Tutorial 8

prepared by Anya Tafliovich[1]

July 23, 2003

[1]based on material provided by Diane Horton, Eric Joanis, Alfredo Gabaldon

1 Lists in Prolog

```
count(L,E,N)    the list L contains N copies of E.
Precondition:   L and E are instantiated.

count([],_,0).
count([E|Rest],E,N) :- count(Rest,E,N1), N is N1 + 1.
count([X|Rest],E,N) :- \+(X=E), count(Rest, E, N).



element_at(X,L,K)   X is the  kth  element in list L; ( start with 1)
Precondition:       K is instantiated.

element_at(X,[X|_],1).
element_at(X,[_|L],K) :- K > 1, K1 is K - 1, element_at(X,L,K1).


Another solution: K does not have to be instantiated.

element_at(X,[X|_],1).
element_at(X,[_|L],K) :- element_at(X,L,K1), K is K1 + 1.

Note: The 1st solution never goes into infinite loop,
      but it fails if K is a variable.
```

```
delete(E,L1,L2)   list L2 is list L1 with exactly one instance of E "deleted".

delete(E,[E|Rest],Rest).
delete(E,[X|Rest],[X|Rest2]) :- delete(E,Rest,Rest2).


Notice: we don't have X \= E in the recursive case.
That's because the query

 ?- delete(1,[1,2,1],L).

should say

 L = [2,1];
 L = [1,2];
 no

A side-effect of this is that the query

 ?- delete(1,[1,1],L).

will also say

 L = [1];
 L = [1];
 no

(deleting the second copy of 1).
```

```
reverse(L1,L2)  L2 is the list L1 in reverse order
Precondition:   L1 is instantiated.

reverse([],[]).
reverse([A|X],Z) :- reverse(X,Y), append(Y,[A],Z).

Notice the use of 'append' here.
This solution is ``slow'' - quadratic time.

Draw a Prolog search tree for:

 reverse([1, 2, 3], L).
```

```
permutation(L1,L2)   L2 is some permutation of L1,
                     i.e., it has the same elements, but in any order.
Precondition:        L1 is instantiated.


permutation([],[]).
permutation(L1,[X|Rest2]) :- delete(X,L1,Rest1),    % defined earlier
                             permutation(Rest1,Rest2).


Another Solution:

permutation([],[]).
permutation(L1,[X|R2]) :- append(P1,[X|P2],L1),
                          append(P1,P2,R1),
                          permutation(R1,R2).

Again, notice how 'append' is used here.
```

2 Additional Material

```
reverse(L1,L2)  L2 is the list L1 in reverse order
Precondition:   L1 is instantiated.

Solution 2:

reverse(X,Z) :- reverse_acc(X,[],Z).

reverse_acc([],Y,Y).
reverse_acc([A|X],Y,Z) :- reverse_acc(X,[A|Y],Z).


Y is called an ‘‘accumulator’’ variable.
This solution is ‘‘fast’’ -- linear time.


Draw a search tree for :

  reverse([1,2,3],L)
```

Draw a search tree for :

```
reverse(L,[1,2,3])
```

We get infinite recursion after giving out the answer.

Thus, we need a precondition:
  the first list is instantiated to a list of known length,
  i.e., the tail is [], and not an uninstantiated variable.

The following implementation works with any input:

```
reverse(L,R) :- knownlength(L), reverse_acc(L,[],R).
reverse(L,R) :- \+knownlength(L), reverse_acc(R,[],L).

knownlength(L) :- \+var(L),
 (L=[]; L=[_|R], knownlength(R)).

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

Draw a search tree for:

```
  reverse(L,[1,2,3])
```

to see that it now works.