CSC/MAT-220: Discrete Structures

Lab 1: SML Tutorial

Due September 4, 2017

Installing SML/NJ.

Standard ML of New Jersey (abbreviated SML/NJ) is a compiler for the Standard ML '97 programming language with additional libraries, tools, and documentation. To install SML/NJ on your computer, follow the instructions below.

Windows: Download and run the installer from `http://smlnj.cs.uchicago.edu/dist/working/110.81/smlnj-110.81.msi`. This will add an item for SML of New Jersey to your start menu and add the command *sml* that you can use at the command line.

Mac: Download and run the installer from `http://smlnj.cs.uchicago.edu/dist/working/110.81/smlnj-x86-110.81.pkg`. The standard installation location is /usr/local/smlnj, if you changed that location then take note below. Open the terminal and enter the following command:

export PATH="$PATH:/usr/local/smlnj/bin"

Now you can use the command *sml* at the command line.

Using SML/NJ.

To begin a SML interactive session, simply enter the command *sml* at the command line. This will result in a message similar to the one below.

```
[thomas@localhost ~]$ sml
Standard ML of New Jersey v110.81 [built: Fri Aug 18 09:33:46 2017]
-
```

The hyphen indicates that SML is awaiting an expression to evaluate, an equal sign indicates the expression is still in progress. Note that SML is a functional programming language with some interesting features. For one, it consists of expressions to be evaluated, as opposed to statements or commands. Much of the information that follows has been adapted from the work in [1].

The SML Basics.

Computation in ML consists of evaluation of expressions. Each expression has three characteristics:

- It may or may not have a *type*.
- It may or may not have a *value*.
- It may or may not cause an *effect*.

Type: Every expression is required to have at least one type; those that do are said to be well-typed, those that don't are said to be ill-typed. Any ill-typed expression is considered ineligible for evaluation. The type checker determines whether or not an expression is well-typed, rejecting with an error those that are not. For example, *3 + 4;* is well-typed and *3.5 + 4;* is ill-typed.

Value: A well-typed expression is evaluated to determine its value, it is has one. For example, the expression *val x = 2* has a value, whereas the expression *print "Hello World"* does not. If the expression has a value, the form of that value is predicted by its type. For example, the value of the expression $x = y$; (only one equal sign!) is either true or false, since it has type bool.

Effect: Effects include raising an exception, modifying memory, performing input or output, and sending a message on the network. Note that the type of an expression does not indicate what possible effects might take place. The plan is to discuss effects close to the end of the course.

Data Types.

There are many data types in SML, here we review the most common.

- Type name: int
  - Values: ..., ~3, ~2, ~1, 0, 1, 2, 3, ... (note that tilde denotes a negative number in SML!)
  - Operators: +, -, *, /, =, <, ...
- Type name: real
  - Values: 3.14, ~2.17, ...
  - Operators: +, -, *, /, =, <, ...
- Type name: char
  - Values: #"a", #"b"
  - Operators: ord, =, <, ...
- Type name: string

  - Values: “abc”, “123”, ...
  - Operators: ^ , size, =, <, ...

- Type name: bool
  - Values: true, false
  - Operators: if *expr1* then *expr2* else *expr3*, andalso (and), orelse (or), not

Note that some types share the same operators, this is known as *overloading*. For example, in an expression involving addition the type checker attempts to determine which form of addition (fixed point or floating point) should be used. Note that SML does not perform any implicit conversions between types. Therefore, if the expression has mixed types then it is ill-typed.

Functions.

Like all functional programming languages, a key feature of SML is the function. So far, we have the ability to calculate the values of expressions and to bind these values to variables. Functions can be used to abstract the data from a calculation, leaving behind the skeleton of the calculation.

A function in SML is a value of function type of the form $type \rightarrow type'$. The *type* is the domain type, and the $type'$ is the co-domain type. For example, Math.sqrt is a primitive function of type $real \rightarrow real$. With the square root function, we can build a function expression for the fourth root:

fn x : real => Math.sqrt (Math.sqrt x);

This function may be applied to an argument by writing an expression such as

(fn x : real => Math.sqrt (Math.sqrt x)) 16.0;

This calculation proceeds by binding the variable x to the argument 16.0, then evaluating the expression Math.sqrt (Math.sqrt x) in the presence of this binding. When the evaluation is complete, the binding of x is dropped.

Writing the function expression and argument at the same time can become tedious. So, we bind the function expression to a variable as follows.

$$\begin{aligned} &val\ fourthroot\ :\ real\ ->\ real\ = \\ &\quad fn\ x\ :\ real\ =>\ Math.sqrt\ (Math.sqrt\ x); \end{aligned}$$

This notation for defining functions can be cumbersome. Therefore, SML provides a more concise syntax for defining functions.

fun fourthroot (x:real):real = Math.sqrt (Math.sqrt x);

The meaning in both definitions are the same, we are binding *fn x : real => Math.sqrt (Math.sqrt x)* to the variable fourthroot.

Text Editor

You will write your SML code in a text editor of your choice and turn in the assignment to the appropriate dropbox folder. The file you turn in should have your name and lab number in the file name.

Keep in mind that you will want to choose a text editor that has an SML mode built in or one that can be installed, so that your code has color coding to help you identify when you've made a mistake. There are many great options to choose from, and ultimately the choice is yours. However, I do encourage you to consider using a lightweight text editor such as *gedit*.

Assignment.

At the top of the file include a commented line as follows:

(*Your Name, Lab Number, Date*)

Be sure to comment generously throughout, so I can follow your steps.

1. Declare a function *greeting* that returns a greeting formed from a string argument (intended to be someones name). Following the function declaration, print the result of the function with your name as the argument.

2. Declare a function *nand* that takes a pair of boolean variables (x:bool,y:bool) and returns $\neg(x \wedge y)$.

3. Declare a function *min* that takes a triple of int variables (x:int,y:int,z:int) and returns the minimum integer. Make use of the *if then else* operators.

4. Declare a function *express* to test if the expression in Problem 13 of the Chapter 1 Self Test is a tautology. Evaluate the function at the necessary variables to determine if the expression is a tautology, and print out your answer with explanation.

References

[1] Robert Harper, *Programming in Standard ML*, Carnegie Mellon University, 2011.