02157 Functional programming

Michael R. Hansen
DTU Compute
October 9, 2018

Mandatory assignment 2

This is the second 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, October 25, 2018**. Submissions handed in after the deadline will face an *administrative rejection.*
- The assignment has two tasks:

  Task 1: This task has the form of a paper and pencil exercise. You may consider scanning in you hand-written solution to this task and submit it in the form of a pdf-file.

  Tasks 2: This task concerns a programming problem. A solution should have the form of 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 two files to Inside under Assignment 2: a pdf-fil containing the solution to Task 1 and a single F# file (*file*`.fsx` or *file*`.fs`) containing the solutions to Task 2. The files 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 files. 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.

Task 1

The programming problem presented on the next page concerns *terms* defined by the following type declaration:

```
type Term = | V of string | C of int | F of string * Term list
```

This type declaration provides rules for generating values of type `Term`.

1. Formulate these rules in your own words.

The following five values of type `Term` are mentioned in the problem formulation on the next page:

```
V "x"
C 3
F("f0",[])
F("f1",[C 3;F("f0",[])])
F("max",[V "x";C 3])
```

2*a*. Show how your rules can be used to generate each of the above five values.
2*b*. Give figures showing trees for the above five values.

Task 2

Make a solution to the problem presented on the next page.

Terms

In this problem we consider *terms* of the following form:

- A *variable* x is a *term*,
- an *integer constant* c is a *term*, and
- if f is a *function symbol* and $t_1, t_2, \ldots t_n$ are n terms, where $n \geq 0$, then $f(t_1, t_2, \ldots, t_n)$ is a term. The term $f(t_1, t_2, \ldots, t_n)$ is called a *function application*.

We capture terms in F# by the following type declaration:

```
type Term = | V of string | C of int | F of string * Term list
```

where `V` is the constructor for variables, `C` is the constructor for integer constants and `F` is the constructor for function applications. Notice that variables and function symbols are characterized by strings, that is, `V "x"`, `C 3`, `F("f0",[])`, `F("f1",[C 3;F("f0",[])])` and `F("max",[V "x";C 3])` are 5 values of type `Term`.

A term t is called a *ground term* if it does not contain any variables. That is, 3 out of the above 5 terms are ground terms.

1. Declare a function `isGround: Term -> bool` to check whether a term is a ground term.
2. Declare a function `toString: Term -> string` that gives textual representations of terms, as illustrated by the following example:
   ```
   let t6 = F("f3",[F("f2",[C 1; C 2]); F("f1",[V "x"]); F("f0",[])]);;
   toString t6;;
   > val it : string = "f3(f2(1,2),f1(x),f0())"
   ```
   That is, the textual representations of the arguments in *ts* of a function application `F(`f`,`ts`)` are separated by comma (”,”).
3. Declare a function `subst` (for substitution) such that `subst` x t' t is the term obtained from t by replacing every occurrence of `V` x in t with t'. State the type of `subst`.

We say that function symbol f is used with *arity* n when f is applied to n terms in an application. For example, `"f0"`, `"f1"`, `"f2"` and `"f3"` are used with arities $0, 1, 2$ and 3, respectively, in `t6`.

A term t is *illegal* if it contains two (or more) function applications involving a function symbol f, that is used with different arities. Otherwise t is called a *legal* term. All example terms above are legal, whereas `F("g", [C 2; F("g",[])])` is an illegal term because function symbol `"g"` is used twice, one time with arity 2 and one time with arity 0.

4. Declare a function `extractArities: Term -> Map<string,int> option`. The value of `extractArities` t is `None` when t is an illegal term. Otherwise `extractArities` t = `Some` m, where m is a map. The keys of m are the function symbols occurring in t and (f, n) is a entry of m, when f is used with arity n in t. For example, `extractArities t6` gives `Some` m, where m contains 4 entries (`"f0"`,0), (`"f1"`,1), (`"f2"`,2) and (`"f3"`,3).