More on Data Types and ML

CS152

Chris Pollett

Nov. 12, 2008.

Outline

- Simple Types
- Type Constructors

- As we talk about the above we'll continue to introduce ML.

Introduction

- Last day, we said types were a way to classify program data, we defined data types as a set of values, and we defined some things we 'd like to be able to do with types like type checking, type inference, etc.
- We are now going to look at the different kinds of types that programming languages have.
- We'll also talk about the specific case of ML.

Simple Types

- **Simple types** are types that have no other structure than their inherent arithmetic or sequential structure.
- Every language comes with a set of **predefined types** which are simple. For example, in C and Java we have types like int, char, float, etc.
- The pre-defined types in ML are:
  - Integers (int). Literals look like: 0, 123, ~12 (notice ~ used for minus sign)
  - Reals (real). Literals look like: 0.1, ~0.77, 1.1E9, ~2.1E~22
  - Booleans (bool). Literals look like: true and false.
  - Strings (string). Literals look like: "", "hi there", "\n", etc. As in C, \ is used to escape characters and we can use them for special characters.

Some Operations on Predefined Types in ML

- For reals and ints we can use the arithmetic operations +, -, *, /, ~.
  - These can be written either infix or prefix:

    ```
    2 *3; (* or *) op* (2,3);
    ```

  - In general, if one define a function:

    ```
    fun my_mult (x, y) = x*y;
    ```

    one can make it prefix using the command:

    ```
    infix my_mult;
    ```

- For strings we can use the operation ^ to concatenate strings.
- For bools we can use the comparison operators: =, <, >, <=, >=, <> (not equals). We can also build up expressions with not, andalso, orelse: 1 = 2 orelse 2<3;

Other kinds of Simple Types

- Many languages support enumerated types.
  - For example, in C one can use the keyword enum: enum Color {Red, Green, Blue};

    Here Red, Green, Blue are ordered as they abbreviate 0, 1, 2
  - In ML, one uses a syntax like:

    datatype Color_Type = Red | Green | Blue;

    And no assumptions about how they are stored can be used. It should be noted ML also supports an analog of typedef: type <identifier> = <type expression>; (* for example, type my_int = int; *)
- Ada supports creating new types with subrange declarations:

  Type Unit_Interval is range 0.0..1.0;

Type Constructors

- Since data types are sets, set operations can be used to construct new types out of existing ones.
- Some operations we can use include product, union, function set, and subset.

Cartesian Product

- Given two sets U,V we can create a new set consisting of ordered pairs from these sets:

  U x V = {(u, v) | u is in U and v is in V}

- In ML, * is used for Cartesian Product. We can write declaration like:

  type my_int_pair = int * int;

  val a:my_int_pair = (2, 3);

- Typically, one has a function to select out of such a product called a **component selector**:

  #1(2, 3); (*returns 2 *) #2(2,3); (*returns 3*)

- The inputs to a function can be typically viewed as a cartesian product.

Records

- Closely related to a cartesian product, is the notion of a record.
- In a product the components of the object are named 1, 2, 3… or 0,1,2.. (depending or the language).
- A **record** is a like a product but where the components can be given meaningful names like ssn, age, etc.
- Roughly, a record corresponds to a C struct:

```
struct Person {char *name; int age;};
```

- ML lets one define and use records using syntax like:

```
type person = {name: string, age:int};
val my_person:person = {name="bob", age=12};
#name(my_person);
```

Variant Records

- Another way of building a new type from two old ones, is to use a union type or variant record.
- Recall we say this in YACC when we dealt with yytype. In C, the syntax for creating new unions might look like:

  ```
  union IntOrReal {int i; double r;} b;
  ```

  Recall, this allocates memory for the larger of the two items and we can store one or the other kind of item in b.
- In ML we can create variant records using datatype:

  ```
  datatype IntOrReal = IsInt of int | IsReal of real;
  val x = IsReal(2.3);
  fun my_print x = case x of
      IsInt(i) => print("integer") |
      IsReal(r ) => print("real");
  ```

Subset

- Some languages like Ada allow you to create types as subsets of existing types.
- For instance, in Ada you might be able to do:

```
Subtype IRInt is IntOrReal(IsInt);
```