Overview of ML

• ML stands for “metalanguage.”

• Originally conceived for building theorem provers

• First compiler 1977

• Standardized as SML

• Our implementation: Standard ML of New Jersey — SML/NJ
Characteristics of ML

- Functional programming language — But it’s not LISP!

- Interpreted and compiled

- Eager evaluation

- Strongly typed — type inference

- Polymorphic types

- Pattern matching — reminiscent of Prolog

- Garbage collection — Look for the GC messages!
Syntax

- Identifiers: [a-zA-Z][a-zA-Z’]*. E.g., j”’3_z, An_identifier_longer_than_most

- Some keywords:

  abstype and andalso as
  case datatype do else
  end eqtype exception fn
  fun functor handle if
  in include infix infixr
  let local nonfix of
  op open orelse raise
  rec sharing sig signature
  struct structure then type
  val while with withtype

- Default left-to-right processing. Few parentheses
Basic Types

- Booleans
- Numbers: integers and reals
- Strings
- Unit
Booleans

- Type bool

- Operations: not, andalso, orelse

  - not true;
  val it = false : bool

  - true andalso not true;
  val it = false : bool

  - false orelse true;
  val it = true : bool

  - false orelse not true;
  val it = false : bool

  - false orelse not it;
  val it = true : bool
Integers

- Type int

- Operations include +, -, ^, *, div, mod, =, <>, <, >, <=, >=, real

- `~5 + 20 * 4;
val it = 75 : int`

- `100 div 7;
val it = 14 : int`

- `100 mod 7;
val it = 2 : int`

- `27 <> ^11;
val it = true : bool`

- `real 1127;
val it = 1127.0 : real`
Real Numbers

• Type real

• Operations include +, -, ~, *, /, =, <>, <, >, <=, >=, floor

  - 100 / 7;
  stdin:31.5 Error: overloaded variable
  not defined at type symbol: /
  type: int

  - real 100 / real 7;
  val it = 14.2857142857 : real

  - 100.0 / 7.0;
  val it = 14.2857142857 : real

  - floor it;
  val it = 14 : int
Strings

- Type string

- A character is a string of length 1.

- Operations include ^ (concatenation), =, <>

  - "dog" ^ " " ^ "days";
  val it = "dog days" : string

  - it <> "DOG DAYS";
  val it = true : bool
Unit

- Type `unit`

- A data type with a single element, denoted `()`

- Think of the empty list in LISP

- Will be interesting later for defining infinite data structures
Value Declarations

- **Bind an identifier to a value:**

  ```
  val identifier = expression ;
  ```

- **Examples:**

  - `val x = 3 * 11;
  val x = 33 : int`

  - `val y = x mod 9;
  val y = 6 : int`

- The identifier gets a type from the value it is bound to:

  - `val z = "first binding";
  val z = "first binding" : string`

  - `val z = 2;
  val z = 2 : int`
Constructed Types

- Tuples
- Records
- Lists
- Functions
Tuples

- Standard cartesian product — 2-tuple also called an (ordered) pair

- The 0-tuple is (), unit

- Syntax is a parenthesized list separated by commas:
  
  - (12.7,true,4);
  val it = (12.7,true,4) :
    real * bool * int

- Selector #k:
  
  - val third = #3 (12.7,true,4);
  val third = 4 : int
Records

- Cartesian product with **labels** on the fields

- Syntax:

  - val Alice = {student="Alice Jones", id = 307, QCA = 3.97};
  val Alice = {QCA=3.97,id=307, student="Alice Jones"}
  : {QCA:real, id:int, student:string}

- Selector **#label**: 

  - #student Alice;
  val it = "Alice Jones" : string

  - #id Alice;
  val it = 307 : int
Lists

- A sequence of items between [ and ] separated by commas:
  
  ```
  [2,3,5,7,11,13,17];
  val it = [2,3,5,7,11,13,17] : int list
  ```
  
  This is a list of integers.

- All list items must be of the same type:
  
  ```
  [(4,"joe"),(12,"margaret")];
  val it = [(4,"joe"),(12,"margaret")] : (int * string) list
  ```

  - `[4,"joe"];
  `stdIn:50.1-50.10 Error: operator and operand don’t agree [literal] operator domain: int * int list operand: int * string list in expression: 4 :: "joe" :: nil`
Lists Continued

- The empty list is [] or nil:

  - nil;
  val it = [] : 'a list

- Lists can be constructed using the cons operator :::

  - 2 :: [3,5,7,11,13,17];
  val it = [2,3,5,7,11,13,17] : int list

  - 2 :: 3 :: 5 :: 7 :: 11 :: 13 ::
    = 17 :: nil;
  val it = [2,3,5,7,11,13,17] : int list

- The append operator is @.

  - [2,3,5,7] @ [11,13,17];
  val it = [2,3,5,7,11,13,17] : int list
Lists Concluded

- Lists can be deconstructed using a pattern.

- val head :: tail = [2,3,5,7,11,13,17];
  stdIn:1.1-51.4 Warning: binding not exhaustive
  
  head :: tail = ...

val head = 2 : int
val tail = [3,5,7,11,13,17] : int list

- head; tail;
val it = 2 : int
val it = [3,5,7,11,13,17] : int list

but we need cases to exhaust all possibilities (later).
Functions

- A function is a first class object.

- Every function has a single argument:

  - op+;
  val it = fn : int * int -> int

  - op+ 3 5;
  stdIn:63.1-63.8 Error: operator and operand don’t agree [literal]
    operator domain: ’Z * ’Z
    operand: int
    in expression:
      + 3

  though that argument may be a tuple or other constructed type:

  - op+ (3,5);
  val it = 8 : int
Defining Functions

- The ML syntax for the \( \lambda \) expression

\[ \lambda x. e \]

is

\[ fn \ x \Rightarrow \ e \]

- Examples:

  - \[ fn \ t \Rightarrow 3 + t; \]
    \[ val \ it = fn : int \rightarrow int \]

  - \[ fn \ (p,q) \Rightarrow (p : int) \ast q; \]
    \[ val \ it = fn : int \ast int \rightarrow int \]

  - \( (fn \ t \Rightarrow 3 + t) \ 23; \)
    \[ val \ it = 26 : int \]
Naming Functions

- We can just use `val` to name a function:

  - `val plus3 = fn t => 3 + t;
  val plus3 = fn : int -> int

  - `val times = fn (p,q) => (p : int) * q;
  val times = fn : int * int -> int

  - `plus3 18;
  val it = 21 : int

- An alternate syntax uses `fun`:

  - `fun plus3 t = 3 + t;
  val plus3 = fn : int -> int

  - `fun times (p,q) = (p : int) * q;
  val times = fn : int * int -> int

  - `times (11,47);
  val it = 517 : int`
Conditional Expressions

The keywords if, then, and else are used to make a conditional expression.

- if true then "cat" else "dog";
  val it = "cat" : string

- (fn (b,p,q) => if b then p else q)
  (true,"cat","dog");
  val it = "cat" : string
Recursive Functions

• Numerical functions:

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

  - fact 12;
  val it = 479001600 : int

  - fun fib n = if n <= 1 then n
    else fib(n-1) + fib(n-2);
  val fib = fn : int -> int

  - fib 12;
  val it = 144 : int

  - fib 15;
  val it = 610 : int
Tail Recursion

- More space efficient to use tail recursion:

  - fun facti(n,p) = if n=0 then p else
    facti(n-1,n*p);
  val facti = fn : int * int -> int

  - facti(12,1);
  val it = 479001600 : int

  - fun itfib (n,prev,curr) : int =
    if n=1 then curr else
    itfib(n-1,curr,prev+curr);
  val itfib = fn : int * int * int -> int

  - itfib(12,0,1);
  val it = 144 : int

  - itfib(15,0,1);
  val it = 610 : int
Deconstructing Lists

- fun null [] = true
  = | null (_::_) = false;
val null = fn : 'a list -> bool

- null ["dog","cat"];
val it = false : bool

- fun length [] = 0
  = | length (_::tail) = 1 + length tail;
val length = fn : 'a list -> int

- length ["dog","cat"];
val it = 2 : int

- fun sum [] = 0
  = | sum (head::tail) = head + sum tail;
val sum = fn : int list -> int

- sum [1,2,3,4,5,6,7,8,9,10];
val it = 55 : int
Append Function

- fun append ([], l) = l
  =  | append (x::y, z) = x::append(y, z);
val append = fn : 'a list * 'a list -> 'a list

- infix @;
infix @
- fun [] @ l = l
  =  | (x::y) @ z = x :: (y @ z);
val @ = fn : 'a list * 'a list -> 'a list

- append ([10,9,8,7,6],[1,2,3,4,5]);
val it = [10,9,8,7,6,1,2,3,4,5] : int list

- [10,9,8,7,6] @ [1,2,3,4,5];
val it = [10,9,8,7,6,1,2,3,4,5] : int list
Reverse Function

- fun rev l =
  = let fun revto ([],y) = y
  = | revto (xhead::xtail,y) =
      revto (xtail,xhead::y)
  = in revto (l,[]) end;
val rev = fn : 'a list -> 'a list

- rev ["mouse","cat","dog"];
val it = ["dog","cat","mouse"] : string list

- rev [20,19,18,17,16,15,14,13,12,11];
val it = [11,12,13,14,15,16,17,18,19,20] : int list
Type Inferencing

- fn x => 3 + x;
val it = fn : int -> int

- fn (y,z) => y / z;
val it = fn : real * real -> real

- fun repeat(str,n) =
  = if n <= 0 then "" else
  = str ^ repeat(str,n-1);
val repeat = fn : string * int -> string

- repeat("010",5);
val it = "010010010010010" : string
Type Inferencing Continued

- infix plus; fun a plus b = (a:int) + b;
  infix plus
  val plus = fn : int * int -> int

- 3.0 plus 5.5;
  stdIn:23.1-23.13 Error: operator and
  operand don’t agree [tycon mismatch]
  operator domain: int * int
  operand: real * real
  in expression:
    (3.0 plus 5.5)