PROLOG-List-Operations-2 – Documentation

Csc 600-01

Keawa Rozet

Code available at: <https://github.com/krozet/PROLOG-List-Operations-2>

Prompt

Write a PROLOG program that includes the following operations with lists:

membership testing (is an element member of a list?)

first element

last element

two adjacent elements

three adjacent elements

append list1 to list2 producing list3

delete element from a list

append element to a list

insert element in a list

compute the length of list

reverse a list

check whether a list is a palindrome

display a list

For each of these operations write your implementation of the operation and show an example of its use. If a predicate already exists (predefined in PROLOG), modify its name (e.g. myappend or append1). Lists to be processed can be created by an auxiliary program, defined as facts, or entered from the keyboard.

What my code does

In my code you will see that everything is very straightforward and relies heavily on recursion. The only note worthy rule to explain is my check\_palindrome rule. I take the two lists and delete the head off of the first list. I then take that same head and search through the second list to see if it is present there, and if it is, I delete it. I repeat this process until either both lists are empty (signaling it was a palindrome), or until a value cannot be found in the second list (signaling it was not a palindrome).

In my examples I showed cases of where all the rules succeed, and sometimes where they fail to prove that they won’t work for close-to-successful situations. In my last example of check\_palindrome you will see that I changed the last value to ‘9’ and it failed the test.

Code

mymember(X, [X|\_]).

mymember(X, [\_|T]) :-

mymember(X, T).

first\_element(X, [X|\_]).

last\_element(X, [X]).

last\_element(X, [\_|T]) :-

last\_element(X, T).

two\_adjacent\_elements(X,Y, [X,Y|\_]).

two\_adjacent\_elements(X,Y, [\_|T]) :-

two\_adjacent\_elements(X,Y,T).

three\_adjacent\_elements(X,Y,Z, [X,Y,Z|\_]).

three\_adjacent\_elements(X,Y,Z, [\_|T]) :-

three\_adjacent\_elements(X,Y,Z,T).

myappend([], H, H).

myappend([H|T1], H2, [H|T3]) :-

myappend(T1, H2, T3).

mydelete(X, [X|T], T).

mydelete(X, [H|T], [H|T1]) :-

mydelete(X, T, T1).

myinsert(X, Y, XY) :-

mydelete(X, XY, Y).

mylength([], 0).

mylength([\_|T], L) :-

mylength(T, L1), L is L1 + 1.

myreverse([], []).

myreverse([H|T], Rev) :-

myreverse(T, RT), myappend(RT, [H], Rev).

check\_palindrome([], []).

check\_palindrome([H|T], P) :-

mydelete(H,P,P1), check\_palindrome(T, P1).

display\_list([]) :- nl.

display\_list([H|T]) :-

write(H), tab(1),

display\_list(T).

Example

> swipl

Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.4)

SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.

Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org

For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- [listops].

true.

?- mymember(4, [1,2,3,4,5,6,7,8,9]).

true ;

false.

?- first\_element(1, [1,2,3]).

true.

?- first\_element(2, [1,2,3]).

false.

?- last\_element(3, [1,2,3]).

true .

?- last\_element(2, [1,2,3]).

false.

?- two\_adjacent\_elements(X,Y, [1,2,3,4,5,6,7,8,9]).

X = 1,

Y = 2 ;

X = 2,

Y = 3 ;

X = 3,

Y = 4 ;

X = 4,

Y = 5 ;

X = 5,

Y = 6 ;

X = 6,

Y = 7 ;

X = 7,

Y = 8 ;

X = 8,

Y = 9 ;

false.

?- three\_adjacent\_elements(X,Y,Z, [1,2,3,4,5,6,7,8,9]).

X = 1,

Y = 2,

Z = 3 ;

X = 2,

Y = 3,

Z = 4 ;

X = 3,

Y = 4,

Z = 5 ;

X = 4,

Y = 5,

Z = 6 ;

X = 5,

Y = 6,

Z = 7 ;

X = 6,

Y = 7,

Z = 8 ;

X = 7,

Y = 8,

Z = 9 ;

false.

?- myappend([1,2,3], [4,5,6], X).

X = [1, 2, 3, 4, 5, 6].

?- mydelete(3, [1,2,3,4,5], X).

X = [1, 2, 4, 5] ;

false.

?- myinsert(1, [2,3,4,5], X).

X = [1, 2, 3, 4, 5] ;

X = [2, 1, 3, 4, 5] ;

X = [2, 3, 1, 4, 5] ;

X = [2, 3, 4, 1, 5] ;

X = [2, 3, 4, 5, 1] ;

false.

?- mylength([1,2,3,4,5,6,7,8,9], X).

X = 9.

?- myreverse([1,2,3,4,5,6,7,8,9], X).

X = [9, 8, 7, 6, 5, 4, 3, 2, 1].

?- check\_palindrome([1,2,3,4,5,6,7,8,9],[3,2,1,6,4,5,7,9,8]).

true .

?- check\_palindrome([1,2,3,4,5,6,7,8,9],[3,2,1,6,4,5,7,9,9]).

false.

?- display\_list([1,2,3,4,5]).

1 2 3 4 5

true.