UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

FUNCTIONAL PROGRAMMING AND SPECIFICATION (LEVEL 9)

FUNCTIONAL PROGRAMMING AND SPECIFICATION (LEVEL 10)

Friday 21st May 2010

09:30 to 11:30

Year 3 Courses

Convener: K. Kalorloti
External Examiners: K. Eder, A. Frisch

INSTRUCTIONS TO CANDIDATES

Answer any TWO questions.

All questions carry equal weight.

CALCULATORS MAY NOT BE USED IN THIS EXAMINATION

1. **Save all of your answers to this question in a file** `q1.sml` **and submit that single file when you are finished using the command:** `examsubmit q1.sml`

   Consider the following datatype of "fat lists":

```
datatype 'a fatlist = nil
                    | cons of 'a * 'a fatlist
                    | append of 'a fatlist * 'a fatlist
```

   (a) Define variants of the list functions `length`, `reverse` and `map` for fat lists. Define a function `null : 'a fatlist -> bool` which tells if a fat list is empty (contains no elements) or not. [*7 marks*]

   (b) Define a function `fold : ('a * 'b -> 'b) -> 'b -> 'a fatlist -> 'b` that is analogous to `foldr` on lists. Give a function `f` and a value `v` such that `fold f v` l is `null` l for all fat lists l. Give a function `g` and a value `w` such that `fold g w` l is `length` l for all fat lists l. [*6 marks*]

   (c) Define a function `slim : 'a fatlist -> 'a list` which converts a fat list to an ordinary list of its elements in the same order. Give a function `f` (depending on `g`) and a value `v` such that `slim(fold f v` l`)` = `slim(map g` l`)` for all fat lists l [*6 marks*]

   (d) Is there a function `f` (depending on `g`) and a value `v` such that `fold f v` l = `map g` l for all fat lists l? Either give such an `f` and `v` or else briefly justify why they cannot exist. [*6 marks*]

2. **Save all of your answers to this question in a file `q2.sml` and submit that single file when you are finished using the command: `examsubmit q2.sml`**

   A *bonsai tree* is a binary tree with limited depth. The maximal permitted depth of each bonsai tree is established when it is first created. New nodes are added at the leaves; this fails if it would cause the tree to grow beyond its maximal permitted depth.

   (a) Implement bonsai trees in SML, recalling that ordinary binary trees can be defined by

   ```
   datatype 'a tree = empty | node of 'a * 'a tree * 'a tree
   ```

   Provide a definition of the type of bonsai trees, `'a btree`, and the following functions on bonsai trees:

   `create : int -> 'a btree`
   Create an empty bonsai tree with the given maximal permitted depth.

   `add : 'a * 'a btree -> 'a btree`
   Add an element if there is space; raise an exception if not.
   An empty tree has depth 0, so `add("label",create 0)` will raise an exception while `add("label",create 1)` will succeed.

   `prune : int * 'a btree -> 'a btree`
   Restrict maximal permitted depth to the depth given, removing all growth that is deeper than this.

   `graft : 'a * 'a btree * 'a btree -> 'a btree`
   Let $b1$ and $b2$ be bonsai trees with maximal permitted depth $d1$ and $d2$ respectively. Then `graft`$(a, b1, b2)$ is a bonsai tree with maximal permitted depth `Int.max`$(d1, d2)$ formed by joining $b1$ and $b2$ at their roots with a as label, and pruning as necessary.

   Please include

   ```
   load "Int";
   ```

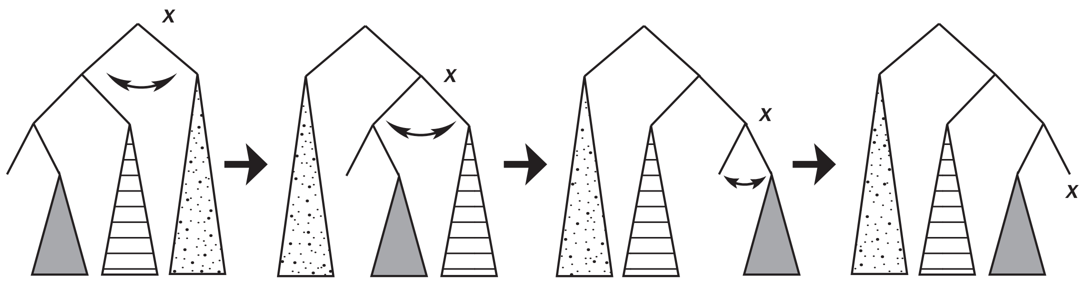
   before the code for `graft` in order to load `Int` from the library, making `Int.max` available. [*15 marks*]

QUESTION CONTINUES ON NEXT PAGE

(b) Add a function

```
balance : 'a btree ->  'a btree
```

that creates a balanced version of a bonsai tree by inserting all of its elements, one by one, into an empty bonsai tree. You can maintain balance of the resulting tree as it is built up by simply using a version of `add` that switches left and right subtrees of each node that it visits in the course of adding an element to the leftmost leaf, like so:



This yields a bonsai tree having the same elements as the original, and the same maximal permitted depth, but with the subtrees at each node having depths that differ by at most 1. (**Note:** It isn't easy to see *why* this algorithm produces a balanced tree, but it does! You don't need to understand why it works in order to write the program.) [*10 marks*]

3. **Your answers to this question should be split into two separate files; see below for details.**

   Consider implementing integer arithmetic from scratch. Some operations are more basic than other operations. For instance:

   addition can be defined in terms of successor ($n \mapsto n+1$), negation and ≤ 0

   subtraction can be defined in terms of negation and addition

   multiplication can be defined in terms of addition, negation, zero, ≤ 0 and successor

   division can be defined in terms of zero, ≤ 0, negation, subtraction and successor

   Here is SML code that implements arithmetic as sketched above. You will find this code in a file named `arith.sml`.

```
type integer = int
val zero = 0
fun succ(n:integer)= n+1
fun neg(n:integer) = ~n
fun leqzero(n:integer) = n<=0
fun output(n:integer) = Int.toString n

fun plus(n,m) =
  if leqzero n andalso leqzero(neg n) then m
  else if leqzero n then plus(succ n,neg(succ(neg m)))
       else plus(neg(succ(neg n)),succ m)
fun minus(n,m) = plus(n,neg m)
fun times(n,m) =
  if leqzero n andalso leqzero(neg n) then zero
  else if leqzero n then neg(times(neg n,m))
       else plus(m,times(neg(succ(neg n)),m))
fun divide(n,m) =
  if leqzero n andalso leqzero(neg n) then zero
  else if leqzero n then neg(divide(neg n,m))
       else if leqzero m then neg(divide(n,neg m))
            else if not(leqzero(minus(m,n))) then zero
                 else succ(divide(minus(n,m),m))
```

QUESTION CONTINUES ON NEXT PAGE

(a) **Save all of your answers to this sub-question in a file** `q3a.sml` **and submit that file when you are finished using the command:** `examsubmit q3a.sml`

Organize this code into a series of SML modules. Start with a structure containing `zero`, `succ`, `neg`, `leqzero` and `output`. Addition, subtraction, multiplication and division should be defined in separate functors, separating each of these function definitions from the definitions of the functions on which it depends. All modules, except substructures, should have explicit signatures, using opaque ascription. A maximum of 75% credit will be given for a solution that uses transparent ascription.

Once all of your modules have been defined, use them to define a structure `Arith` that contains all of the functions mentioned above. The following computation (see `arith.sml`):

```
let val one = Arith.succ Arith.zero
    val two = Arith.succ one
    val four = Arith.plus(two,two)
    val result = Arith.divide(Arith.times(four,four),
                              Arith.minus(four,one))
in Arith.output result
end
```

should produce the string `"5"`. [*15 marks*]

(b) **Save all of your answers to this sub-question in a file** `q3b.sml` **and submit that file when you are finished using the command:** `examsubmit q3b.sml`

Consider the simultaneous use of two different representations of integers. Revise your functors for addition and subtraction to allow an integer represented one way to be added to and subtracted from an integer represented another way. (Do not attempt to revise subsequent modules to use these revised functors.) Again, a maximum of 75% credit will be given for a solution that uses transparent ascription. [*10 marks*]