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 AΘ = BΘ.

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 AΘ = BΘ.
- 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 *Θ' = Θ σ.*

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 *Θ' = Θ σ*. 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)`

Θ = *{*X/`alice`, Y/Z*}*

Θ' = *{*X/`alice`, Y/`bob` , Z/`bob`*}*

Θ' = Θ σ, where *σ* = *{*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 unfifier *Θ* such that there is no strictly more general unifier of A and B.
- **Example:**

A = `studies(X, Y)`

B = `studies(alice, Z)`

Θ = {X/`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:** `Θ = {}, 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, Θ = {}` 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" */*

Θ = **{}**

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`

Θ = **{}**

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`

Θ = {}

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`

`Θ = {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`

`Θ = {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`

`Θ = {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)`

`Θ = {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

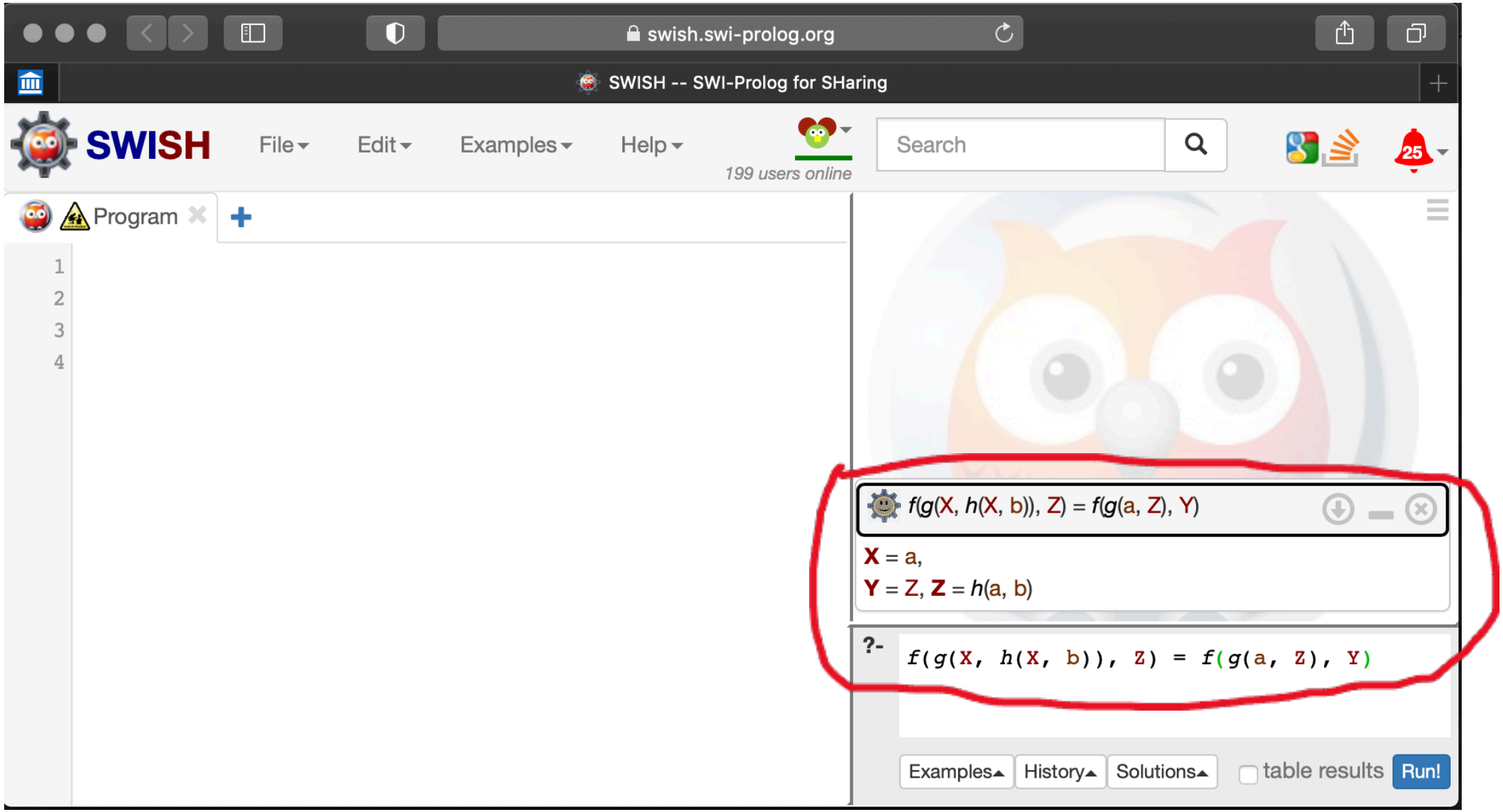
Θ = {X/a, Z/h(a,b), Y/h(a,b)}

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)
```

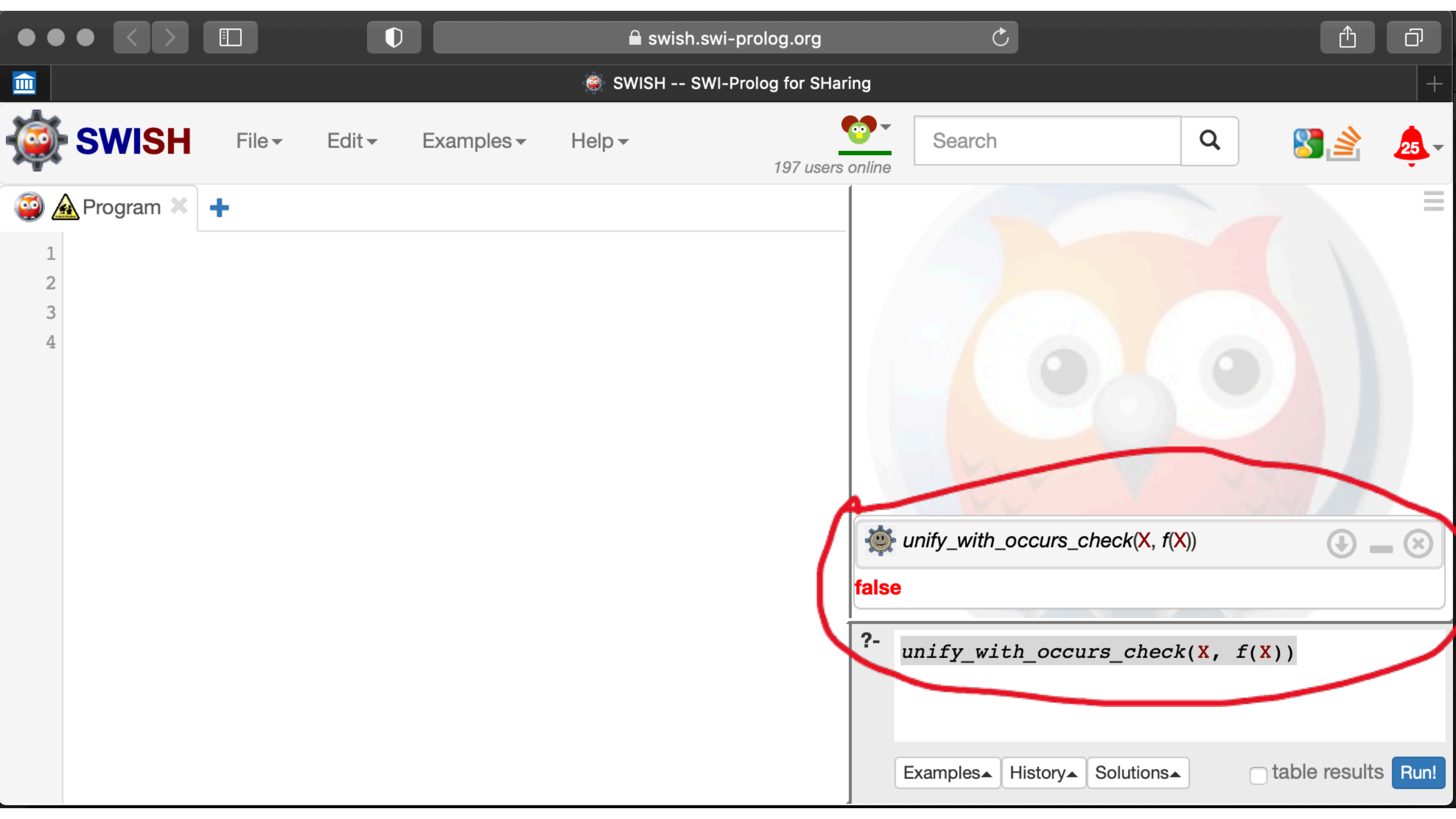


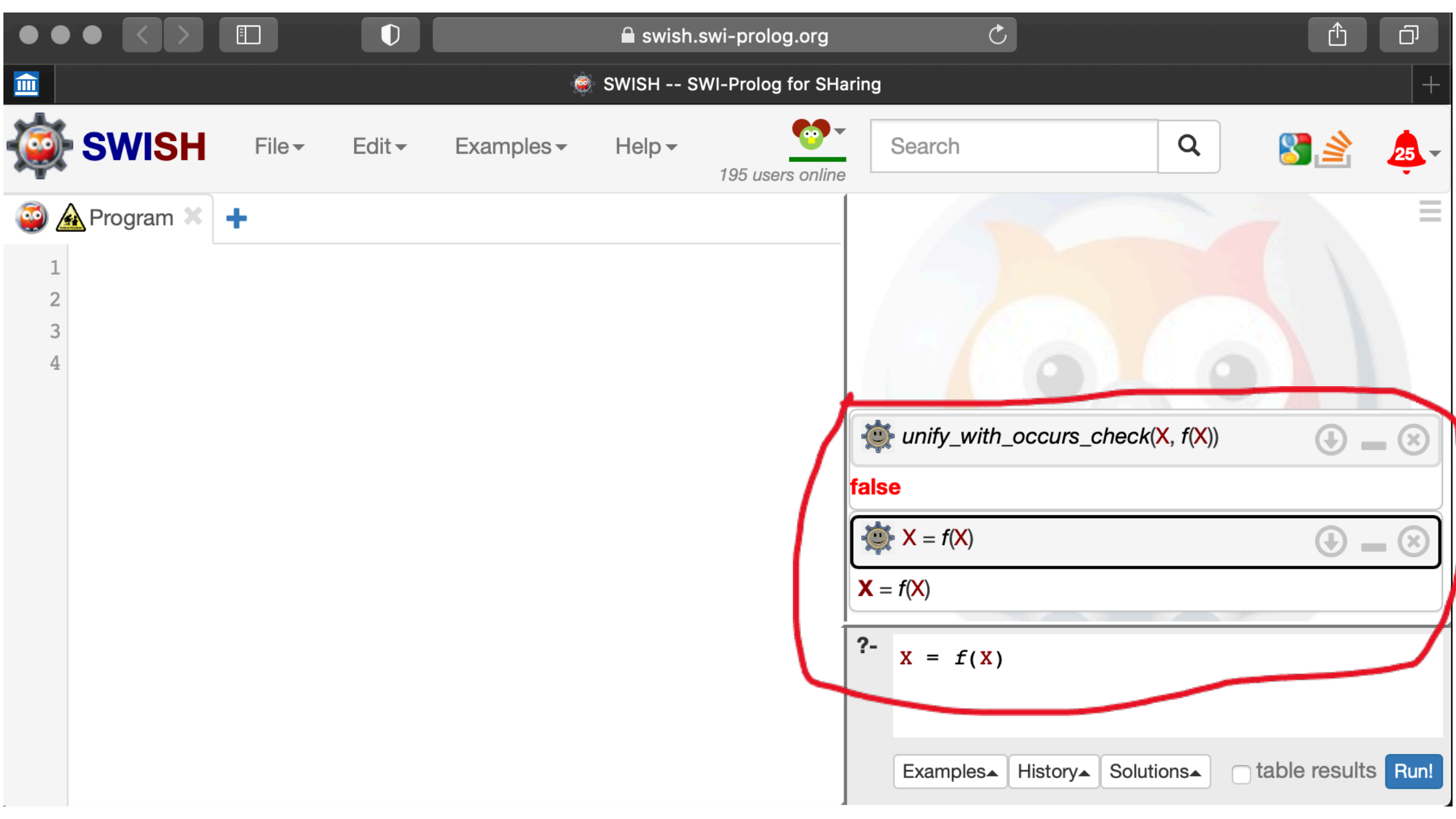
The “Occurs Check”

- 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
SWIS -- SWI-Prolog for SHaring
SWISH
File
Edit
Examples
Help
197 users online
Search
25
Program
1
2
3
4
unify_with_occurs_check(X, f(X))
false
?-
unify_with_occurs_check(X, f(X))
Examples
History
Solutions
table results
Run!


swish.swi-prolog.org
SWISH -- SWI-Prolog for SHaring
SWISH
File
Edit
Examples
Help
195 users online
Search
25
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!

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
```

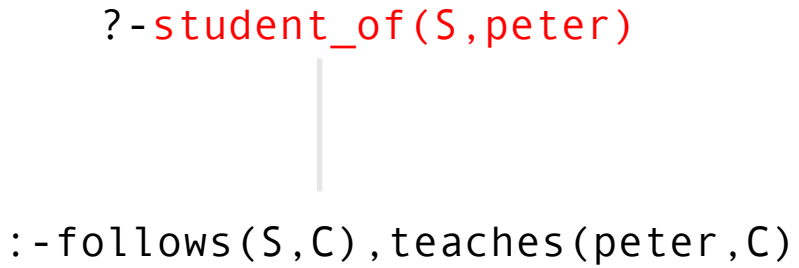
SLD-trees

```
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)
```

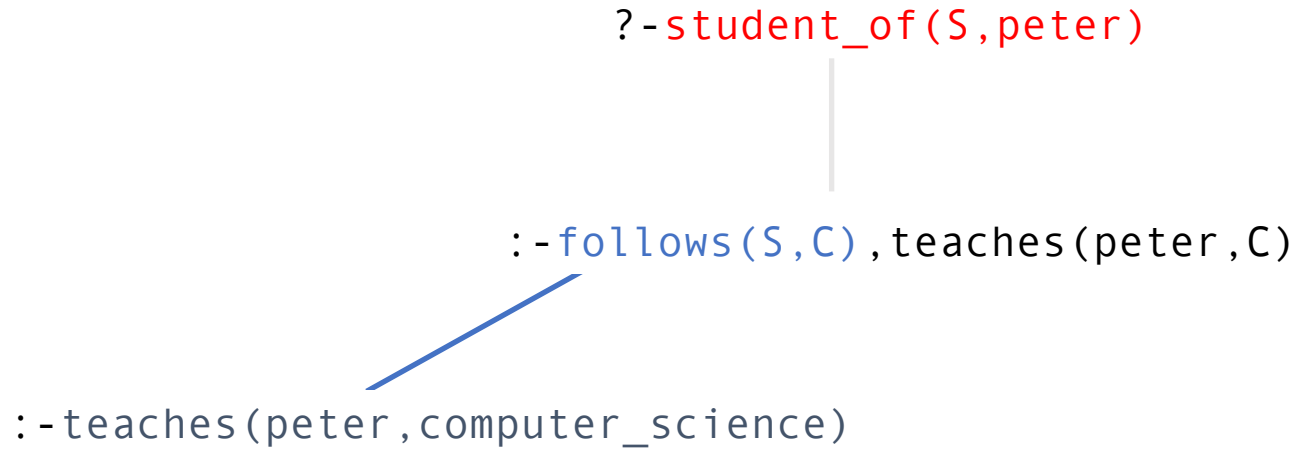
SLD-trees

```
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).
```



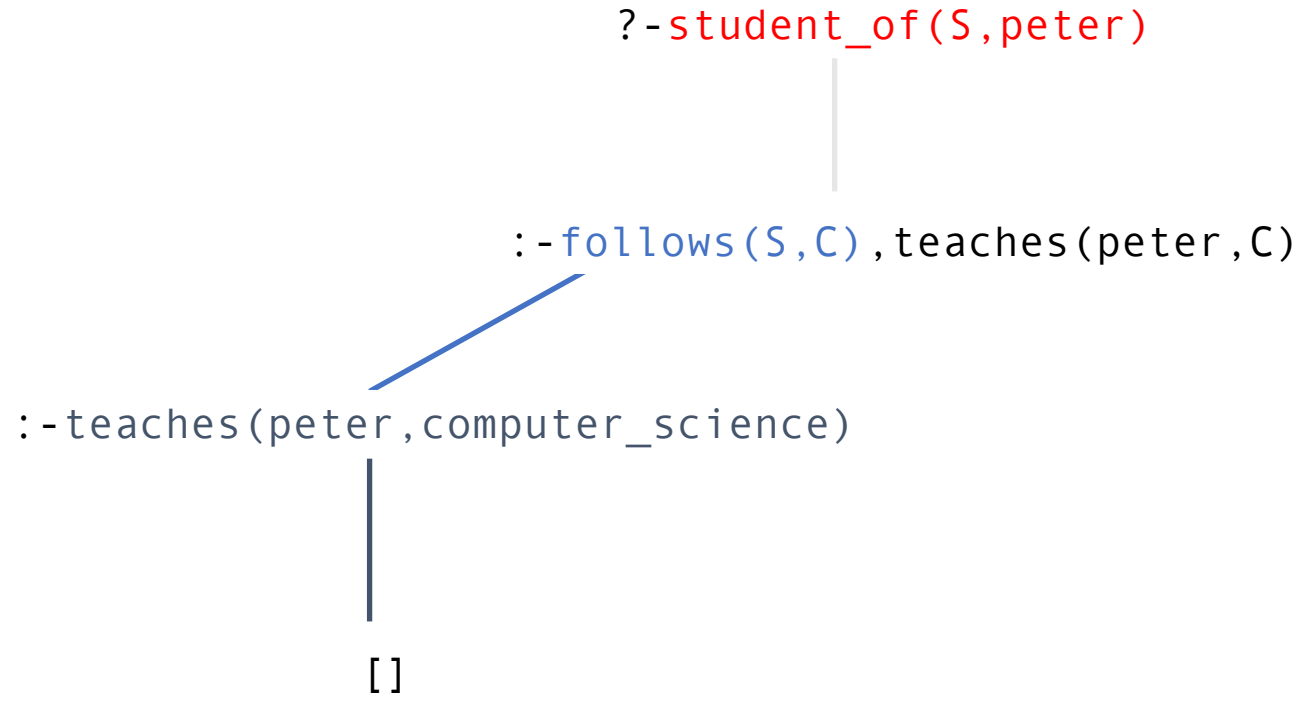
SLD-trees

```
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).
```



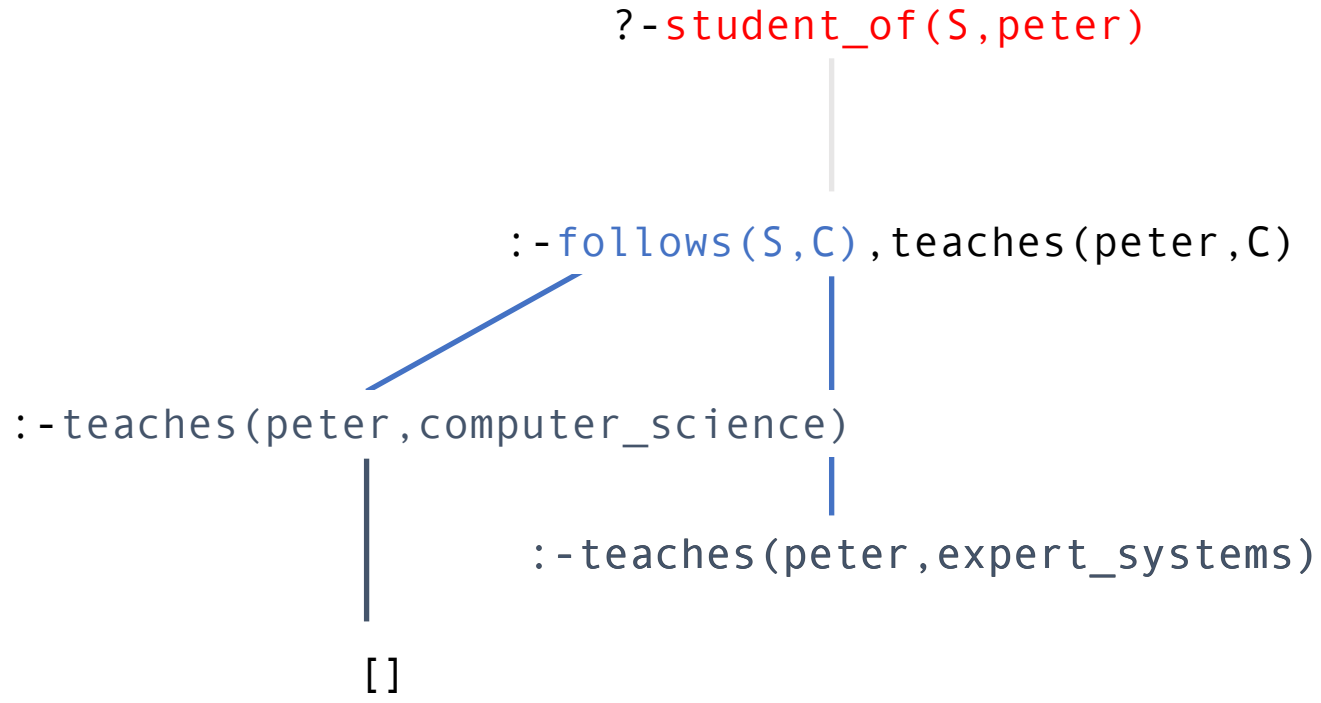
SLD-trees

```
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).
```



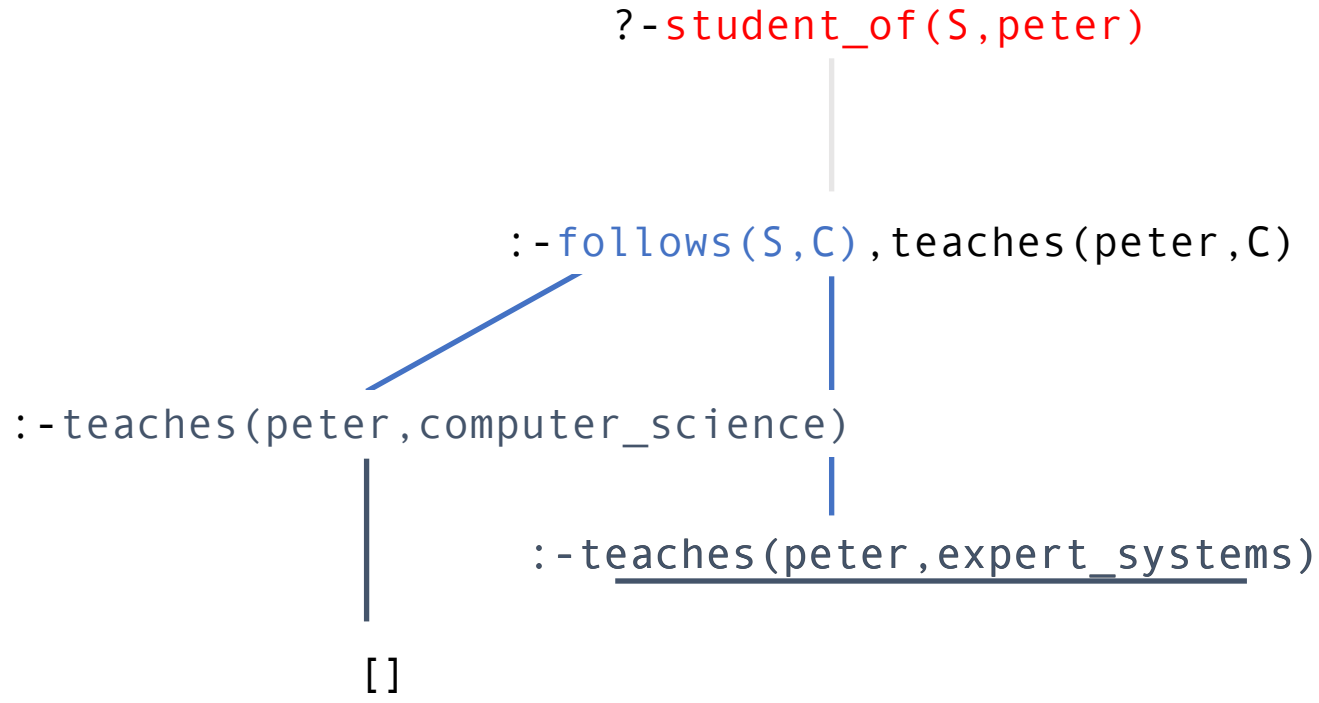
SLD-trees

```
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).
```



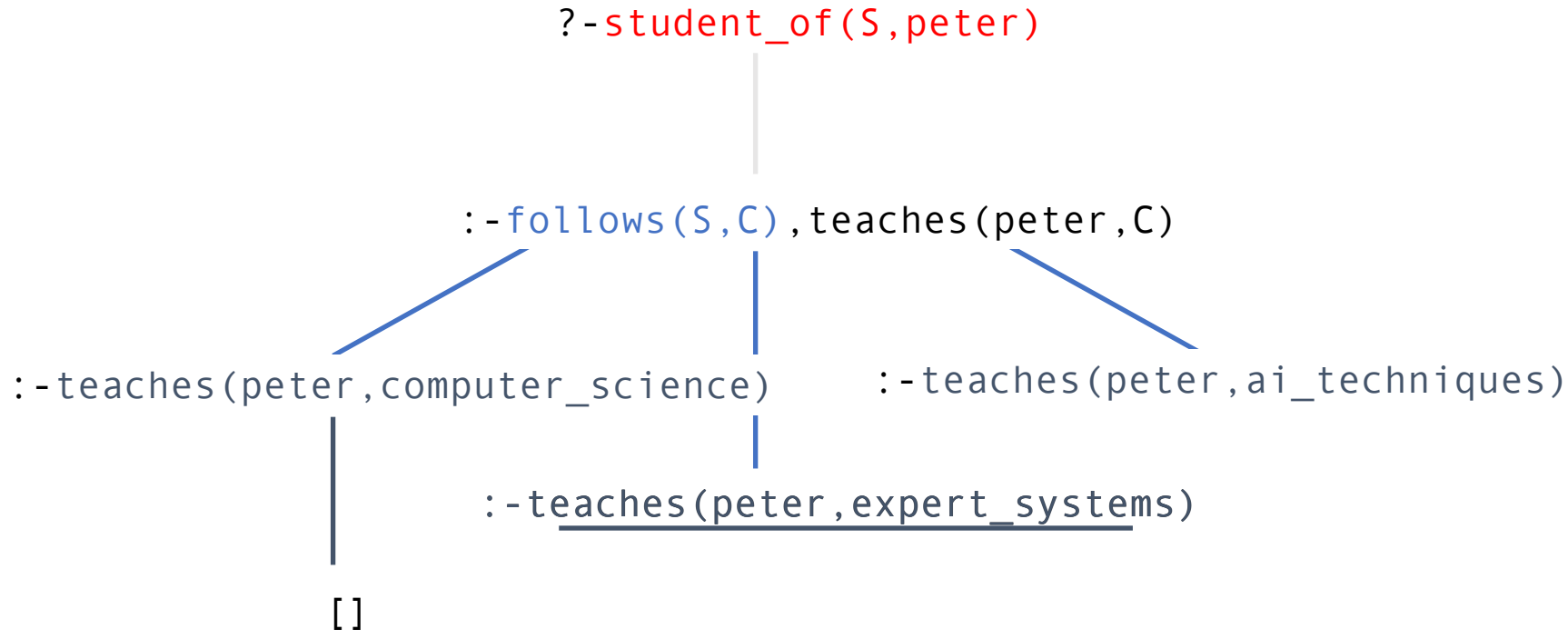
SLD-trees

```
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).
```



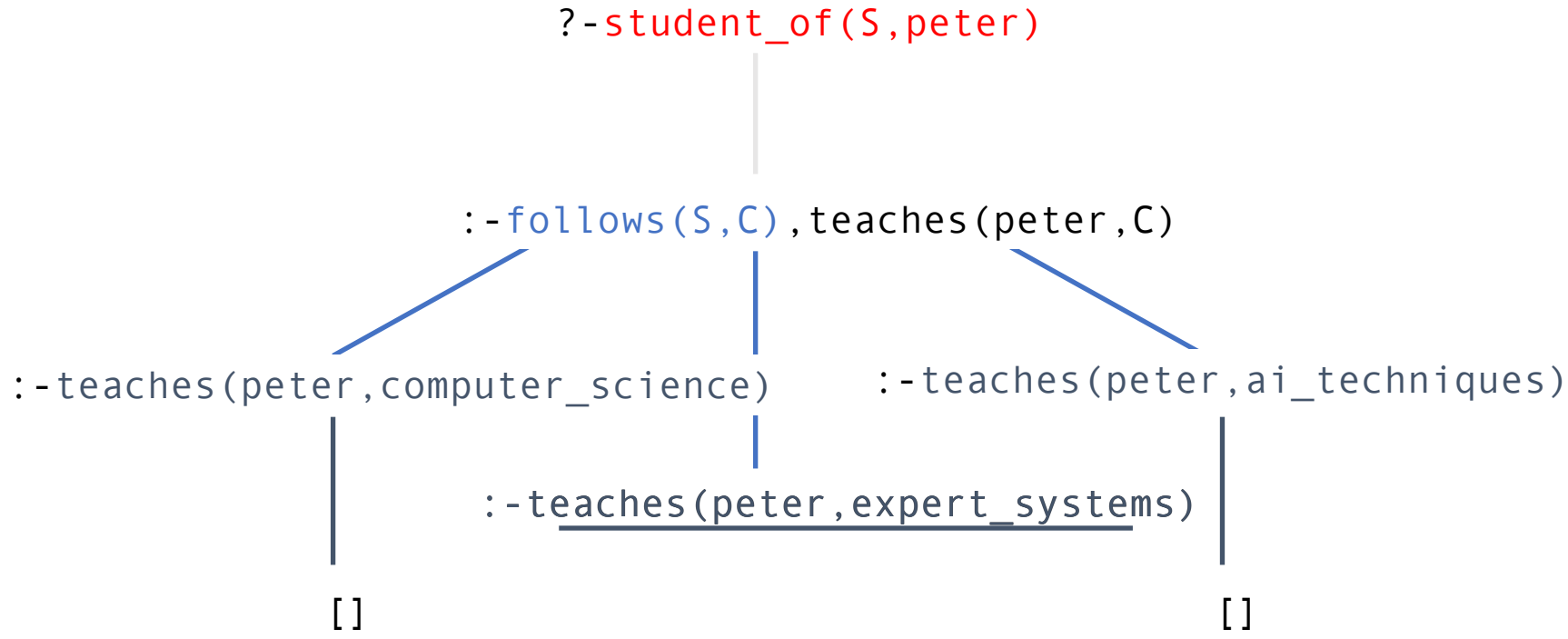
SLD-trees

```
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).
```

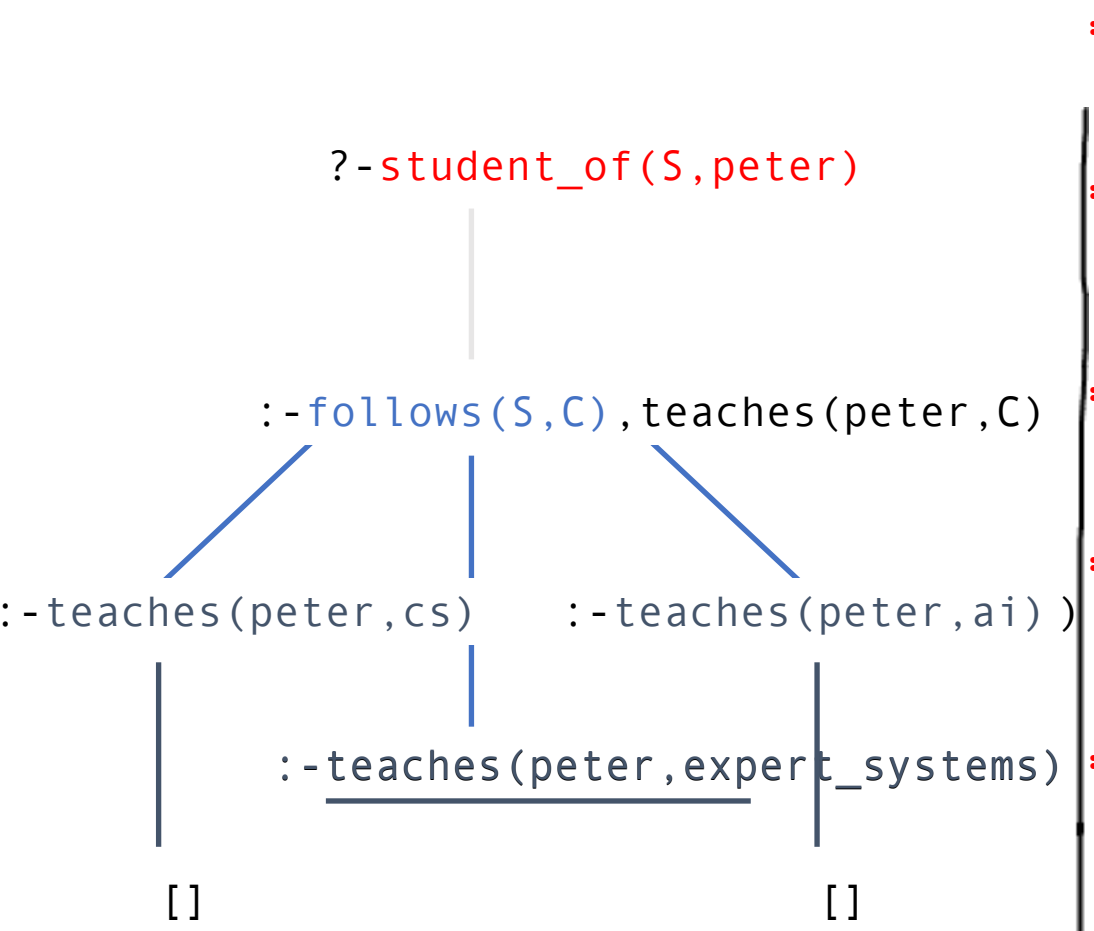


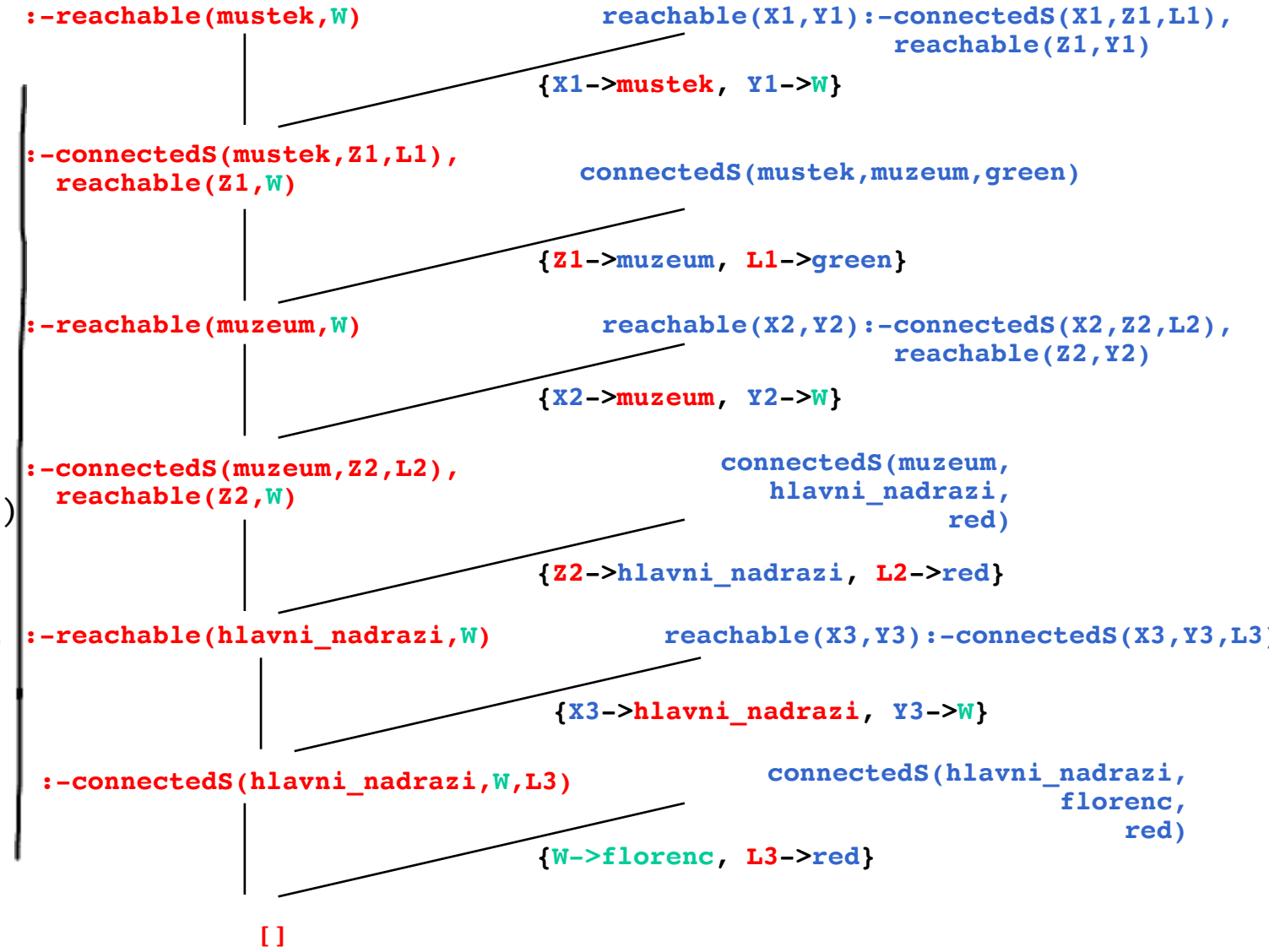
SLD-trees

```
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).
```



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!

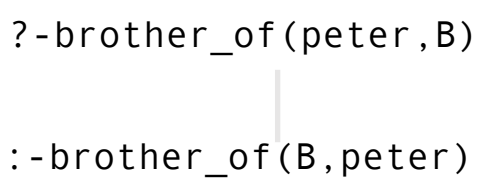
Infinite SLD-trees

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

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

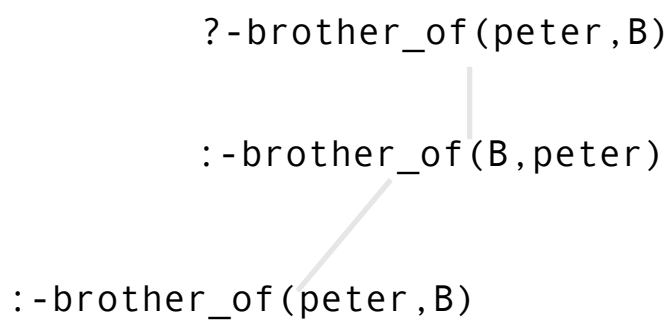
Infinite SLD-trees

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



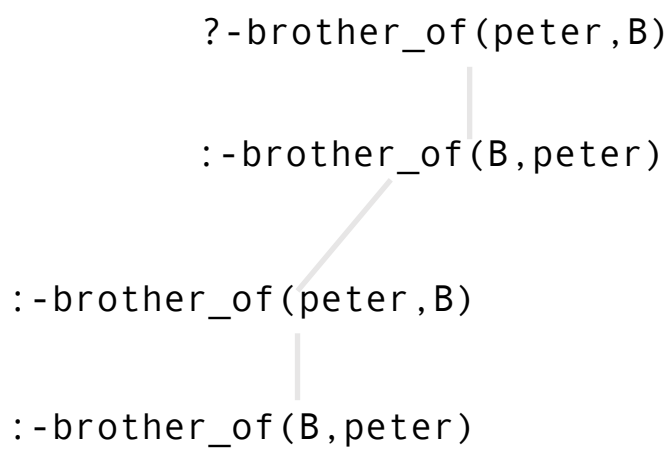
Infinite SLD-trees

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



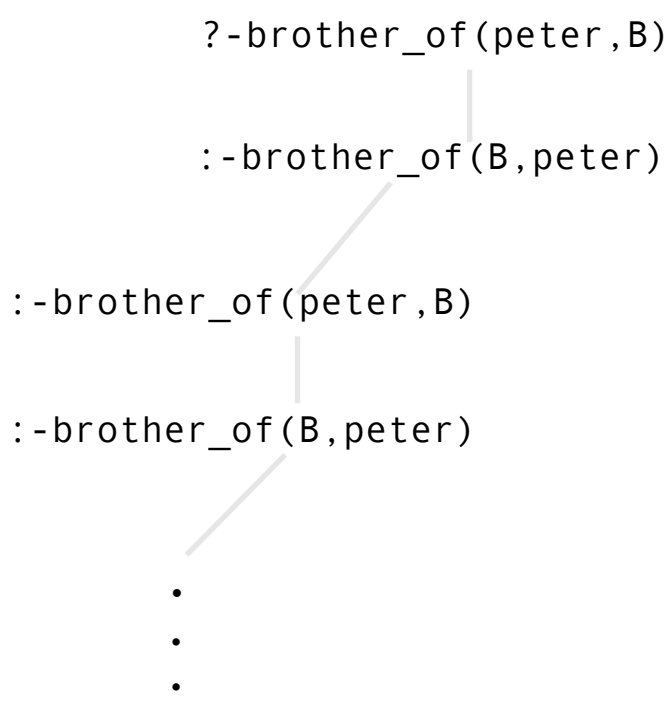
Infinite SLD-trees

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



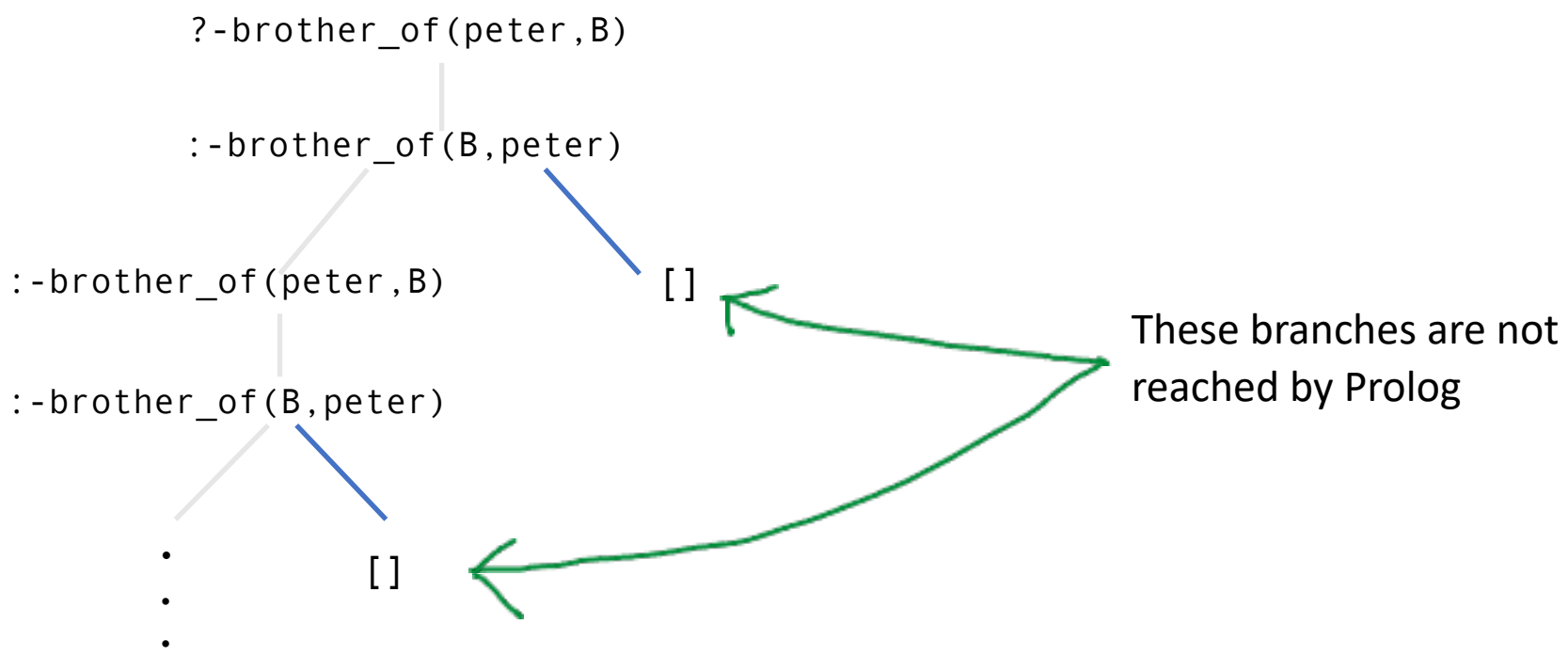
Infinite SLD-trees

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

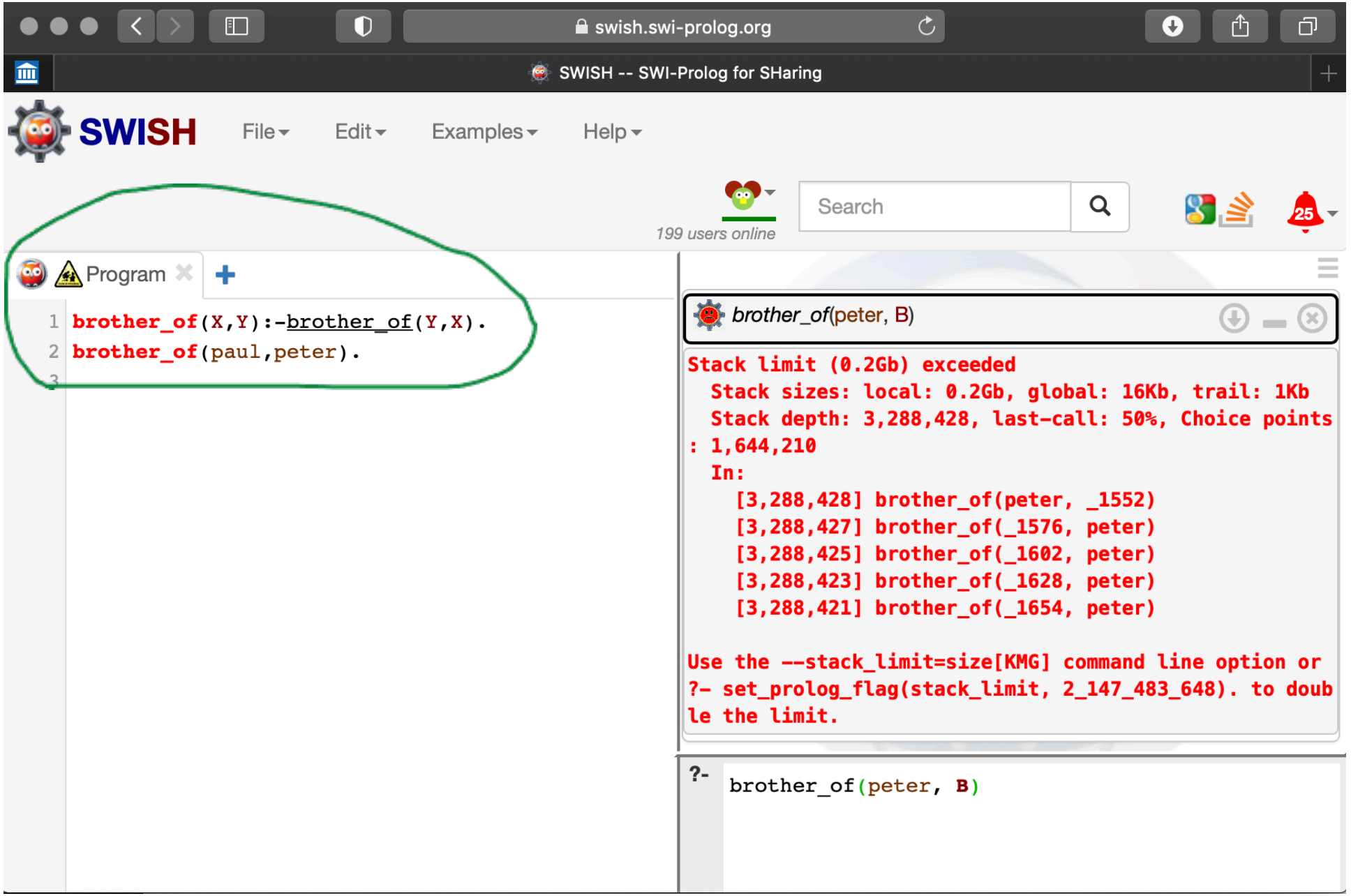


Infinite SLD-trees

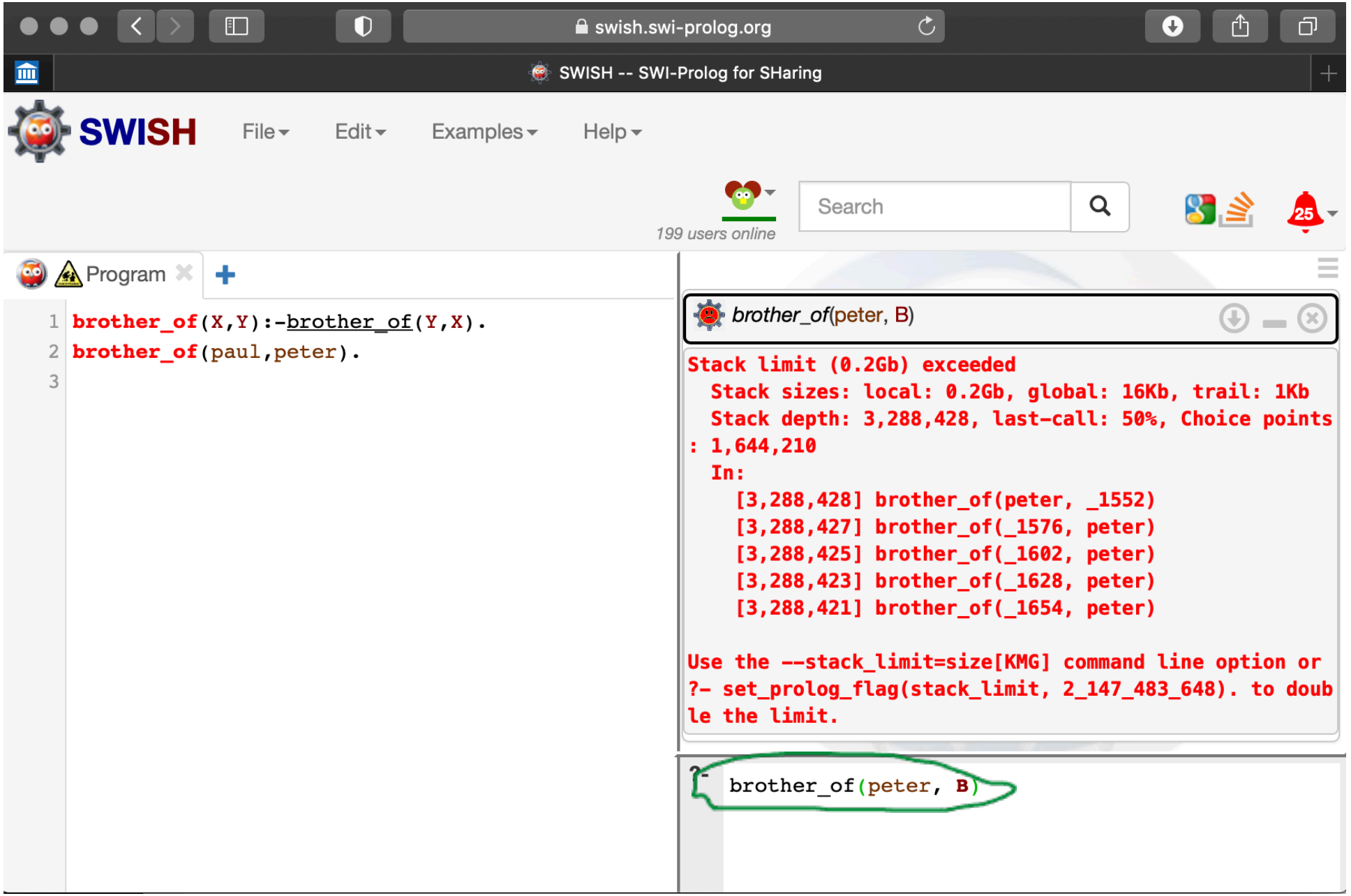
```
brother_of(X,Y):-brother_of(Y,X).
brother_of(paul,peter).
```



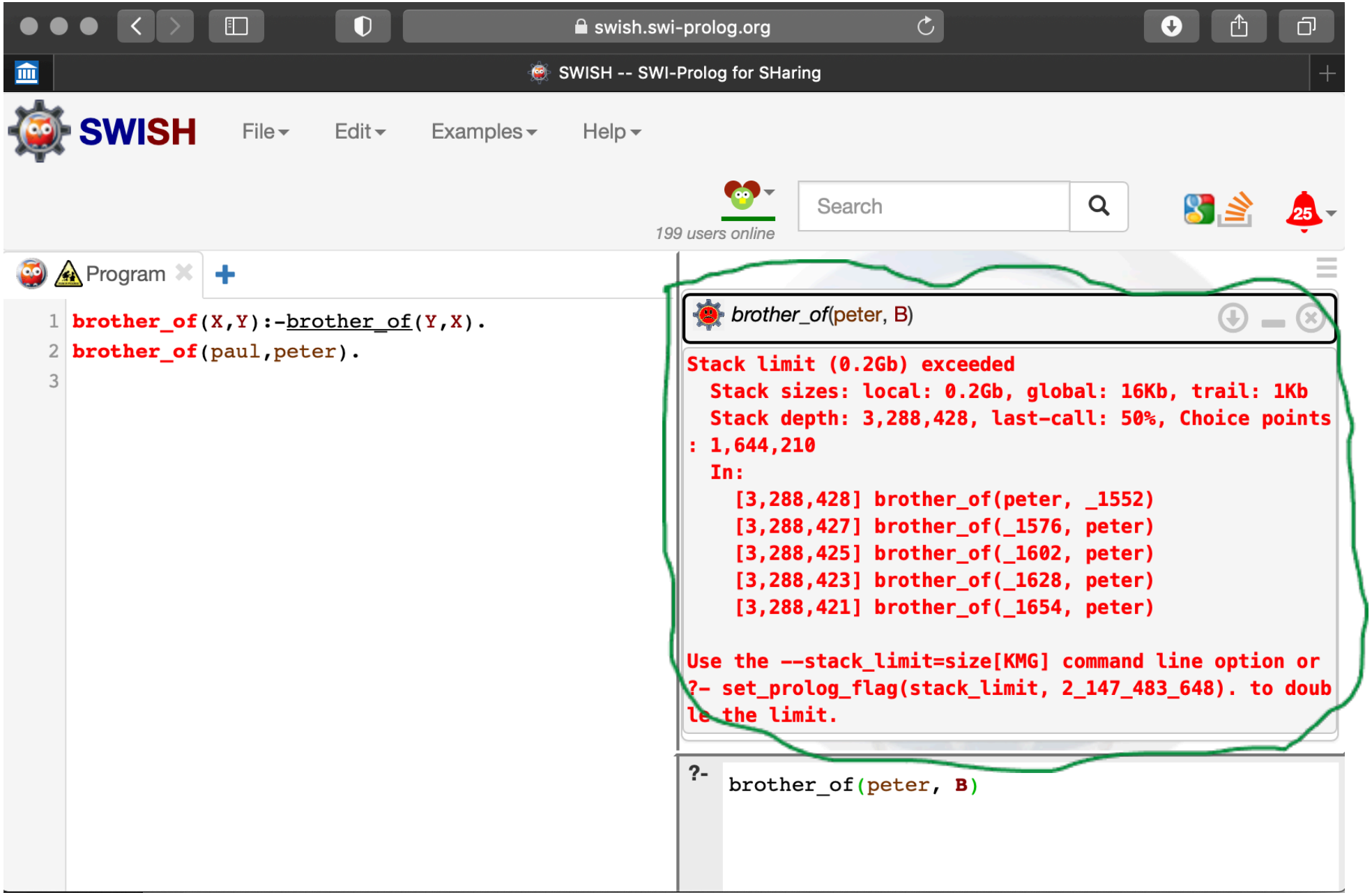
So Prolog will get stuck…



So Prolog will get stuck…



So Prolog will get stuck…

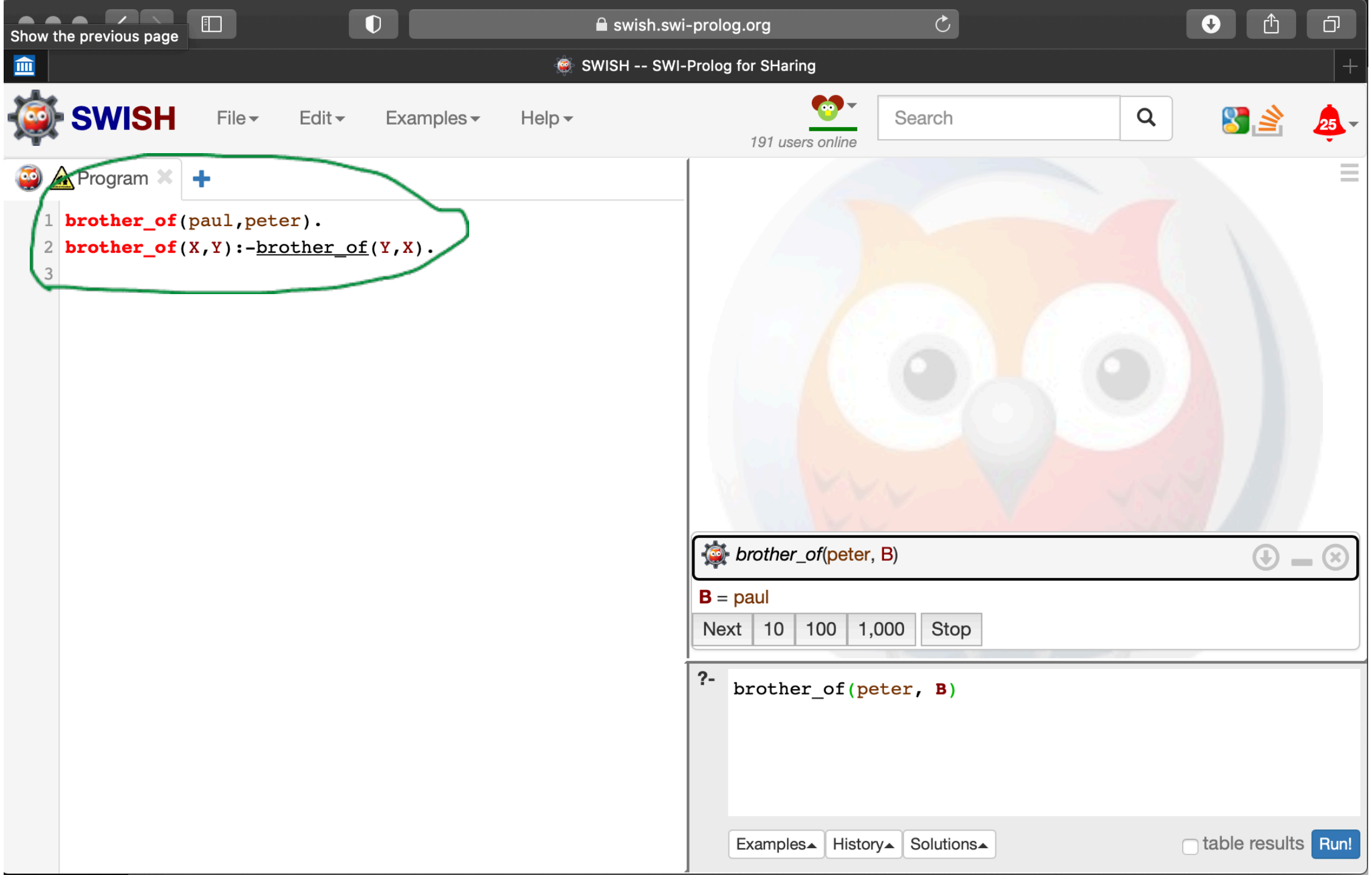


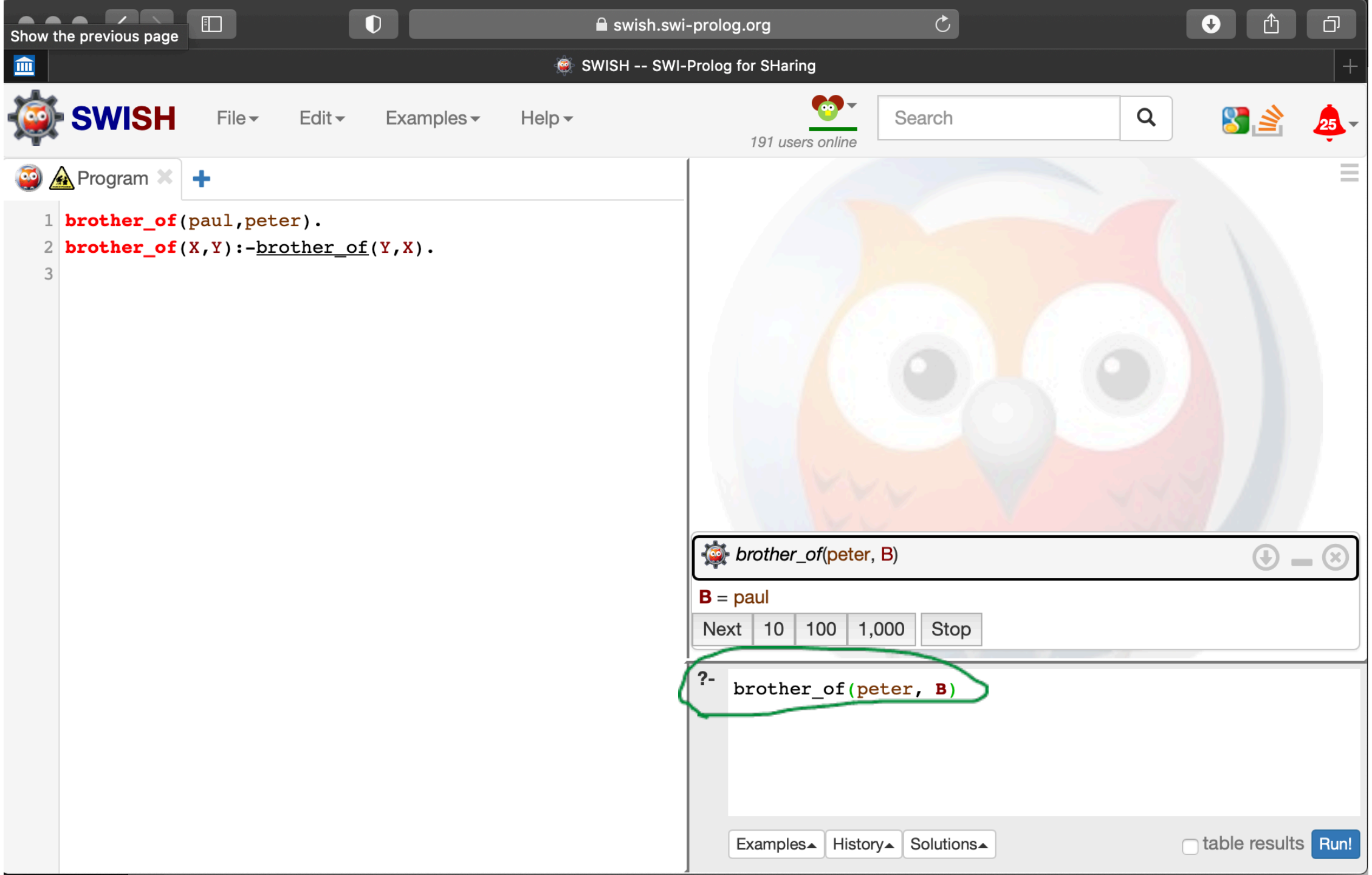
Infinite SLD-trees

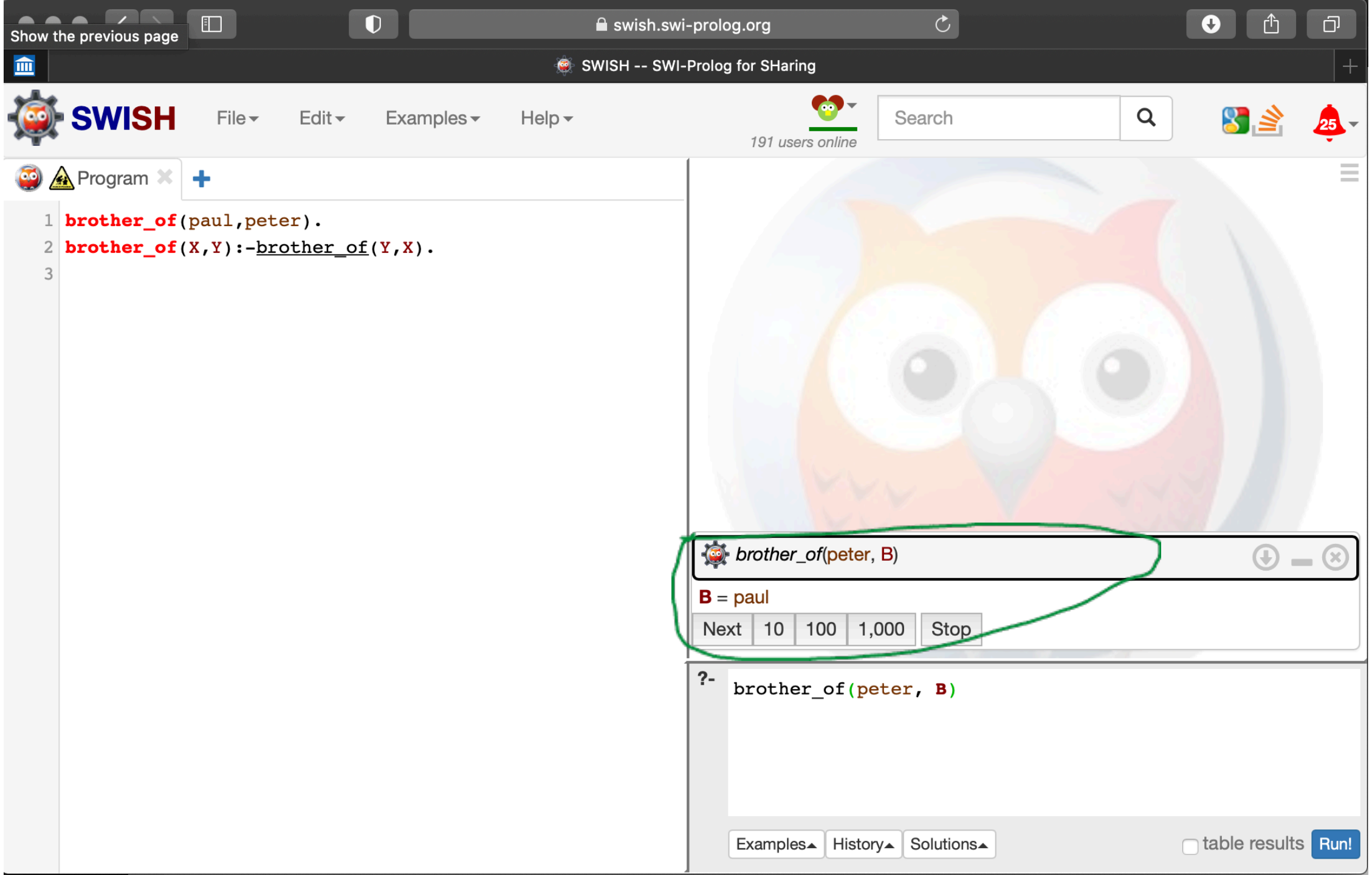
```
brother_of(X,Y):-brother_of(Y,X).
brother_of(paul,peter).
```

Infinite SLD-trees

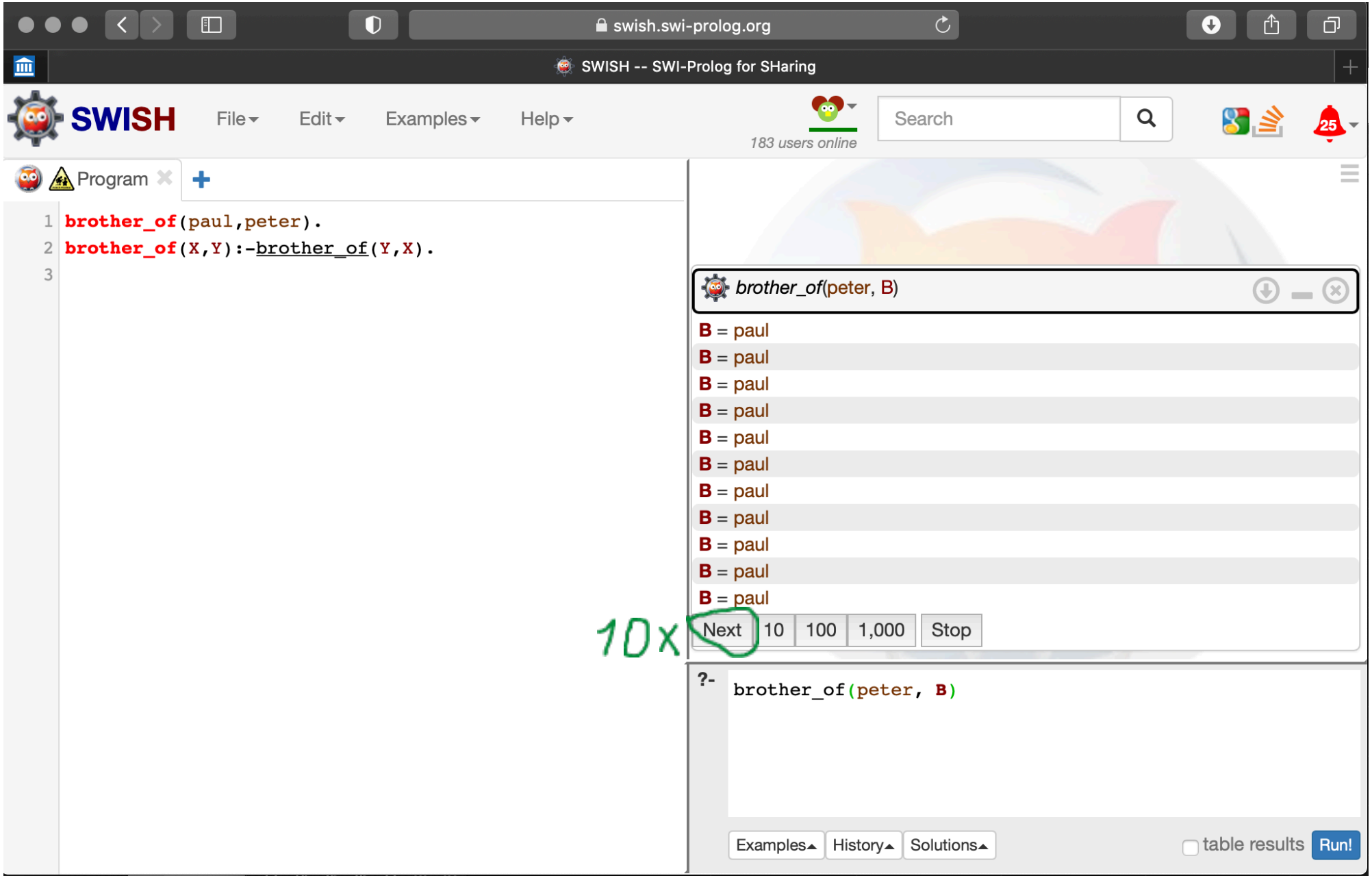
```
brother_of(X,Y):-brother_of(Y,X).
brother_of(paul,peter).
```


Show the previous page
swish.swi-prolog.org
SWISH -- SWI-Prolog for SHaring
SWISH
File
Edit
Examples
Help
191 users online
Search
25
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)
Examples
History
Solutions
table results
Run!

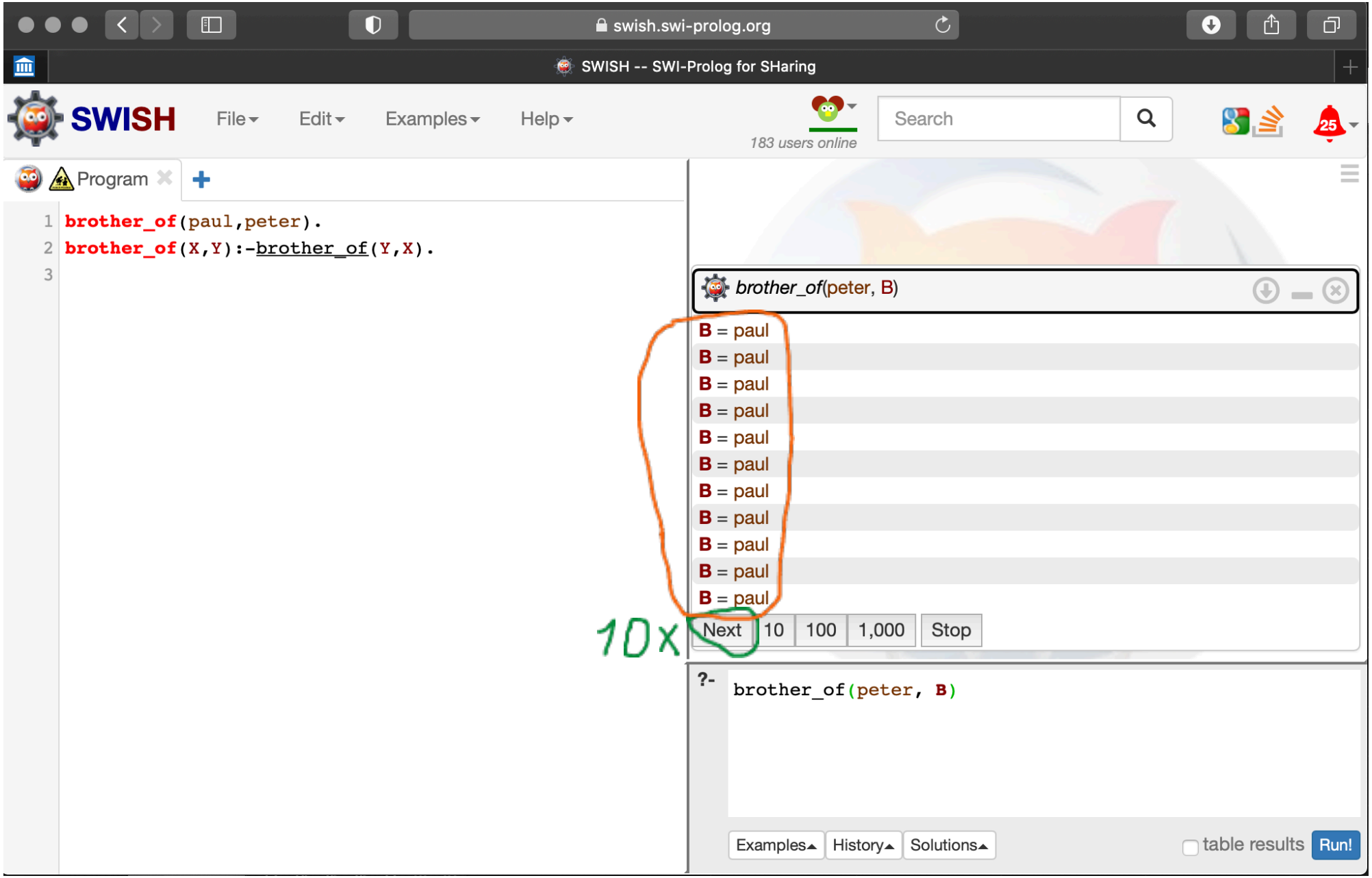

Show the previous page
swish.swi-prolog.org
SWISH -- SWI-Prolog for SHaring
SWISH
File
Edit
Examples
Help
191 users online
Search
25
Program
1 brother_of(paul,peter).
2 brother_of(X,Y):-brother_of(Y,X).
3
brother_of(peter, B)
B = paul
Next
10
100
1,000
Stop
?-
brother_of(peter, B)
Examples
History
Solutions
table results
Run!


Show the previous page
swish.swi-prolog.org
SWISH -- SWI-Prolog for SHaring
SWISH
File
Edit
Examples
Help
191 users online
Search
25
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)
Examples
History
Solutions
table results
Run!

The other branches can still be accessed…



The other branches can still be accessed…



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).
```

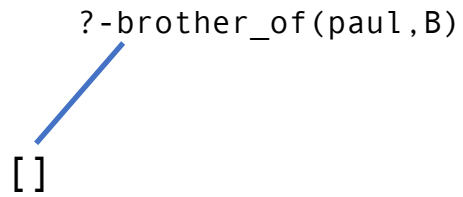
Infinite SLD-trees (Another Example)

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

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

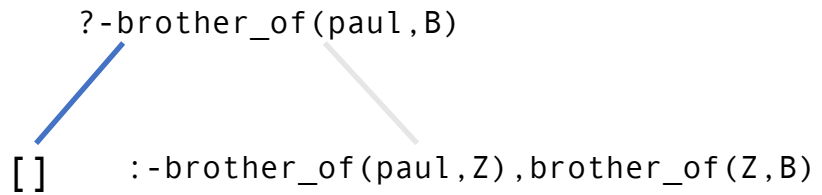
Infinite SLD-trees (Another Example)

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



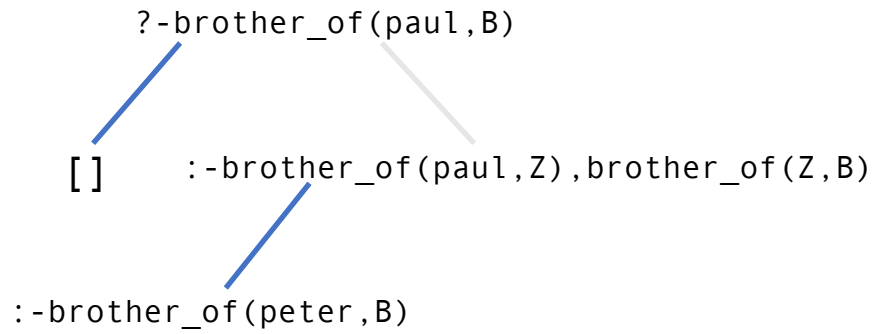
Infinite SLD-trees (Another Example)

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



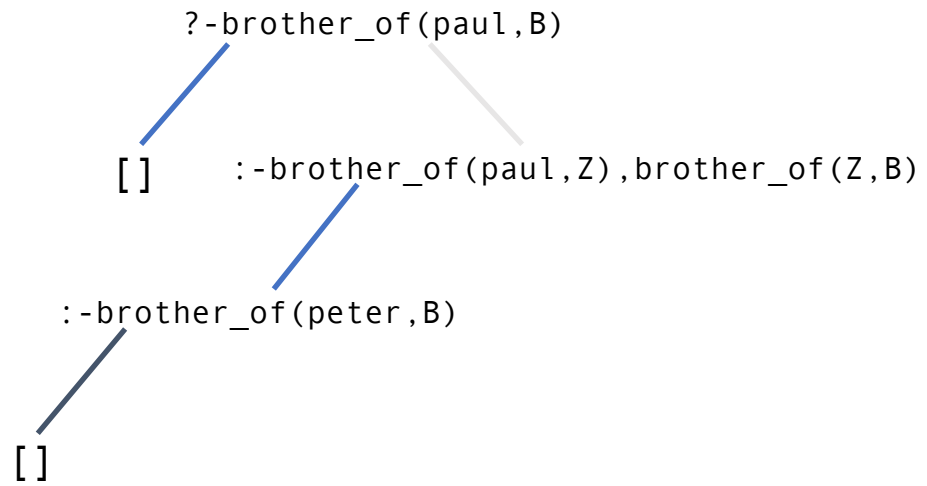
Infinite SLD-trees (Another Example)

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



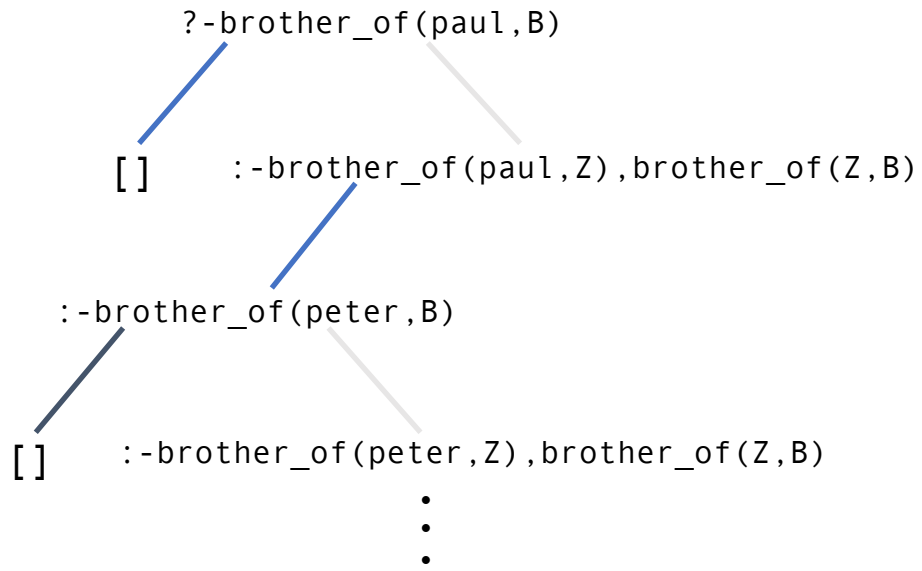
Infinite SLD-trees (Another Example)

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



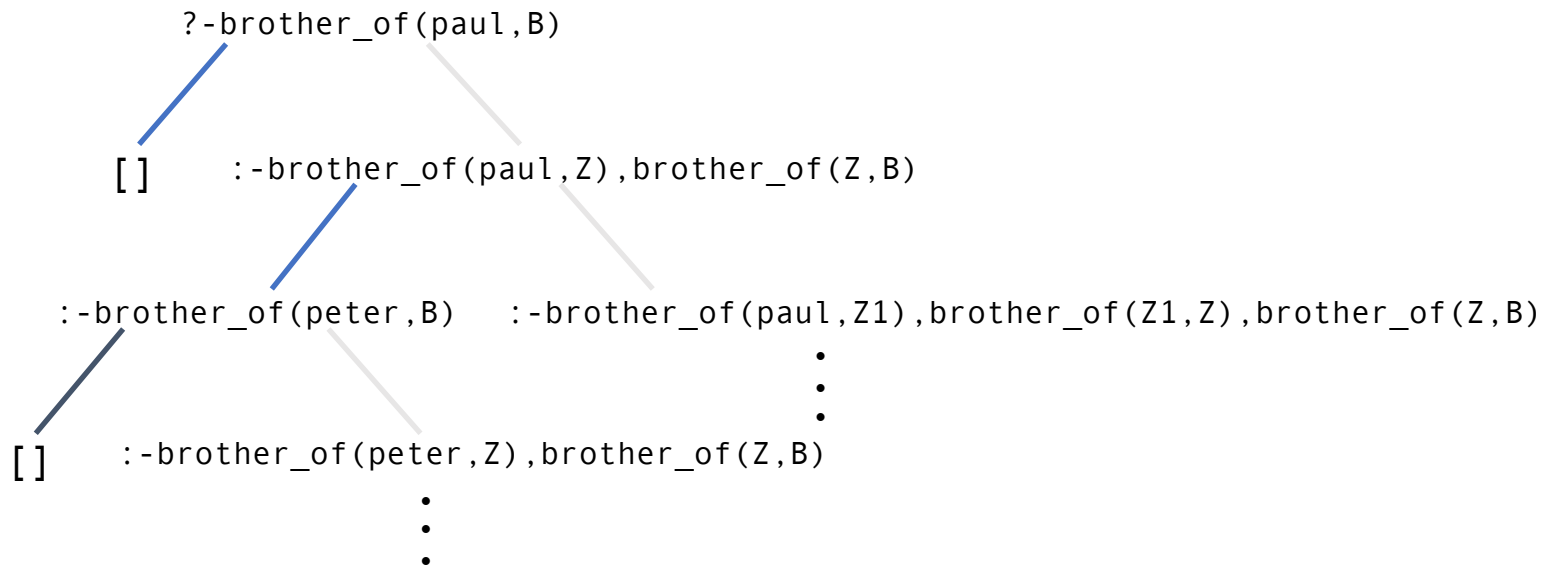
Infinite SLD-trees (Another Example)

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



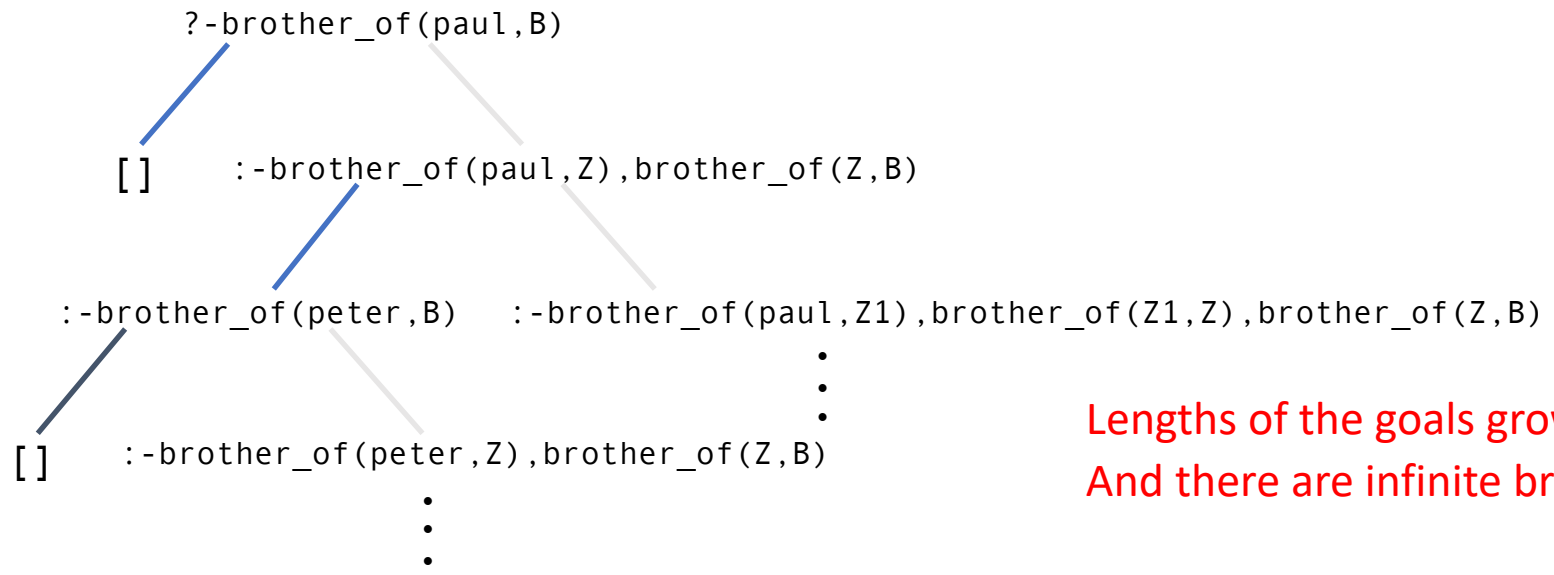
Infinite SLD-trees (Another Example)

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

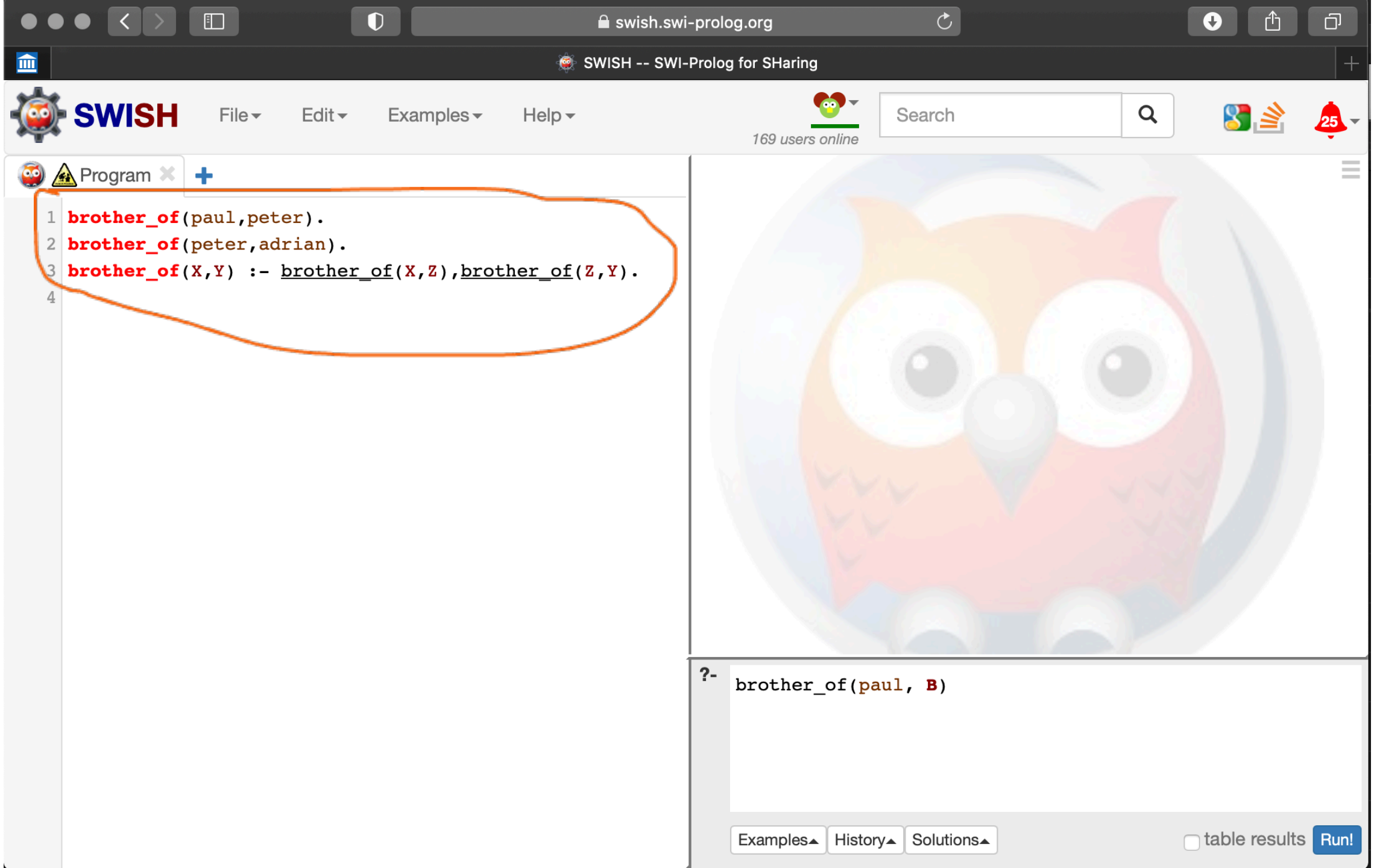


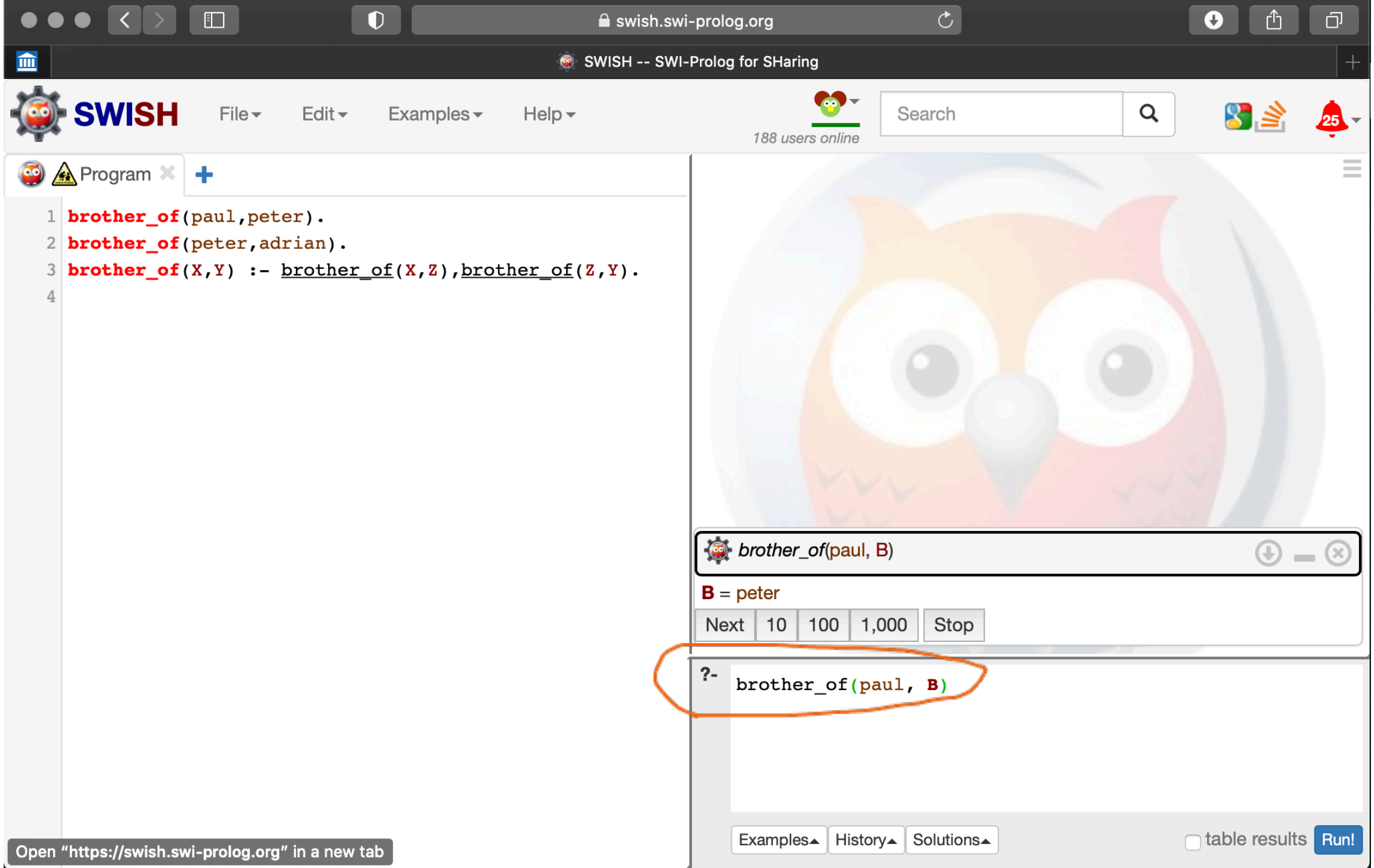
Infinite SLD-trees (Another Example)

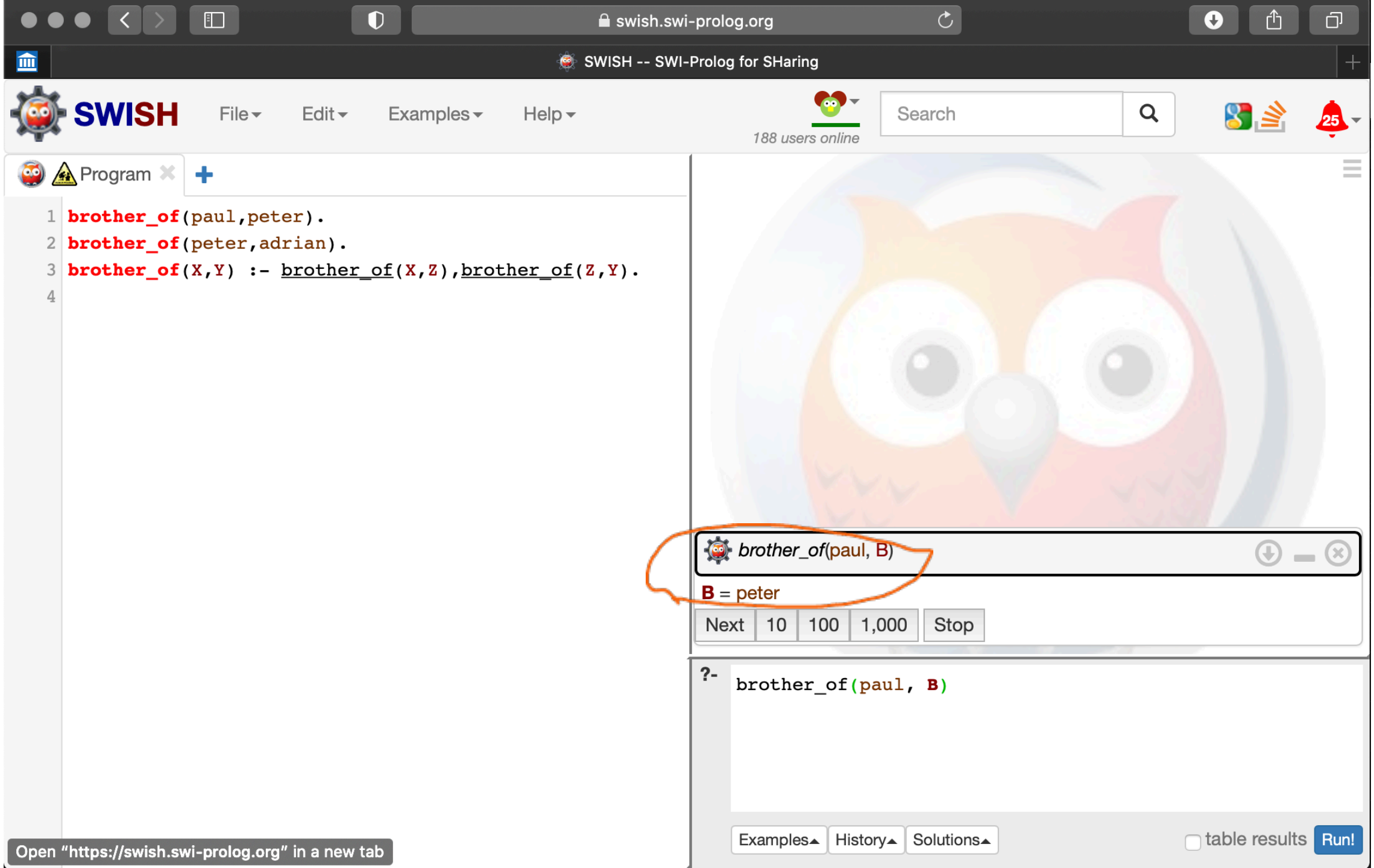
```
brother_of(paul,peter).
brother_of(peter,adrian).
brother_of(X,Y):-brother_of(X,Z), brother_of(Z,Y).
```

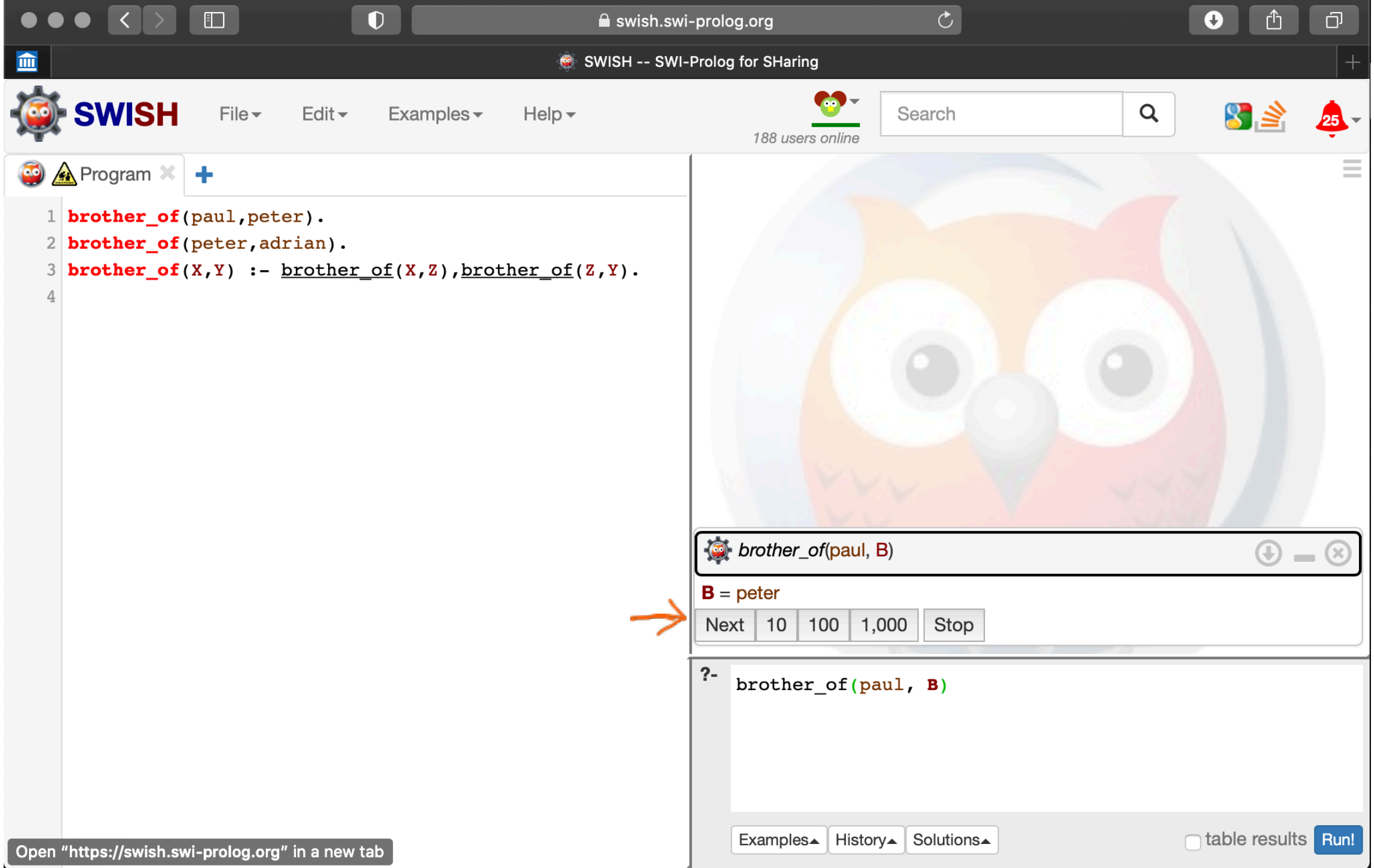


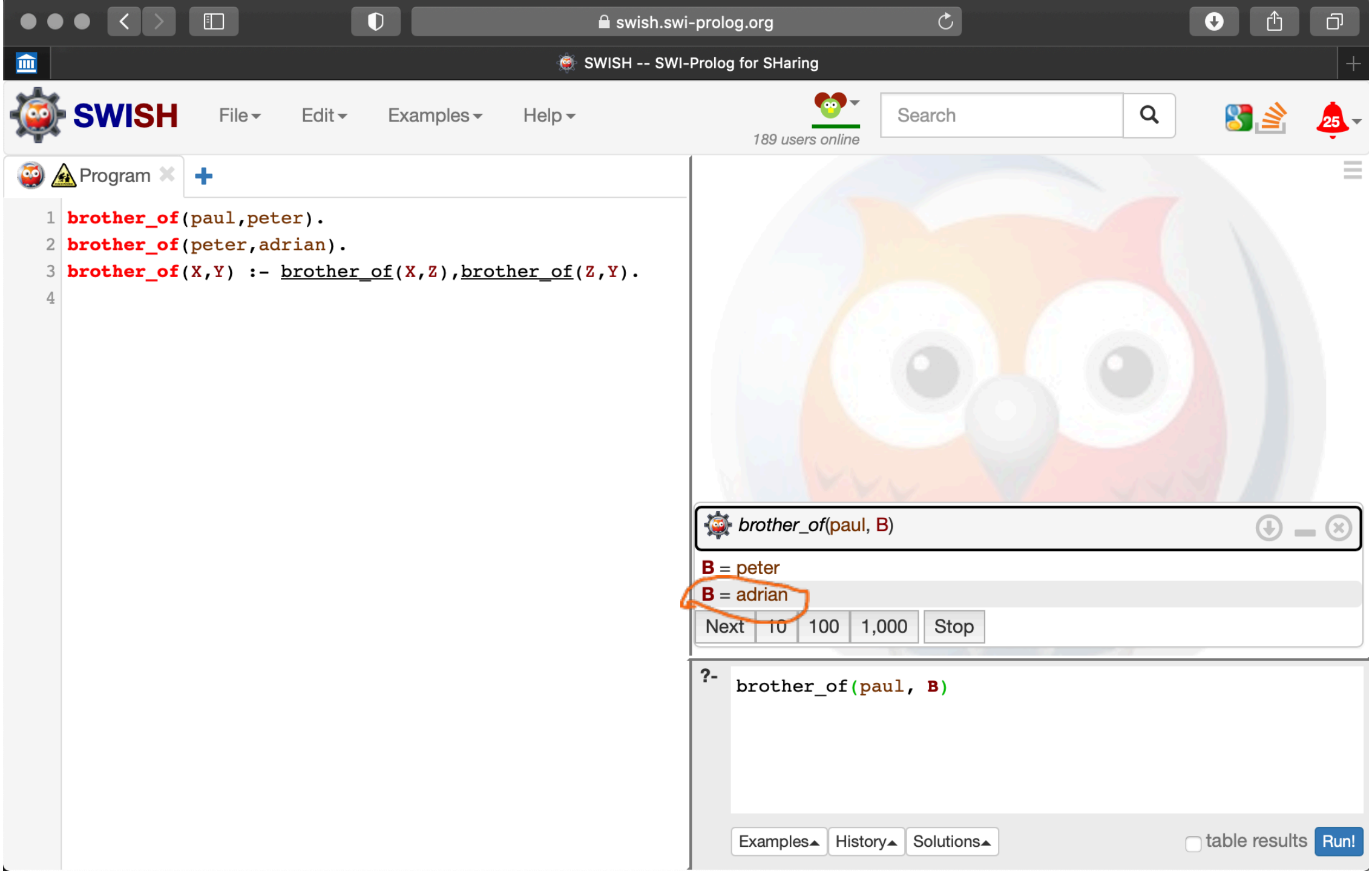
Lengths of the goals grow!
And there are infinite branches!

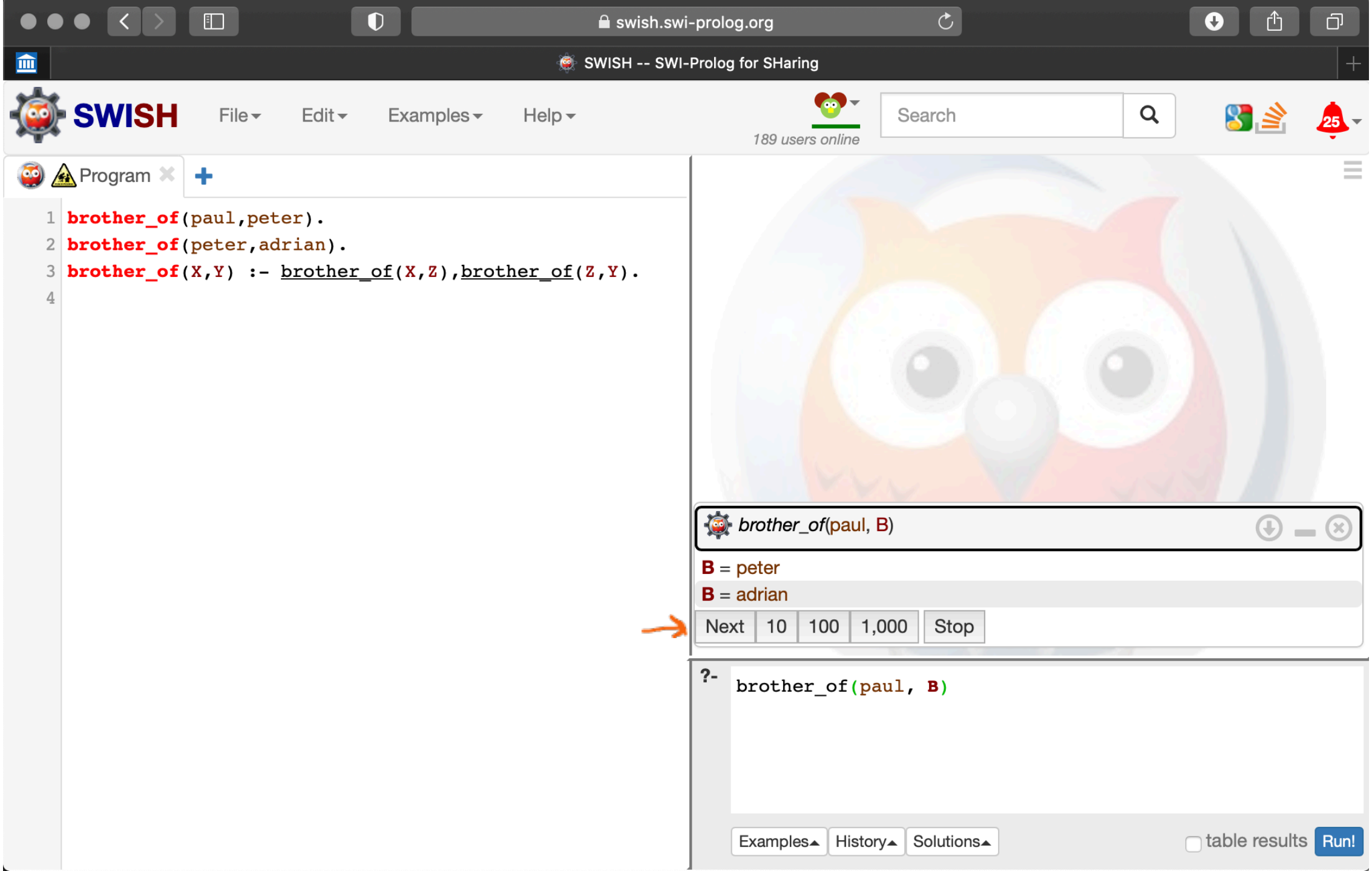

swish.swi-prolog.org
SWISH -- SWI-Prolog for SHaring
SWISH
File
Edit
Examples
Help
169 users online
Search
25
Program
1 brother_of(paul,peter).
2 brother_of(peter,adrian).
3 brother_of(X,Y) :- brother_of(X,Z),brother_of(Z,Y).
4
?-
brother_of(paul, B)
Examples
History
Solutions
table results
Run!


swish.swi-prolog.org
SWIS -- SWI-Prolog for SHaring
SWISH
File
Edit
Examples
Help
188 users online
Search
25
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


swish.swi-prolog.org
SWIS -- SWI-Prolog for SHaring
SWISH
File
Edit
Examples
Help
188 users online
Search
25
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


swish.swi-prolog.org
SWIH -- SWI-Prolog for SHaring
SWISH
File
Edit
Examples
Help
188 users online
Search
25
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


swish.swi-prolog.org
SWISH -- SWI-Prolog for SHaring
SWISH
File
Edit
Examples
Help
189 users online
Search
25
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)
Examples
History
Solutions
table results
Run!


swish.swi-prolog.org
SWISH -- SWI-Prolog for SHaring
SWISH
File
Edit
Examples
Help
189 users online
Search
25
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)
Examples
History
Solutions
table results
Run!

swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

SWISH File Edit Examples Help

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

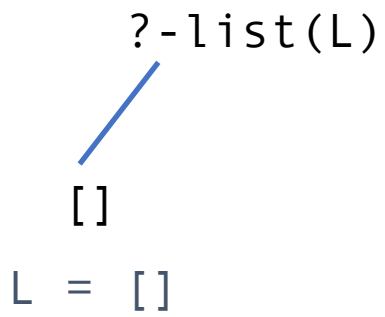
```
list([]).
list([H|T]):-list(T).
```

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

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

Exercise 3.2

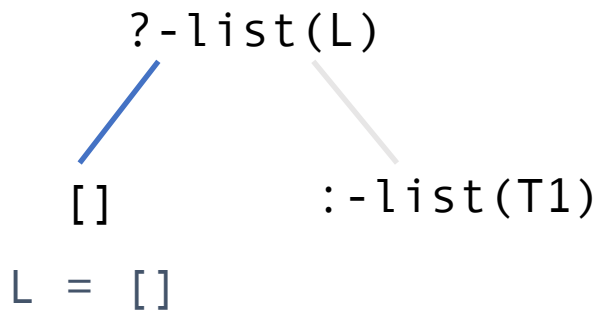
```
list([]).
list([H|T]):-list(T).
```




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

Exercise 3.2

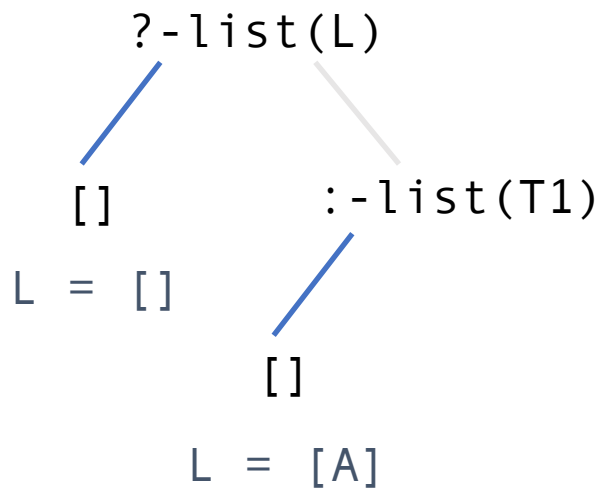
```
list([]).
list([H|T]):-list(T).
```




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

Exercise 3.2

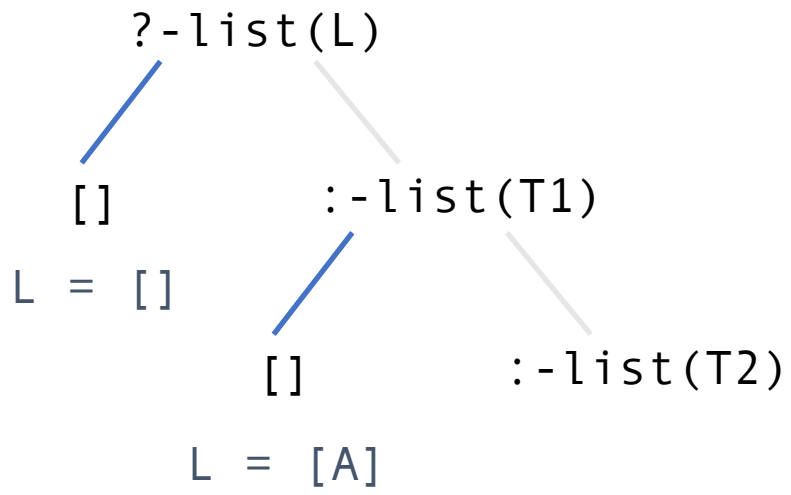
```
list([]).
list([H|T]):-list(T).
```




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

Exercise 3.2

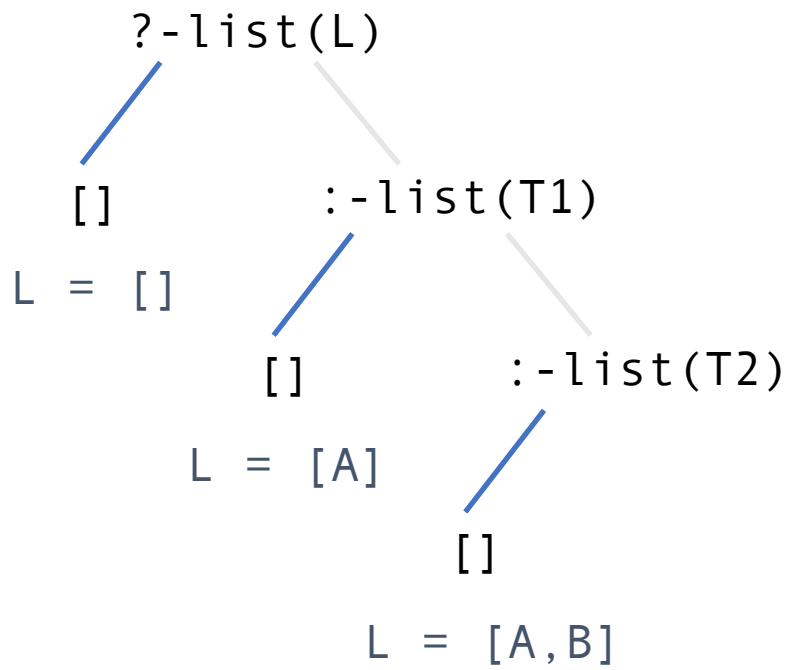
```
list([]).
list([H|T]):-list(T).
```




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

Exercise 3.2

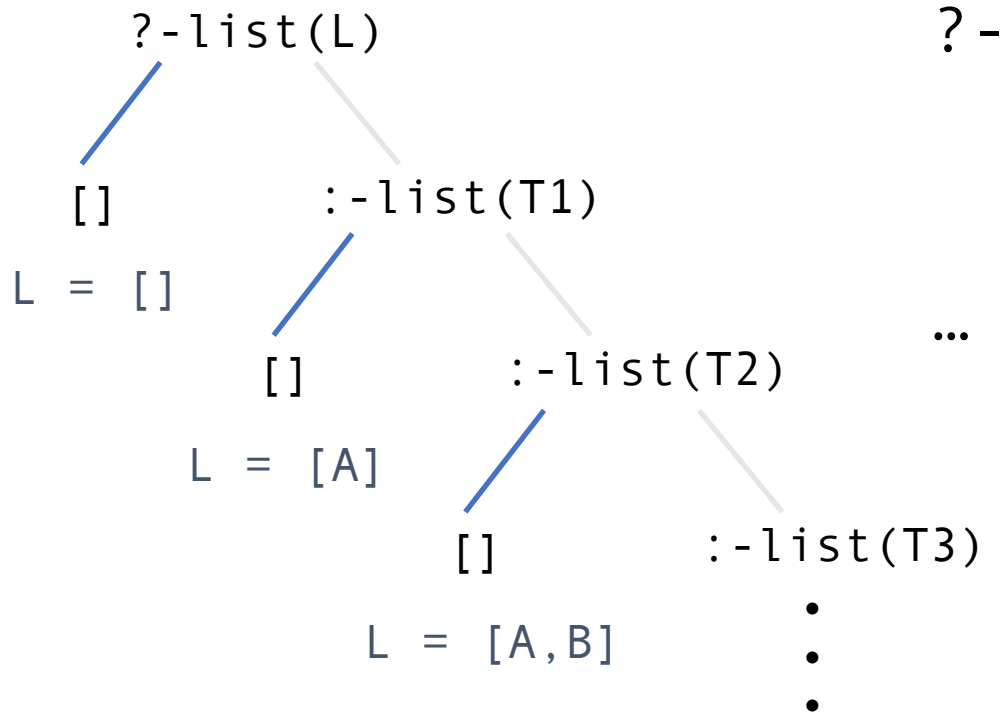
```
list([]).
list([H|T]):-list(T).
```



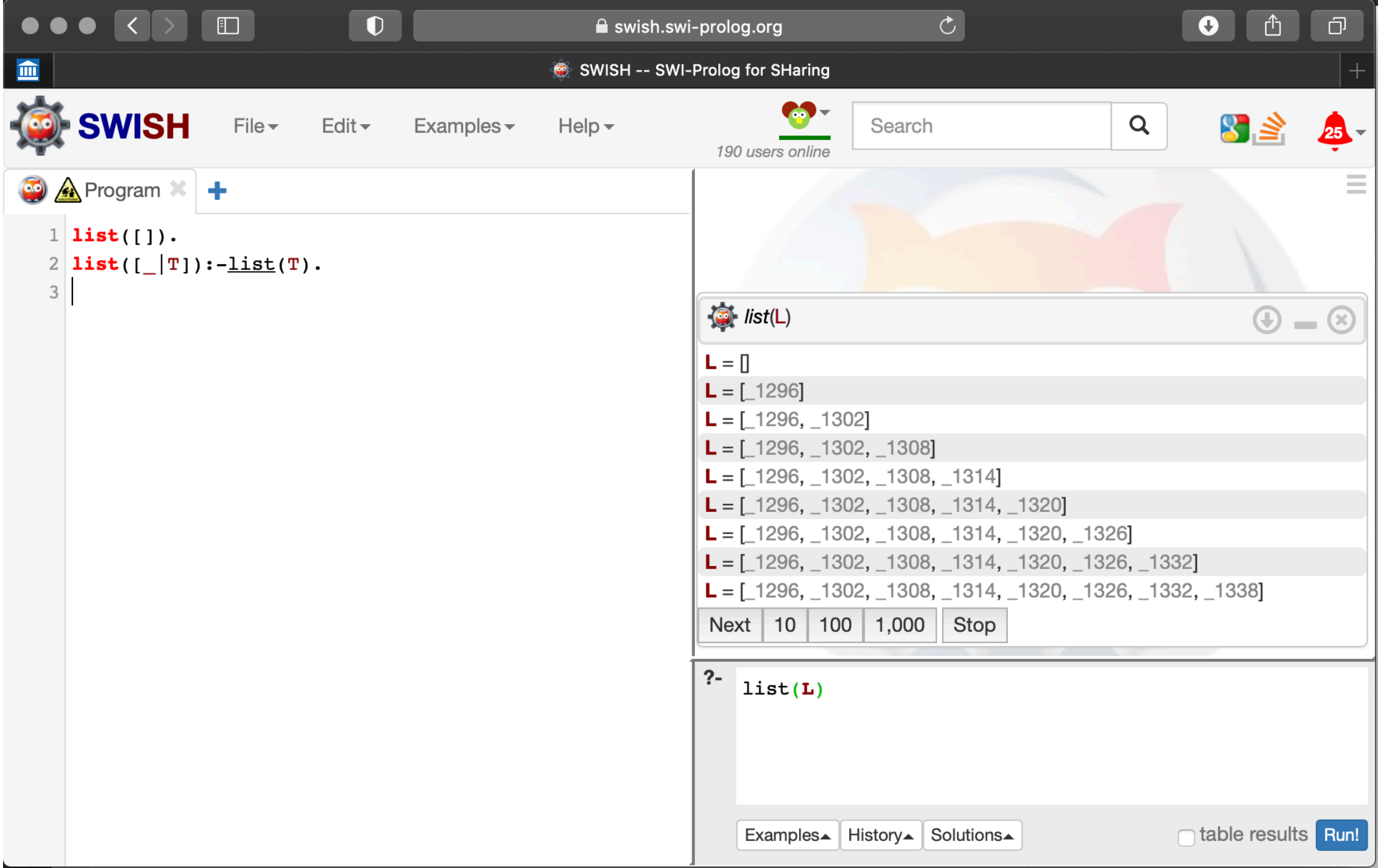

```
?-list(L).
   L = [];
   L = [A];
   L = [A,B];
```

Exercise 3.2

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




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


swish.swi-prolog.org
SWISH -- SWI-Prolog for SHaring
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!

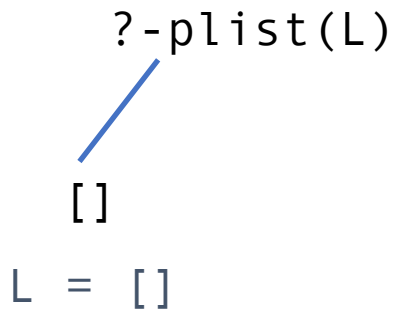
Another example

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

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

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

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

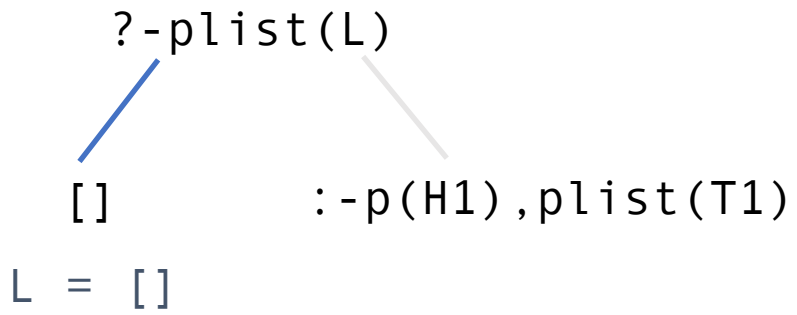
Another example




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

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

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

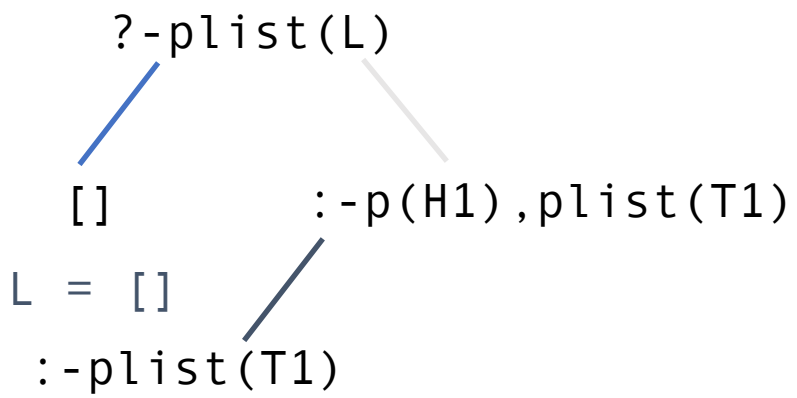
Another example




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

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

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

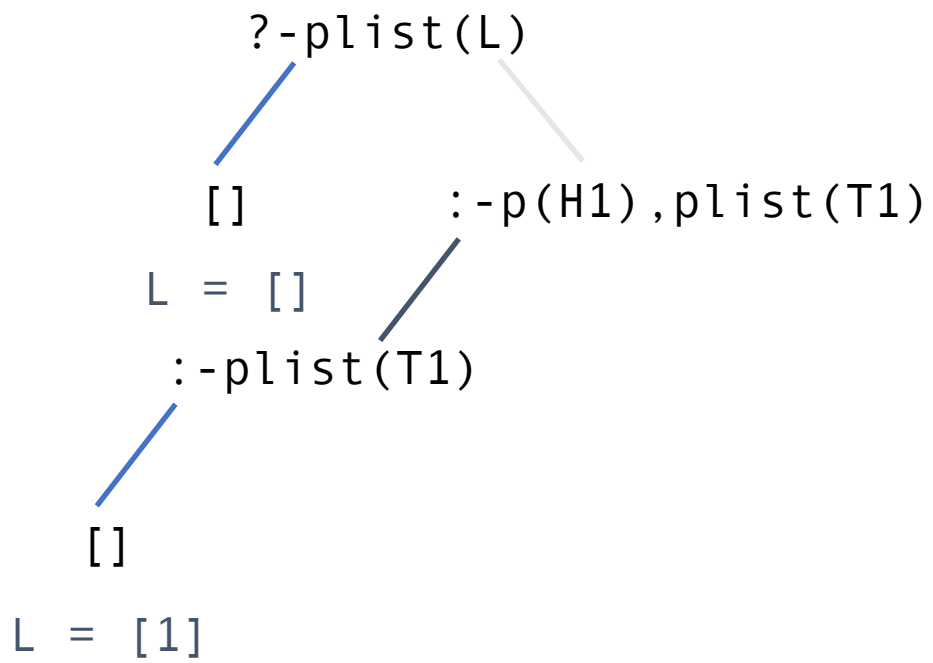
Another example




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

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

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

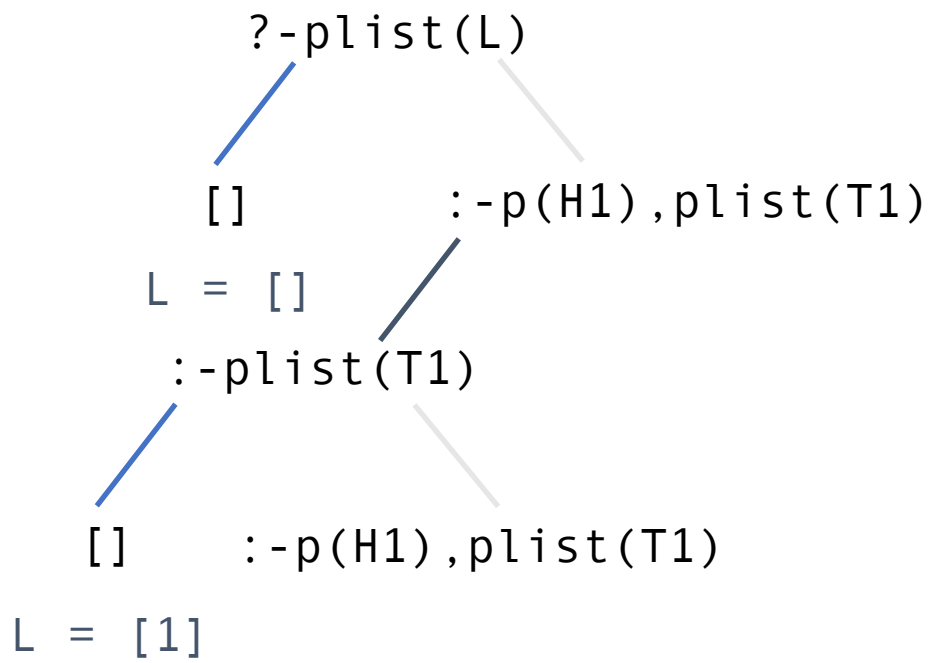
Another example




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

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

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

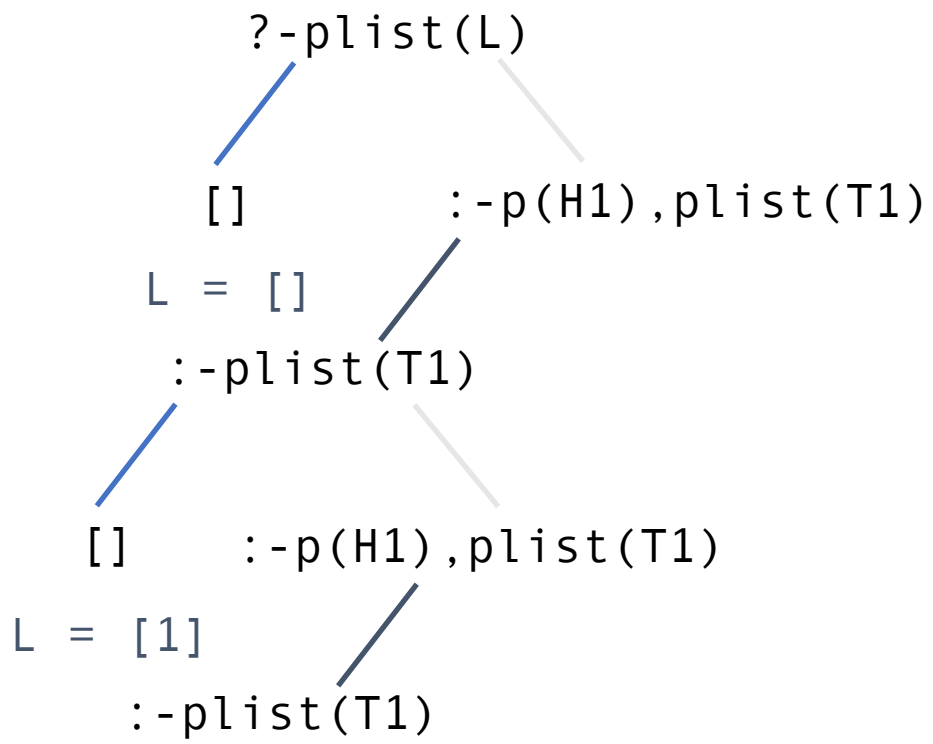
Another example




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

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

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

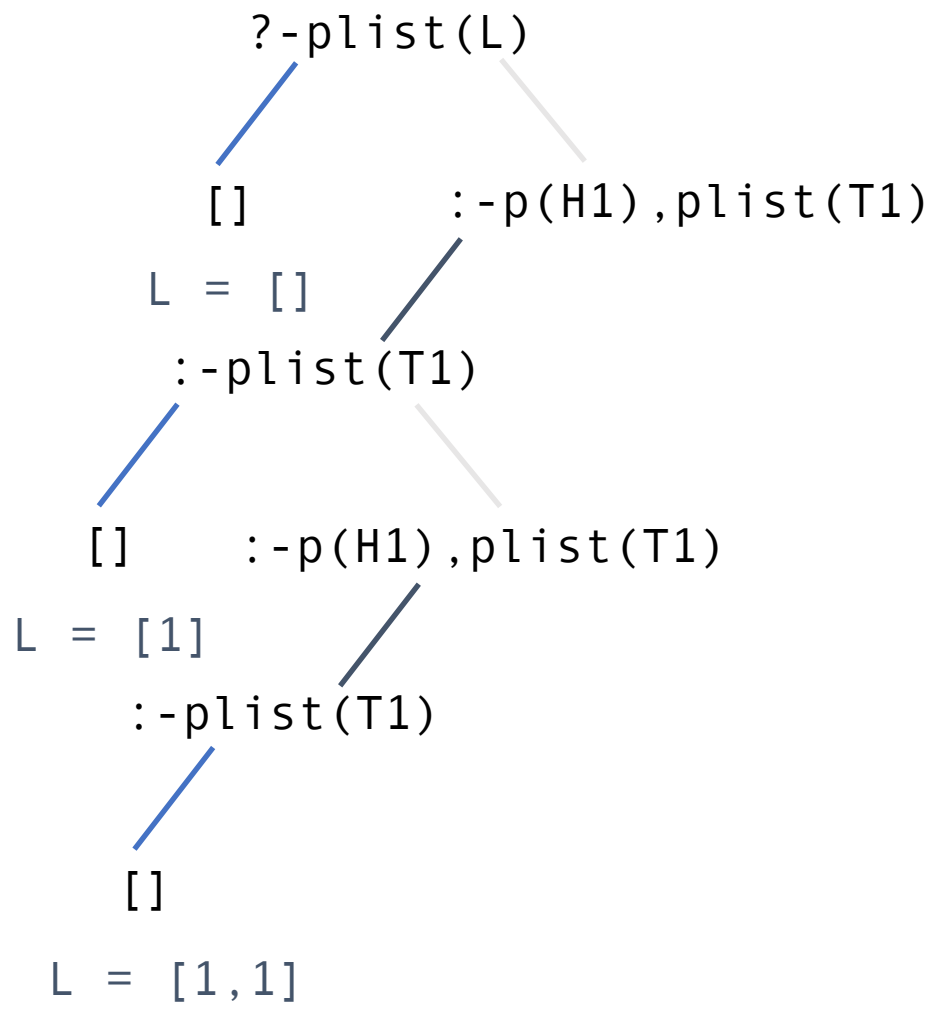
Another example




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

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

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

Another example

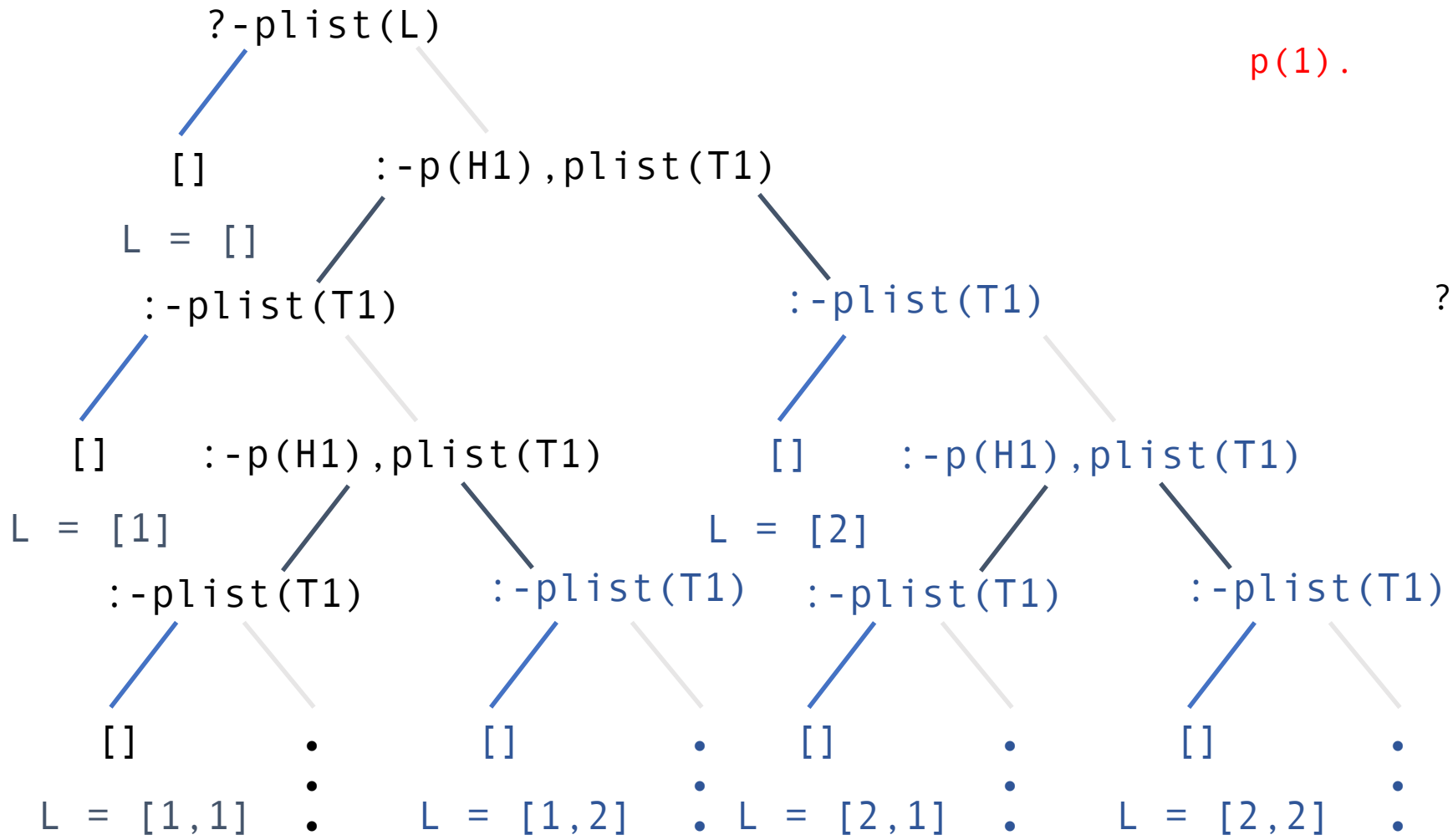



```
plist([]).
plist([H|T]):-
        p(H),plist(T).

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

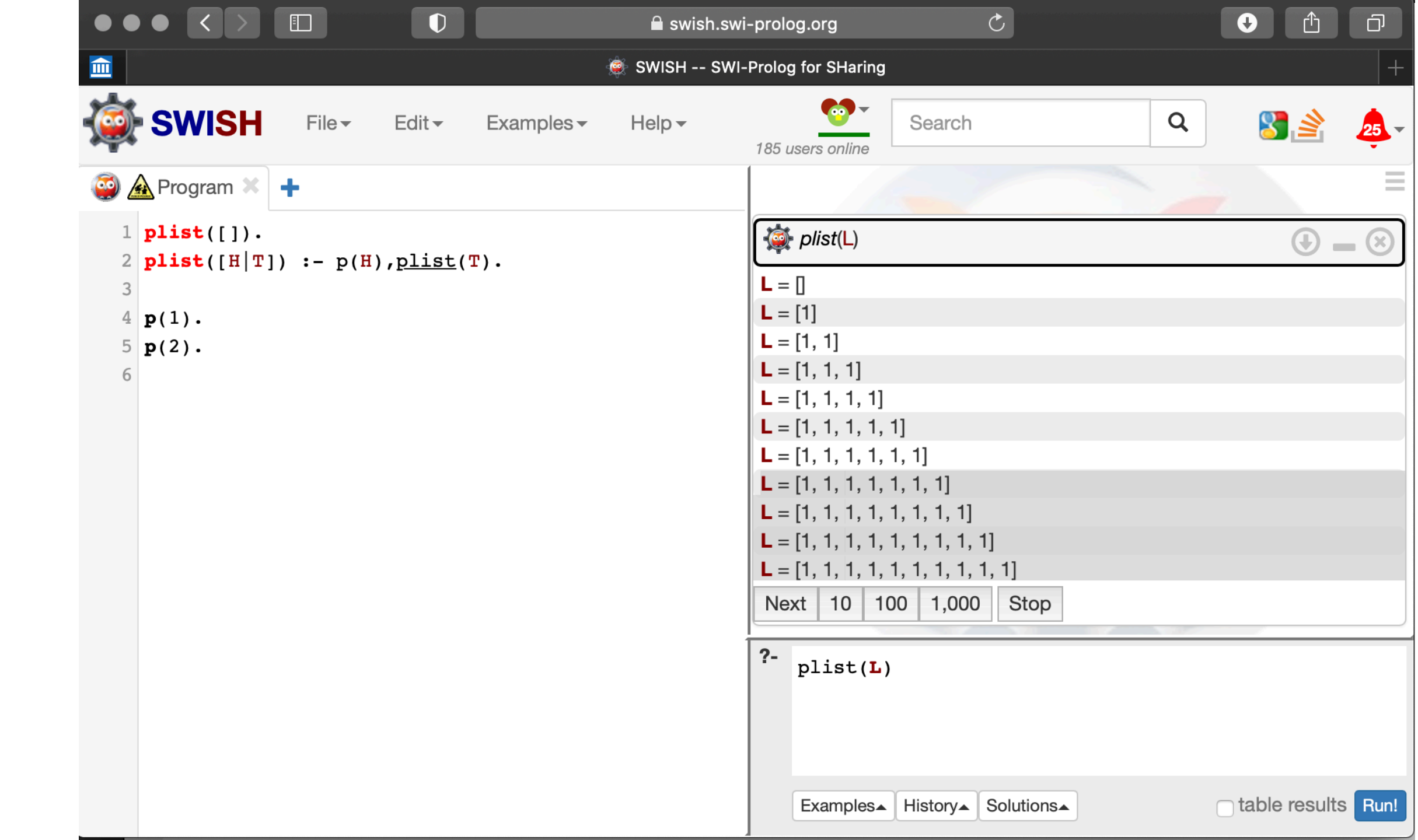
```
?-plist(L).
  L=[];
  L=[1];
  L=[1,1];
```

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];
   …
```


swish.swi-prolog.org
SWISH -- SWI-Prolog for SHaring
SWISH
File
Edit
Examples
Help
185 users online
Search
25
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!

swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

SWISH 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

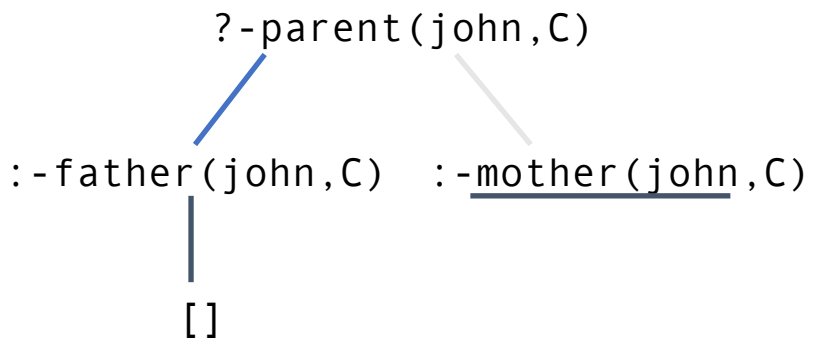
?- *plist*([X,Y,2|T]), L = [X,Y,2|T]

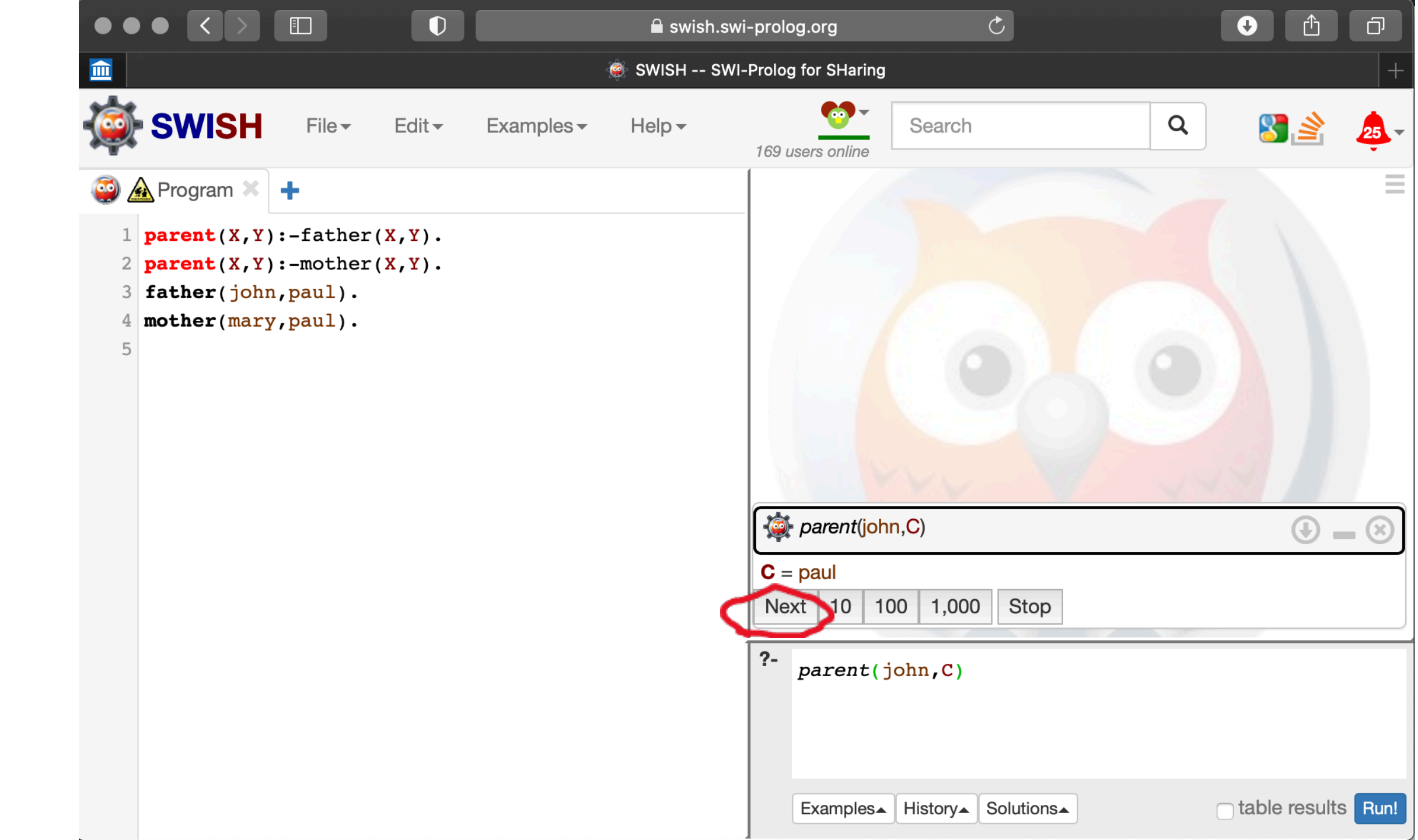
Examples History Solutions table results Run!

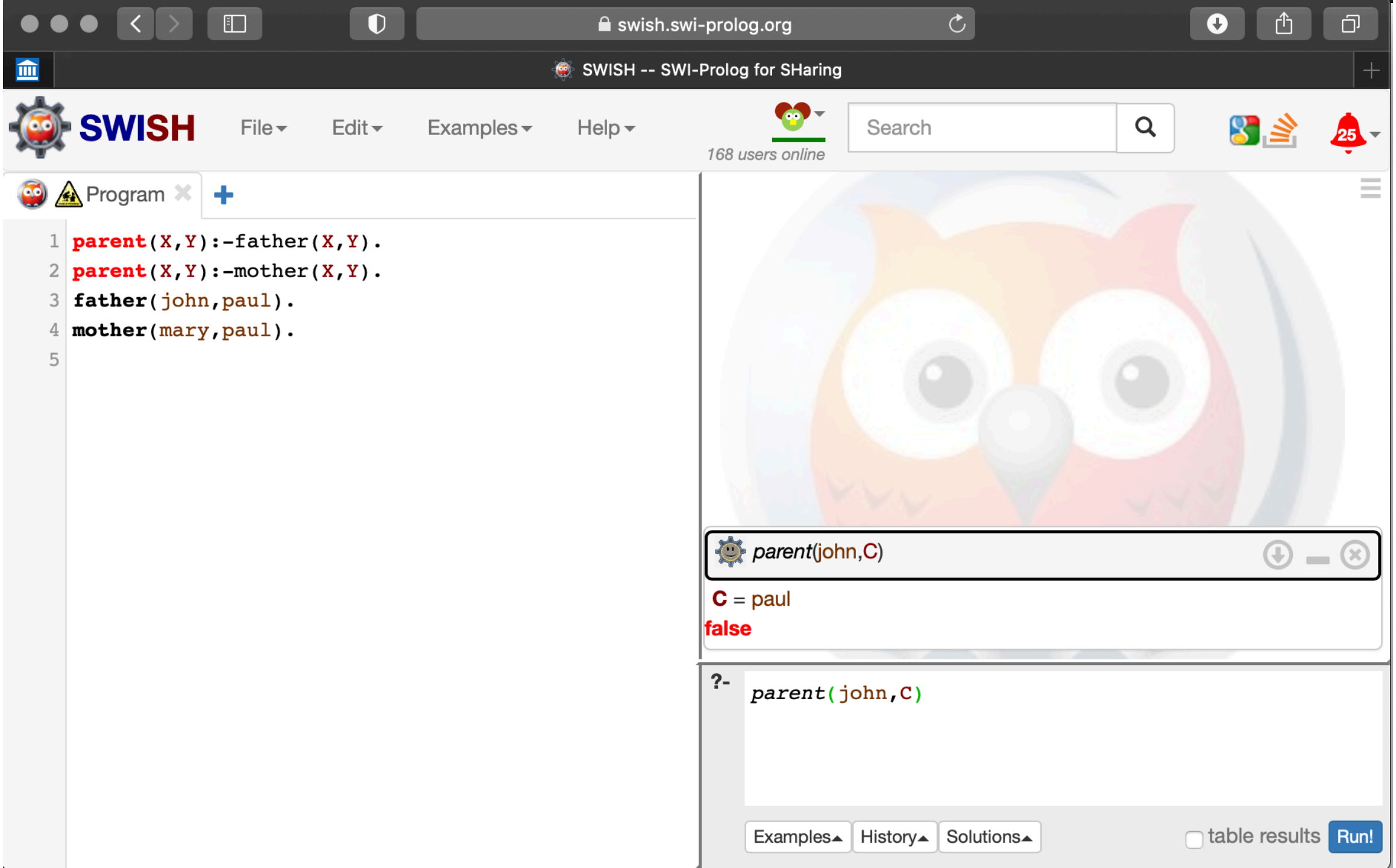
Cut

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
SWISH
File
Edit
Examples
Help
169 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
Next
10
100
1,000
Stop
?-
parent(john,C)
Examples
History
Solutions
table results
Run!


swish.swi-prolog.org
SWISH -- SWI-Prolog for SHaring
SWISH
File
Edit
Examples
Help
168 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
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).
```

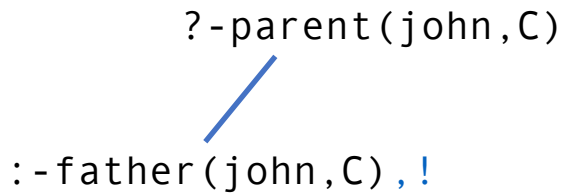
CUT

```
?-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).
```

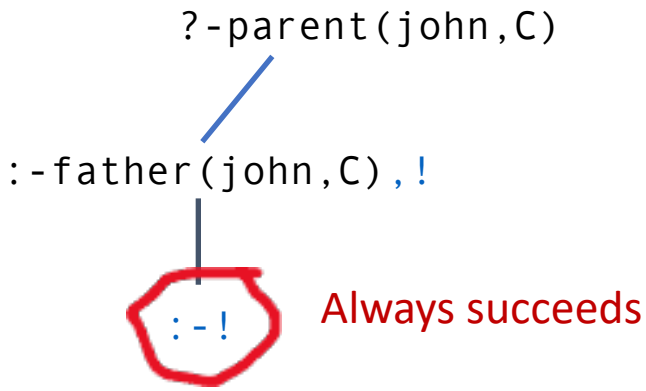
CUT



Pruning by means of cut

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

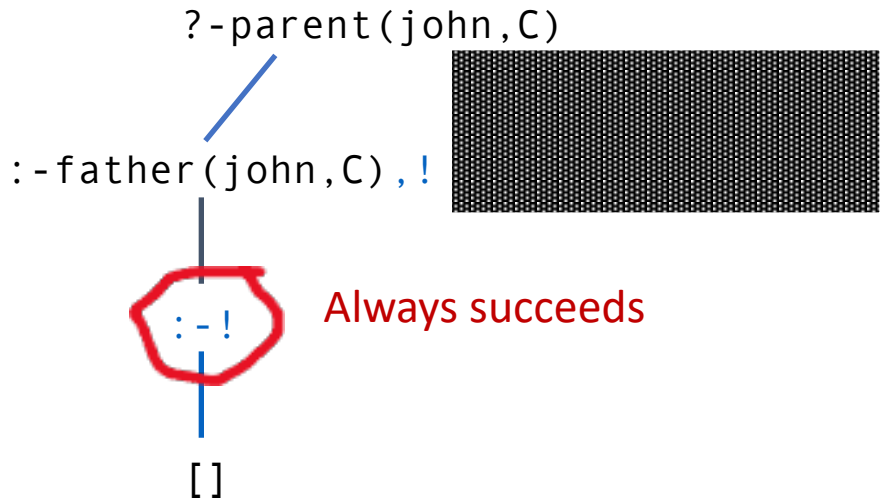
CUT



Pruning by means of cut

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

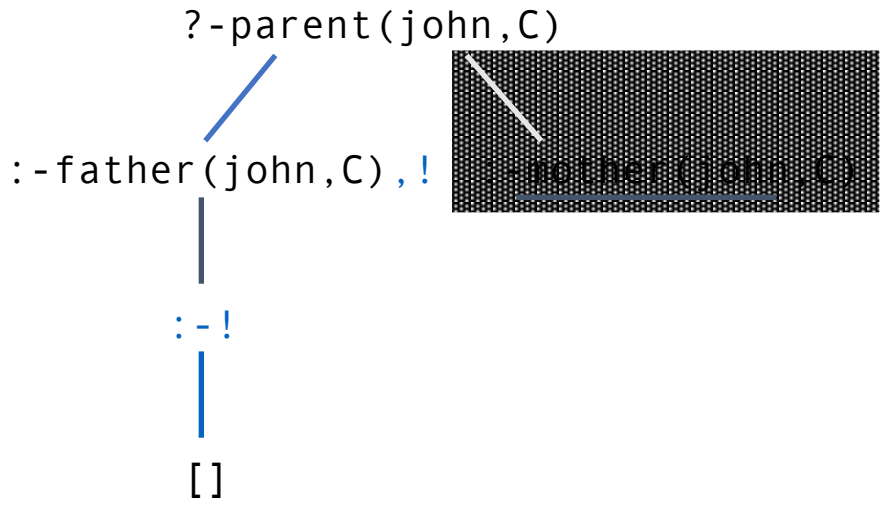
CUT

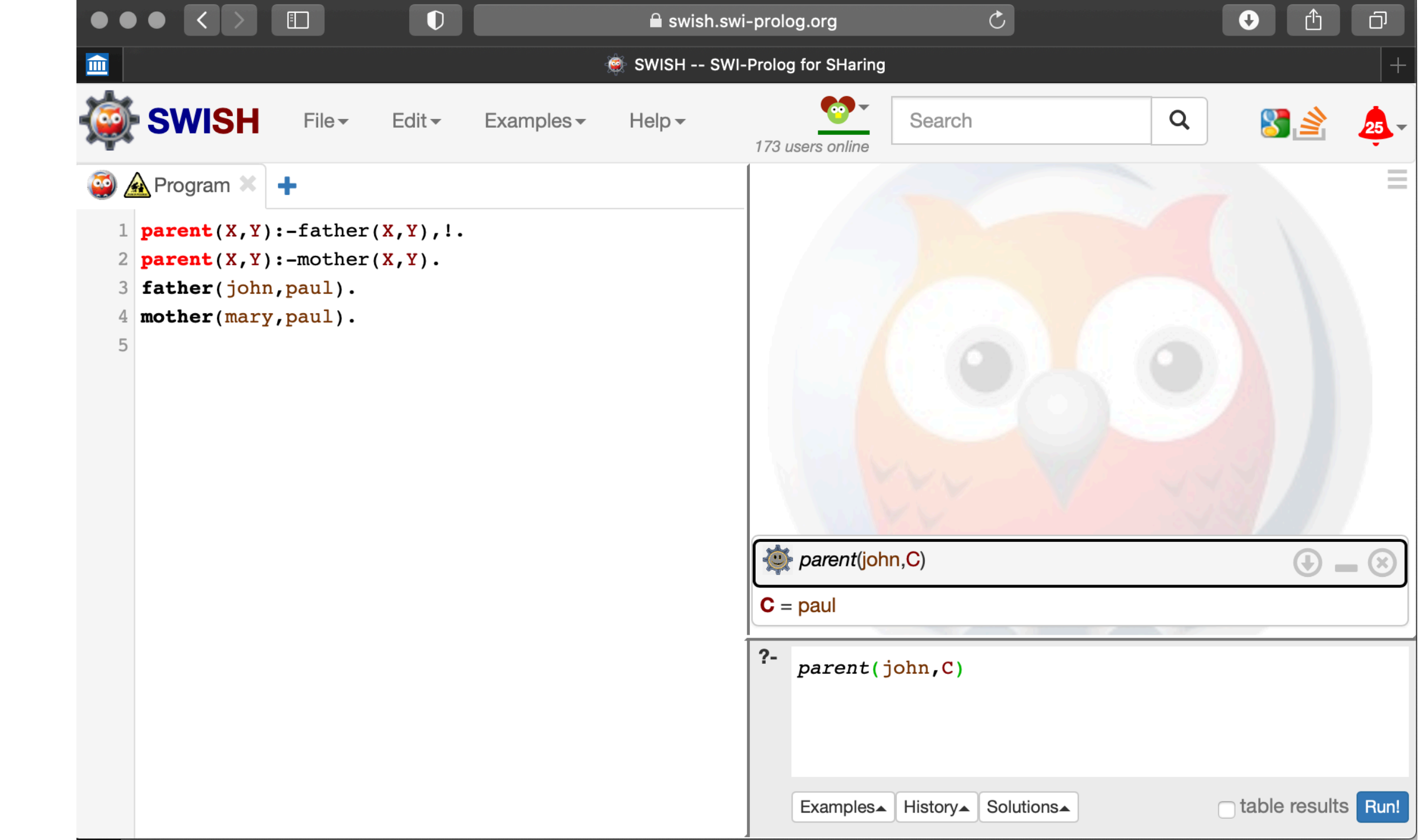


Pruning by means of cut

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

CUT

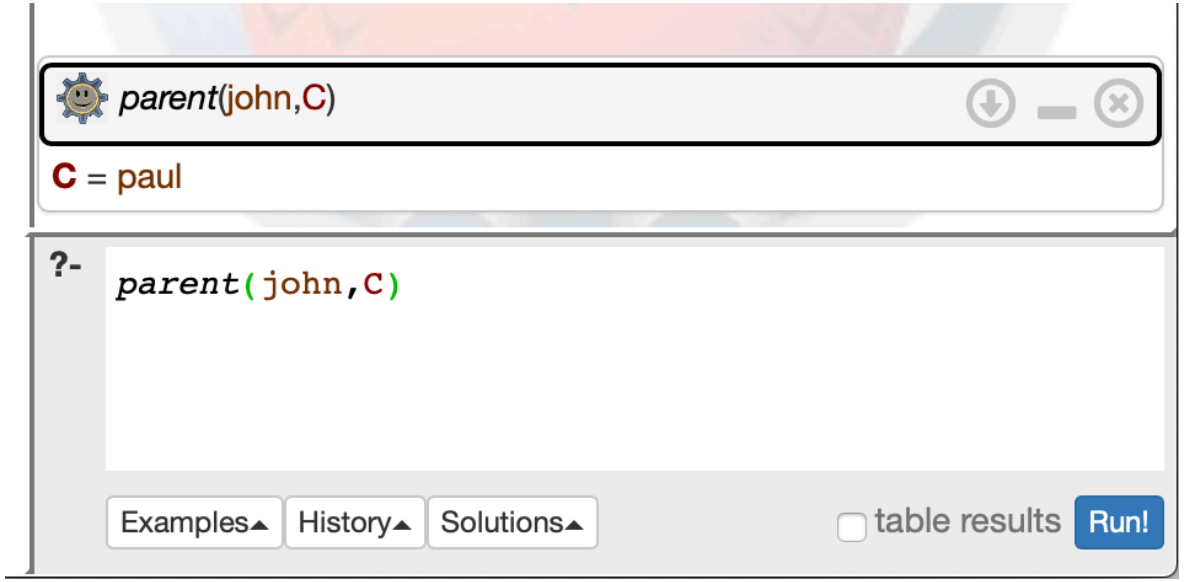



swish.swi-prolog.org
SWISH -- SWI-Prolog for SHaring
SWISH
File
Edit
Examples
Help
173 users online
Search
25
Program
1 parent(X,Y):-father(X,Y),!.
2 parent(X,Y):-mother(X,Y).
3 father(john,paul).
4 mother(mary,paul).
5
parent(john,C)
C = paul
?-
parent(john,C)
Examples
History
Solutions
table results
Run!

Compare

Without cut

With cut



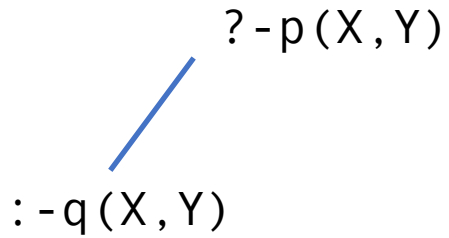
The effect of cut

```
?-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).
```

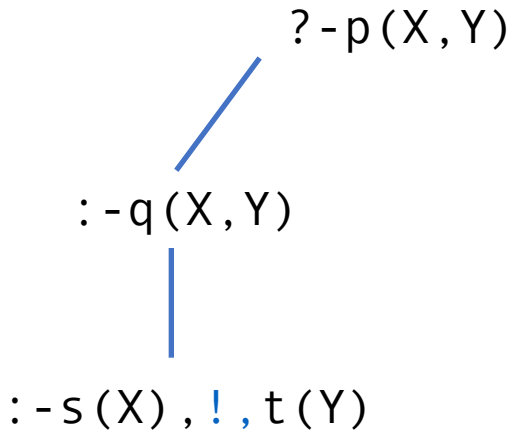
The effect of cut

```
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).
```



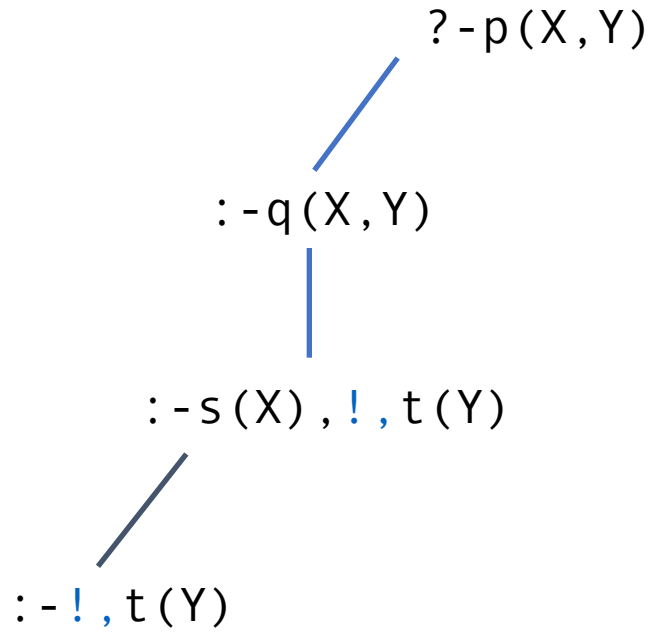
The effect of cut

```
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).
```



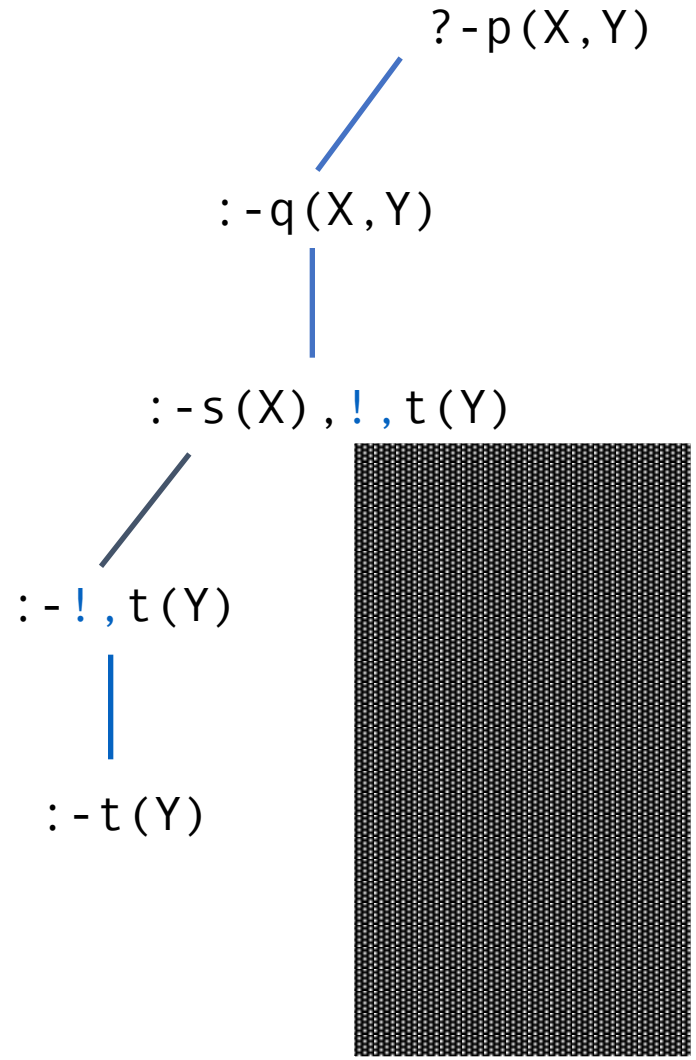
The effect of cut

```
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).
```



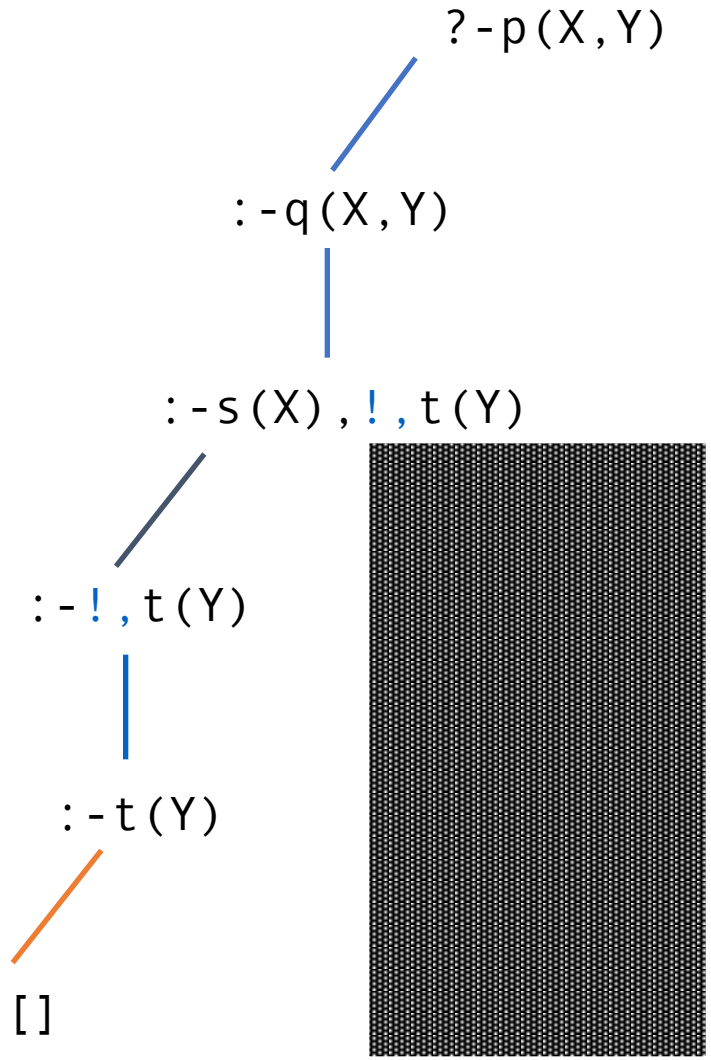
The effect of cut

```
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).
```



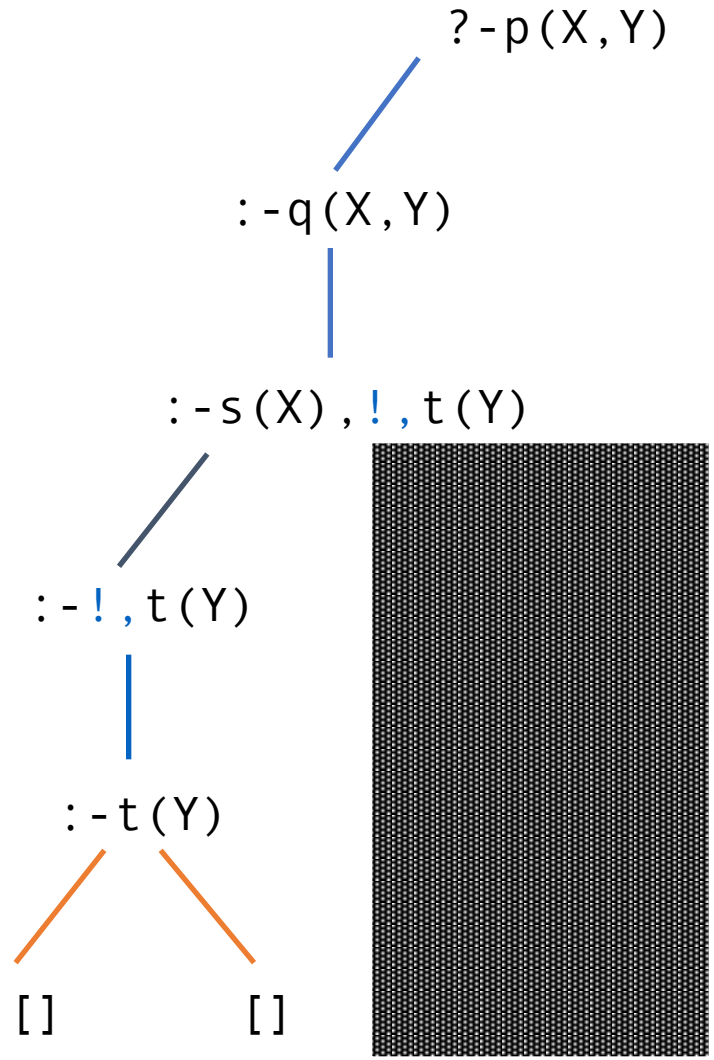
The effect of cut

```
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).
```



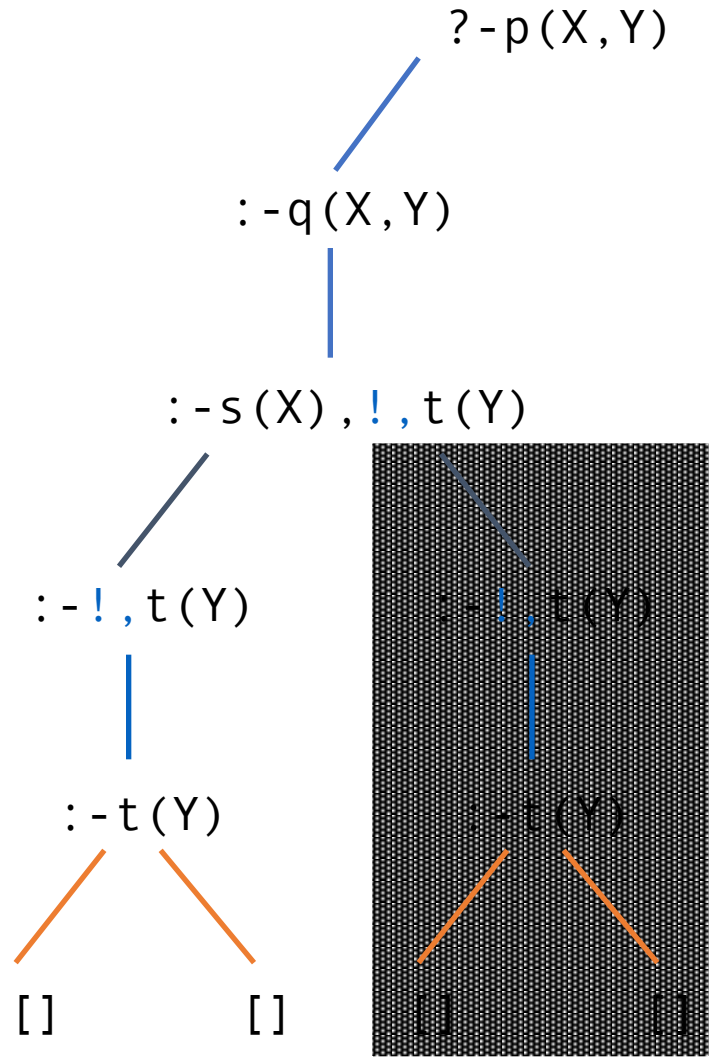
The effect of cut

```
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).
```



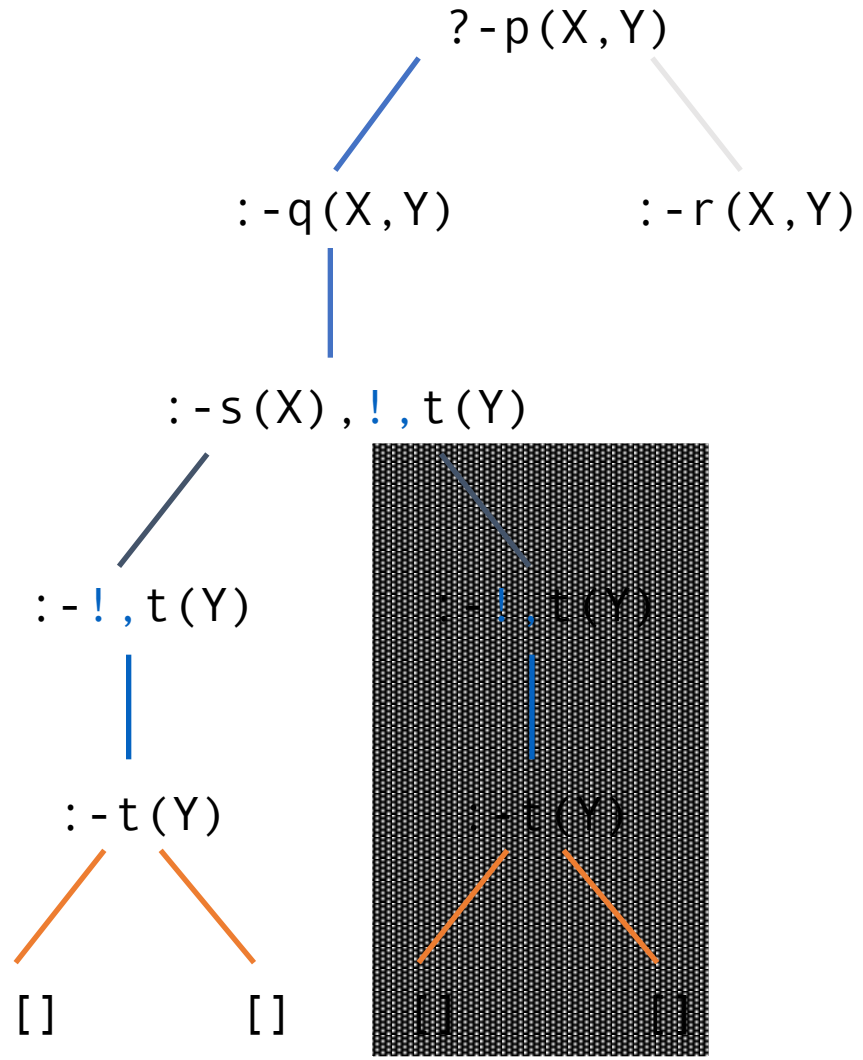
The effect of cut

```
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).
```



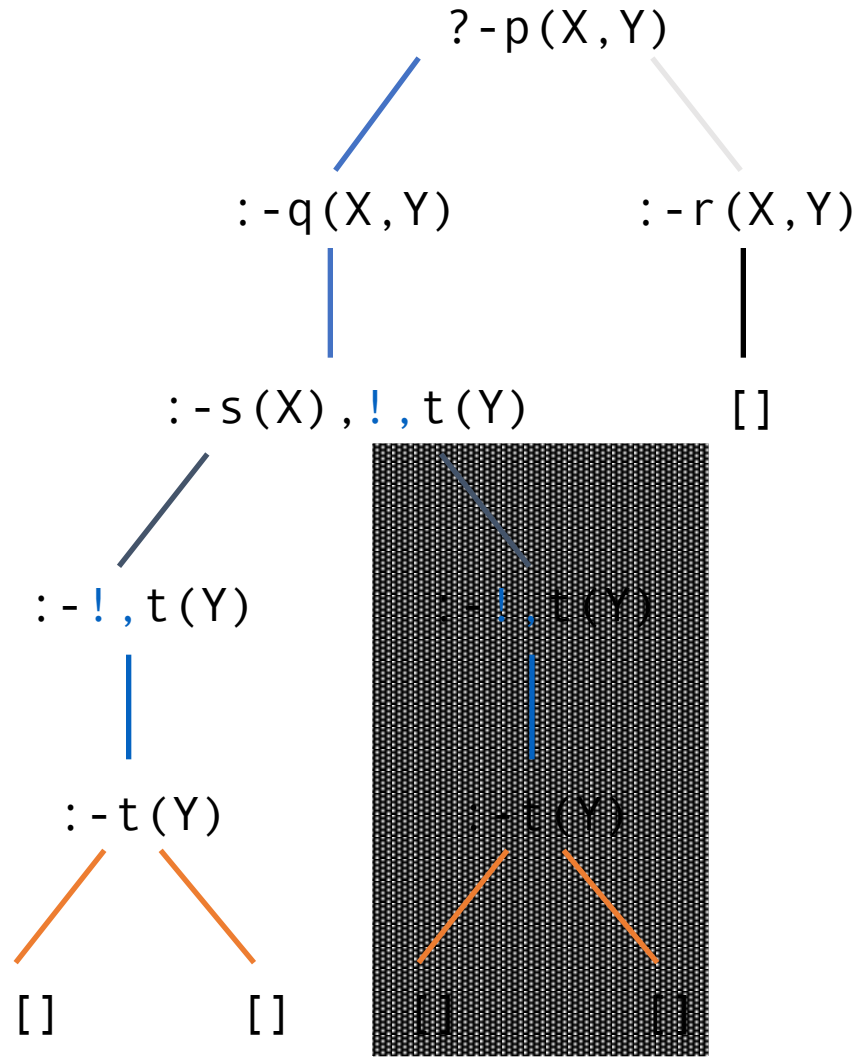
The effect of cut

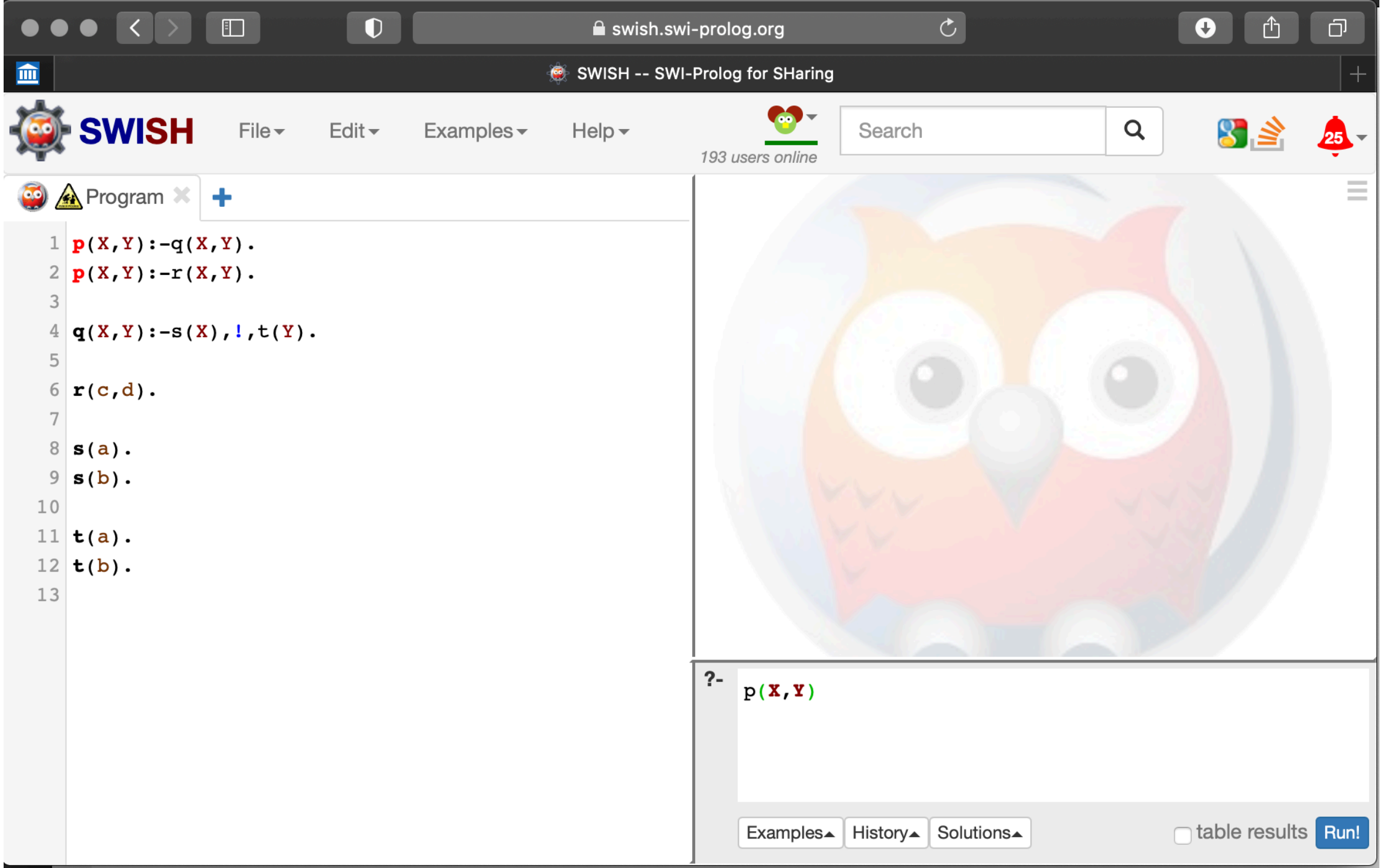
```
p(X,Y):-q(X,Y).
p(X,Y):-r(X,Y).

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

r(c,d).

s(a).
s(b).

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

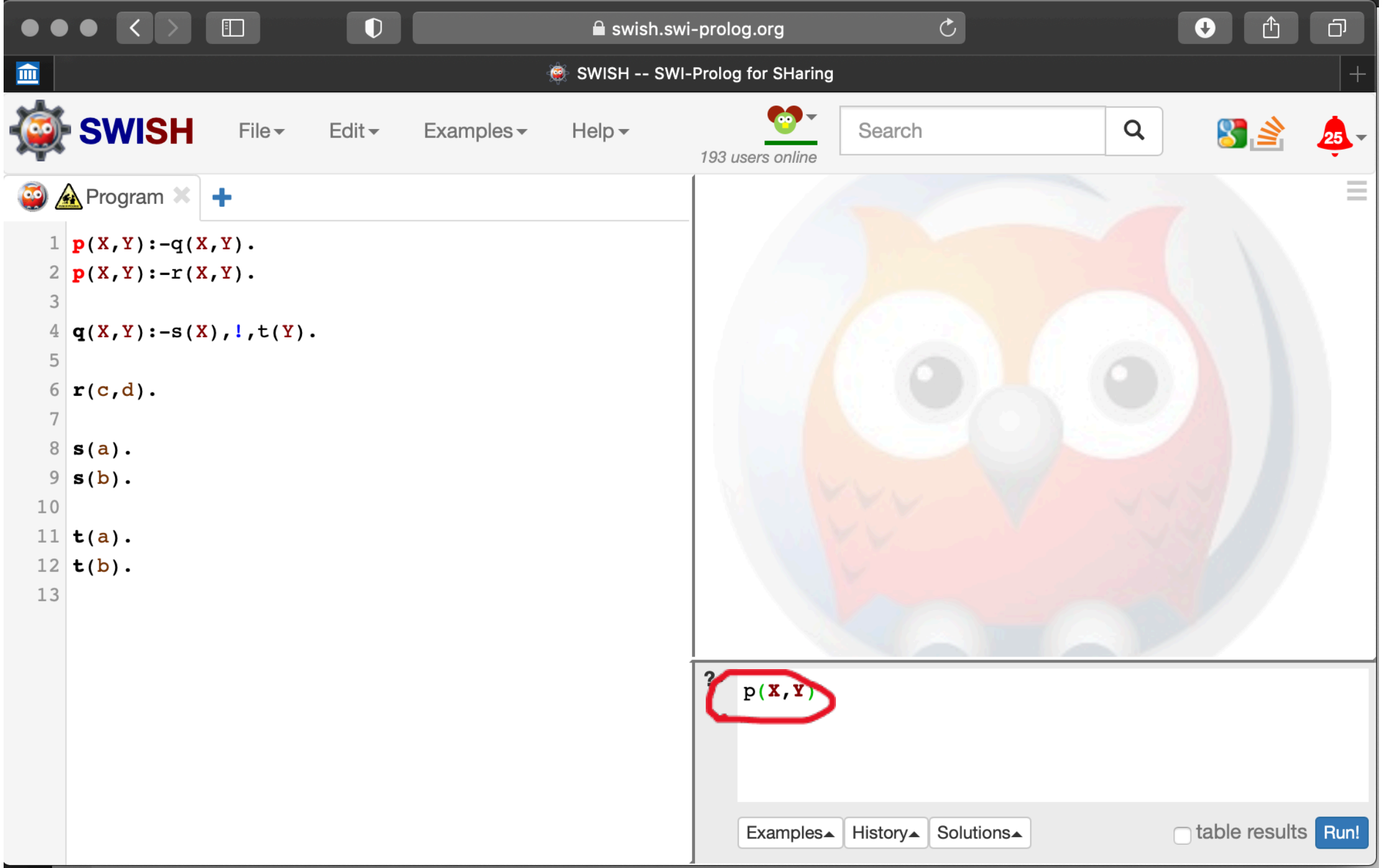


The effect of cut

```
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
SWISH
File
Edit
Examples
Help
193 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(X,Y)
Examples
History
Solutions
table results
Run!


swish.swi-prolog.org
SWISH -- SWI-Prolog for SHaring
SWISH
File
Edit
Examples
Help
193 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(X,Y)
Examples
History
Solutions
table results
Run!

swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

SWISH File Edit Examples Help

192 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(X,Y)

X = Y, Y = a

Next 10 100 1,000 Stop

?- p(X,Y)

Examples History Solutions table results Run!

swish.swi-prolog.org

SWIH -- SWI-Prolog for SHaring

SWISH File Edit Examples Help

192 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

Next | 10 | 100 | 1,000 | Stop

?- p(X,Y)

Examples | History | Solutions | table results | Run!

swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

SWISH 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(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

SWISH 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
```

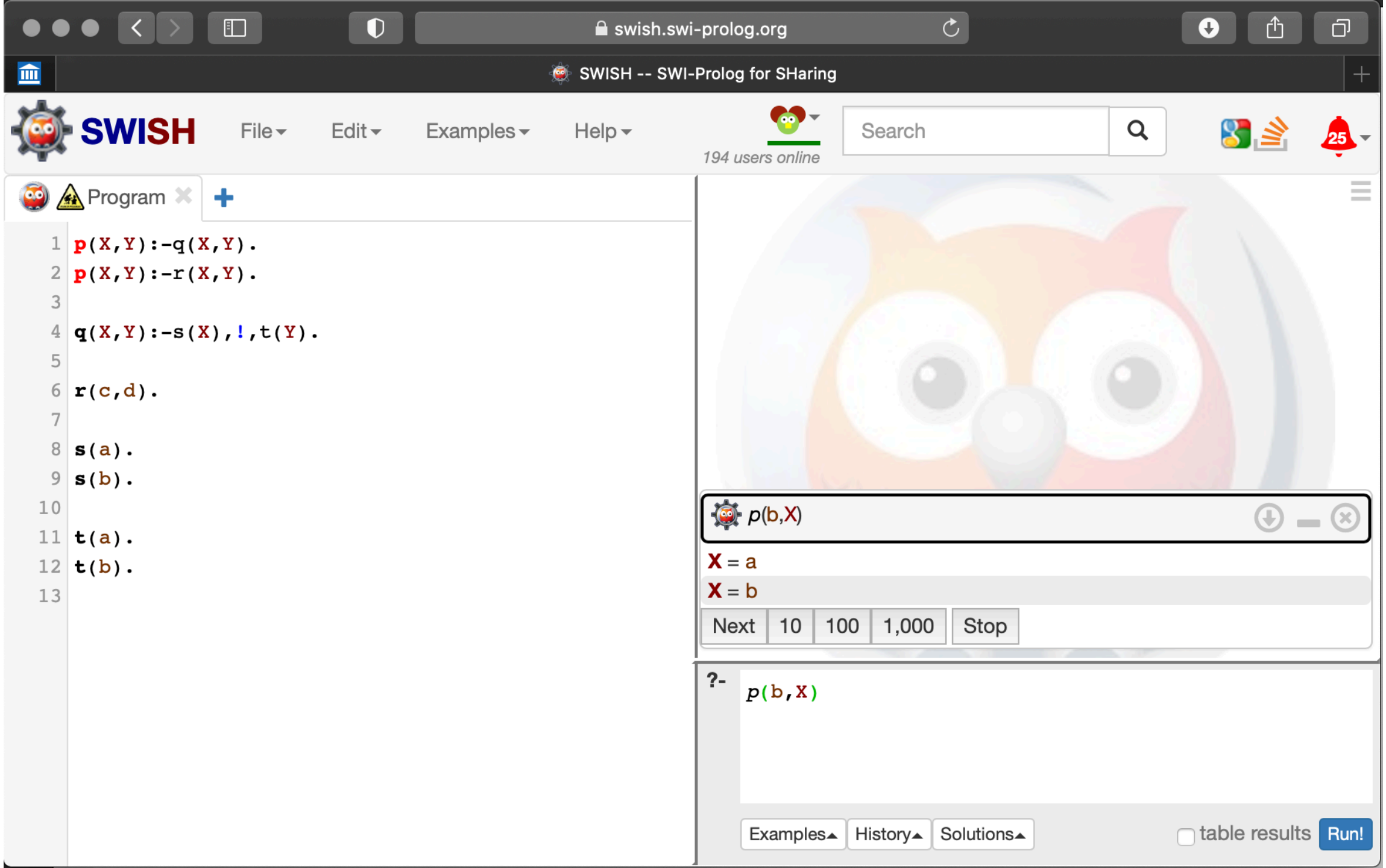
p(b,X)

X = a

Next 10 100 1,000 Stop

?- p(b,X)

Examples History Solutions table results Run!


swish.swi-prolog.org
SWISH -- SWI-Prolog for SHaring
SWISH
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!

swish.swi-prolog.org

SWISH -- SWI-Prolog for SHaring

SWISH 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

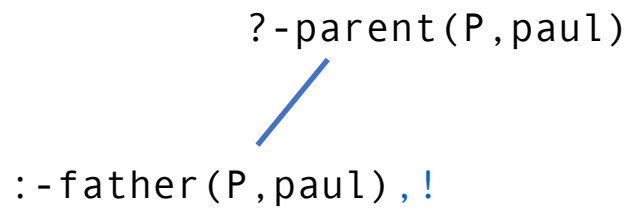
Pruning away success branches

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

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

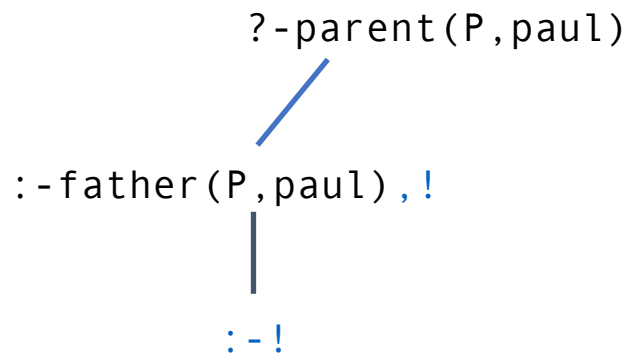
Pruning away success branches

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



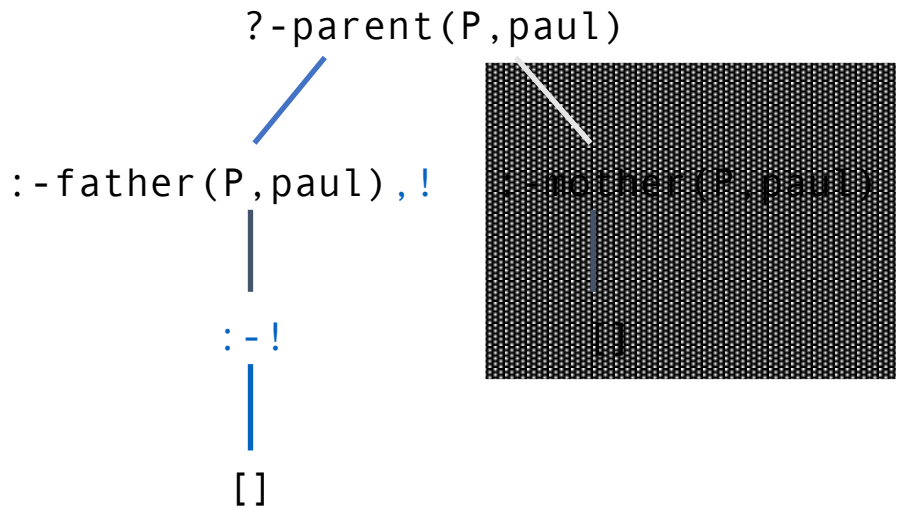
Pruning away success branches

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



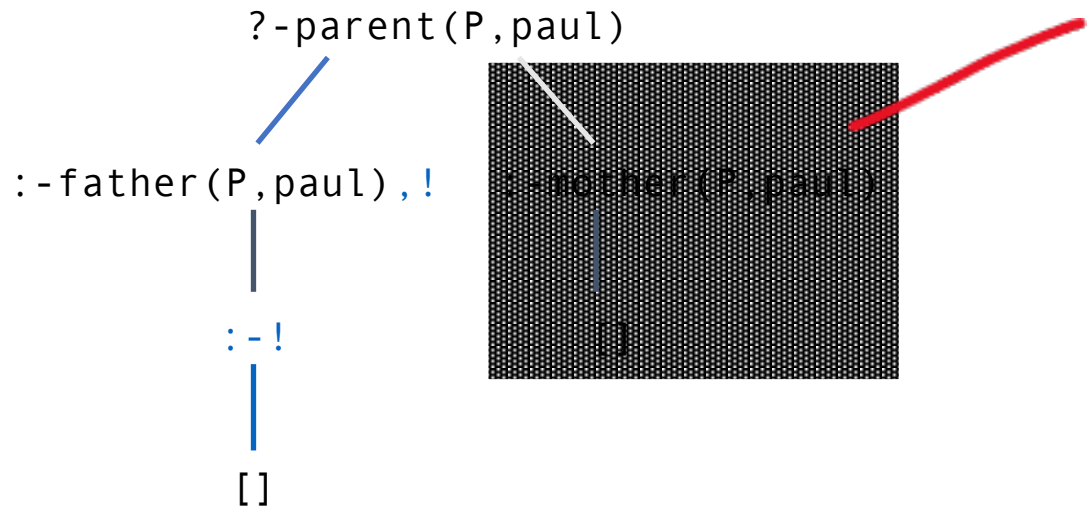
Pruning away success branches

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

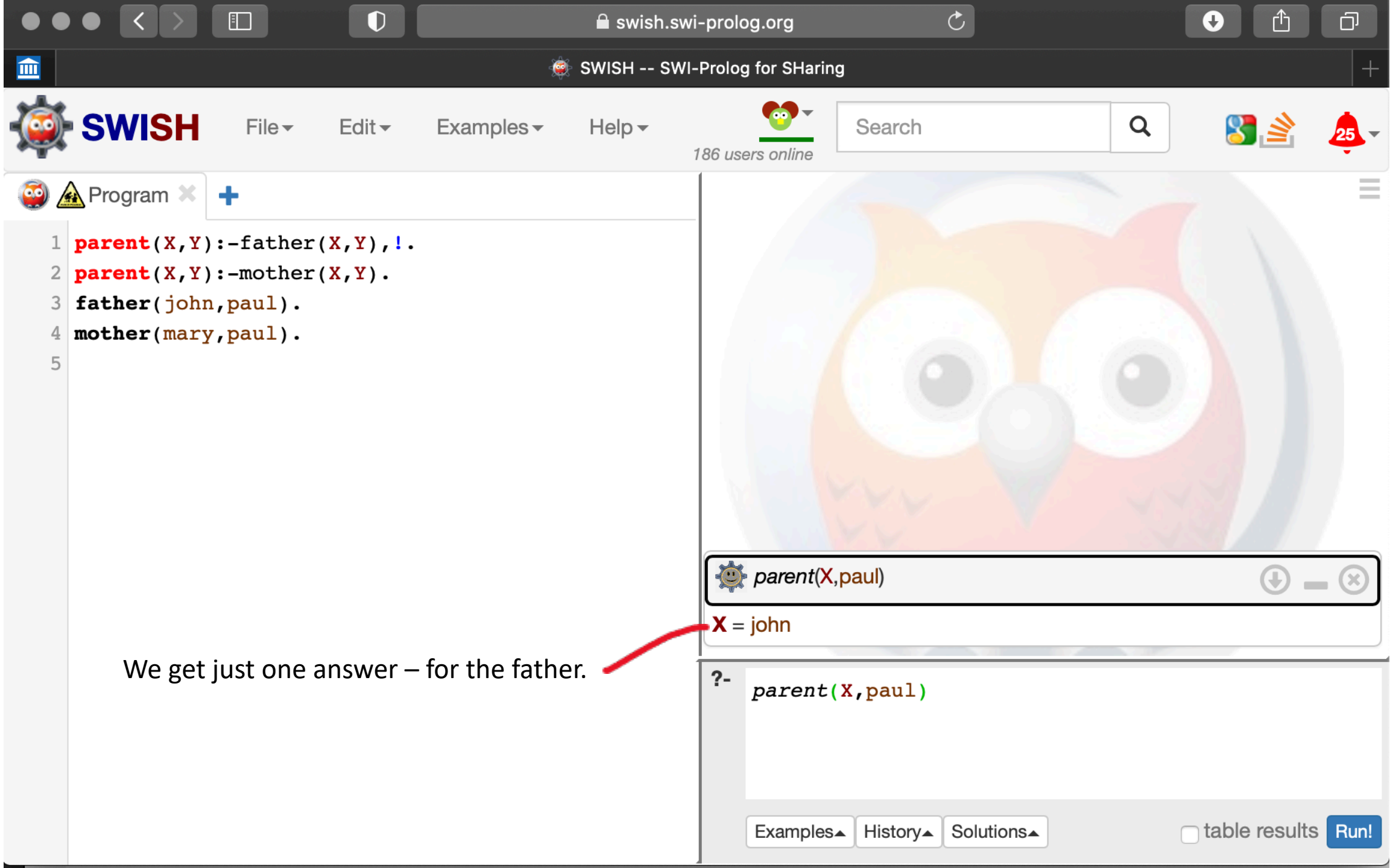


Pruning away success branches

```
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.



We get just one answer – for the father.

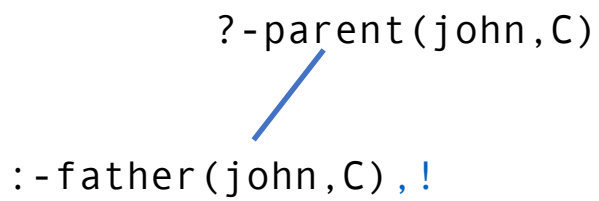
Pruning away success branches

```
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)
```

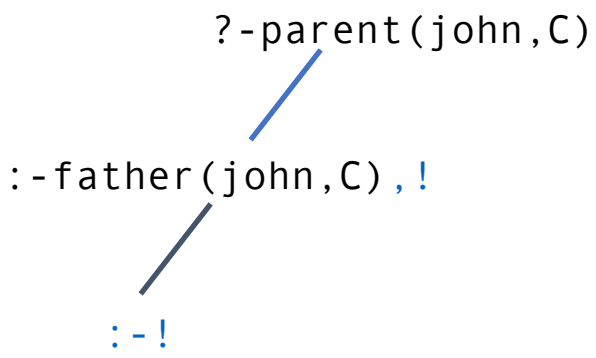
Pruning away success branches

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



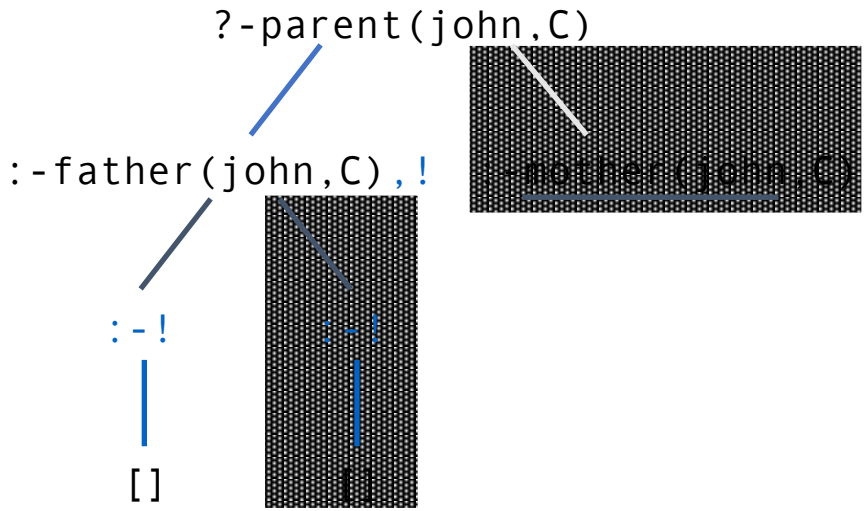
Pruning away success branches

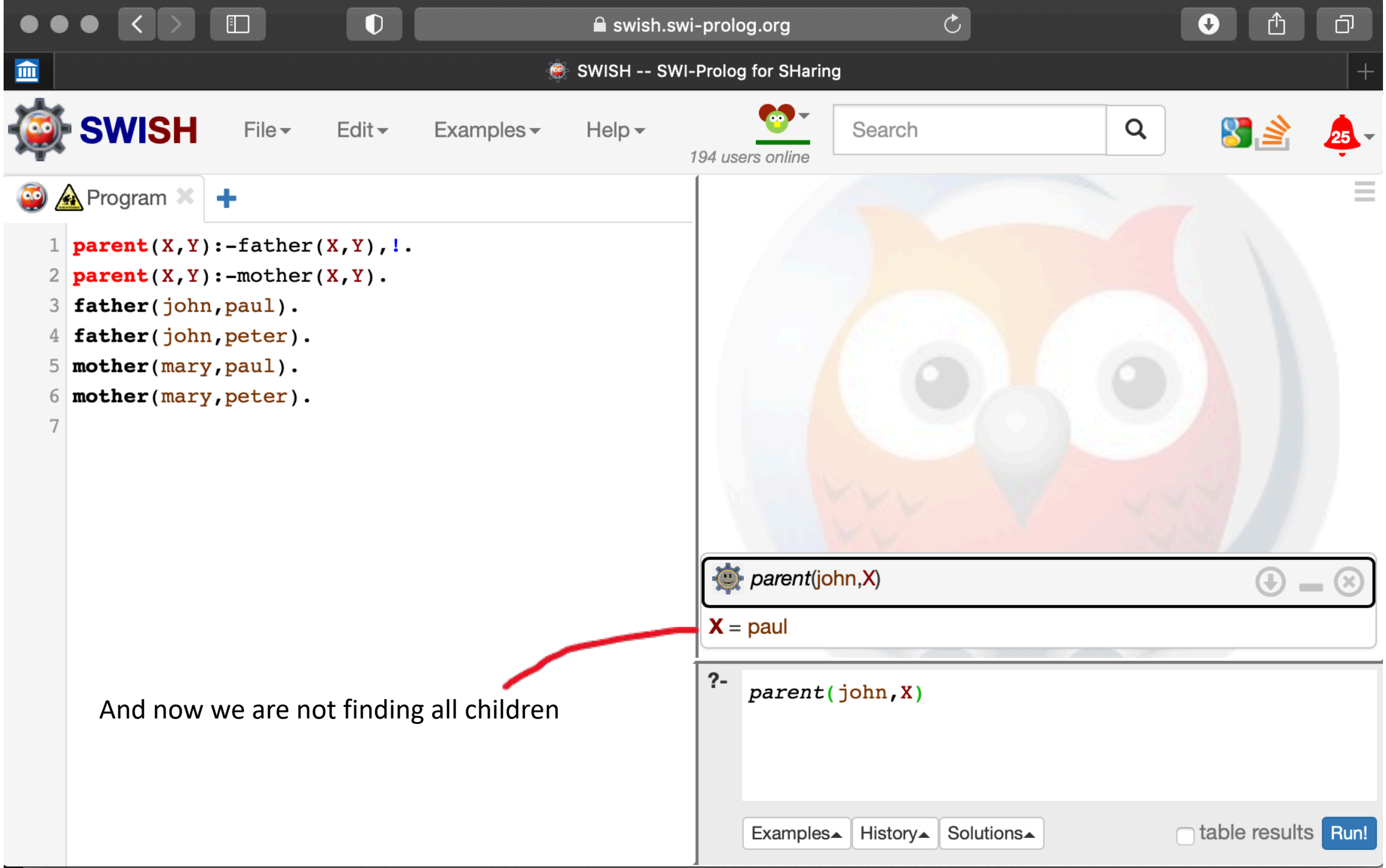
```
parent(X,Y):-father(X,Y),!.
parent(X,Y):-mother(X,Y).
father(john,paul).
father(john,peter).
mother(mary,paul).
mother(mary,peter).
```



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
SWISH
File
Edit
Examples
Help
194 users online
Search
25
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)
Examples
History
Solutions
table results
Run!
And now we are not finding all children

So be careful....

And now… Negation as Failure

Cut vs not

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

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

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

Cut vs not

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

```
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 vs not

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

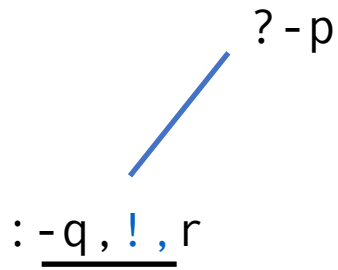
```
?-p
```

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

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

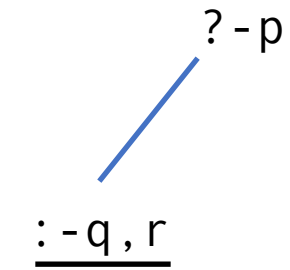
```
?-p
```

Cut vs not

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

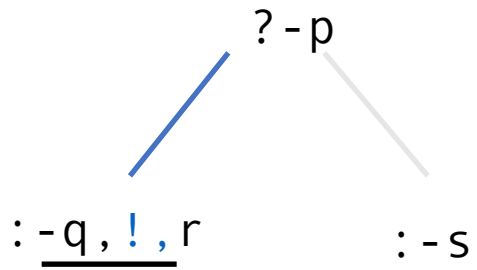



```
p:-q,r.
p:-not(q),s.
s.

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

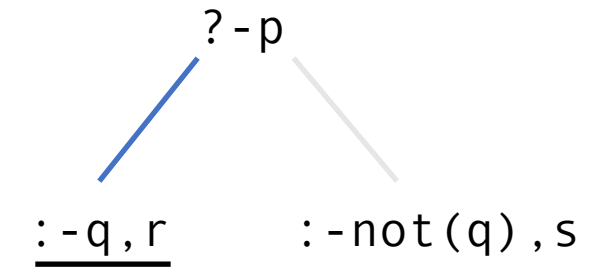


Cut vs not

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

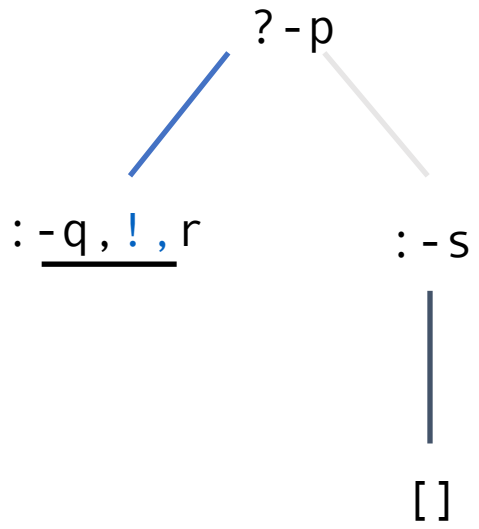



```
p:-q,r.
p:-not(q),s.
s.

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

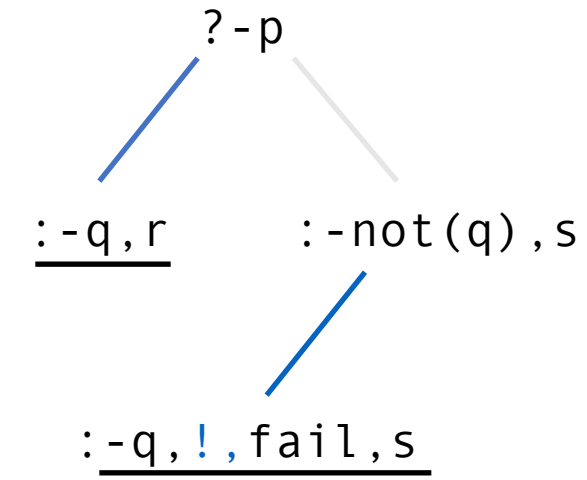


Cut vs not

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

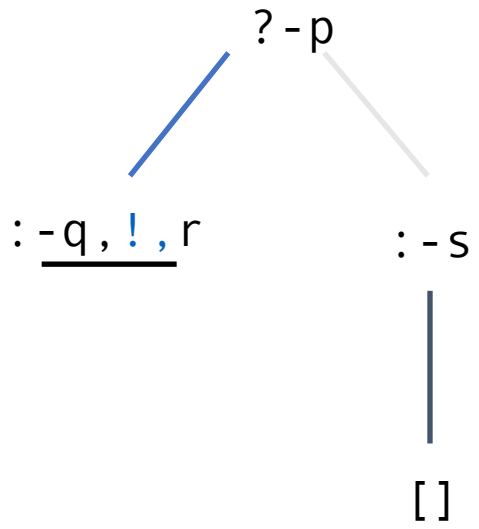



```
p:-q,r.
p:-not(q),s.
s.

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

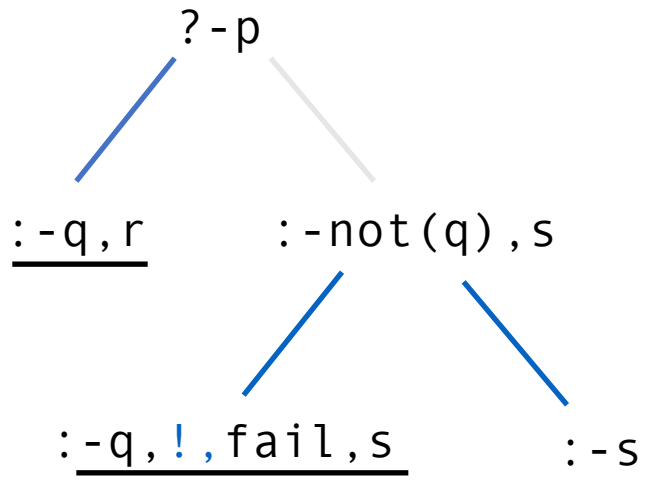


Cut vs not

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

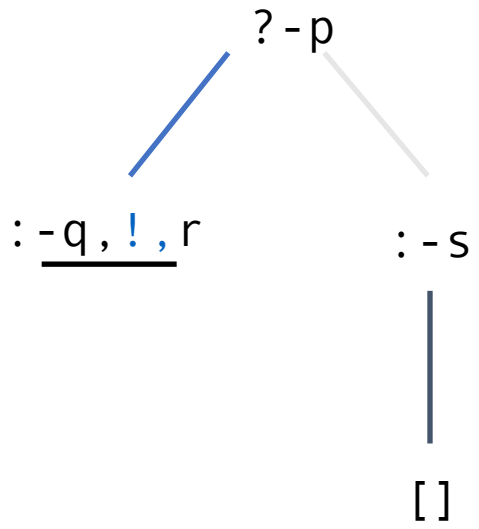



```
p:-q,r.
p:-not(q),s.
s.

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

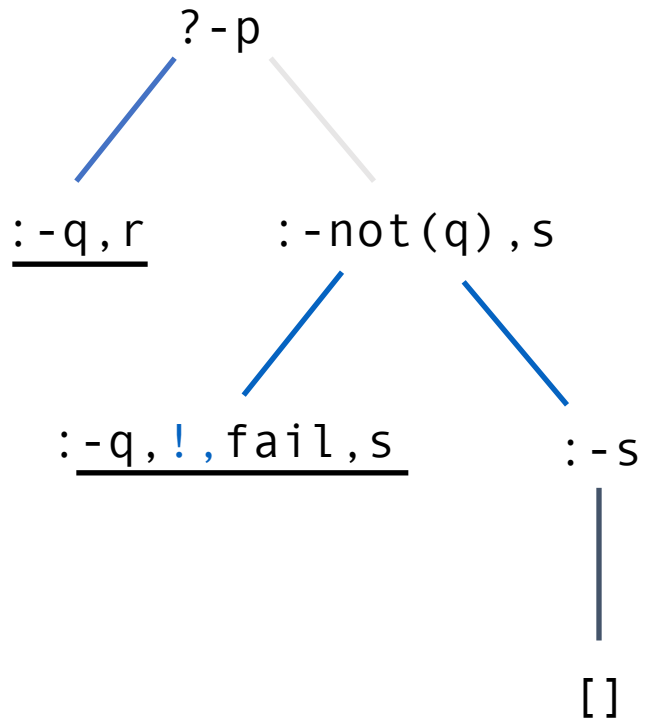


Cut vs not

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




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

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



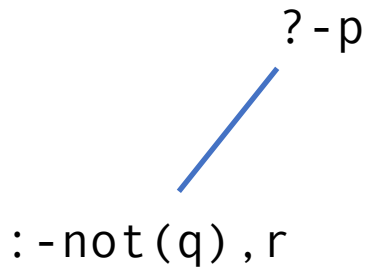
An example: How it works when `:-not(q)` fails

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

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

```
?-p
```

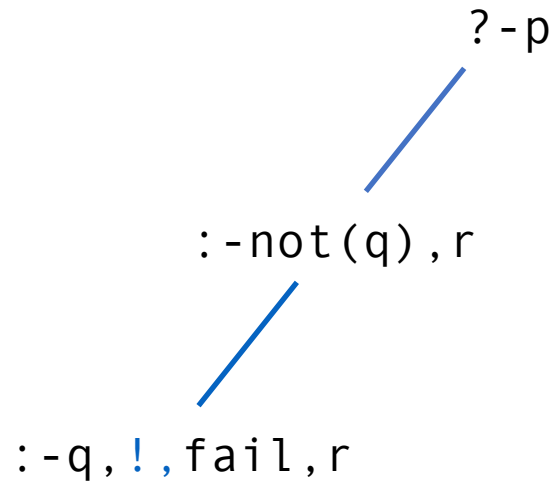
An example: How it works when `:-not(q)` fails

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

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



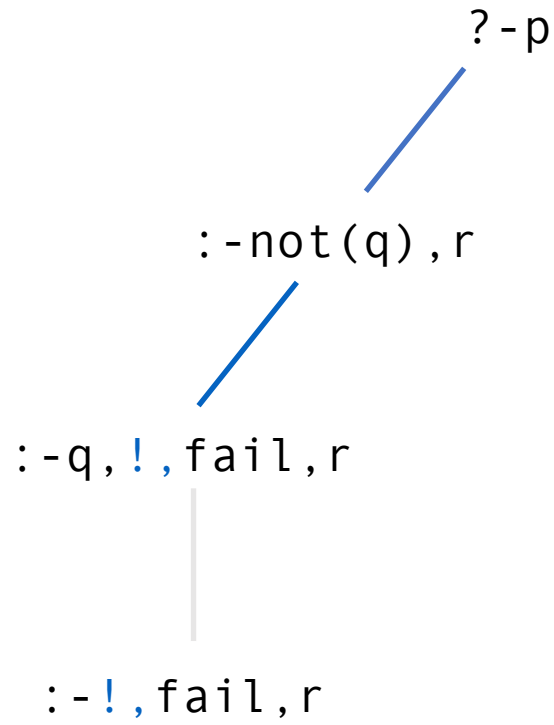
An example: How it works when :-not(q) fails

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

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



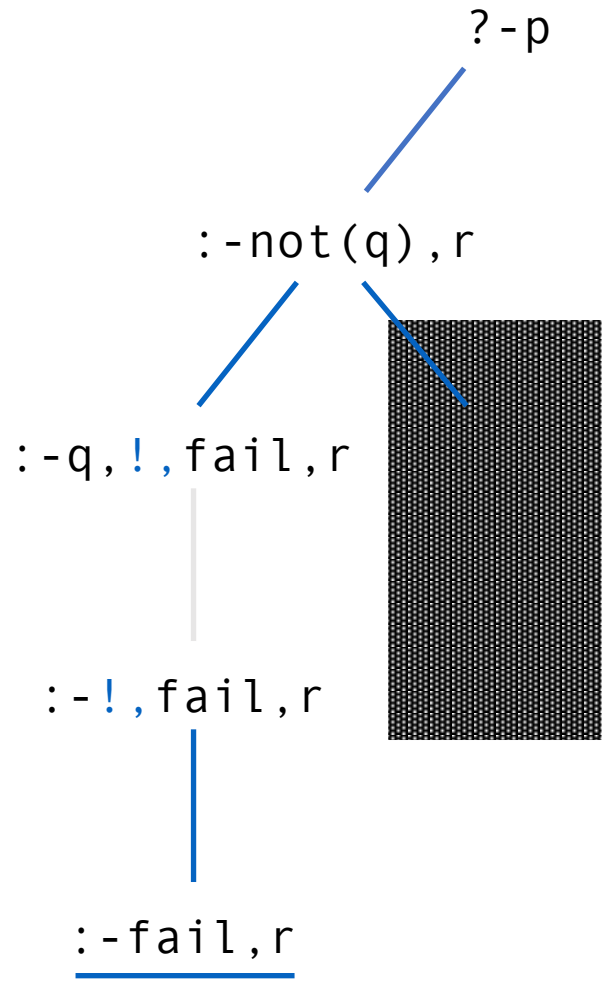
An example: How it works when :-not(q) fails

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

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



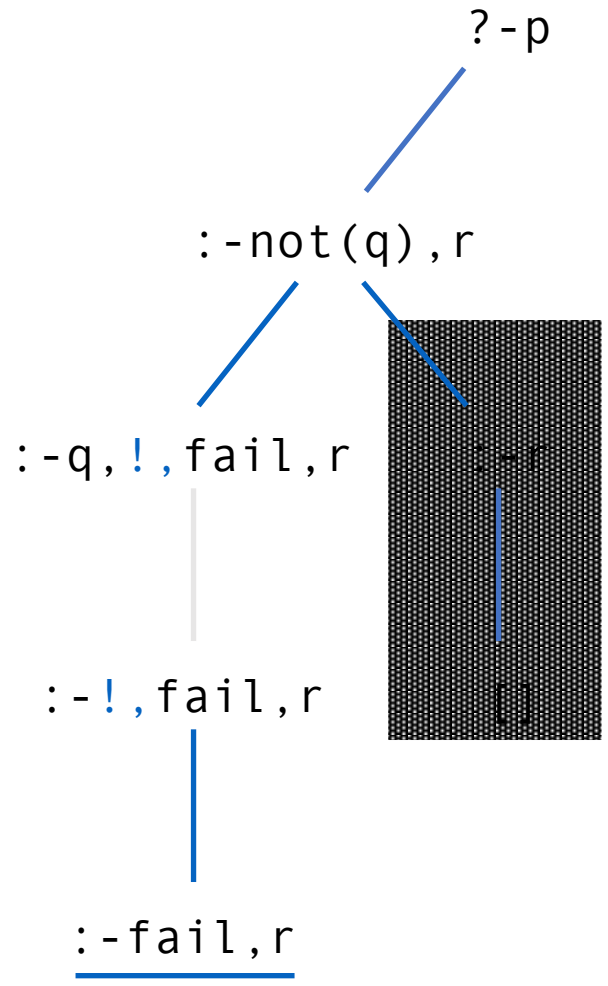
An example: How it works when `:-not(q)` fails

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

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



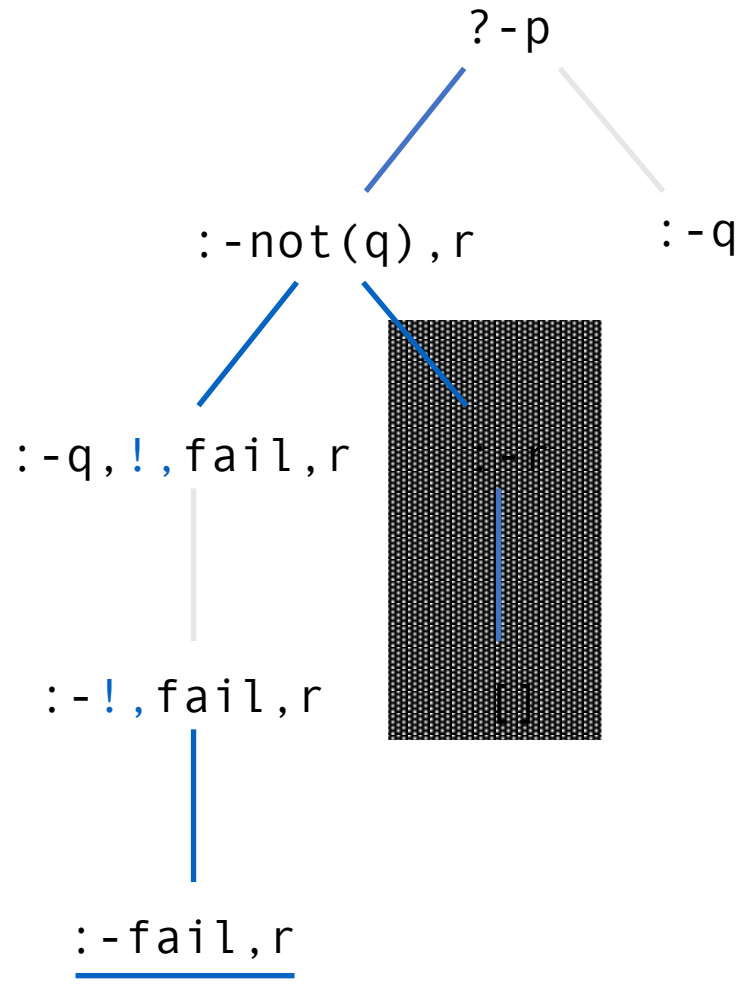
An example: How it works when `:-not(q)` fails

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

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



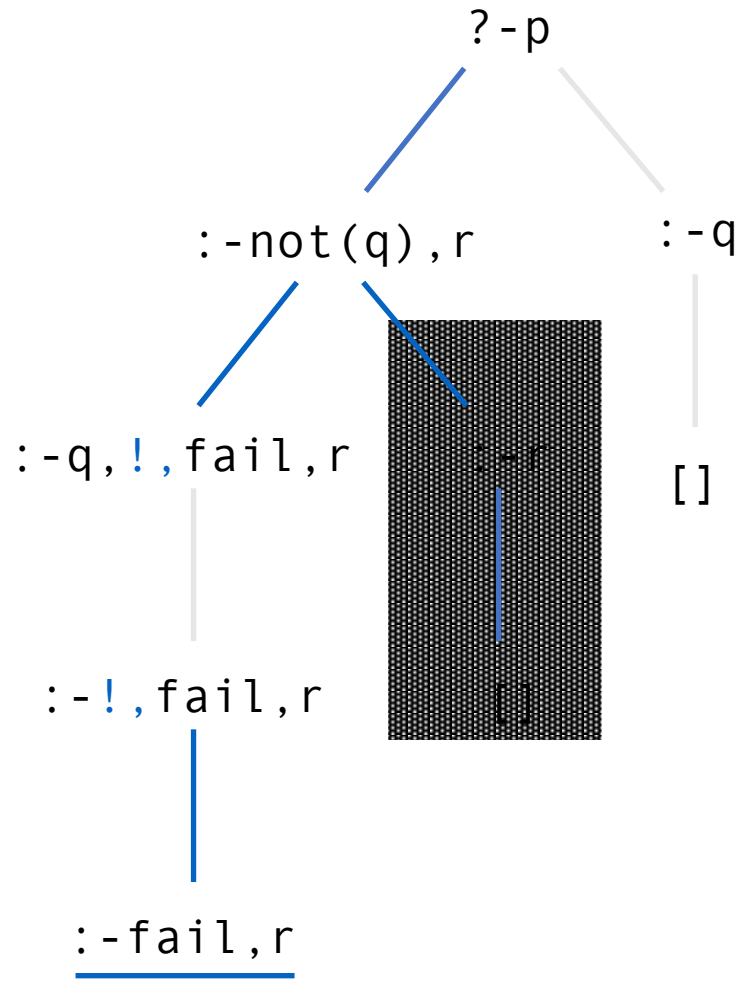
An example: How it works when `:-not(q)` fails

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

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



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

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

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



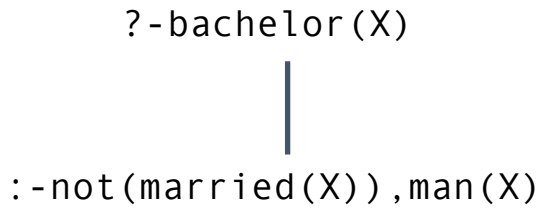
Prolog's `not` is unsound

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

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

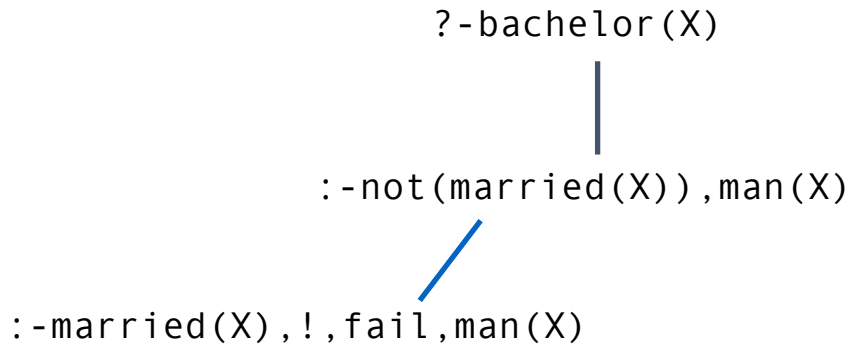
Prolog's `not` is unsound

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



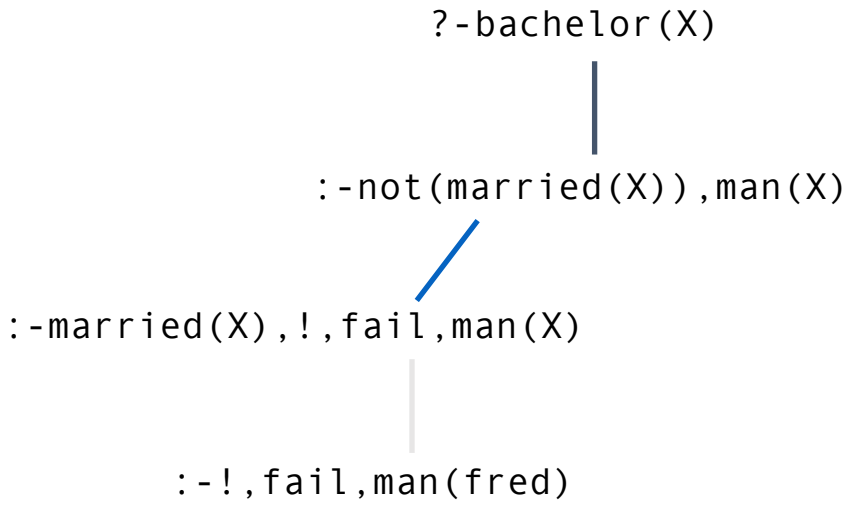
Prolog's `not` is unsound

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



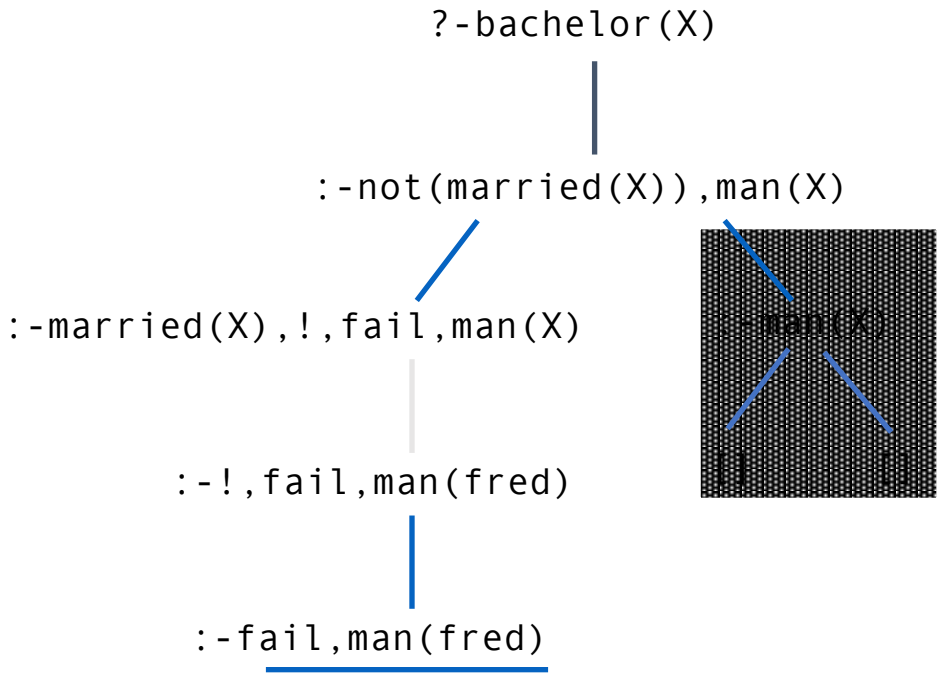
Prolog's not is unsound

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



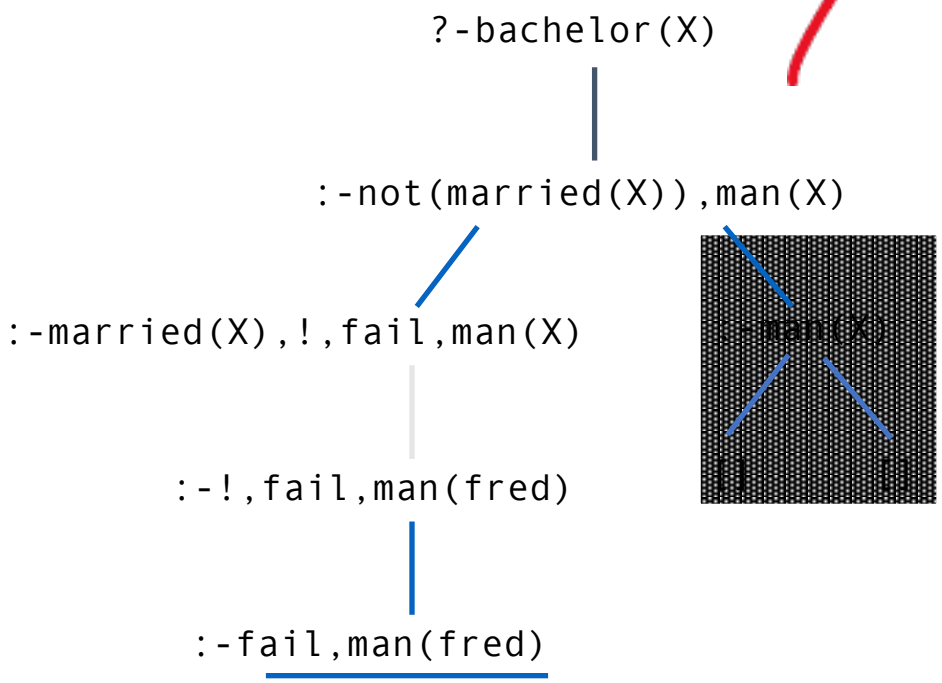
Prolog's not is unsound

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

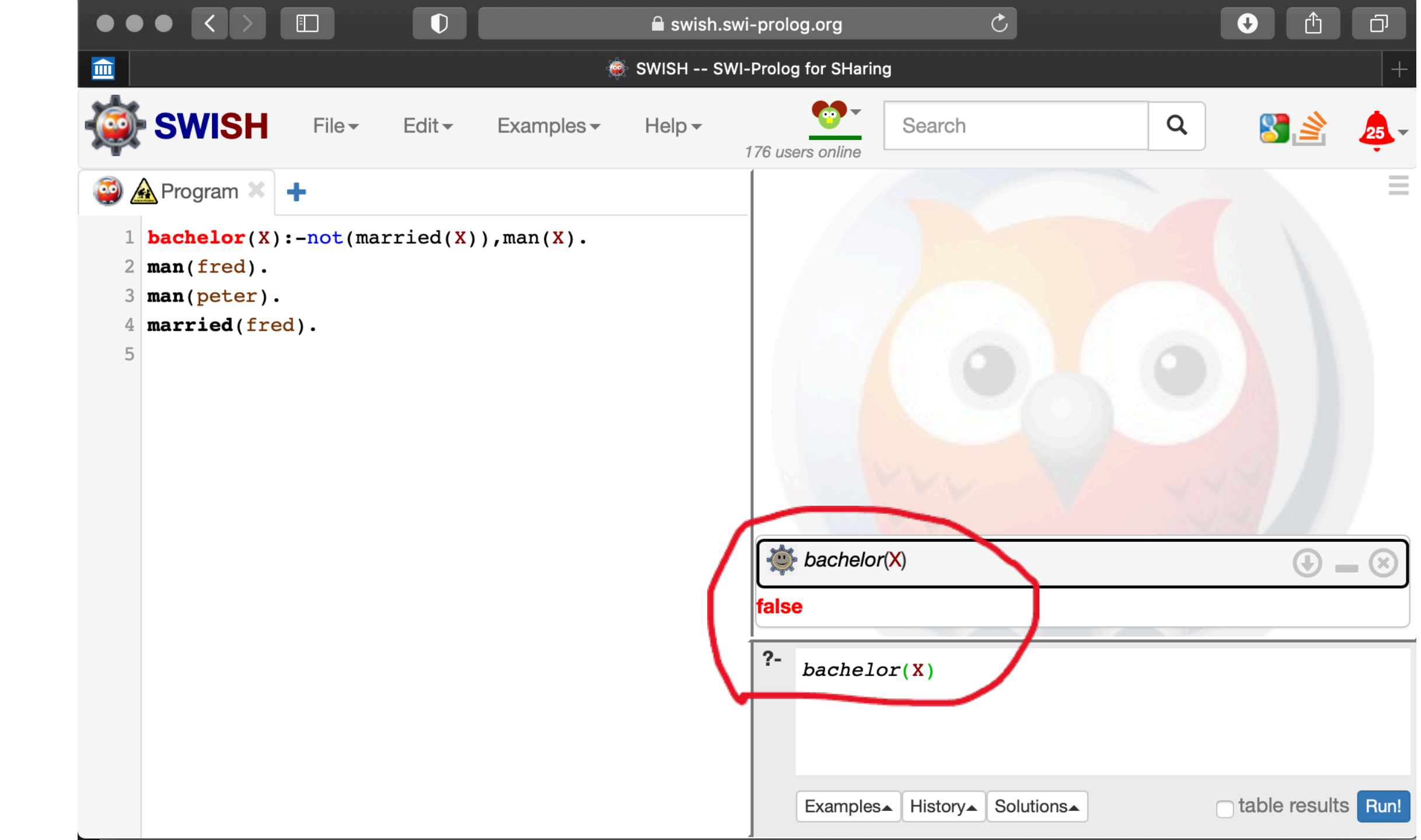


Prolog's `not` is unsound

```
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!


swish.swi-prolog.org
SWIS -- SWI-Prolog for SHaring
SWISH
File
Edit
Examples
Help
176 users online
Search
25
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).
```

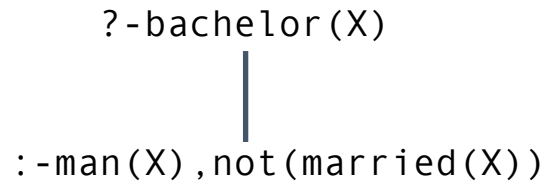
This will "ground" X.

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

Prolog's `not` is unsound – Avoiding the Problem

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

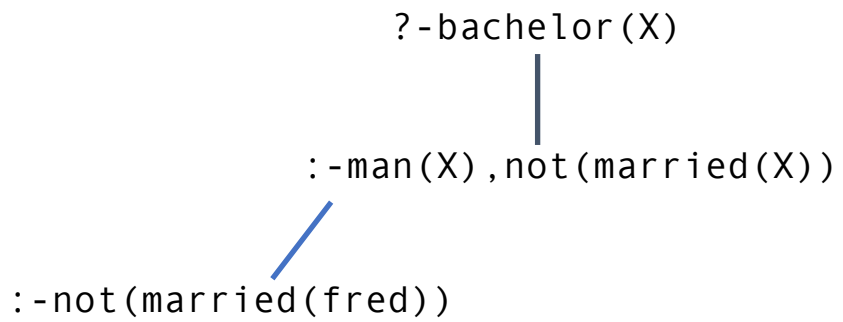
This will "ground" X.



Prolog's `not` is unsound – Avoiding the Problem

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

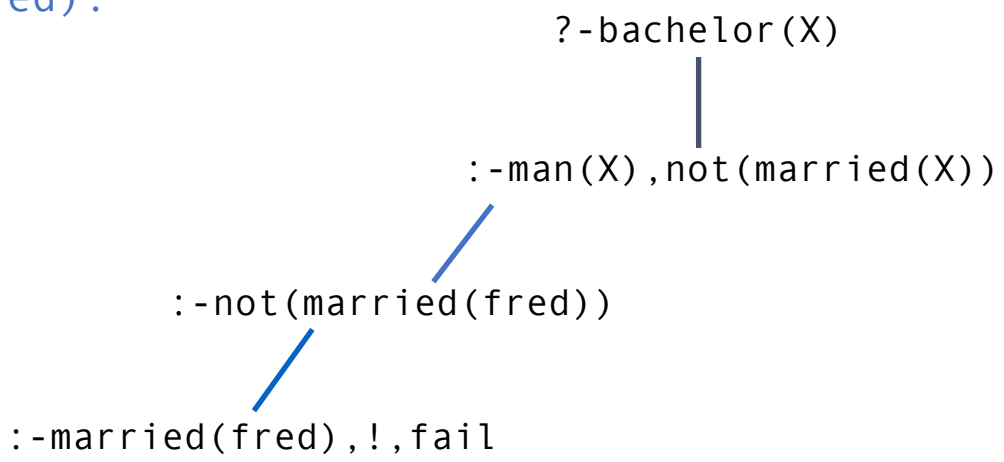
This will "ground" X.



Prolog's `not` is unsound – Avoiding the Problem

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

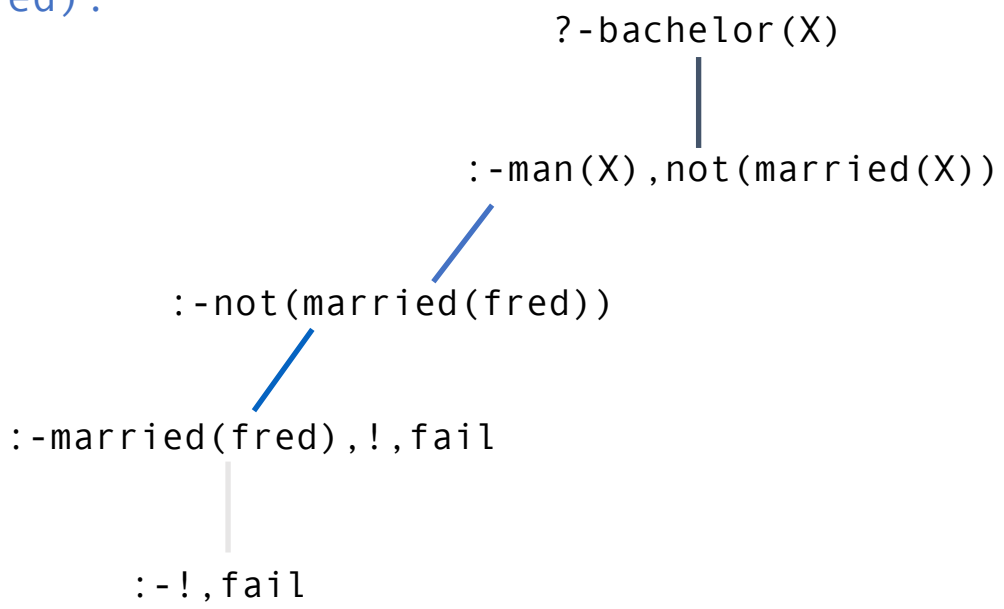
This will "ground" X.



Prolog's `not` is unsound – Avoiding the Problem

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

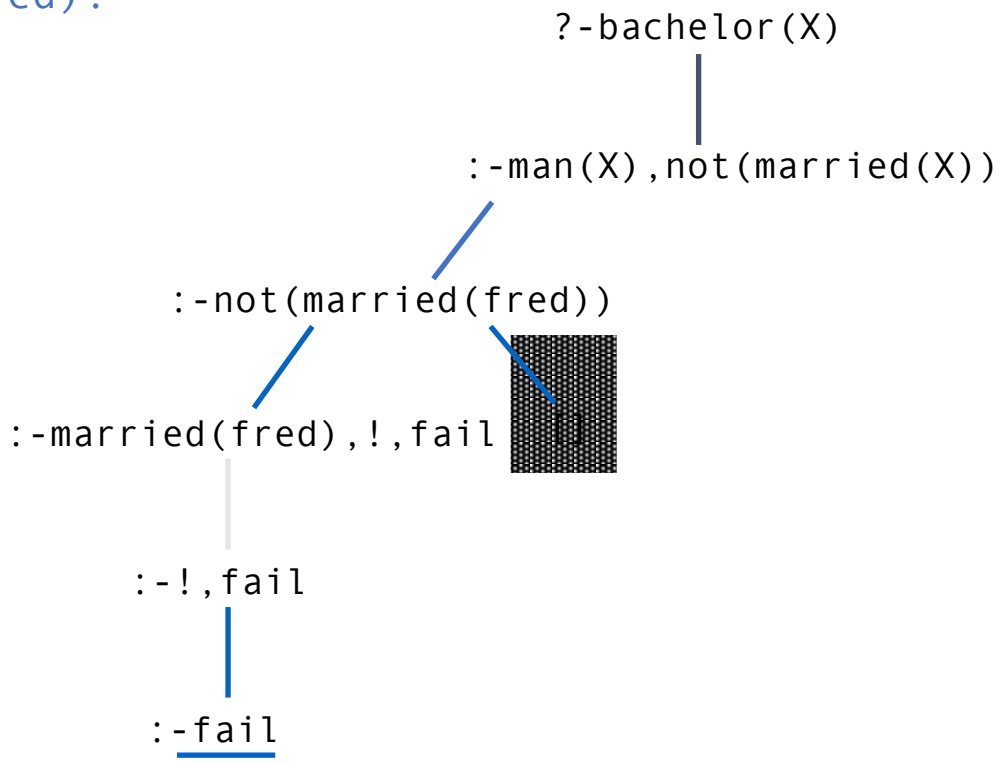
This will "ground" X.



Prolog's `not` is unsound – Avoiding the Problem

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

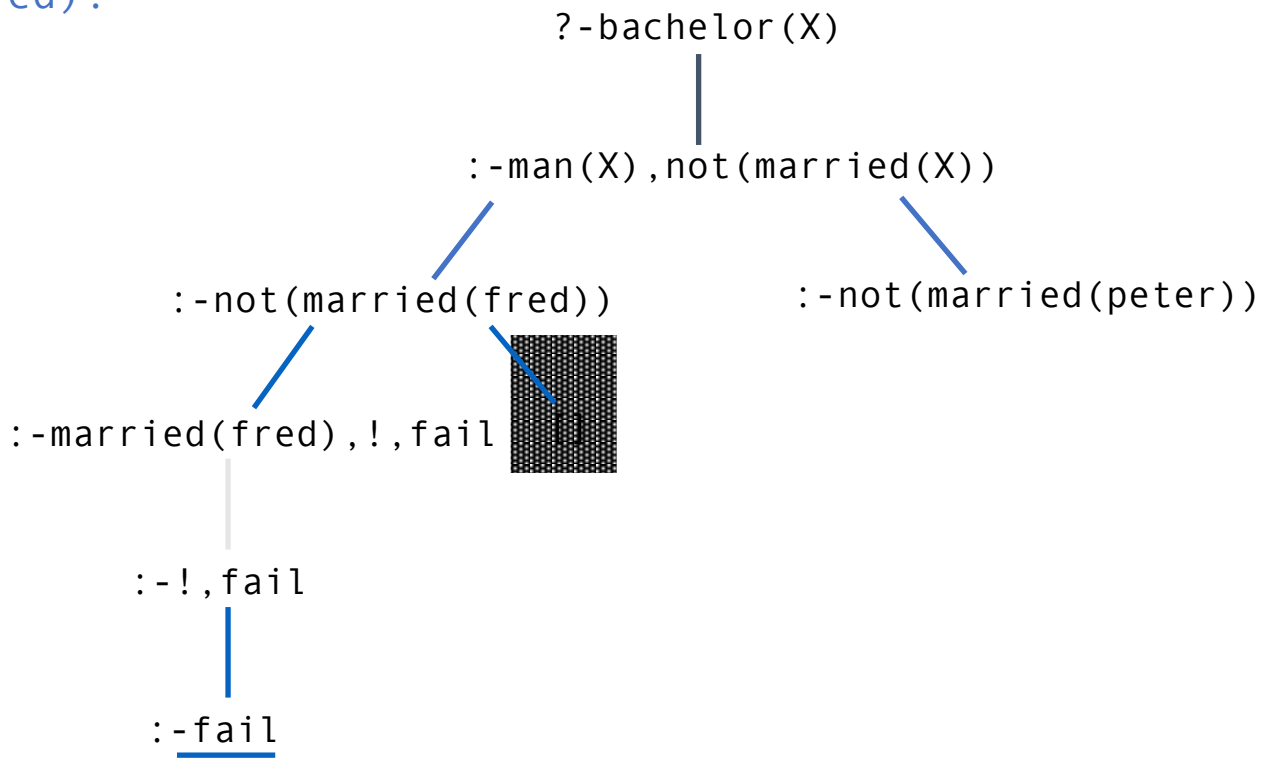
This will "ground" X.



Prolog's `not` is unsound – Avoiding the Problem

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

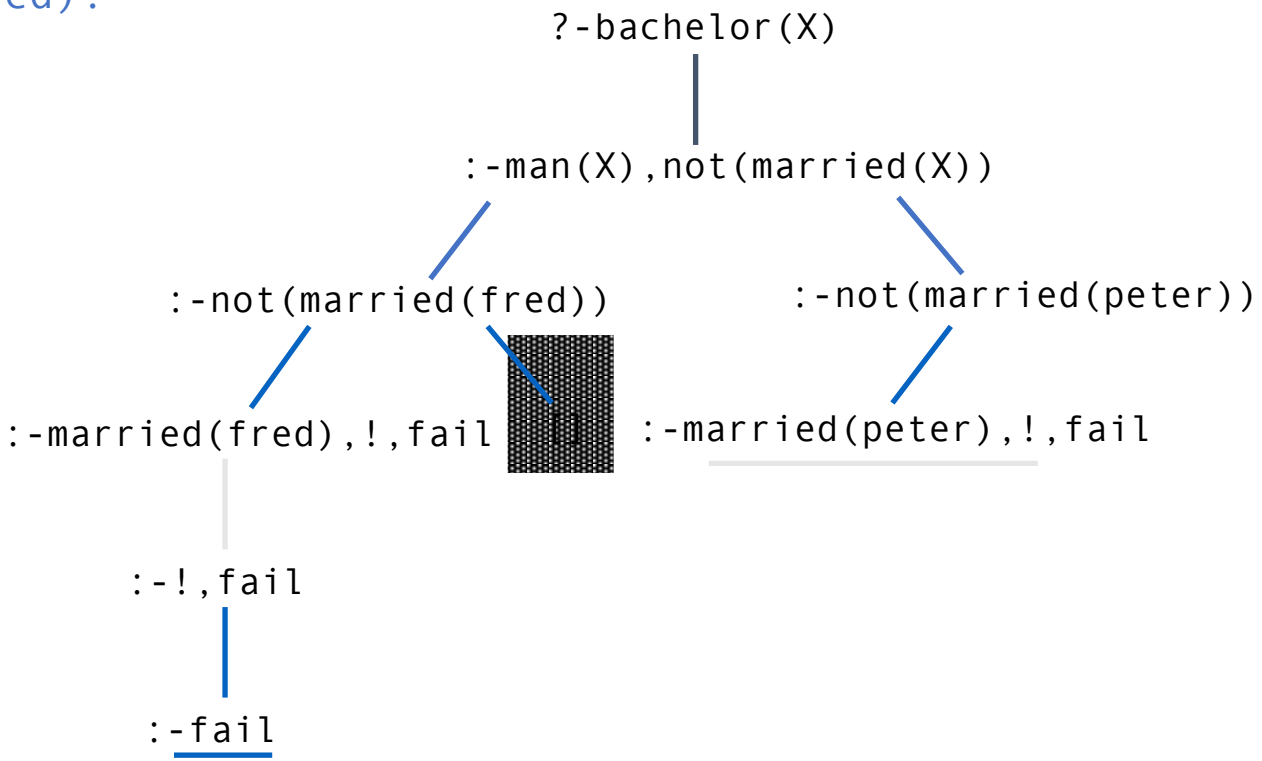
This will "ground" X.



Prolog's `not` is unsound – Avoiding the Problem

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

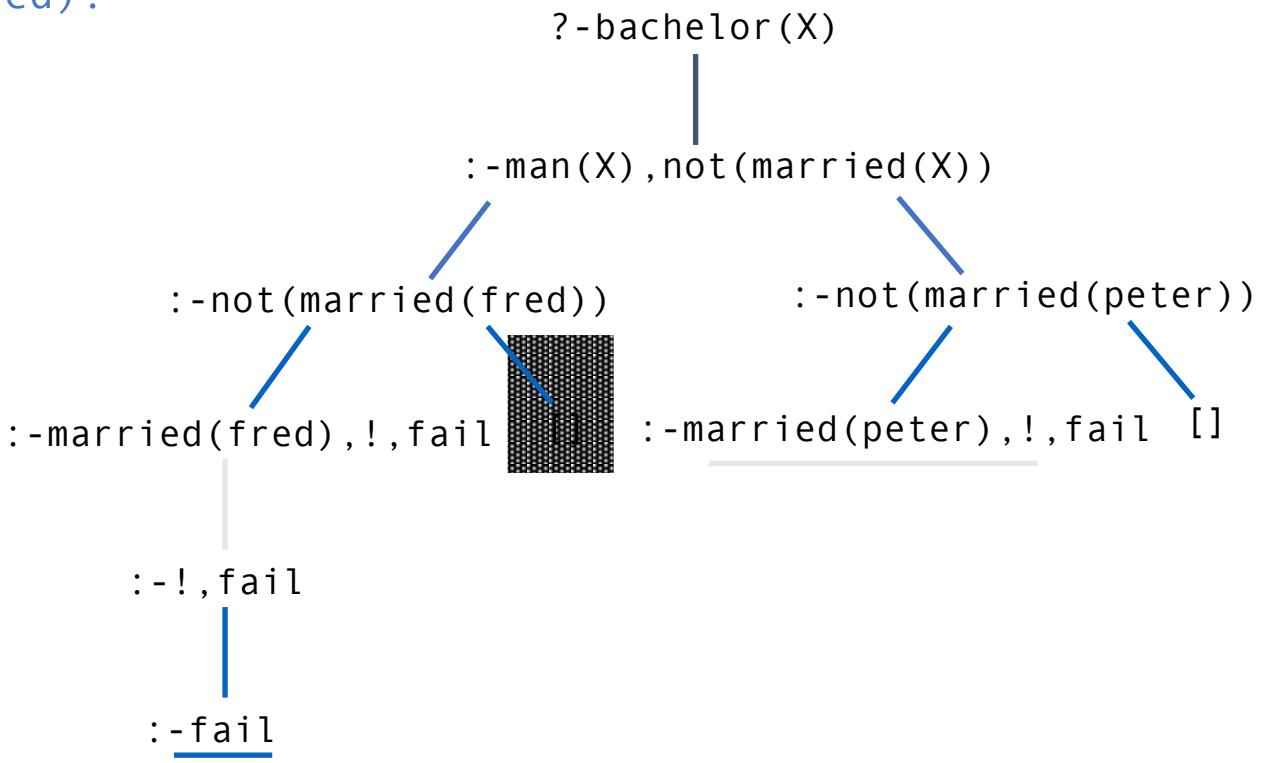
This will "ground" X.

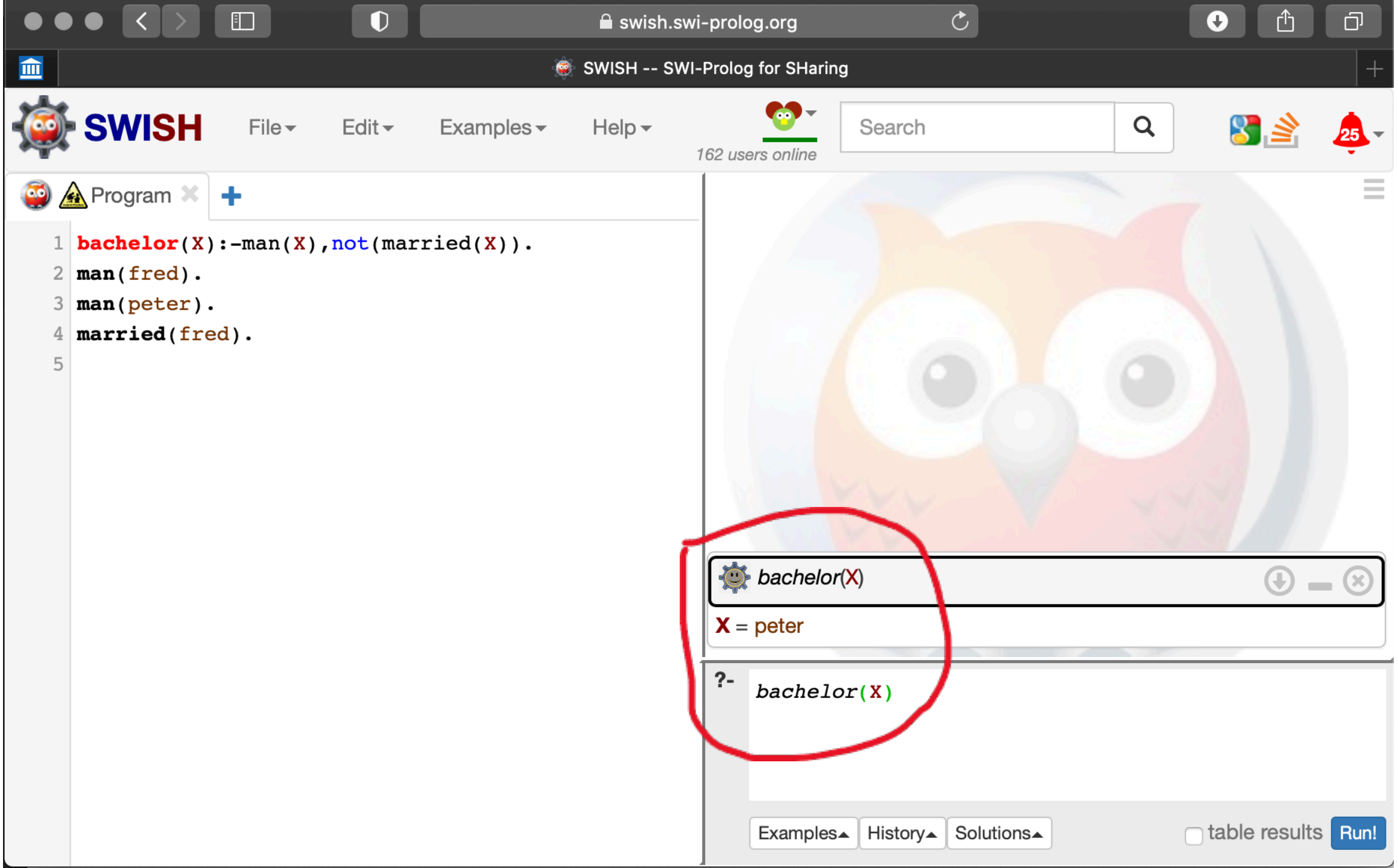


Prolog's `not` is unsound – Avoiding the Problem

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

This will "ground" X.




swish.swi-prolog.org
SWISH -- SWI-Prolog for SHaring
SWISH
File
Edit
Examples
Help
162 users online
Search
25
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

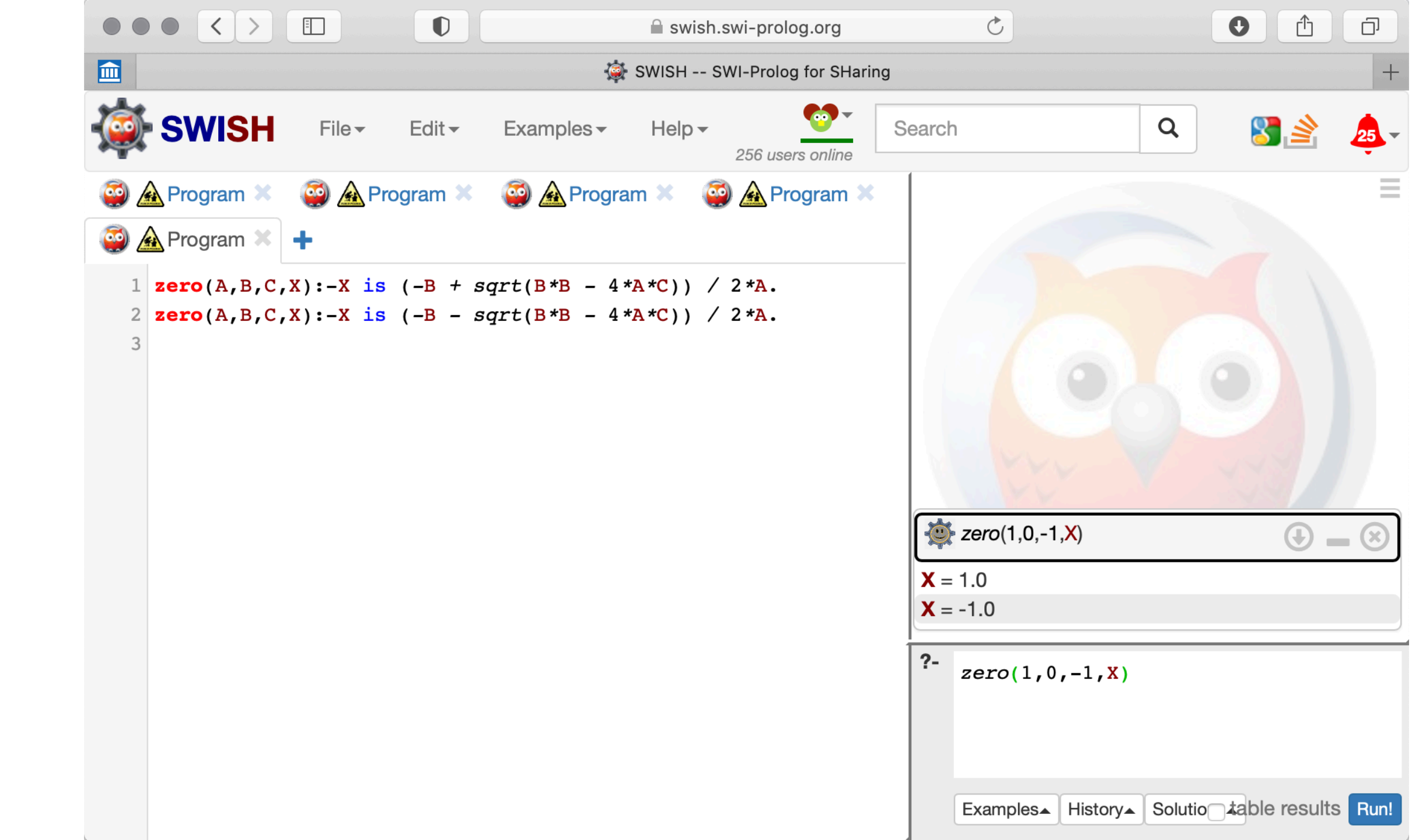
Prolog arithmetic vs.unification

```
?-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.
```


swish.swi-prolog.org
SWISH -- SWI-Prolog for SHaring
SWISH
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
Solutio
table results
Run!