IT UNIVERSITY BFNP SPRING 2016

Assignment 8 for Tuesday 15 March 2016
Version 1.0 of 2016-03-18 sestoft@itu.dk

These exercises concern computation expressions (or monads) in F#, and more precisely, the expression evaluator example.

You should build on the lecture's example code, found in file http://www.itu.dk/people/sestoft/bachelor/computationexpression.fs

Exercise 1. Extend the expression language and monadic evaluators with single-argument functions such as ABS(e1) which evaluates e1 and produces its absolute value. Do this by adding a new case Prim1 of string * expr to the expr datatype. Create suitable variants of all the monadic evaluators; you should not have to change the monad definitions (OptionBuilder, SetBuilder, TraceBuilder) at all. Abstract out the action of ABS on its argument in new auxiliary functions opEvalOpt1, opEvalSet1 and opEvalTrace1 similar to the existing functions opEvalOpt, opEvalSet and opEvalTrace for two-argument primitives. Try the new evaluators on eg these expressions:

```
let expr10 = Prim1("ABS", Prim("+", CstI(7), Prim("*", CstI(-9), CstI(10))))
let expr11 = Prim1("ABS", Prim("+", CstI(7), Prim("/", CstI(9), CstI(0))))
let expr12 = Prim("+", CstI(7), Prim("choose", Prim1("ABS", CstI(-9)), CstI(10)))
```

Exercise 2. Extend the expression language and the monadic evaluators with a three-argument function such as +(e1, e2, e3) that is basically two applications of "+", as in, +(+(e1,e2),e3). Do this by adding a new constructor Prim3 of string * expr * expr * expr to the expr type.

You may alternatively add a more general facility for functions with n>=1 arguments, such as SUM(e1, ..., en), adding a suitable constructor to the expr type.

Implement evaluation of such three-argument (or multi-argument) constructs in the monadic evaluators.

Exercise 3. Create a new family of evaluation functions optionTraceEval. These evaluators should combine the effect of the original optional evaluator (optionEval) and the original tracing evaluator (traceEval).

This can be done in several ways, for instance corresponding to (A) return type int trace option, for an evaluator that returns no trace if a computation fails; or (B) the result type int option trace, for an evaluator that returns a partial trace up until some computation (eg division by zero) fails.

3.1: Make both a standard explicit version of (A) and a monadic version. You need to create a new monad OptionTraceABuilder, among other things.

3.2: Make both a standard explicit version of (B) and a monadic version. You need to create a new monad OptionTraceBBuilder, among other things.