Tallinn University of Technology

Department of Computer Science

Past exam paper: Autumn 2013

Advanced Programming

ITT8060

Time allowed TWO Hours

Answer ALL FOUR questions

No calculators, mobile phones or other electronic devices capable of storing or retrieving text may be used.

Two A4 pages of handwritten notes are permitted.
The print text book (Real World Functional Programming) is allowed.
DO NOT open the examination paper until instructed to do so

Question 1: True or False

Please circle **T** if the following statement is true and **F** if the statement is false.

a. **(T) (F)** The value of `List.fold (+) -1 [1;2;3;4]` is 9. (2 points)

b. **(T) (F)** Evaluating the expression `([1;2] :> System.Object) :?> int list` will succeed. (2 points)

c. **(T) (F)** For abitrary n, accessing the last element of `[|1 .. n|]` will take linear time (as a function of n). (2 points)

d. **(T) (F)** Evaluating the expression `fst (lazy (1,2))` will fail because of type mismatch. (2 points)

e. **(T) (F)** Taking the head of an empty list will fail at run time. (2 points)

f. **(T) (F)** Evaluating the expression `let x = printfn "hello"; 2` will print `hello` to the screen (2 points)

g. **(T) (F)** The type of `[(1,2,3)]` is `(int * int * int) list`. (2 points)

h. **(T) (F)** The type of `Some (Some 42)` is `int option`. (2 points)

i. **(T) (F)** Evaluating the expression `Option.bind Some None` returns `Some None`. (2 points)

j. **(T) (F)** The expression `let rec t = seq {yield 1; yield! t}` generates an infinite sequence of increasing integers (2 points)

k. **(T) (F)** The expression `List.map (fun n -> n + 2) [1,3,5,7]` returns `[3,3,5,7]` (2 points)

l. **(T)(F)** The type definition

```
type Tree =
          | Leaf of int
          | Node of Tree * Tree
```

defines leaf labelled trees (3 points)

Question 2: Trees

Expressions made of only strings and concatenation can be represented using the following tree data structure:

```
type STree =
  | Val of string
  | Concat of STree * STree
```

a. Define an element of the `STree` type that corresponds to the informal representation `("a" @ "b") @ ("c" @ "d")`. (3 points)

b. Given this function definition:

```
let rec f x =
  match x with
    | Val s -> Val (s.ToUpper())
    | Concat (a,b) -> Concat(a,f b)
```

(i) Evaluate the expression
`f (Concat (Concat (Concat (Val "a", Val "b"), Val "c"), Val "d"))`.
(3 points)

(ii) Give the type of `f`. (3 points)

(iii) Explain in words what the function `f` does. (3 points)

c. Write a function that appends an exclamation mark `"!"` to every string in the tree. (5 points)

d. Write a function `flatten : STree -> string` which concatenates all strings in the tree from left to right, e.g. flattening a tree corresponding to `("a" @ "b") @ ("c" @ "d")` would result in a string "abcd". (5 points)

e. What is the type of the function `g` defined below? (3 points)

```
let rec g f x =
  match x with
   | Val n -> Val (f n)
   | Concat (a,b) -> Concat (g f a,b)
```

Question 3: Lists

a. Define a function `first : 'a list -> 'a` which returns the first element of a list. Define this function using pattern matching. If this list is empty return `failwith "oops"`. (5 points)

b. Given the following function `g`:

```
let rec g x y =
  match y with
    | a :: b :: cs  -> a :: x b :: g x cs
    | d             -> d
```

(i) Evaluate the expression `g (fun x -> x+1)  [1..4]`. (3 points)

(ii) Give the type of `g`. (3 points)

(iii) Explain in words what the function `g` does. (3 points)

c. The function `zip` is supposed to create a list of pairs from the pair of lists given as arguments.

```
let zip (xs,ys) =
  match xs,ys with
     | []      , _         -> []
     | x :: xs', y :: ys' -> (x , y) :: zip (xs,ys)
```

(i) Identify all the bugs. (6 points)

d. Given a function `min : int -> int -> int` which returns the minimum of two arguments, each in the range -100 to 100, write a function `minList : int list -> int` that computes the minimum of a list of integers. (5 points)

Question 4: Option

There are several 3-valued logics. Kleene 3-valued logic is used in SQL database engines to deal with comparisons involving *null* values.

The behaviour of the 3-valued negation (NOT) can be given as follows:

A	NOT A
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

After noticing that the `option` type adds one value to the set of values of the type it wraps, we decide to use `bool option` type to implement such a logic, with TRUE implemented as `Some true`, FALSE as `Some false` and UNKNOWN as `None`.

a. Write a function that converts from `bool` value to `bool option`. (2 points)

b. Write a function that converts from `bool option` to `bool`
(hint: use `failwith "oops"` in the case of `None`). (2 points)

c. Implement the 3-valued negation function `kleeneNeg : bool option -> bool option` in F# by using pattern matching. (4 points)

d. In Kleene logic the behaviour of the disjunction (OR) function can be given by the following table:

A OR B	A = TRUE	A = UNKOWN	A = FALSE
B = TRUE	TRUE	TRUE	TRUE
B = UNKNOWN	TRUE	UNKNOWN	UNKNOWN
B = FALSE	TRUE	UNKNOWN	FALSE

Implement the Kleene 3-valued logic disjuction as
`kleeneOr : bool option -> bool option -> bool option`. (5 points)

e. Given that the implication in Kleene logic is defined as $A \rightarrow B = NOT(A)\ OR\ B$, implement Kleene implication as
`kleeneImpl : bool option -> bool option -> bool option`. (3 points)

f. In Kleene 3-valued logic it is possible to assign integer values to FALSE = 0, UNKNOWN = 1 and TRUE = 2 and use the built in `min` function to compute the conjunction (AND). For example A AND B = MIN (A,B).

(i) Write a function `kleeneToInt : bool option -> int`. (2 points)

(ii) Write a function `kleeneAnd : bool option -> bool option -> bool option` that computes the conjunction of 2 arguments, and uses the built in `min : int -> int -> int` function. (3 points)

g. Given the following function `g`:

```
let g x = Option.map not x
```

(The type of `Option.map` is `('a -> 'b) -> 'a option -> 'b option` and the type of the Boolean `not` function is `bool -> bool`.)

(i) Give the type of `g`. (2 points)

(ii) Evaluate the expression `g None`. (2 points)