CS 345

Introduction to ML

Vitaly Shmatikov

Reading Assignment

- Mitchell, Chapter 5.3-4

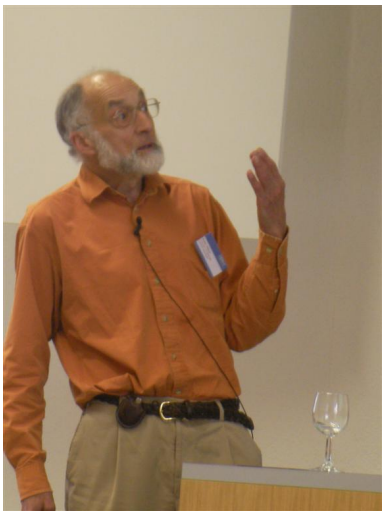
ML

- General-purpose, non-C-like, non-OO language
  - Related languages: Haskell, Ocaml, F#, …
- Combination of Lisp and Algol-like features
  - Expression-oriented
  - Higher-order functions
  - Garbage collection
  - Abstract data types
  - Module system
  - Exceptions
- Originally intended for interactive use

Why Study ML ?

- Types and type checking
  - General issues in static/dynamic typing
  - Polymorphic type inference
- Memory management
  - Static scope and block structure, activation records
  - Higher-order functions
- Control
  - Type-safe exceptions
  - Tail recursion and continuations

History of ML



- Robin Milner
  - Stanford, U. of Edinburgh, Cambridge
  - 1991 Turing Award
- Logic for Computable Functions (LCF)
  - One of the first automated theorem provers
- Meta-Language of the LCF system

slide 5

Logic for Computable Functions

- Dana Scott (1969)
  - Formulated a logic for proving properties of typed functional programs
- Robin Milner (1972)
  - Project to automate logic
  - Notation for programs
  - Notation for assertions and proofs
  - Need to write programs that find proofs
    - Too much work to construct full formal proof by hand
  - Make sure proofs are correct

slide 6

LCF Proof Search

- Tactic: function that tries to find proof

tactic(formula) = { succeed and return proof / search forever / fail }

- Express tactics in the Meta-Language (ML)
- Use type system to facilitate correctness

slide 7

Tactics in ML Type System

- Tactic has a functional type

  tactic : formula → proof

- Type system must allow "failure"

tactic(formula) = { succeed and return proof / search forever / fail and raise exception }

slide 8

Function Types in ML

f : A → B means

for every x ∈ A,

f(x) = { some element y=f(x) ∈ B
       { run forever
       { terminate by raising an exception

In words, "if f(x) terminates normally, then f(x)∈ B."
Addition never occurs in f(x)+3 if f(x) raises exception.

This form of function type arises directly from motivating application for ML. Integration of type system and exception mechanism mentioned in Milner's 1991 Turing Award lecture.

Higher-Order Functions

- Tactic is a function
- Method for combining tactics is a function on functions
- Example:

  $f(tactic_1, tactic_2)$ =

  λ formula. try $tactic_1$(formula)

  else $tactic_2$ (formula)

We haven't seen λ-expressions yet
(think of them as functions for now)

Basic Overview of ML

- Interactive compiler: read-eval-print
  - Compiler infers type before compiling or executing
  - Type system does not allow casts or other loopholes
- Examples

```
- (5+3)-2;
> val it = 6 : int
- if 5>3 then "Bob" else "Fido";
> val it = "Bob" : string
- 5=4;
> val it = false : bool
```


Basic Types

- Booleans
  - true, false : bool
  - if … then … else … (types must match)
- Integers
  - 0, 1, 2, … : int
  - +, * , … : int * int → int and so on …
- Strings
  - "Austin Powers"
- Reals
  - 1.0, 2.2, 3.14159, … decimal point used to disambiguate

Compound Types

- Tuples
  - (4, 5, “noxious”) : int * int * string type
- Lists
  - nil
  - 1 :: [2, 3, 4]
- Records
  - {name = “Fido”, hungry=true}
    : {name : string, hungry : bool} type

Patterns and Declarations

- Patterns can be used in place of variables
  <pat> ::= <var> | <tuple> | <cons> | <record> …
- Value declarations
  - General form: val <pat> = <exp>

```
val myTuple = (“Conrad”, “Lorenz”);
val (x,y) = myTuple;
val myList = [1, 2, 3, 4];
val x::rest = myList;
```

  - Local declarations

```
let val x = 2+3 in x*4 end;
```

Functions and Pattern Matching

- Anonymous function
  - fn x => x+1; like function (…) in JavaScript
- Declaration form

  fun <name> <pat_1> = <exp_1>

  | <name> <pat_2> = <exp_2> …

  | <name> <pat_n> = <exp_n> …
- Examples
  - fun f (x,y) = x+y; actual argument must match pattern (x,y)
  - fun length nil = 0

    | length (x::s) = 1 + length(s);

Functions on Lists

- Apply function to every element of list

  fun map (f, nil) = nil

  | map (f, x::xs) = f(x) :: map (f,xs);

  Example: map (fn x => x+1, [1,2,3]); ⇨ [2,3,4]
- Reverse a list

  fun reverse nil = nil

  | reverse (x::xs) = append ((reverse xs), [x]);

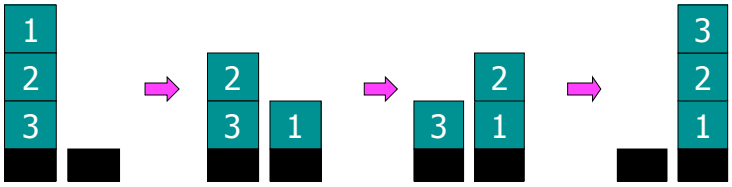
  How efficient is this? Can you do it with only one pass through the list?
- Append lists

  fun append (nil, ys) = ys

  | append (x::xs, ys) = x :: append(xs, ys);

More Efficient Reverse Function

```
fun reverse xs =
    let fun rev(nil, z) = z
    |       rev(y::ys, z) = rev(ys, y::z)
    in rev( xs, nil )
end;
```



Datatype Declarations

- General form

  datatype <name> = <clause> | … | <clause>

  <clause> ::= <constructor> |<constructor> of <type>

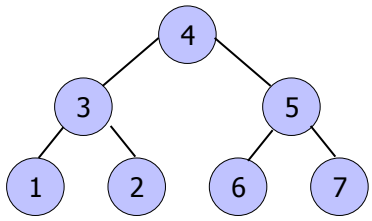
- Examples
  - datatype color = red | yellow | blue
    - Elements are red, yellow, blue
  - datatype atom = atm of string | nmbr of int
    - Elements are atm(“A”), atm(“B”), …, nmbr(0), nmbr(1), ...
  - datatype list = nil | cons of atom*list
    - Elements are nil, cons(atm(“A”), nil), …
      cons(nmbr(2), cons(atm(“ugh”), nil)), ...

Datatypes and Pattern Matching

- Recursively defined data structure

  datatype tree = leaf of int | node of int*tree*tree

  node(4, node(3,leaf(1), leaf(2)),
  node(5,leaf(6), leaf(7))
)



- Recursive function

  fun sum (leaf n) = n
  | sum (node(n,t1,t2)) = n + sum(t1) + sum(t2)

Example: Evaluating Expressions

- Define datatype of expressions

  datatype exp = Var of int | Const of int | Plus of exp*exp;

  Write (x+3)+y as Plus(Plus(Var(1),Const(3)), Var(2))

- Evaluation function

  fun ev(Var(n)) = Var(n)
  | ev(Const(n)) = Const(n)
  | ev(Plus(e1,e2)) = …

  ev(Plus(Const(3),Const(2))) ⇒ Const(5)

  ev(Plus(Var(1),Plus(Const(2),Const(3)))) ⇒
  ev(Plus(Var(1), Const(5))

Case Expression

- Datatype

```
datatype exp = Var of int | Const of int | Plus of exp*exp;
```

- Case expression

```
case e of
    Var(n) =>  ...   |
    Const(n) => .... |
    Plus(e1,e2) => ...
```


Evaluation by Cases

```
datatype exp =  Var of int | Const of int | Plus of exp*exp;

fun ev(Var(n)) = Var(n)
|   ev(Const(n)) = Const(n)
|   ev(Plus(e1,e2)) =  (case ev(e1) of
          Var(n) => Plus(Var(n),ev(e2))      |
        Const(n) => (case ev(e2) of
                Var(m) => Plus(Const(n),Var(m))      |
              Const(m) => Const(n+m)                 |
            Plus(e3,e4) => Plus(Const(n),Plus(e3,e4)) )  |
      Plus(e3,e4) => Plus(Plus(e3,e4),ev(e2)) );
```

ML Imperative Features

- Remember l-values and r-values?
  - Assignment `y := x+3`
    - `y` refers to location (l-value)
    - `x+3` refers to contents (r-value)
- ML reference cells and assignment
  - Different types for location and contents

x : int	non-assignable integer value
y : int ref	location whose contents must be integer
!y	the contents of cell y
ref x	expression creating new cell initialized to x

  - ML form of assignment

y := x + 3	place value of x+3 in location (cell) y
y := !y + 3	add 3 to contents of y and store in location y

slide 23

Reference Cells in ML

- Variables in most languages
  - Variable names a storage location
  - Contents of location can be read, can be changed
- ML reference cells
  - A mutable cell is another type of value
  - Explicit operations to read contents or change contents
  - Separates naming (declaration of identifiers) from "variables"

slide 24

Imperative Examples in ML

- Create cell and change contents

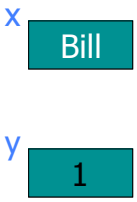
```
val x = ref "Bob";
x := "Bill";
```

- Create cell and increment

```
val y = ref 0;
y := !y + 1;
```

- "while" loop

```
val i = ref 0;
while !i < 10 do i := !i +1;
!i;
```



slide 25

Core ML

- Basic Types
  - Unit
  - Booleans
  - Integers
  - Strings
  - Reals
  - Tuples
  - Lists
  - Records
- Patterns
- Declarations
- Functions
- Polymorphism
- Overloading
- Type declarations
- Exceptions
- Reference cells

slide 26

Related Languages

- ML family
  - Standard ML – Edinburgh, Bell Labs, Princeton, …
  - CAML, OCAML – INRIA (France)
    - Some syntactic differences from Standard ML (SML)
    - Object system
- Haskell
  - Lazy evaluation, extended type system, monads
- F#
  - ML-like language for Microsoft .NET platform
    - *"Combining the efficiency, scripting, strong typing and productivity of ML with the stability, libraries, cross-language working and tools of .NET. "*
  - Compiler produces .NET intermediate language

slide 27