Programming Languages

Homework 4

Due 2:20 pm, May 27, 2009

1. Read Chapter 3 (The Module System; pp. 45–64) of *Notes on Programming Standard ML of New Jersey*, by Riccardo Pucella. The pdf file is available at
`http://www.cs.cornell.edu/riccardo/prog-smlnj/notes-011001.pdf`

2. (3 points) This programming assignment is a continuation of homework 2, in which you have implemented the denotational semantics of the While programs in Standard ML. Again, we will be using the following datatypes:

```
datatype iexpr = Int     of int
               | Var     of string
               | Plus    of iexpr * iexpr
               | Minus   of iexpr * iexpr
               | Times   of iexpr * iexpr
               | Divide  of iexpr * iexpr

datatype bexpr = Bool      of bool
               | And       of bexpr * bexpr
               | Or        of bexpr * bexpr
               | Not       of bexpr
               | Eq        of iexpr * iexpr
               | Less      of iexpr * iexpr
               | LessEq    of iexpr * iexpr
               | Greater   of iexpr * iexpr
               | GreaterEq of iexpr * iexpr

datatype prog = Empty
              | Assignment  of string * iexpr
              | Sequence    of prog * prog
              | Conditional of bexpr * prog * prog
              | While       of bexpr * prog
```

Now, you are asked to write several ML structures and a ML functor that together construct a modular implementation of the denotational semantics. The implementation is modular in that you can plug-in different implementations of the states, and use different integer arithmetic for the While programs.

You are given the following two ML signatures `DICTIONARY` and `ARITHMETIC`:

```
signature DICTIONARY =
sig
  type 'a state

  val init: 'a -> 'a state
  val modify: 'a state -> string -> 'a -> 'a state
  val lookup: 'a state -> string -> 'a
end

signature ARITHMETIC =
sig
  type t
  val i2t: int -> t
  val t2i: t -> int

  val plus:   t * t -> t
  val minus:  t * t -> t
  val times:  t * t -> t
  val divide: t * t -> t

  val eq:        t * t -> bool
  val less:      t * t -> bool
  val lesseq:    t * t -> bool
  val greater:   t * t -> bool
  val greatereq: t * t -> bool
end
```

Note that the signature `DICTIONARY` describes the interface to a module that implements the states. The purposes of functions `init`, `modify`, and `lookup` shall be obvious to you. Also, signature `ARITHMETIC` implements the necessary arithmetic operations on values of type `t`.

You are asked to write two ML structures that match the signature `DICTIONARY`: one uses functions to represent the states, while other represents states with property lists (with default values). That is, you are asked to complete the following two structures:

```
structure DictByFun: DICTIONARY =
struct
  type 'a state = string -> 'a

  fun init       a = ...
  fun modify s x a = ...
  fun lookup s x   = ...
end

structure DictByList: DICTIONARY =
struct
  type 'a state = 'a * (string * 'a) list

  fun init    i          = ...
  fun modify (i, l) x a = ...
  fun lookup (i, l) x   = ...
end
```

You are also asked to write two ML structures that match the signature `ARITHMETIC`: one uses integer arithmetic and the other one uses natural number arithmetic (which only deals with integer numbers greater than or equal to 0).

```
structure Int: ARITHMETIC =
struct
  type t = int

  fun i2t i = i
  fun t2i i = i

  ...
end


structure Nat: ARITHMETIC =
struct
  type t = int
  fun i2t i = if i < 0 then 0 else i
  fun t2i i = i

  ...
end
```

Then you shall write a functor `Semantics` with the following interface:

```
functor Semantics (structure Dict:  DICTIONARY
                   structure Arith: ARITHMETIC):
  sig
    val e_i : iexpr -> Arith.t Dict.state -> Arith.t
    val e_b : bexpr -> Arith.t Dict.state -> bool
    val c : prog -> Arith.t Dict.state -> Arith.t Dict.state
  end
=
struct
  fun e_i ...

  fun e_b ...

  fun c   ...
end
```

Check that you have all pieces fitting together, again using the following While program:

```
val ex_4_5 = Sequence (Assignment ("x", Int 0),
                       Sequence (Assignment ("y", Int 0),
                                 While (LessEq (Var "x", Var "z"),
                                        Sequence (Assignment ("y", Plus (Var "y", Var "x")),
                                                  Assignment ("x", Plus (Var "x", Int 1))))))
```

and with the following ML code:

```
structure S1 = Semantics (structure Dict  = DictByFun
                          structure Arith = Int)

val all0 = DictByFun.init (Int.i2t 0)
val s0   = DictByFun.modify all0 "z" (Int.i2t 2)

val s = S1.c ex_4_5 s0

val x = Int.t2i (DictByFun.lookup s "x")
val y = Int.t2i (DictByFun.lookup s "y")
val z = Int.t2i (DictByFun.lookup s "z")


structure S2 = Semantics (structure Dict  = DictByList
                          structure Arith = Nat)

val all0' = DictByList.init (Nat.i2t 0)
val s0'   = DictByList.modify all0' "z" (Nat.i2t 2)

val s' = S2.c ex_4_5 s0'

val x' = Nat.t2i (DictByList.lookup s' "x")
val y' = Nat.t2i (DictByList.lookup s' "y")
val z' = Nat.t2i (DictByList.lookup s' "z")
```

3. (3 points) Write an interesting structure that also implements signature `DICTIONARY` (e.g., one that uses binary search trees, or hash tables). Write another structure that implements signature `ARITHMETIC` (e.g., one that uses fractional numbers such as $\frac{1}{2}$, or one that uses modulo arithmetic). Plug-in these two structures to functor `Semantics` and show some examples that they really do work together.

   Also, in Chapter 3, pp. 54-55, of *Notes on Programming Standard ML of New Jersey*, **opaque signature matching** (`:>`) of ML is explained. Show that you understand it by using opaque signature matching in some of your code, and explain why functions `i2t` and `t2i` are needed in structures that implement `ARITHMETIC`.

4. (2 points) Exercise 9.5 (p. 276).

5. (2 points) Exercise 11.4 (p. 330).

1 PLEASE NOTE, NO EXCEPTION

- Homework is due **before the class begins** on May 27. Late homework will not be accepted.
- For this programming assignment, you must hand in **printout of the code, as well as the testing data and result**. Programs must be accompanied by their documentations.
- You are expected to do the homework by yourself. Discussion among peers is encouraged but **copying from others is a shame**.