Introduction to ML

10-11-2004

Opening Discussion

- One the minute essay before the exam I asked you to tell me what you think are the greatest strengths and weaknesses of Scheme. I want people to share those thoughts.
- There was one thing that no one brought up, but which I feel is critical. It is related to something a few people did mention.
- Today we switch to ML, a newer functional language that is very different from Scheme.

ML on our Machines

- Dr. Konstam installed SML/NJ on all of these machines. This is Standard ML of New Jersey, which is one of the main distributions of ML. You can access it with the command “sml”.
- It is like “scm” in the interface so you might use vi to edit code. Note that vi color codes properly for files ending with .sml and they can be loaded in with the use command.
- To get out of sml, press Ctrl-D.

ML Buzzwords

- Obviously ML is a functional language so it has minimal side effects and it allows the use of higher-order functions.
- ML is also polymorphic and it supports the use of abstract data types. These are more completely supported in ML than in Scheme.
- ML does “rule-based programming” through pattern matching.
- Most importantly in my mind, ML is strongly typed and statically typed. This means that ML checks the type of everything before running.

ML Numeric Expressions

- ML uses infix notation and full expressions end with a ';'.
- Numbers constants look just like in other languages except that '~' is used for negative.
- “true” and “false” are boolean constants.
- The order of operations for +, -, *, and / is what you would expect. Also have div and mod for integers.
- When we get to calling functions, you can use a syntax that looks much more like C than Scheme (later we learn the distinctions).

Strings

- String in ML look a lot like strings in C. We put double quotes around them and can use '\' for “escape characters” that can't be represented other ways.
- For characters ML uses something between Scheme and C. You give a # followed by a string of one character. (i.e. #”a”)
- Strings in ML can be concatenated with ^ used an an infix operator.

Booleans

- ML has the standard infix comparison operators: =, <, >, <=, >=, <>. Note how equality and inequality are different from C.
- Reals can not be compared with = or <>.
- We also have boolean operators in ML. They are “not” which applies to a single boolean as well as “orelse” and “andalso”. The latter two are short-circuit operators.
- not has higher precedence than arithmetic operators so you often need parentheses with it.

If-Then-Else

- The conditional expression in ML is quite straightforward: if *cond* then *expr1* else *expr2*. You must have an else.
- The meaning of this is fairly straightforward. Oddly, we don't use if that much in ML for reasons you will see when we talk about functions in a few days.
- Note that ML is case sensitive so all these things are lower case only. You can use upper case in your function names though.

Type Consistency in ML

- Scheme was very loose and free with types. ML isn't. Expressions are checked for type correctness in ML. Operators can't mix types (1+2.0 is illegal as are #”a”^”ab” and 1/3).
- Also, the types in the then and else of an if must match.
- Types can be coerced with functions like real, floor, ceil, round, trunc, ord, chr, and str.

Variables and Environment

- The naming of values and functions in ML works much like C as combinations of letters and some symbols.
- One difference is that a name starting with ' represents a type in ML.
- In ML we bind values to names with val. It's like define in Scheme. One thing to note is that you can rebind a name to something new later which makes some things much easier than what you did in Scheme.
- The most recent expression is stored in “it”.

Minute Essay

- What do you see so far as the most significant differences between ML and Scheme?
- Remember that assignment #4 if due on Wednesday.