

Language Study: ML

For your consideration:

Rather than teach Prolog or LISP in our “Language Study” course, might we consider ML?

About ML

ML is similar to LISP, with (hd, tl, ::) rather than (car, cdr, cons). Like Prolog, ML supports unification. For a longer (and better) introduction, I found this, from the overview page of *Programming in Standard ML* by Robert Harper, CMU:

Standard ML is a type-safe programming language that embodies many innovative ideas in programming language design. It is a statically typed language, with an extensible type system. It supports polymorphic type inference, which all but eliminates the burden of specifying types of variables and greatly facilitates code re-use. It provides efficient automatic storage management for data structures and functions. It encourages functional (effect-free) programming where appropriate, but allows imperative (effect-ful) programming where necessary. It facilitates programming with recursive and symbolic data structures by supporting the definition of functions by pattern matching. It features an extensible exception mechanism for handling error conditions and effecting non-local transfers of control. It provides a richly expressive and flexible module system for structuring large programs, including mechanisms for enforcing abstraction, imposing hierarchical structure, and building generic modules. It is portable across platforms and implementations because it has a precise definition. It provides a portable standard basis library that defines a rich collection of commonly-used types and routines.

ML at Other Schools

- Pace University covers ML in their graduate compiler and programming language theory courses.
- Carnegie Mellon University teaches ML in several courses, including “Principles of Programming”, “Type Systems”, and others.
- Purdue University teaches ML in “Compiling and Programming Systems”.
- New York University teaches ML in “Honors Programming Languages”.
- Syracuse University uses ML in “The Logic Basis of Computing” and “Formal Specification and Verification of Hardware” courses.

ML Examples

Some list functions:

```
fun length nil = 0
  | length (_::t) = 1 + length t

fun append (nil, l) = l
  | append (h::t, l) = h :: append (t, l)

fun rev nil = nil
  | rev (h::t) = rev t @ [h]

local
  fun rev_helper (nil, a) = a
    | rev_helper (h::t, a) = rev_helper (t, h::a)
in
  fun rev l = rev_helper (l, nil)
end
```

A lexical analyzer for a calculator grammar:

```
datatype lexresult = DIV | EOF | EOS | ID of string | LPAREN |
                  NUM of int | PLUS | PRINT | RPAREN | SUB | TIMES

val linenum = ref 1
val error = fn x => output(std_out, x ^ "\n")
val eof = fn () => EOF
%%
%structure CalcLex
alpha=[A-Za-z];
digit=[0-9];
ws = [\ \t];
%%
\n      => (inc linenum; lex());
{ws}+   => (lex());
"/"     => (DIV);
";"     => (EOS);
"("     => (LPAREN);
{digit}+ => (NUM (revfold (fn(a,r)=>ord(a)-ord("0")+10*r) (explode yytext)
0));
")"     => (RPAREN);
"+"     => (PLUS);
{alpha}+ => (if yytext="print" then PRINT else ID yytext);
"-"     => (SUB);
"*"     => (TIMES);
.       => (error ("calc: ignoring bad character "^yytext); lex());
```

Programming in the ML Style

Thanks to its functional nature and methods of abstraction, ML is a convenient vehicle by which to explore advanced programming techniques. The purpose is to give the students an understanding of advanced programming techniques not easily seen in other languages such as Java or C++, including new ways of reasoning about programs. However, ML's expressive power allows students to ignore its features and instead program in an imperative manner. If they do, they'll miss the whole point. For this reason I will grade them on style. They will be required to program in a functional style, using higher-order abstraction, modularity, typing, and recursion.