

CS 152: *Programming Language Paradigms*



Prolog continued: math, cuts, & lists

Prof. Tom Austin

San José State University

Math in Prolog

Arithmetic in Prolog

```
heists(joker, 97).
```

```
heists(penguin, 18).
```

```
heists(catwoman, 31).
```

```
heists(scarecrow, 42).
```

```
combined_heists(X, Y, Total) :-
```

```
    heists(X, XN), heists(Y, YN),
```

```
    Total = XN + YN.
```

```
?- combined_heists(catwoman,  
                    scarecrow,  
                    T) .
```

```
T = 31+42.
```

Using "is" operator

```
combined_heists(X, Y, Total) :-  
    heists(X, XN), heists(Y, YN),  
    Total is XN + YN.
```

...

```
?- combined_heists(catwoman,  
                    scarecrow, T).
```

```
T = 73.
```

The Cut Operator



"Learn Prolog Now"
section 10.2

The Cut Operator

Motivation:

- Prolog may needlessly backtrack
- We wish to stop the backtracking to optimize our code.

max example (no cuts)

$\text{max}(X, Y, Y) :- X \leq Y.$


$\text{max}(X, Y, X) :- X > Y.$

Using max

?- max(2, 3, M) .

M = 3 ;

false.



Why continue
the search?

?- max(2, 1, M) .

M = 2 .

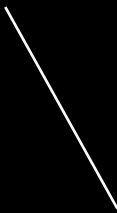
Two types of cuts (!)

- **A green cut**
 - improves performance or memory usage
 - Does not alter results
- **A red cut**
 - controls resolution to prevent future matches
 - changes the results
 - is considered "bad form"

max example (no cuts)

$\text{max}(X, Y, Y) :- X \leq Y.$

$\text{max}(X, Y, X) :- X > Y.$



If true, no
need to
keep
searching

max example, with green cut

$$\max (X, Y, Y) :- X =< Y,$$

! .

$$\max (X, Y, X) :- X > Y.$$

Last rule, so
no cut needed.


Red Cut Example

Batman is enemies with all villains,
unless the villain
is also a romantic
interest.



Red Cut Example

```
enemy(batman, X) :-  
    romantic_interest(X),  
    !,  
    fail.
```



No backtracking once
we make it here.

```
enemy(batman, X) :- villain(X).
```

Red Cut Example

```
bad_breakup(batman, talia).  
bad_breakup(batman, poison_ivy).  
  
enemy(batman, X) :-  
    romantic_interest(X),  
    !,  
    bad_breakup(batman, X).  
enemy(batman, X) :- villain(X).
```

Avoiding red cut

```
bad_breakup(batman, talia).
```

```
bad_breakup(batman, poison_ivy).
```

```
enemy(batman, X) :- villain(X),  
                    \+ romantic_interest(X).
```

```
enemy(batman, X) :- villain(X),  
                    bad_breakup(batman, X).
```

Alternate syntax
for not

Lists in Prolog

List

- Syntax for head/tail:
[Head|Tail]
- Syntax for multiple elements
[1, 2, 3, 4]
- Prolog list solutions are often recursive.

myappend

% Base case

myappend([], L2, L2).

% Recursive case

myappend([H|T1], L2, [H|T2]) :-
 myappend(T1, L2, T2).

Using myappend

?—

Using myappend

```
?- myappend([1,2], [3,4], Result).
```

```
Result = [1, 2, 3, 4].
```

```
?-
```

Using myappend

```
?- myappend([1,2], [3,4], Result).
```

```
Result = [1, 2, 3, 4].
```

```
?- myappend([1,2], [3,4], [1,2,3,4]).
```

```
true.
```

```
?-
```

Using myappend

```
?- myappend([1,2], [3,4], Result).
```

```
Result = [1, 2, 3, 4].
```

```
?- myappend([1,2], [3,4], [1,2,3,4]).
```

```
true.
```

```
?- myappend(Prefix, [3,4], [1,2,3,4]).
```

```
Prefix = [1, 2] ;
```

```
false.
```

```
?-
```

myreverse

```
myreverse([], []).
```

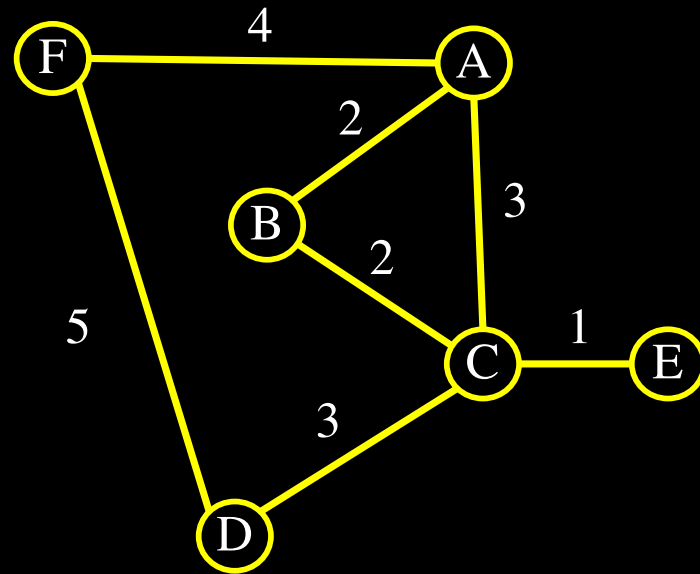
```
myreverse([H|T], L) :-  
    myreverse(T, RT),  
    append(RT, [H], L).
```


in_list & quicksort

(in-class)

graph.prolog facts

```
edge(a, b, 2).  
edge(b, a, 2).  
edge(a, c, 3).  
edge(c, a, 3).  
edge(a, f, 4).  
edge(f, a, 4).  
edge(b, c, 2).  
edge(c, b, 2).  
edge(c, d, 3).  
edge(d, c, 3).  
edge(c, e, 1).  
edge(e, c, 1).  
edge(d, f, 5).  
edge(f, d, 5).
```



graph.prolog rules

```
find_path(Start, End, Cost, Path) :-  
    edge(Start, End, Cost),  
    Path = [Start, End].
```

```
find_path(Start, End, TotalCost, Path) :-  
    edge(Start, X, InitCost),  
    find_path(X, End, RestCost, TailPath),  
    TotalCost is InitCost + RestCost,  
    Path = [Start|TailPath].
```

Debugging Prolog

```
?- find_path(a, c, TC, P).
```

```
TC = 3,
```

```
P = [a, c] ;
```

```
TC = 4,
```

```
P = [a, b, c] ;
```

```
TC = 7,
```

```
P = [a, b, a, c] ;
```

```
TC = 8,
```

```
P = [a, b, a, b, c] ;
```

```
TC = 11,
```

```
P = [a, b, a, b, a, c] ;
```

```
TC = 12,
```

```
P = [a, b, a, b, a, b, c]
```

Debugging Prolog

- To walk through Prolog's steps:
 ?- trace.
 true.
- Run your queries normally, hitting enter to step forward
- To stop tracing:
 ?- notrace.
 true.
- <Example in class>

Lab: Graph

Fix `graph.prolog` to avoid retracing steps.

Add a `Visited` variable to `find_path`.

Initially, `Visited` is an empty list.

At each step, add the current node.

Do not try a node if it has been visited.

Batman, scary villains redux

(in-class)

Homework

- Airline reservation system.

- Sample fact:

```
flight(sfo, lax, 8:00, 9:20, 86.31) .
```

- Details in Canvas.