An Introduction to ML

Handout #28
CS251 Lecture 18
March 12, 2002
Integers I

- \(1 + 2;\)
  val it = 3 : int

- \(2 + 3 \times 4;\)
  val it = 14 : int

- \((2 + 3) \times 4;\)
  val it = 20 : int

- \(\text{val } a = 5 \times 6;\)
  val a = 30 : int

- \((a \div 7) + (a \mod 7);\)
  val it = 6 : int
Integers II

- 3 - 5;
val it = ~2 : int

- -17;

stdIn:21.1 Error: expression or pattern begins with
infix identifier "-"

stdIn:21.1–21.4 Error: operator and operand don't agree
[literal]
operator domain: 'Z * 'Z
operand:        int
in expression:
  - 17

- ~17;
val it = ~17 : int
Reals

- 1.2 + 3.0;

```val``
```
val it = 4.2 : real
```

- 1.2 + 3;

```
stdIn:26.1-26.8 Error: operator and operand don't agree
[literal]
operator domain: real * real
operand: real * int
in expression:
1.2 + 3
```

- 30 / 7;

```
stdIn:27.4 Error: overloaded variable not defined at
type
symbol: /
type: int
```

- 30.0 / 7.0;

```val``
```
val it = 4.28571428571 : real
```

Booleans

- 1 < 2;
  val it = true : bool

- 1 > 2;
  val it = false : bool

- not (1 > 2);
  val it = true : bool

- not 1 > 2;
  stdIn:30.1-30.10 Error: operator and operand don't agree [literal]
  operator domain: bool
  operand: int
  in expression:
    not 1
Short-Circuit Operators

- \( (1 < 2) \text{ andalso } (3 < 4) \);
val it = true : bool

- \( \text{false andalso } (3 < (4 \text{ div 0})) \);
val it = false : bool

- \( 1 < 2 \text{ andalso } 3 < 4 \);
val it = true : bool

- \( (1 > 2) \text{ orelse } (3 < 4) \);
val it = true : bool

- \( \text{true orelse } (3 < (4 \text{ div 0})) \);
val it = true : bool
Conditionals

- `if 1 < 2 then 3 + 4 else 5 * 6;
  val it = 7 : int`

- `if 1 > 2 then 3 + 4 else 5 * 6;
  val it = 30 : int`

- `if 1 < 2 then 3 + 4 else 5 < 6;
  stdIn:39.1-39.31 Error: types of rules don't agree [literal]
  earlier rule(s): bool -> int
  this rule: bool -> bool
  in rule: false => 5 < 6`

- `if 1 + 2 then 3 + 4 else 5 * 6;
  stdIn:1.1-31.18 Error: case object and rules don't agree [literal]
  rule domain: bool
  object: int
  in expression:
    (case (1 + 2)
     of true => 3 + 4
      | false => 5 * 6)`
Strings

- "foo";
val it = "foo" : string

- val s = "bar" ;
val s = "bar" : string

- "foo" ^ s ^ "baz";
val it = "foobaz" : string

- print ("int = " ^ (Int.toString (1 + 2)));
int = 3val it = () : unit

- print ("bool = " ^ (Bool.toString (1 < 2)) ^ "\n");
bool = true
val it = () : unit

- print ("string = " ^ s ^ "\n");
string = "bar"
val it = () : unit
Common printing errors

- print ("int = " ^ (Int.toString 1 + 2));
  stdIn:46.35 Error: overloaded variable not defined at type symbol: +
  type: string

- print "int = " ^ (Int.toString (1 + 2));
  stdIn:1.1-41.18 Error: operator and operand don't agree [tycon mismatch]
    operator domain: string * string
    operand:         unit * string
  in expression:
    print "int = " ^ Int.toString (1 + 2)

(* A correct version *)
- print ("int = " ^ (Int.toString (1 + 2)));

(* This also works *)
- print ("int = " ^ Int.toString (1 + 2));

Tuples

- `val t = (1 + 2, 3 < 4, "cs" ^ "251");`

val t = (3,true,"cs251") : int * bool * string

- `#1(t);`

val it = 3 : int

- `#2(t);`

val it = true : bool

- `#3(t);`

val it = "cs251" : string

- `val (a,b,c) = t;`

val a = 3 : int (* Evaluating a declaration *)
val b = true : bool (* can produce a set of *)
val c = "cs251" : string (* bindings *)

- `a * 2;`

val it = 6 : int
Let and Pattern Matching I

- let val (x, y) = (1+2, 3*4) in (x+y, x*y, x<y) end;
  val it = (15, 36, true) : int * int * bool

- let val (x, y) = (1+2, 3*4) = in (x+y, x*y, x<y) (* "=" is a continuation marker. *) = end; (* It is only used in interpreter, *not* in files. We omit in future. *)
  val it = (15, 36, true) : int * int * bool

- let val p as (x, y) = (1+2, 3*4) in (x+y, x<y, p) end;
  val it = (15, true, (3, 12)) : int * bool * (int * int)

- let val (x, y) = (1+2, 3*4, 5-6) in x + y end;
  stdIn:61.5-61.32 Error: pattern and expression in val dec don't agree [tycon mismatch]
  pattern: 'Z * 'Y
  expression: int * int * int
  in declaration:
    (x, y) =
      (case (1 + 2, 3 * 4, 5 - 6)
        of (x, y) => (x, y))
Let and Pattern Matching II

- let val (x,y) = (1+2, 3*4)
  val w = x+y
  val z = x*y
  in (w+z, w*x, y+z)
end;

val it = (51,45,48) : int * int * int
Local

The `local` construct allows defining a collection of bindings in the scope of local declarations.

```ml
- local val d = 1+2
  in  val e = d + 1
       val f = d * 2
  end;
val e = 4 : int
val f = 6 : int
```
Lists I

- `val L1 = [1+2, 3-4, 5*6];`
`val L1 = [3,~1,30] : int list`

- `val L2 = [1 < 2, 3 > 4];`
`val L2 = [true,false] : bool list`

- `val L3 = [1 + 2, 3 < 4];`

`stdIn:69.10-69.24 Error: operator and operand don't agree`

[literal]

operator domain: int * int list
operand: int * bool list

in expression:

`1 + 2 :: (3 < 4) :: nil`

- `hd(L1);`
`val it = 3 : int`

- `tl(L2);`
`val it = [false] : bool list`
Lists II

- `tl(tl(L2));`  (* Java style invocation syntax *)
  `val it = [] : bool list`

- `(tl (tl L2));`  (* Scheme style invocation syntax *)
  `val it = [] : bool list`

- `null(tl(L2));`
  `val it = false : bool`

- `null(tl(tl(L2)));`
  `val it = true : bool`

- `val L4 = (7+8) :: L1;`
  `val L4 = [15,3,~1,30] : int list`

- `L1;`
  `val it = [3,~1,30] : int list`
Pattern Matching with Lists

- `let val [a,b,c] = L1 in (a + b)*c end;`
  
  `stdIn:80.5-80.21 Warning: binding not exhaustive`
  
  `a :: b :: c :: nil = ...`

  `val it = 60 : int`

- `let val (x :: xs) = L1 in ((x * 2) :: xs) @ [x+1] end;`
  
  `stdIn:81.5-81.23 Warning: binding not exhaustive`
  
  `x :: xs = ...`

  `val it = [6,~1,30,4] : int list`
Functions I

- val inc = fn \( x \mapsto x + 1 \);

val inc = fn : int \to int

- inc 3; (* or "inc(3)" or "(inc 3)" *)

val it = 4 : int

- fun pos y = y > 0;

val pos = fn : int \to bool

- pos 17;

val it = true : bool

- fun avg (a,b) = (a + b) div 2;

val avg = fn : int * int \to int

- avg (3,8);

val it = 5 : int
Functions II

- fun avgCurried a b = (a + b) div 2;
val avgCurried = fn : int -> int -> int

- avgCurried 3 8;
val it = 5 : int

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

- fact 5;
val it = 120 : int
Tracing Factorial

```ml
- fun trace_fact n =
  let val _ = print ("Entering fact(" ^ (Int.toString n) ^ ")\n");
    val result = if n < 0 then 1 else n * (trace_fact(n-1))
    val _ = print ("Exiting fact(" ^ (Int.toString n) ^ ") = " ^ (Int.toString result) ^ "\n");
  in result
  end;
val trace_fact = fn : int -> int

- trace_fact(3);
  Entering fact(3)
  Entering fact(2)
  Entering fact(1)
  Entering fact(0)
  Exiting fact(0) = 1
  Exiting fact(1) = 1
  Exiting fact(2) = 2
  Exiting fact(3) = 6
val it = 6 : int
```
Higher-Order Functions I

- \texttt{fun} app5 \texttt{f} = \texttt{f} 5;
\texttt{val} app5 = \texttt{fn} : (int \to 'a) \to 'a (* 'a means “any type” *)

- app5 inc;
\texttt{val} it = 6 : int

- app5 pos;
\texttt{val} it = \texttt{true} : bool

- \texttt{fun} create_sub \texttt{n} = \texttt{fn} \texttt{x} => \texttt{x} - \texttt{n};
\texttt{val} create_sub = \texttt{fn} : int \to int \to int

- (create_sub 2);
\texttt{val} it = \texttt{fn} : int \to int

- (app5 create_sub);
\texttt{val} it = \texttt{fn} : int \to int

- ((app5 create_sub) 3);
\texttt{val} it = \texttt{~2} : int
Higher-Order Functions II

- (create_sub app5);

stdIn:135.1-135.18 Error: operator and operand don't agree
  [tycon mismatch]
  operator domain: int
  operand:         (int -> 'Z) -> 'Z
in expression:
  create_sub app5

- fun create_sub2 n x = x - n; (* Curried function *)
val create_sub2 = fn : int -> int -> int

- fun avg2 a b = (a + b) div 2;
val avg2 = fn : int -> int -> int

- app5 (avg2 15);
val it = 10 : int

- app5 (fn x => avg(15, x));
val it = 10 : int
Composition I

- `fun` compose f g x = f(g(x));

```
val compose = fn : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b
```

- `compose` pos inc;

```
val it = fn : int -> bool
```

- `compose` pos inc 5;

```
val it = true : bool
```
Composition II

- fun id x = x;

val identity = fn : 'a -> 'a

- fun repeated f n =
  if n = 0 then id else compose f (repeated f (n - 1));

val repeated = fn : ('a -> 'a) -> int -> 'a -> 'a

- repeated inc 5 3;
val it = 8 : int

- repeated inc 5;
val it = fn : int -> int

- repeated inc;
val it = fn : int -> int -> int
List Functions

- fun sumlist lst = 
  case lst of 
    [] => 0 
    | (x :: xs) => x + (sumlist xs); 
val sumlist = fn : int list -> int 

- sumlist [5, 1, 3, 2]; 
val it = 11 : int 

- fun sumlist2 [] = 0 
  | sumlist2 (x :: xs) = x + (sumlist2(xs)); 
val sumlist2 = fn : int list -> int
Higher-Order List Functions

- `fun` `map` `f` `[]` = `[]`
  `| map` `f` `(x :: xs)` = `(f x)` :: `(map` `f` `xs)`;
`val` `map` = `fn` : `'a` -> `'b` -> `'a` list -> `'b` list

- `map` `inc` `L1`;
`val` `it` = `[4,0,31]` : `int` list

- `map` `pos` `L1`;
`val` `it` = `[true,false,true]` : `bool` list

- `map` `(fn` `x` => `(x, x*2))` `L1`;
`val` `it` = `[(3,6),(~1,~2),(30,60)]` : `(int` * `int)` list
Scope I

- `val a = 1+2;`  
  `val a = 3 : int`

- `fun add_a x = x + a;`  
  `val add_a = fn : int -> int`

- `fun try a = add_a a;`  
  `val try = fn : int -> int`

- `try 100;`  
  `val it = 103 : int (* Like Scheme, ML has static scope *)`

- `val a = 17; (* This is a new a; previous a unchanged *)`  
  `val a = 17 : int`

- `try 100;`  
  `val it = 103 : int (* Uses previous a *)`
Scope II

Function declarations are sequential by default:
- let fun isEven n = if n = 0 then true else isOdd(n-1)
  fun isOdd n = if n = 0 then false else isEven(n-1)
  in map isOdd [0,1,2]
end;

stdIn:179.44–179.49 Error: unbound variable or constructor: isOdd

The and keyword must be used for mutually recursive function declarations (or declaration before use).
- let fun isEven n = if n = 0 then true else isOdd(n-1)
  and isOdd n = if n = 0 then false else isEven(n-1)
  in map isOdd [0,1,2]
end;
val it = [false,true,false] : bool list

In ML, only functions can be defined recursively (compare to Scheme’s letrec.)
User-Defined Datatypes I

datatype Figure =
  Circle of real (* radius *)
  | Square of real (* side length *)
  | Rect of real * real (* width x height *)

val pi = 3.14159

fun perim (Circle radius) = 2.0*pi*radius
  | perim (Square side) = 4.0*side
  | perim (Rect (width,height)) = 2.0*(width+height)

fun double (Circle r) = (Circle (2.0*r))
  | double (Square s) = (Square (2.0*s))
  | double (Rect (w,h)) = (Rect (2.0*w,2.0*h))
User-Defined Datatypes II

Here are the types of the datatypes and functions on the previous slide:

```
datatype Figure = Circle of real | Rect of real * real | Square of real
val pi = 3.14159 : real
val perim = fn : Figure -> real
val double = fn : Figure -> Figure
```