PART I. Draw proof trees for the following queries. For each predicate give the rules you used to draw the corresponding proof tree.

1. ?- member(2,[1,2,3]).
2. ?- prefix(X,[1,2]).
3. ?- append([1,2],[3,4],X).
4. ?- reverse([1,2],X).

For this part, hand in the definition of the predicates as well as the corresponding proof trees. For reverse, treat append as if it were a built-in relation (i.e., you don’t have to do the proof tree for append when doing reverse).

PART II. Create a file hw9.pl that contains definitions for the following “functions” (relations). Turn in a hardcopy print-out of hw9.pl in class by the due date. Note that many of the following predicates are already defined in Prolog libraries (although with different names). For this assignment, you must write these “from scratch”, without using these built-in predicates, to gain experience using Prolog. For each, you should try different “binding patterns” (i.e., different arguments as inputs versus outputs).

5. Define a relation min_list(Xs,X) that returns the smallest of a given list of values. E.g.: min_list([7,1,9],X) should return X = 1. You can ignore the case when the list is empty.

6. Define a relation sum_list(Xs,X) that adds all numbers of a list. E.g.: sum_list([1,2,3],X) should return X = 6.

7. Define a relation my_elems(Xs,Ys) that takes two lists of values and returns true if all the values in the first list are in the second list. Examples: my_elems [d,b] [a,b,c,d] should return true whereas my_elems([1,2],[0,1,3,4]) should return false. Note you can use member to implement my_elems.

8. Define a relation my_rem_elem(X,Ys,Zs) that takes a value, a list, and another list with all occurrences of the value in the first list removed. Example: my_rem_elem(1,[2,1,3,2,1,4],Zs) should return Zs = [2,3,2,4].

9. Define a relation my_repl(A,B,Xs,Ys) that takes a value A, a value B, a list, and another list with all the A values in the first list replaced by B values. Example: my_repl(2,8,[1,2,3,2],Ys) should return Ys = [1,8,3,8].

10. Define a relation my_elem_sum(X,Xs,N) that takes a value, a list, and the sum of all occurrences of X in Xs. Examples: my_elem_sum(3,[1,2,3,2,3,4,3],N) should return N = 9 and both my_elem_sum(3,[1,2],N) and my_elem_sum(3,[],N) should return N = 0.