02157 Functional programming

Michael R. Hansen
DTU Compute
October 31, 2019

Mandatory assignment 3

This is the third of four mandatory assignments in 02157 Functional programming. It is a requirement for exam participation that 3 of the 4 mandatory assignments are approved. The mandatory assignments can be solved individually or in groups of 2 or 3 students.

Acceptance of a mandatory assignment from previous years does NOT apply this year.

- Your solution should be handed in **no later than Thursday, November 7, 2019**. Submissions handed in after the deadline will face an *administrative rejection*.
- You should solve the problem presented on the following two pages and your solution should be contained in a single F# file (*file*`.fsx` or *file*`.fs`). In your solution you are allowed to introduce helper functions; but you must also provide a declaration for each of the required functions, so that it has exactly the type and effect asked for. Each function declaration should be accompanied by a few illustrative test cases.
- Do not use imperative features, like assignments, arrays and so on in your programs. Failure to comply with that will result in an *administrative rejection of your submission*.
- To submit you should upload a single F#-file to Inside under Assignment 3: The file should start with full names and study numbers for all members of the group. If the group members did not contribute equally to the solution, then the role of each student must be explicitly stated.
- Your F# solution must be a complete program that can be uploaded to F# Interactive without encountering compilation errors. Failure to comply with that will result in an *administrative rejection of your submission*.
- Be careful that you submit the right file. A submission of a wrong file will result in an *administrative rejection of your submission*.
- **DO NOT COPY solutions** from others and **DO NOT SHARE your solution** with others. Both cases are considered as fraud and will be reported.

Trees for Expressions

We now consider *expression trees* for simple arithmetic expressions constructed from *identifiers* and *integer constants* using *addition* (or sum) and *let expressions*. A let-expression `Let`(x, e_1, e_2) reads: "let x be defined by e_1 in e_2". We capture such expressions in F# by the following type declaration:

```
type ExprTree =
   | Const of int
   | Ident of string
   | Sum of ExprTree * ExprTree
   | Let of string * ExprTree * ExprTree
```

where `Const` is the constructor for integer constants, `Ident` is the constructor for identifiers, `Sum` is the constructor for addition and `Let` is the constructor for let-expressions.

1. Declare four values of type `ExprTree` using all four constructors.
2. Declare a function that gives an integer list with the constants occurring in an expression tree. State the type of the function in a comment.

The *free identifiers* of an expression tree are (intuitively) the identifiers for which you need values in order to (be able to) compute an integer value for the expression tree. For example:

- `Sum(Const 2, Const 3)` has no free identifier – you can compute the value 5 for this expression tree.
- `"x"` and `"z"` are the free identifiers of `Sum(Ident "x", Ident "z")` as you need values for `"x"` and `"z"` in order to compute an integer value for this expression tree.
- `Let("x", Const 2, Sum(Ident "x", Ident "x"))` has no free identifier – its value 4 can be computed using the given definition for `"x"`.
- The expression tree `Let("x", Sum(Const 2,Ident "x"), Sum(Ident "x",Const 3))` has `"x"` as free identifier. A value for the occurrence of `"x"` in `Sum(Const 2,Ident "x")` is needed in order to compute an integer value for this let-expression.

The *free identifiers* of an expression tree can be defined recursively as follows:

- `Const` n has no free identifier.
- `Ident` x has the string x as it only free identifier.
- The free identifiers of `Sum`(e_1, e_2) consists of the free identifiers of e_1 and the free identifiers of e_2.
- The free identifiers of `Let`(x, e_1, e_2) consists of the free identifiers of e_1 and the identifiers obtained by deleting x from the free identifiers of e_2.

3. Declare a function extracting the set of the free identifiers occurring in an expression tree. State the type of the function in a comment.

4. Declare a function `subst` x n t, where x is a string, n is an integer, and t is an expression tree. The value of the function is the expression tree obtained from t by replacing every free occurrence of the identifier given by x with the constant given by n. State the type of the function in a comment.

5. Show how to extend the type `ExprTree` with a new constructor `App` so that, for example, `App("f", [])`, `App("g", [Const 2])`, `App("h", [Const 2; Ident "x"; Const 3])` and `App("h", [Const 2; Sum(Ident "x", Const 3)])` will be values having type `ExprTree`. Extend your solutions to Questions 2, 3 and 4 to cope with the new constructor.