CS 502: Compiling and Programming Systems

Assignment 1. August 27, 2008

Due Date: September 9, 2008

1 An Interpreter for Mini-ML

For this assignment, you will implement an interpreter for Mini-ML, a simplified subset of Standard ML. The sources provided to you consist of the following directories and files:

1. `Absyn`: contains signatures and implementations for the abstract syntax of mini-ML, and the definition of built-in datatypes (e.g., `intlist, true, false`).
2. `Parser`: provides a lexer and parser for mini-ML generated from specifications fed to lex and yacc.
3. `TypChk`: a mini-ML type checker.
4. `Util`: a collection of utilities including a pretty printer and error handler.
5. `Test`: a directory containing sample test cases.
6. `Doc`: a directory containing this file
7. `interp.sml`: the interpreter.

To start the interpreter, first compile the sources using `CM.make ‘‘sources.cm’’`[1] in SML/NJ, and then evaluate `MiniML.interpreter()` . This will start the main interpreter loop.

There are currently five operations that you can apply on the interpreter:

1. :l *filename* will load the contents of *filename* which must be an absolute path. The contents are then parsed, type-checked, and and the resulting types for all expressions and declarations pretty-printed.
2. :p *expression* will parse *expression* and print the resulting parse tree.
3. :t *expression* will parse and type-check *expression.*
4. :s *var* will print the type of a variable that was previously bound via a **val** or **fun** declaration in this interpreter session.
5. :q quits the interpreter.

[1]If you choose to implement the assignment using another SML implementation besides SML/NJ, you will need to change the `sources.cm` file appropriately to conform to the syntax and specification of package building for that implementation.

If you simply type an expression without any command prefix, the interpreter will parse, type-check, and evaluate the expression.

You are responsible for replacing every call to `Error.runtime` found in `interp.sml` whose string argunent is of the form `"***not implemented: <procedure>***"` with code that provides the missing functionality. You are free to change the signatures of any procedure in `interp.sml` provided you supply appropriate documentation.

Sample Output

Here is the sample output of the reference implementation to give you an idea what a correctly working interpreter should display. You can find the files used in the `Test` directory.

```
MiniML> :l  /Users/suresh/teach/purdue/502-Fall2008/project/interpreter/Test/test-map.mml

Closure(fn (l:intlist) =>
  (case l of Nil => Nil | Cons (x, xs) => Cons (f x, (map f) xs)))

Expression type : ((int *  intlist) -> intlist)

Closure(case x of (0, l) => l | (n, l) => iotaHelper (n - 1, Cons (n, l)))

Expression type : (int -> intlist)

Closure(iotaHelper (n, Nil))

Expression type : intlist

val l = [1 2 3 ]

Expression type : intlist

val m = [2 3 4 ]

MiniML> :l  /Users/suresh/teach/purdue/502-Fall2006/project/interpreter/Test/test-fact.mml

Expression type : ((int *   ref int) -> int)

Closure(case a of
  (0, x) => !x | (n, x) => let val \_ = x :=  n * (!x) in fact (n - 1, x) end )

Expression type :  ref int
```

```
val n = ref 1

Expression type : int

val z = 120

Expression type : ((int *  (int -> int)) -> int)

Closure(case a of
  (0, k) => k 1
  | (n, k) =>
      let val k' = fn (z:int) => fact1 (z, fn (v:int) => k n * v) in k' n - 1 end )

Expression type : int

val z = 120

Expression type : int

val foo = 120

MiniML> :l /Users/suresh/teach/purdue/502-Fall2008/project/interpreter/Test/test-mergesort.m
Expression type : intlist

[2 3 4 6 8 ]

MiniML> :l /Users/suresh/teach/purdue/502-Fall2008/project/interpreter/Test/test-hanoi.mml
Expression type : intlist

[1 3 1 2 3 2 1 3 2 1 2 3 1 3 ]

MiniML> :l /Users/suresh/teach/purdue/502-Fall2008/project/interpreter/Test/test-tak.mml
Expression type : int

10
```