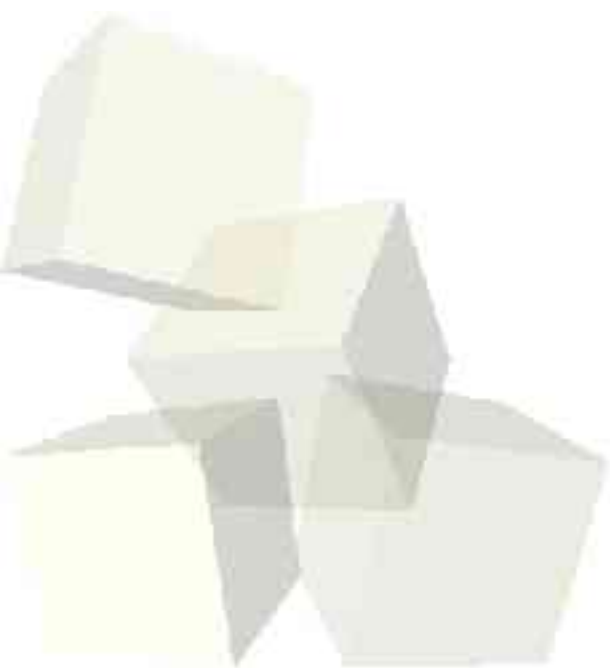





Defining New Types in ML

10-29-2004





Opening Discussion

- Name some common higher order functions. How do we curry functions in ML?
 - In your assignment using a higher order function could be quite helpful if you wanted to try to keep things general. How would you do that?
 - How do you define new types in other languages you know? How did we approximate this in Scheme?
- 



Known Aspects of ML Typing

- So far we know a number of things about the ML typing system. There are a number of basic types: int, real, string, char, and bool.
- We can build product types or tuples that group other types together in a fixed form. Denoted with $*$.
- There are function types that go from one type to another. Denoted as \rightarrow .
- We have also seen two “type constructors”: lists and options. These can be made from any type.



Type Definitions

- The first way we can create types in ML is simply to give a new name to an existing type. We do this with the declaration type.
 - ♦ `type <ident> = <type expression>`
- This is very similar to a typedef in C or C++.
- It can help prevent us from having to type in long types. For example, the following could be used for your items in some situations.
 - ♦ `type item = string * int * real * real;`
- Equality checks don't care about this “declared” type.



Parameterized Type Definitions

- A parameterized type definition gives you flexibility.
 - ♦ `type (<type parameter list>) <ident> = <type expression>`
- The parameter list is a comma separated list of type variables. Remember, type variables begin with '.
- Your book uses the example of a mathematical map. It's really a list of tuples with a domain and range type.
 - ♦ `type ('d,'r) mapping = ('d * 'r) list;`



Datatypes

- We can define our own new types with the datatype keyword. This is followed by a name (typically starting lowercase) called the type constructor which equals a list of data constructors (typically starting uppercase).
- For example, a simple example might be as follows:
 - ◆ `datatype fruit = Apple | Pear | Grape;`
- This says that something of type fruit is either an Apple, Pear, or Grape.



More on Basics of Datatypes

- We can use these types just as we would other types. For example we could write the function
 - ♦ `fun isApple(x) = (x=Apple);`
- Note that this simple use of a datatype is like an enum in C except that it is typesafe. (enums in C are actually ints and for that reason aren't typesafe.)
- The type `fruit` in this example is not an abbreviation for anything, it is a completely new type.




Constructor Expressions in Datatype Definitions

- The full form of a datatype declaration is as follows:
 - ♦ datatype (<type parameter list>) <ident> =
<constructor expr1> | <constructor expr2> | ... |
<constructor exprN>
- The constructor expression has the form <constructor name> with an optional “of <type>”. The of provides a parameter similar to that for an exception.
- Like an exception, the constructor wraps the value and we pull it out by matching a pattern.



Datatypes as Unions

- A datatype can give us something similar to a union in C where something can be one type or another.
 - Your book uses the example of a type that is either a pair or a single.
 - ♦ `datatype ('a,'b) element = P of 'a * 'b | S of 'a;`
 - We can use patterns to determine if the argument is of type P or S and treat it accordingly.
- 



Recursive Datatypes

- Datatypes can also provide us with recursive types because the type following of is a type expression can be the type constructor we are building.
- For example, a general binary tree could have the declaration
 - ◆ `datatype 'label btree = Empty | Node of 'label * 'label btree * 'label btree;`
- Note that the data constructor `Empty` basically serves the role of a null pointer in other languages.



Mutually Recursive Datatypes

- If you remember back, we said that mutually recursive functions (where foo1 calls foo2 and foo2 calls foo1) can be defined in one fun statement with an and between them. The same thing can happen for mutually recursive data types where type1 includes type2 and type2 includes type1.
- Use the datatype keyword once and put an “and” between the definitions.



Code

- You should note that we could easily define the option type ourselves with a datatype.
- Also note that datatypes don't give us any more fundamental power, but they do give us more expressivity and make things safer with better type checking.
- Let's write some code that uses datatypes to store things in ML. Perhaps we could start building an implementation of a set type like what we had done in Scheme.



Minute Essay

- What are your thoughts on datatypes in ML?
They are similar to some constructs in other languages, yet likely different than anything you have seen in any other language.
- Remember that assignment #6 is due today.

