Data Types and ML

CS152

Chris Pollett

Nov. 10, 2008.

Outline

- A Brief Introduction to ML
- Data Types and Type Information

ML

- ML stands for Meta Language.
- It is a strongly-typed functional language created by Robert Milnor at the University of Edinburgh in the 1970s.
- It was originally designed as the underlying programming language for theorem provers that made use of the Logic for Computable Functions (LCF).
- Provers (HOL, HOL Lite, Isabelle) that use ML have been used to do verification of computer hardware systems as well as to verify math proofs. See recent AMS article
- ML is closely related to the derivative language OCAML, F# (Microsoft), and is one of the most influential functional programming languages.

Getting started with ML

- For this class, we will be using Standard ML of New Jersey, which can be downloaded from: http://www.smlnj.org/
- Once you have set your path variable correctly you can run ML at the command prompt with a line like:

  sml
- This gets you into interactive mode. The prompt will look like a hyphen. To get out of ML type CTRL-d.
- To read a file into ML type a line like:
  - use "myfile.sml";
- Comments in ML look like (* comment *)

More ML

- At the command prompt you can type in things you would like to evaluate:

  1+2*3;

  – *val it = 7; int*

- Italics denote the response from SML. Here '*it*' is a special variable that receives the value of any expression typed in interactive mode, val stand for value, and int is the type of the result.
- To tell ML to evaluate something you need to terminate your line with a semi-colon. Hitting a new line without a semicolon, will cause ML to prompt you with an = which is ML's way of asking for more input.

ML Type Inference

- Here is an example of defining a function in ML:

  fun fact n = if n = 0 then 1 else n * fact(n - 1);

- After typing this line ML will respond with:

  val fact = fn : int -> int

- This gives a complete typing for this function. It says it is a function from the integers to the integers.
- Notice we didn't have to tell ML what the types of things were: From the assignment n = 0, ML inferred that n had to be an integer and used that to complete the type spec of the function.
- This is called **type inference**. The particular algorithm used by ML is called Hindley-Milner type checking.
- We could have written the above function as:

  fun fact (n: int): int = if n =0 then 1 else n*fact(n -1);

- To use our function we can type fact 5;

Data Types and Type Information

- Typing plays a particularly important role in ML, but it also plays a role in most programming languages.
- So we are going to spend some time now discussing typing in programming languages and also learn ML as we do it.
- To begin program data is often classified according to its **type.**
- A **data type** at its simplest is a set of values. Sometimes people extend the definition to also include as part of the type ,the operations that we allow on it.
- An example type might be an int, and an example operation might be multiplication on int's.

What things do we need to be able to say/do about types?

- **type-checking** - if we use data of a given type in an expression, statement, function, is it being used consistently? I.e., do the types of the inputs to a function math the types of the data given to it?
- **type-inference** - we saw this on the last slide
- **type-constructions** - this is a mechanism to define new instances of a given type. For example, in C we can do: int c; In ML, we can do: val c =3.2; (* uses type inference *)
- **type-declaration** - sometimes we need to be able to build more complicated types out of simple ones.
- **type-equivalence** - if we allow user defined types, the translator often needs some mechanism for telling if two type-declarations define the same type.

Type Systems

- The methods used for constructing types, the type equivalence algorithm, the type inference and correctness rules, are collectively referred to as the **type system** of the programming language.
- If a language definition specifies a complete type system that can be applied statically and guarantees that all data-corrupting errors will be detected at the earliest point, the the language is said to be **strongly typed**. Ex: Ada, ML, Haskell
- If the language allows loopholes then it is said to be **weakly typed**. Ex: C
- Languages without static types systems are usually called **untyped**. Ex: Lisp, Scheme, Perl, PHP Javascript, etc.