CSC/MAT-220: Discrete Structures
Lab 2: ML and LaTeX

Due September 22, 2017

Part 1: ML

Overview. At its core, ML consists of expressions to be evaluated. Each expression has three characteristics: type, value, effect. Some examples of types are int, real, char, string, bool, and the functional type. It is important to note that ML does not perform implicit conversion, like Python does. So, for example, *3.5+4* is ill-typed and will result in an error. All well-typed expression is evaluated to determine its value. For example, the expression *2+3;* returns *val it = 5 : int*; while the expression *print "Hello";* returns *val it = () : unit*. Note that () unit is an empty tuple that acts as a trivial placeholder for expressions that are side-effect based. As noted earlier, we will discuss effects later on in this course.

Bindings. Just as in other languages, in ML we can store values in variables and then use these variables in proceeding expressions. However, in contrast to other languages, variables in ML do not vary. It is for this reason that we adjust our terminology and say that a value is bound to a variable. For example, the expression below.

val x = 2;

will bind the value 2 to the variable x. This type of binding is called a *value binding*. In addition, we may bind types, see the expression below.

type count = int;

The power in *type bindings* will become more clear when we have more types to work with.

At this point, we note that the value binding above forces the compiler to implicitly determine the variable's type. We can explicitly declare this for the compiler, see the example below.

val pi : real = 3.14 and e : real = 2.17;

Note that, *and* is a logical operator for evaluating two expressions. Whereas, *andalso* is an operator for combining two boolean variables.

The purpose of a binding is to make a variable or type available for use within a particular scope. In ML, scope is static; meaning, the scope of a binding is determined by text, and not the order of evaluation. In the previous examples we assumed global scope, and in what follows we discuss how to limit scope.

Limiting Scope. The scope of a binding may be limited using *let* and *local* expressions. The form of these expressions are as follows.

let *dec* in *exp*	local *dec* in *dec'*
The scope of the declaration *dec* is limited to the expression *exp*.	The scope of the declaration *dec* is limited to the declaration *dec'*.

To get a slight appreciation for the power in limiting scope, consider the following example.

$$
\begin{aligned}
&val\ m\ :\ int\ =\ 2\\
&val\ r\ :\ int\ =\\
&let\\
&\quad val\ m\ int\ =\ 3\\
&\quad val\ n\ int\ =\ m * m\\
&in\\
&\quad m * n\\
&end\ * m;
\end{aligned}
$$

This expression returns a value of 54 for the variable r. This happens because the binding of m is temporarily overridden during the evaluation of the *let* expression, and restored upon completion of this evaluation.

Functions, Bindings, and Scope Revisited. The function allows us to abstract the calculation of the value of an expression. The functional data type in ML is of the form $type \rightarrow type^{'}$. In Lab 1, we gave several syntax examples for working with functions. By far, the most concise was as follows

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

It is important to note that functions in ML are *first-class*. Meaning, they may be computed as the value of an expression, binded to a variable, and passed as an argument of a function. In the above example, we are in fact binding the function value *fn x : real => Math.sqrt (Math.sqrt x);* to the variable *fourthroot*. Also, we are explicitly declaring the type of the function *fourthroot* as $real \rightarrow real$. Now, consider the following lines of code

$$
\begin{aligned}
&val\ x : real\ = 1.0;\\
&val\ y : real\ =\ 2.0;\\
&fun\ f(x : real) : real\ =\ x + y;\\
&fun\ g(y : real) : real\ =\ x + y;\\
&fun\ h(x : real, y : real, f : real- > real, g : real- > real) : real\ =\\
&\quad let\ val\ x : real\ =\ 3.0\ in\ f(x)\ end * g(y) * f(x);
\end{aligned}
$$

Note that the occurrence of x in the body of the function f refer to the parameter of f, whereas the occurrence of x in the body of the function g refer to the previous val binding. To make things more interesting, the function h has a local val binding which is shadowing the parameter x. Before proceeding to the next paragraph, determine the value of the following expression.[1]

h(˜1.0,3.0,f,g);

Tuples.

Intermission: ML Assignment

Part 2: LaTeX

[1] The answer is 20.0, here is why: during the evaluation of the let expression the val binding of $x = 3.0$ shadows the parameter x, $f(3.0) = 5.0$; once the *let* expression is completed, the parameter values are *x=˜1.0* and $y = 3.0$, and $g(3.0) = 4.0$ and *f(˜1.0)=1.0*.