CSC/MAT-220: Discrete Structures
Lab 6: ML

Due November 3, 2017

Type Inference. Up to this point, our programs have been written in *explicitly typed style*. That is, whenever we have introduced a variable, we have assigned it a type; i.e. every variable in a pattern has a type associated with it. A particularly pleasant feature of ML is that it allows you to omit the type information whenever it can be determined from context. For example, there is no need to give a type to the variable s in the function

$$fn\ s{:}string => s\ \hat{}\ \text{“ ”}$$

since we are using the string concatenation operator. Therefore, we may write this expression as

$$fn\ s => s\ \hat{}\ \text{“ ”}$$

allowing ML to insert ":string" for us. This is called the *principle typing property* of ML: whenever type information is omitted, there is always a most general way to recover the omitted type information. If there is no way to recover the type information, then the expression is ill-typed. Otherwise, there is a best way to "fill in the blanks", which will (almost) always be found by the compiler.

An interesting example, is the identity function:

$$fn\ x => x$$

Since this function merely returns x as the result without performing any computation, there is no constraint on the type of x. Since the function is the same no matter which type is chosen, it is said to be *polymorphic*. Therefore, we say that the type of the identity function is $type \to type$.

There is clearly a pattern here, which can be understood by the notion of a *type scheme*: a type expression involving one or more type variables standing for an unknown. An *instance* of a type scheme is obtained by replacing each of the type variables occurring within the expression with a specific type. For example, the type scheme $a \to a$ has instances $int \to int$, $string \to string$, $(int * int) \to (int * int)$, and $(b \to b) \to (b \to b)$, among infinitely many others. In contrast, it does not have the type $int \to string$ as instance, since we are constrained to replace all occurrences of a type variable by the same type scheme. However, the type scheme $a \to b$ has both $int \to int$ and $int \to string$ as instances.

Polymorphic Definitions.

Overloading.