Lecture 7: Recursion over lists

append( )

```
append ([ ],  List, List).                                                    (*) /* list is empty
append ([ X | List1], List2, [X | Result]) :- append(List1, List2, Result).   (**) /* list is not empty
```

ex:

```
    ?- append([a,b,c],[d,e,f],Result).          /*find Result=[a,b,c,d,e,f].
    ?- append([a,b,c,], Final, [a,b,c,d,e,f]).  /* find  Final=[d,e,f].
    ?- append(Init, [d,e,f], [a,b,c,d,e,f]).    /* find Init = [a,b,c].
    ?- append(Int, Final, [a,b,c,d,e,f]).       /* find split given list
```

member(X , List)

- membr is true if X occurs somewhere in a List.

Case analysis

Case 1, List is empty, then predicate is false for any element (no rule)

Case 2, List not empty, at least 1 element, then list is;

List = [Head | Tail]. (two sub-cases)

a) Head, of List is X.

member(X, [Head | Tail]) :- Head = X. (*)

b) Head, of List is not X.

member(X, [Head | Tail]) :- not Head, member(X, Tail). (*)

ex:

```
        ?- member(b, [a,b,c])           /* yes
        ?- member(X, [a,b,c])           /* returns a,b,c
```

replaceFirst(X, Y, L1, L2)

Case analysis

Case 1, Input list L1 is empty, then output L2 is also empty

replaceFirst(X, Y, [ ], [ ]).

Case 2, Input list L1 begins with X, L1 = [X | Tail]. True, L2 begins with Y and tail is identical to Tail

replaceFirst(X, Y, [X | Tail]. [Y| Tail]).

Case 3: L1 not empty, but begins with Z, L1[Z | Tail] not (X = Z) program skips Z and recursively search X in Tail 1

replaceFirst(X, Y, [Z | Tail1], [Z | Tail2]) :-
not X = Z, replaceFirst(X, Y, Tail1, Tail2).

replaceAll(X, Y, L1, L2).

- L2 result of replacing all occurrences of X in L1 by Y

Case analysis

Case 1: **replaceAll(X, Y, [ ], [ ]).**

Case 2: **replaceAll(X, Y, [X | Tail1], [Y | Tail2]) :-**
replaceAll(X, Y, Tail1, Tail2).

Case 3: r**eplaceAll(X, Y, [Z | L1], [Z | L2]) :-**
not X = Z, replaceAll(X, Y, L1, L2).

ex:

? - replaceAll(p, n, [p,a,p,a], L).
L = m,a,m,a

sum (L, S)

- true if S is sum of list

Case analysis

Case 1: input is empty, S is 0
sum([ ], 0). **(*)**

Case 2: input has 1 el, sum = el.
sum([X1] , X1).

Case 3: input has more than 1, X1, X2… , sum = X1 + X2…
sum([X1, X2..] S) :- sum([X1, X2, M)]. S is X3+M

revision
sum([Head | Tail], S) :- sum(Tail, M), S is Head + M **(**)**

length(List, N)

- true if N is num of el. in List

Case analysis

Case 1: input is empty, length is 0.
length([ ], 0). **(*)**

Case Z: input increases, length increases
length([X1, X2, X3], L) :- length([X1, X2)], M) L is M + 1

revision
length([Head | Tail]), L) :- length(Tail, M), L is M+1 **(**)**

Lecture 7: Recursion over lists

append( )

append ([ ], List, List). (*) /* list is empty
append ([X | List1], List2, [X | Result]) :- append(List1, List2, Result). (**) /* list is not empty
ex:

?- append([a,b,c],[d,e,f],Result). /*find Result=[a,b,c,d,e,f].
?- append([a,b,c,], Final, [a,b,c,d,e,f]). /* find Final=[d,e,f].

?- append(Init, [d,e,f], [a,b,c,d,e,f]).   /* find Init = [a,b,c].

?- append(Int, Final, [a,b,c,d,e,f]).   /* find split given list

member(X , List)

- membr is true if X occurs somewhere in a List.

Case analysis

Case 1, List is empty, then predicate is false for any element (no rule)

Case 2, List not empty, at least 1 element, then list is;

List = [Head | Tail]. (two sub-cases)

a) Head, of List is X.

member(X, [Head | Tail]) :- Head = X. (*)

b) Head, of List is not X.

member(X, [Head | Tail]) :- not Head, member(X, Tail). (*)

ex:

?- member(b, [a,b,c])   /* yes

?- member(X, [a,b,c])   /* returns a,b,c

replaceFirst(X, Y, L1, L2)

Case analysis

Case 1, Input list L1 is empty, then output L2 is also empty

replaceFirst(X, Y, [ ], [ ]).

Case 2, Input list L1 begins with X, L1 = [X | Tail]. True, L2 begins with Y and tail is identical to Tail

replaceFirst(X, Y, [X | Tail]. [Y| Tail]).

Case 3: L1 not empty, but begins with Z, L1[Z | Tail] not (X = Z) program skips Z and recursively search X in Tail 1

replaceFirst(X, Y, [Z | Tail1], [Z | Tail2]) :-
not X = Z, replaceFirst(X, Y, Tail1, Tail2).

replaceAll(X, Y, L1, L2).

- L2 result of replacing all occurrences of X in L1 by Y

Case analysis

Case 1: **replaceAll(X, Y, [ ], [ ]).**

Case 2: **replaceAll(X, Y, [X | Tail1], [Y | Tail2]) :-**
replaceAll(X, Y, Tail1, Tail2).

Case 3: r**eplaceAll(X, Y, [Z | L1], [Z | L2]) :-**
not X = Z, replaceAll(X, Y, L1, L2).

ex:

? - replaceAll(p, n, [p,a,p,a], L).
L = m,a,m,a

sum (L, S)

- true if S is sum of list

Case analysis

Case 1: input is empty, S is 0

sum([ ], 0). **(*)**

Case 2: input has 1 el, sum = el.

sum([X1] , X1).

Case 3: input has more than 1, X1, X2… , sum = X1 + X2…

sum([X1, X2..] S) :- sum([X1, X2, M)]. S is X3+M

revision

sum([Head | Tail], S) :- sum(Tail, M), S is Head + M **(**)**

length(List, N)

- true if N is num of el. in List

Case analysis

Case 1: input is empty, length is 0.

length([ ], 0). **(*)**

Case Z: input increases, length increases

length([X1, X2, X3], L) :- length([X1, X2)], M) L is M + 1

revision

length([Head | Tail]), L) :- length(Tail, M), L is M+1 **(**)**