CSC/MAT-220: Discrete Structures
Lab 3: ML

Due September 22, 2017

Aggregate Data Structures

Last time we discussed the tuple, which is one of the many aggregate data structures featured in ML. If the number of elements in a tuple becomes too large, then it can be difficult to remember which position plays what role. For this reason, it is natural to attach a label to each component of the tuple that mediates access to it. This is exactly what a *record type* is for.

Record Types   A record type has the form $\{lab_1 : type_1, \ldots, lab_n : type_n\}$, where $n \geq 0$, and all of the labels lab_i are distinct. For example, the code below is a record type binding called hyperlink.

$$type\ hyperlink\ =$$
$$\{\ protocol\ :\ string,$$
$$address\ :\ string,$$
$$display\ :\ string\}$$

Furthermore, below is a record value binding called mailto_thc.

$$val\ mailto\_thc\ :\ hyperlink\ =$$
$$\{\ protocol\ =\ \text{``}mailto\text{''},$$
$$address\ =\ \text{``}thcameron@davidson.edu\text{''},$$
$$display\ =\ \text{``}Thomas\ Cameron\text{''}\}$$

As with tuples, we can use pattern matching to extract each element of mailto_thc, as is done below.

$$val\ \{\ protocol{=}x,\ display{=}y,\ address{=}z\ \}\ =\ mailto\_thc$$

Note that the order in the above expression does not matter. What's more, we can use the wild card symbol to extract only the variables of interest.

$$val\ \{\ display{=}x,\ protocol{=}\_,\ address{=}\_\ \}\ =\ mailto\_thc$$

Just as in math notation, the ellipsis can be used to denote patterns in records, as illustrated by the following example.

$$val\ \{\ address{=}x,\ ...\ \}\ =\ mailto\_thc$$

Finally, we note that ML provides a convenient abbreviated form for record pattern matching, in which we can make the labels also act as variables. See the example below.

$$val\ \{\ protocol,\ address,\ display\ \}\ =\ mailto\_thc$$

Multiple Arguments and Multiple Results   As we witnessed in Lab 2, the aggregate data types featured in ML allow for a pleasing regularity in the design of functions with multiple arguments or multiple results. For example, the function below computes the length of a vector in $\mathbb{R}^2$ determined by the point (x_1, x_2).

$$fun\ length(x1{:}real,\ x2{:}real){:}real\ =\ Math.sqrt(x1{*}x1\ +\ x2{*}x2)$$

Here is another example, where the arguments are the coefficients a, b, c of the polynomial $ax^2 + bx + c$ and the result is the roots of the polynomial.

$$fun\ quad\_root(a : real,\ b : real,\ c : real) : real * real\ =$$
$$let$$
$$val\ f1\ =\ \ \tilde{}b/(2.0 * a)$$
$$val\ f2\ =\ \ Math.sqrt(b * b - 4.0 * a * c)/(2.0 * a)$$
$$in$$
$$(f1 + f2, f1 - f2)$$
$$end$$

In Lab 2, we made extensive use of pattern matching to access specific elements of a tuple. As some of you discovered, this is such a common need that ML has a pre-defined *sharp notation* which is defined as follows

$$fun\ \#i\ (\_,\ ...,\ \_,\ x,\ \_,\ ...,\ \_) = x$$

and results in the ith component x.

A similar notation is provided for record selection based on labels and is defined as follows

$$fun\ \#lab\ \{lab{=}x,...\}\ =\ x$$

and results with the component x whose label is lab. Further examples of sharp notation will not be given, since its use is strongly discouraged [1].

Case Analysis

One of the most powerful features of ML is the ability to write functions that perform expressions on a case by case basis. To understand how this is done, we first must define *homogeneous types* and *heterogenous types*. If all values of a type have the same form, then that type is said to be homogeneous; for example, every tuple of type int*real has the same form. Therefore, when type pattern matching against the pair (x:int, y:real) there is no room for ambiguity. In contrast, values that have more than one form are said to be heterogeneous; for example, the value of type int might be 0, 1, or ˜1. Corresponding to each of the values of these types is a pattern that matches only that value; attempting

to match any other value will fail. Below is three examples of pattern matching against values of a heterogenous type.

$$val\ 0\ =\ 1 - 1$$
$$val\ (0, x)\ =\ (2 - 2, \#\text{“}a''\text{”})$$
$$val\ 0\ =\ 2 - 1$$

The first two bindings succeed, the third fails. In the second example, the variable x is bound to the character #"a"; no variables are bound in the first or third examples.

Clausal Function Expressions   The type *bool* is one of the most basic examples of a heterogeneous type: its values are either true or false. Functions may be defined on boolean variables using clausal definitions that match against the patterns true and false. In the example below we define the function *lneg* which returns the logical negation of a boolean variable.

$$val\ lneg\ :\ bool \rightarrow bool\ =$$
$$fn\ true\ =>\ false\ |\ false\ =>\ true$$

The function expression is broken into two cases of the form $pat_1 => exp_1$ and $pat_2 => exp_2$, where both patterns are of type bool and both expressions are of type bool. In general, the clausal function expression has the following form.

$$fn\ pat_1\ =>\ exp_1$$
$$|\ pat_2\ =>\ exp_2$$
$$\vdots$$
$$|\ pat_n\ =>\ exp_n$$

where pat_i has $type_1$ and exp_i has $type_2$. The resulting function has type $type_1 \rightarrow type_2$.

As before, ML provides *fun* for a more concise way of defining a function. However, the *fun* form uses an equal sign to separate the pattern from the expression in a clause, and therefore the notation can become hard to read. Consider the same logical negation function defined using *fun* below.

$$fun\ lneg\ true\ =\ false$$
$$|\ lneg\ false\ =\ true$$

Assignment

Please do each of the following problems and put your code/comments in a file named *yourname_lab3.sml*. In addition, include the following commented line at the top of the file

(* Name, Lab \#, Date *)

I. Repeat examples with context, comment out any examples that were intended to display ill typed expressions

II. Build Complex Record Type

III. Correct quad_root to be able to return complex roots

IV.

References

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