

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 Θ 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 Θ such that $A\Theta = B\Theta$.
- A unifier Θ is more general (*technically speaking, we should be saying more or equally general*) than a unifier Θ' iff there exists a substitution σ such that $\Theta' = \Theta \sigma$.

Unification - Example

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

Example:

A = studies(X, Y)

B = studies(alice, Z)

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

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

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

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 Θ 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, Θ 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 Θ . Add the substitution x/y to Θ .
- 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(t1,...,tk) and y is f(u1,...,uk) **then**
- 8: **Push** on the stack: t1 = u1, t2 = u2, ..., tk = uk .
- 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:

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

$\Theta = \{\}$

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 at swish.swi-prolog.org. The interface includes a navigation bar with icons for back, forward, search, and refresh, and a title bar "SWISH -- SWI-Prolog for SHaring". A sidebar on the left shows a "Program" tab with a file icon and a "+" button, and numbers 1 through 4. The main area features a large owl logo. A red box highlights the query and its results:

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

Results:

- $X = a,$
- $Y = Z, Z = h(a, b)$

Revised Query:

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

At the bottom, there are links for "Examples", "History", "Solutions", and "Run!", along with a checkbox for "table results".

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`

swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

File ▾ Edit ▾ Examples ▾ Help ▾

197 users online

Search

Program +

unify_with_occurs_check(X, f(X))

false

?- unify_with_occurs_check(X, f(X))

Examples ▾ History ▾ Solutions ▾

table results Run!

The screenshot shows the SWISH web interface for SWI-Prolog. At the top, there's a navigation bar with tabs for File, Edit, Examples, and Help. On the right side of the header, there's a search bar, a user count of "197 users online", and some social sharing icons. The main workspace is titled "Program" and contains the following text:

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

A large red oval highlights the second line of the output, which is "false". Below the workspace, there are links for Examples, History, and Solutions, along with checkboxes for "table results" and "Run!". The background features a cartoon owl logo.

swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

File ▾ Edit ▾ Examples ▾ Help ▾ 195 users online

Search

Program +

1
2
3
4

unify_with_occurs_check(X, f(X))
false

X = f(X)
X = f(X)

?- X = f(X)

Examples ▾ History ▾ Solutions ▾ table results Run!

The screenshot shows the SWISH web interface for SWI-Prolog. At the top, there's a navigation bar with icons for back, forward, search, and user status (195 users online). Below the bar, there's a menu with 'File', 'Edit', 'Examples', 'Help', and a user icon. To the right of the menu is a search bar with a magnifying glass icon. On the left, there's a sidebar with a gear icon labeled 'SWISH', a 'Program' tab (selected), and a '+' button. The main area has a vertical scroll bar on the right. A large, semi-transparent owl logo is in the background. In the center, a red oval highlights a query window. The window contains the following text:
unify_with_occurs_check(X, f(X))
false
X = f(X)
X = f(X)
?- X = f(X)
At the bottom of the interface are buttons for 'Examples', 'History', 'Solutions', a checkbox for 'table results', and a blue 'Run!' button.

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).
```

?- **student_of(S,peter)**

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

: - **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).
```

?- `student_of(S,peter)`

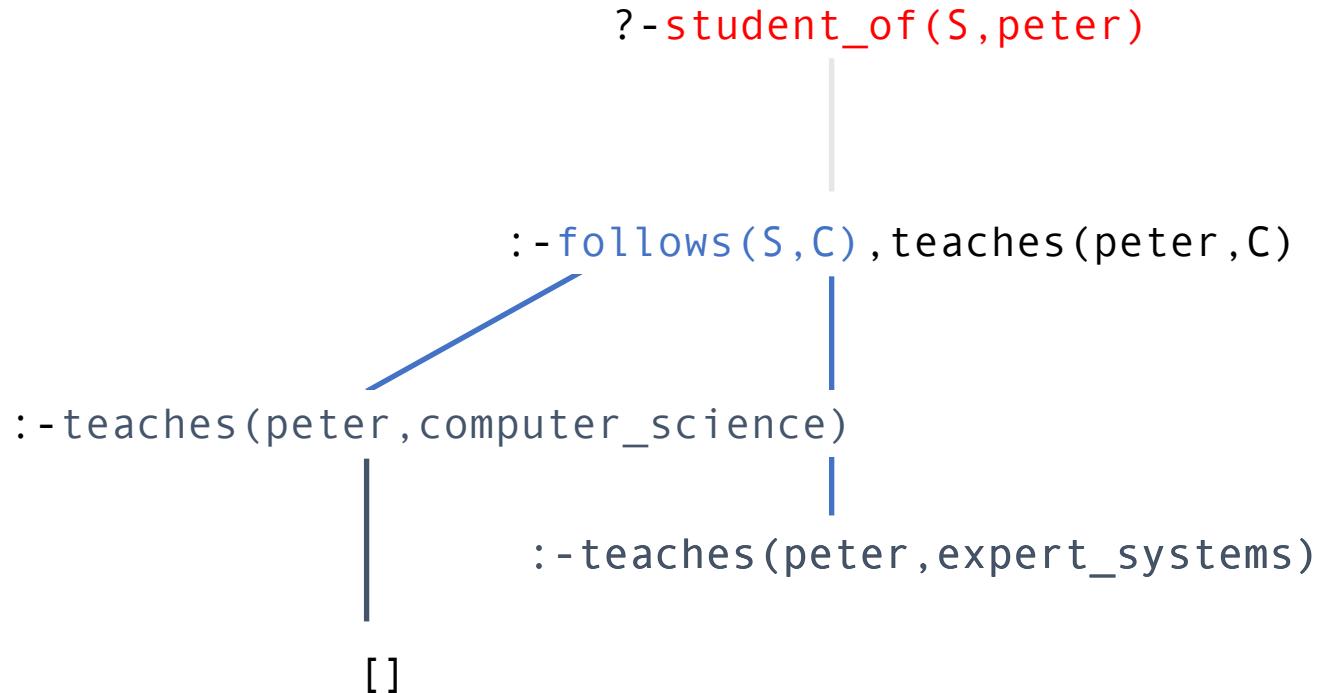
: - `follows(S,C), teaches(peter,C)`

: - `teaches(peter,computer_science)`

[]

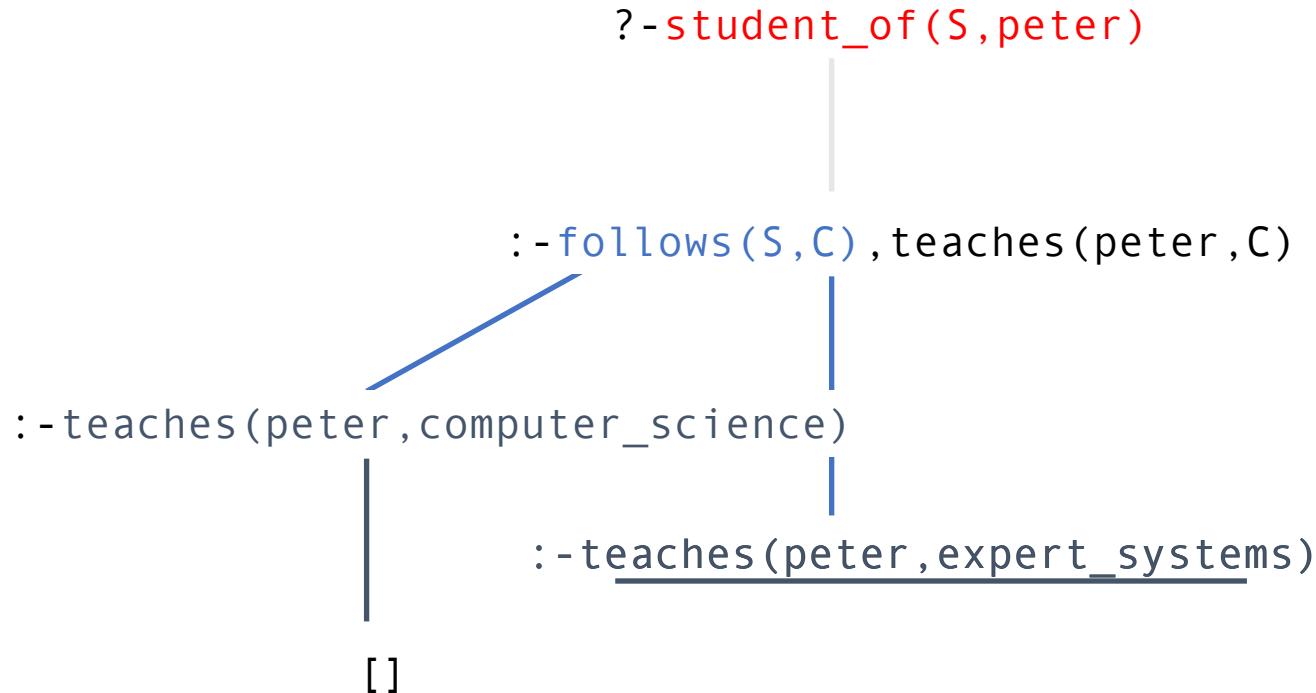
SLD-trees

```
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).
```



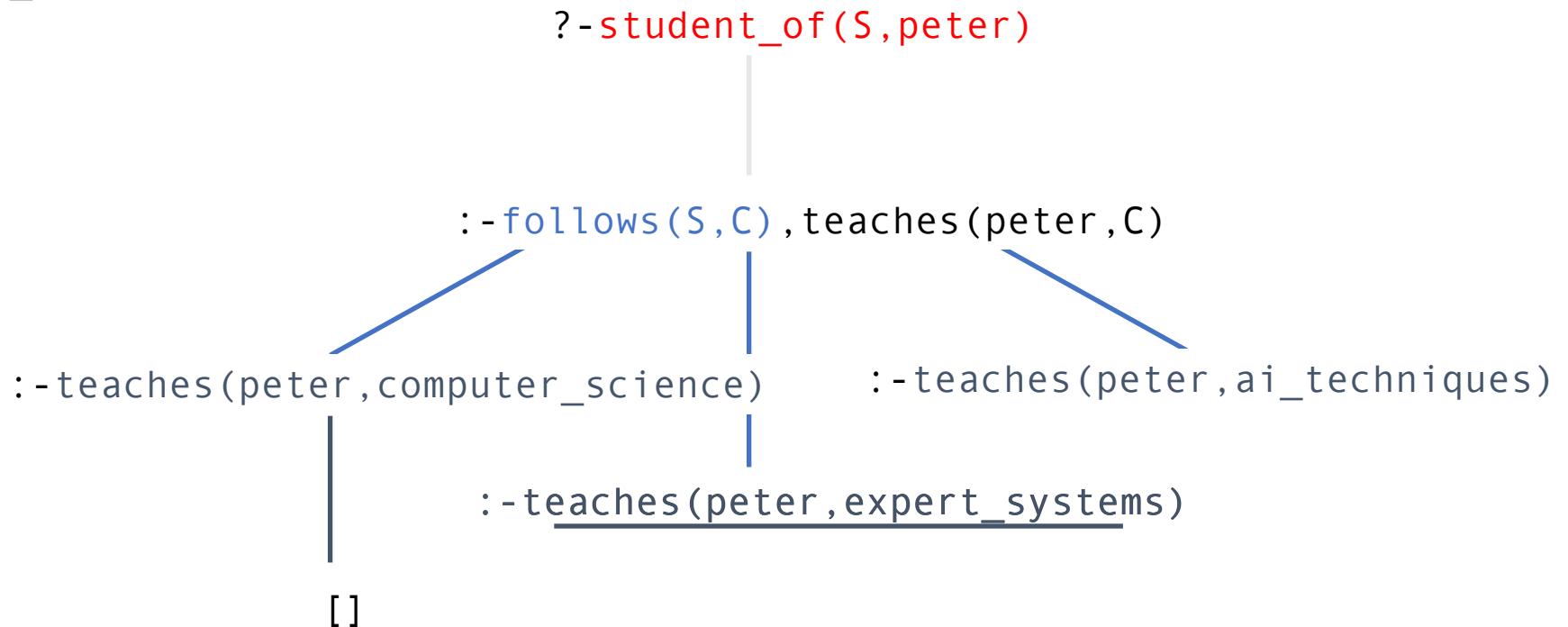
SLD-trees

```
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).
```



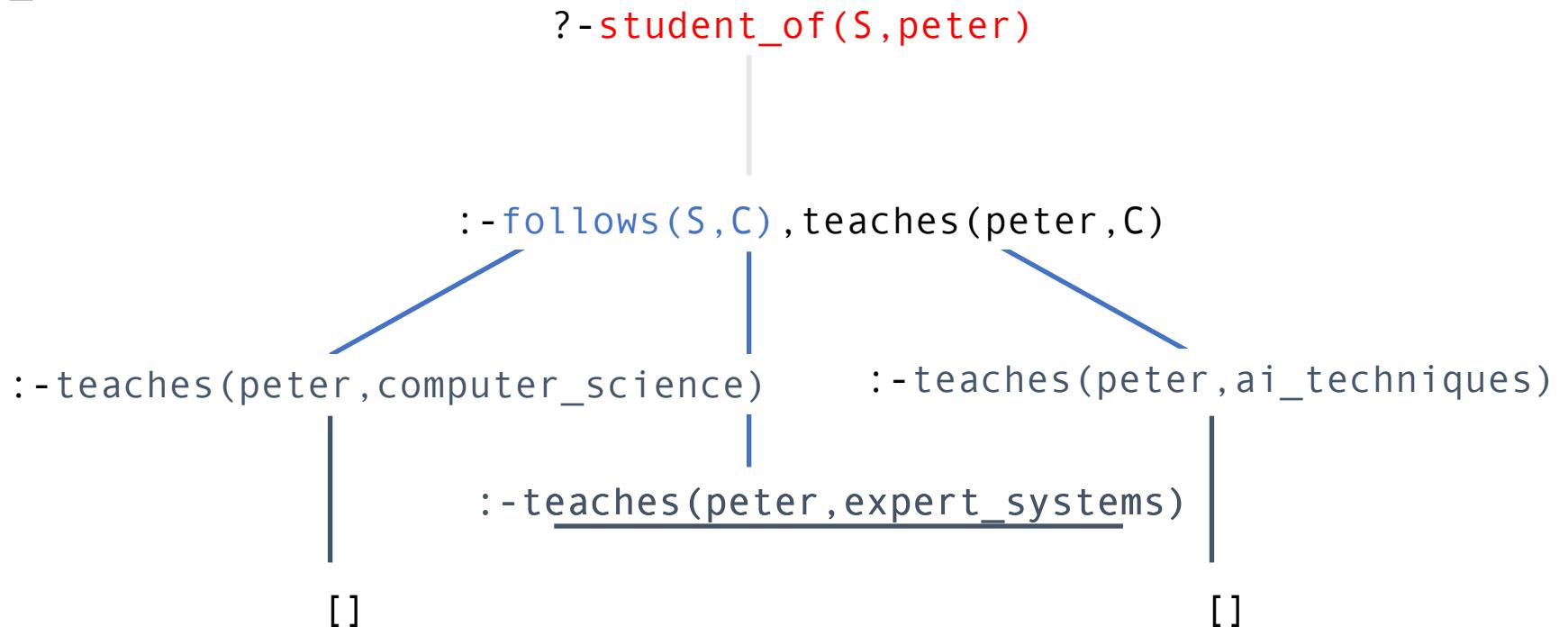
SLD-trees

```
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).
```

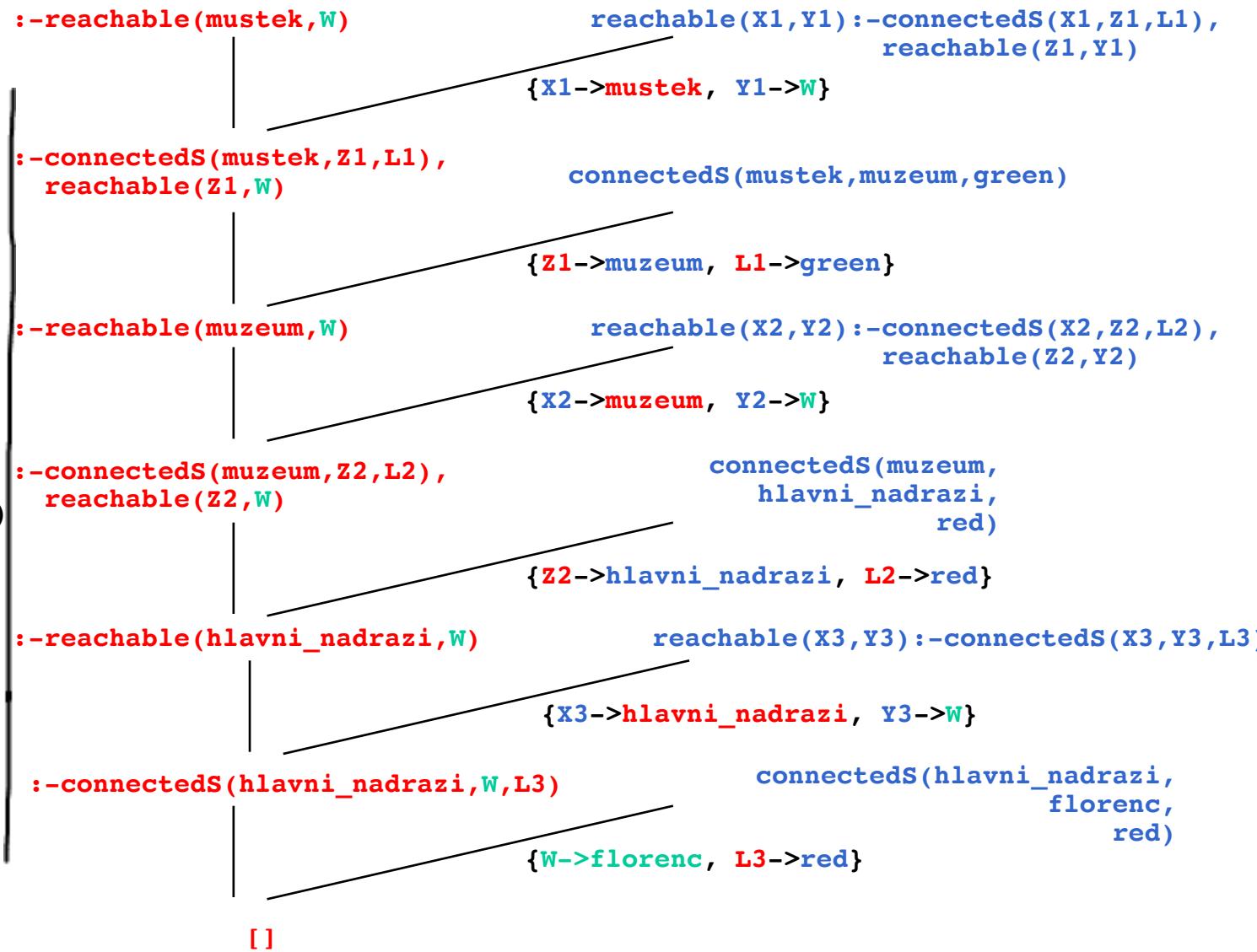
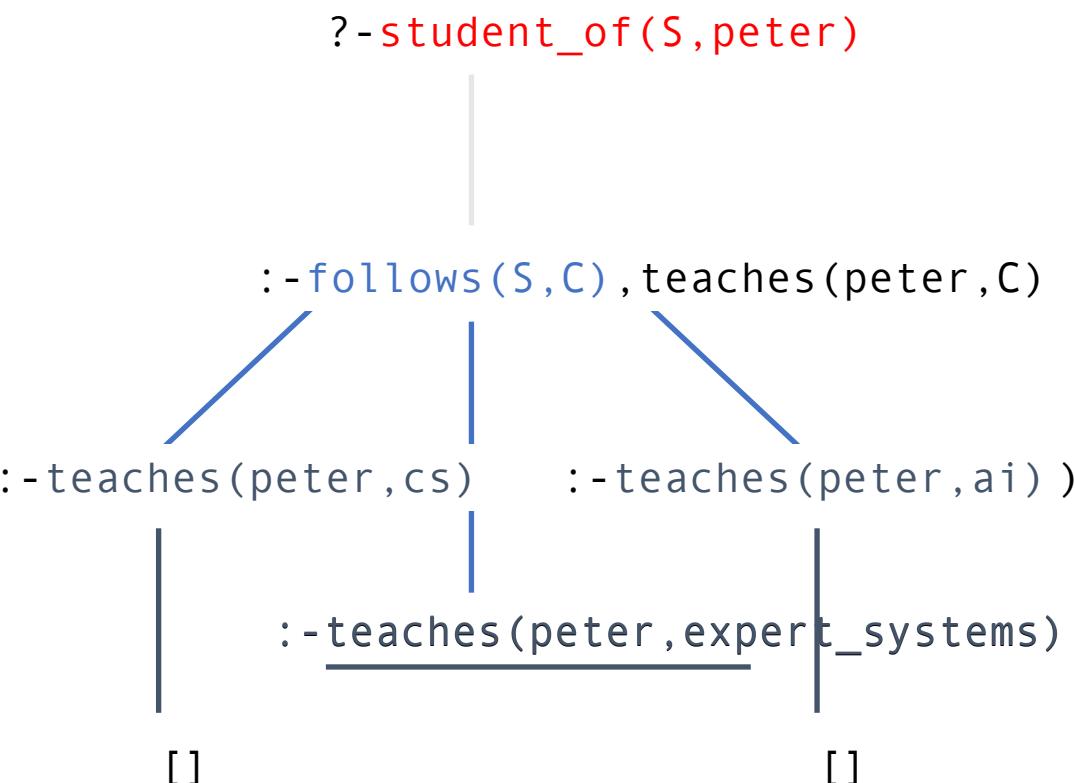


SLD-trees

```
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 vs Proof Trees



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)
```

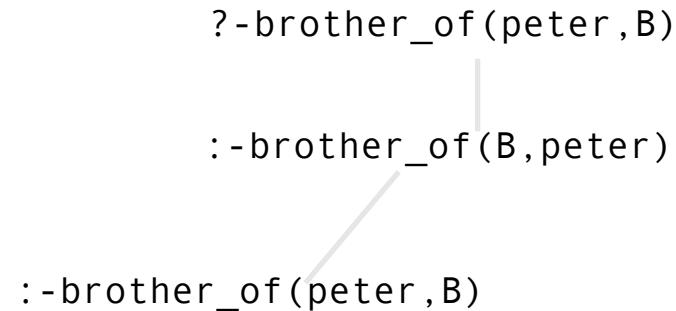
Infinite SLD-trees

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

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

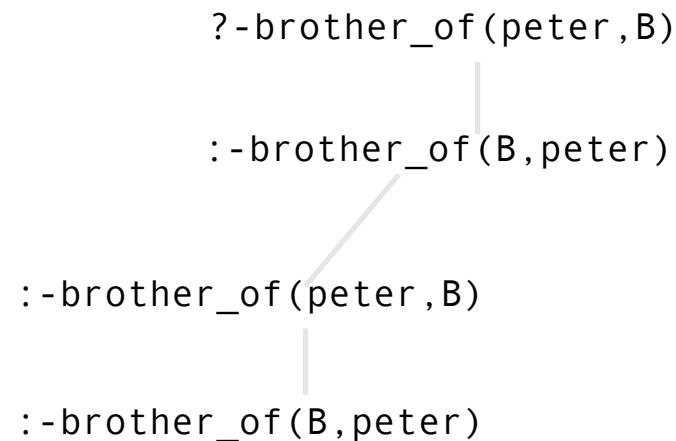
Infinite SLD-trees

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



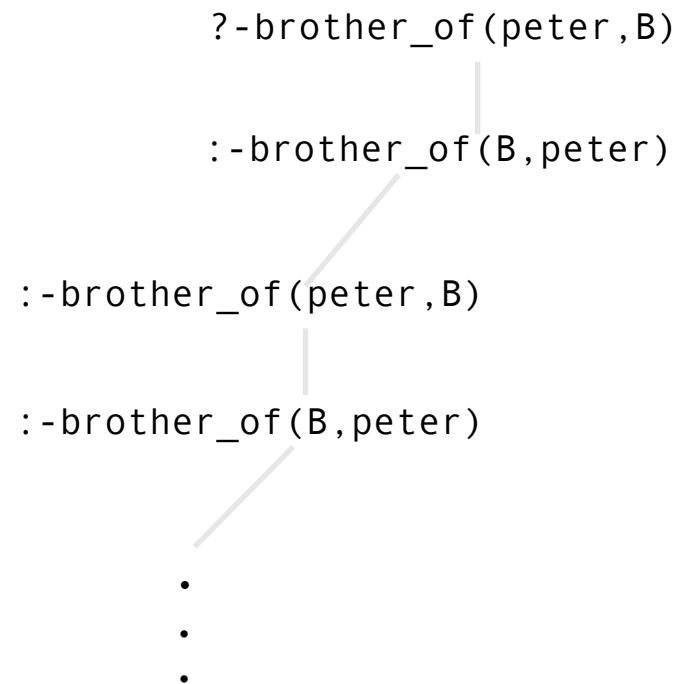
Infinite SLD-trees

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



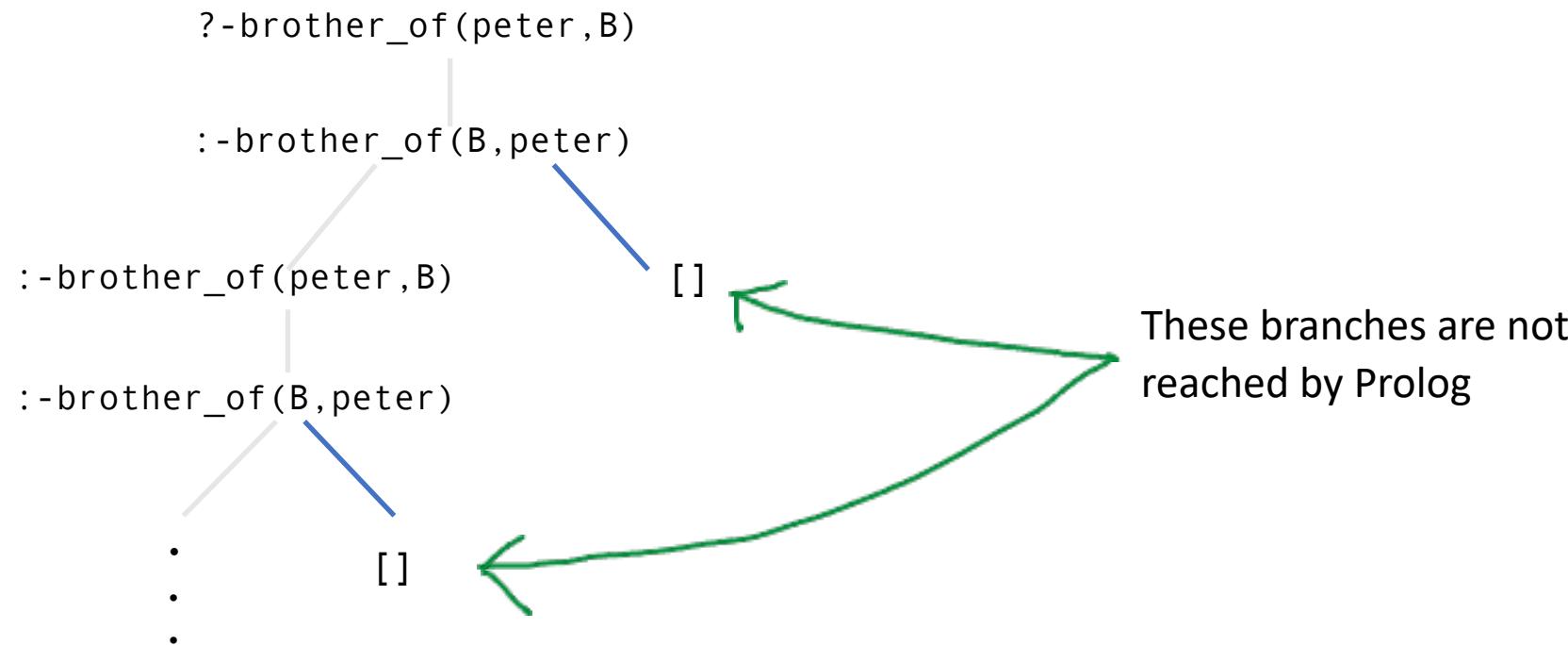
Infinite SLD-trees

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



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-based Prolog interface. On the left, a green oval highlights the program code in the 'Program' tab:

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

In the main window, a query is being run:

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

This results in the following error message:

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)

A note at the bottom suggests increasing the stack limit:

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

So Prolog will get stuck...

The screenshot shows the SWISH web interface for SWI-Prolog. The top navigation bar includes links for File, Edit, Examples, and Help. On the left, there's a sidebar with a program editor containing the following code:

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

The main workspace shows a query window with the goal `brother_of(peter, B)`. The response is a stack overflow error message:

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)

A note at the bottom suggests increasing the stack limit:

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

The query `brother_of(peter, B)` is highlighted with a green oval in the bottom workspace.

So Prolog will get stuck...

The screenshot shows the SWISH web-based Prolog interface. On the left, there is a code editor window titled "Program" containing the following Prolog code:

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

To the right, a query window is open with the goal `?- brother_of(peter, B)`. A large green oval highlights the error message displayed in the query window:

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.

The URL in the browser bar is `swish.swi-prolog.org`.

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).
```

Show the previous page



File ▾

Edit ▾

Examples ▾

Help ▾



191 users online

Search



```
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)

Show the previous page



File ▾

Edit ▾

Examples ▾

Help ▾



191 users online

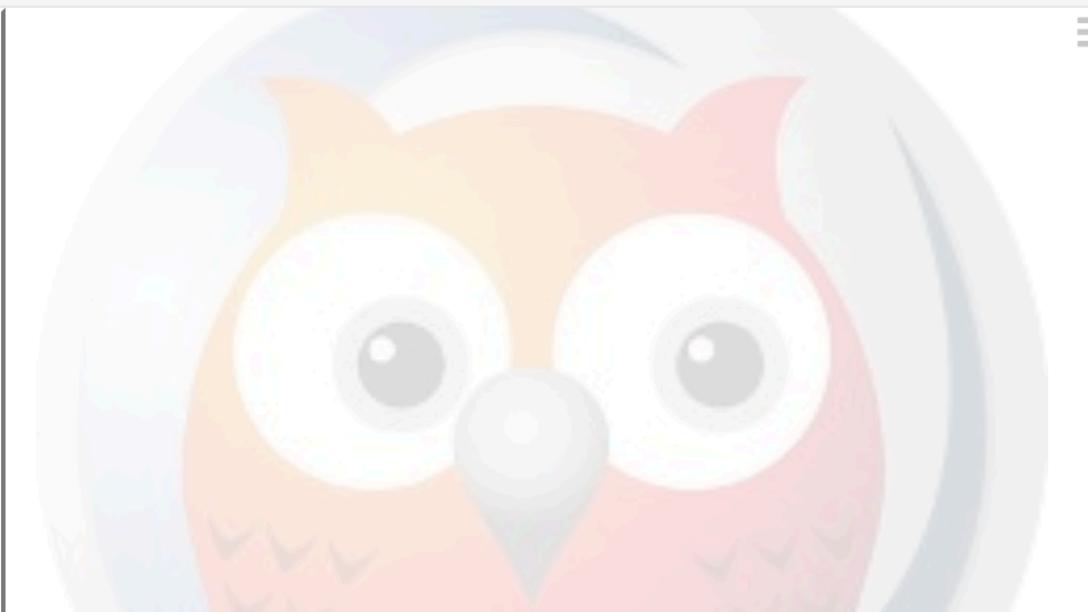
Search



Owl 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)

Examples ▾ History ▾ Solutions ▾

table results Run!

Show the previous page



Search



191 users online

Owl Program X

File ▾

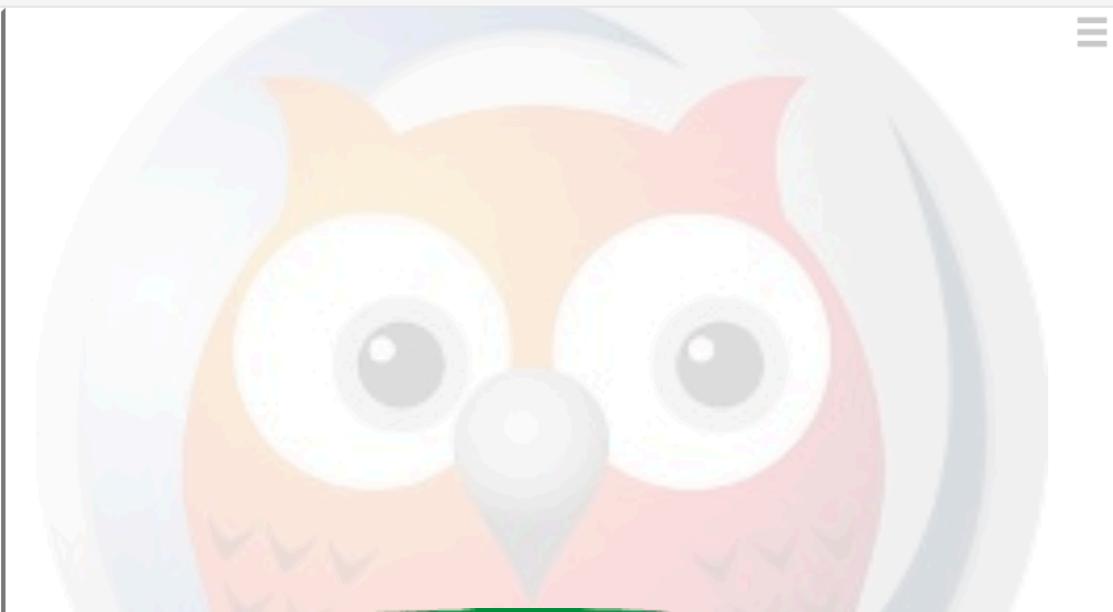
Edit ▾

Examples ▾

Help ▾

+

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



owl brother_of(peter, B)

B = paul

Next 10 100 1,000 Stop

?- brother_of(peter, B)

Examples ▾ History ▾ Solutions ▾

table results Run!

The other branches can still be accessed...

The screenshot shows the SWISH -- SWI-Prolog for SHaring web application. In the top-left corner, there's a sidebar with a logo, a search bar containing 'Search', and a bell icon with '25' notifications. The main area has tabs for 'Program' (selected) and '+'. The code editor contains the following Prolog code:

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

In the bottom-right panel, a query is being run:

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

The results pane shows the variable **B** being instantiated to **paul** 10 times, with a green annotation '10x' pointing to the result list. Below the results is a pagination control with buttons for 'Next', '10', '100', '1,000', and 'Stop'. At the bottom of the page, there are links for 'Examples▲', 'History▲', 'Solutions▲', and buttons for 'table results' and 'Run!'. The status bar at the bottom indicates '183 users online'.

The other branches can still be accessed...

The screenshot shows the SWISH web interface at swish.swi-prolog.org. The top navigation bar includes links for File, Edit, Examples, Help, and a search bar. A sidebar on the left contains a program editor with the following code:

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

The main query window displays the result of the query `?- brother_of(peter, B)`. The results are listed in a scrollable table, with the first ten results highlighted by a red oval. A green oval highlights the pagination controls below the table, which include buttons for Next, 10, 100, 1,000, and Stop. The text "10x" is written in green next to the pagination controls. At the bottom of the query window, the query `?- brother_of(peter, B)` is repeated.

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)
```

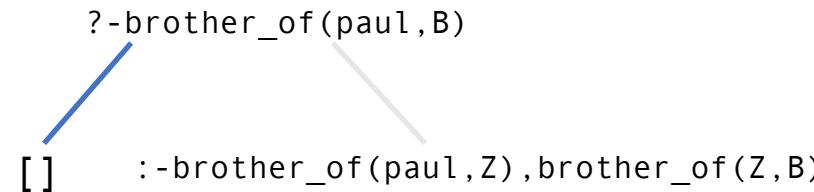
Infinite SLD-trees (Another Example)

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

?-brother_of(paul,B)
[]

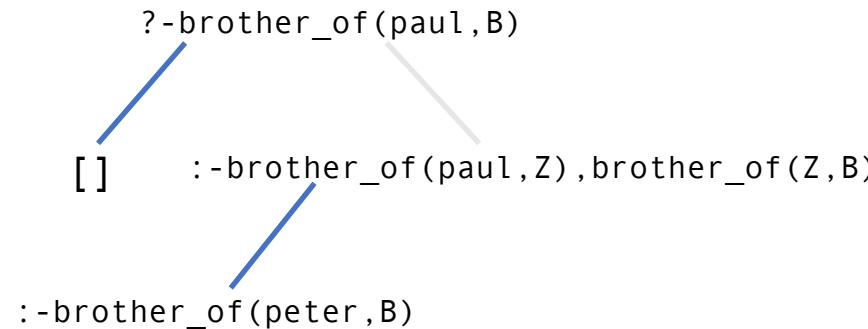
Infinite SLD-trees (Another Example)

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



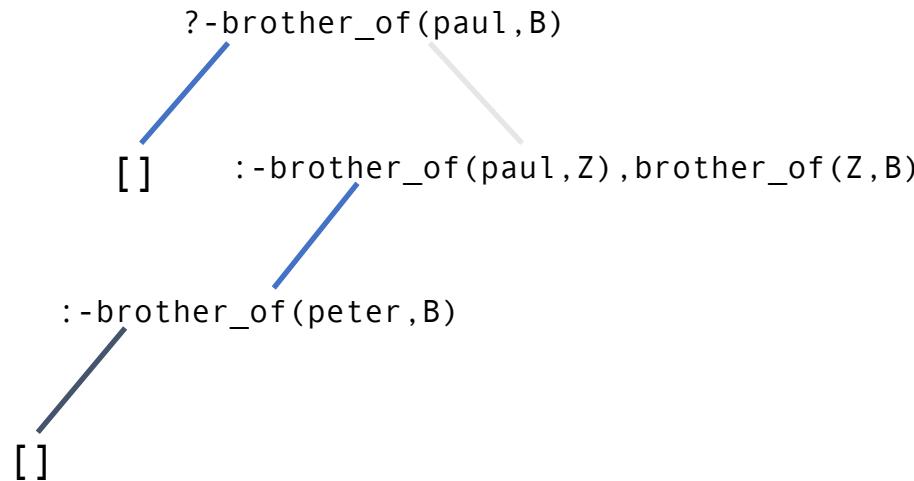
Infinite SLD-trees (Another Example)

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



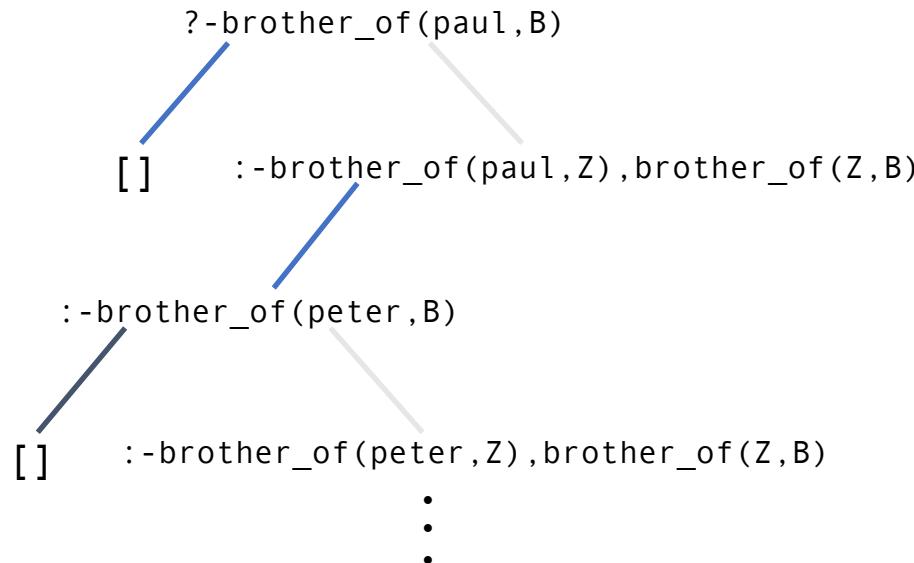
Infinite SLD-trees (Another Example)

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



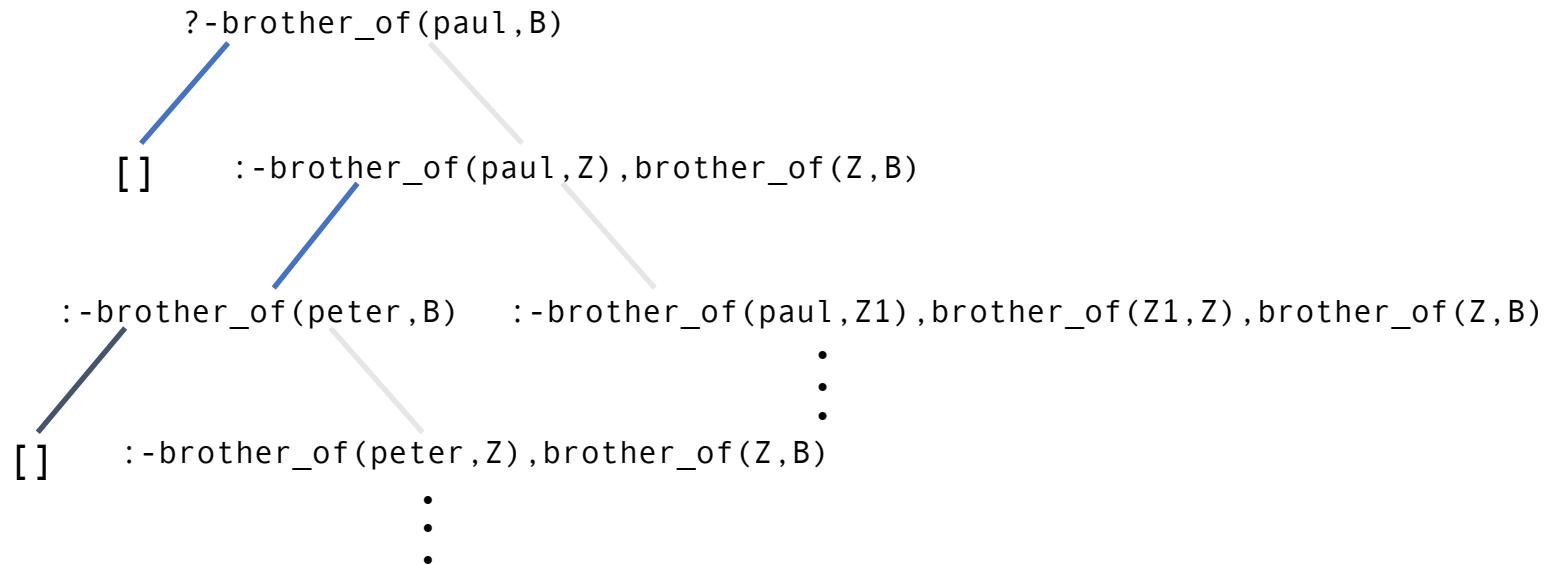
Infinite SLD-trees (Another Example)

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



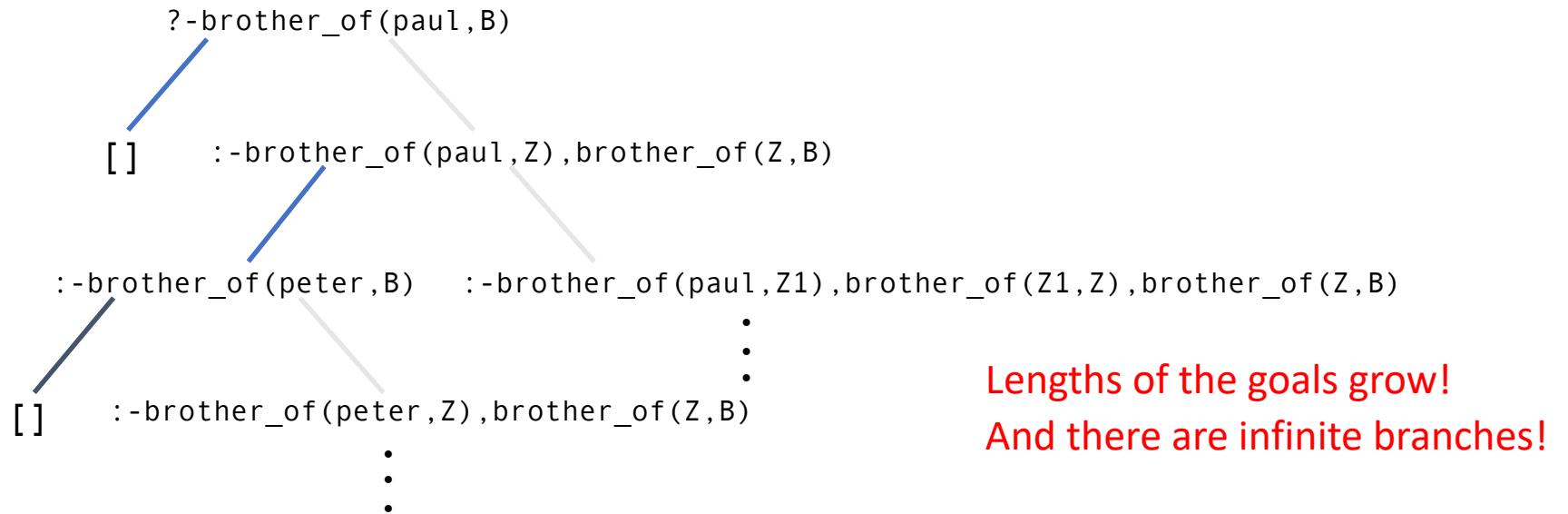
Infinite SLD-trees (Another Example)

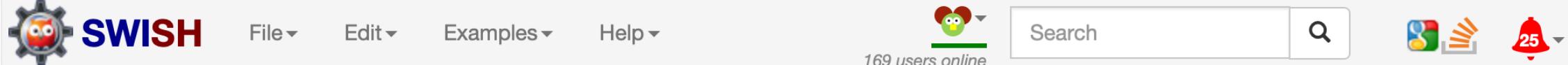
```
brother_of(paul,peter).  
brother_of(peter,adrian).  
brother_of(X,Y) :- brother_of(X,Z), brother_of(Z,Y).
```



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 +

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

?- brother_of(paul, B)

Examples▲ History▲ Solutions▲

table results Run!

SWISH -- SWI-Prolog for SHaring



188 users online

Search 



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)

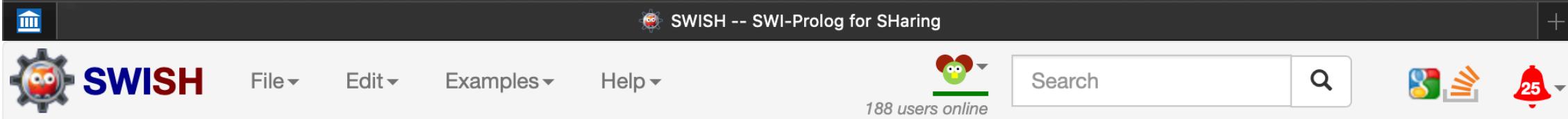
B = peter

Next 10 100 1,000 Stop

?- brother_of(paul, B)

Examples▲ History▲ Solutions▲ table results Run!

Open "https://swish.swi-prolog.org" in a new tab



Program

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

brother_of(paul, B)

B = peter

Next | 10 | 100 | 1,000 | Stop

?- brother_of(paul, B)

Examples▲ History▲ Solutions▲ table results Run!

SWISH -- SWI-Prolog for SHaring



File ▾ Edit ▾ Examples ▾ Help ▾



188 users online



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)

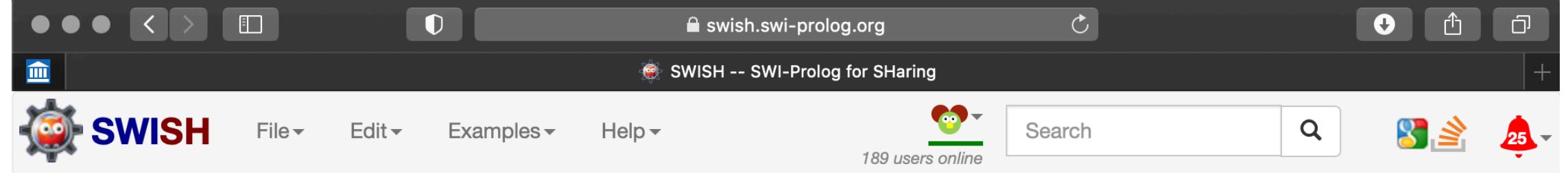
B = peter

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).
```

189 users online

brother_of(paul, B)

B = peter
B = adrian

Next 10 100 1,000 Stop

?- brother_of(paul, B)

Examples▲ History▲ Solutions▲ table results Run!

swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

189 users online

Search

Program

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

brother_of(paul, B)

B = peter
B = adrian

Next 10 100 1,000 Stop

?- brother_of(paul, B)

Examples▲ History▲ Solutions▲

table results Run!

An orange arrow points from the 'brother_of(paul, B)' query result towards the '?-' prompt.

swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

185 users online

Search

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

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)

Examples▲ History▲ Solutions▲

table results Run!

Exercise 3.2

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

?- list(L)

?- list(L) .

Exercise 3.2

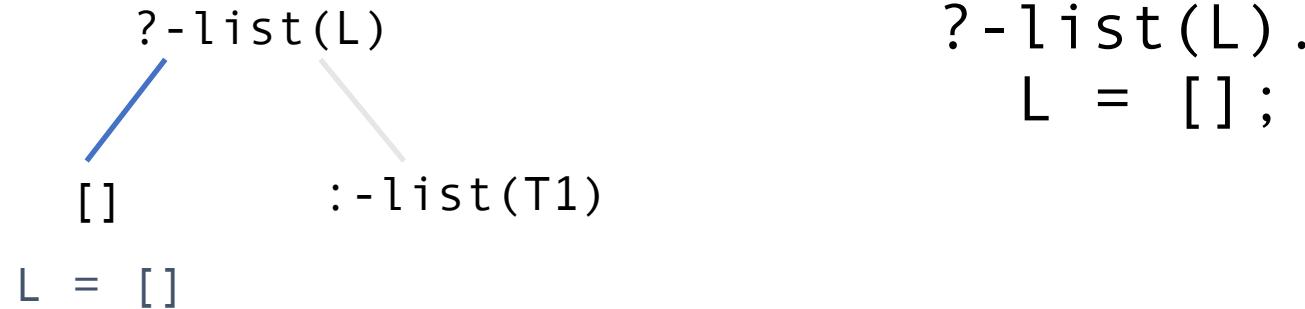
```
list([]).  
list([H|T]) :- list(T).
```

?- list(L)
[]
L = []

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

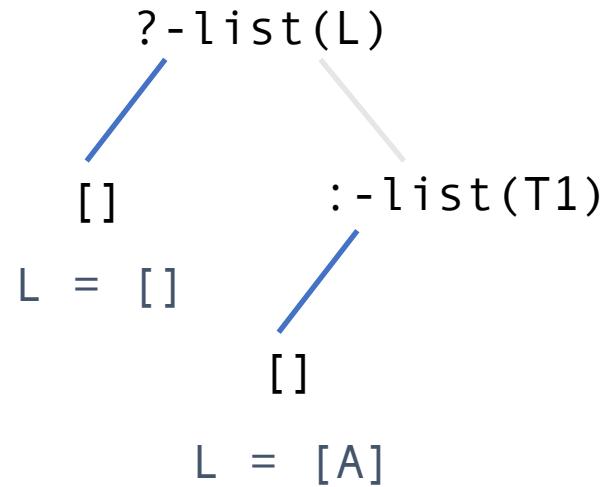
Exercise 3.2

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



Exercise 3.2

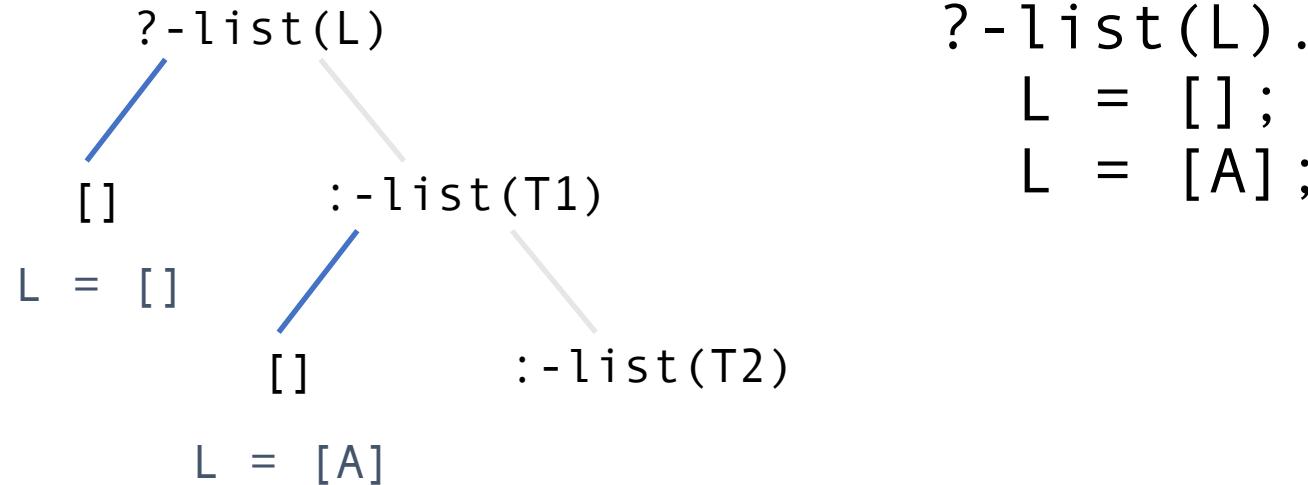
```
list([]).  
list([H|T]) :- list(T).
```



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

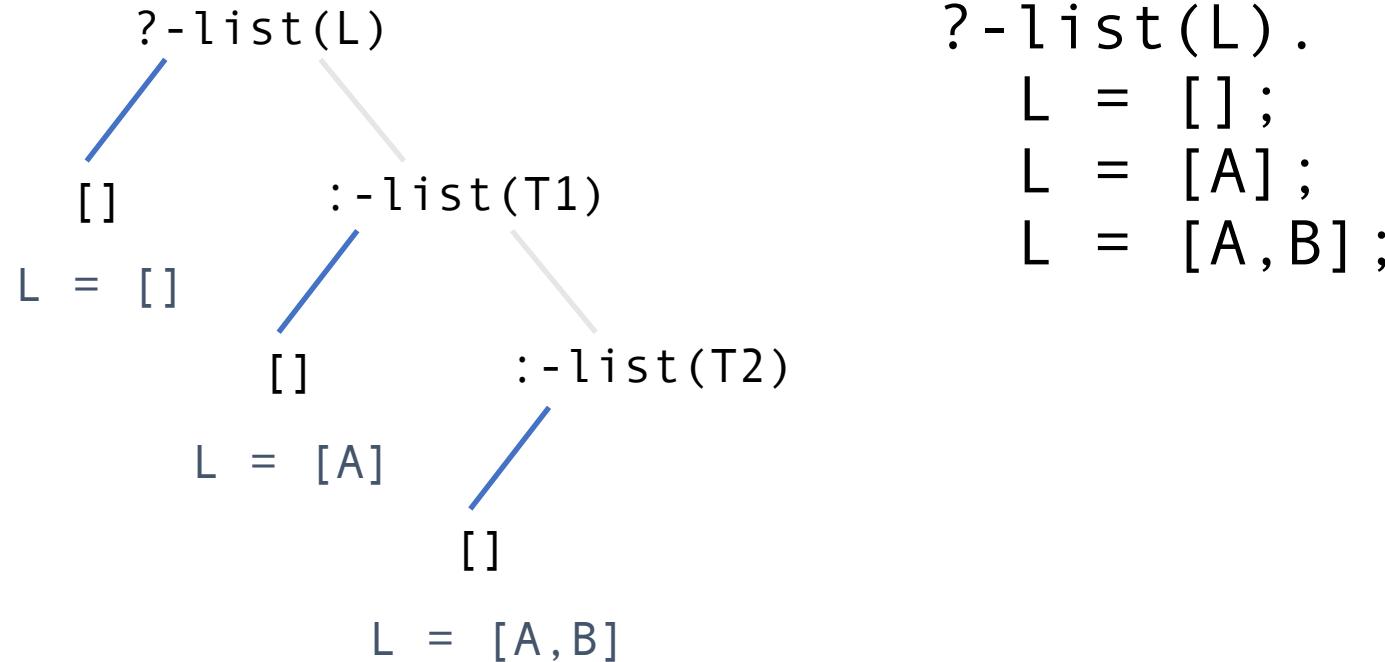
Exercise 3.2

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



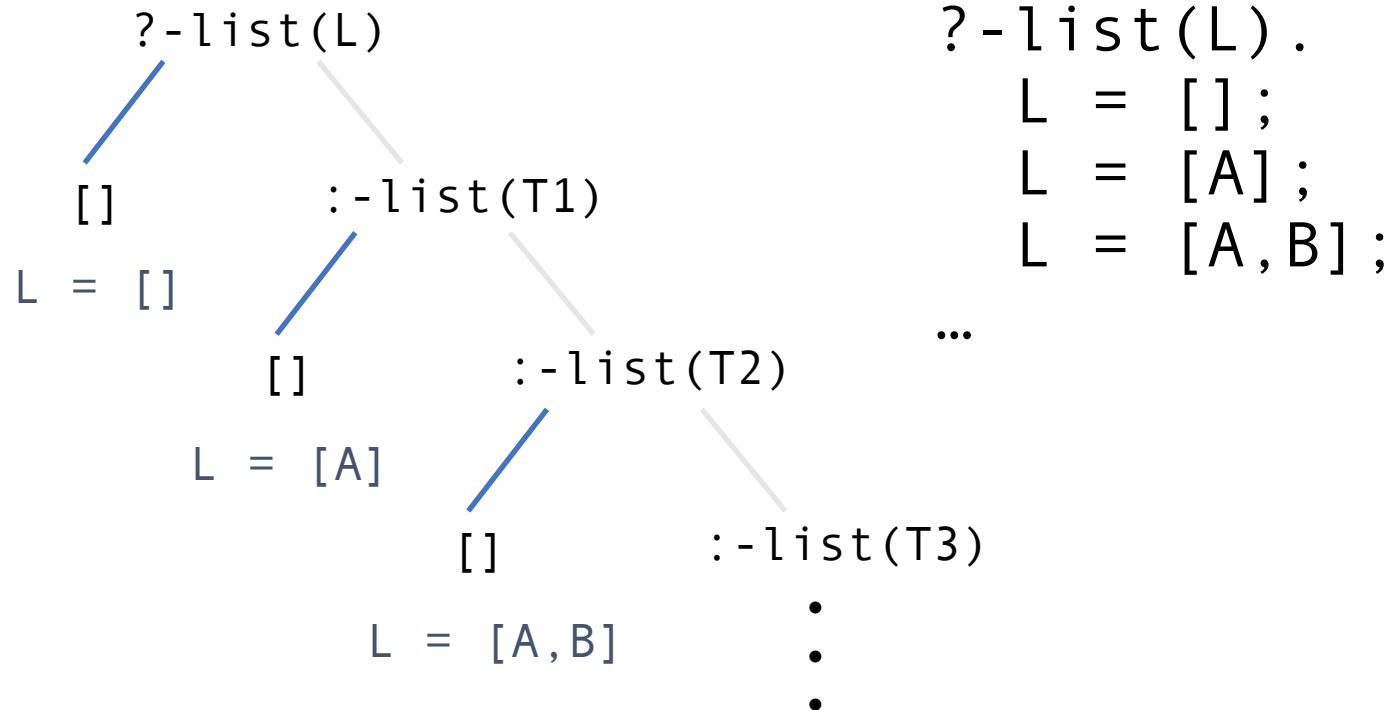
Exercise 3.2

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



Exercise 3.2

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



swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

190 users online

Search

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(L)
      ↗
[]          :- p(H1), plist(T1)
L = []
```

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

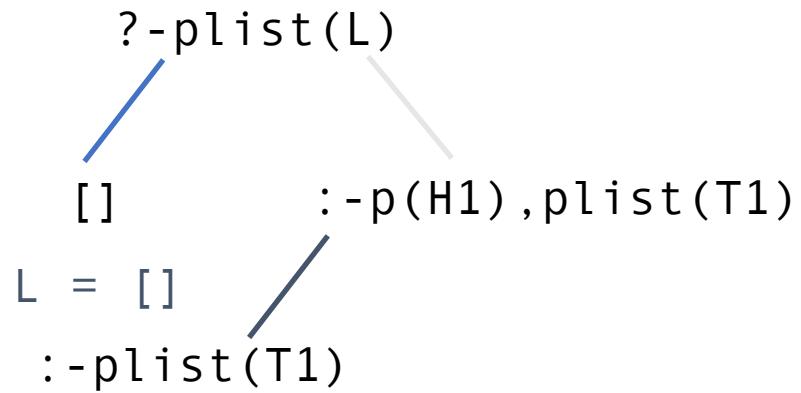
```
p(1).           p(2).
```

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

Another example

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

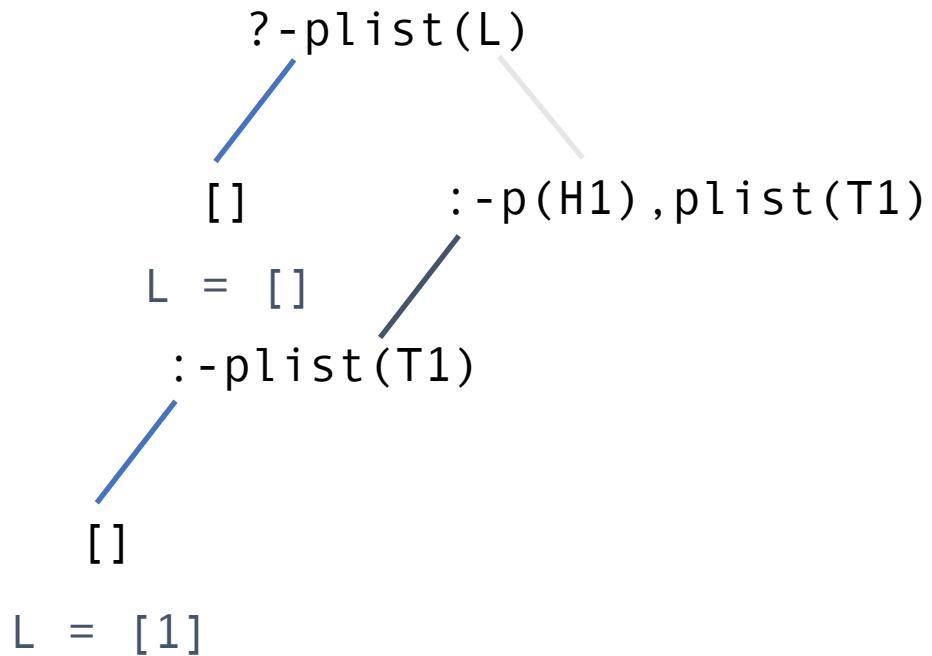
```
p(1).          p(2).
```



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

Another example

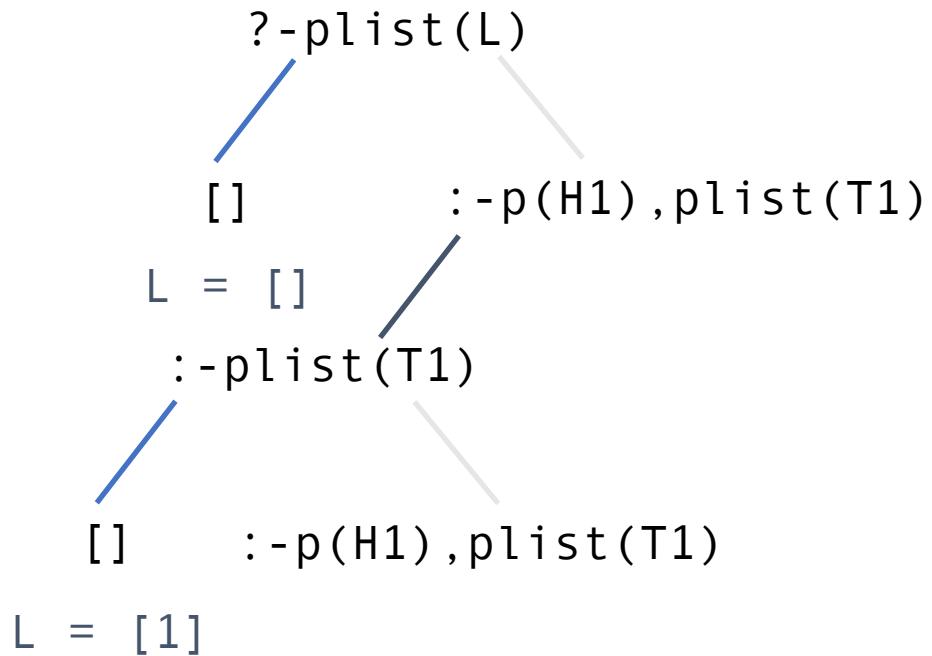
```
plist([]).  
plist([H|T]) :-  
    p(H), plist(T).  
  
p(1).          p(2).
```



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

Another example

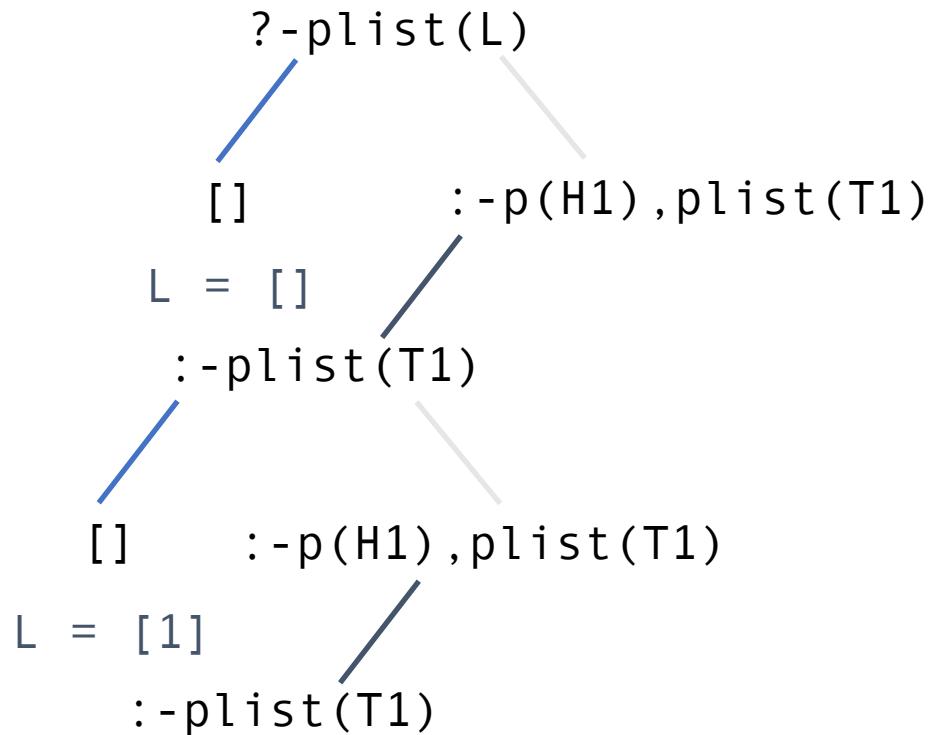
```
plist([]).  
plist([H|T]) :-  
    p(H), plist(T).  
  
p(1).          p(2).
```



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

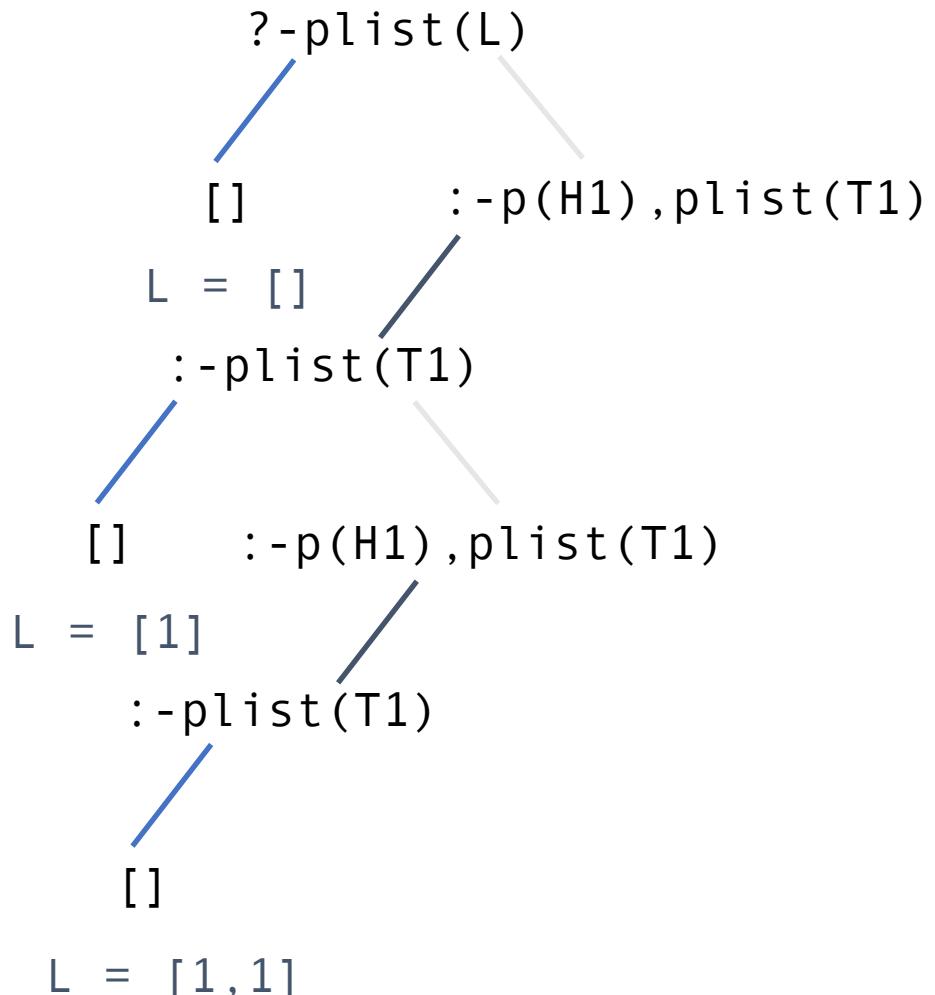
Another example

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



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

Another example

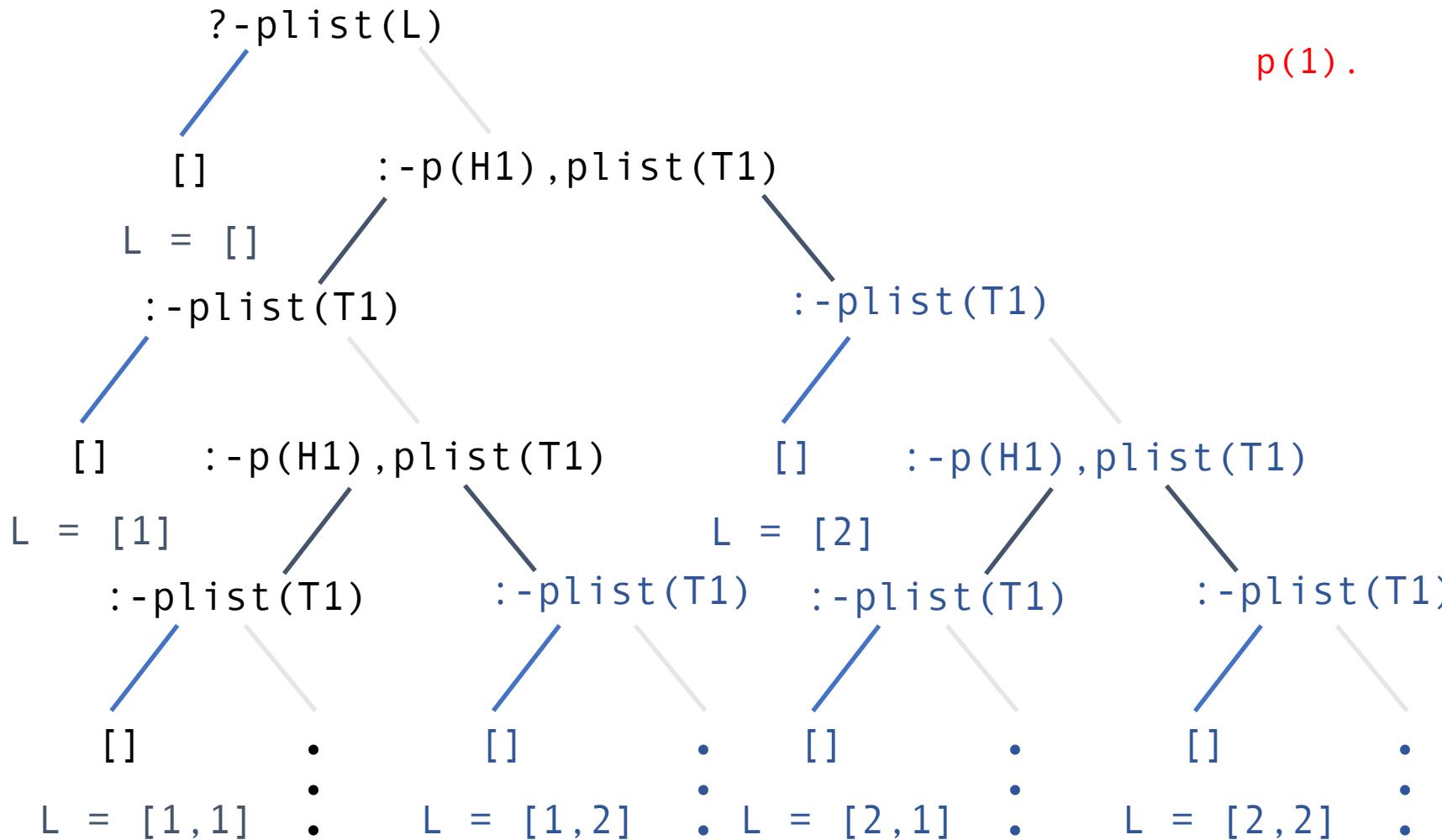


```
plist([]).
plist([H|T]) :-
    p(H), plist(T).

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

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

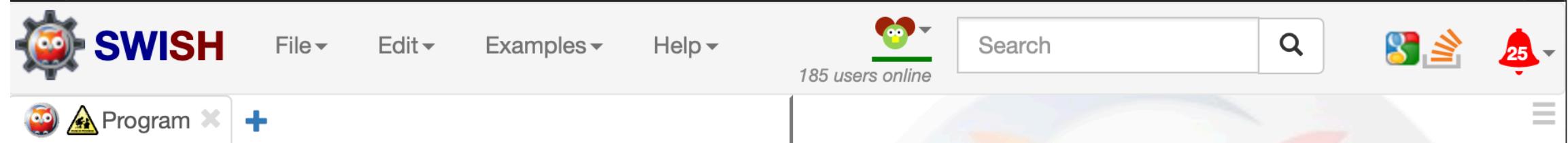
Another example (and so on)



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

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

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



Program plist([]).

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

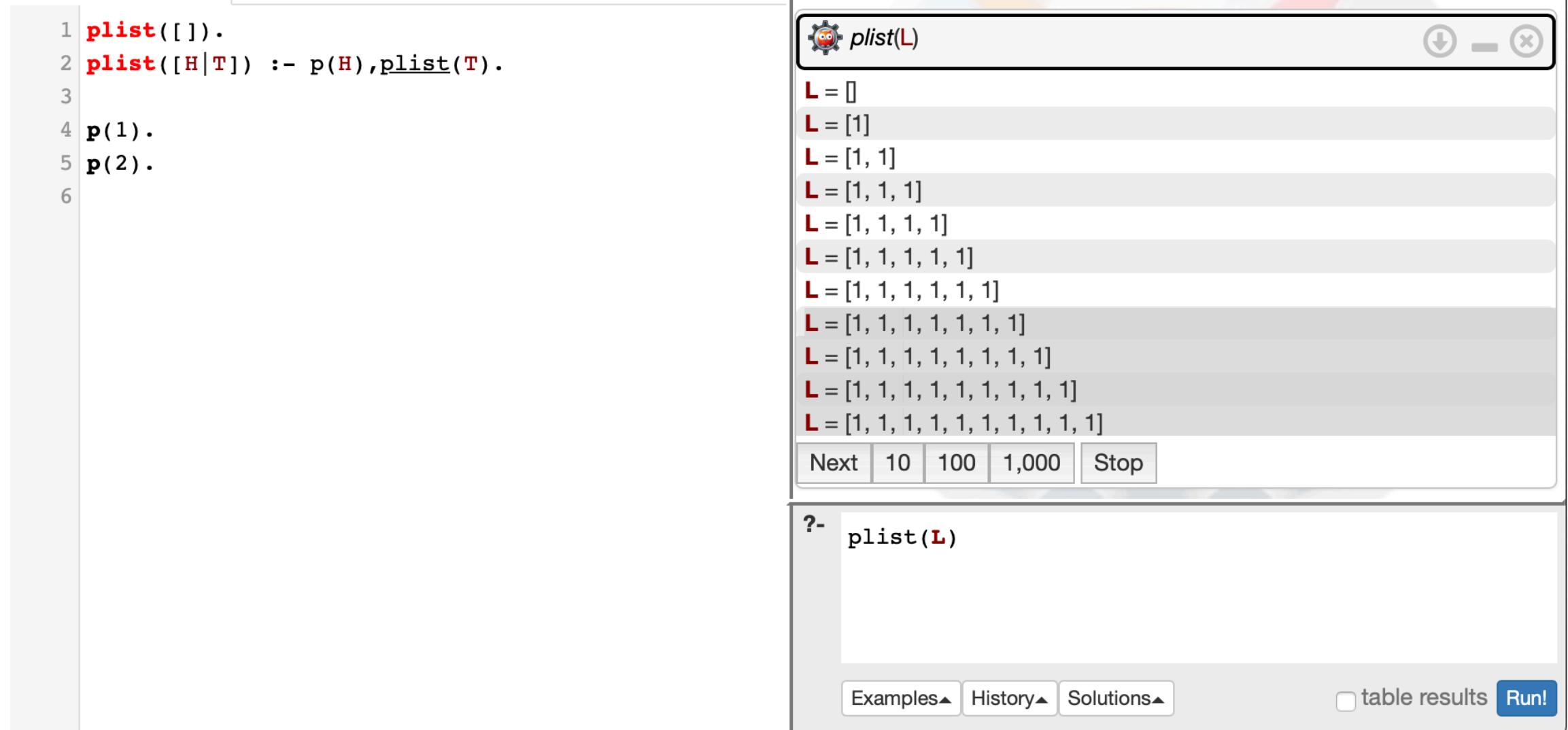
1
2
3
4
5
6

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]

Next | 10 | 100 | 1,000 | Stop

?- plist(L)



swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

File ▾ Edit ▾ Examples ▾ Help ▾ 180 users online

Search 

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

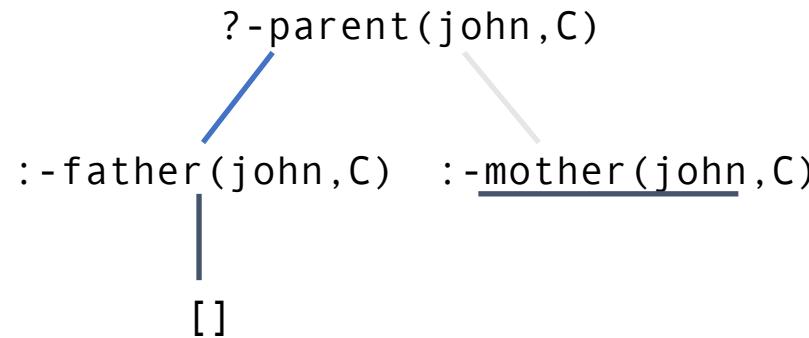
?- plist([X,Y,2|T]), L = [X,Y,2|T]

Examples ▾ History ▾ Solutions ▾ table results Run!

Cut

Pruning by means of cut

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



swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

File ▾ Edit ▾ Examples ▾ Help ▾

Search

169 users online

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!

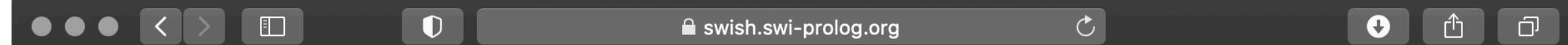
The screenshot shows the SWISH web interface for SWI-Prolog. At the top, there's a navigation bar with tabs for File, Edit, Examples, and Help. A search bar is also at the top. Below the header, there's a sidebar with a logo of an owl and a 'Program' tab. The main area contains a Prolog query window and a results window.

In the query window, the user has entered the following code:

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

Below the code, the user has run the query `parent(john,C)`. The results show that `C = paul`. There is a red circle around the "Next" button in the result table, which has options for 10, 100, 1,000, and Stop.

In the results window, the query `?- parent(john,C)` is shown again.



SWISH

[File](#) ▾[Edit](#) ▾[Examples](#) ▾[Help](#) ▾

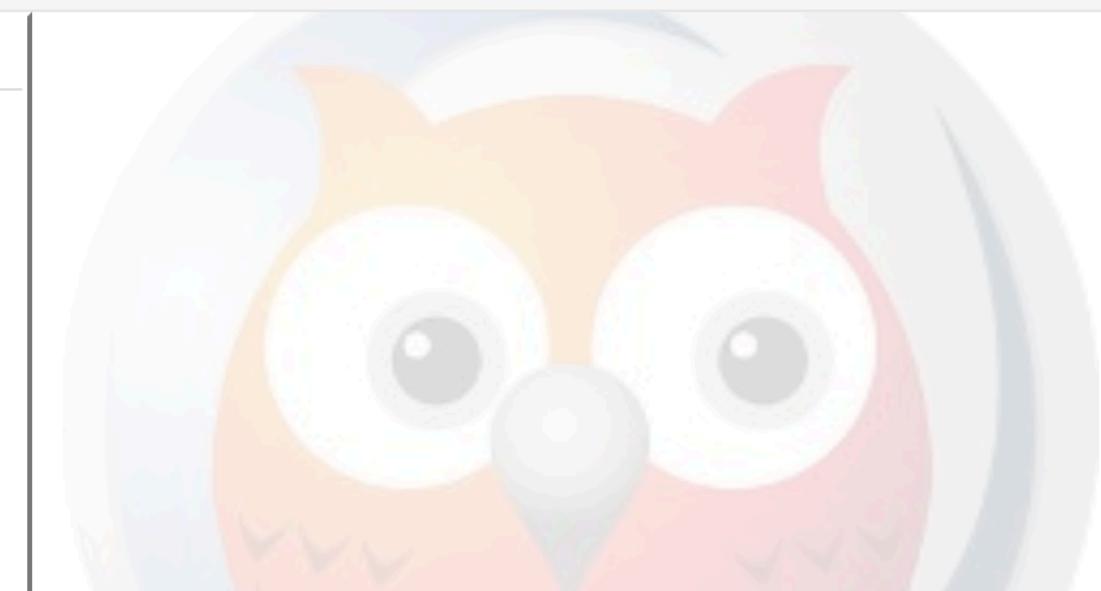
SWISH -- SWI-Prolog for SHaring

168 users online



Owl 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

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).
```

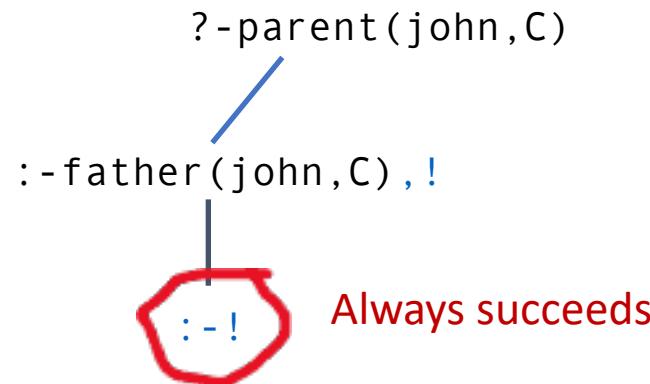
CUT

```
?-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).
```

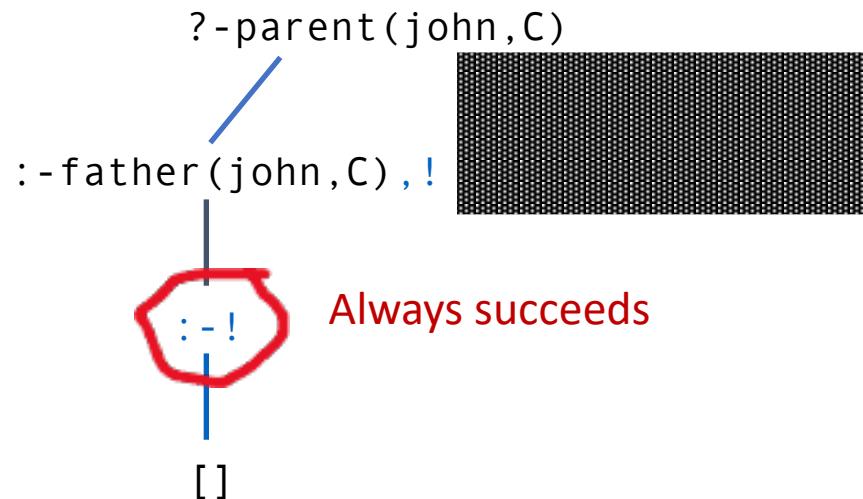
CUT



Pruning by means of cut

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

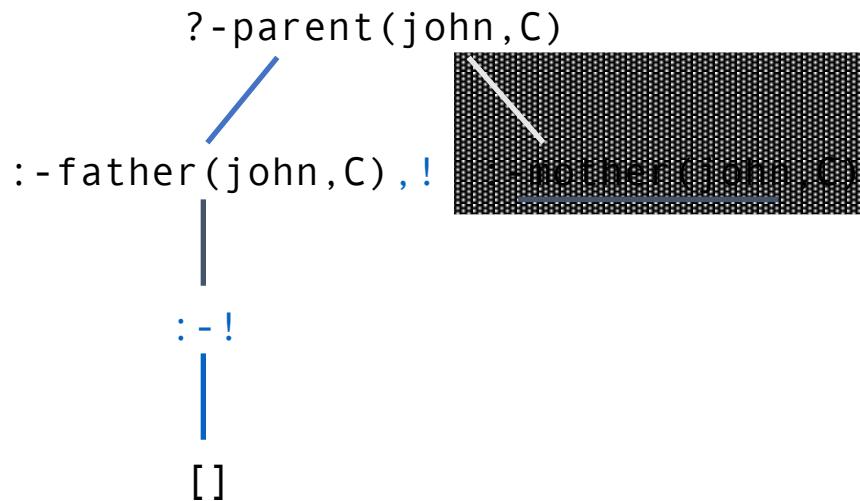
CUT



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 ▾



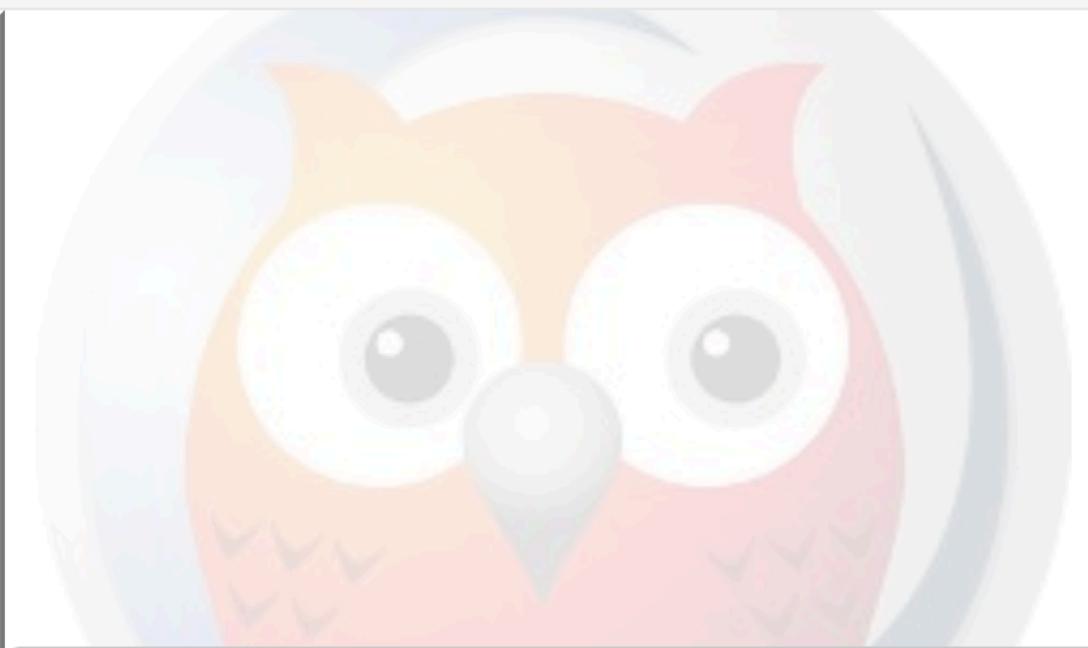
173 users online

Search



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

The screenshot shows a logic query interface with the following details:

- Query bar: `parent(john,C)`
- Variable binding: `C = paul`
- Result table controls: `Next` (highlighted with a red box), `10`, `100`, `1,000`, `Stop`
- Result table:
 - Header: `?- parent(john,C)`
 - Body: A single row showing `parent(john,C)` with `C = paul`.
- Bottom navigation: `Examples▲`, `History▲`, `Solutions▲`, `table results`, `Run!`

With cut

The screenshot shows a logic query interface with the following details:

- Query bar: `parent(john,C)`
- Variable binding: `C = paul`
- Result table:
 - Header: `?- parent(john,C)`
 - Body: A single row showing `parent(john,C)` with `C = paul`.
- Bottom navigation: `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).
```

? - p(X, Y)
 |
 : - q(X, Y)

The effect of cut

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

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

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

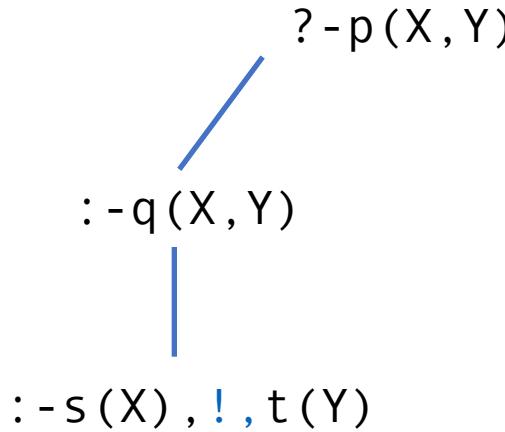
r(c, d) .

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) .

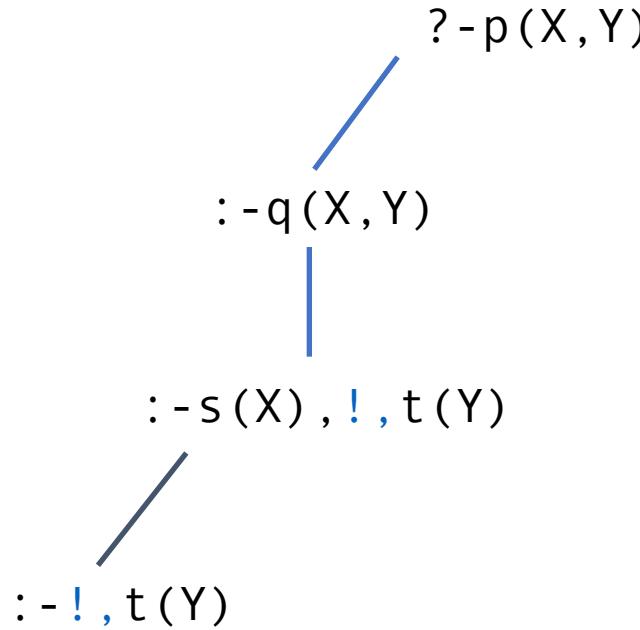
r(c, d) .

s(a) .

s(b) .

t(a) .

t(b) .



The effect of cut

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

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

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

$r(c, d).$

$s(a).$

$s(b).$

$t(a).$

$t(b).$

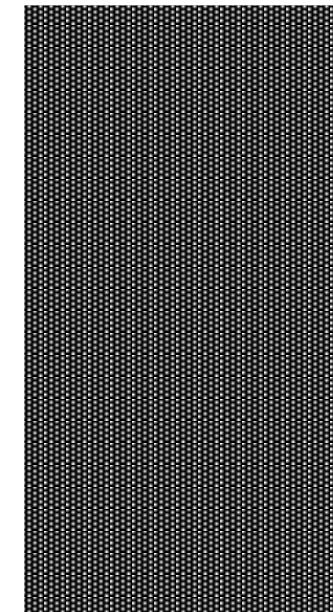
? - $p(X, Y)$

: - $q(X, Y)$

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

: - $!, t(Y)$

: - $t(Y)$



The effect of cut

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

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

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

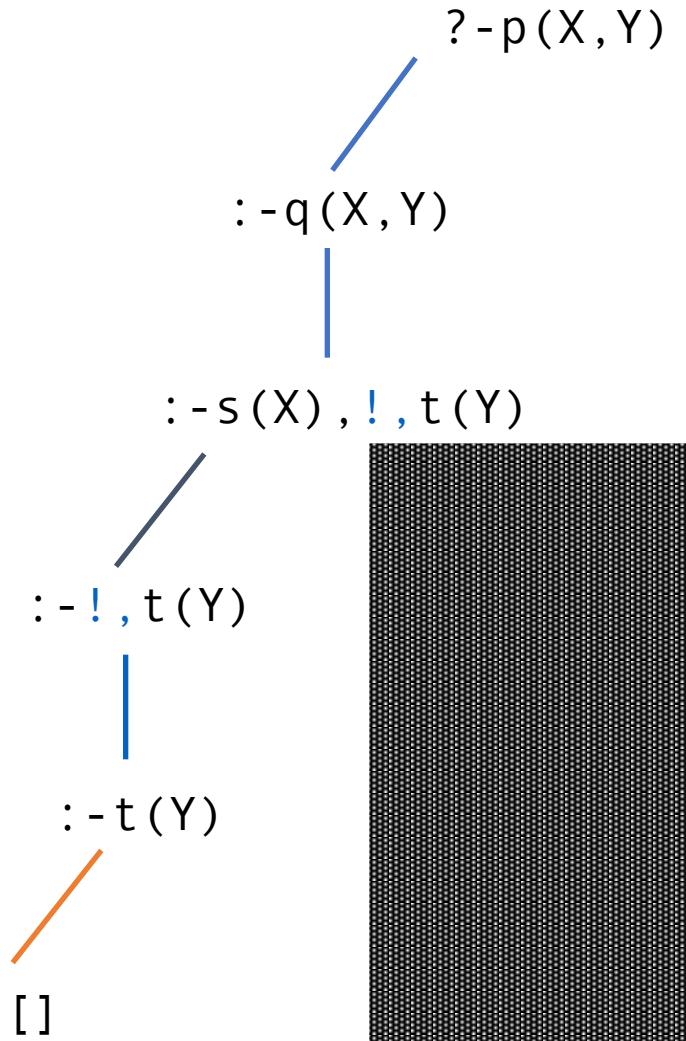
$r(c, d).$

$s(a).$

$s(b).$

$t(a).$

$t(b).$



The effect of cut

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

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

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

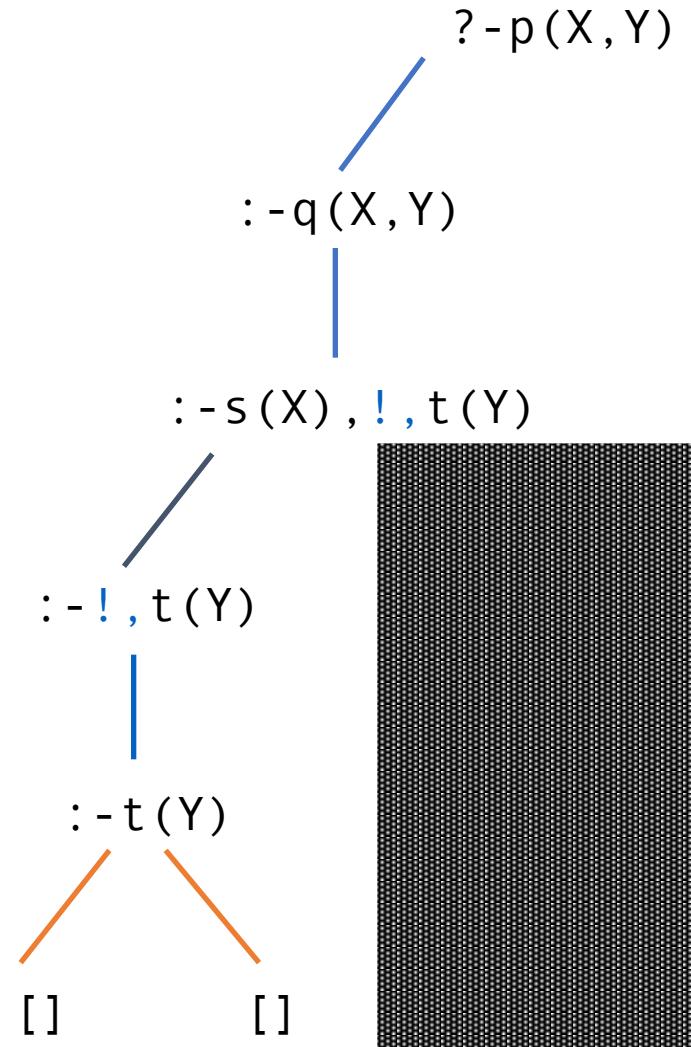
$r(c, d).$

$s(a).$

$s(b).$

$t(a).$

$t(b).$



The effect of cut

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

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

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

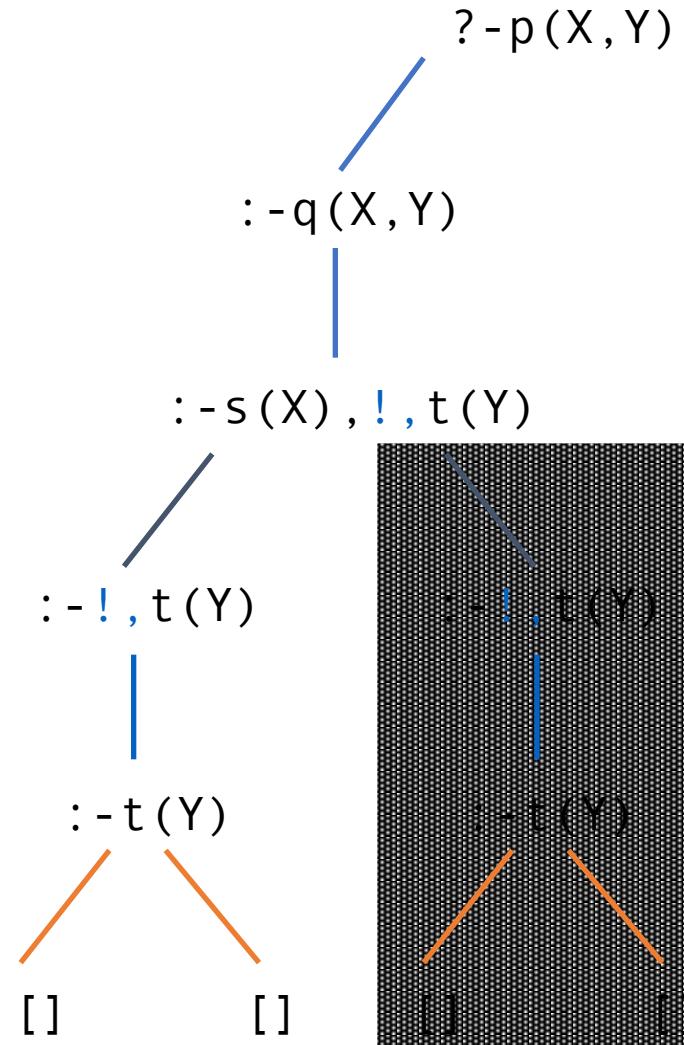
r(c, d) .

s(a) .

s(b) .

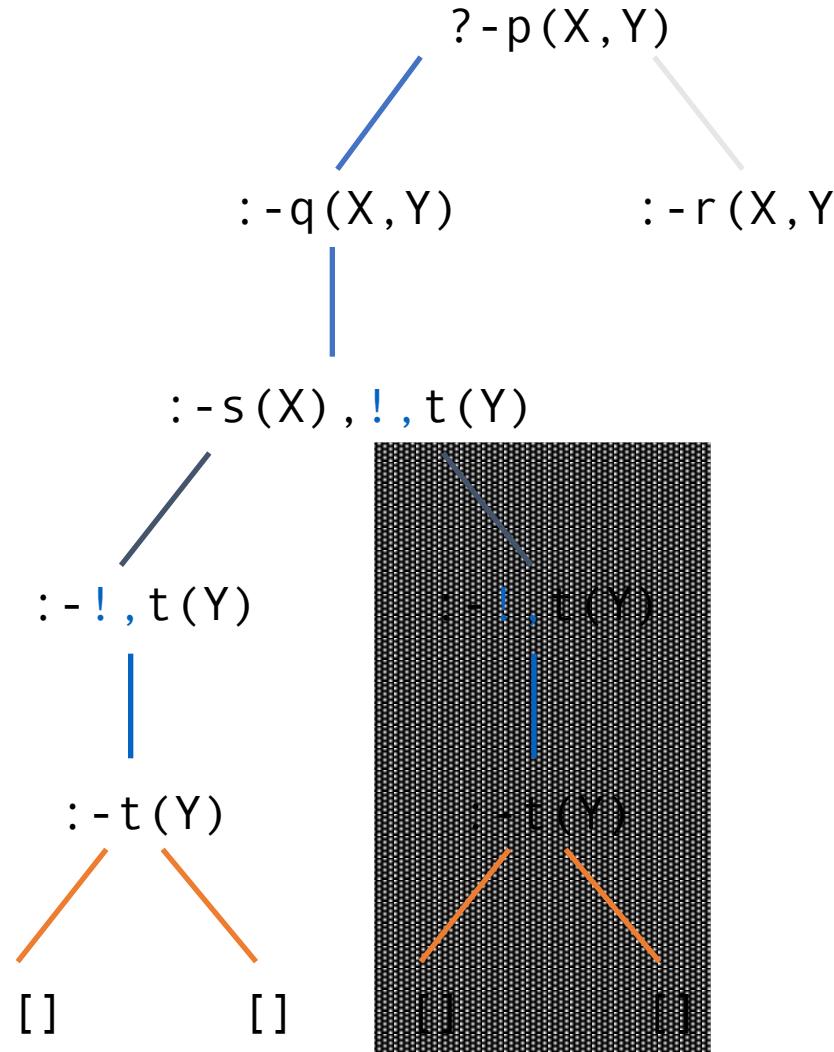
t(a) .

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

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

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

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

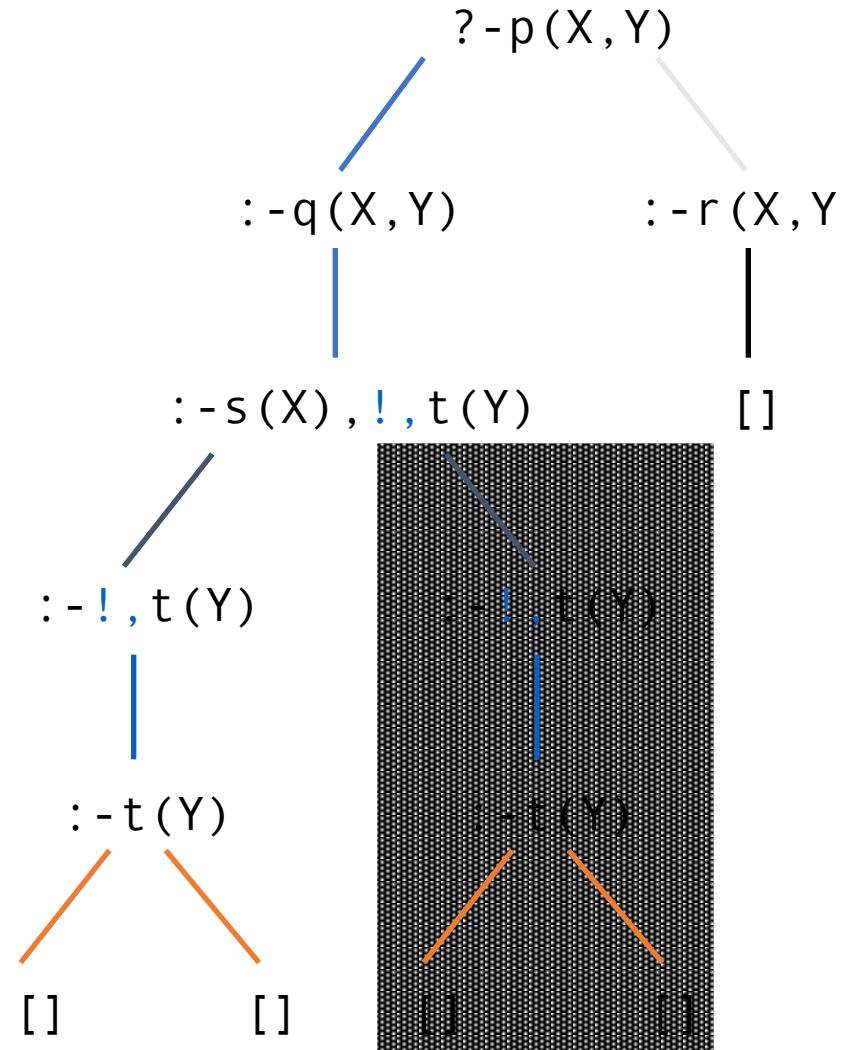
$r(c, d).$

$s(a).$

$s(b).$

$t(a).$

$t(b).$



swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

File ▾ Edit ▾ Examples ▾ Help ▾ 193 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)

Examples ▾ History ▾ Solutions ▾

table results Run!

swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

File ▾ Edit ▾ Examples ▾ Help ▾ 193 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)

Examples ▾ History ▾ Solutions ▾

table results Run!



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)

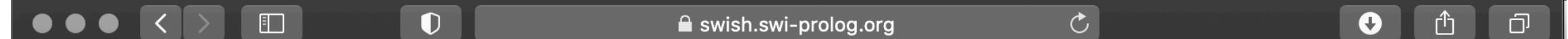
X = Y, Y = a

Next 10 100 1,000 Stop

?- **p**(**X**,**Y**)

Examples History Solutions

table results Run!



File ▾

Edit ▾

Examples ▾

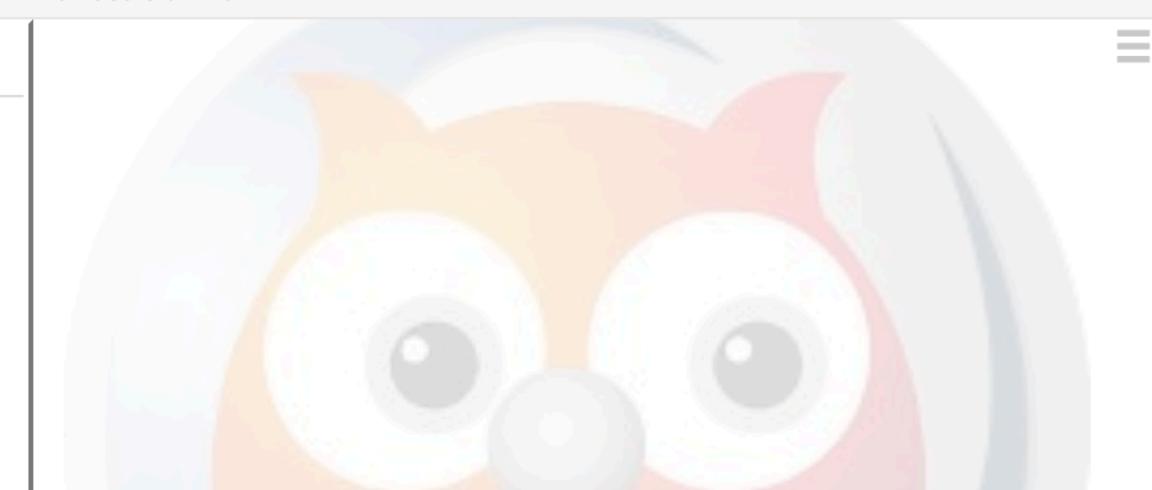
Help ▾



192 users online

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)



X = Y, Y = a

X = a,
Y = b

Next 10 100 1,000 Stop

?- p(x,y)

Examples ▾ History ▾ Solutions ▾

 table results Run!

swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

File ▾ Edit ▾ Examples ▾ Help ▾

Search

194 users online

Program +

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

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

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

r(c,d).

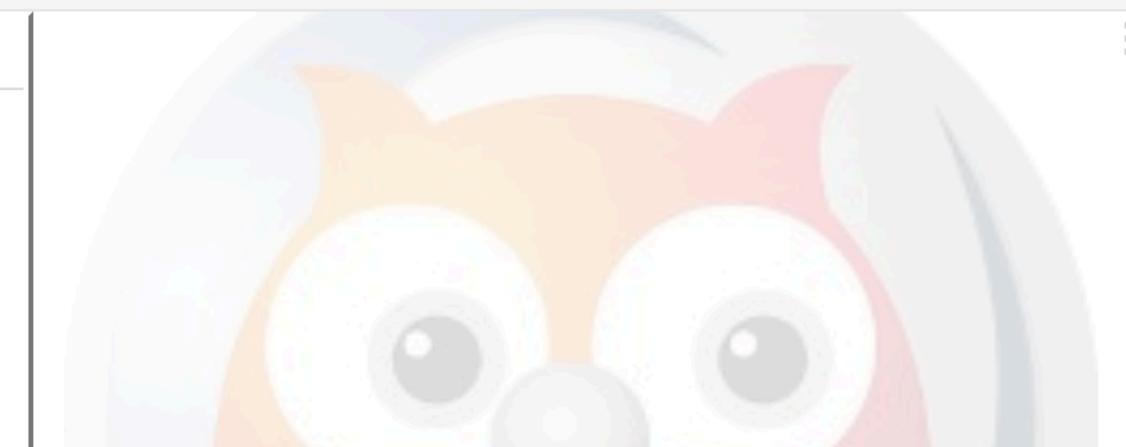
s(a).

s(b).

t(a).

t(b).

194 users online



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...

swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

195 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(b,X)

X = a

Next 10 100 1,000 Stop

?- p(b,X)

Examples▲ History▲ Solutions▲

table results Run!

swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

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
```

 194 users online

 p(b,X)

X = a
X = b

Next 10 100 1,000 Stop

?- p(b,X)

Examples ▾ History ▾ Solutions ▾

table results Run!

swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

File ▾ Edit ▾ Examples ▾ Help ▾ 195 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(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

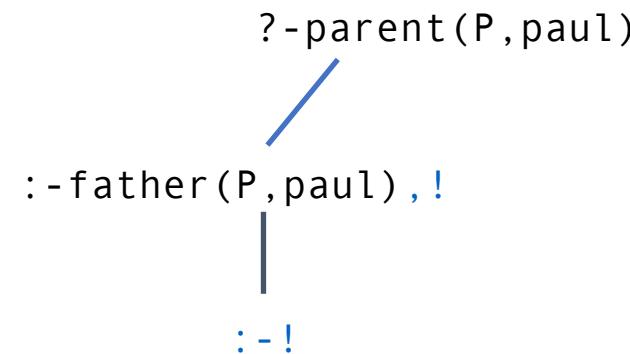
```
parent(X,Y) :- father(X,Y), !.  
parent(X,Y) :- mother(X,Y).  
father(john,paul).  
mother(mary,paul).
```

?-parent(P,paul)

:-father(P,paul), !

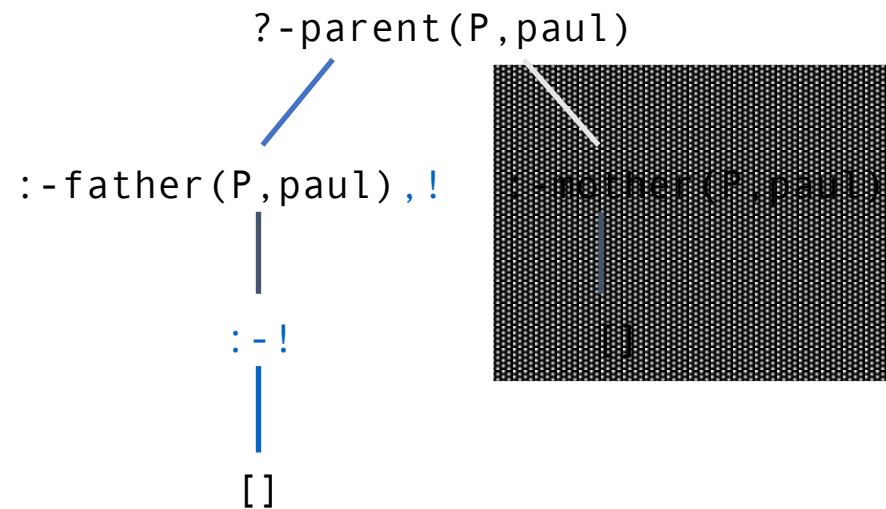
Pruning away success branches

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



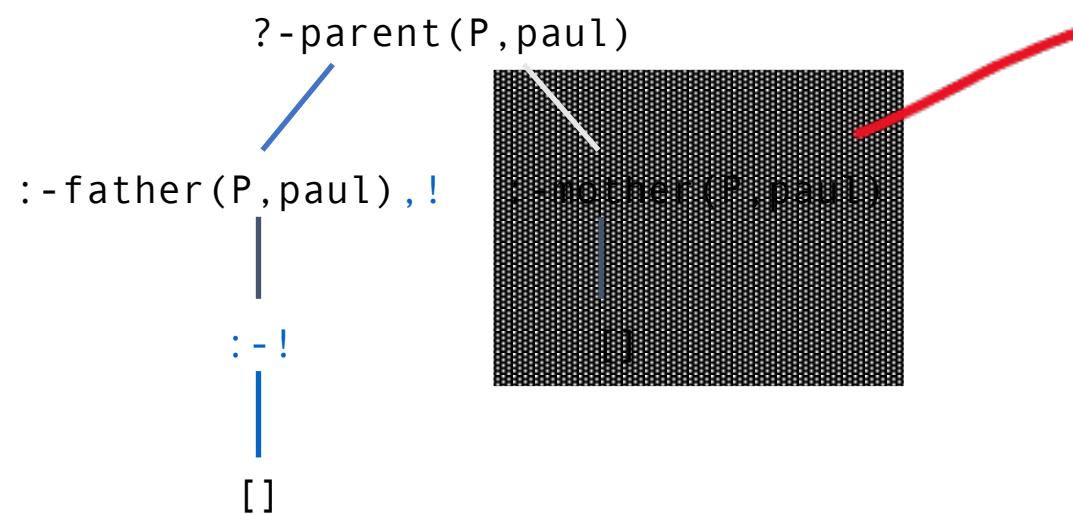
Pruning away success branches

```
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.

swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

File ▾ Edit ▾ Examples ▾ Help ▾ 186 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).
```

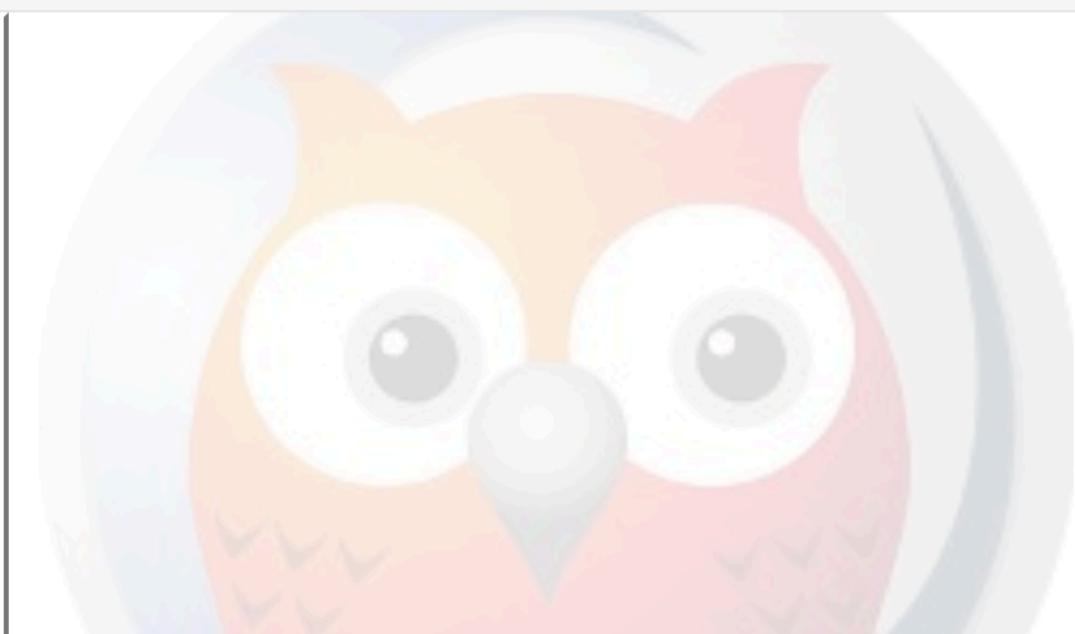
We get just one answer – for the father.

parent(X,paul)

X = john

?- parent(X,paul)

Examples ▾ History ▾ Solutions ▾ table results Run!



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

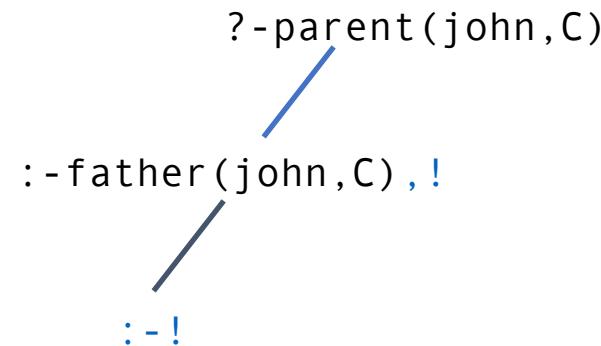
```
parent(X,Y) :- father(X,Y), !.  
parent(X,Y) :- mother(X,Y).  
father(john,paul).  
father(john,peter).  
mother(mary,paul).  
mother(mary,peter).
```

?-parent(john,C)

:-father(john,C), !

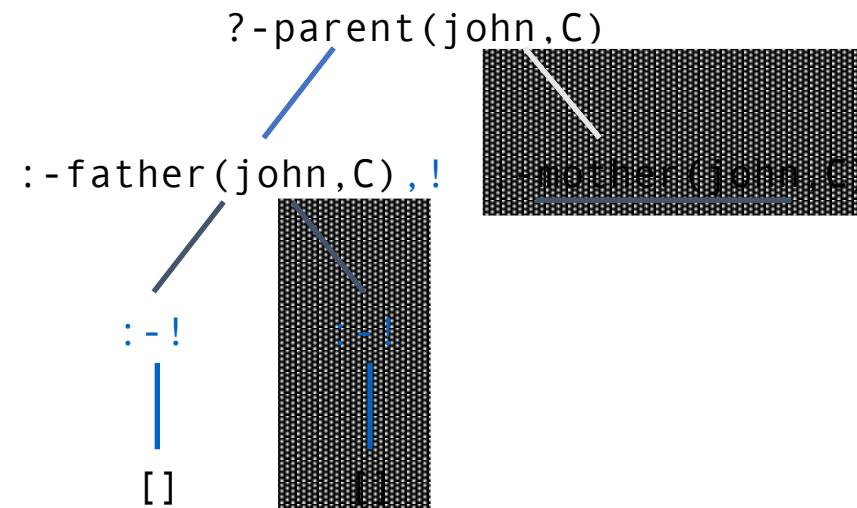
Pruning away success branches

```
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).
```



swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

File ▾ Edit ▾ Examples ▾ Help ▾ 194 users online

Search

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).
```

parent(john,X)

X = paul

?- parent(john,X)

And now we are not finding all children

Examples ▾ History ▾ Solutions ▾ table results Run!



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

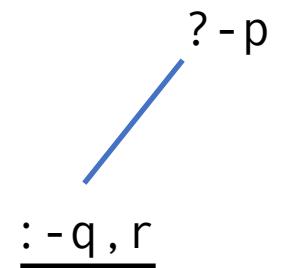
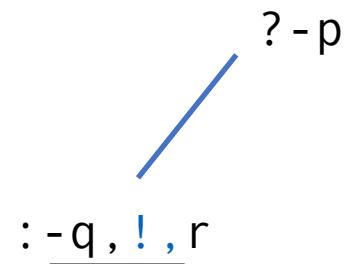
Cut

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

VS

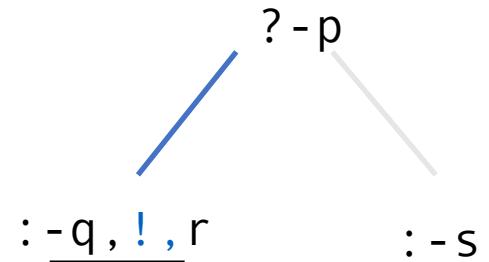
not

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



Cut

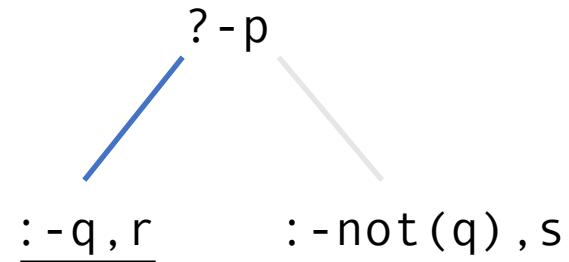
```
p:-q,! ,r.  
p:-s.  
s.
```



VS

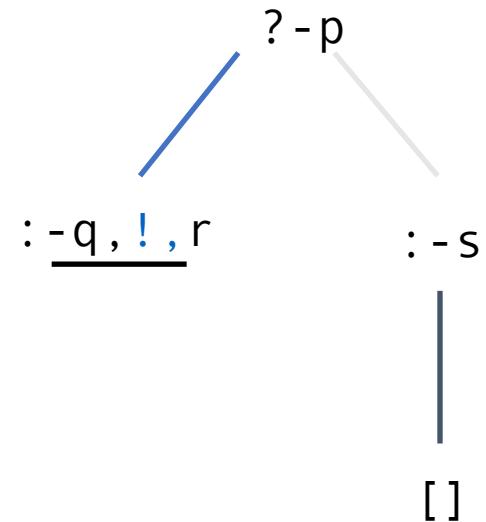
not

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



Cut

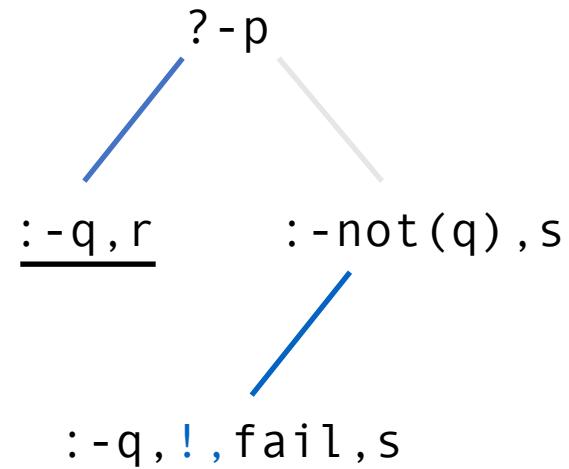
```
p:-q,! ,r.  
p:-s.  
s.
```



VS

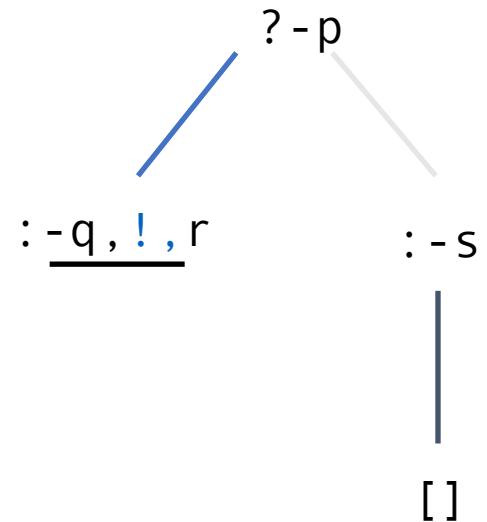
not

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



Cut

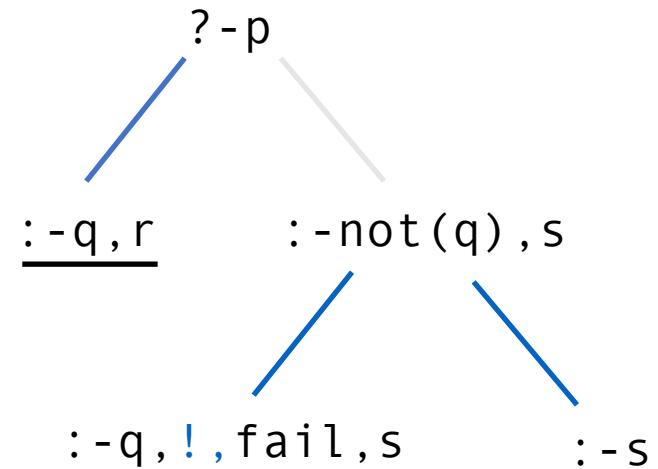
```
p:-q,! ,r.  
p:-s.  
s.
```



VS

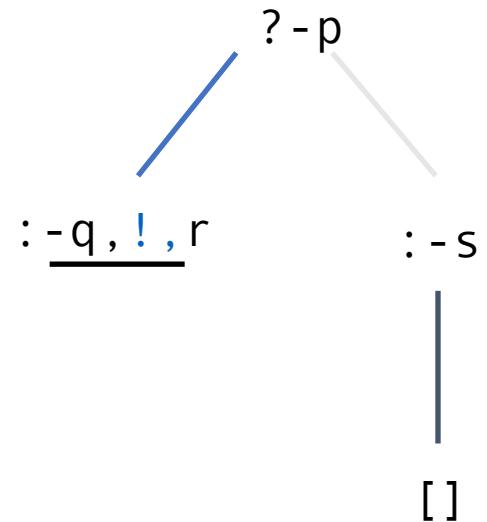
not

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



Cut

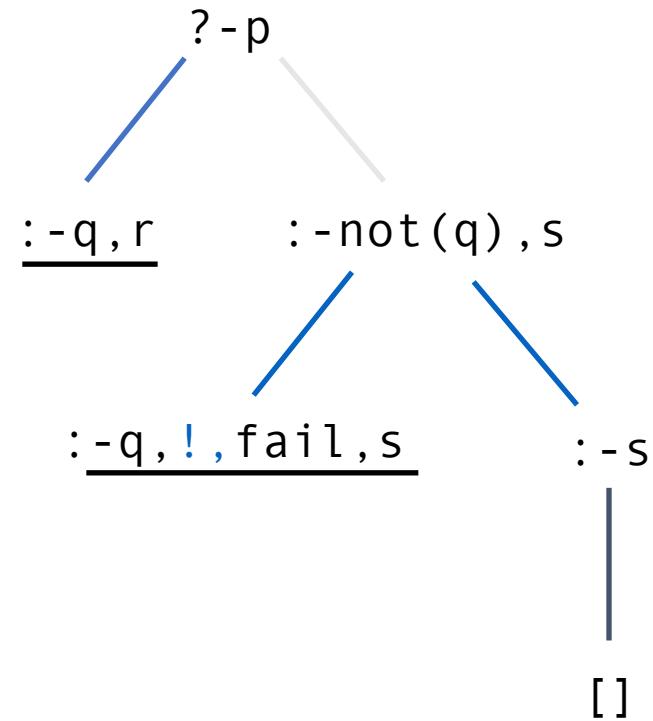
```
p:-q,! ,r.  
p:-s.  
s.
```



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

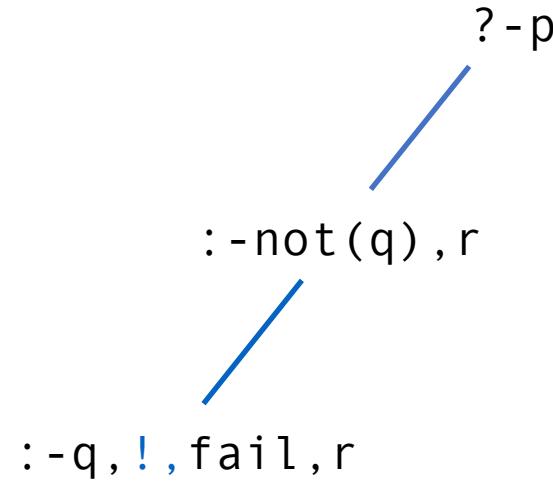
```
p:-not(q),r.  
p:-q.  
q.  
r.  
not(Goal):-Goal,! ,fail.  
not(Goal).
```

?-p

:-not(q),r

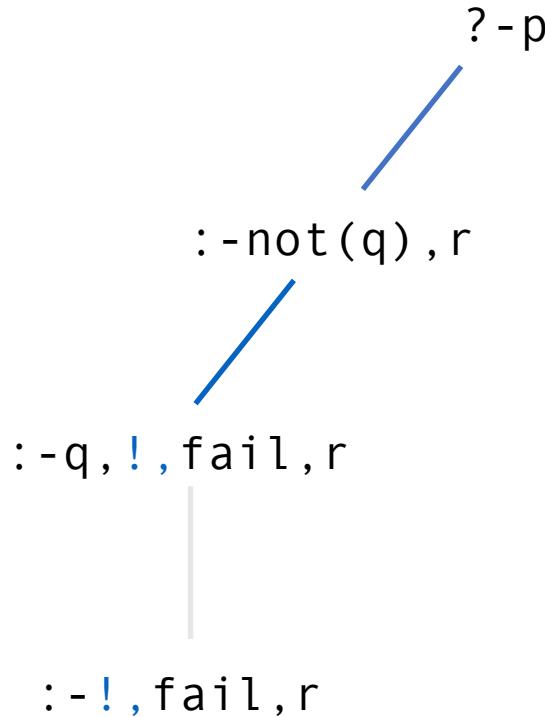
An example: How it works when `: - not (q)` fails

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



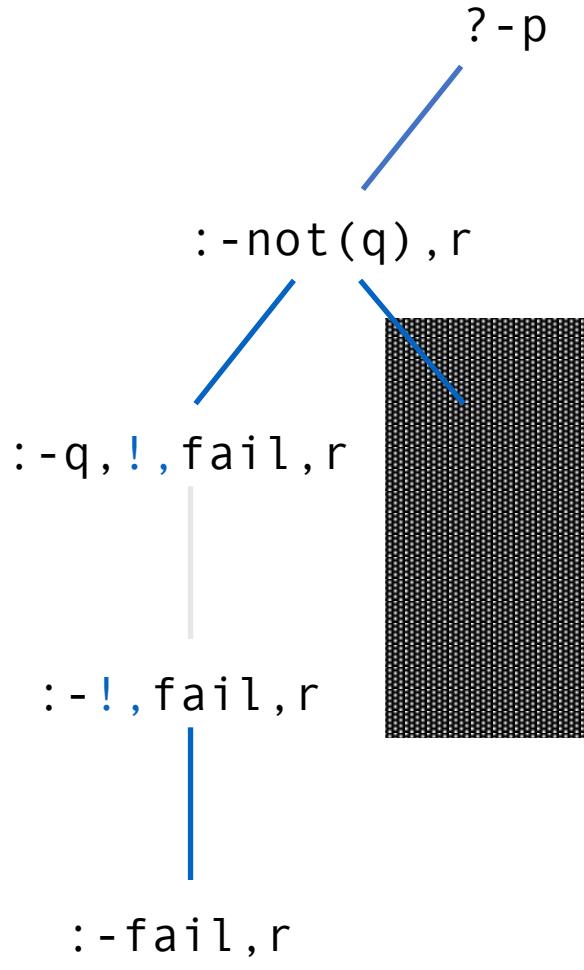
An example: How it works when `: - not (q)` fails

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



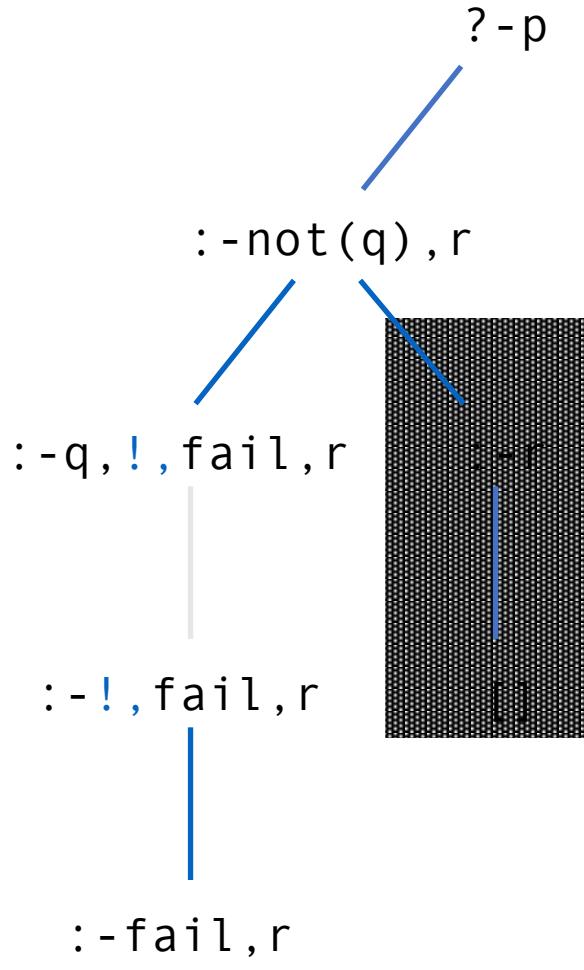
An example: How it works when `: - not (q)` fails

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



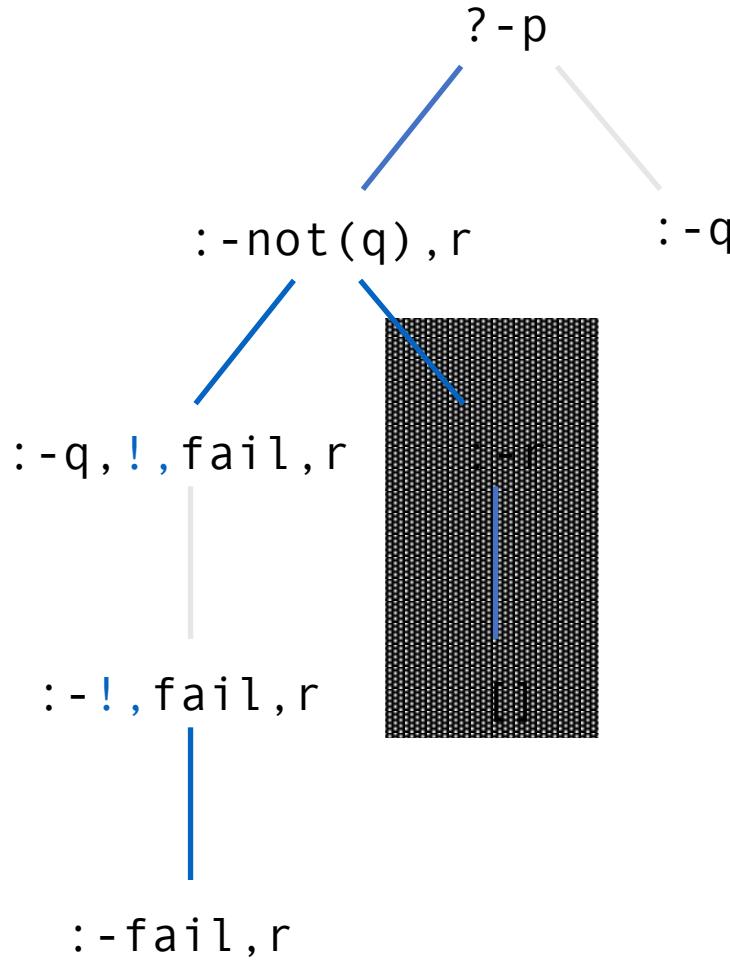
An example: How it works when `: - not (q)` fails

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



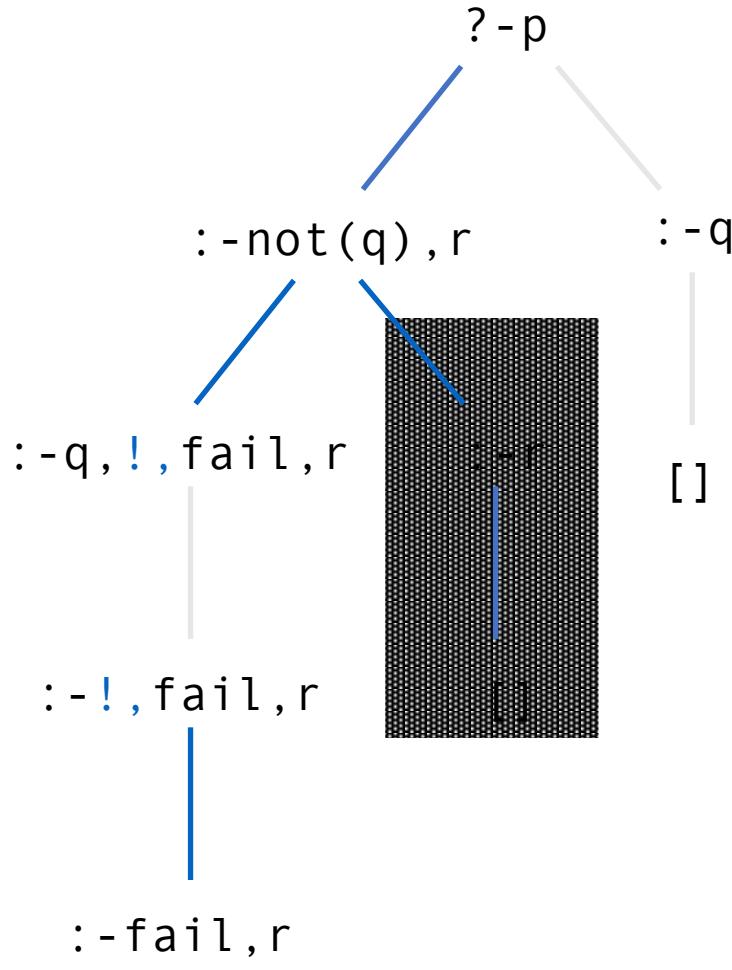
An example: How it works when `: - not(q)` fails

```
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)
```

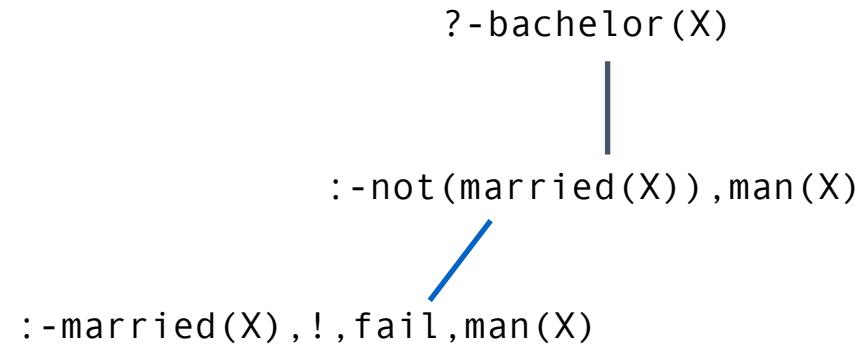
Prolog's not is unsound

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

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

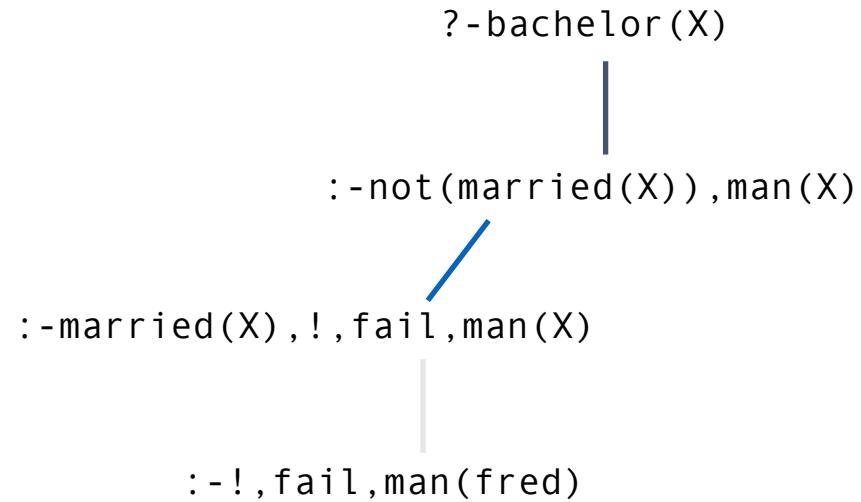
Prolog's not is unsound

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



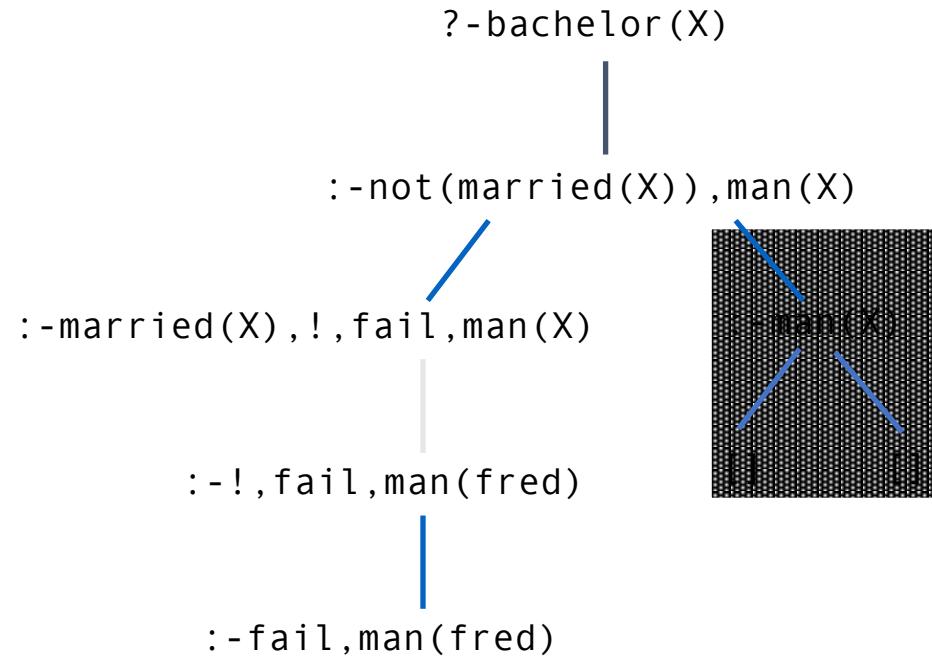
Prolog's `not` is unsound

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



Prolog's `not` is unsound

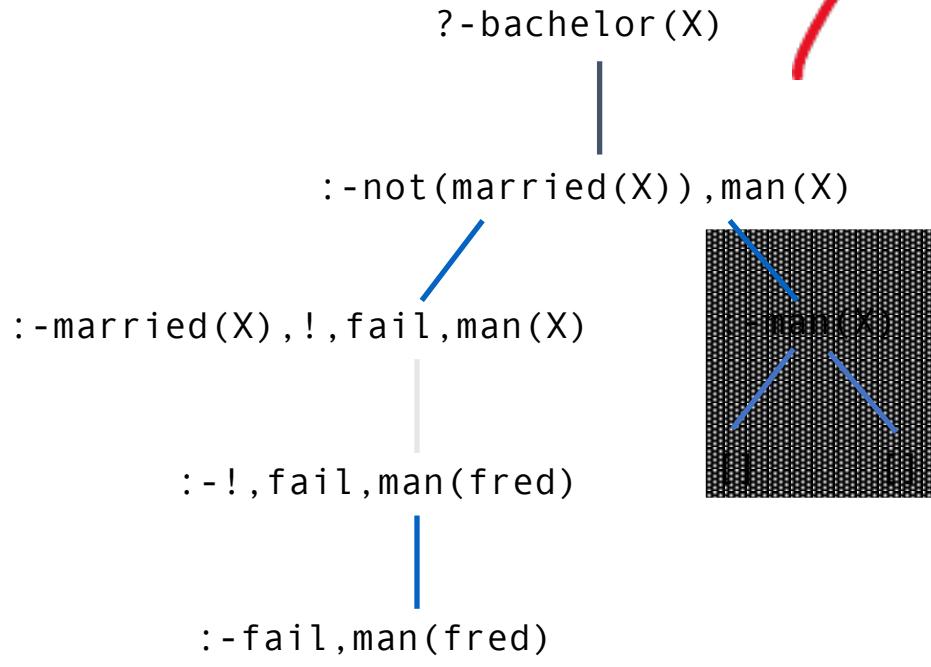
```
bachelor(X) :- not(married(X)), man(X).  
man(fred).  
man(peter).  
married(fred).
```



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!





File ▾

Edit ▾

Examples ▾

Help ▾



176 users online

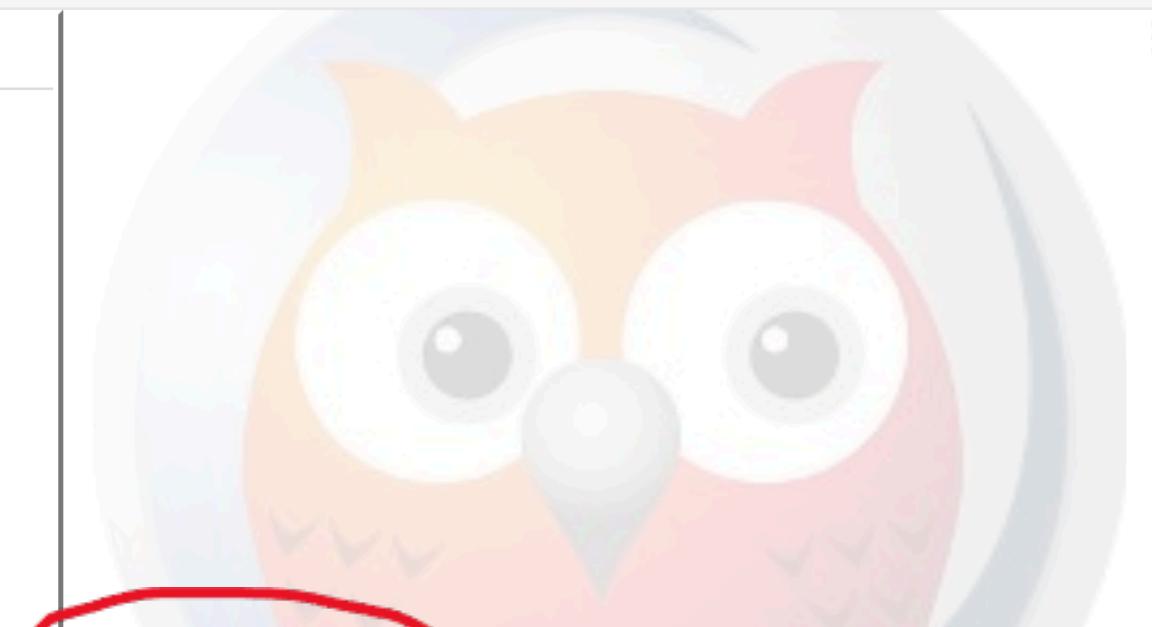
Search



Program



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



bachelor(X)

false

?- bachelor(X)

Examples ▾ History ▾ Solutions ▾

table results Run!

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).
```

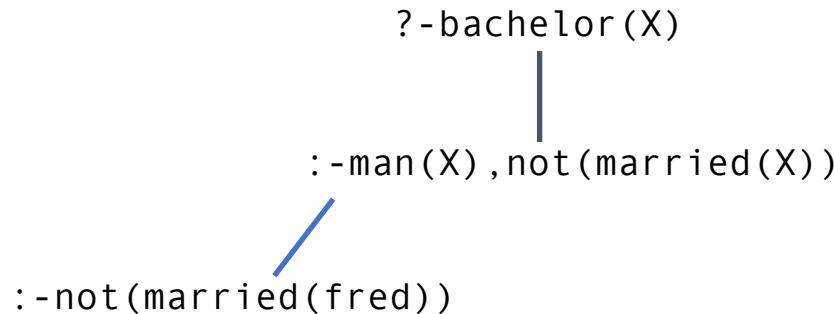
This will "ground" X.

```
?- 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).
```

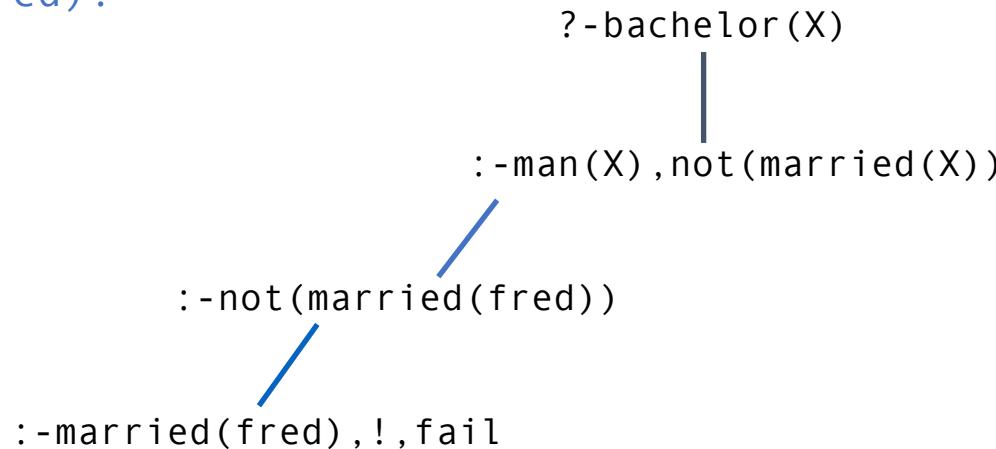
This will "ground" X.



Prolog's not is unsound – Avoiding the Problem

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

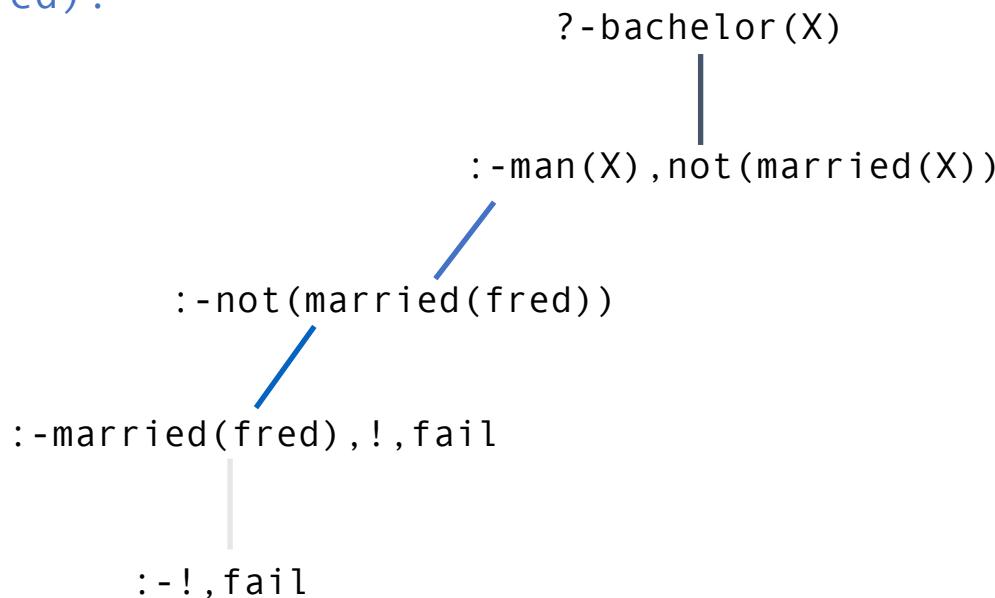
This will "ground" X.



Prolog's not is unsound – Avoiding the Problem

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

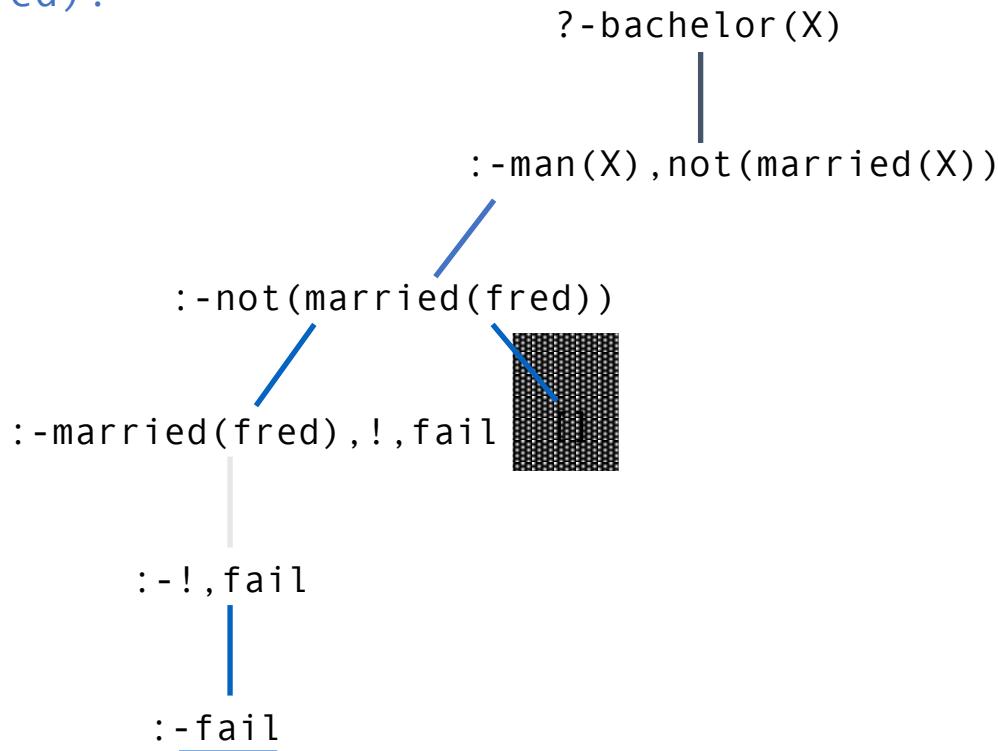
This will "ground" X.



Prolog's not is unsound – Avoiding the Problem

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

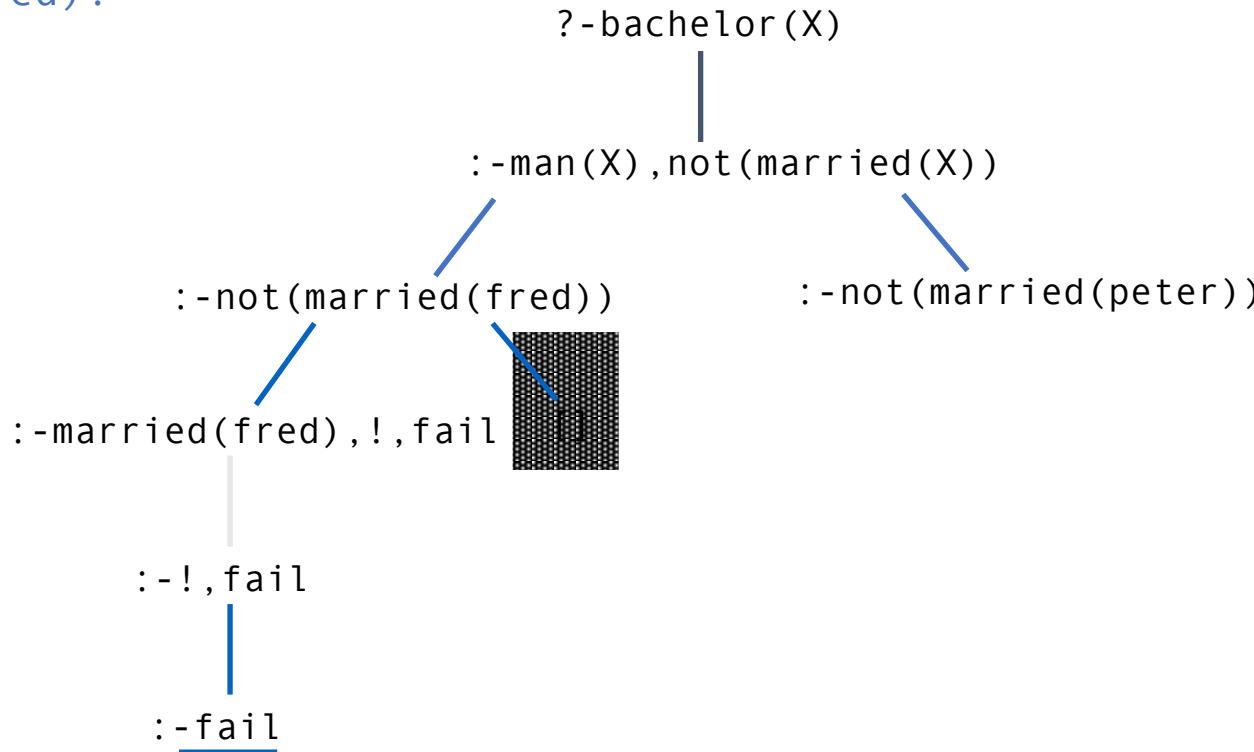
This will "ground" X.



Prolog's not is unsound – Avoiding the Problem

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

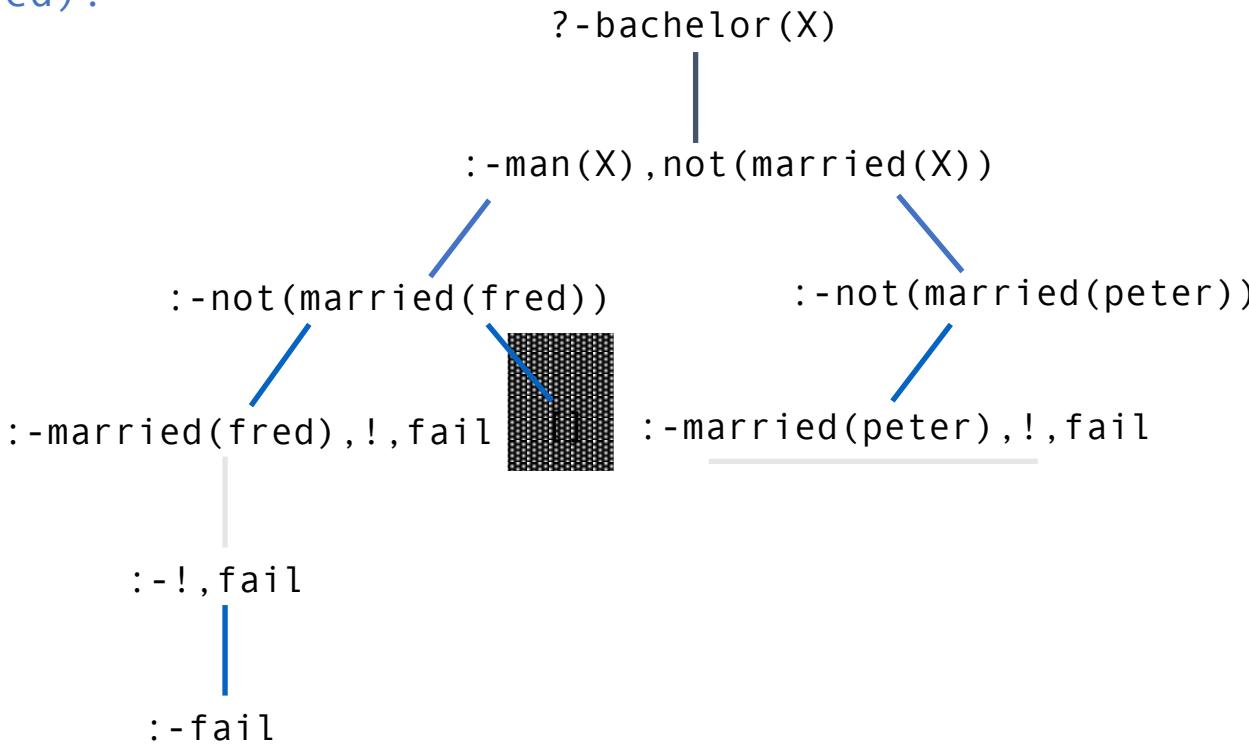
This will "ground" X.



Prolog's not is unsound – Avoiding the Problem

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

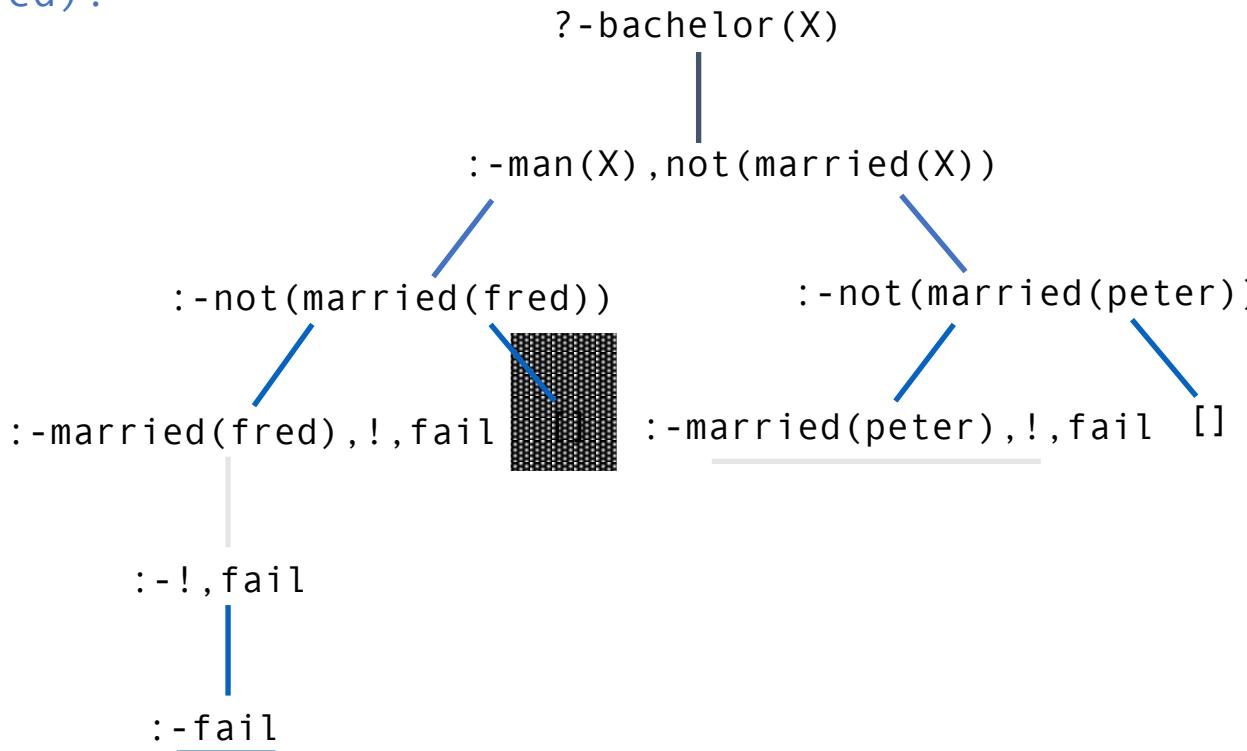
This will "ground" X.



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.





SWISH

File ▾

Edit ▾

Examples ▾

Help ▾

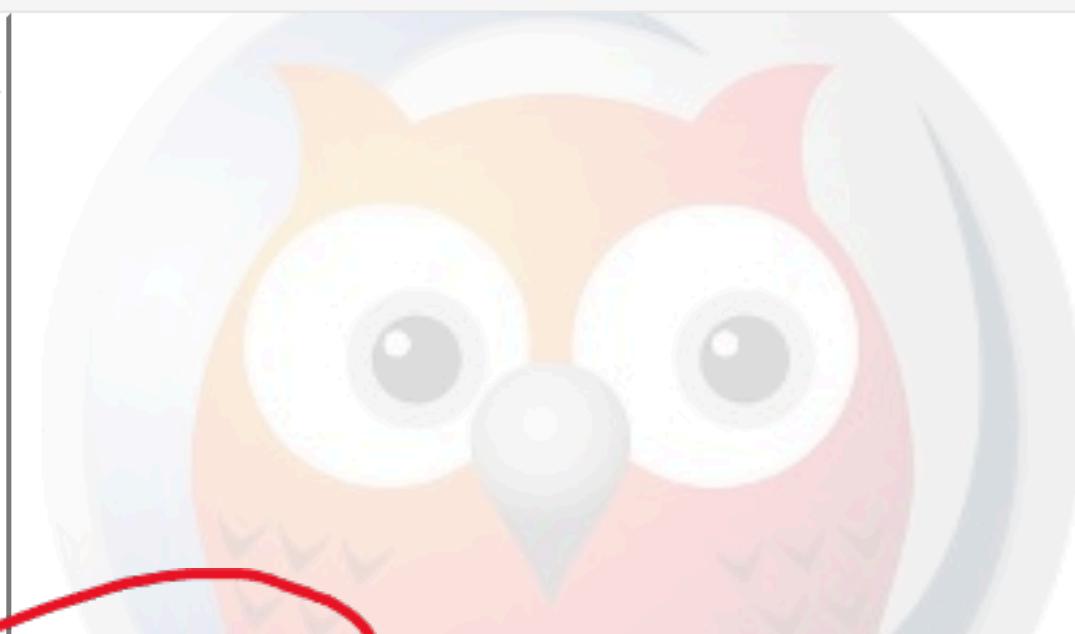


162 users online

Search

Program X

```
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!

Arithmetc in Prolog

Prolog arithmetic vs.unification

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

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

?- 9 **is** 5+7-3.
Yes

?- 9 = 5+7-3.
No

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

?- 9 = X+7-3.
No

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

?- 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.
```

swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

SWISH

File ▾ Edit ▾ Examples ▾ Help ▾ 256 users online

Search

Program 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 ▾ Solution ▾ Table results Run!