Introduction to SML

SML is an expression-based (functional) language.

1. why SML in 416?
2. statements vs. expressions
3. basic SML expressions
   - literals, variable references, function calls, conditionals, ...
4. typing issues
5. tuples and lists
6. definitions and programs
Why SML?

- **Well-understood foundations:** This is a course about the foundations of programming languages, and the theory/fundations of SML have been studied more in recent years than almost any other language.

- **Well-designed:** Robin Milner, the principal designer of SML received the Turing Award, in part, because of his work on SML.

- **Advanced features:** Many of the features of SML, such as parametric polymorphism, pattern matching, and advanced modules are very elegant and do not appear in other languages like Java, C++, etc.

- **Supported by industry:** Bell Labs is the leading developer of SML implementations. Used for telephony applications (note the use of Erlang by Ericsson).

- **Very high-level:** Using SML lets us describe language processors very succinctly (much more concisely than any imperative language).

- **Clean:** SML is useful for various critical applications where programs need to be proven correct (e.g., protocol implementations – Ensemble).
Why SML? (continued)

- **It's different than Java:** At some point in your career, you will have to learn a new language. This course prepares you for that by forcing you to learn a new language (SML) quickly. In addition, compared to Java, C, etc., SML uses a totally different style to describe computation. This forces you to think more deeply (mental pushups!).

- **There's more!** There are also several different concurrent versions of SML, object-oriented extensions, libraries for programming X-windows applications, etc.
Statements vs. Expressions

Statement:

• ...

Examples:

m := 5;
n := 2;
result := 1;
while n > 0 do
  result := result * m;
  n := n - 1
end while;
write result;

Statement-oriented/imperative languages:

• Pascal, C, C++, Ada, FORTRAN, COBOL, etc.
Statements vs. Expressions

Expression:
  • ...

Examples:
A := 2 + 3;

(power 5 2)
a = (b = c++) + 1;

Pure expressions: ...

Expression-oriented/functional languages:
  • Scheme, ML, Lisp, Haskel, Miranda, FP, etc.
Basic SML Expressions

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

• **Integers:** 0, 22, \~353, 0x12, ...

• **Reals:** 12.0, 3E\~2, 3.14e12

• **Booleans:** true, false

• **Strings:** "KSU", "foo\n",

• **Characters:** "x", "A", \"n"
Example Session

- 0;
- it + 3;
- it;
- -234 + 2;
- 12.0;
- -3 + 2;
- 4
= + 3;
- 12. + 3.1;
- "KSU";
- "foo\n";
- #"x";
- #"gh";
Arithmetic Operators

**Precedence:** lowest to highest

- +, –
- *, /, div, mod
- ~

**Also:**

- ML is case sensitive (*cf.* mod)
- associativity and precedence as in Pascal or C
- operators associate to the left
- may add parentheses
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";
  - "c" < "b";
  - "abc" < "acb";
  - "stuv" < "stu";
Boolean Operators

not, andalso, orelse

- behave like C !, &&, || — not like Pascal
- “short-circuit” operation

  - (1 < 4) orelse ((5 div 0) < 2);

  - ((5 div 0) < 2) orelse (1 < 4);

Are the boolean operations commutative?
If-then-else Expressions

Examples:

- if 4 < 3 then "a" else "bcd";

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

- if (if f then t else t) then
  (if f then t else f) else
  (if t then f else t);

- if t = f then (5 div 0) else 6;

- if t = true then 7 else "foo";
Typing Issues

ML is strongly typed:
(strong/weak = how much)

• ...

• ...

ML is statically typed:
(static/dynamic = when)

• ...

Slides #2: Intro to SML
Coercions

From integers to reals:

- real(11);
- 5.0 + 11;
- 5.0 + real(11);

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
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 string to character?
Identifiers

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

- + - / * < > = ! @ # $ % ~ & ‘ ~ \ | ? :
Consider from Pascal: \( A := B + 2; \)

- ... 
- ... 
- ...

Pascal, C, Java, Fortran, etc.:

\[
\text{memory cell } \langle \text{loc} \rangle \\
+---------------+ \\
\langle \text{var} \rangle \quad \text{==} \quad | \quad \langle \text{value} \rangle \quad | \\
+---------------+ \\
\]

- ...
- ...
-...

- environment:
  - store:
Variables

**SML**: variables bound to values

\[ <\text{var}> \; == \; <\text{value}> \]

- variables bind directly to values
- there is no indirection
- a binding cannot be modified (!!)
- no assignment (!!)
- one mapping
  - environment: maps variables to values
Top-level Environment

Example session:
- val a = 2;
- val b = 3;
- val c = a + b;
- val a = c + 2;
- val c = c + 2;

Diagram of environment:

<table>
<thead>
<tr>
<th>var</th>
<th>value</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

Slides #2: Intro to SML
Top-level Environment

**Note:** declarations at the top-level may seem like assignments.... but they’re not!

**Example session:**

- val a = 2;
  
  val a = 2 : int
  
- fun myfun x = x + a;
  
  val myfun = fn : int -> int
  
- val a = 4;
  
  val a = 4 : int
  
- myfun(5);
  
  val it = ... (* what’s the answer? *)

**Assessment:**

- Technically speaking, all of ML (including the top-level environment) is *statically scoped*
- New definitions of the same variable don’t overwrite old definitions; they *shadow* the old definitions
- For efficiency, old definitions may be garbage collected if they are not referred to.
Tuples

A tuple is a fixed-sized ordered collection of two or more values.

Example session:

- val t = (1, "a", true);
val t = (1,"a",true) : int * string * bool
- #3(t);

- #1(t);

- val s = (4, t);

- #2(#2(s));

- (4);

- ();

- #(1+1)(t);

- #2 t;

- #4(t);
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)];``

- `["a"];``

- `["a",2];``

- `[1], [2], [3]];``

- `[];`
Polymorphic List Operations

- empty list: [] : 'a list
- head, tail:
  - hd : 'a list -> 'a
  - tl : 'a list -> 'a list
- append: @ : 'a list * 'a list -> 'a list
- cons: :: : 'a * 'a list -> 'a list

Example session:

- val l = [1,2,3];
  val l = [1,2,3] : int list
  - hd(l);

  - hd(['a','b','c']);

  - tl(tl(l));

  - tl(tl(l)) @ l;

  - 3 @ l;

  - 3 :: l;
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
Examples

- "abc" ^ implode(["f","o","o"] ^ "bar";
- ([4,5],[2],[ord("c")]);
- "abc" > "foo";
- 7 :: 5;
- ["a","b","c","d"];
- 20 + (if "c" < "C" then 5 else 10);
- ((),(),[()],());
Simple ML programs are generally a sequence of function definitions

fun push (val, stack)
  ...;
  ...;

fun pop (stack)
  ...;
  ...;

fun empty (stack)
  ...;
  ...;

fun make-stack (val)
  ...;
  ...;
Summary

ML is an expression-based (functional) language.

1. statements vs. expressions

2. basic ML expressions
   - literals, variable references, function calls, conditionals, tuples, lists

3. strong static typing

Next lecture: user-defined functions
## Terms and Concepts

<table>
<thead>
<tr>
<th>statement</th>
<th>expression</th>
<th>statement-based language</th>
</tr>
</thead>
<tbody>
<tr>
<td>literal</td>
<td>special form</td>
<td>expression-based language</td>
</tr>
<tr>
<td>conditional</td>
<td>coercion</td>
<td>strong typing</td>
</tr>
<tr>
<td>static typing</td>
<td>language paradigms</td>
<td>pure</td>
</tr>
<tr>
<td>side-effect</td>
<td>identifier</td>
<td>keyword</td>
</tr>
<tr>
<td>bound</td>
<td>binding</td>
<td>environment</td>
</tr>
</tbody>
</table>

- contrast imperative vs. functional languages
- contrast statement vs. expression
- pick out pure expressions