Final assignment for
Proving with Computer Assistance (2IF48)

Please read the instructions in the section "Final Assignment" at

<div align="center">

http://www.win.tue.nl/~akoprows/teaching/Coq.

</div>

Few remarks:

- After giving any definition try to evaluate it on some concrete examples to see whether it works as expected. For this you may find the command `Eval compute in t`, which evaluates term `t`, useful.

- You may need to browse Coq standard library http://coq.inria.fr/library and the reference manual http://coq.inria.fr/V8.1/refman/index.html. The former to find definitions and results that may ease your development (some references included in the exercises); in particular you may be interested in modules like `Coq.Arith` (for definitions and properties involving the natural numbers) or `Coq.List` (for definitions and nice notations for lists). The reference manual may be useful to learn new tactics, learn more about tactics you know etc. (unfortunately the reference manual is not very "beginners-friendly").

A. Binary Trees and the minimum function

1. Define a type `Tree` to represent binary trees of natural numbers[1]. 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 that).

2. Define a predicate `In` that given a natural number and a tree indicates whether the tree contains a node with this number. Use this predicate to prove correctness of `treeMin`, that is:

   - that there is a node in the tree with the value returned by `treeMin` and
   - that the values in all nodes of the tree are greater or equal than the value returned by `treeMin`.

3. Define a predicate `Ordered` on `Tree` to express that a tree is a binary search tree. Define a function `leftmost` that given a tree will return a value of its leftmost node.

4. Use those functions to prove that in a binary search tree the minimal element is the leftmost node.

---

[1]For this exercise it is ok if you define tree in such a way that only leaves contain natural numbers and not the internal nodes. This has the strange effect that you cannot make an empty tree (the smallest tree is a leaf with a number) but will make this exercise easier for you (functions `treeMin` and `leftmost` are total then).

B. Search in Binary Search Trees

1. Define a type `Tree` to represent binary trees of natural numbers. Define a predicate `Ordered` on `Tree` to express that a tree is a binary search tree.

2. Define a function `Mirror` that takes a tree and returns its mirror image by recursively swapping the left and the right subtrees at every internal node of the tree. Prove that for every tree `T`, if `Mirror (Mirror T)` is a binary search tree, then also `T` is a binary search tree.

3. Define a function `SlowSearch` that, given an *arbitrary* binary tree and a natural number, checks whether the number occurs in the tree. Then define a function `FastSearch` that, given a *binary search tree* and a natural number, checks more efficiently whether the number occurs in the tree.

4. Prove that `SlowSearch` and `FastSearch` are equivalent for binary search trees.

C. Sorting of Binary Search Trees

1. Define a type `Tree` to represent binary trees of natural numbers. Define a predicate `Ordered` on `Tree` to express that a tree is a binary search tree.

2. 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.[2]

3. Prove the correctness of `Insert`:

    ∀ T: Tree ∀ n: nat, Ordered T -> Ordered (Insert T n).

4. Define a function `Sort` that takes an arbitrary tree and sorts it.[3] Prove that for every tree `Sort` returns a binary search tree.

D. Formalization of the simply-typed λ-calculus ($\lambda_\rightarrow$)

The purpose of this exercise is to work with a formalization of simply typed lambda terms in Coq (a script containing this formalization is provided).

Given the definition of simply typed lambda terms, encoded using so-called De Bruijn indices[4], prove that typing for simply typed lambda calculus is decidable. That is: prove that for any untyped term 't' and context (environment) 'E' *either* there exists a type 'A' such that 'E |- t : A', *or* that this term is not typable in the given context. For more details and explanation, see the comments and definitions in the Coq-script.

---

[2]You may let `Insert T n = T` if `n` already occurs in `T`.

[3]Hint: you can define two auxiliary functions, one that stores the elements of the tree in a list, and one that builds a binary search tree from the elements of a list.

[4]For background information: the original article introducing the De Bruijn indices, is available at: `http://alexandria.tue.nl/repository/freearticles/597549.pdf`.

E. Custom assignment

You are by all means encouraged to come up with the exercise yourself! You can either take your favorite data-structure and prove some simple properties about it or you may try to formalize some mathematical theorem. In either case in order to make sure that your idea is not too difficult (or too easy) for the final assignment it needs to be approved in advance. Please send your proposals to `A.Koprowski@tue.nl`.