**ML**

CSCI 334
Stephen Freund

---

**Language History**

- Lisp
- Algol 60
- Algol 68
- Pascal
- ML
- Modula

---

**Algol 60**

```
real procedure average(A,n);
real array A; integer n;
begin
  real sum;
  sum := 0;
  for i = 1 step 1 until n do
    sum := sum + A[i];
  average := sum/n
end;
```

---

**ML**

- Combination of Lisp and Algol-like features
- Expression-oriented
- Higher-order functions
- Garbage collection
- Static types
- Abstract data types
- Module system
- Exceptions

---

**Goals in study of ML**

- Types, type checking, polymorphism
- Memory management
- Control Structures

---

**Robin Milner and ML’s Origins**

- Dana Scott, 1969
  - LCF
  - logic for stating theorems about programs
- Robin Milner
  - automated theorem proving for LCF
  - Hard search problem
  - Incomplete: may not find proof
  - ML: meta-language for writing programs (tactics) to find proofs

---

*Dafny Example*
**Tactics**

- Tactics guide search in theorem prover
  - "Try induction to prove T"
  - "Assume X and derive contradiction"
  - "Try A and then B"

- Tactic is *partial* function from formula -> proof
  - finds proof
  - never terminates
  - reports an error

**Language Ideas to Support Tactics**

- Type system
  - guarantees correctness of generated proof

- Exception handling
  - deals with tactics that fail (Turing Award)

- Higher-order functions
  - composition of tactics
    - fun compose(t1, t2) =
      \( \lambda \text{formula. if } t1(\text{formula}) \text{ then } \ldots \) 
      else if \( t2(\text{formula}) \) 

**Running ML**

- Type sml on Unix machines
- System will give you prompt
- Enter expression or declarations to evaluate:
  - 3 + 5;
    val it = 8 : int
  - it * 2;
    val it = 16 : int
  - val six = 3 + 3;
    val six = 6 : int
  - Or "sml < file.ml"

**Defining Functions**

- Example
  - fun succ x = x + 1;
    val succ = fn : int -> int
  - succ 12;
    val it = 13 : int
  - 17 * (succ 3);
    val it = 68 : int;

- Or:
  - val succ = fn x => x + 1;
    val succ = fn : int -> int

**Recursion**

- All functions written using recursion and if.. then.. else (and patterns):
  - fun fact n =
    if n = 0 then 1 else n * fact (n-1);

- if..then..else is an expression:
  - if 3<4 then "moo" else "cow";
    val it = "moo" : string
  - types of branches must match

**Local Declarations**

- fun cylinderVolume diameter height =
  let val radius = diameter / 2.0;
    fun square y = y * y
    in
      3.14 * square(radius) * height
    end;
  val cylinderVolume = fn : real -> real -> real
  - cylinderVolume 6.0 6.0;
    val it = 169.56 : real
Built-in Data Types

- **unit**
  - only value is ()
- **bool**
  - true, false
  - operators not, andalso, orelse
- **int**
  - ..., -2, -1, 0, 1, 2, ...
  - +, -, *, div, mod, abs
  - =, <, <=, etc.

Built-in Data Types

- **real**
  - 3.17, 2.2, ...
  - +, -, *, /
  - <, <=, etc.
  - no conversions from int to real: 2 + 3.3 is bad
  - no equality (test that -0.001 < x-y < 0.001, etc.)
- **strings**
  - "moo"
  - "moo" ^ "cow"

Overloaded Operators

- +, -, etc. defined on both int and real
- Which one to use depends on operands:
  - fun succ x = x + 1
    val succ = fn : int -> int
  - fun double x = x * 2.0
    val double = fn : real -> real
  - fun double x = x + x
    val double = fn : int -> int

Type Declarations

- Can add types when type inference does not work
  - fun double (x:real) = x + x;
    val double = fn : real -> real
  - fun double (x:real) : real = x + x;
    val double = fn : real -> real

Compound Types

- Tuples, Records, Lists
- Tuples
  - (14, "moo", true): int * string * bool
- Functions can take tuple argument
  - fun power (exp,base) =
    if exp = 0 then 1
    else base * power(exp-1,base);
    val power = fn : int * int -> int
  - power(3,2);

Curried Functions (named after Curry)

- Previous power
  - fun power (exp,base) =
    if exp = 0 then 1
    else base * power(exp-1,base);
    val power = fn : int * int -> int
- Curried power function
  - fun cpower exp =
    fn base =>
    if exp = 0 then 1
    else base * cpower (exp-1) base;
    val cpower = fn : int -> (int -> int)
Curried Functions (named after Curry)

- Previous power
  - fun power (exp, base) =
    if exp = 0 then 1
    else base * power(exp - 1, base);
  - val power = fn : int * int -> int

- Curried power function
  - fun cpower exp base =
    if exp = 0 then 1
    else base * cpower (exp - 1) base;
  - val cpower = fn : int -> (int -> int)

Why is this useful?
- fun cpower exp base =
  if exp = 0 then 1
  else base * cpower (exp - 1) base;
- val cpower = fn : int -> (int -> int)

Can define
- val square = cpower 2
  val square = fn : int -> int
- square 3;
  val it = 9 : int

Curried Functions

Records

- Like tuple, but with labeled elements:
  { name="Gus", salary=3.33, id=11 }:
  { name:string, salary:real, id:int };
- Selector operator:
  - val x =
    { name="Gus", salary=3.33, id=11 };  
  - #salary(x);
  - val it = 3.33 : real
  - #name(x);
  - val it = "Gus" : string

Lists

- Examples
  - [1, 2, 3, 4], ["wombat", "numbat"]
  - nil is empty list (sometimes written [])
  - all elements must be same type
- Operations
  - length
    length [1,2,3] \Rightarrow 3
  - @
    [1,2]@[3,4] \Rightarrow [1, 2, 3, 4]
  - ::
    1::[2,3] \Rightarrow [1, 2, 3]
  - map
    map succ [1,2,3] \Rightarrow [2,3,4]

Patterns Matching

- List is one of two things:
  - nil
  - "first elem" :: "rest of elems"
- [1, 2, 3] = 1::[2,3] = 1::2::[3] = 1::2::3::nil

- Can define function by cases
  - fun product (nil) = 1
  - "first elem" :: "rest of elems"
  - [1, 2, 3] = 1::[2,3] = 1::2::[3] = 1::2::3::nil

  fun product (nil) = 1
  | product (x::xs) = x * product (xs);
Patterns on Integers

Patterns on integers

fun listInts 0 = [0]
  | listInts n = n::listInts(n-1);

listInts 3 ⇒ [3, 2, 1, 0];

More on patterns for other data types next time

Many Types Of Lists

1::2::nil : int list
"wombat"::"numbat"::nil : string list

What type of list is nil?
- nil;
val it = [] : 'a list

Polymorphic type
- 'a is a type variable that represents any type
- 1::nil : int list
  "a"::nil : string list

The Length Function

Another Example

fun length (nil) = 0
  | length (x::xs) = 1 + length (xs);

What is the type of length?

How about this one:

fun id x = x;

Polymorphism

fun length (nil) = 0
  | length (x::xs) = 1 + length (xs);
val it = fun 'a list -> int

fun id x = x;
val it = fun 'a -> 'a

Type variable represents any type

Patterns and Other Declarations

Patterns can be used in place of variables

Most basic pattern form
- val <pattern> = <exp>;

Examples
- val x = 3;
- val tuple = ("moo", "cow");
- val (x,y) = tuple;
- val myList = [1, 2, 3];
- val w::rest = myList;
- val v::_ = myList;

Datatype

public static final int NORTH = 1;
public static final int SOUTH = 2;
public static final int EAST = 3;
public static final int WEST = 4;

public move(int x, int y, int dir) {
  switch (dir) {
    case NORTH: ...
    case ...}
}
**Datatype**

datatype Direction =
  NORTH | SOUTH | EAST | WEST;

fun move((x,y),NORTH) = (x,y-1)
  | move((x,y),SOUTH) = (x,y+1)
  ...
  ;