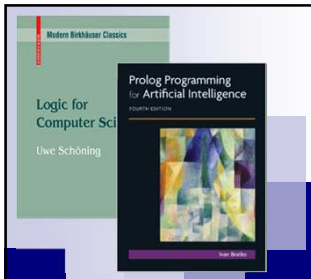

FIU FLORIDA INTERNATIONAL UNIVERSITY

PROLOG (Examples I)

Dr. Antonio L. Bajuelos

FIU School of Computing & Information Sciences

Lists. SWI trace example

conc([], L, L).

conc([X | L1], L2, [X | L3]) :- conc(L1,L2,L3).

member1(X,L) :- conc(L1, [X | L2], L).

?- member1(c,[a,b,c]).

```
[trace] 1 ?- member1(c,[a,b,c]).
   Call: (6) member1(c, [a, b, c]) ? creep
   Call: (7) conc(_G564, [c|_G555], [a, b, c]) ? creep
   Call: (8) conc(_G558, [c|_G555], [b, c]) ? creep
   Call: (9) conc(_G561, [c|_G555], [c]) ? creep
   Exit: (9) conc([], [c], [c]) ? creep
   Exit: (8) conc([b], [c], [b, c]) ? creep
   Exit: (7) conc([a, b], [c], [a, b, c]) ? creep
   Exit: (6) member1(c, [a, b, c]) ? creep
true .
```

2

Lists. SWI trace example

conc([], L, L).

conc([X | L1], L2, [X | L3]) :- conc(L1,L2,L3).

member1(X,L) :- conc(L1, [X | L2], L).

?- member1(a,[a,b,c]).

```
[trace] 2 ?- member1(a,[a,b,c]).
   Call: (6) member1(a, [a, b, c]) ? creep
   Call: (7) conc(_G564, [a|_G555], [a, b, c]) ? creep
   Exit: (7) conc([], [a, b, c], [a, b, c]) ? creep
   Exit: (6) member1(a, [a, b, c]) ? creep
true .
```

3

Lists. SWI trace example

conc([], L, L).

conc([X | L1], L2, [X | L3]) :- conc(L1,L2,L3).

member1(X,L) :- conc(L1, [X | L2], L).

?- member1(d,[a,b,c]).

```
[trace] 3 ?- member1(d,[a,b,c]).
   Call: (6) member1(d, [a, b, c]) ? creep
   Call: (7) conc(_G564, [d|_G555], [a, b, c]) ? creep
   Call: (8) conc(_G558, [d|_G555], [b, c]) ? creep
   Call: (9) conc(_G561, [d|_G555], [c]) ? creep
   Call: (10) conc(_G564, [d|_G555], []) ? creep
   Fail: (10) conc(_G564, [d|_G555], []) ? creep
   Fail: (9) conc(_G561, [d|_G555], [c]) ? creep
   Fail: (8) conc(_G558, [d|_G555], [b, c]) ? creep
   Fail: (7) conc(_G564, [d|_G555], [a, b, c]) ? creep
   Fail: (6) member1(d, [a, b, c]) ? creep
false.
```

4

Lists. SWI trace example

conc([], L, L).

conc([X | L1], L2, [X | L3]) :- conc(L1,L2,L3).

rotate([H|T],R) :- conc(T,[H],R).

?- rotate([a,b,c,d], X).

```
[trace] 1 ?- rotate([a,b,c,d],L).
   Call: (6) rotate([a, b, c, d], _G501) ? creep
   Call: (7) conc([b, c, d], [a], _G501) ? creep
   Call: (8) conc([c, d], [a], _G582) ? creep
   Call: (9) conc([d], [a], _G585) ? creep
   Call: (10) conc([], [a], _G588) ? creep
   Exit: (10) conc([], [a], [a]) ? creep
   Exit: (9) conc([d], [a], [d, a]) ? creep
   Exit: (8) conc([c, d], [a], [c, d, a]) ? creep
   Exit: (7) conc([b, c, d], [a], [b, c, d, a]) ? creep
   Exit: (6) rotate([a, b, c, d], [b, c, d, a]) ? creep
L = [b, c, d, a].
```

5

Lists. SWI trace example

(1) If X is the **head** of the list then the result, after deletion, is the tail of the list.

(2) If X is in the **tail** then it is deleted from there.

del(X, [X | Tail], Tail).

del(X, [Y | Tail], [Y | Tail1]) :- del(X, Tail, Tail1).

?- del(a,[a,b,a],L).

```
[trace] 3 ?- del(a,[a,b,a],L).
   Call: (6) del(a, [a, b, a], _G481) ? creep
   Exit: (6) del(a, [a, b, a], [b, a]) ? creep
L = [b, a] ;
   Redo: (6) del(a, [a, b, a], _G481) ? creep
   Call: (7) del(a, [b, a], _G557) ? creep
   Call: (8) del(a, [a], _G560) ? creep
   Exit: (8) del(a, [a], []) ? creep
   Exit: (7) del(a, [b, a], [b]) ? creep
   Exit: (6) del(a, [a, b, a], [a, b]) ? creep
L = [a, b] ;
   Redo: (8) del(a, [a], _G560) ? creep
   Call: (9) del(a, [], _G563) ? creep
   Fail: (9) del(a, [], _G563) ? creep
   Fail: (8) del(a, [a], _G560) ? creep
   Fail: (7) del(a, [b, a], _G557) ? creep
   Fail: (6) del(a, [a, b, a], _G481) ? creep
false.
```

6

Lists. SWI trace example

del(X, [X | Tail], Tail).

del(X, [Y | Tail], [Y | Tail1]) :- del(X, Tail, Tail1).

?- del(a,L,[1,2,3]).

```
[trace] 4 ?- del(a,L,[1,2,3]).
   Call: (6) del(a, _G480, [1, 2, 3]) ? creep
   Exit: (6) del(a, [a, 1, 2, 3], [1, 2, 3]) ? creep
L = [a, 1, 2, 3] ;
   Redo: (6) del(a, _G480, [1, 2, 3]) ? creep
   Call: (7) del(a, _G557, [2, 3]) ? creep
   Exit: (7) del(a, [a, 2, 3], [2, 3]) ? creep
   Exit: (6) del(a, [1, a, 2, 3], [1, 2, 3]) ? creep
L = [1, a, 2, 3] ;
   Redo: (7) del(a, _G557, [2, 3]) ? creep
   Call: (8) del(a, _G560, [3]) ? creep
   Exit: (8) del(a, [a, 3], [3]) ? creep
   Exit: (7) del(a, [2, a, 3], [2, 3]) ? creep
   Exit: (6) del(a, [1, 2, a, 3], [1, 2, 3]) ? creep
L = [1, 2, a, 3] ;
   Redo: (8) del(a, _G560, [3]) ? creep
   Call: (9) del(a, _G563, []) ? creep
   Exit: (9) del(a, [a], []) ? creep
   Exit: (8) del(a, [3, a], [3]) ? creep
   Exit: (7) del(a, [2, 3, a], [2, 3]) ? creep
   Exit: (6) del(a, [1, 2, 3, a], [1, 2, 3]) ? creep
L = [1, 2, 3, a] ;
   Redo: (9) del(a, _G563, []) ? creep
   Fail: (9) del(a, _G563, []) ? creep
   Fail: (8) del(a, _G560, [3]) ? creep
   Fail: (7) del(a, _G557, [2, 3]) ? creep
   Fail: (6) del(a, _G480, [1, 2, 3]) ? creep
false.
```


Lists. SWI trace example

- **S** is a **sublist** of **L** if
  - (1) **L** can be decomposed into two lists, **L1** and **L2**, and
  - (2) **L2** can be decomposed into two lists, **S** and some **L3**

sublist(S,L) :- conc(L1,L2,L), conc(S,L3,L2).

?- sublist(S,[a,b,c]).

```
[debug] 6 ?- sublist(S,[a,b,c]).
S = [] ;
S = [a] ;
S = [a, b] ;
S = [a, b, c] ;
S = [] ;
S = [b] ;
S = [b, c] ;
S = [] ;
S = [c] ;
S = [] ;
false.
```

Lists. Example #1

- **Find the last element of a list.**

  Example:
  ?- my_last(X,[a,b,c,d]).
  X = d

- **Solution:**

```
% P01 (*): Find the last element of a list

% my_last(X,L) :- X is the last element of the list L
%    (element,list) (?,?)

% Note: last(?Elem, ?List) is predefined

my_last(X,[X]).
my_last(X,[_|L]) :- my_last(X,L).
```

9

Lists. Example #2

- **Find the Kth element of a list.**

  The first element in the list is the element #1.
  Example:
  ?- element_at(X,[a,b,c,d,e],3).
  X = c

- **Solution:**

```
% P03 (*): Find the K'th element of a list.
% The first element in the list is number 1.

% element_at(X,L,K) :- X is the K'th element of the list L
%    (element,list,integer) (?,?,+)


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

10

Lists. Example #3

- **Eliminate consecutive duplicates of list elements.**
- If a list contains repeated elements they should be replaced with a single copy of the element. The order of the elements should not be changed.

  Example:
  ?- compress([a,**a,a,a**,b,c,**c,**a,**a,**d,e,**e,e,e**],X).
  X = [a,b,c,a,d,e]

- **Solution:**

```
% P08 (**): Eliminate consecutive duplicates of list elements.

% compress(L1,L2) :- the list L2 is obtained from the list L1 by
%    compressing repeated occurrences of elements into a single copy
%    of the element.
%    (list,list) (+,?)

compress([],[]).
compress([X],[X]).
compress([X,X|Xs],Zs) :- compress([X|Xs],Zs).
compress([X,Y|Ys],[X|Zs]) :- X \= Y, compress([Y|Ys],Zs).
```

Lists. Example #4

- **Duplicate the elements of a list.**

  Example:
  ?- dupli([a,b,c,c,d],X).
  X = [a,a,b,b,c,c,c,c,d,d]

- **Solution:**

```
% P14 (*): Duplicate the elements of a list

% dupli(L1,L2) :- L2 is obtained from L1 by duplicating all elements.
%    (list,list) (?,?)

dupli([],[]).
dupli([X|Xs],[X,X|Ys]) :- dupli(Xs,Ys).
```

Lists. Example #5

- **Split a list into two parts; the length of the first part is given.**

  Example:
  ?- split([a,b,c,d,e,f,g,h,i,k],3,L1,L2).
  L1 = [a,b,c]
  L2 = [d,e,f,g,h,i,k]

- **Solution:**

```
% P17 (*): Split a list into two parts

% split(L,N,L1,L2) :- the list L1 contains the first N elements
%    of the list L, the list L2 contains the remaining elements.
%    (list,integer,list,list) (?,+,?,?)

split(L,0,[],L).
split([X|Xs],N,[X|Ys],Zs) :- N > 0, N1 is N - 1, split(Xs,N1,Ys,Zs).
```

13

Lists. Example #6

- **Summing elements of a list of numbers**

  Example:
  sumlist([2,4,6,8,10],S).
  S = 30

- **Solution:**

```
sumlist([],0).
sumlist([H|T],N) :- sumlist(T,N1), N is N1+H.
```

14

Lists. Example #7

- **Remove the Kth element from a list.**

  Example:
  ?- remove_at(X,[a,b,c,d],2,R).
  X = b
  R = [a,c,d]

- **Solution:**

```
% The first element in the list is number 1.

% remove_at(X,L,K,R) :- X is the K'th element of the list L; R is the
%    list that remains when the K'th element is removed from L.
%    (element,list,integer,list) (?,?,+,?)

remove_at(X,[X|Xs],1,Xs).
remove_at(X,[Y|Xs],K,[Y|Ys]) :- K > 1,
   K1 is K - 1, remove_at(X,Xs,K1,Ys).
```

15

Lists. Example #8

- **Insert an element at a given position into a list.**

  Example:
  ?- insert_at(alfa,[a,b,c,d],2,L).
  L = [a,alfa,b,c,d]

- **Solution:**

```
% insert_at(X,L,K,R) :- X is inserted into the list L such that it
%    occupies position K. The result is the list R.
%    (element,list,integer,list) (?,?,+,?)

insert_at(X,L,K,R) :- remove_at(X,R,K,L).
```

16

Lists. Example #10

- **Create a list containing all integers within a given range.**

  Example:
  ?- range(4,9,L).
  L = [4,5,6,7,8,9]

- **Solution:**

```
% range(I,K,L) :- I <= K, and L is the list containing all
%    consecutive integers from I to K.
%    (integer,integer,list) (+,+,?)

range(I,I,[I]).
range(I,K,[I|L]) :- I < K, I1 is I + 1, range(I1,K,L).
```