Functional Programming
Mid-term Exercise

(deadline: December 11, 2014)

Submission of your solution is due on December 11 2014, at 23:59. Submit the OCaml source code (tasks 5, 6, 7) and a PDF report (max 2 pages) in a tgz or zip file via email to your Proseminar supervisor. Use as subject "[FP Proseminar] mid-term task". The OCaml developments can use any standard-library modules, but do not use imperative features. Development is individual.

Report should also contain a section about code execution or usage. Justification of algorithm choice should be highlighted clearly in your report. If you use some external documentation, cite your references in the report.

The task is split into several subtasks, evaluated separately, that concert both practical OCaml programming and theoretical knowledge of the λ-calculus, as well as relate them to other models of computation.

1. Provide a definition of a Turing machine (5 points)

2. Define Turing completeness, and answer the following questions both theoretically (what does the machine or λ-calculus term do) and practically, do you know any such existing encodings. (10 points)

   a) Is it possible to encode a Turing machine in the λ-calculus?

   b) Is it possible to implement the λ-calculus on a Turing machine?

3. Datatypes (10 points)

   a) What are the properties of data available in functional programming?

   b) How to encode a Turing machine with such datatypes?

4. Lambda-calculus terms

   a) Given an alphabet Σ, that contains n symbols, express the symbols as suitable lambda terms (5 points)
   hint: The boolean type is a type that contains two values. Remember the λ-terms for *true* and *false* and the way they allow the definition of the *ite* function. Can you extend this to a type that has three values and an `if-then-else1-else2` expression that allows matching them? What about n symbols?

   b) The tape of a Turing machine can be expressed as two lists, one representing the elements on the left of the current position and one representing the elements on the right. Give the lambda-terms that represent a move to the left and a move to the right. (5 points)

c) The transition function. Assume that the Turing machine has only one state. Define a lambda term that realized a one step transition function: reads the symbol on the tape, and depending on the symbol writes a symbol and moves to the left or right. (5 points)

d) Recursion. Given the one step transition function, how to make a recursive one step transition function in the λ-calculus? (5 points)

5. Write an OCaml program that implements a simplified Turing machine. You can simplify the task by assuming that there is only one regular state and one final state. The Turing machine is given by:

- A size of the alphabet (`int`)
- The transition function for the state. For each alphabet symbol: what symbol to write, whether to move left or right and whether to move to the final state: `(int * bool * bool) list`.
- The tape `(int list * int list)`

The function returns if the Turing machine stops. (10 points).

```
tm : int -> (int * bool * bool) list -> int list * int list -> unit
```

6. Write an OCaml program that beta-reduces a given lambda-terms. You can choose the strategy. (10 points).

```
reduce: lambdaterm -> lambdaterm
```

7. Write an OCaml program that transforms a Turing machine definition given as a transition function to a lambda-term. (15 points)

```
encode : int -> (int * bool * bool) list -> int list -> lambdaterm
```