1

Intersection of Two Lists

```
inter([], Y, []).
inter([X|L],Y,[X|Z]) :-
     member(X,Y), inter(L, Y, Z).
inter([X|L],Y,Z) :-
     nonMember(X,Y), inter(L,Y,Z).

nonMember(X, []).
nonMember(X, [Y | Z]) :-
     notEqual(X,Y), nonMember(X,Z).
```

This is awkward. We'd like to be able to define `nonMember` as `not(member)`:

```
nonMember(X, Y) :- not(member(X, Y)).
```

`?- not(P)` succeeds when `P` fails
fails when `P` succeeds

Negation as Failure

2

Where can `not` be used?

- Allowed on the RHS of a clause, as in

```
a :- not(b).
a(X) :- c(X),not(b(X)).
```

- Allowed in a goal

```
?- not(b(5))
?- foo(X), not(b(X))
```

- NOT allowed on the LHS of a clause

```
not(good(george))
```
X

```
not(good(X)) :- bad(X).
```
X

3

Negation as Failure

• `?- append(X, Y, [a]), not(=(X,[]))`
succeeds with `{X/[a], Y/[]}` only

• Definition of `inter`

```
inter([], Y, []).
inter([X | L], Y, [X | Z]) :-
     member(X,Y), inter(L,Y,Z).
inter([X | L], Y, Z) :-
     not(member(X,Y)), inter(L,Y,Z).
```

• Negation as failure NOT the same thing as logical negation. For

```
brother(john, mary).
```

the goal

```
?- not( good(bill))
```

succeeds, but it is not the case that

```
brother(john, mary) |- ¬ good(bill)
```


4

Examples of Using `not`

```
cprolog
C-Prolog version 1.4
| ?- [user].
| likes(peter,food).
| ^D
user consulted 76 bytes 1.78814e-08 sec.
yes
| ?- not(likes(peter,food)).
no
| ?- not(not(likes(peter,food))).
yes
| ?- likes(peter,flowers).
no
| ?- not(likes(peter,flowers)).
yes
| ?- not(not(likes(peter,flowers))).
no
| ?- not(likes(peter,X)).
no
| ?- not(not(likes(peter,X))).
X = _0
yes
```

5

More examples

```
| ?- trace.
yes
| ?- not(not(likes(peter,X))).
   (1) 1 Call: not not likes(peter,_0) ?
   (2) 1 Call: not likes(peter,_0) ?
   (3) 1 Call: likes(peter,_0) ?
   (3) 1 Exit: likes(peter,food)
   (2) 1 Back to: not likes(peter,_0) ?
   (2) 1 Fail: not likes(peter,_0)
   (1) 1 Exit: not not likes(peter,_0)

X = _0

yes
```


©1993 John Mylopoulos

6

Extra Control: The `Cut` Operator (!)

The symbol `!` (pronounced `cut`) provides some control over Prolog's search strategy.

- `!` always succeeds as a goal, but has the side-effect of removing all backtracking choice points developed up to this point since the current clause was invoked. That is, `!` commits the system to all choices made earlier in the search.

- In effect, `!` says that if you get this far, you have made the right choices, and you succeed or fail with that choice, and you may not backtrack over the cut, to use any other clause for the same goal.

- `!` applies only to the clause it is in. If call of this clause fails, the system is still allowed to backtrack into goals that called the clause with the `!`.

`!` can be viewed as a fence:

```
a :- b, c, d, !, e, f, g.
```

Backtracking can take place while `b,c,d` are being established; once the search crosses `!`, it can no longer backtrack back to `b, c` and `d`.

©1993 John Mylopoulos

7

Extra Control: `Cut`

`cut` can be used to define an if-then-else:

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

behaves like:

```
p :- if  q   then  r
     else  s
```

if `q` succeeds, pursue `r`
but `s` is **not** attempted!

if `q` fails, then try `s`.

©1993 John Mylopoulos

8

The Garden of Eden

```
numParents(X,Y) :- eden(X), =(Y,0).
numParents(X,Y) :- =(Y,2).
eden(adam).
eden(eve).

| ?- numParents(adam,Z)
Z = 0;
Z = 2;
no

| ?- numParents(P,N)
P = adam
N = 0;

P = eve
N = 0;

P = _21
N = 2;
no
```

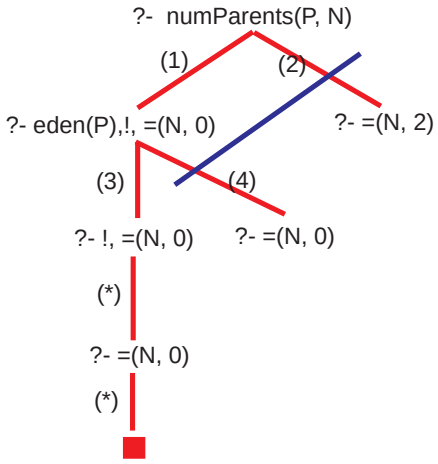

©1993 John Mylopoulos

9

The Garden of Eden (continued)

```
numParents(X,Y) :-
        eden(X),!,=(Y,0).          (1)
numParents(X,Y) :- =(Y,2).         (2)
eden(adam).                        (3)
eden(eve).                         (4)
```

```
| ?- numParents(adam,N).
N = 0;
no
```



©1993 John Mylopoulos

10

The Garden of Eden (cont'd)

```
numParents(X,Y) :- =(Y,2).         (1')
numParents(X,Y) :-
        eden(X),!,=(Y,0).          (2')
eden(adam).                        (3')
eden(eve).                         (4')
```

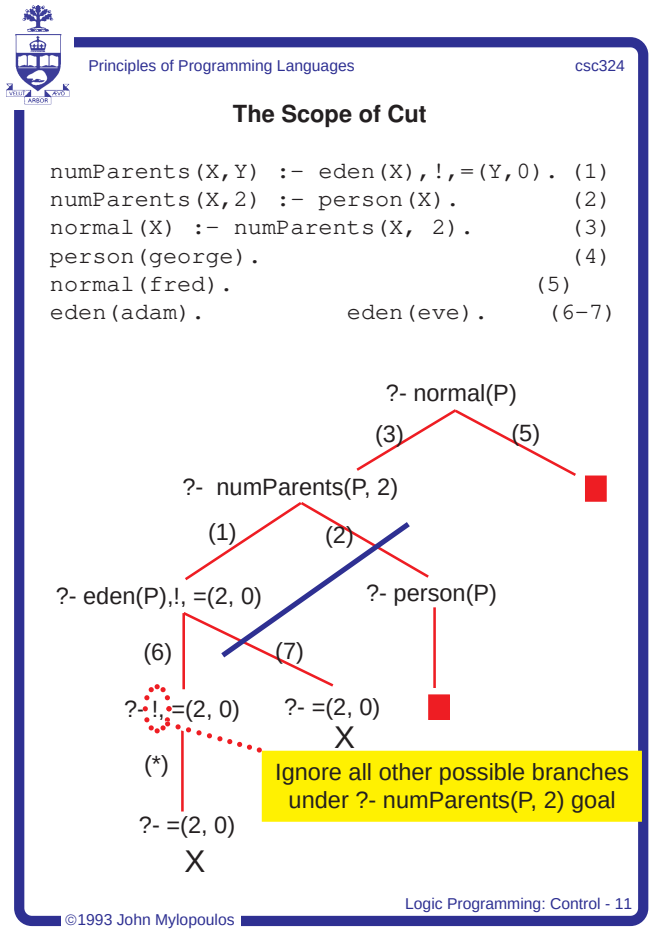
- `?- numParents(adam,N)` returns _____?

```
numParents(adam,0) :- !            (1'')
numParents(eve,0) :-  !            (2'')
numParents(X,2).                   (3'')
```

- `?- numParents(adam,N)` returns `{N/0}`
- `?- numParents(adam,2)` returns _____?

©1993 John Mylopoulos

11

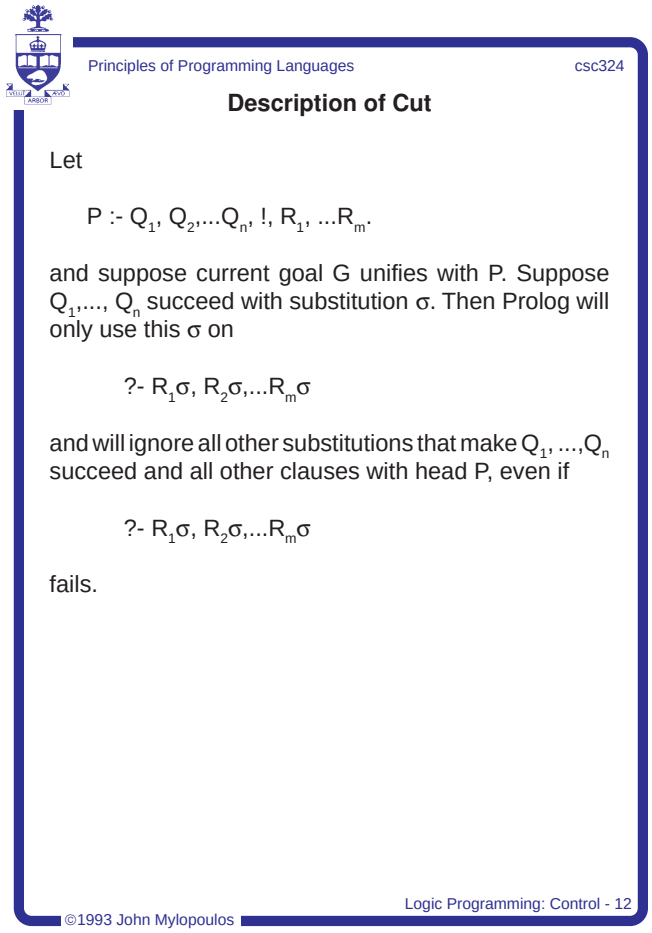


The Scope of Cut

```
numParents(X,Y) :- eden(X),!,=(Y,0).  (1)
numParents(X,2) :- person(X).         (2)
normal(X) :- numParents(X, 2).        (3)
person(george).                       (4)
normal(fred).                      (5)
eden(adam).          eden(eve).     (6-7)
```


©1993 John Mylopoulos

12



Description of Cut

Let

$P :- Q_1, Q_2,...Q_n, !, R_1, ...R_m.$

and suppose current goal G unifies with P. Suppose $Q_1,..., Q_n$ succeed with substitution σ. Then Prolog will only use this σ on

$?- R_1\sigma, R_2\sigma,...R_m\sigma$

and will ignore all other substitutions that make $Q_1, ...,Q_n$ succeed and all other clauses with head P, even if

$?- R_1\sigma, R_2\sigma,...R_m\sigma$

fails.

©1993 John Mylopoulos

13

Description of Cut

Exercise:

Trace the behaviour of `member` using the definition:

```
member(X, [X | _]) :- !.
member(X, [_ | Y]) :- member(X, Y).
```

for various combinations of arguments, for example

```
?- member(c,[a,b,c,d])
```

returns _____?

```
?- member(X,[a,b,c,d])
```

returns _____?

©1993 John Mylopoulos

14

Extra Control: `fail`

This special literal always fails, causing the initiation of backtracking.

Example: We want to represent the facts

"Maria likes animals, except for snakes"

```
likes(maria,X) :- animal(X)
```

won't do because it does not exclude snakes. Try instead

```
likes(maria,X) :- snake(X),!,fail.
likes(maria,X) :- animal(X).
```

`not` can actually be defined in terms of `fail`:

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


©1993 John Mylopoulos

15

Control - Repeat

`repeat` is a built-in goal that permits "iteration", and is typically used to implement repeated inputs from the user, without recursion.

`repeat` succeeds every time it is tried.

You may think of `repeat` in terms of the following definition:

```
repeat.
repeat :- repeat.
```


©1993 John Mylopoulos

16

For example:

```
tellme :-
   repeat,
   write('enter  input'), nl,
   read(X),
   process(X).

process(X) :- X == exit.
process(X)  :- write(X), nl, fail.
```

`repeat`'s function is demonstrated by:

```
| ?- tellme.
enter input
|: john(19).
john(19)
enter input
|: Z
_0
enter input
|: exit.
yes
```


©1993 John Mylopoulos