Create a file ec2.pl that implements the following API for a binary search tree. Turn in a hardcopy print-out of ec2.pl in class by the due date with examples showing your implementation works, a cover sheet, and a design document. A binary tree should be represented using a node(v, l, r) structure where v is the value of the node, l is the left subtree, and r is the right subtree. The empty tree should be denoted as nil.

1. Define a relation insert(X,T1,T2) that creates a new tree T2 identical to T1 but with the new value X. E.g.: insert(4,nil,T) should return T = node(4,nil,nil) and insert(2,node(4,nil,nil),T) should return T = node(4,node(2,nil,nil),nil).

2. Define a relation make_tree(Xs,T) that creates a tree T from a list of values Xs. Use your insert relation when defining make_tree. Note that you will need to define a helper make_tree/3 relation such that make_tree/2 is defined as:

   make_tree(Xs,T) :- make_tree(Xs,nil,T).

3. Define a relation find(X,T) that returns true if the value X is in the tree T. E.g.: find(2,node(4,node(2,nil,nil),nil)) should return true whereas find(1,node(4, nil,nil)) and find(1,nil) should both return false.

4. Define a relation max(X,Y,Z) such that Z is the larger value of X and Y. E.g.: max(2,3,Z) should return Z = 3.

5. Define a relation height(T,N) that returns the height of the tree T. Note that your height relation should use your max. E.g.: height(nil,N) should return N = 0 and height(node(4,node(2,nil,nil),nil),N) should return N = 2.

6. Define a relation sum_tree(T,N) that takes a tree T of numeric values and returns the sum of their values N.

7. Define a relation inorder(T,Xs) that returns the values in the tree T in sorted order in the list Xs.

8. Define a relation remove(X,T1,T2) that creates a new tree T2 identical to T1 but without the value X. E.g.: remove(2,nil,T) should return T = nil and remove(4, node(4,node(2,nil,nil),nil),T) should return T = node(2,nil,nil). Note you will need to define helper relations to implement this relation.