CSCI-344
Programming Language Concepts (Section 3)

Lecture 23

Prolog: Cuts

Instructor: Hossein Hojjat

November 2, 2016

Where we are

Done:

- Logical Programming in Prolog
- Lists

This session:

- Puzzle Example
- Cuts in Prolog

Puzzle Example

Baker, Cooper, Fletcher, Miller and Smith live in a five-story building. Baker doesn't live on the 5^{th} floor and Cooper doesn't live on the 1st floor. Fletcher doesn't live on the top or bottom floors, and he is not on a floor adjacent to Smith or Cooper. Miller lives on the some floor above Cooper.

Who lives on what floors?

Puzzle Example

Baker, Cooper, Fletcher, Miller and Smith live in a five-story building.
Baker doesn't live on the 5^{th} floor and Cooper doesn't live on the 1st floor.
Fletcher doesn't live on the top or bottom floors, and he is not on a floor adjacent to Smith or Cooper. Miller lives on the some floor above Cooper.

Who lives on what floors?

- We set up the solution as a list of five entries:

```
[floor(_,5),floor(_,4),floor(_,3),floor(_,2),floor(_,1)]
```

- Don't care variables are placeholders for the five names:
  `baker, cooper, fletcher, miller, smith`

Puzzle Example

Baker, Cooper, Fletcher, Miller and Smith live in a five-story building.
Baker doesn't live on the 5^{th} floor and Cooper doesn't live on the 1st floor.
Fletcher doesn't live on the top or bottom floors, and he is not on a floor adjacent to Smith or Cooper. Miller lives on the some floor above Cooper.

Who lives on what floors?

- We set up the solution as a list of five entries:

```
[floor(_,5),floor(_,4),floor(_,3),floor(_,2),floor(_,1)]
```

- Don't care variables are placeholders for the five names:
  `baker, cooper, fletcher, miller, smith`
- We associate variables `B,C,F,M,S` $\in \{1,2,3,4,5\}$ with five persons

Puzzle Example

Baker, Cooper, Fletcher, Miller and Smith live in a five-story building. Baker doesn't live on the 5^{th} floor and Cooper doesn't live on the 1st floor. Fletcher doesn't live on the top or bottom floors, and he is not on a floor adjacent to Smith or Cooper. Miller lives on the some floor above Cooper.

Who lives on what floors?

```
floors([floor(_,5),floor(_,4),
        floor(_,3),floor(_,2),floor(_,1)]).
building(Floors) :- floors(Floors),
```

Puzzle Example

Baker, Cooper, Fletcher, Miller and Smith live in a five-story building.
Baker doesn't live on the 5^{th} floor and Cooper doesn't live on the 1st floor.
Fletcher doesn't live on the top or bottom floors, and he is not on a floor adjacent to Smith or Cooper. Miller lives on the some floor above Cooper.

Who lives on what floors?

```
floors([floor(_,5),floor(_,4),
        floor(_,3),floor(_,2),floor(_,1)]).
building(Floors) :- floors(Floors),
  member(floor(baker, B), Floors), not(B is 5),
  member(floor(cooper, C), Floors), not(C is 1),
```

Puzzle Example

Baker, Cooper, Fletcher, Miller and Smith live in a five-story building.
Baker doesn't live on the 5th floor and Cooper doesn't live on the 1st floor.
Fletcher doesn't live on the top or bottom floors, and he is not on a floor adjacent to Smith or Cooper. Miller lives on the some floor above Cooper.

Who lives on what floors?

```
floors([floor(_,5),floor(_,4),
        floor(_,3),floor(_,2),floor(_,1)]).
building(Floors) :- floors(Floors),
  member(floor(baker, B), Floors), not(B is 5),
  member(floor(cooper, C), Floors), not(C is 1),
  member(floor(fletcher, F), Floors),
                        not(F is 1), not(F is 5),
  not(S is (F + 1)),  not(F is (S + 1)),
  not(F is (C + 1)),  not(C is (F + 1)),
```

Puzzle Example

Baker, Cooper, Fletcher, Miller and Smith live in a five-story building.
Baker doesn't live on the 5th floor and Cooper doesn't live on the 1st floor.
Fletcher doesn't live on the top or bottom floors, and he is not on a floor adjacent to Smith or Cooper. Miller lives on the some floor above Cooper.

Who lives on what floors?

```
floors([floor(_,5),floor(_,4),
        floor(_,3),floor(_,2),floor(_,1)]).
building(Floors) :- floors(Floors),
  member(floor(baker, B), Floors), not(B is 5),
  member(floor(cooper, C), Floors), not(C is 1),
  member(floor(fletcher, F), Floors),
                        not(F is 1), not(F is 5),
  not(S is (F + 1)),  not(F is (S + 1)),
  not(F is (C + 1)),  not(C is (F + 1)),
  member(floor(miller, M), Floors), (M > C),
```

Puzzle Example

Baker, Cooper, Fletcher, Miller and Smith live in a five-story building.
Baker doesn't live on the 5^{th} floor and Cooper doesn't live on the 1st floor.
Fletcher doesn't live on the top or bottom floors, and he is not on a floor adjacent to Smith or Cooper. Miller lives on the some floor above Cooper.

Who lives on what floors?

```
floors([floor(_,5),floor(_,4),
        floor(_,3),floor(_,2),floor(_,1)]).
building(Floors) :- floors(Floors),
  member(floor(baker, B), Floors), not(B is 5),
  member(floor(cooper, C), Floors), not(C is 1),
  member(floor(fletcher, F), Floors),
                        not(F is 1), not(F is 5),
  not(S is (F + 1)),  not(F is (S + 1)),
  not(F is (C + 1)),  not(C is (F + 1)),
  member(floor(miller, M), Floors), (M > C),
  member(floor(smith, S), Floors).
```

Puzzle Example

Baker, Cooper, Fletcher, Miller and Smith live in a five-story building.
Baker doesn't live on the 5^{th} floor and Cooper doesn't live on the 1st floor.
Fletcher doesn't live on the top or bottom floors, and he is not on a floor adjacent to Smith or Cooper. Miller lives on the some floor above Cooper.

Who lives on what floors?

```
?- building(X).
X = [floor(miller, 5),
     floor(fletcher, 4),
     floor(baker, 3),
     floor(cooper, 2),
     floor(smith, 1)].
```

Impure Features

- Declarative programming: no side-effects
- Sometimes declarative programming languages offer **"impure"** expressions
- μScheme supports global variable definitions
- Prolog supports cut `!` predicate to have side effect on backtracking behavior
- Forces Prolog to stop searching for alternatives when an answer is found
- Cut prunes the search space: it can remove unwanted answers

Cut Example

```
f(1).
member(X, [X|_]).
member(X, [_|T]) :- member(X,T).
```

- The following query succeeds three times

```
?- member(2,[2,3,2,4]),f(Y).
Y = 1;
Y = 1.
```

- What if we are only interested in a single answer?
- We don't want to consider the alternative solutions after first answer

```
f(1).
member(X, [X|_]) :- !.
member(X, [_|T]) :- member(X,T).
```

Cut Example

```
f(1).
member(X, [X|_]).
member(X, [_|T]) :- member(X,T).
```

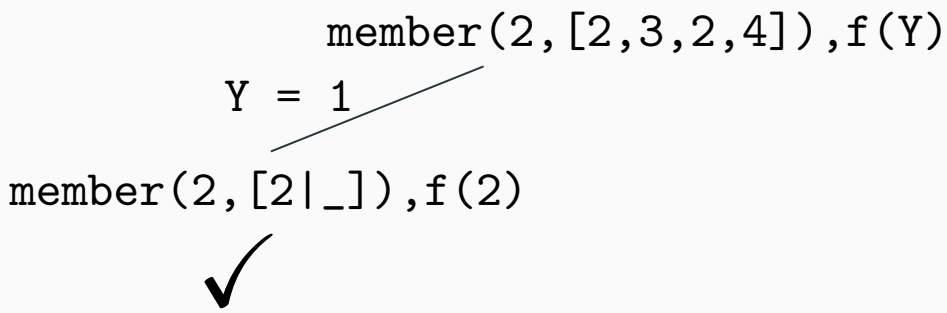
```
?- member(2,[2,3,2,4]),f(Y).
```

member(2,[2,3,2,4]),f(Y)

Cut Example

```
f(1).
member(X, [X|_]).
member(X, [_|T]) :- member(X,T).
```

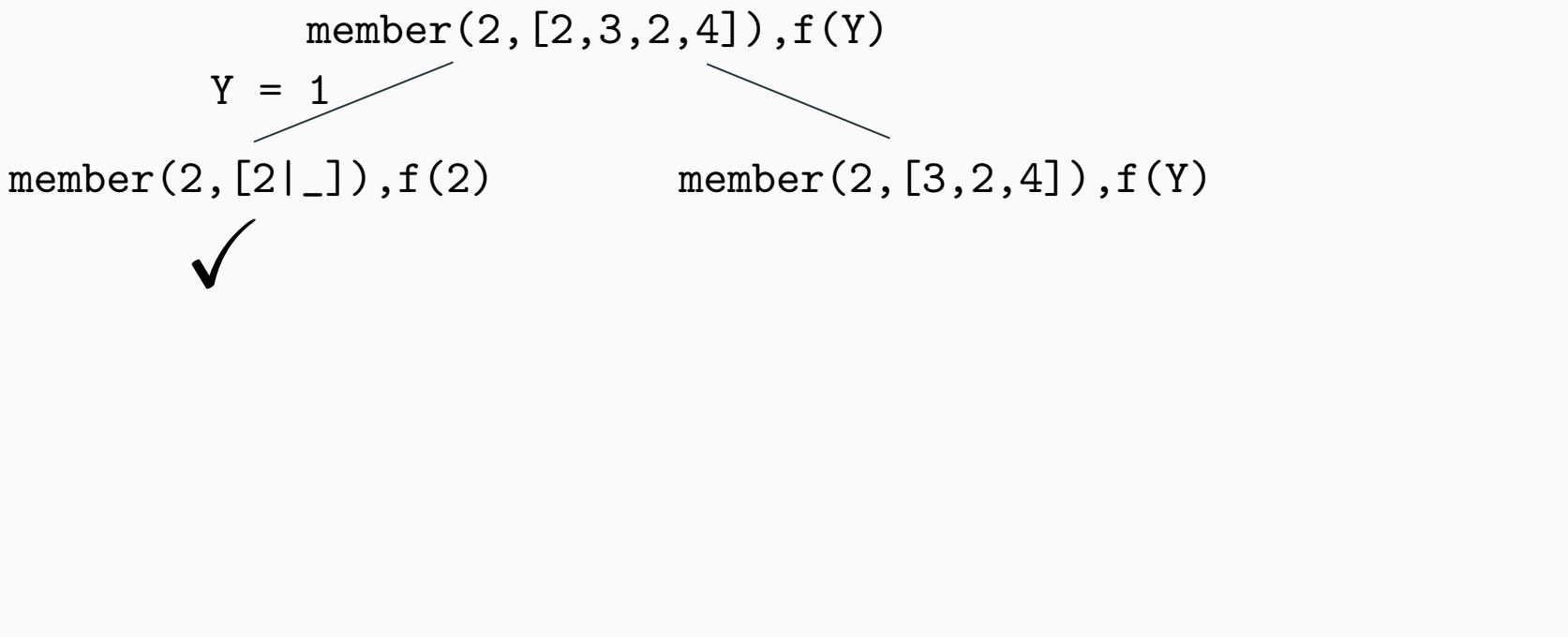
```
?- member(2,[2,3,2,4]),f(Y).
```



Cut Example

```
f(1).
member(X, [X|_]).
member(X, [_|T]) :- member(X,T).
```

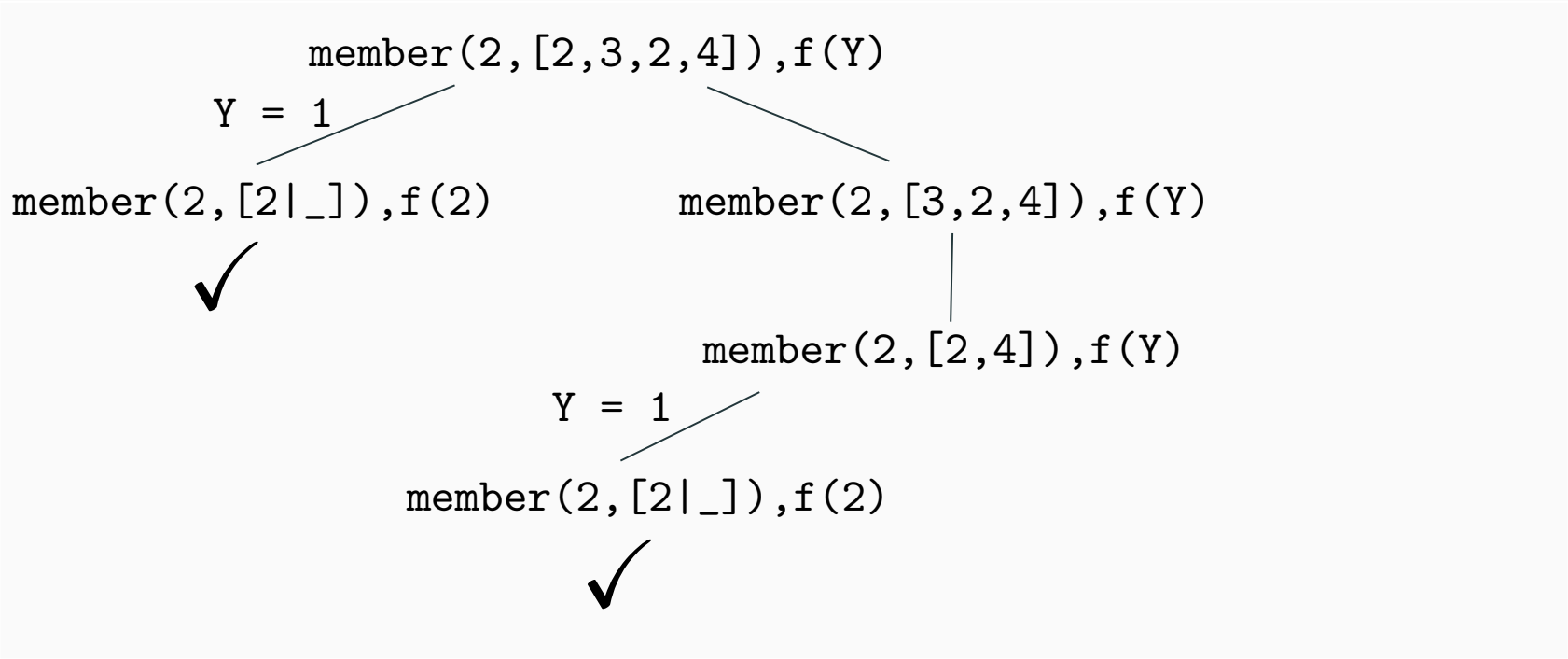
```
?- member(2,[2,3,2,4]),f(Y).
```



Cut Example

```
f(1).
member(X, [X|_]).
member(X, [_|T]) :- member(X,T).
```

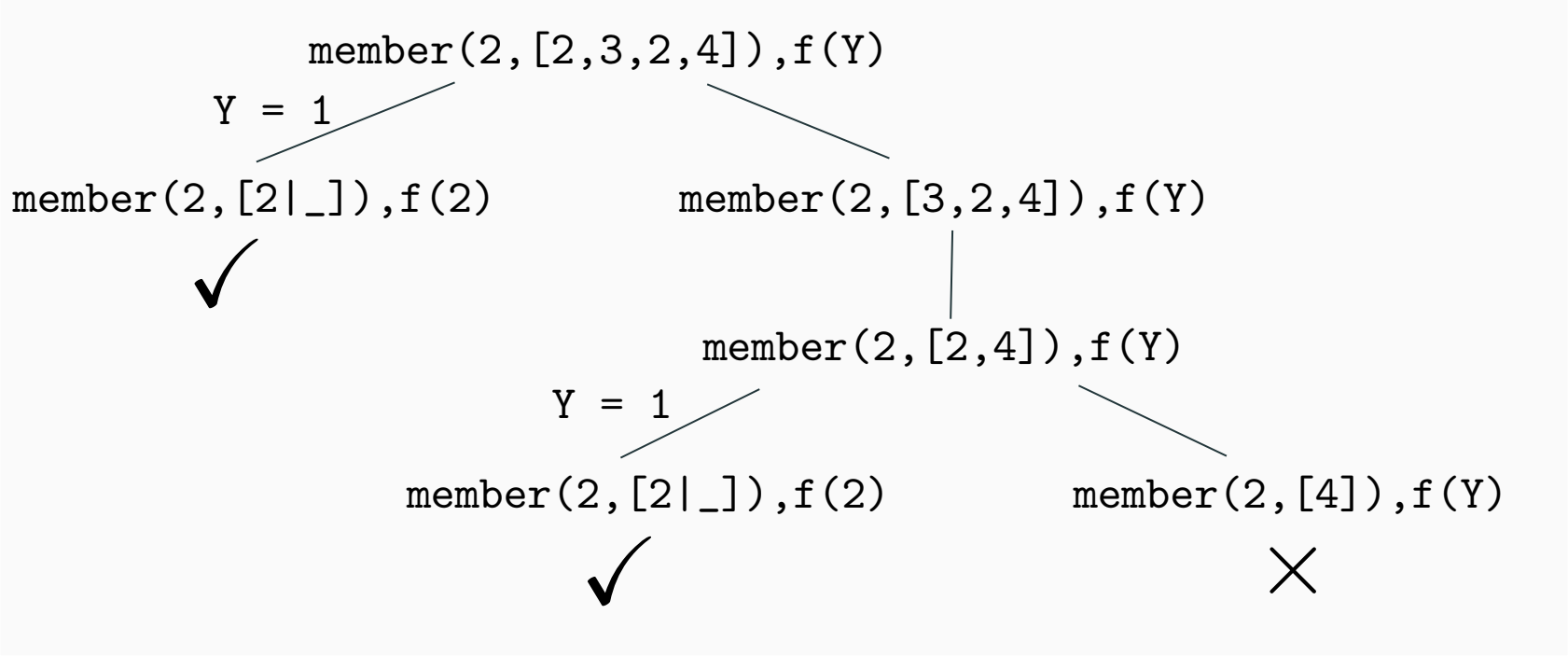
```
?- member(2,[2,3,2,4]),f(Y).
```



Cut Example

```
f(1).
member(X, [X|_]).
member(X, [_|T]) :- member(X,T).
```

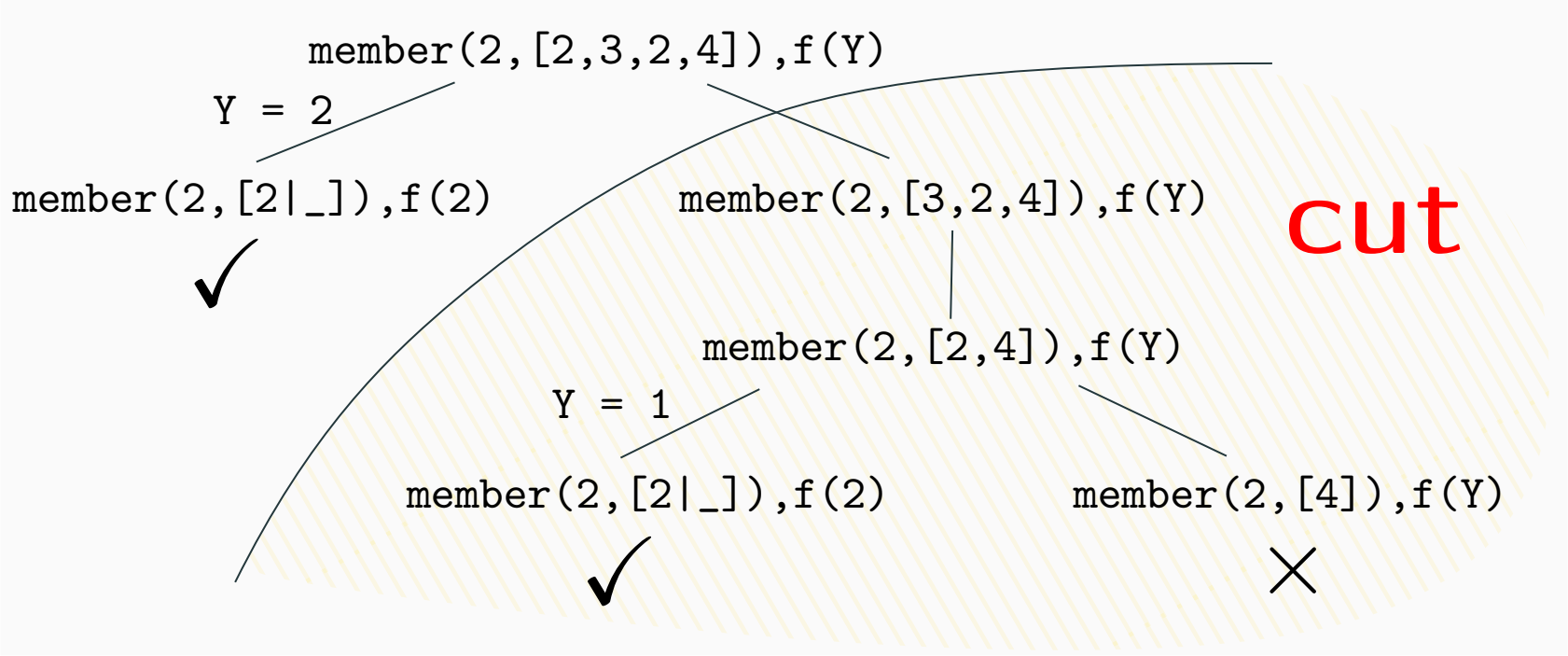
```
?- member(2,[2,3,2,4]),f(Y).
```



Cut Example

```
f(1).
member(X, [X|_]):- !.
member(X, [_|T]) :- member(X,T).
```

```
?- member(2,[2,3,2,4]),f(Y).
```



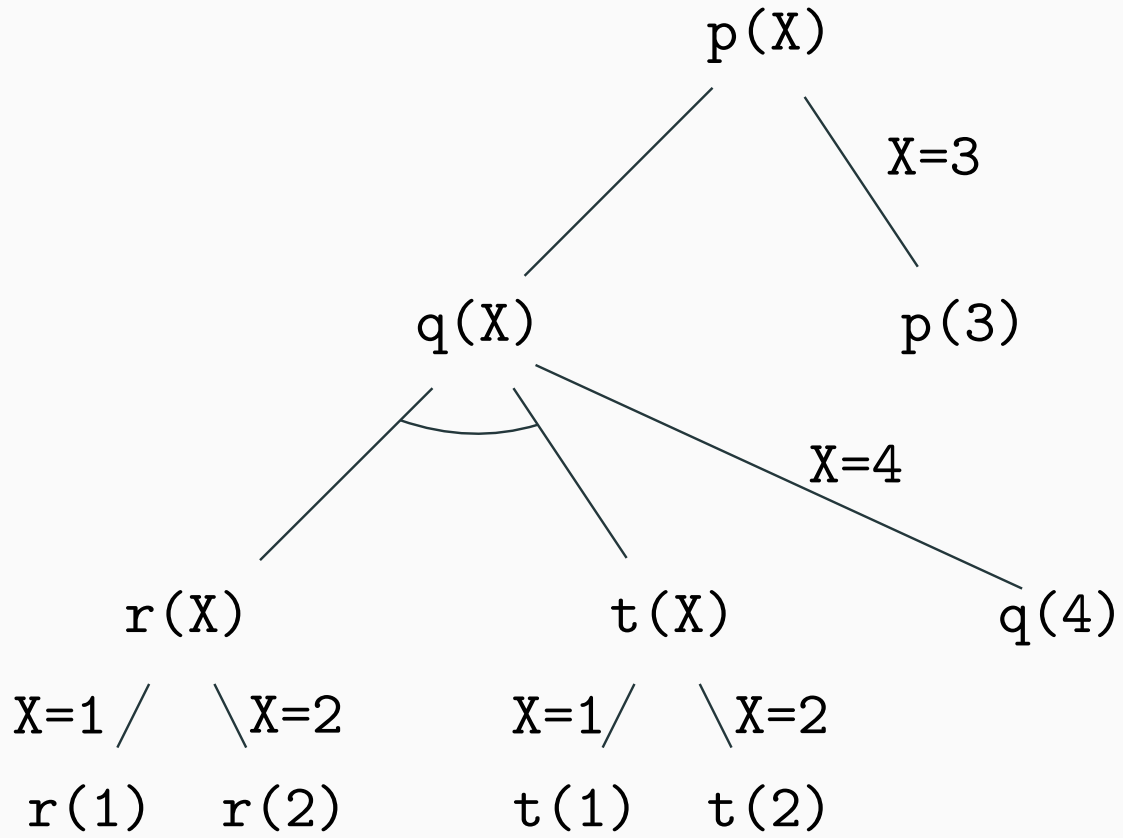
Cut

- Cut `!` is a predicate that always succeeds
- Prolog commits to all the choices that have been made from the point where the rule was applied up through the cut
- In a rule of the form:

  `q :- p1,...,pn, ! , r1,...,rm`
- when we reach the cut it commits us to:
  - Using this particular clause for `q`
  - Choices of variables made when evaluating `p1` ,..., `pn`

Cut-Example

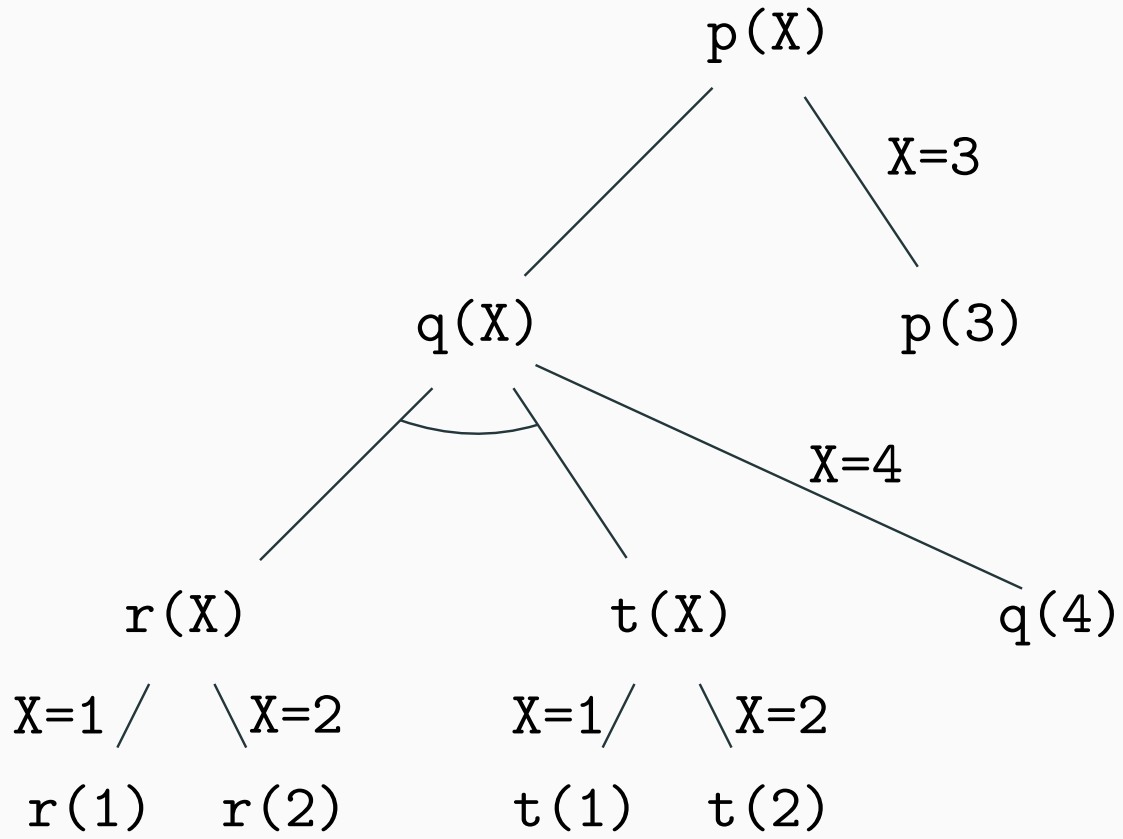
```
-> p(X) :- q(X).
q(X) :- r(X), t(X).
p(3).
q(4).
r(1).
r(2).
t(1).
t(2).
?- p(X).
```



Cut-Example

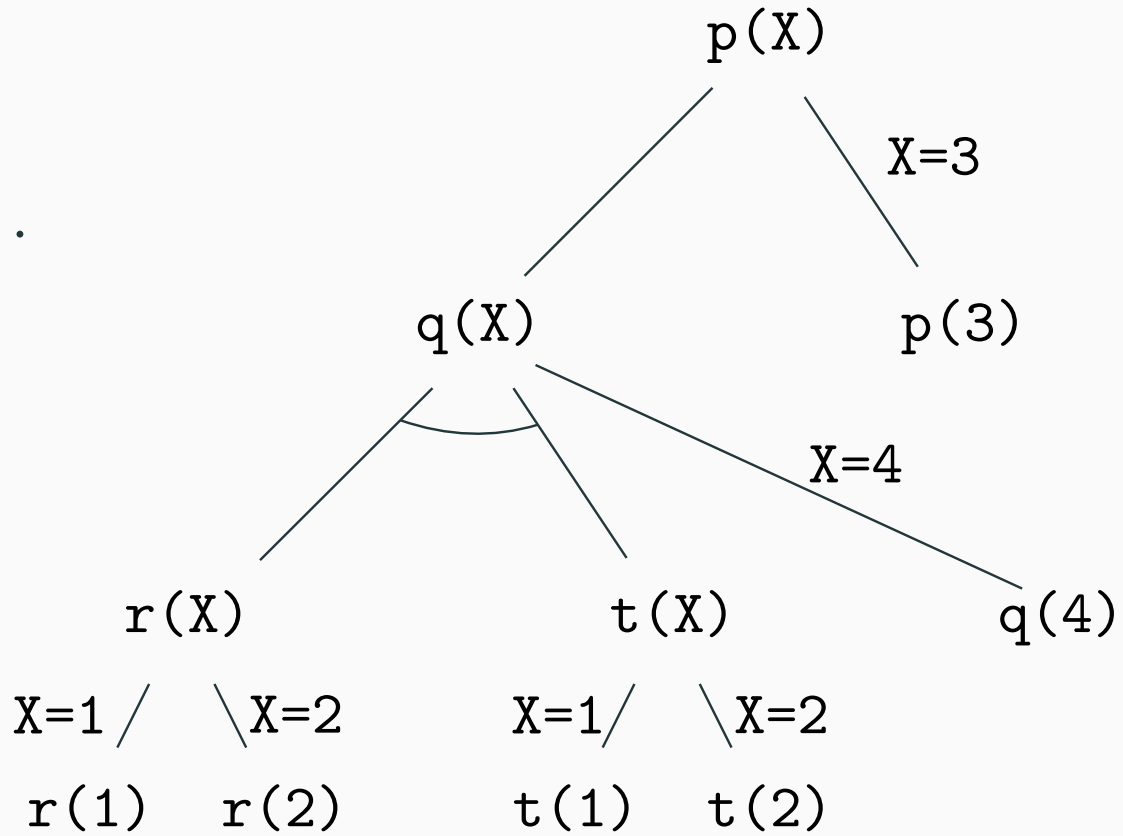
```
-> p(X) :- q(X).
q(X) :- r(X), t(X).
p(3).
q(4).
r(1).
r(2).
t(1).
t(2).
?- p(X).

X = 1;
X = 2;
X = 4;
X = 3;
```



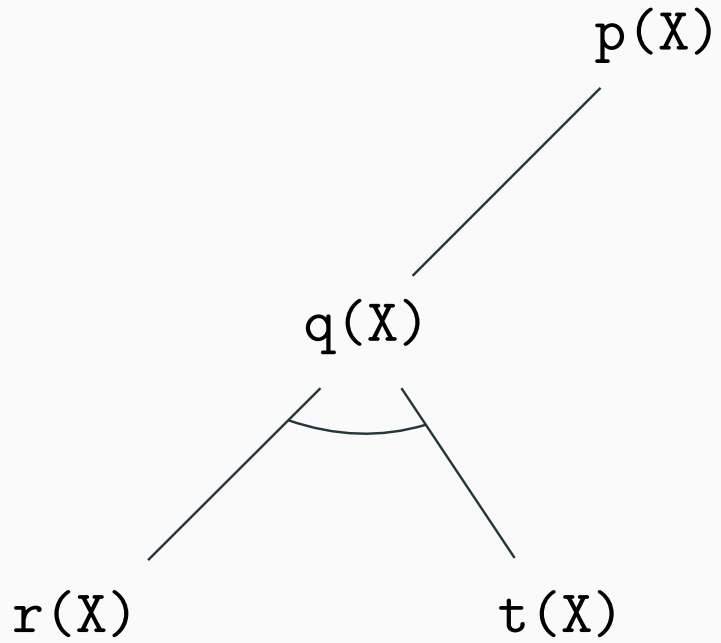
Cut-Example

```
-> p(X) :- q(X).
q(X) :- r(X), !, t(X).
p(3).
q(4).
r(1).
r(2).
t(1).
t(2).
?- p(X).
```



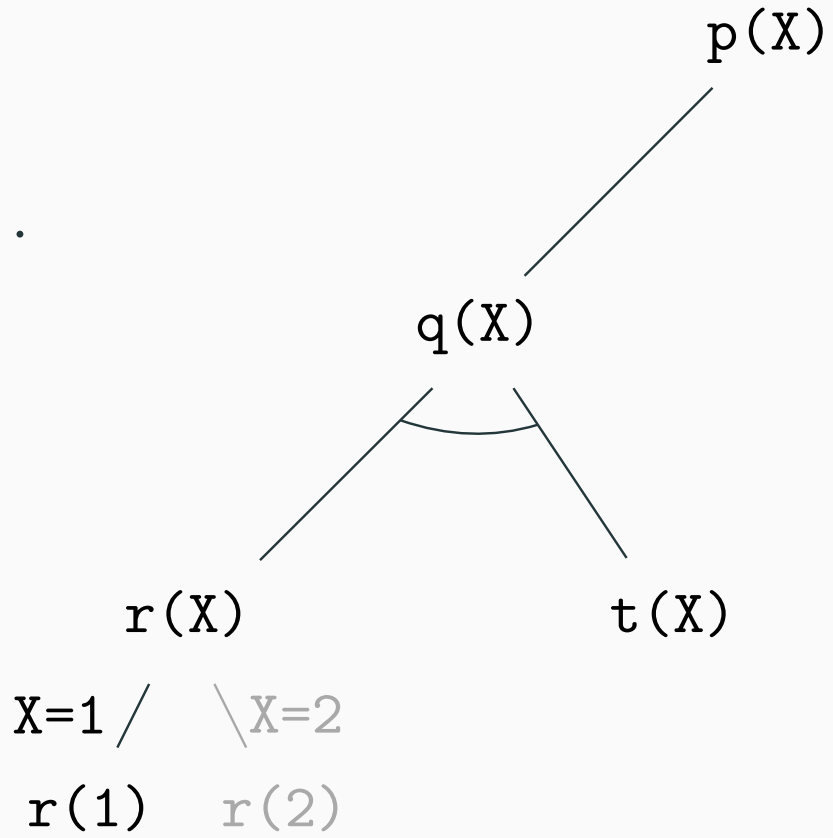
Cut-Example

```
-> p(X) :- q(X).
q(X) :- r(X), !, t(X).
p(3).
q(4).
r(1).
r(2).
t(1).
t(2).
?- p(X).
```



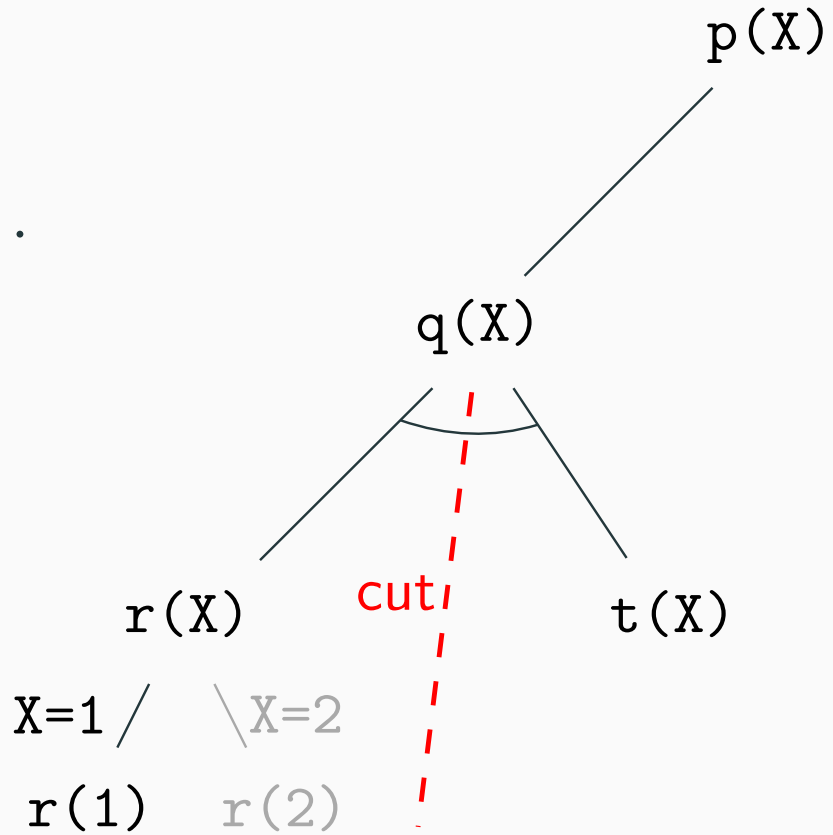
Cut-Example

```
-> p(X) :- q(X).
q(X) :- r(X), !, t(X).
p(3).
q(4).
r(1).
r(2).
t(1).
t(2).
?- p(X).
```



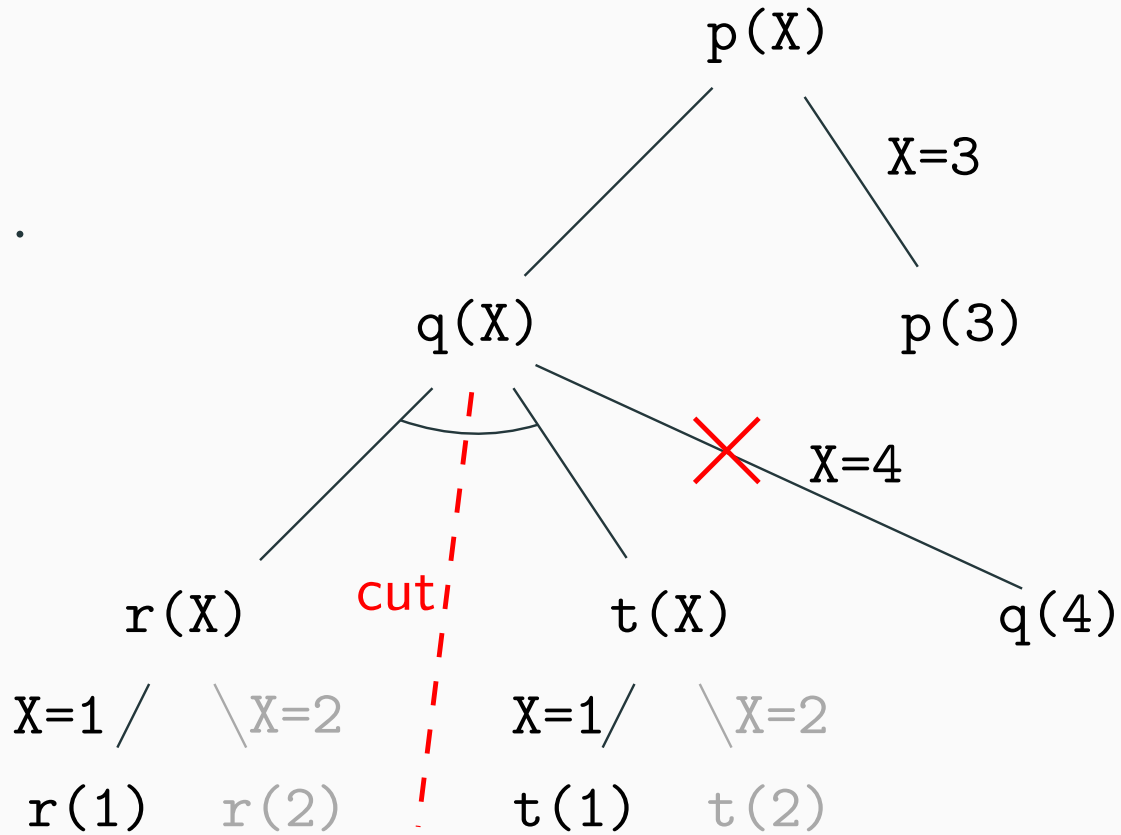
Cut-Example

```
-> p(X) :- q(X).
q(X) :- r(X), !, t(X).
p(3).
q(4).
r(1).
r(2).
t(1).
t(2).
?- p(X).
```



Cut-Example

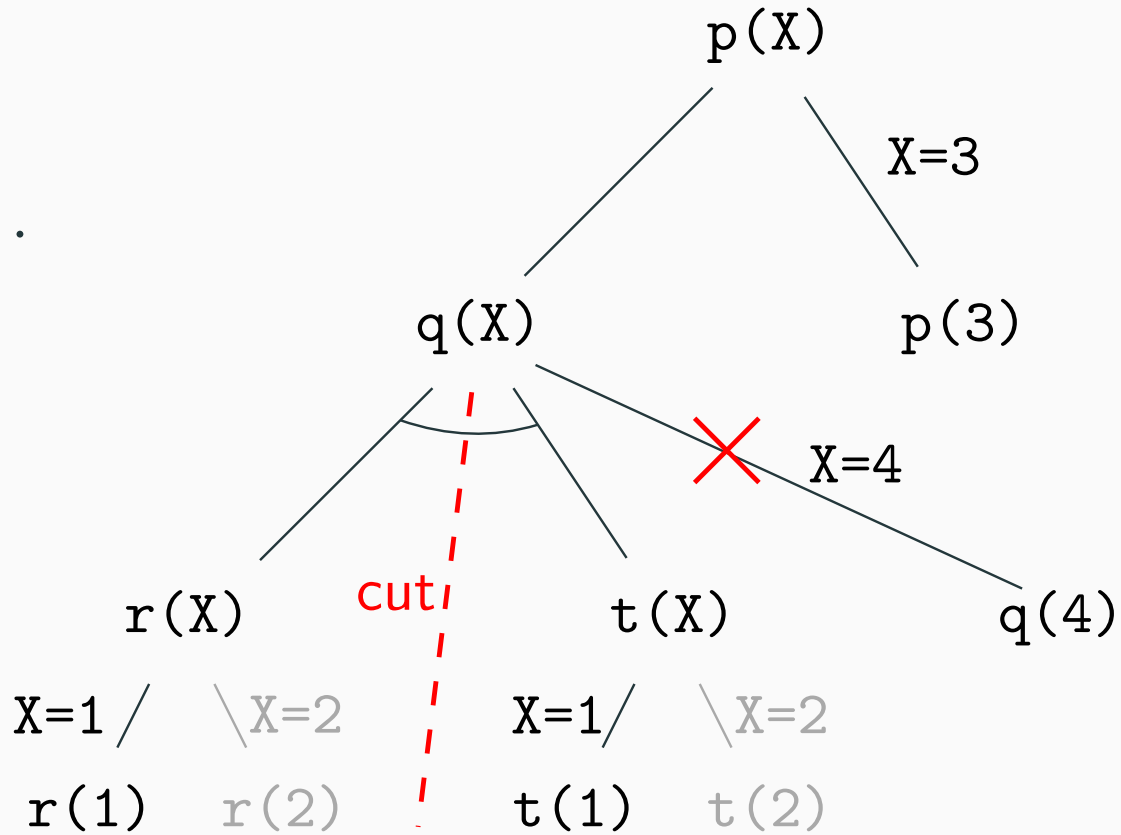
```
-> p(X) :- q(X).
q(X) :- r(X), !, t(X).
p(3).
q(4).
r(1).
r(2).
t(1).
t(2).
?- p(X).
```



Cut-Example

```
-> p(X) :- q(X).
q(X) :- r(X), !, t(X).
p(3).
q(4).
r(1).
r(2).
t(1).
t(2).
?- p(X).

X = 1;
X = 3;
```



Using Cuts

- Predicate `max` succeeds if third argument is the maximum of first two arguments

```
max(X,Y,Y):- X =< Y.
max(X,Y,X):- X>Y.
```

```
?- max(5,10,Y).
Y = 10;
```

- Inefficiency: if you ask for more answers after `Y=10`, it tries to match the next clause

max with cut

```
max(X,Y,Y):- X =< Y, !.
max(X,Y,X):- X>Y.
```

- If `X =< Y` succeeds, the cut commits us to this choice, and the second clause is never considered
- If `X =< Y` fails, Prolog goes on to the second clause

Cut Classification

Good/Green Cuts

- Introduced to make program more efficient by eliminating "known" useless computations
- Without the cuts, the program produces the same solutions (perhaps requiring more time or space)
- Logical meaning of the program remains the same

Bad/Red Cuts

- Introduced to make program more efficient by eliminating some meaningful solutions
- Without the cuts, the program might produce different solutions
- Changes the logical meaning of the program

Summary

This Lecture

- Prolog: Cuts

Next Lecture

- Type Inference