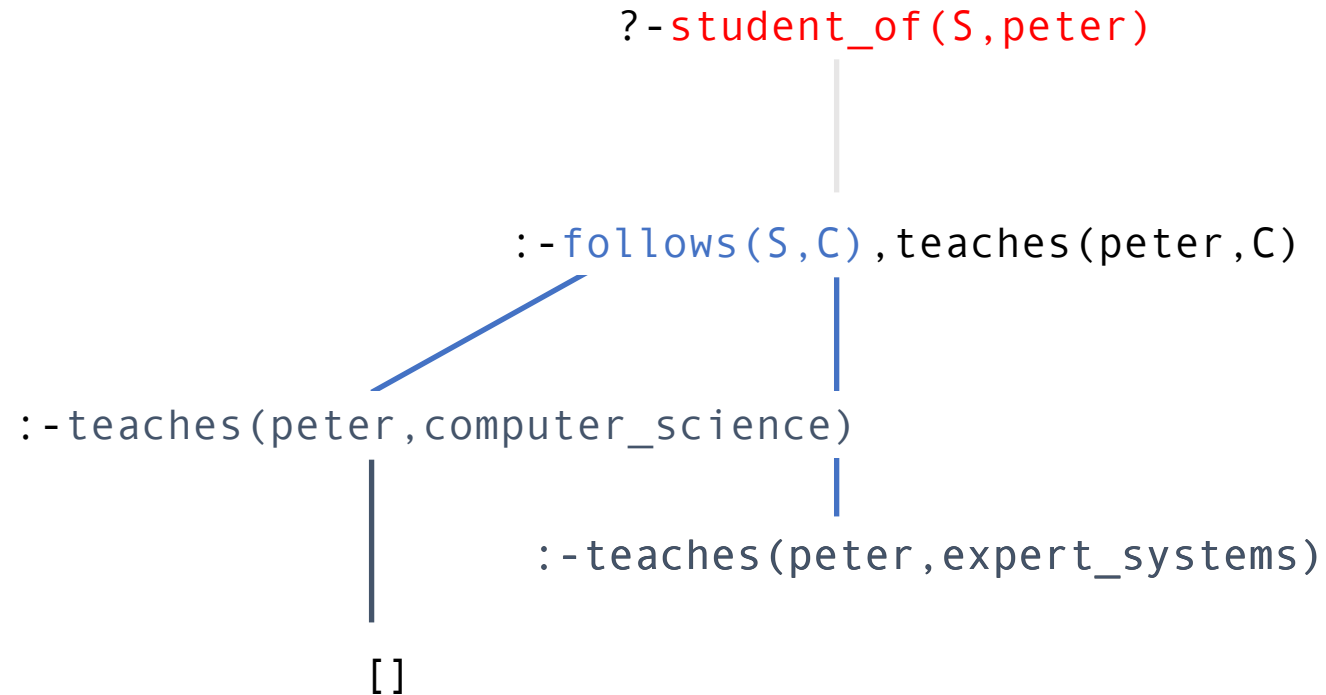
# SLD-trees

```
student_of(X,T):-follows(X,C),teaches(T,C).  
follows(paul,computer_science).  
follows(paul,expert_systems).  
follows(maria,ai_techniques).  
teaches(adrian,expert_systems).  
teaches(peter,ai_techniques).  
teaches(peter,computer_science).
```



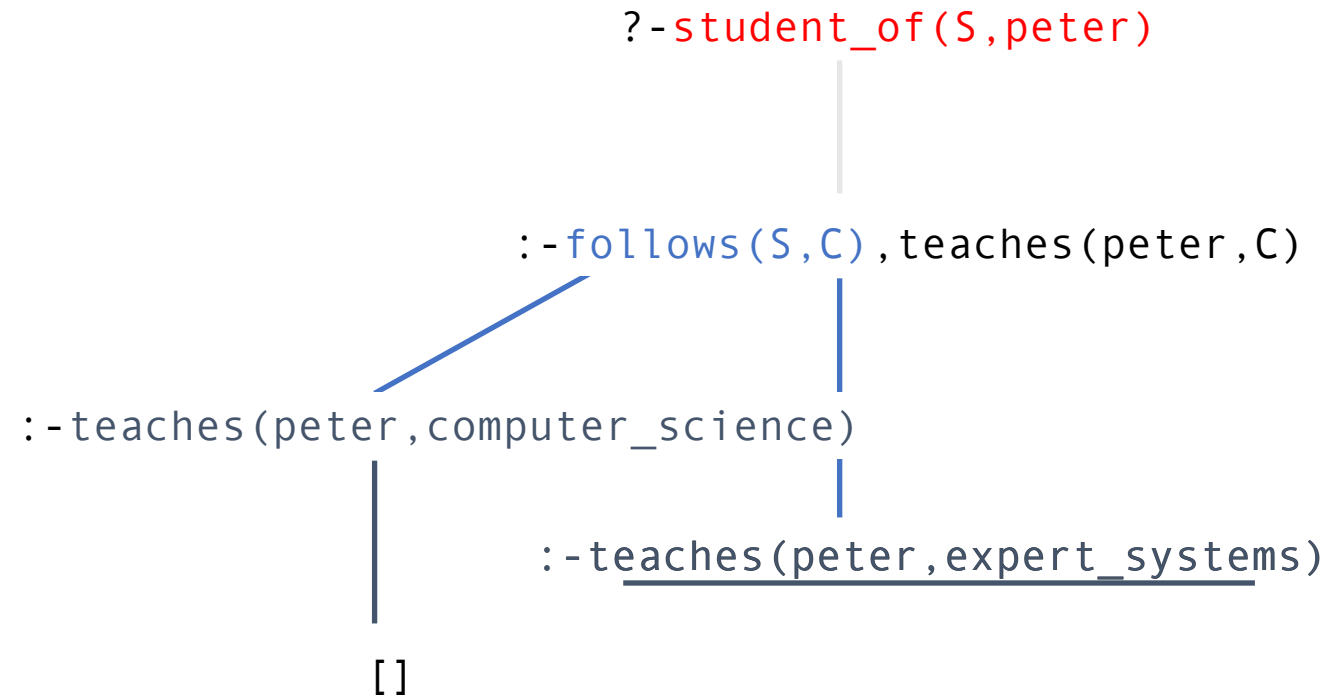
# SLD-trees

```
student_of(X,T):-follows(X,C),teaches(T,C).  
follows(paul,computer_science).  
follows(paul,expert_systems).  
follows(maria,ai_techniques).  
teaches(adrian,expert_systems).  
teaches(peter,ai_techniques).  
teaches(peter,computer_science).
```



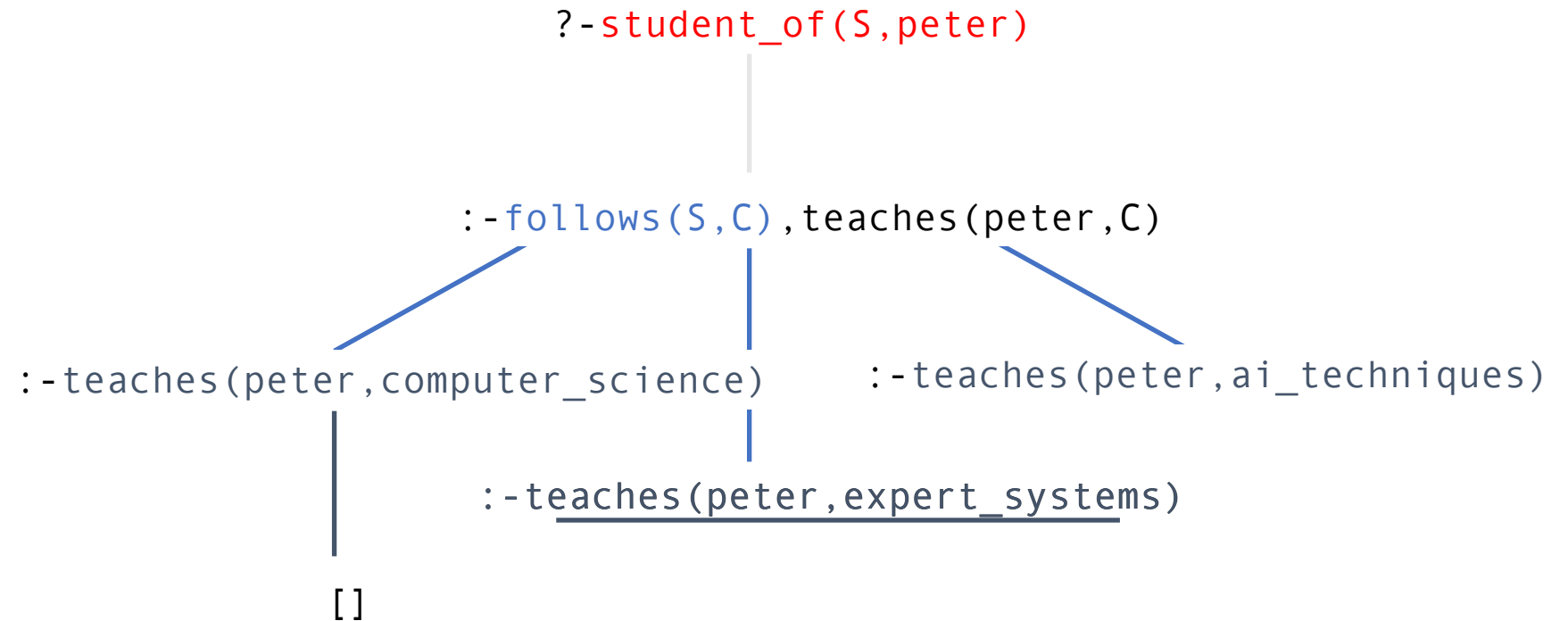
# SLD-trees

```
student_of(X,T):-follows(X,C),teaches(T,C).  
follows(paul,computer_science).  
follows(paul,expert_systems).  
follows(maria,ai_techniques).  
teaches(adrian,expert_systems).  
teaches(peter,ai_techniques).  
teaches(peter,computer_science).
```



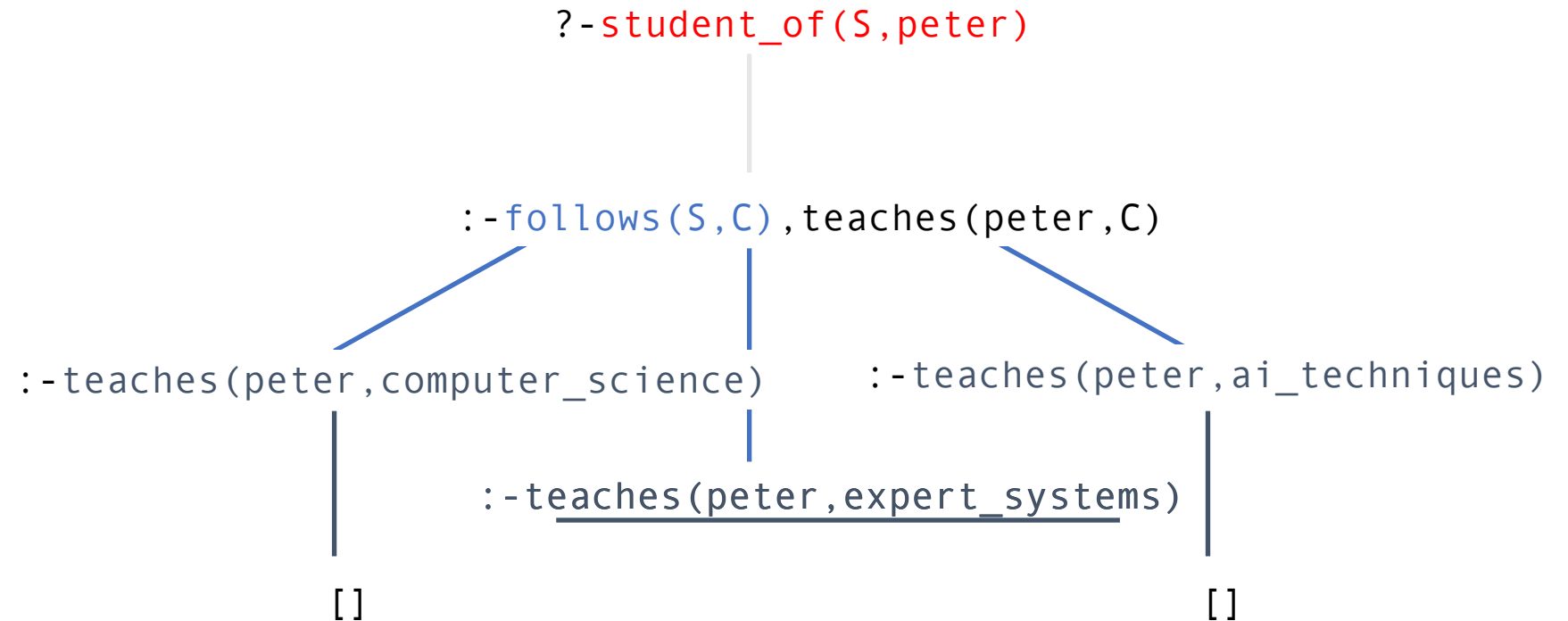
# SLD-trees

```
student_of(X,T):-follows(X,C),teaches(T,C).  
follows(paul,computer_science).  
follows(paul,expert_systems).  
follows(maria,ai_techniques).  
teaches(adrian,expert_systems).  
teaches(peter,ai_techniques).  
teaches(peter,computer_science).
```



# SLD-trees

```
student_of(X,T):-follows(X,C),teaches(T,C).  
follows(paul,computer_science).  
follows(paul,expert_systems).  
follows(maria,ai_techniques).  
teaches(adrian,expert_systems).  
teaches(peter,ai_techniques).  
teaches(peter,computer_science).
```







# Important!

- The order of literals in the body of a rule
- ... as well as the order of rules are important!
  
- Sometimes the order determines efficiency of the program, sometime it determines if the program even finishes!

# Infinite SLD-trees

```
brother_of(X,Y):-brother_of(Y,X).  
brother_of(paul,peter).
```

```
?-brother_of(peter,B)
```

# Infinite SLD-trees

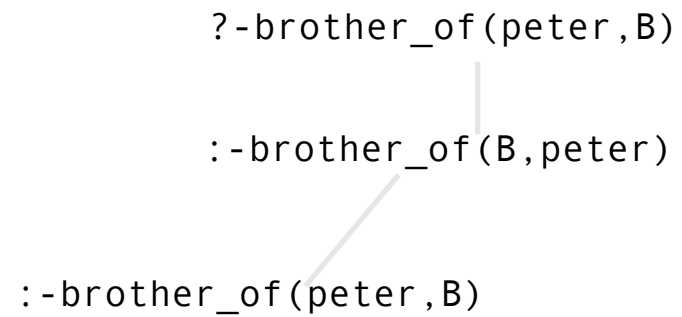
```
brother_of(X,Y):-brother_of(Y,X).  
brother_of(paul,peter).
```

```
?-brother_of(peter,B)  
      |  
:-brother_of(B,peter)
```

# Infinite SLD-trees

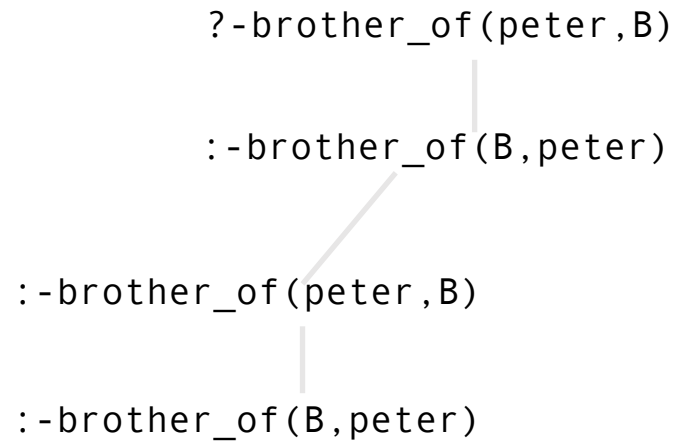
```
brother_of(X,Y):-brother_of(Y,X).  
brother_of(paul,peter).
```

```
?-brother_of(peter,B)  
:-brother_of(B,peter)  
:-brother_of(peter,B)
```



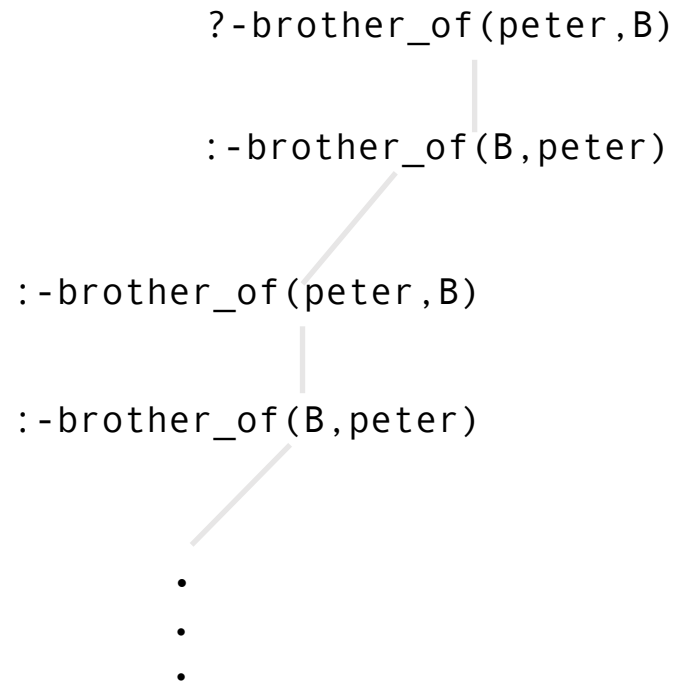
# Infinite SLD-trees

```
brother_of(X,Y):-brother_of(Y,X).  
brother_of(paul,peter).
```



# Infinite SLD-trees

```
brother_of(X,Y):-brother_of(Y,X).  
brother_of(paul,peter).
```







# So Prolog will get stuck...

The screenshot shows the SWISH web interface at `swish.swi-prolog.org`. The main editor contains the following Prolog code:

```
1 brother_of(X,Y):-brother_of(Y,X).  
2 brother_of(paul,peter).  
3
```

The code is circled in green. The execution output on the right shows a stack limit error:

```
brother_of(peter, B)  
Stack limit (0.2Gb) exceeded  
Stack sizes: local: 0.2Gb, global: 16Kb, trail: 1Kb  
Stack depth: 3,288,428, last-call: 50%, Choice points  
: 1,644,210  
In:  
[3,288,428] brother_of(peter, _1552)  
[3,288,427] brother_of(_1576, peter)  
[3,288,425] brother_of(_1602, peter)  
[3,288,423] brother_of(_1628, peter)  
[3,288,421] brother_of(_1654, peter)  
  
Use the --stack_limit=size[KMG] command line option or  
?- set_prolog_flag(stack_limit, 2_147_483_648). to double  
the limit.  
  
?- brother_of(peter, B)
```

# So Prolog will get stuck...

The screenshot shows the SWISH web interface at `swish.swi-prolog.org`. The interface includes a menu bar with "File", "Edit", "Examples", and "Help", a search bar, and a notification bell showing 25 alerts. The main area is divided into a code editor on the left and an execution window on the right.

**Code Editor:**

```
1 brother_of(X,Y):-brother_of(Y,X).  
2 brother_of(paul,peter).  
3
```

**Execution Window:**

`brother_of(peter, B)`

**Stack limit (0.2Gb) exceeded**  
Stack sizes: local: 0.2Gb, global: 16Kb, trail: 1Kb  
Stack depth: 3,288,428, last-call: 50%, Choice points  
: 1,644,210  
In:  
[3,288,428] `brother_of(peter, _1552)`  
[3,288,427] `brother_of(_1576, peter)`  
[3,288,425] `brother_of(_1602, peter)`  
[3,288,423] `brother_of(_1628, peter)`  
[3,288,421] `brother_of(_1654, peter)`

Use the `--stack_limit=size[KMG]` command line option or  
`?- set_prolog_flag(stack_limit, 2_147_483_648).` to double the limit.

`?- brother_of(peter, B)`

# So Prolog will get stuck...

The screenshot shows the SWISH web interface at `swish.swi-prolog.org`. The interface includes a menu bar with 'File', 'Edit', 'Examples', and 'Help', a search bar, and a notification bell showing 25 alerts. A tab titled 'Program' is active, displaying the following Prolog code:

```
1 brother_of(X,Y):-brother_of(Y,X).  
2 brother_of(paul,peter).  
3
```

On the right side, a console window titled 'brother\_of(peter, B)' shows a stack overflow error:

```
Stack limit (0.2Gb) exceeded  
Stack sizes: local: 0.2Gb, global: 16Kb, trail: 1Kb  
Stack depth: 3,288,428, last-call: 50%, Choice points  
: 1,644,210  
In:  
[3,288,428] brother_of(peter, _1552)  
[3,288,427] brother_of(_1576, peter)  
[3,288,425] brother_of(_1602, peter)  
[3,288,423] brother_of(_1628, peter)  
[3,288,421] brother_of(_1654, peter)
```

Below the error message, a suggestion is provided:

```
Use the --stack_limit=size[KMG] command line option or  
?- set_prolog_flag(stack_limit, 2_147_483_648). to double  
the limit.
```

At the bottom of the console, the prompt '?-' is followed by the query:

```
?- brother_of(peter, B)
```

# Infinite SLD-trees

```
brother_of(X,Y):-brother_of(Y,X).  
brother_of(paul,peter).
```

# Infinite SLD-trees



```
brother_of(X,Y):-brother_of(Y,X).  
brother_of(paul,peter).
```

  Program ✕ +

```
1 brother_of(paul,peter).  
2 brother_of(X,Y):-brother_of(Y,X).  
3
```



 `brother_of(peter, B)` ⏴ ⏵ ✕

**B = paul**  
Next 10 100 1,000 Stop

?- `brother_of(peter, B)`



Program x +

```
1 brother_of(paul,peter).  
2 brother_of(X,Y):-brother_of(Y,X).  
3
```



brother\_of(peter, B)

B = paul

Next 10 100 1,000 Stop

?- brother\_of(peter, B)



Program

```
1 brother_of(paul,peter).  
2 brother_of(X,Y):-brother_of(Y,X).  
3
```

brother\_of(peter, B)

B = paul

Next 10 100 1,000 Stop

?- brother\_of(peter, B)



The other branches can still be accessed...

The screenshot shows the SWISH web interface. The browser address bar displays `swish.swi-prolog.org`. The page title is "SWISH -- SWI-Prolog for SHaring". The interface includes a menu with "File", "Edit", "Examples", and "Help". A search bar and a notification bell showing "25" are also present. The main area is divided into a code editor and a results window.

The code editor contains the following Prolog code:

```
1 brother_of(paul,peter).  
2 brother_of(X,Y):-brother_of(Y,X).  
3
```

The results window shows the query `brother_of(peter, B)` and its output:

```
B = paul  
B = paul  
B = paul  
B = paul  
B = paul  
B = paul  
B = paul  
B = paul  
B = paul  
B = paul  
B = paul
```

Below the results, there are buttons for "Next", "10", "100", "1,000", and "Stop". A green handwritten "10x" is next to the "10" button. At the bottom of the results window, there are buttons for "Examples", "History", and "Solutions", along with a checkbox for "table results" and a "Run!" button.

The other branches can still be accessed...

The screenshot shows the SWISH web interface. The browser address bar displays `swish.swi-prolog.org`. The page title is "SWISH -- SWI-Prolog for SHaring". The interface includes a menu with "File", "Edit", "Examples", and "Help". A search bar is present, along with a notification for "183 users online" and a bell icon showing "25" notifications.

The main editor area contains a Prolog program:

```
1 brother_of(paul,peter).  
2 brother_of(X,Y):-brother_of(Y,X).  
3
```

The execution results window shows the query `brother_of(peter, B)` and a list of 10 results, all of which are `B = paul`. These results are circled in orange. Below the list, there are buttons for "Next", "10", "100", "1,000", and "Stop". The number "10x" is written in green next to the "10" button.

The bottom panel shows the query `?- brother_of(peter, B)` and a "Run!" button. There are also buttons for "Examples", "History", and "Solutions". A checkbox for "table results" is present.

# Infinite SLD-trees (Another Example)

```
brother_of(paul,peter).  
brother_of(peter,adrian).  
brother_of(X,Y):-brother_of(X,Z), brother_of(Z,Y).
```

# Infinite SLD-trees (Another Example)

```
brother_of(paul,peter).  
brother_of(peter,adrian).  
brother_of(X,Y):-brother_of(X,Z), brother_of(Z,Y).
```

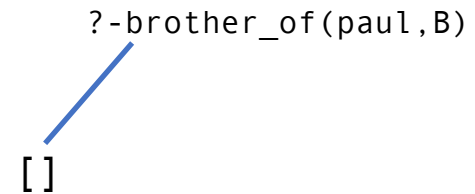
```
?-brother_of(paul,B)
```

# Infinite SLD-trees (Another Example)

```
brother_of(paul,peter).
```

```
brother_of(peter,adrian).
```

```
brother_of(X,Y):-brother_of(X,Z), brother_of(Z,Y).
```

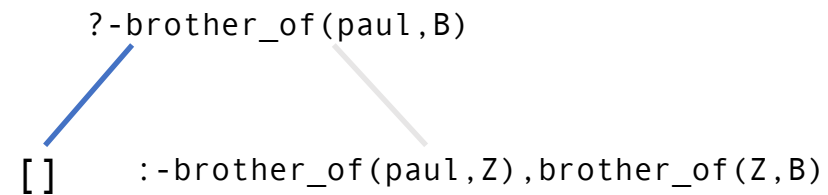


# Infinite SLD-trees (Another Example)

```
brother_of(paul,peter).
```

```
brother_of(peter,adrian).
```

```
brother_of(X,Y):-brother_of(X,Z), brother_of(Z,Y).
```

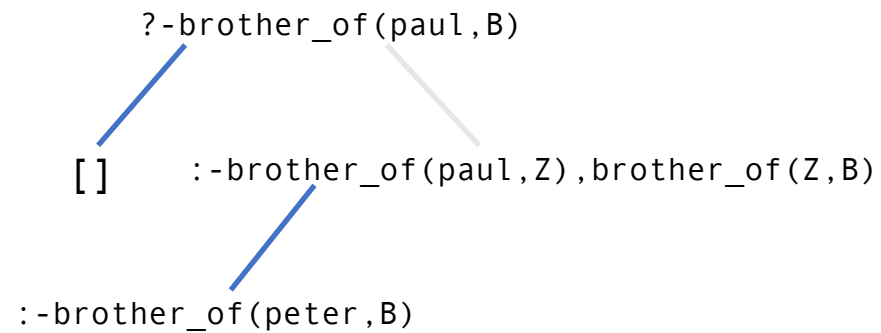


# Infinite SLD-trees (Another Example)

```

brother_of(paul,peter).
brother_of(peter,adrian).
brother_of(X,Y):-brother_of(X,Z), brother_of(Z,Y).

```

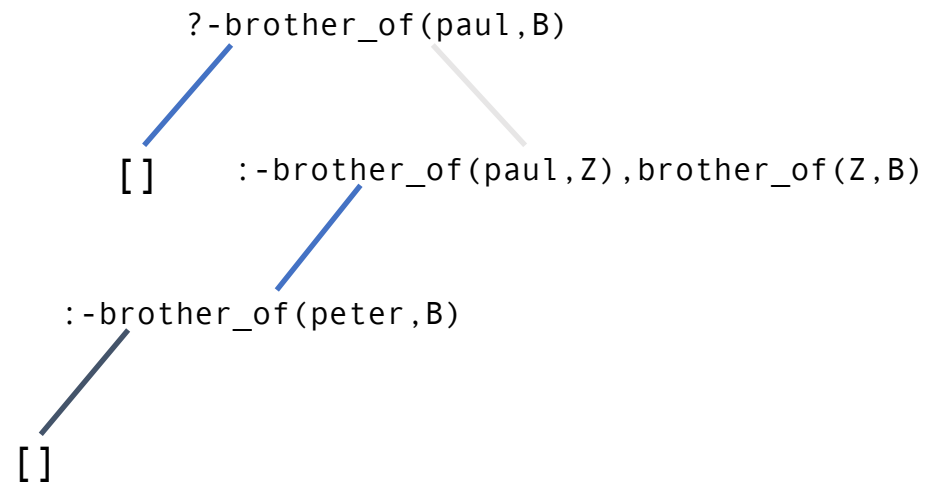


# Infinite SLD-trees (Another Example)

```

brother_of(paul,peter).
brother_of(peter,adrian).
brother_of(X,Y):-brother_of(X,Z), brother_of(Z,Y).

```





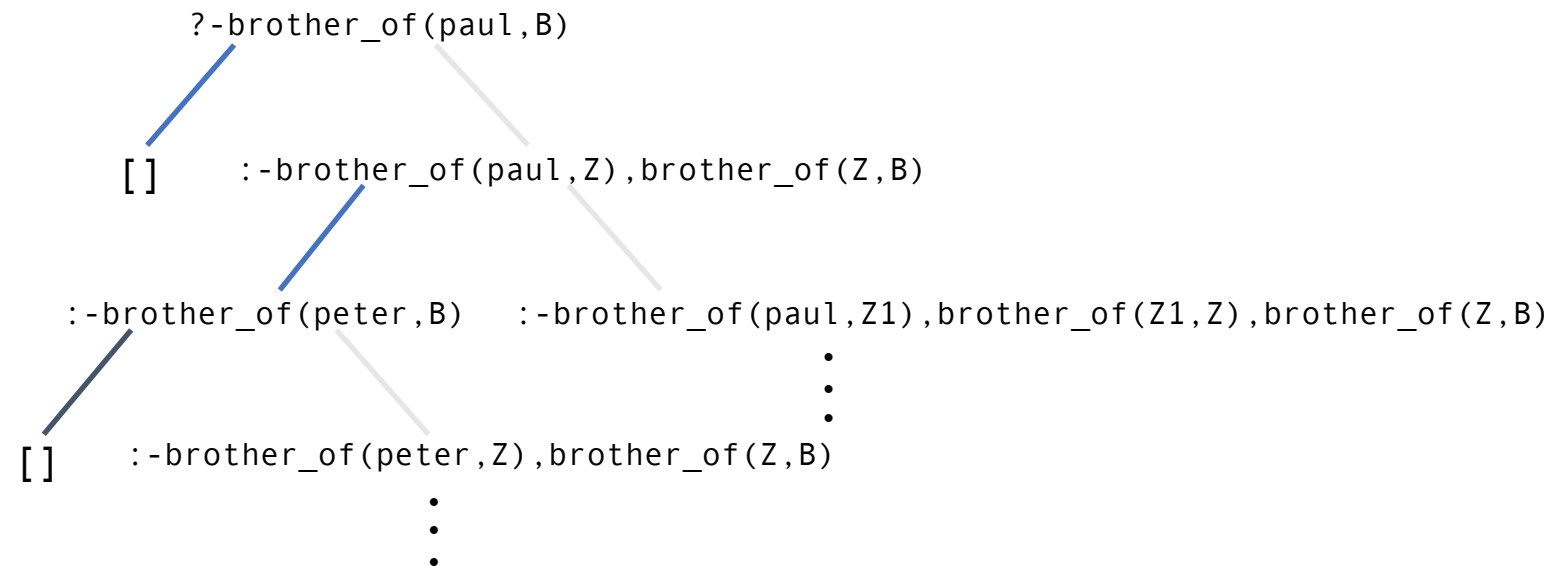


# Infinite SLD-trees (Another Example)

```
brother_of(paul,peter).
```

```
brother_of(peter,adrian).
```

```
brother_of(X,Y):-brother_of(X,Z), brother_of(Z,Y).
```







Program x +

```
1 brother_of(paul,peter).  
2 brother_of(peter,adrian).  
3 brother_of(X,Y) :- brother_of(X,Z),brother_of(Z,Y).  
4
```



?- brother\_of(paul, B)



File Edit Examples Help



188 users online

Search



Program x +

```
1 brother_of(paul,peter).
2 brother_of(peter,adrian).
3 brother_of(X,Y) :- brother_of(X,Z),brother_of(Z,Y).
4
```

  
brother\_of(paul, B)

B = peter

Next 10 100 1,000 Stop

?- brother\_of(paul, B)

Examples History Solutions

 table results

Run!

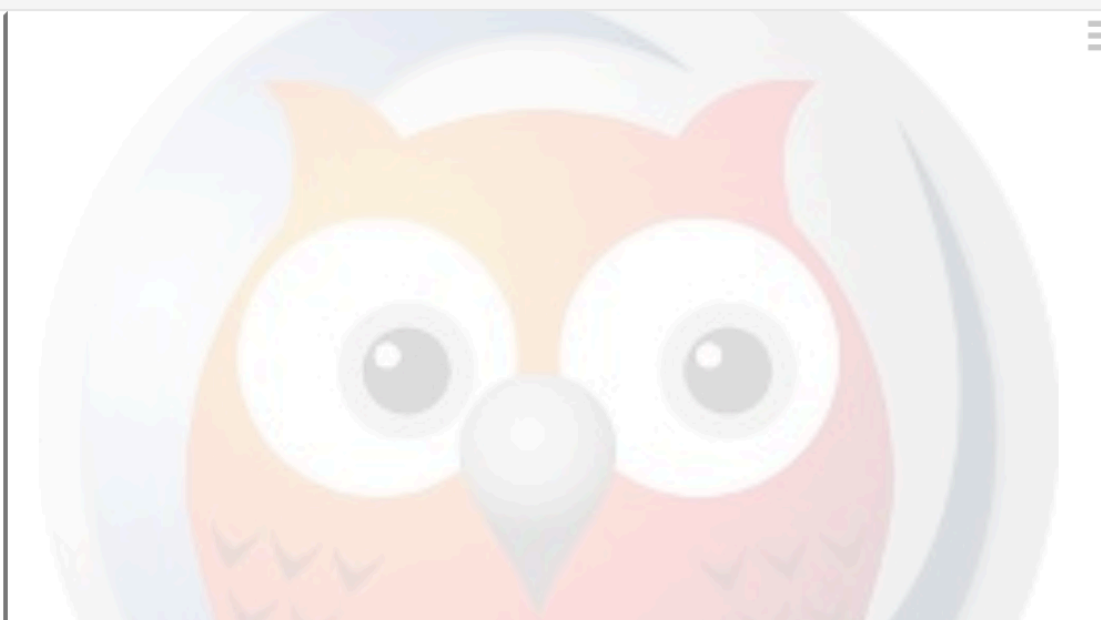


Program x +

```

1 brother_of(paul,peter).
2 brother_of(peter,adrian).
3 brother_of(X,Y) :- brother_of(X,Z),brother_of(Z,Y).
4

```



brother\_of(paul, B)

B = peter

Next 10 100 1,000 Stop

?- brother\_of(paul, B)

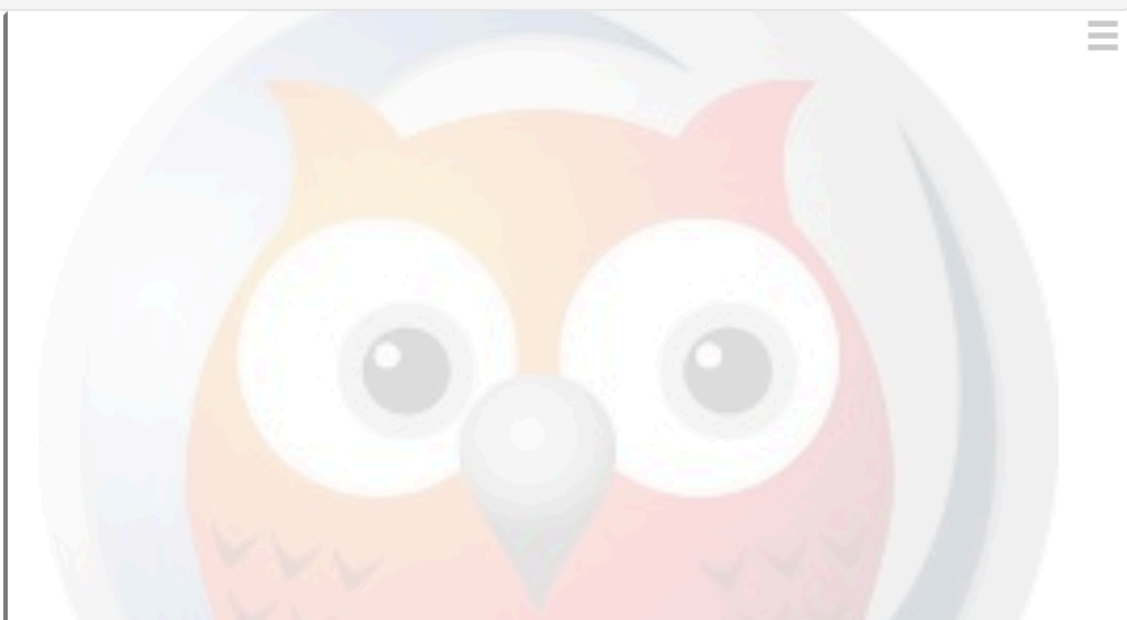


Program x +

```

1 brother_of(paul,peter).
2 brother_of(peter,adrian).
3 brother_of(X,Y) :- brother_of(X,Z),brother_of(Z,Y).
4

```



brother\_of(paul, B)

B = peter

Next 10 100 1,000 Stop



?- brother\_of(paul, B)

Program ✕ +

```
1 brother_of(paul,peter).
2 brother_of(peter,adrian).
3 brother_of(X,Y) :- brother_of(X,Z),brother_of(Z,Y).
4
```

  
 `brother_of(paul, B)` ⌵ ▬ ✕**B** = peter**B** = adrian

Next 10 100 1,000 Stop

?- brother\_of(paul, B)



Program +

```
1 brother_of(paul,peter).
2 brother_of(peter,adrian).
3 brother_of(X,Y) :- brother_of(X,Z),brother_of(Z,Y).
4
```

  
**brother\_of(paul, B)****B = peter****B = adrian**

Next 10 100 1,000 Stop

?- brother\_of(paul, B)



Program x +

```
1 brother_of(paul,peter).
2 brother_of(peter,adrian).
3 brother_of(X,Y) :- brother_of(X,Z),brother_of(Z,Y).
4
```

brother\_of(paul, B)

B = peter

B = adrian

**Stack limit (0.2Gb) exceeded**

Stack sizes: local: 0.2Gb, global: 19.6Mb, trail: 1Kb

Stack depth: 2,567,670, last-call: 0%, Choice points: 14

Probable infinite recursion (cycle):

[2,567,670] brother\_of(adrian, \_1326)

[2,567,669] brother\_of(adrian, \_1352)

?- brother\_of(paul, B)

# Exercise 3.2

```
list([]).  
list([H|T]):-list(T).
```

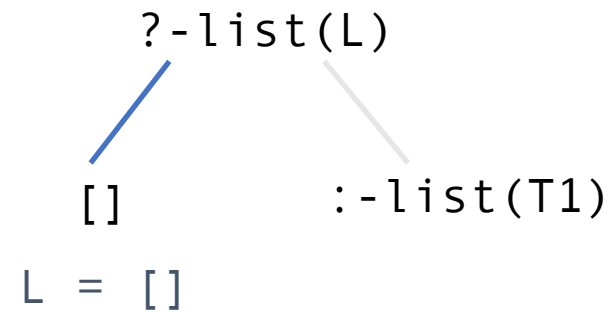
```
?-list(L)
```

```
?-list(L).
```



# Exercise 3.2

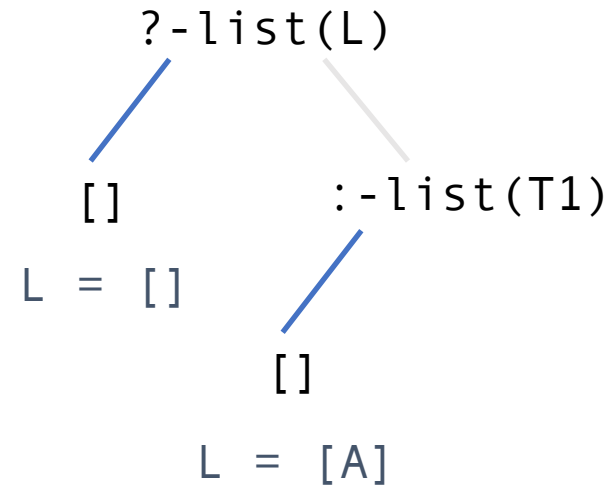
```
list([]).  
list([H|T]):-list(T).
```



```
?-list(L).  
L = [];
```

# Exercise 3.2

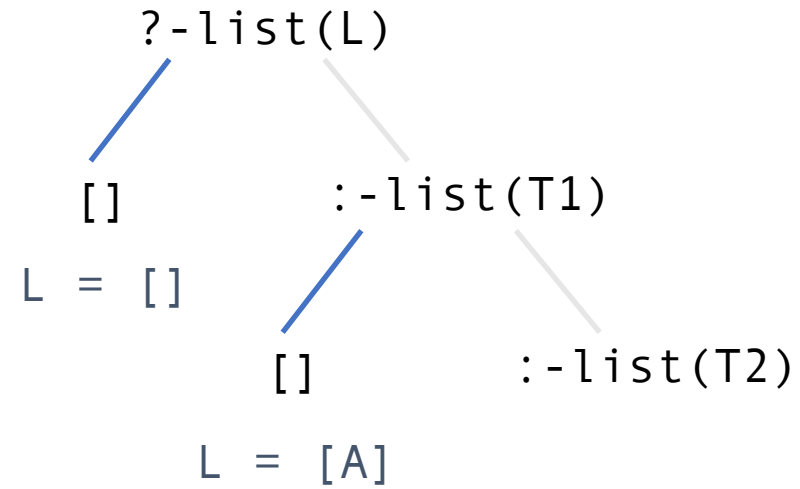
```
list([]).  
list([H|T]):-list(T).
```



```
?-list(L).  
L = [];  
L = [A];
```

# Exercise 3.2

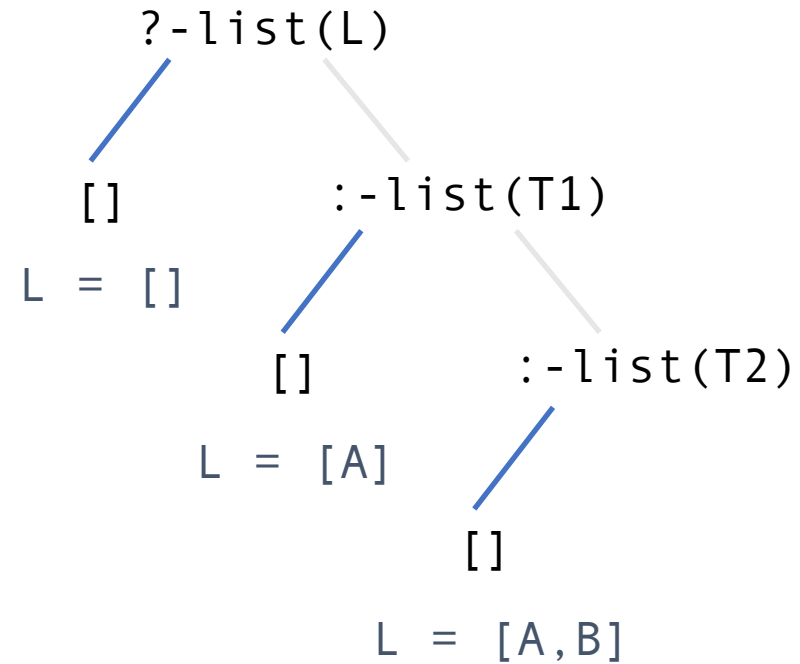
```
list([]).  
list([H|T]):-list(T).
```



```
?-list(L).  
L = [];  
L = [A];
```

# Exercise 3.2

```
list([]).  
list([H|T]):-list(T).
```

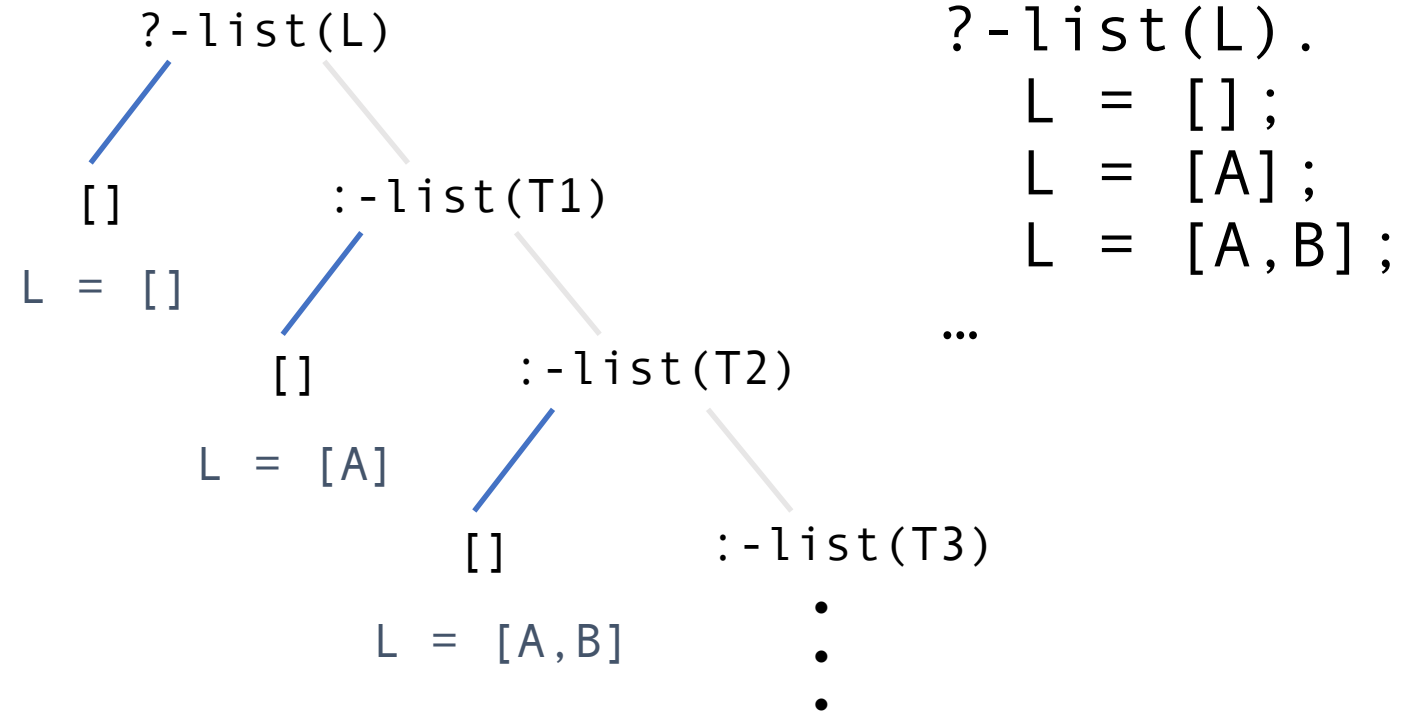


```
?-list(L).  
L = [];  
L = [A];  
L = [A,B];
```



# Exercise 3.2

```
list([]).  
list([H|T]):-list(T).
```



**SWISH**

File ▾

Edit ▾

Examples ▾

Help ▾



190 users online

Search



25

Program ✕ +

```
1 list([]).
2 list([_|T]):-list(T).
3 |
```

list(L)

L = []

L = [\_1296]

L = [\_1296, \_1302]

L = [\_1296, \_1302, \_1308]

L = [\_1296, \_1302, \_1308, \_1314]

L = [\_1296, \_1302, \_1308, \_1314, \_1320]

L = [\_1296, \_1302, \_1308, \_1314, \_1320, \_1326]

L = [\_1296, \_1302, \_1308, \_1314, \_1320, \_1326, \_1332]

L = [\_1296, \_1302, \_1308, \_1314, \_1320, \_1326, \_1332, \_1338]

Next

10

100

1,000

Stop

?- list(L)

Examples ▲

History ▲

Solutions ▲

 table results

Run!

## Another example

```
?-plist(L)
```

```
plist([]).  
plist([H|T]):-  
    p(H),plist(T).
```

```
p(1).      p(2).
```

```
?-plist(L).
```

## Another example

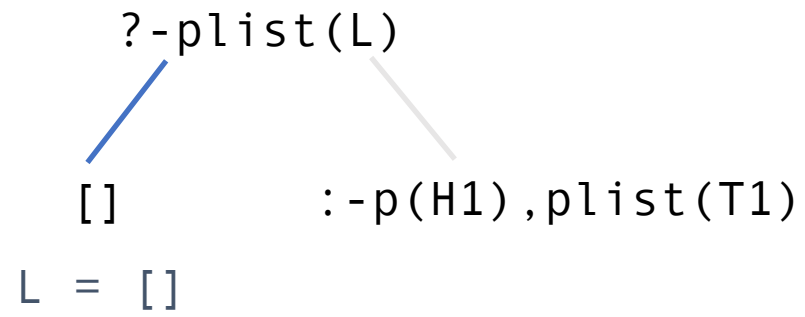
?-plist(L)  
|  
[]  
L = []

```
plist([]).  
plist([H|T]):-  
    p(H),plist(T).
```

```
p(1).      p(2).
```

```
?-plist(L).
```

## Another example

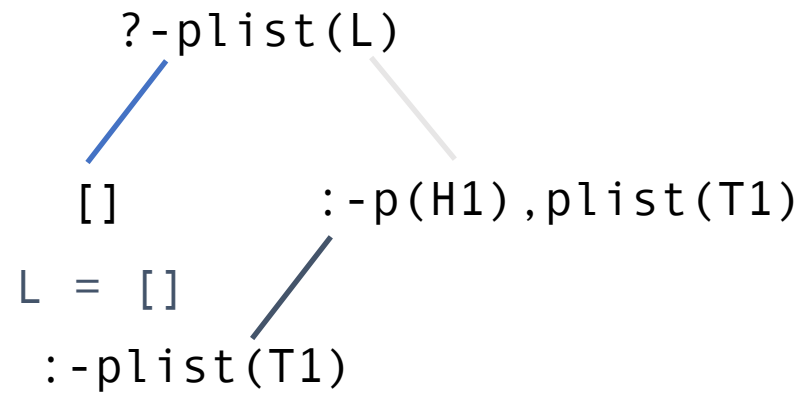


```
plist([]).  
plist([H|T]):-  
    p(H),plist(T).
```

```
p(1).      p(2).
```

```
?-plist(L).  
L=[];
```

## Another example

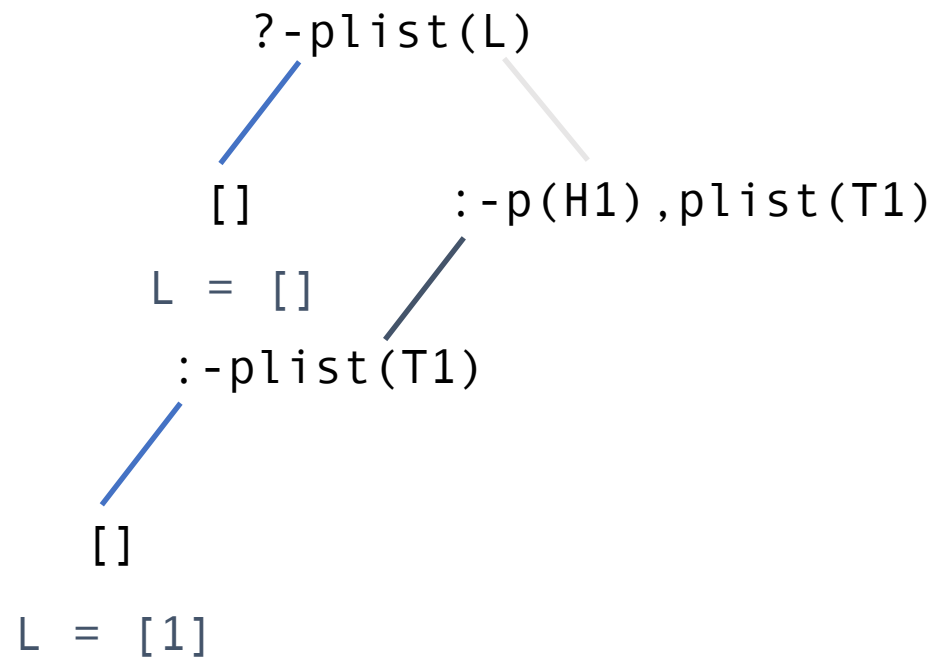


```
plist([]).  
plist([H|T]):-  
    p(H),plist(T).
```

```
p(1).      p(2).
```

```
?-plist(L).  
L=[];
```

## Another example

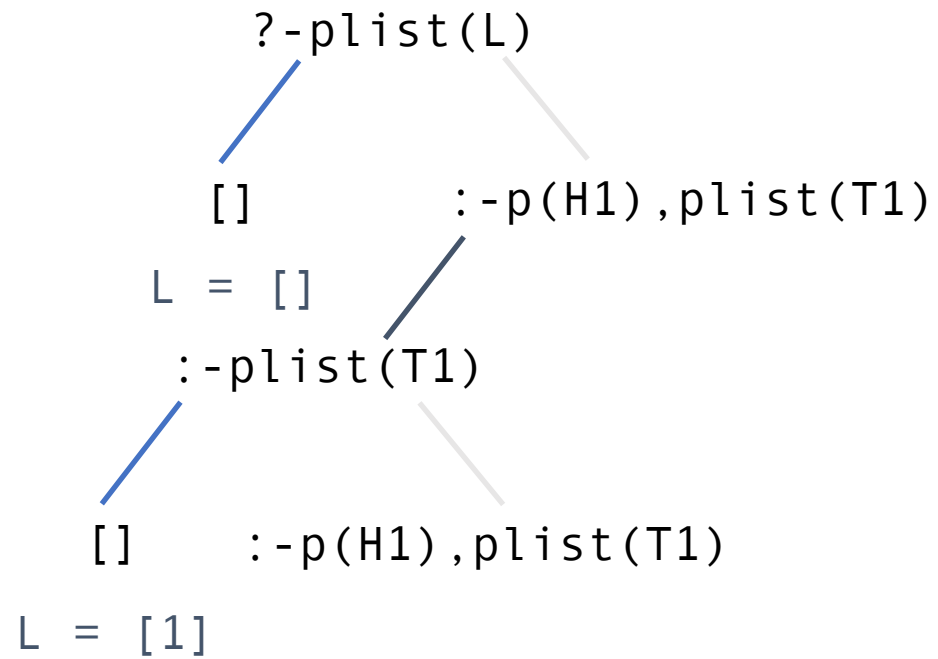


```
plist([]).
plist([H|T]):-
    p(H),plist(T).
```

```
p(1).      p(2).
```

```
?-plist(L).
L=[];
L=[1];
```

## Another example



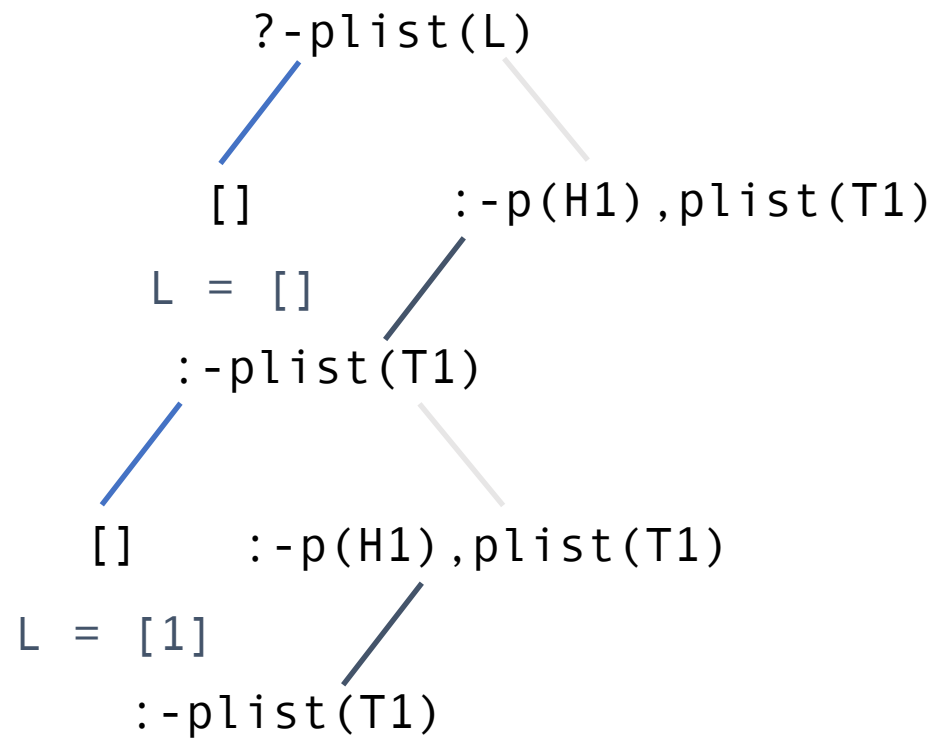
```
plist([]).
plist([H|T]):-
    p(H),plist(T).
```

```
p(1).      p(2).
```

```
?-plist(L).
L=[];
L=[1];
```



## Another example

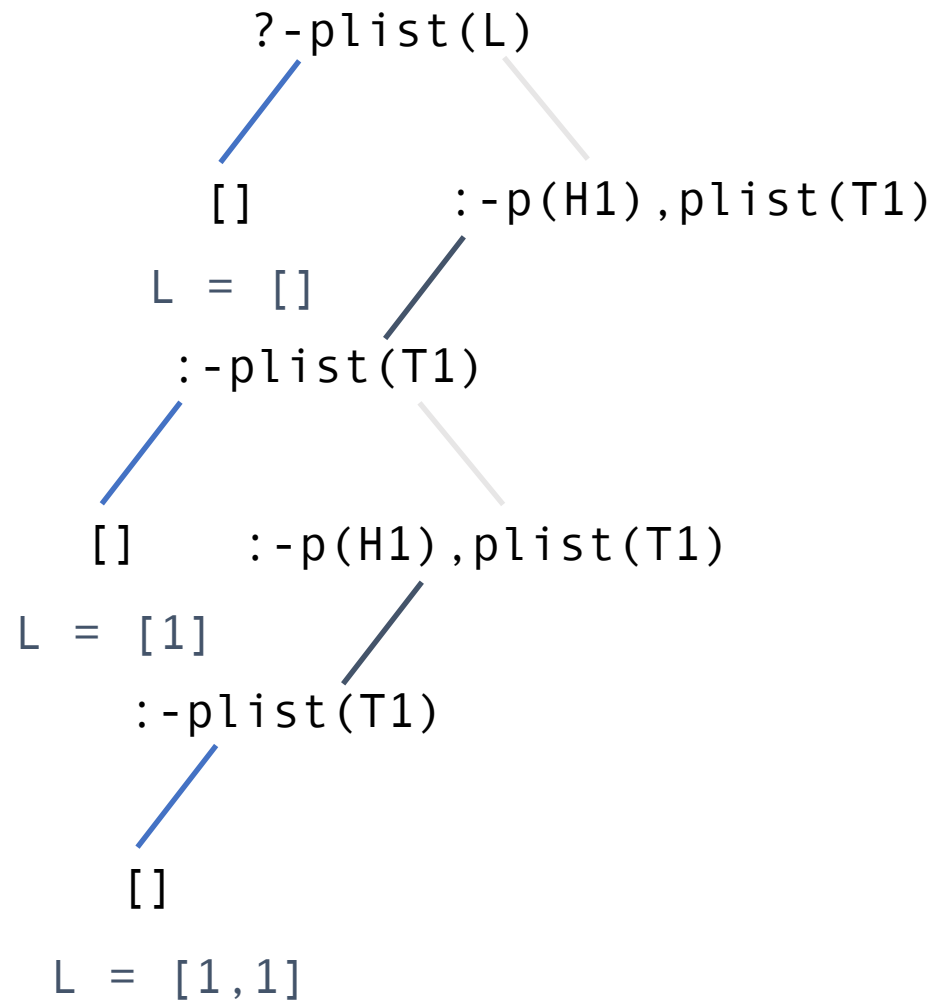


```
plist([]).
plist([H|T]):-
    p(H),plist(T).
```

```
p(1).      p(2).
```

```
?-plist(L).
L=[];
L=[1];
```

## Another example



```
plist([]).
plist([H|T]):-
    p(H),plist(T).
```

```
p(1).      p(2).
```

```
?-plist(L).
L=[];
L=[1];
L=[1,1];
```

# Another example (and so on)

```

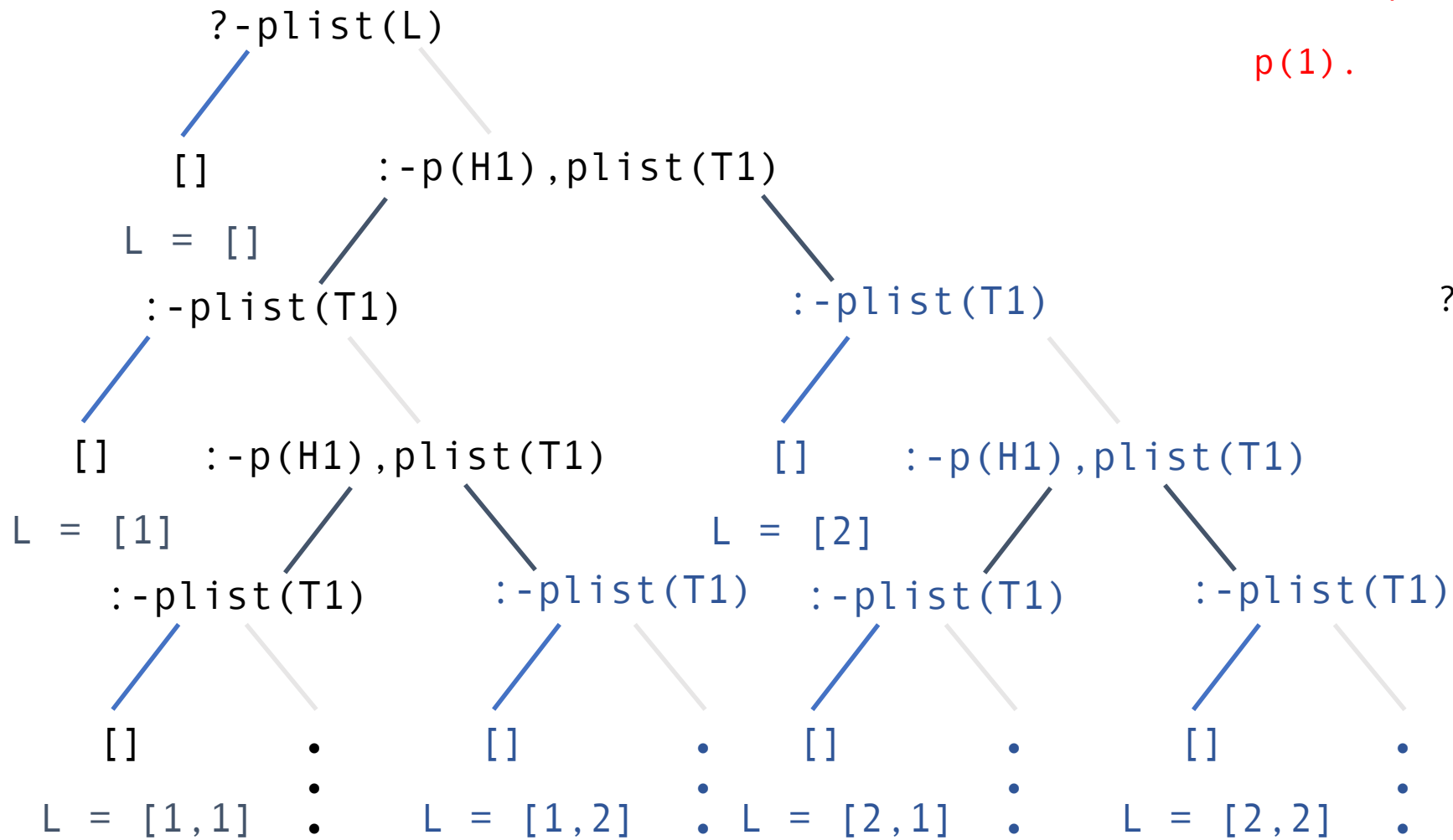
plist([]).
plist([H|T]):-
    p(H),plist(T).

```

```

p(1).      p(2).

```



```

?-plist(L).
L=[];
L=[1];
L=[1,1];
...

```

Program +

```
1 plist([]).  
2 plist([H|T]) :- p(H),plist(T).  
3  
4 p(1).  
5 p(2).  
6
```

**plist(L)**

```
L = []  
L = [1]  
L = [1, 1]  
L = [1, 1, 1]  
L = [1, 1, 1, 1]  
L = [1, 1, 1, 1, 1]  
L = [1, 1, 1, 1, 1, 1]  
L = [1, 1, 1, 1, 1, 1, 1]  
L = [1, 1, 1, 1, 1, 1, 1, 1]  
L = [1, 1, 1, 1, 1, 1, 1, 1, 1]  
L = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

Next 10 100 1,000 Stop

?- **plist(L)**

Examples History Solutions

table results

Run!



Program

```

1 plist([]).
2 plist([H|T]) :- p(H),plist(T).
3
4 p(1).
5 p(2).
6

```

`plist([X,Y,2|T]), L = [X,Y,2|T]`

L = [1, 1, 2],

T = [],

X = Y, Y = 1

L = [1, 1, 2, 1],

T = [1],

X = Y, Y = 1

L = [1, 1, 2, 1, 1],

T = [1, 1],

X = Y, Y = 1

L = [1, 1, 2, 1, 1, 1],

T = [1, 1, 1],

X = Y, Y = 1

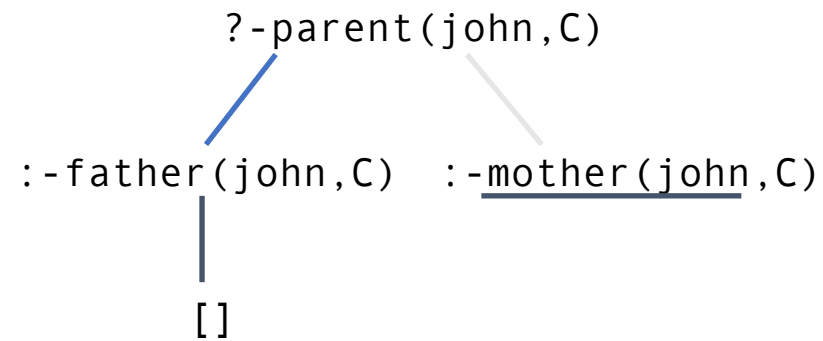
Next 10 100 1,000 Stop

?- plist([X,Y,2|T]), L = [X,Y,2|T]

Cut

# Pruning by means of cut

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





File ▾

Edit ▾

Examples ▾

Help ▾

169 users online

Search



Program × +

```
1 parent(X,Y):-father(X,Y).
2 parent(X,Y):-mother(X,Y).
3 father(john,paul).
4 mother(mary,paul).
5
```

  
 `parent(john,C)``C = paul``Next` 10 100 1,000 Stop`?- parent(john,C)`

Examples ▲

History ▲

Solutions ▲

 table results

Run!





File ▾

Edit ▾

Examples ▾

Help ▾

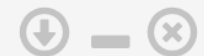
168 users online

Search



Program × +

```
1 parent(X,Y):-father(X,Y).
2 parent(X,Y):-mother(X,Y).
3 father(john,paul).
4 mother(mary,paul).
5
```

  
 `parent(john,C)`**C** = paul**false**?- `parent(john,C)`

Examples ▲

History ▲

Solutions ▲

 table results

Run!

# Pruning by means of cut

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

CUT

```
?-parent(john,C)
```

# Pruning by means of cut

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

CUT

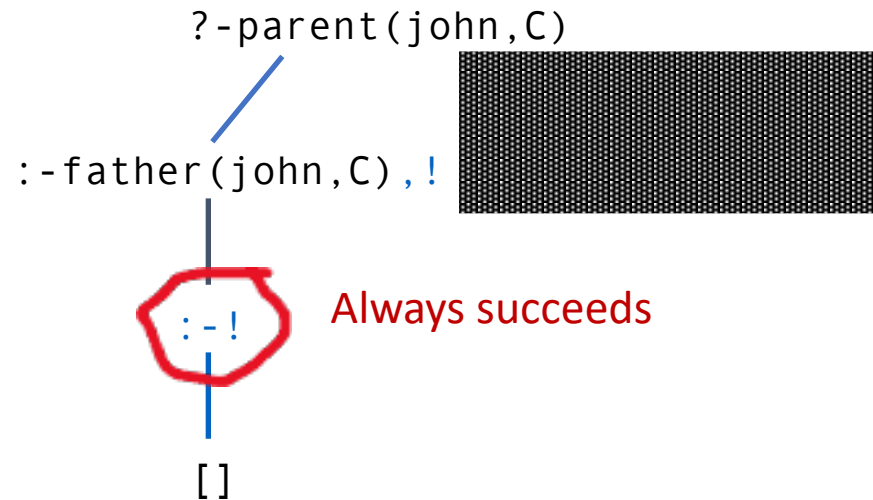
```
?-parent(john,C)  
  /  
:-father(john,C),!
```



# Pruning by means of cut

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

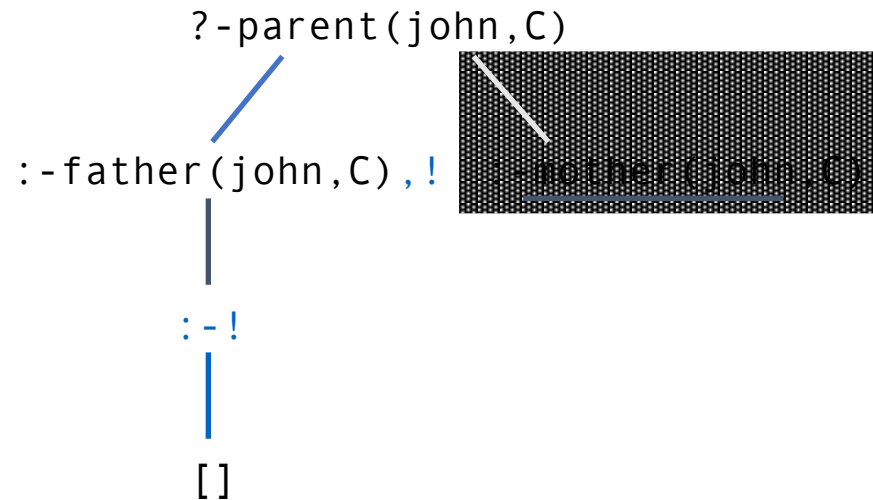
CUT



# Pruning by means of cut

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

CUT





File ▾

Edit ▾

Examples ▾

Help ▾



173 users online

Search



25

Program × +

```
1 parent(X,Y):-father(X,Y),!.
2 parent(X,Y):-mother(X,Y).
3 father(john,paul).
4 mother(mary,paul).
5
```

`parent(john,C)``C = paul``?- parent(john,C)`

Examples ▲

History ▲

Solutions ▲

 table results

Run!

# Compare

## Without cut

parent(john,C)

C = paul

Next 10 100 1,000 Stop

?- parent(john,C)

Examples History Solutions table results Run!

## With cut

parent(john,C)

C = paul

?- parent(john,C)

Examples History Solutions table results Run!



# The effect of cut

?-p(X,Y)

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

$p(X, Y) :- q(X, Y) .$

$p(X, Y) :- r(X, Y) .$

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

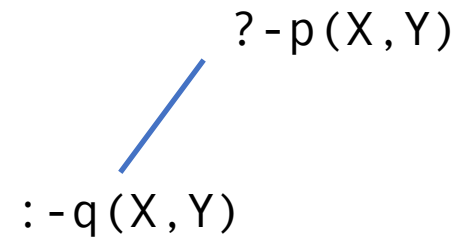
$r(c, d) .$

$s(a) .$

$s(b) .$

$t(a) .$

$t(b) .$



# The effect of cut

$p(X, Y) :- q(X, Y) .$

$p(X, Y) :- r(X, Y) .$

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

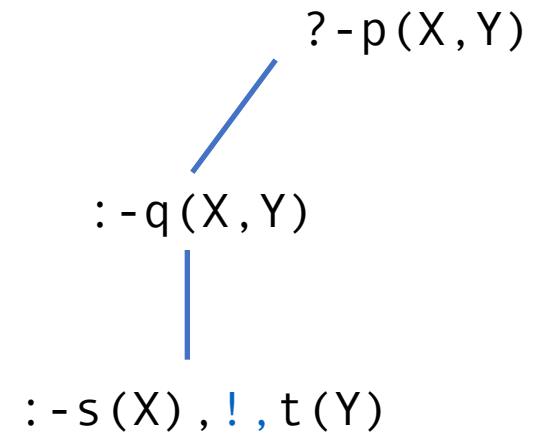
$r(c, d) .$

$s(a) .$

$s(b) .$

$t(a) .$

$t(b) .$



# The effect of cut

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

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

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

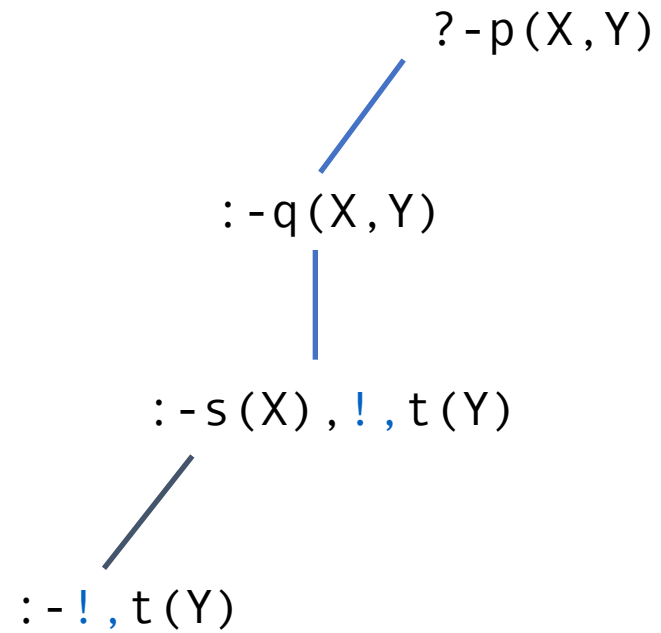
`r(c,d).`

`s(a).`

`s(b).`

`t(a).`

`t(b).`





# The effect of cut

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

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

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

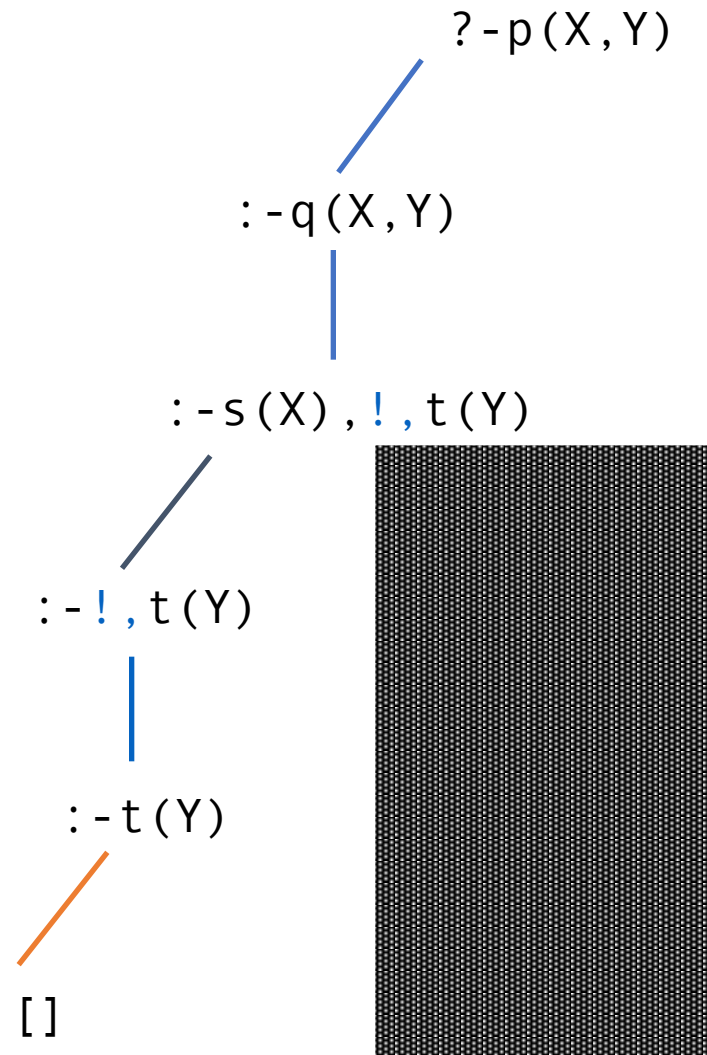
`r(c,d).`

`s(a).`

`s(b).`

`t(a).`

`t(b).`



# The effect of cut

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

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

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

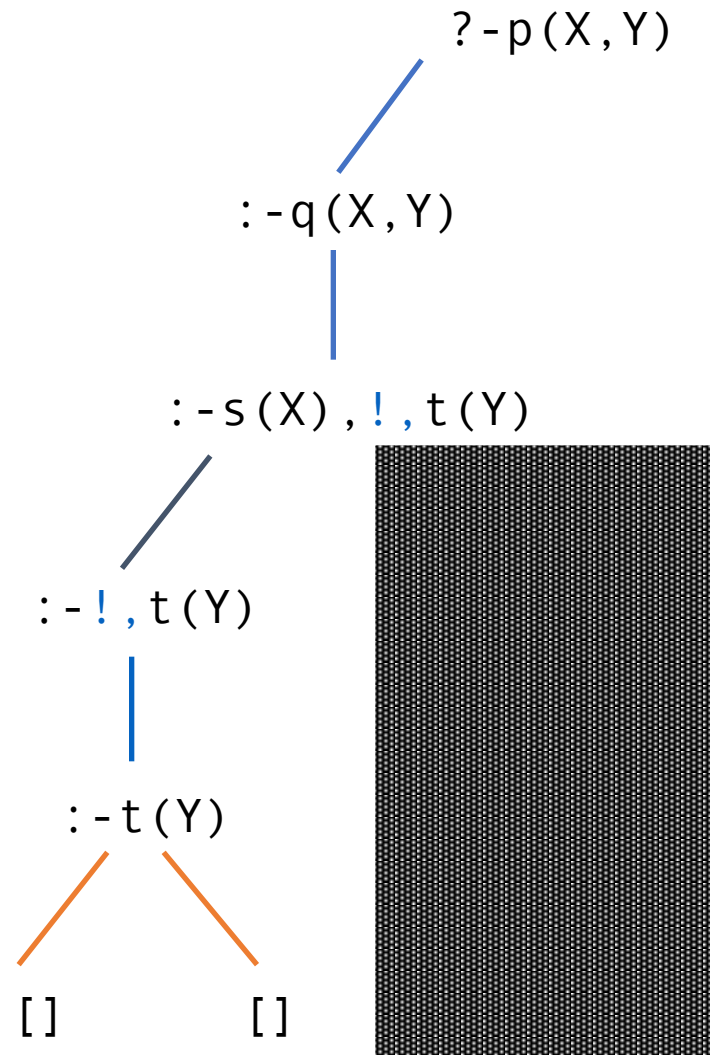
`r(c,d).`

`s(a).`

`s(b).`

`t(a).`

`t(b).`



# The effect of cut

$p(X, Y) :- q(X, Y) .$

$p(X, Y) :- r(X, Y) .$

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

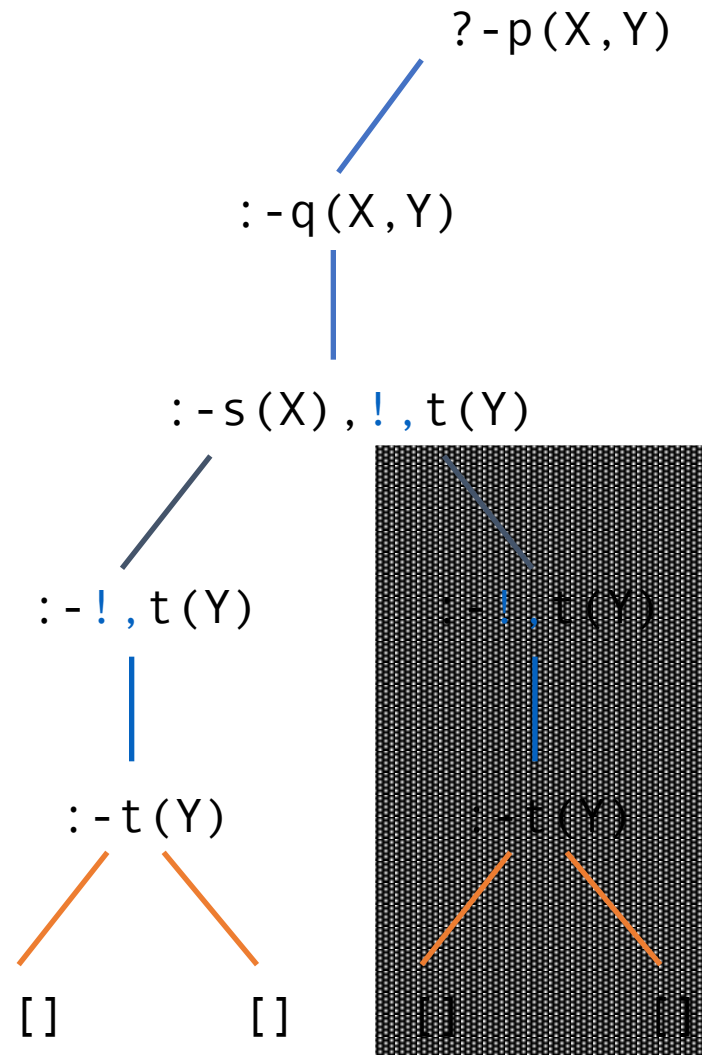
$r(c, d) .$

$s(a) .$

$s(b) .$

$t(a) .$

$t(b) .$





# The effect of cut

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

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

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

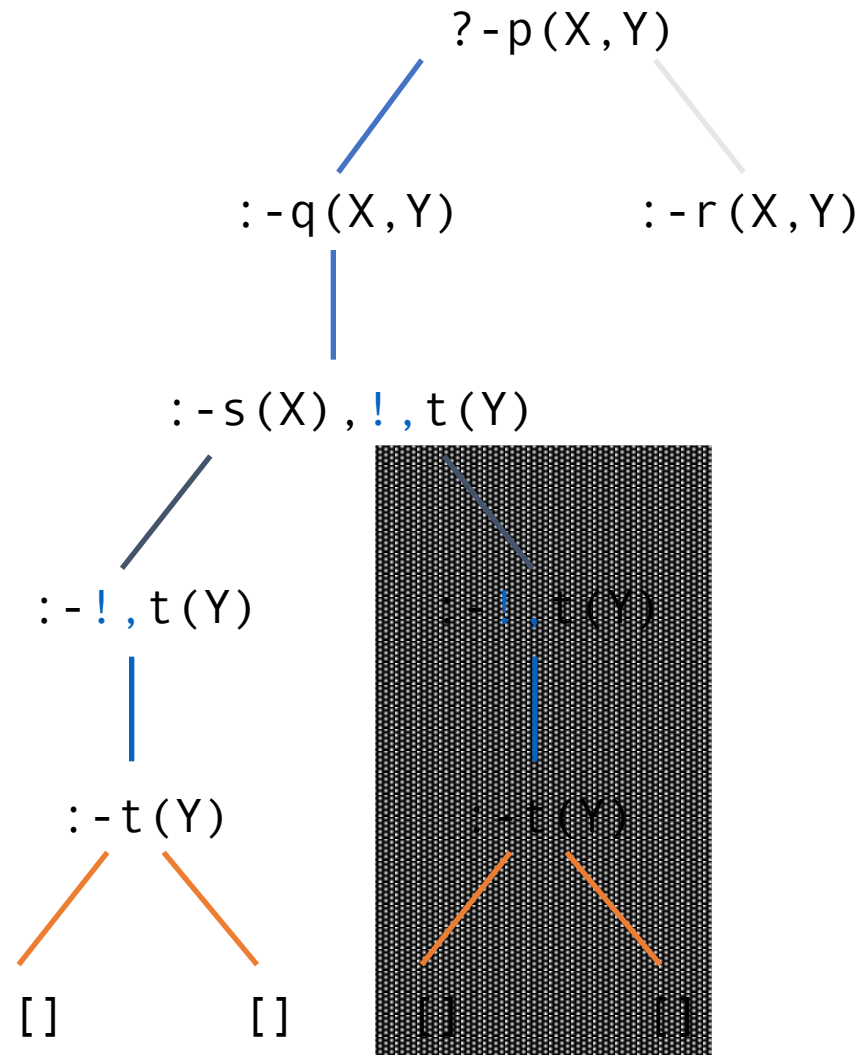
`r(c,d).`

`s(a).`

`s(b).`

`t(a).`

`t(b).`



# The effect of cut

$p(X, Y) :- q(X, Y) .$

$p(X, Y) :- r(X, Y) .$

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

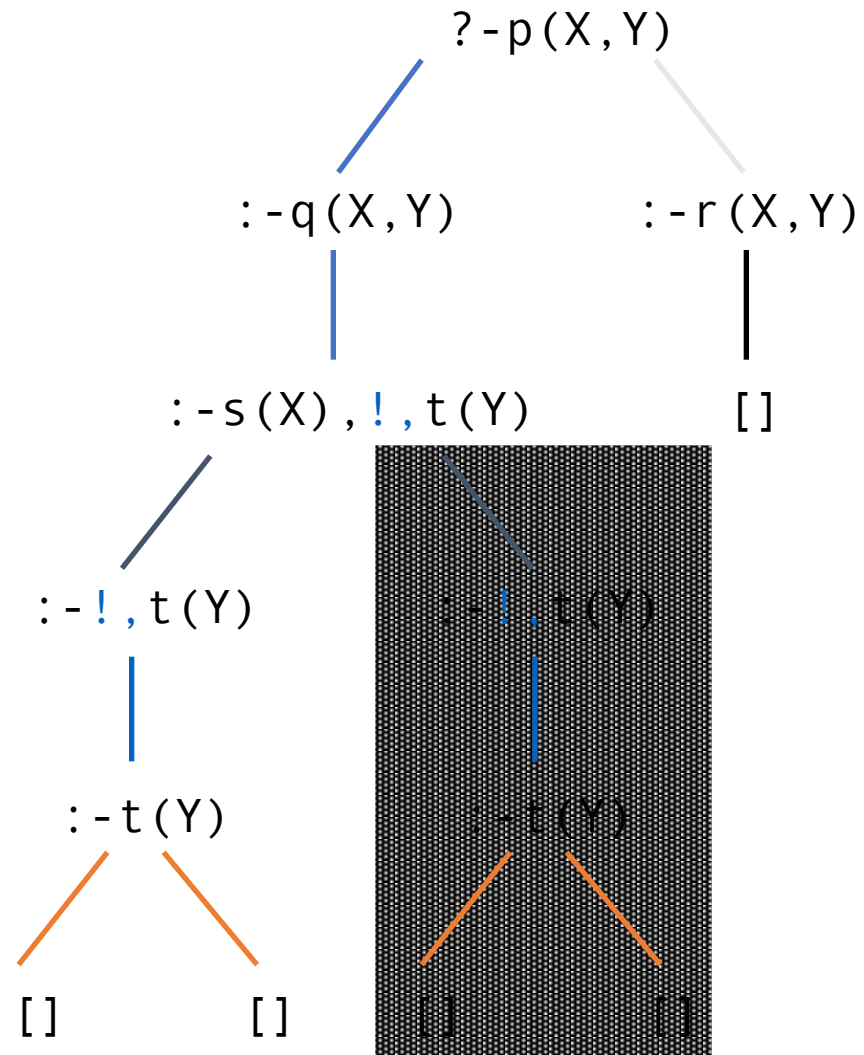
$r(c, d) .$

$s(a) .$

$s(b) .$

$t(a) .$

$t(b) .$





Program x +

```
1 p(X,Y):-q(X,Y).
2 p(X,Y):-r(X,Y).
3
4 q(X,Y):-s(X),!,t(Y).
5
6 r(c,d).
7
8 s(a).
9 s(b).
10
11 t(a).
12 t(b).
13
```



?- p(X,Y)



Program

```
1 p(X,Y):-q(X,Y).
2 p(X,Y):-r(X,Y).
3
4 q(X,Y):-s(X),!,t(Y).
5
6 r(c,d).
7
8 s(a).
9 s(b).
10
11 t(a).
12 t(b).
13
```



? p(X,Y)



Program

```
1 p(X,Y):-q(X,Y).
2 p(X,Y):-r(X,Y).
3
4 q(X,Y):-s(X),!,t(Y).
5
6 r(c,d).
7
8 s(a).
9 s(b).
10
11 t(a).
12 t(b).
13
```

p(X,Y)

X = Y, Y = a

Next 10 100 1,000 Stop

?- p(X,Y)

Examples History Solutions

 table results [Run!](#)

Program +

```
1 p(X,Y):-q(X,Y).
2 p(X,Y):-r(X,Y).
3
4 q(X,Y):-s(X),t(Y).
5
6 r(c,d).
7
8 s(a).
9 s(b).
10
11 t(a).
12 t(b).
13
```

p(X,Y)

 $X = Y, Y = a$  $X = a,$  $Y = b$ 

Next 10 100 1,000 Stop

?- p(X,Y)



File Edit Examples Help



194 users online

Search



Program

```
1 p(X,Y):-q(X,Y).
2 p(X,Y):-r(X,Y).
3
4 q(X,Y):-s(X),!,t(Y).
5
6 r(c,d).
7
8 s(a).
9 s(b).
10
11 t(a).
12 t(b).
13
```

p(X,Y)

**X = Y, Y = a****X = a,****Y = b****X = c,****Y = d**

?- p(X,Y)

Examples▲

History▲

Solutions▲

 table results

Run!

**But...**



Program +

```
1 p(X,Y):-q(X,Y).
2 p(X,Y):-r(X,Y).
3
4 q(X,Y):-s(X),!,t(Y).
5
6 r(c,d).
7
8 s(a).
9 s(b).
10
11 t(a).
12 t(b).
13
```

p(b,X)

X = a

Next 10 100 1,000 Stop

?- p(b,X)



File

Edit

Examples

Help



194 users online

Search



25

Program ✕ +

```
1 p(X,Y):-q(X,Y).
2 p(X,Y):-r(X,Y).
3
4 q(X,Y):-s(X),!,t(Y).
5
6 r(c,d).
7
8 s(a).
9 s(b).
10
11 t(a).
12 t(b).
13
```

p(b,X)

**X** = a**X** = b

Next

10

100

1,000

Stop

?-

p(b,X)

Examples

History

Solutions

 table results

Run!



File ▾

Edit ▾

Examples ▾

Help ▾



195 users online

Search



25

Program × +

```
1 p(X,Y):-q(X,Y).
2 p(X,Y):-r(X,Y).
3
4 q(X,Y):-s(X),!,t(Y).
5
6 r(c,d).
7
8 s(a).
9 s(b).
10
11 t(a).
12 t(b).
13
```

p(b,X)

**X = a****X = b****false**

?-

p(b,X)

Examples ▲

History ▲

Solutions ▲

 table results

Run!

Beware of unintended effects

# Pruning away success branches

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

```
?-parent(P,paul)
```

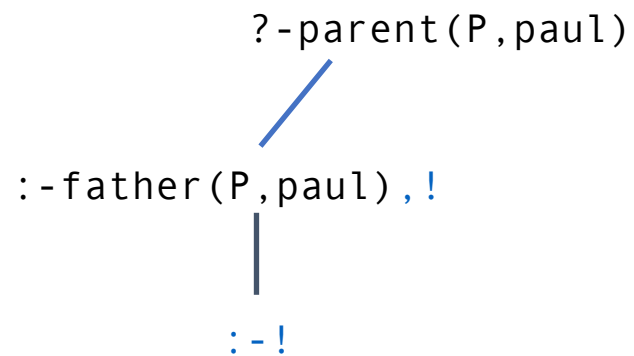
# Pruning away success branches

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

```
?-parent(P,paul)  
  /  
:-father(P,paul),!
```

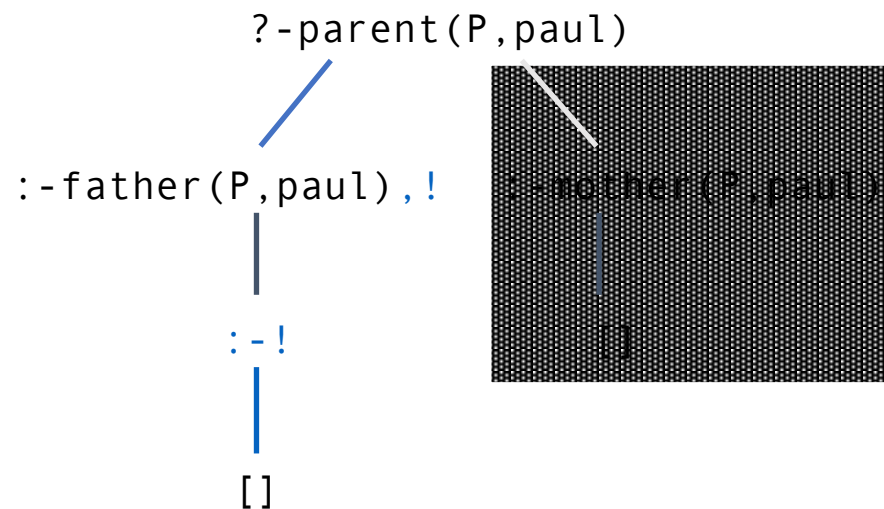
# Pruning away success branches

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



# Pruning away success branches

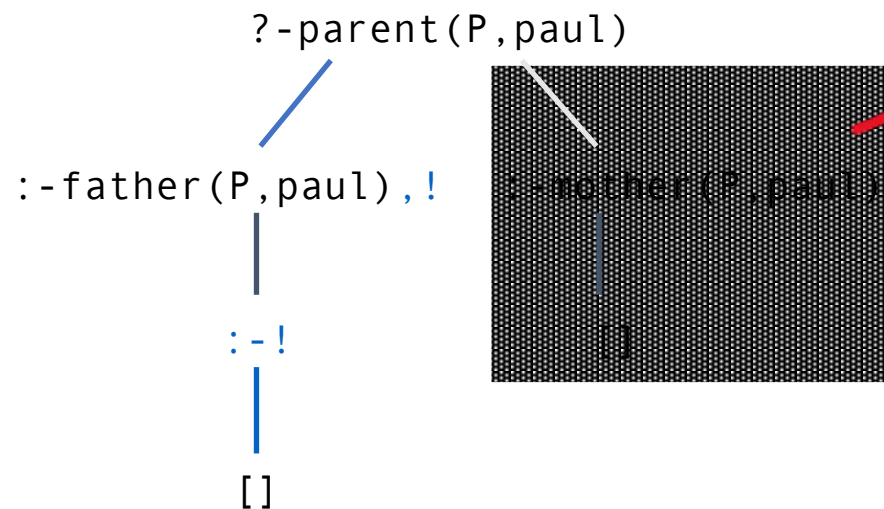
```
parent(X,Y):-father(X,Y),!.  
parent(X,Y):-mother(X,Y).  
father(john,paul).  
mother(mary,paul).
```





# Pruning away success branches

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



We may not get all the results we'd expect.



Program

```
1 parent(X,Y):-father(X,Y),!.  
2 parent(X,Y):-mother(X,Y).  
3 father(john,paul).  
4 mother(mary,paul).  
5
```



parent(X,paul)  
X = john

We get just one answer – for the father.

?- parent(X,paul)

# Pruning away success branches

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

```
?-parent(john,C)
```

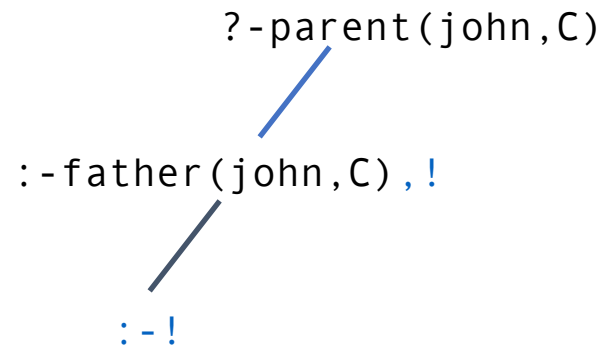
# Pruning away success branches

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

```
?-parent(john,C)  
  /  
:-father(john,C),!
```

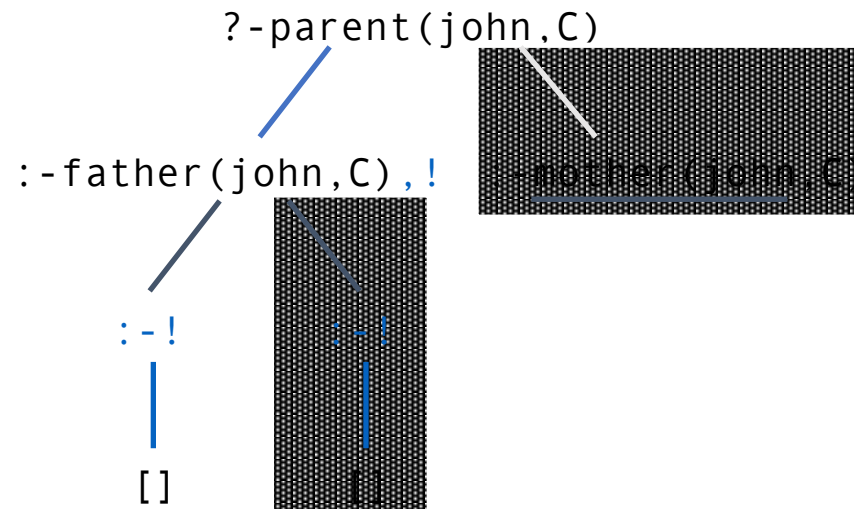
# Pruning away success branches

```
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

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





Program

```

1 parent(X,Y):-father(X,Y),!.
2 parent(X,Y):-mother(X,Y).
3 father(john,paul).
4 father(john,peter).
5 mother(mary,paul).
6 mother(mary,peter).
7

```



parent(john,X)  
 X = paul

?- parent(john,X)

And now we are not finding all children



So be careful....



And now... Negation as Failure

# Cut

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

# VS

# not

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

# Cut

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

# VS

# not

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

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



This is syntactic sugar for `call(Goal)`

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

# Cut

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

?-p

# VS

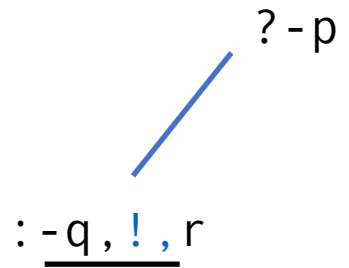
# not

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

?-p

# Cut

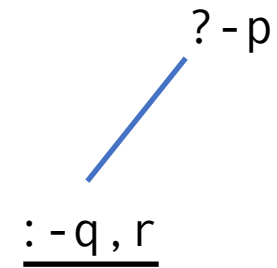
```
p: -q, !, r.  
p: -s.  
s.
```



# VS

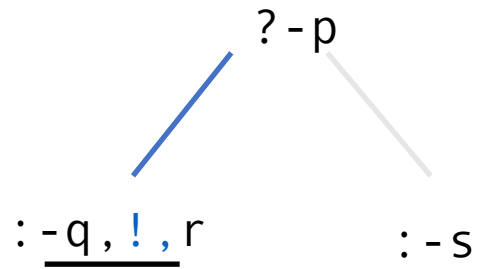
# not

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



# Cut

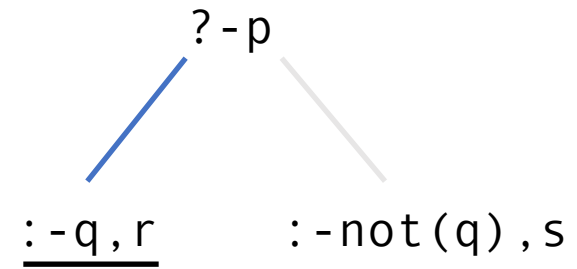
```
p: -q, !, r.  
p: -s.  
s.
```



# VS

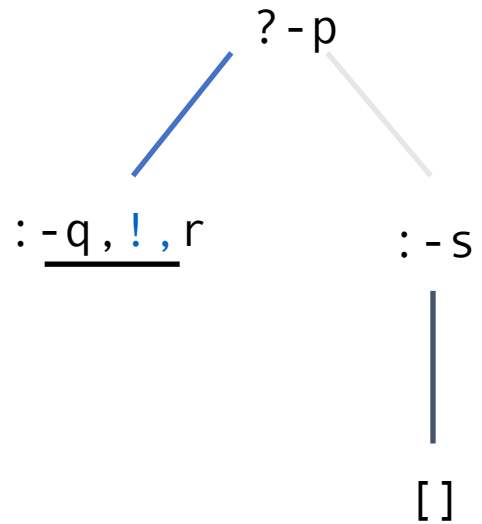
# not

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



# Cut

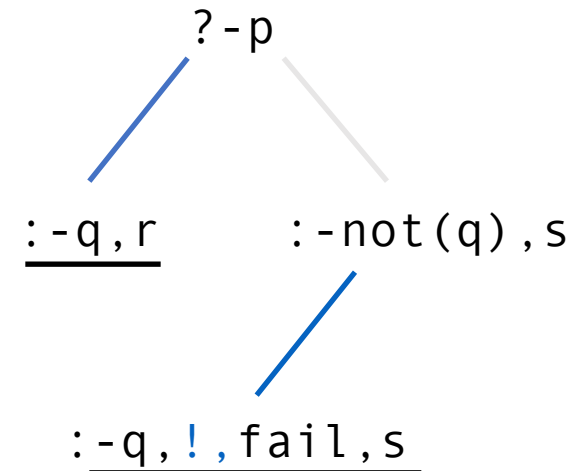
```
p: -q, !, r.  
p: -s.  
s.
```



# VS

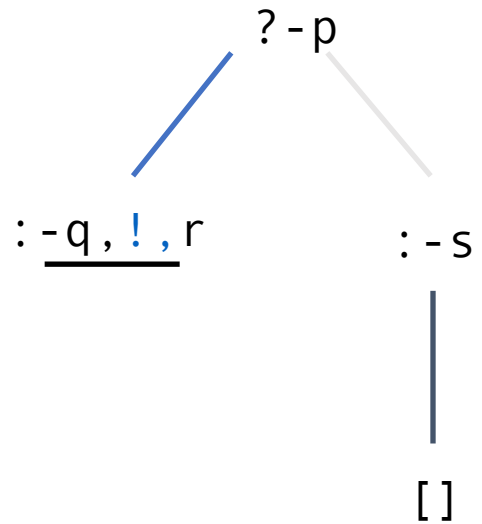
# not

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



# Cut

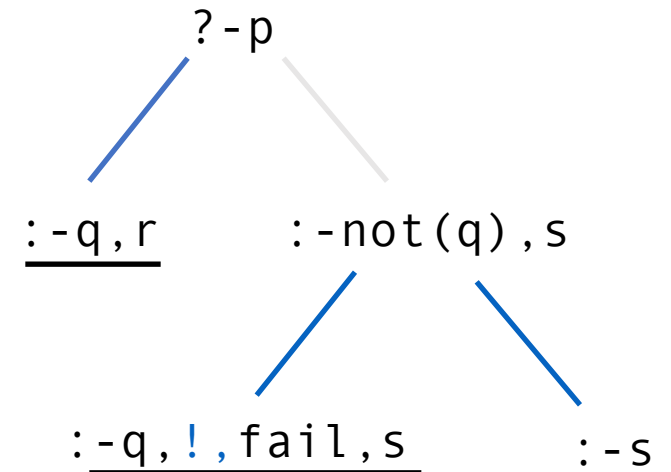
```
p:-q,!,r.  
p:-s.  
s.
```



# VS

# not

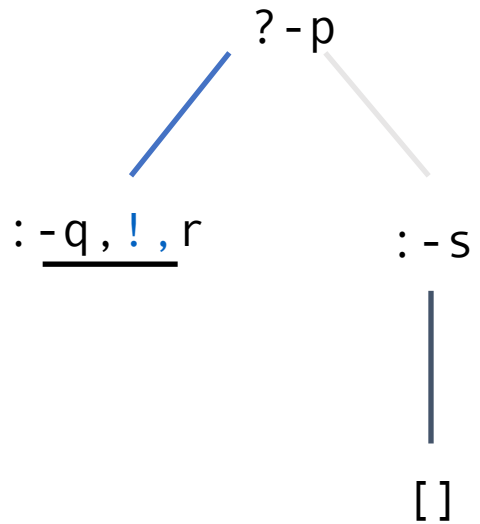
```
p:-q,r.  
p:-not(q),s.  
s.  
  
not(Goal):-Goal,!,fail.  
not(Goal).
```





# Cut

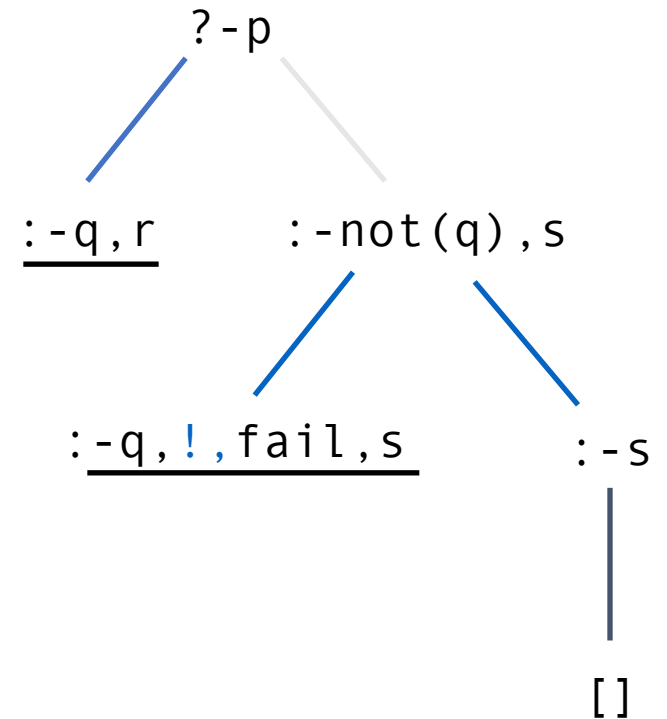
```
p:-q,!,r.  
p:-s.  
s.
```



# VS

# not

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



# An example: How it works when `:-not(q)` fails

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

```
p:-q.
```

```
q.
```

```
r.
```

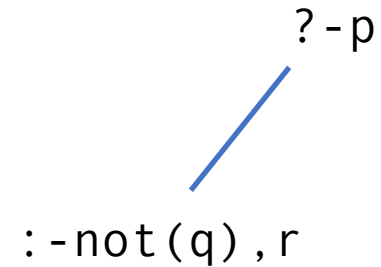
```
not(Goal):-Goal,!,fail.
```

```
not(Goal).
```

```
?-p
```

# An example: How it works when `:- not (q)` fails

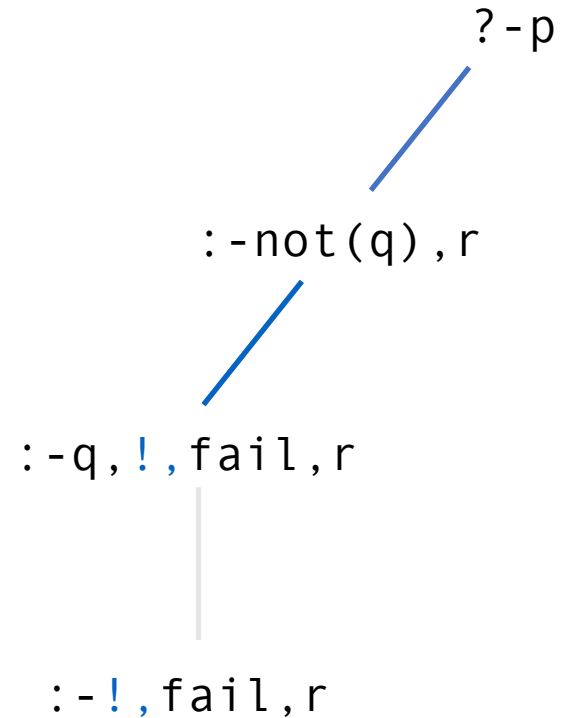
```
p:-not(q),r.  
p:-q.  
q.  
r.  
  
not(Goal):-Goal,!,fail.  
not(Goal).
```





# An example: How it works when `:-not(q)` fails

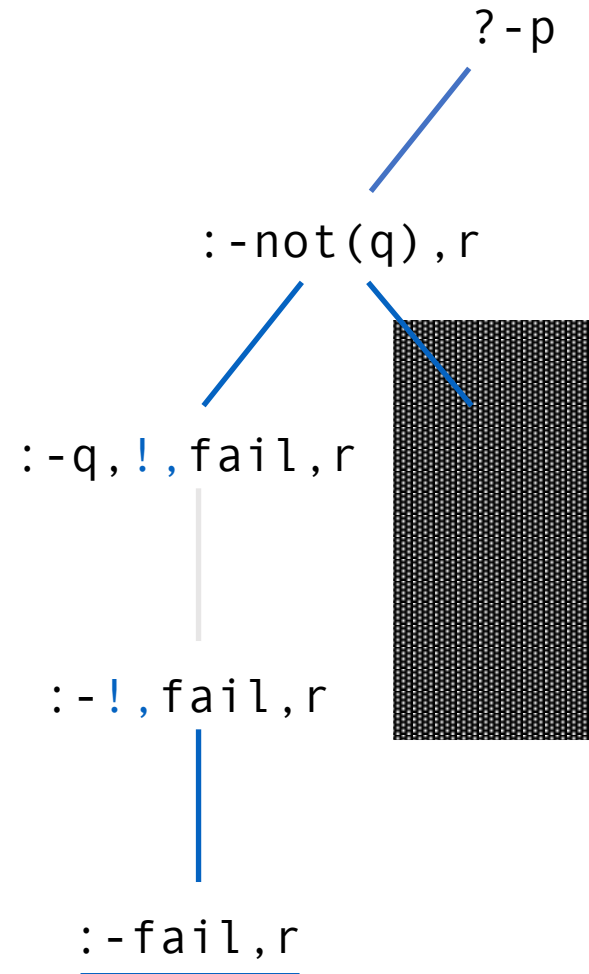
```
p:-not(q),r.  
p:-q.  
q.  
r.  
  
not(Goal):-Goal,!,fail.  
not(Goal).
```



# An example: How it works when `:-not(q)` fails

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

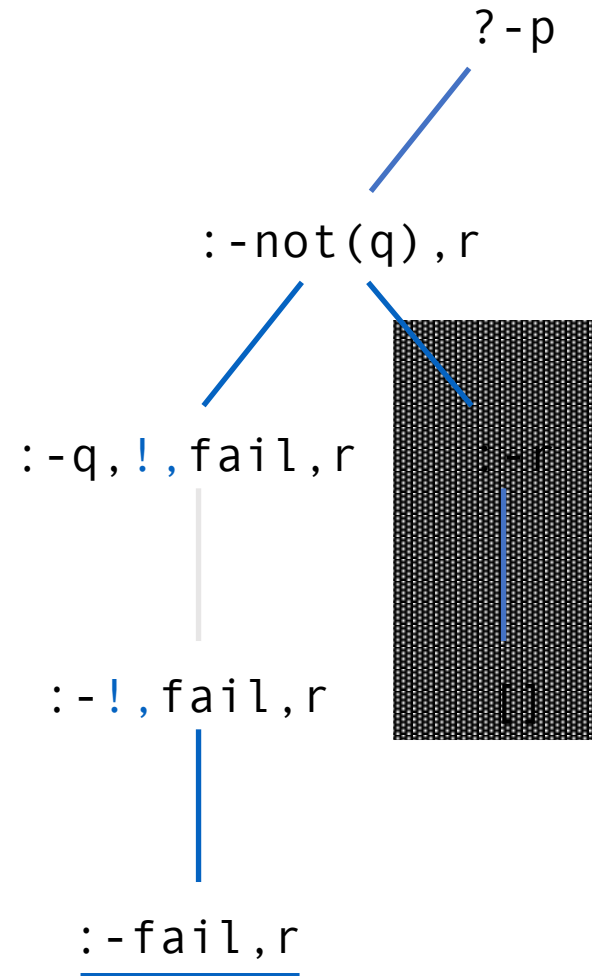
```
not(Goal):-Goal,!,fail.  
not(Goal).
```



# An example: How it works when `:-not(q)` fails

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

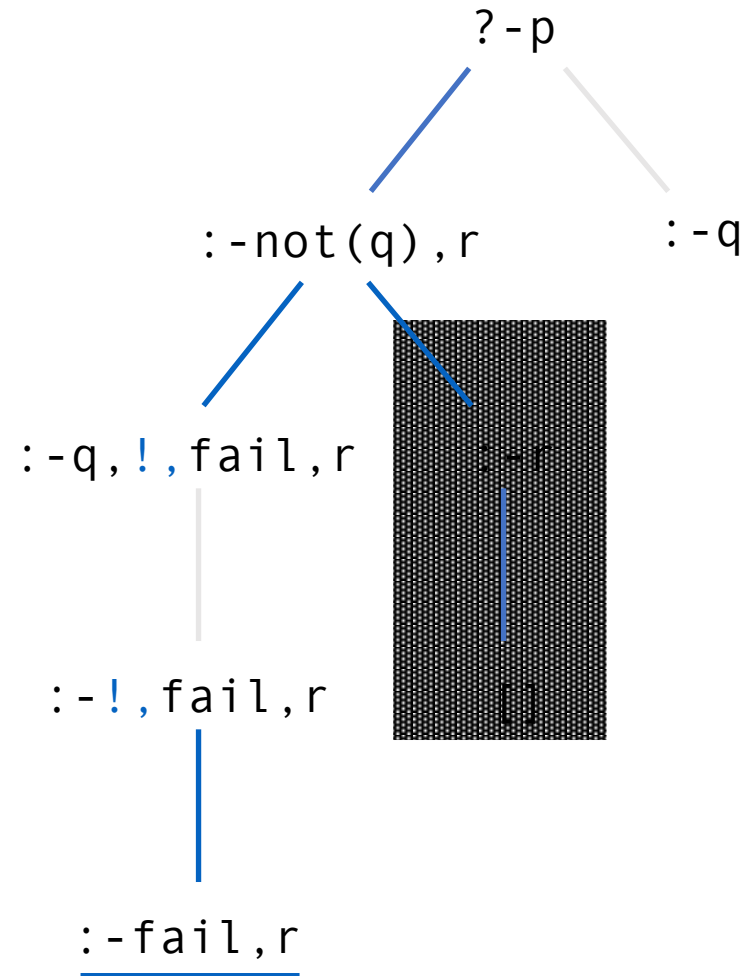
```
not(Goal):-Goal,!,fail.  
not(Goal).
```



# An example: How it works when `:-not(q)` fails

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

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

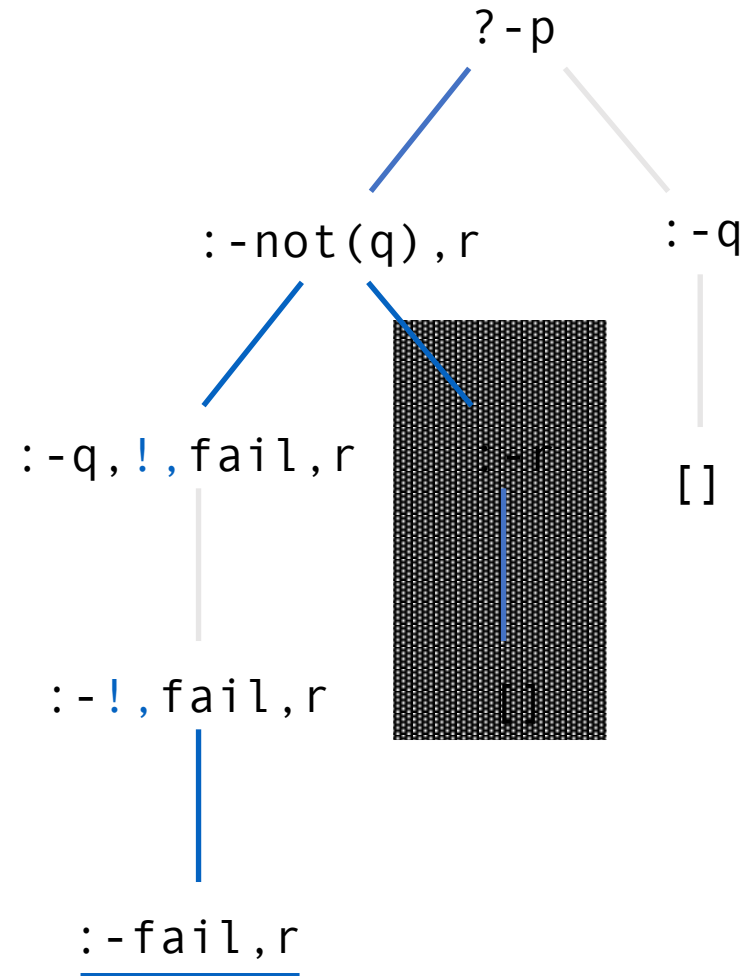




# An example: How it works when `:-not(q)` fails

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

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



# Prolog's not is unsound

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

```
?-bachelor(X)
```

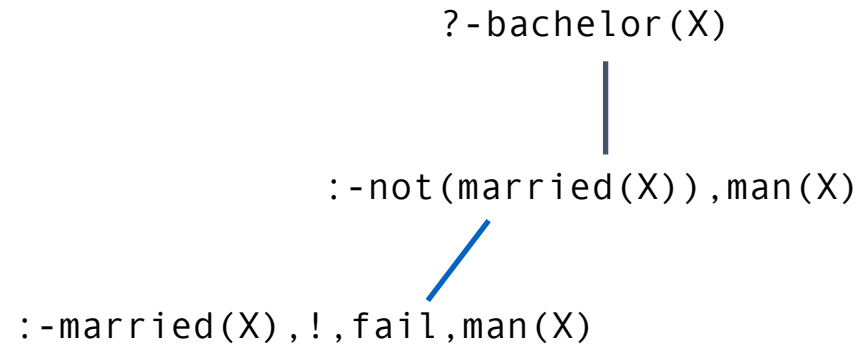
# Prolog's not is unsound

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

```
?-bachelor(X)  
|  
:-not(married(X)), man(X)
```

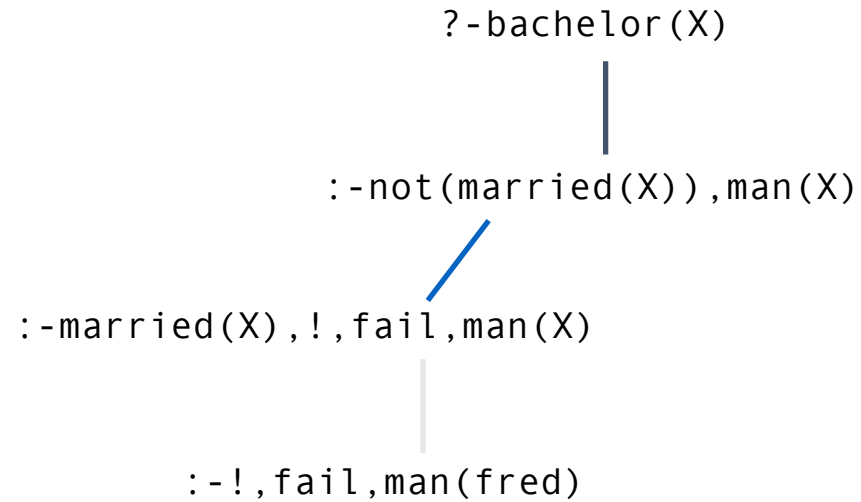
# Prolog's not is unsound

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



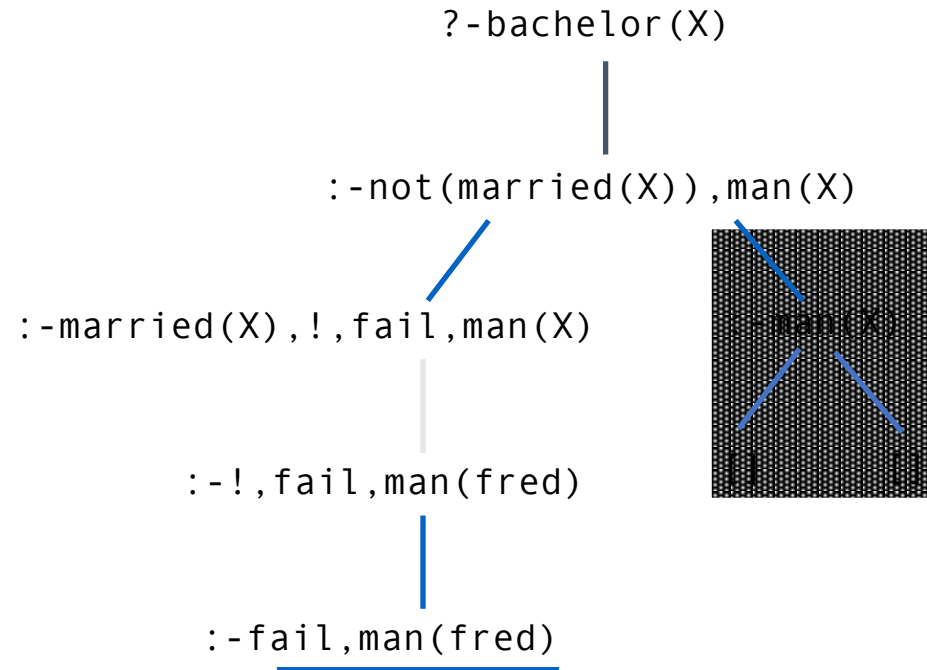
# Prolog's not is unsound

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



# Prolog's not is unsound

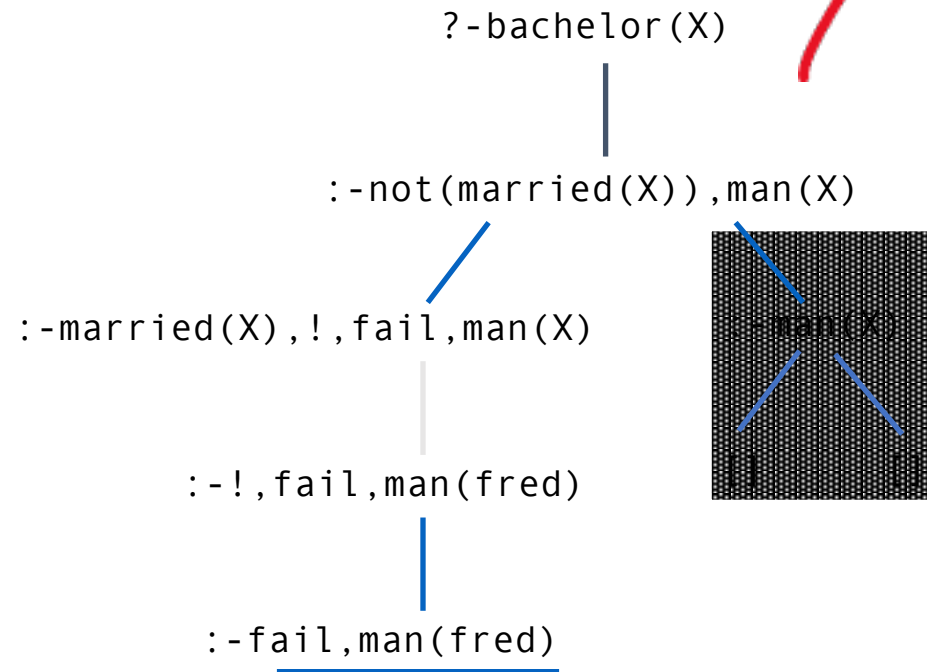
```
bachelor(X):-not(married(X)),man(X).  
man(fred).  
man(peter).  
married(fred).
```



# Prolog's not is unsound

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

But that's not "correct"! `peter` is a bachelor and Prolog did not find the corresponding answer!





Program

```
1 bachelor(X):-not(married(X)),man(X).  
2 man(fred).  
3 man(peter).  
4 married(fred).  
5
```

bachelor(X)

**false**

?- bachelor(X)



# Prolog's not is unsound – Avoiding the Problem

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

# Prolog's not is unsound – Avoiding the Problem



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

# Prolog's not is unsound – Avoiding the Problem

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

This will "ground" X.

# Prolog's not is unsound – Avoiding the Problem

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


This will "ground" X.

```
?-bachelor(X)
```

# Prolog's not is unsound – Avoiding the Problem

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

This will "ground" X.

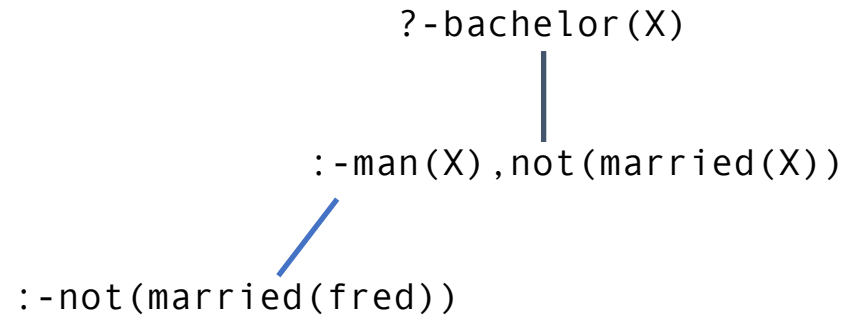


```
?-bachelor(X)  
|  
:-man(X),not(married(X))
```

# Prolog's not is unsound – Avoiding the Problem

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

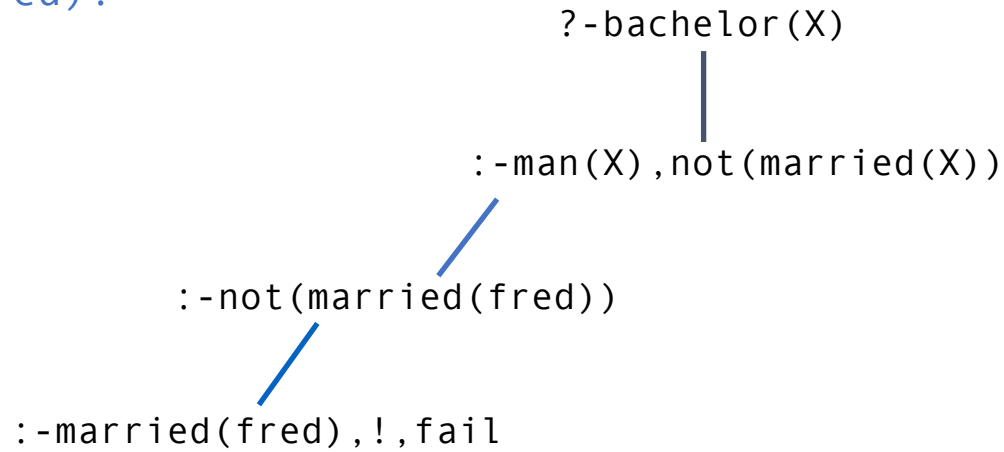
This will "ground" X.



# Prolog's not is unsound – Avoiding the Problem

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

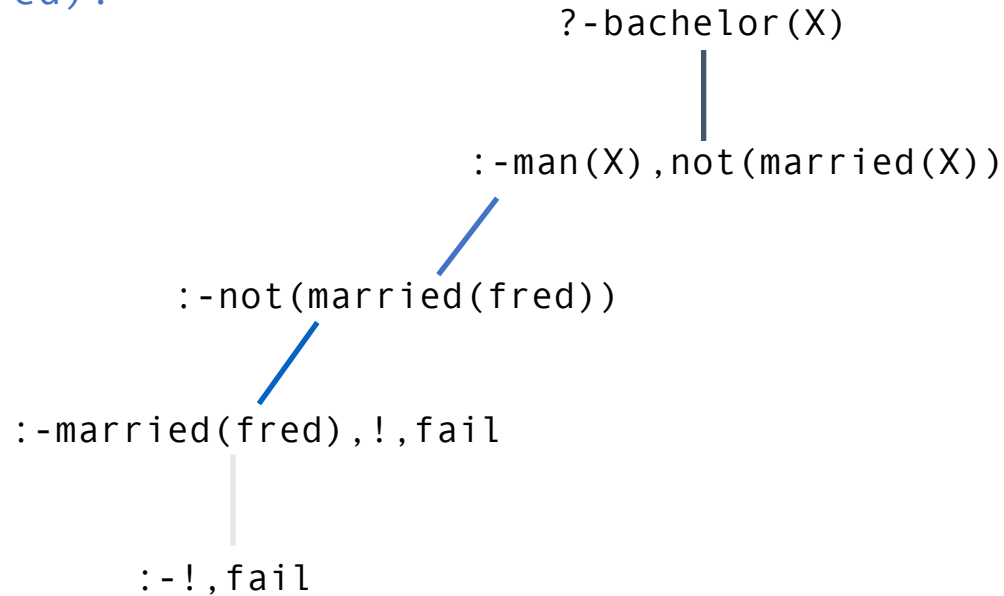
This will "ground" X.



# Prolog's not is unsound – Avoiding the Problem

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

This will "ground" X.

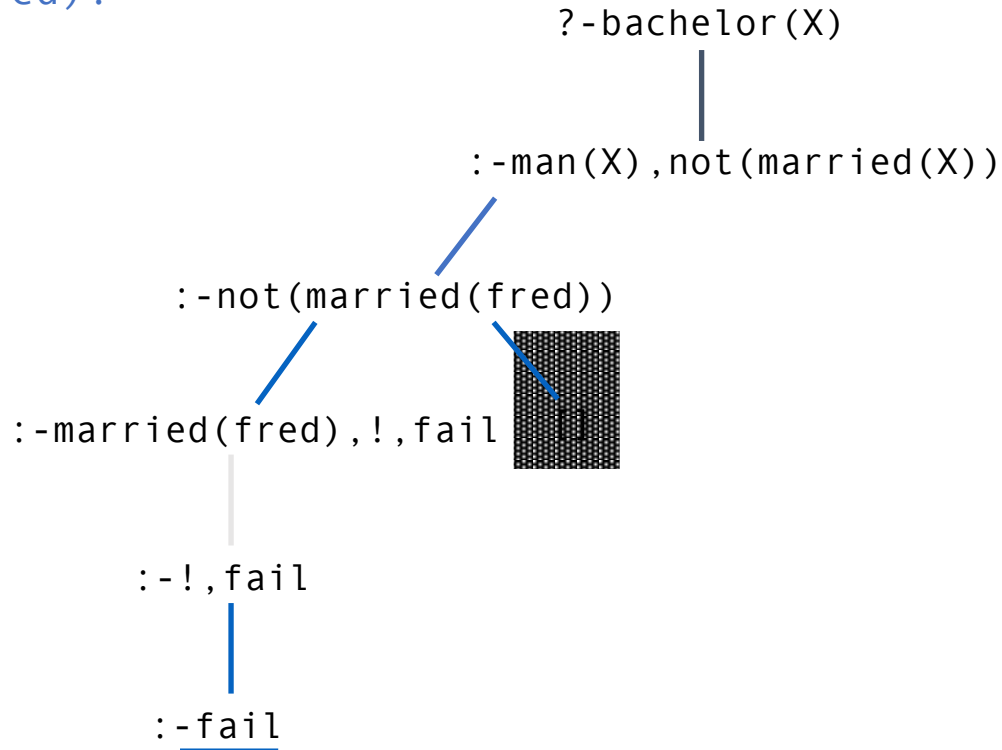




# Prolog's not is unsound – Avoiding the Problem

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

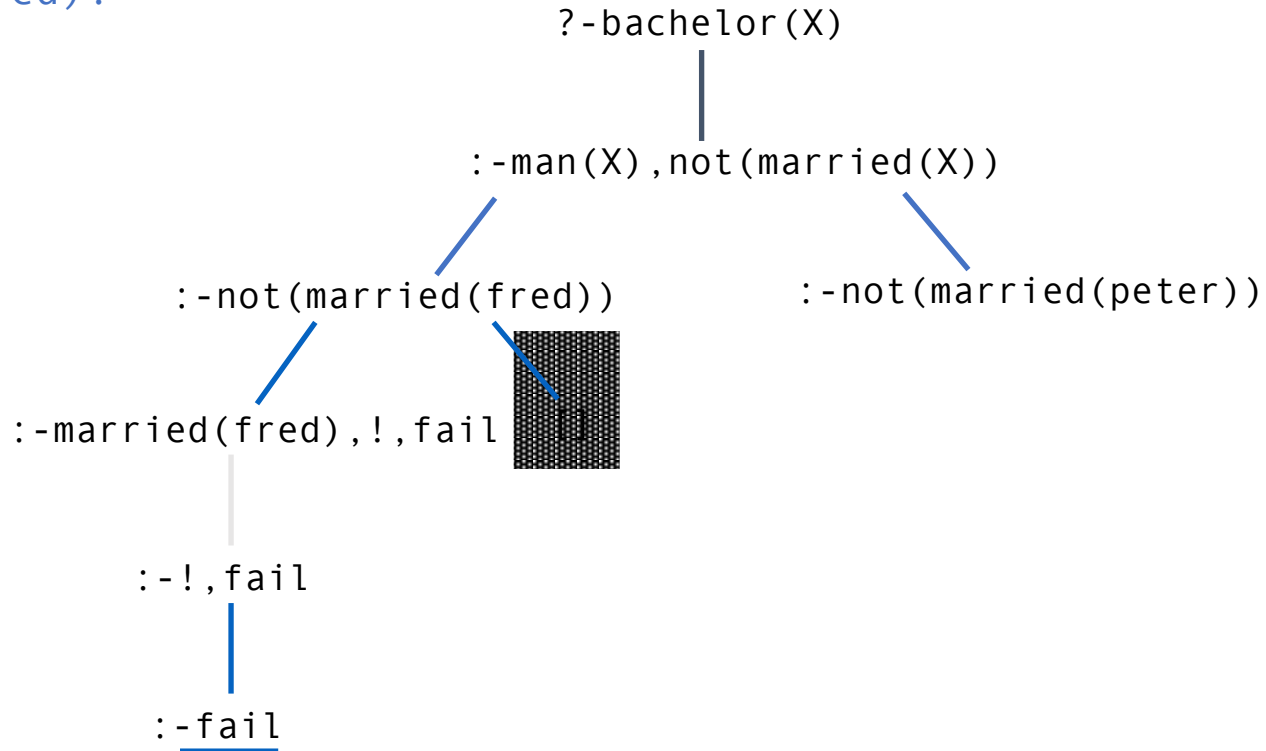
This will "ground" X.



# Prolog's not is unsound – Avoiding the Problem

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

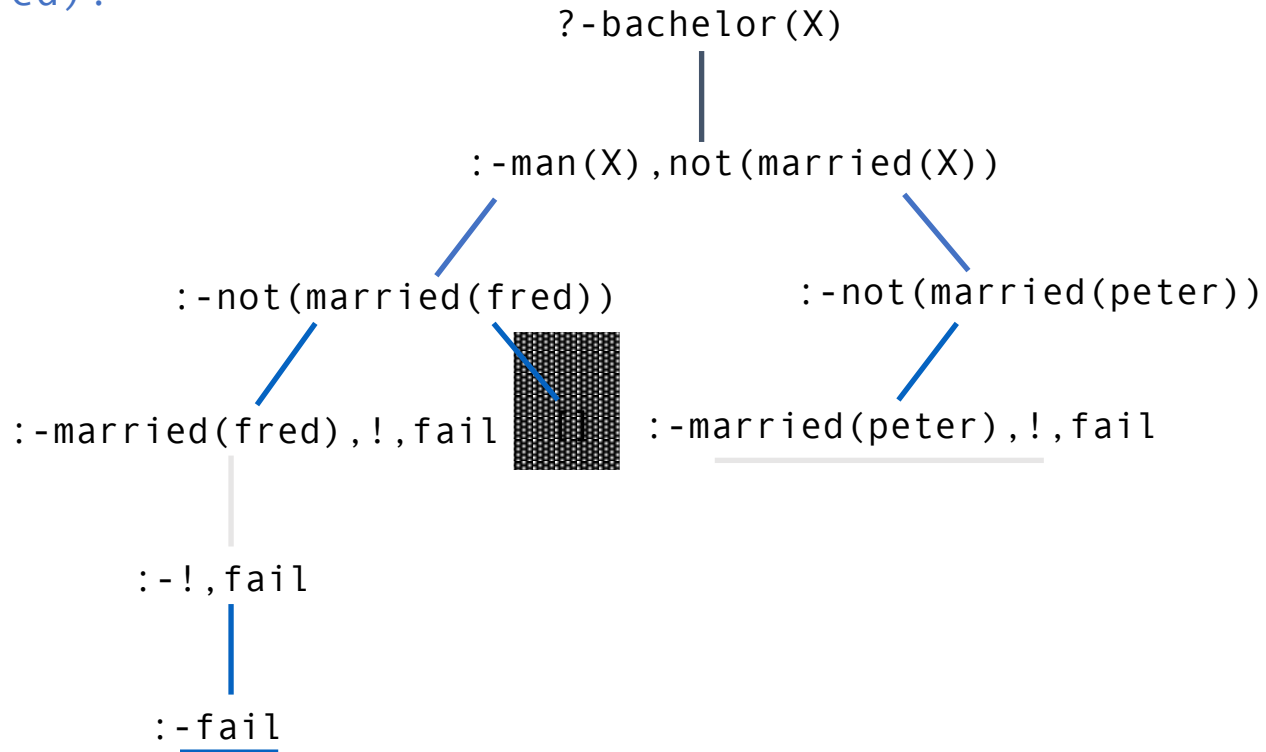
This will "ground" X.



# Prolog's not is unsound – Avoiding the Problem

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

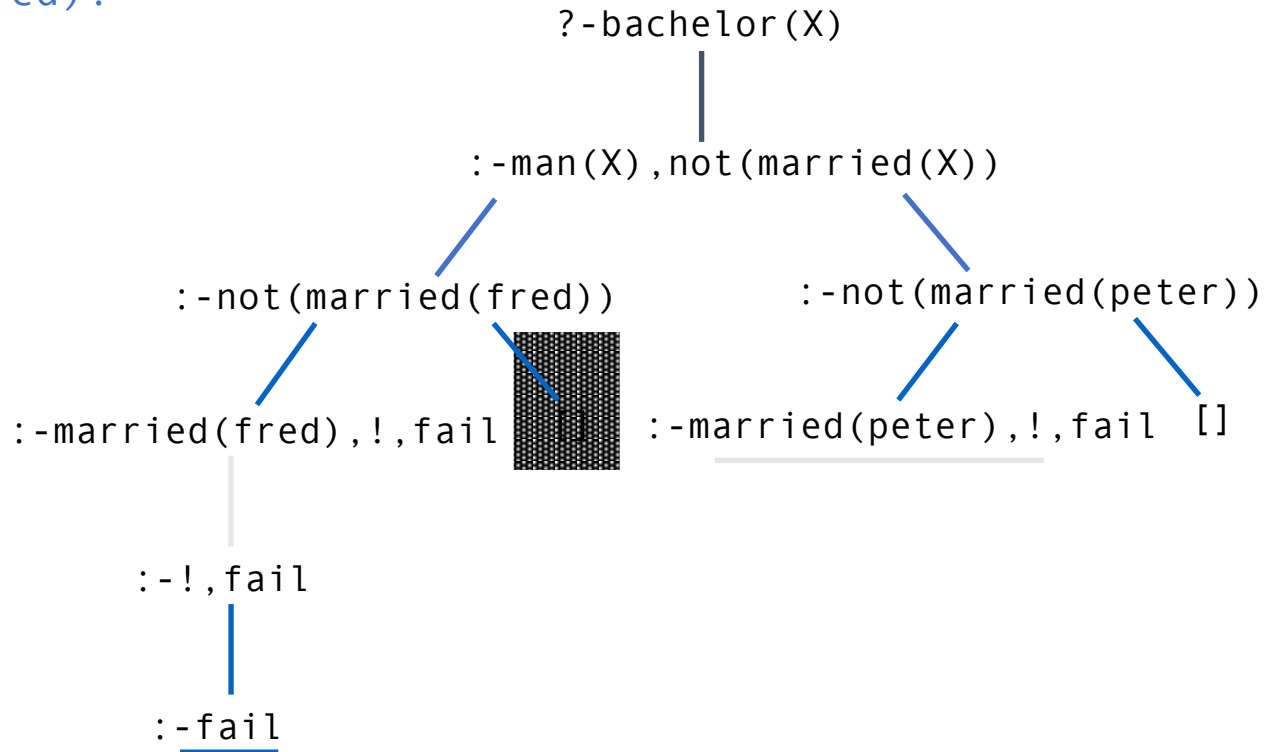
This will "ground" X.



# Prolog's not is unsound – Avoiding the Problem

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

This will "ground" X.





File ▾

Edit ▾

Examples ▾

Help ▾



162 users online

Search



Program × +

```
1 bachelor(X):-man(X),not(married(X)).
2 man(fred).
3 man(peter).
4 married(fred).
5
```

bachelor(X)

X = peter

?- bachelor(X)

Examples ▲

History ▲

Solutions ▲

 table results

Run!

# Arithmetic in Prolog

# Prolog arithmetic vs. unification

?-X is 5+7-3.  
X = 9

?-9 is 5+7-3.  
Yes

?-9 is X+7-3.  
Error in arithmetic expression

?-X is 5\*3+7/2.  
X = 18.5

?-X = 5+7-3.  
X = 5+7-3

?-9 = 5+7-3.  
No

?-9 = X+7-3.  
No

?-X = Y+7-3.  
X = \_947+7-3  
Y = \_947

# Exercise 3.9

`zero(A,B,C,X):- X is (-B + sqrt(B*B - 4*A*C)) / 2*A.`

`zero(A,B,C,X):- X is (-B - sqrt(B*B - 4*A*C)) / 2*A.`





File

Edit

Examples

Help



256 users online

Search



25

Program

Program

Program

Program


Program



```
1 zero(A,B,C,X):-X is (-B + sqrt(B*B - 4*A*C)) / 2*A.
```

```
2 zero(A,B,C,X):-X is (-B - sqrt(B*B - 4*A*C)) / 2*A.
```

```
3
```

 zero(1,0,-1,X)**X** = 1.0**X** = -1.0

```
?- zero(1,0,-1,X)
```

Examples

History

Solutions

table results

Run!