

Prolog Lecture 2 examples

Lecturer: Anwar Mamat

Disclaimer: *These notes may be distributed outside this class only with the permission of the Instructor.*

1.1 Prolog code examples

1.1.1 Get the first element of a list

Listing 1: return first element

```
1 first ([], []).  
2 first ([H|_], H).
```

1.1.2 Get the last element of a list

Listing 2: return last element

```
1 last ([], []).  
2 last ([X], X).  
3 last ([_|T], R):-last(T, R).
```

1.1.3 Sum of a list

Listing 3: sum of a element

```
1 sum ([], 0).  
2 sum ([H|T], R):-sum(T, R2), R is H + R2.
```

1.1.4 Sum of a list (Tail recursive)

Listing 4: sum of a element

```
1 sum2 ([], Acc, Acc).  
2 sum2 ([H|T], Acc, L2):-sum2(T, Acc, L), L2 is L+H.
```

1.1.5 Add 1 to each member of the list

Listing 5: Add one

```

1 add1 ([], []).
2 add1 ([H|T], [H2|T2]):- H2 is H+1, add1(T, T2).
```

1.1.6 Add N to each member of the list

Listing 6: Add N

```

1 addN ([], _, []).
2 addN ([H|T], N, [H2|T2]):- H2 is H+N, addN(T, N, T2).
```

1.1.7 Member of a list

Listing 7: Member of a list

```

1 my_member(X, [X|_]).
2 my_member(X, [_|T]):- my_member(X, T).
```

1.1.8 Duplicate list (Tail recursive)

Listing 8: Duplicate a element

```

1 dup ([], []).
2 dup ([H|T], [H,H|T2]):- dup(T, T2).
3 %dup ([a,b,c], [a,a,b,b,c,c]).
```

1.1.9 Length of a list (Tail recursive)

Listing 9: length of a element

```

1 my_length ([], 0).
2 my_length ([_|T], R):- my_length(T, R2), R is R2+1.
```

1.1.10 Reverse a list (Tail recursive)

Listing 10: reverse a list

```

1 my_reverse(X,F) :- rev_helper(X, [], F).
2 rev_helper ([], A, A).
3 rev_helper ([H|T], A, F) :-
4     rev_helper(T, [H|A], F).
```

1.1.11 Merge 2 Lists

Listing 11: merge 2 sorted list into 1 sorted list

```

1 merge(Xs, [], Xs).
2 merge([], Ys, Ys).
3 merge([X|Xs], [Y|Ys], [X|Zs]) :- X < Y, merge(Xs, [Y|Ys], Zs).
4 merge([X|Xs], [Y|Ys], [X,Y|Zs]) :- X = Y, merge(Xs, Ys, Zs).
5 merge([X|Xs], [Y|Ys], [Y|Zs]) :- X > Y, merge([X|Xs], Ys, Zs).

```

1.1.12 Merge Sort

Listing 12: merge sort

```

1 merge(X, [], X) :- !.
2 merge([], X, X) :- !.
3 merge([H1|T1], [H2|T2], [H1|T3]) :- H1 < H2, merge(T1, [H2|T2], T3).
4 merge([H1|T1], [H2|T2], [H2|T3]) :- H1 > H2, merge([H1|T1], T2, T3).
5 merge([H1|T1], [H2|T2], [H1,H2|T3]) :- merge(T1, T2, T3).
6
7 halve([], [], []).
8 halve([H], [H], []).
9 halve([H1,H2|T], [H1|T1], [H2|T2]) :- halve(T, T1, T2).
10
11 divide(L, L1, L2) :- halve(L, L1, L2).
12
13 merge_sort([], []).      % empty list is already sorted
14 merge_sort([X], [X]).    % single element list is already sorted
15 merge_sort(List, Sorted) :-
16   List = [_ , _ | _], divide(List, L1, L2),      % list with at least two elements is divided into
17   merge_sort(L1, Sorted1), merge_sort(L2, Sorted2), % then each part is sorted
18   merge2(Sorted1, Sorted2, Sorted).

```

1.1.13 Bogosort

Listing 13: bogosort

```

1 is_sorted([]).
2 is_sorted([_]).
3 is_sorted([X1,X2|T]) :- X1 <= X2, is_sorted([X2|T]).
4
5 remove(X, [X|L], L).
6 remove(X, [H|L], [H|M]) :- remove(X, L, M).
7
8 permutation([], []).
9 permutation(L, [X|Xs]) :- remove(X, L, Rest), permutation(Rest, Xs).
10
11 mysort(Xs, Ys) :-
12   permutation(Xs, Ys),

```

```
13 | is_sorted(Ys).
```

1.1.14 Palindrome

Listing 14: palindrome

```
1 | palindrome(X):-reverse(X,X).
```

1.1.15 Sublist

Listing 15: sublist

```
1 | suffix(X,Y):-append(_ ,X,Y).  
2 | prefix(X,Y):-append(X,_ ,Y).  
3 | sublist(X,Y):-  
4 |     suffix(X,Z),prefix(Z,Y).
```

References

□