Intro to SML

Olin Shivers

SML

- Higher-order, functional language.
  Functions are "first-class" values (like Scheme).
- Strongly typed.
  Compiler catches all type errors.
- Rich, expressive type system—a powerful tool.
  Includes polymorphic types.
- Infix syntax: a + (b*c)
  (Not like Scheme)
- Has side-effects (not like Haskell).
- Other important features: GC, exceptions, modules, pattern matching.

Practically a domain-specific language for writing compilers.

SML expressions

SML is *expression-oriented*. Every expression has

- a value,
- a type,
- an effect.

(Well, most times. . .)

Integers

```
tinhau$ sml
Standard ML of New Jersey, Version 110.0.7,
September 28, 2000 [CM; autoload enabled]
- 1 + 2;
val it = 3 : int

- 2+3*4;
val it = 14 : int

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

- val a = 5 * 6;
val a = 30 : int

- (a div 7) + (a mod 7);
GC #0.0.0.0.1.11:   (0 ms)
val it = 6 : int
```

Integers II

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

- -17;
stdIn:23.1 Error: expression or pattern begins with infix ide
stdIn:23.1-23.4 Error: operator and operand don't agree [lite
  operator domain: 'Z * 'Z
  operand:         int
  in expression:
    - 17

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

Reals

```
- 1.2 + 3;
stdIn:25.1-25.8 Error: operator and operand don't agree [lite
  operator domain: real * real
  operand:         real * int
  in expression:
    1.2 + 3

- 30 / 7;
stdIn:1.4 Error: overloaded variable not defined at type
  symbol: /
  type: int

- 30 div 7;
val it = 4 : int

- 30.0 / 7.0;
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:31.1-31.10 Error: operator and operand don't agree [lit
  operator domain: bool
  operand:         int
  in expression:
    not 1
```

Conditionals

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

- if 1 < 2 then 3 + 4 else 5 < 6;
stdIn:33.1-33.31 Error: types of rules don't agree [literal]
  earlier rule(s): bool -> int
  this rule: bool -> bool
  in rule:
    false => 5 < 6
```

Strings

```
- "Foo";
val it = "Foo" : string

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

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

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

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

Common printing errors

```
- print( "int = " ^ (Int.tostring 1 + 2));
stdIn:39.20-39.32 Error: unbound variable or constructor: tos

- print "int = " ^ (Int.tostring(1 + 2));
stdIn:34.12-35.9 Error: unbound variable or constructor: tost
stdIn:1.1-35.17 Error: operator and operand don't agree [tycc
  operator domain: string * string
  operand:         unit * _
  in expression:
    print "int = " ^ <errorvar> (1 + 2)
```

Tuples & Patterns

```
- val t = (1+2,  3<4,  "foo");
val t = (3,true,"foo") : int * bool * string

- #1(t);
val it = 3 : int

- #2(t);
val it = true : bool

- #3 t;  (* Look ma, no parens. *)
val it = "foo" : string

- val (a,_,c) = t;   (* "_" is don't-care pattern. *)
val a = 3 : int
val c = "foo" : string

- a * 2;
val it = 6 : int
```

Let and pattern matching

```
- 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, 3<4) in x+y end;
stdIn:50.5-50.33 Error: pattern and expression in val dec don
  pattern:    'Z * 'Y
  expression:    int * int * bool
  in declaration:
    (x,y) =
      (case (1 + 2,3 * 4,3 < 4)
        of (x,y) => (x,y))
```

Function definition

```
- fun diffsq(x,y) = (x*x) - (y*y);
val it = fn : int * int -> int

- val sqdiff = (fn (x,y) => (x-y)*(x-y));
val sqdiff = fn : int * int -> int

- sqdiff(8,5);
val it = 9 : int

- val pr = (1,5);
val pr = (1,5) : int * int

- sqdiff pr;
val it = 16 : int

- (if 3 < 2 then sqdiff else diffsq) (1,5);
val it = ~24 : int
```

Functions as values

```
- (if 3 < 2 then op + else (fn (x,y) => x-y*y)) (5,2);
val it = 1 : int
```

Factorial three ways

```
fun fact1 n =
    if n = 0 then 1
    else n * fact1(n-1)

fun fact2 0 = 1
  | fact2 n = n * fact2(n-1);

fun fact3(a, 0) = a
  | fact3(a, n) = fact3(a*n, n-1)

... fact3(1, 5) ...
```

Higher-order procedures

```
- fun adder x = (fn y => x+y);
val adder = fn : int -> int -> int

- (adder 5) 3;
val it = 8 : int

- val f = adder 5;
val f = fn : int -> int

- f 2;
val it = 7 : int
```

Polymorphism

```
fun map(f, [])      = []
  | map(f, x1::xs) = (f x1) :: (map(f, xs))
```

What is type of `map`?

```
map(isEven, [3,8,1,5])
...
map(size, ["The", "rain", "in", "Spain"])
```

More higher-order procedures

```
fun compose(f,g) = (fn x => f(g x))
(* What is type of compose? *)

val add5 = compose(adder 3, adder 2)

   ... (add5 4) ...

fun deriv(f,eps) =
  fn x => (f(x+eps)-f(x-eps)) / (2.0*eps)

val mycos = deriv(sin, 0.001)
```

User-defined datatypes

```
datatype IntTree = ILeaf of int
                 | INode of IntTree * IntTree

val it1 = INode( ILeaf 3, INode(ILeaf 0, ILeaf 2) )

fun addints( ILeaf i )       = i
  | addints( INode(t1, t2) ) = addints(t1) + addints(t2)
```

```
- addints it1;
val it = 5 : int
```

```
fun folditree(f, ILeaf i) = i
  | folditree(f, INode(t1, t2)) =
        f(folditree(f, t1), folditree(f, t2))
```

```
- folditree( op +, it1);
val it = 5 : int

- folditree( Int.max, it1);
val it = 3 : int

- folditree( Int.min, it1);
val it = 0 : int
```

Polymorphism & datatypes

```
datatype 'a btree = Leaf of 'a
                  | Node of 'a btree * 'a btree

fun foldtree(f, Leaf x) = x
  | foldtree(f, Node(t1, t2)) =
        f(foldtree(f, t1), foldtree(f, t2));
```

```
- foldtree( op ^, Node( Leaf "foo",
                        Node( Leaf "bar",
                              Leaf "baz") ));
val it = "foobarbaz" : string

- foldtree( Int.max, Node( Leaf 7,
                           Node( Leaf 22,
                                 Leaf ~3)));
val it = 22 : int

- foldtree( Int.max, Node( Leaf 7, Leaf "foo"));
stdIn:34.2-116.18 Error: operator and operand don't agree [literal]
  operator domain: int Tree * int Tree
  operand:         int Tree * string Tree
  in expression:
    Node (Leaf 7,Leaf "foo")
```

Pitfalls

- Ugly int/real overloading.
- left-associative function application
- redefinition
- The "value restriction"
- grammar problems

Features you need to know

- refs
- records, datatypes & patterns
- modules
- exceptions
- CM
- emacs sml & inferior-sml mode