

Final assignment for Proving with Computer Assistance (2IF48)

Please read the instructions in the section “Final Assignment” at

<http://www.win.tue.nl/~fdechesn/2IF48/>.

For the exercises, you can may use existing Coq-modules, like `Arith` (for definitions and properties involving the natural numbers) or `List` (for definitions and nice notations for lists).

A. 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 for every binary search tree `T` and natural number `n`

$$\text{SlowSearch } T \ n \leftrightarrow \text{FastSearch } T \ n.$$

B. Sorting of Binary 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.¹
3. Prove the correctness of `Insert`:

$$\forall T : \text{Tree} \ \forall n : \text{nat}, (\text{Ordered } T) \rightarrow (\text{Ordered}(\text{Insert } T \ n)).$$

4. Define a function `Sort` that takes an arbitrary binary tree and sorts it.² Prove that for every binary tree, `Sort` returns a binary search tree:

$$\forall T : \text{Tree}, \text{Ordered}(\text{Sort } T).$$

¹You may let `Insert T n = T` if `n` already occurs in `T`.

²Hint: 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.

C. Tree traversal

1. Define a type `Tree` to represent binary trees of natural numbers. Define a function `Leftmost` that produces the leftmost element of a tree.
2. Define a function `Preorder` that lists the elements of a tree when traversing the tree in preorder ('nLR'). Similarly, define a functions `Postorder` and `Inorder`, that produce lists of the elements in postorder ('LRn') and inorder ('LnR') respectively.
3. Define a function `Mirror` that returns the mirror image of a tree, Prove that

$$\forall T : \text{Tree}, \text{Postorder} (\text{Mirror } T) = \text{rev} (\text{Preorder } T).$$

where `rev` is the function that reverses a list (present in the Coq module for lists).

4. Prove that

$$\forall T : \text{Tree}, \text{head} (\text{Inorder } T) = \text{Leftmost } T.$$

where `head` is the function that produces the first element of a list (present in the Coq module for lists).

D. Formalization of the simply-typed λ -calculus (λ_{\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 indices³, 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 ' $E \vdash t : A$ ', *or* that this term is not typable in the given context. For more details and explanation, see the comments and definition in the Coq-script.

E. Custom assignment You may also propose your own assignment. Proposals could involve, for example, formalizing your favorite data structure (e.g. AVL-trees, red-black trees) and proving a number of its basic properties, or some program verification, or, if you want to formalize some mathematics, proving one or more theorems like " $\sqrt{2}$ is irrational". In order to be accepted as final assignment, proposals must be approved in advance. If you want to do a custom assignment, send your proposal to `f.dechesne@tue.nl`.

³For background information: the original article introducing the De Bruijn indices, is available at: <http://alexandria.tue.nl/repository/freearticles/597549.pdf>.