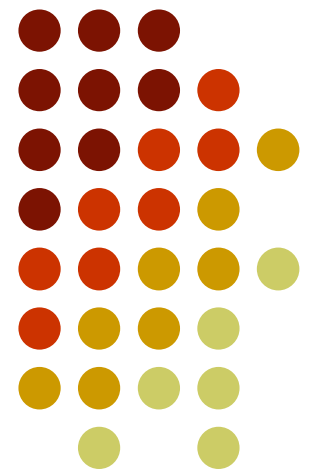


Introduction to Prolog Lists, Recursions, Cuts

CS181: Programming Languages



Topics:

- Lists
- Recursion
- Cuts





Lists

- Variable length ordered sequences of elements
- Very common data structures in non-numeric programming

- A list is either an **empty** list

[]

- or it has two components:

the **head** and the **tail**

.(a,[])



Lists

- There are two ways to represent lists

`. (a, . (b, . (c, [])))`

`[a, b, c]`

- Lists can contain other lists and variables

`[]`

`[the, men, [like, to, fish]]`

`[a, V1, b, [X, Y]]`



Lists

- The list with head X and tail Y is written

$[X|Y]$

List	Head	Tail
$[a, b, c]$	a	$[b, c]$
$[]$	(None)	(None)
$[[a, b], c]$	$[a, b]$	$[c]$
$[a, [b], c]$	a	$[[b], c]$
$[a, [b, c]]$	a	$[[b, c]]$



List Matching

List 1	List 2	Instantiations
[X,Y,Z,]	[john, likes, fish]	X = john Y = likes Z = fish
[X Y]	[cat]	X = cat Y = []
[X Y]	[[cat], dog]	X = [cat] Y = [dog]



List Matching

- Example:

$p([1, 2, 3]).$

$p([john, likes, [football, baseball]]).$

$?- p[X|Y].$

$X=1 \quad Y=[2,3]$

$X=john \quad Y=[likes, [football, baseball]]$



Lists and Recursion

- More examples:

`member(X, [X|L]).`

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

`last(X, [X]).`

`last(X, [Y|L]) :- last(X, L).`



Recursion

- Always define the **boundary condition** and the **recursive step**.
- Careful with the recursive step! The first rule will work, but the second one will loop forever:

ancestor(X,Y):- **parent(X,Y)**.

ancestor(X,Y):- ancestor(X,Y), **parent(Y, Z)**.

ancestor(X,Y):- **parent(X,Y)**.

ancestor(X,Y):- **parent(X,Z)**, ancestor(Z,Y).



Recursion

- Also you must be careful to avoid **circular definitions**:

```
parent(X,Y) :- child(Y,X).  
child(A,B) :- parent(B,A).
```



Recursions

- Another example of a recursion (however, note that Prolog is not an efficient way to do numerical computations):

factorial(0, 1).

factorial(N, F) :- N>0,
 N1 is N-1,
 factorial(N1, F1),
 F is N * F1.



Cuts

- ! is a special PROLOG facility called *the cut*.
- Cuts may be inserted anywhere within a clause to prevent backtracking to previous subgoals, for example:

$p(X) \text{ :- } b(X), c(X), !, d(X), e(X).$



Cuts

- Suppose that this clause has been invoked with a goal matching $p(X)$ and the subgoals $b(X)$ and $c(X)$ have been satisfied. On encountering the cut:
 - 1) The cut will succeed and PROLOG will try to satisfy subgoals $d(X)$ and $e(X)$.
 - 2) If $d(X)$ and $e(X)$ succeed then $p(X)$ succeeds.
 - 3) If $d(X)$ and $e(X)$ do not succeed and backtracking returns to the cut, then the backtracking process will immediately terminate and $p(X)$ fails.



Cuts

- Example:

$\text{max}(A,B,B) \text{ :- } A < B.$

$\text{max}(A,B,A).$

$?- \text{max}(3,4,M).$

$M = 4 \quad ;$

$M = 3$



Cuts

- Confirming the choice of a rule:

$\text{max}(A, B, B) \text{ :- } A < B, !.$
 $\text{max}(A, B, A).$

$?- \text{max}(3, 4, M).$

$M = 4$



Cuts

- Avoiding useless searches:

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

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

?- X is 5,

`member(X, [5, 9, 24, 17, 5, 2]),`

`X < 4.`



Cuts

- Cuts are commonly used in conjunction with the generate-and-test programming paradigm

```
find_just_one_solution(X) :-  
    candidate_solution(X),  
    test_solution(X),  
    !.
```



Cuts

- **cut** – **fail** combination, to express negation:

```
nonsibling(X, Y) :- sibling(X, Y), !, fail.  
nonsibling(X, Y).
```



References

- Clocksin, W.F., and Mellish C.S. *Programming in Prolog*. 4th edition. New York: Springer-Verlag. 1994.
- Aaby, A. *Prolog Tutorial*. Walla Walla College. 1997. On line.
http://cs.wwc.edu/~cs_dept/KU/PR/Prolog.html