Getting and Using SML

Getting SML

- Head over to smlnj.org
  - Click on Downloads
- Or use your favorite package manger (Homebrew, yum, etc.)

Using SML Interactively

- `sml` at the command prompt

. . . we'll talk later about non-interactive usage.

SML Resources

On the web

- Head over to smlnj.org
  - Check out the "Documentation and Literature"
- Google / StackOverflow

Offline

- "Elements of ML Programming" by Jeffrey D. Ullman
  - Numerous used copies online for less than $5

Basic SML Expressions

- constants (i.e., literals)
- variable references
- function application
- conditional expressions

Constants

- **Integers:** 0, 22, 353,...
- **Reals:** 12.0, 3E-2, 3.14e12
- **Booleans:** true, false
- **Strings:** "KSU", "foo\n"
- **Characters:** #"x", #"A", #"\n"

Example Session

```
- 2;
val it = 2 : int
- it + 1;
val it = 3 : int
- it;
val it = 3 : int
- ~234 + 2;
val it = ~232 : int
- 12.0;
val it = 12.0 : real
- 12. + 3.1;
stdIn:16.1 Error: syntax error found at DOT
- "KSU";
val it = "KSU" : string
- "foo\n";
val it = "foo\n" : string
- #"x";
val it = #"x" : char
- #"gh";
... Error: character constant not length 1
```

Arithmetic Operators

Precedence: lowest to highest

- $+$, $-$
- $*$, $/$, div, mod
- ~

Also:

- ML is case sensitive (cf. `mod`)
- associativity and precedence as in other languages
- operators associate to the left
- parentheses are
  - needed only to enforce evaluation order, as in `x * (y + z)`
  - but may be freely added to improve clarity, as in `x + (y * z)`

String Operators

Concatenation:

```
- "abra" ^ "cadabra";
val it = "abracadabra" : string

- "abra" ^ "" ^ "cadabra" ^ "";
val it = "abracadabra" : string

- "abra" ^ ("" ^ "cadabra") ^ "";
val it = "abracadabra" : string
```

- "" (empty string) is identity element
- ^ is associative

Comparison Operators

$$=,\ <,\ >,\ <=,\ >=,\ <>$$

Note:

- cannot use $=$ or $<>$ on reals
  - to avoid problems with rounding
  - use e.g., $<=$ and $>=$ for $=$
- $<$ means "lexicographically precedes" for characters and strings

```
- "a" < "b";
val it = true : bool
- "c" < "b";
val it = false : bool
- "abc" < "acb";
val it = true : bool
- "stuv" < "stu";
val it = false : bool
```

"Problems with Rounding"

Example

- $1.1 + 2.2 = 3.3$ right?

Nope!

- $$\frac{2476979795053773}{2251799813685248} + \frac{2476979795053773}{1125899906842624} \neq \frac{3715469692580659}{1125899906842624}$$

– Possibly Wrong Blog

Boolean Operators

```
not , andalso , orelse
```

- behave like C's `!`, `&&`, `||` — not like Pascal
- not commutative, as "short-circuit" operation

```
- (1 < 4) orelse ((5 div 0) < 2);
val it = true : bool
- ((5 div 0) < 2) orelse (1 < 4);
** error **
```

If-then-else Expressions

Examples:

```
- if 4 < 3 then ''a'' else ''bcd'';
val it = ''bcd'' : string

- val t = true;
val t = true : bool
- val f = false;
val f = false : bool

- if t = f then (5 div 0) else 6;
val it = 6 : int

- if t = true then 7 else ''foo'';
... Error: types of rules don't agree...
  earlier rule(s): bool -> int
  this rule: bool -> string
  in rule:
    false => ''foo''
```

Typing Issues

ML has strong typing:
(strong/weak = how much)

- each value has exactly one type
- for example, 12 is `int` but not `real`
- explicit coercions therefore necessary

ML has static typing:
(static/dynamic = when)

- type-checking occurs *before* programs are run
    - thus `if x = y then 7 else "foo"` is an error
    - but it wouldn't be in a dynamically typed language

These concepts are too often mixed up, even in the Ullman textbook (pages 3 and 143)

Numeric Coercions

From integers to reals:

```
 - real(11);
val it = 11.0 : real
 - 5.0 + 11;
 ... Error: operator and operand mismatch
   operator domain: real * real
   operand:         real * int
   in expression:
     5.0 + 11
 - 5.0 + real(11);
val it = 16.0 : real
```

From reals to integers:

```
 - floor(5.4);
val it = 5 : int
 - ceil(5.4);
val it = 6 : int
 - round(5.5);
val it = 6 : int
 - trunc(~5.4);
val it = ~5 : int
```

Character Coercions

Between characters and integers:

```
  - ord(#"0");
  val it = 48 : int

  - chr(48);
  val it = #"0" : char
```

Between strings and characters:

```
  - str(#"a");
  val it = "a" : string
```

What about from int to string?
What about from string to character?

Identifier Quiz

Which of the following do you think are valid SML identifiers?:

- myVar (Yes)
- myVar_42 (Yes)
- myVar' (Yes)
- ++ (Yes)
- t@coc@t (No)
- %-/-< (Definitely)

Identifier Rules

SML has two classes of identifiers:

- alphanumeric (e.g., `abc`, `abc'`, `A_1`)
- symbolic (e.g., `+`, `$$$`, `%-%`)

Alphanumeric Identifiers: strings formed by

- An upper or lower case letter or the character `'` (called apostrophe or "prime"), followed by
- Zero or more additional characters from the set given in (1) plus the digits and the character _ (underscore).

Symbolic Identifiers: strings composed of

```
+ - / * < > = ! @ # $ % ^ & ` ~ \ | ? :
```

Variables in Pascal

Consider from Pascal: `A := B + 2;`

- B is a *variable reference* (contrast with A)
- a memory location is associated with A
- a stored value (e.g., 5) is associated with B

Pascal, C, Java, Fortran, etc:

```
                     memory  cell  <loc>
                 +-------------------+
  <var>   ==     |      <value>      |
                 +-------------------+
```

- variables bind to locations
- there is a level of indirection
- two mappings
  - environment: maps variables to locations
  - store: maps locations to values

Variables in SML

SML: variables bound to values

$$<var> == <value>$$

- variables bind directly to values
- there is no indirection
- a binding cannot be modified
- there is no assignment
- one mapping
  - environment: maps variables to values

Top-level Environment

```
- val a = 2;
val a = 2 : int
- val b = 3;
val b = 3 : int
- val c = a + b;
val c = 5 : int
- val a = c + 2;
val a = 7 : int
- val c = c + 2;
val c = 7 : int
```

var	value
a	2
b	3
c	5
a	7
c	7

Tuples

Tuple: fixed-size ordered collection of two or more values.

```
- val t = (1, "a", true);
val t = (1,"a",true) : int * string * bool
- #3(t);
val it = true : bool
 - val s = (4, t);
val s = (4,(1,"a",true)) :
            int * (int * string * bool)
- #2(#2(s));
val it = "a" : string
- (4);
val it = 4 : int
- ();
val it = () : unit
- #2 t;
val it = "a" : string
- #4(t);
stdIn:16.1-16.6 Error: ...
```

Lists

ML lists are lists of values of the same type.

Example session:

```
- [1,2,3];
val it = [1,2,3] : int list
- [(1,2),(2,3),(3,4)];
val it = [(1,2),(2,3),(3,4)] :
        (int * int) list
- ["a"];
val it = ["a"] : string list
- ["a",2];
... Error: operator and operand don't agree...
- [[1],[2],[3]];
val it = [[1],[2],[3]] : int list list
- [];
val it = [] : 'a list
```

Tuples vs. Lists: What's the difference?

- Lists: (Always) same types
- Tuples: (Possibly) different types

But ok, can't tuples do it all then?

- Tuples (generally) are sequences of different kinds of stuff, and you deal with the tuple as a coherent unit.
  - A location type might be (latitude, longitude, altitude). We don't really ever do something to each element (like double it) because the tuple only makes sense as a whole unit.
- Lists (generally) are sequences of the same kind of stuff, and you deal with the items individually.
  - A shopping list might be like ["Funfetti Cake Mix", "Eggs", "Oil", "Funfetti frosting"]. When we shop, we want to iterate over the list, and do something with (ie, buy) each item.

– Understanding tuples vs. lists in Python, Paul Bissex

Polymorphic List Operations

empty list	`[]`	:	`'a list`
head	`hd`	:	`'a list → 'a`
tail	`tl`	:	`'a list → 'a list`
append	`@`	:	`'a list * 'a list → 'a list`
cons	`::`	:	`'a * 'a list → 'a list`

Example session:

```
- val ls = [1,2,3];
val ls = [1,2,3] : int list
- hd(ls);
val it = 1 : int
- hd(["a","b","c"]);
val it = "a" : string
- tl(tl(ls));
val it = [3] : int list
- tl(tl(ls)) @ ls;
val it = [3,1,2,3] : int list
- 3 @ ls;
... Error: operator and operand don't agree
- 3 :: ls;
val it = [3,1,2,3] : int list
```

Strings $\leftrightarrow$ List Coercion

Example session:

```
 - explode("abcd");
 val it = [#"a",#"b",#"c",#"d"] : char list
 - implode([#"f",#"o",#"o"]);
 val it = "foo" : string
 - implode(explode("abcd"));
 val it = "abcd" : string
 - explode(implode([#"f",#"o",#"o"]));
 val it = [#"f",#"o",#"o"] : char list
```

Strings ↔ List Coercion

```
- "abc" ^ implode([#"f",#"o",#"o"]) ^ "bar";
val it = "abcfoobar" : string
- ([4,5],[2],[ord(#"c")]);
val it = ([4,5],[2],[99]) :
         int list * int list * int list
- "abc" > "foo";
val it = false : bool
- 7 :: 5;
stdIn:37.1-37.7 Error:
   operator and operand don't agree [literal]
- ["a","b",#"c","d"];
stdIn:1.1-30.2 Error: operator and operand
   don't agree [tycon mismatch]
- 20 + (if #"c" < #"C" then 5 else 10);
val it = 30 : int
- ((),(),[()],([]));
... : unit * unit * unit list * 'a list
```

Summary

ML is an expression-based *(*functional*) language with* strong static *typing.*

Next lecture: user-defined functions