Assignments
Herman Geuvers

1 Assignment: sorting of binary search trees (assign_treeSort)

- We define the inductive type tree representing binary trees of natural numbers. (See the Coq file on Proofweb.)

  Inductive tree : Set :=
  | leaf : tree

- Define a predicate bst on tree to express that a tree is sorted, i.e. it is a binary search tree (see http://en.wikipedia.org/wiki/Binary_search_tree for introduction to binary search trees). If you are up for a real challenge you can try to make this exercise using some variant of self-balancing search trees, see http://en.wikipedia.org/wiki/Self-balancing_binary_search_tree; but before you do that make sure you can solve the basic version of this exercise!

- Define a function insert that takes a binary search tree and a natural number and inserts the number in the right place in the tree.

- Prove correctness of the insert function that is prove that:
  
  bst t -> bst (insert n t) (for all t:tree, n:nat).

- Define a function sort that takes an arbitrary tree and sorts it, i.e. it transforms it into a binary search tree. Hint: you can define two auxiliary functions, one that stores the elements of a tree in a list and one that builds a binary search tree from the elements of a list.

- Prove that the result of the sort function is always a binary search tree.

- Given the predicate occurs expressing that an element belongs to a tree, prove that the sorted version of a tree contains the same elements as the original one, i.e. prove:
  
  "occurs n t <-> occurs n (sort t)" (for all n:nat, t:tree)

2 Assignment: binary trees (assign_btrees)

- We define the inductive type tree representing binary trees of natural numbers. (See the Coq file on Proofweb.)

  Inductive tree : Set :=
  | leaf : tree

- Define a function treeMin that will return the value of the minimal node in a tree. You may want to use Coq.Arith.Min for the minimum function. Note that every function in Coq needs to be total and you will need to decide what this function should return applied
on an empty tree. One possibility is to use the \texttt{option} type. Check the definition of \texttt{option} by doing \texttt{Print option}.

If you find it too difficult to work with \texttt{option} you may try a simpler variant of this exercise: change the definition of \texttt{tree} in such a way that its leafs, instead of internal nodes, that contain values. This approach is less natural as now it is impossible to obtain an empty tree but this is exactly what makes the definition of \texttt{treeMin} much simpler.

- Given the predicate \texttt{occurs} expressing that an element belongs to a tree, prove correctness of the \texttt{treeMin} function, i.e. prove that:
  
  - the minimal element belongs to the tree and
  - that the values in all nodes are greater or equal than the minimal value.

- Define a predicate \texttt{bst} on \texttt{tree} to express that a tree is sorted, i.e. it is a binary search tree (see \url{http://en.wikipedia.org/wiki/Binary_search_tree} for introduction to binary search trees).

- Define a function \texttt{leftmost} that given a tree will return a value of its leftmost node.

- Prove that the minimal element of a binary search tree is its leftmost node.

- Define a function \texttt{search} that given a binary search tree will check whether a given natural number occurs in the tree. It should use the fact that the tree is a binary search tree so it should look only on one branch of a tree, instead of on all of its nodes.

- Prove that the \texttt{search} function is correct, i.e. prove:

  
  "\texttt{bst t -> (occurs n t <-> search n t)}" (for all \texttt{n:nat, t:tree)"}