CITS3242: Programming Paradigms

Lab sheet 4: Emulating dynamic types and sequences

Visual F# Tips: Hovering, Documentation, and Jumping to Definitions

- Hovering the mouse over any variable name (including function names) in a program will display the type of that variable and its documentation.
- Define your own documentation by preceding let definition with a comment that starts with: ///
- Right-clicking on a variable name (including function names) will pop up a menu. From here you can jump to the place where the variable is defined. This works for types and modules (like List, Seq) also.
- This also works for most variables/functions and types in the F# libraries, although most are compiled so you'll only see the “external” view of the code. This is often a convenient way to access the documentation for the libraries (although it is not as nicely formatted as the web documentation). E.g., click on List or Seq.
- You can use the back button in the tool bar just below the menus (under “Test”) to return to where you were in your own code before you jumped to a definition.

Lab details

This lab involves two tasks. The first is building a small library that can be used to emulate a dynamically typed language. The second involves some interesting uses of sequences.

1. Suppose we are writing a program that interfaces to a dynamically-typed language, such as Lisp or Python (via Iron Python for .Net). Such languages essentially use just one type, and all operations accept and return this type (hence they are sometimes called monotyped). To mirror this in F# we can use a type like:

   /// A universal type
   type Uni = Iu of int
   | Flu of float
   | Su of string
   | Bu of bool
   | Lu of Uni List
   | Fu of (Uni -> Uni)

   let uMap : Uni -> Uni -> Uni =
     fun (Fu f) (Lu xs) -> Lu (List.map f xs)

Here \textit{uMap} is an example how to implement functions for the universal type: it is a universal version of the map function for lists. With a flexible type like \textit{Uni}, we can program in similar ways to dynamically-typed languages, by applying and pattern matching with the \textit{Uni} constructors: \textit{Iu, Flu, Su, Bu, Lu, Fu}.
a) Write a similar universal version of the filter function. \( u\text{Filter} : \text{Uni} \rightarrow \text{Uni} \rightarrow \text{Uni} \) (Hint: follow \( u\text{Map} \) above, but use \( \text{List.filter} : (\text{a} \rightarrow \text{bool}) \rightarrow \text{\'a list} \rightarrow \text{\'a list} \), for the first argument to filter, define a local function that passes its argument to \( f \) and then does a pattern match on the result. Use the types to guide you.)

b) Convert your function into a universal: \( uu\text{Filter} : \text{Uni} \) In a form like this \( uu\text{Filter} \) could be passed to a program in a dynamically-typed language, which could then make use of it just like other values in that language.

c) Just like dynamically typed languages, we can create a list that in some sense holds different kinds of things (of course, really the elements have the F# type \( \text{Uni} \)). In practice, you usually don't want to use a type like \( \text{Uni} \) for this purpose – it is much better to define your own union type that only includes those things that you really want. The \( \text{Uni} \) type is thus mostly of interest for interfacing with dynamically-typed languages, and also to demonstrate that unions allow you to do everything that you can do in those languages. As an example of programming with such lists, write a function that counts how many strings are contained within a universal value.

\[
\text{val numStrings : Uni} \rightarrow \text{Uni}
\]

d) \([\text{Challenge}]\) Similarly, write a function that "adds up" everything in a list. All types should be converted to floats, as follows: via the float function for int and string, 0.0 for false, 1.0 for true. For lists, everything should be recursively added. For functions in a list, the function should be applied to the sum for the tail of the list when the function is the head.

\[
\text{val addApplyFloat : Uni} \rightarrow \text{Uni}
\]