Keywords:
mosml; sml; value declarations; static binding; basic types (integers, reals, truth values, characters, strings); function declarations; overloading; tuples; recursion; expression evaluation; call-by-value.

References:
♦ [MLWP, Chapter 2]
♦ SML/NJ (http://www.smlnj.org/)
♦ Moscow ML (http://www.dina.dk/~sestof/mosml.html)

Most functional languages are interactive:
- val pi = 3.14159;
> val pi = 3.14159 : real
- val area = pi * 2.0 * 2.0;
> val area = 12.56636 : real
-

A declaration gives something a name, or binds something to a name. In ML many things can be named: values, types, ...

Running ML

$ mosml
Moscow ML version 2.00 (June 2000)
Enter ‘quit();’ to quit.
-

% sml
Standard ML of New Jersey v110.57
-

Static binding

If a name is redeclared then the new meaning is adopted afterwards, but does not affect existing uses of the name.

val pi = 3.14159;
val radius = 2.0;
val area = pi * radius * radius;
val pi = 0.0;
area;
-
"p01"
- use"p01";
[opening file "p01"]
> val pi = 3.14159 : real
> val radius = 2.0 : real
> val area = 12.56636 : real
> val pi = 0.0 : real
> val it = 12.56636 : real
[closing file "p01"]

The name it always has the value of the last expression typed at top level.

### Arithmetic

#### Integers and Reals

- The type of integers: int.
- Constants: ..., -2, -1, 0, 1, 2, ...
- Built-in operators and functions: Try the following in ML
  - load"Int"; (* needed in mosml, but not in sml *)
  - open Int;

- The type of reals: real.
- Constants:
  - ..., -1E6, -1.41, -1E-10, 1E-10, 1.41, 1E6, ...
- Built-in operators and functions: Try the following in ML
  - load"Real"; open Real;
  - load"Math"; open Math;

### Characters and strings

- The type of characters: char.
- Constants: "A", "a", ..., "1", ..., " ", ..., "\n"
- Built-in operators and functions:
  - Try the following in ML - load"Char"; open Char;

- The type of strings: string.
- Constants:
  - "", " ", "A", "z", "0 a 1 A ... 0 z 1 Z"
  - "Bye, bye ... \n"
- Built-in operators and functions:
  - Try the following in ML - load"String"; open String;

### Truth values

The type of booleans: bool.
- Constants: false true
- Built-in operators and functions:
  - Try the following in ML - load"Bool"; open Bool;
Declaring functions

val pi = 3.14159;
fun square (x:real) = x * x; (* overloading *)
fun area (radius) = pi * square(radius);
area (0.5);
val pi = 0.0;
area 0.5;
~"p02"

Overloading

Certain built-in operators are overloaded, having more than one meaning. For instance, + and * are defined both for integers and reals.

The type of an overloaded function must be determined from the context; occasionally types must be stated explicitly.

- fun int_square (x:int) = x * x;
  > val int_square = fn : int -> int
- fun default_square x = x * x;
  > val default_square = fn : int -> int

NB: SML’97 defines a notion of default type. The SML compiler will resolve the overloading in a predefined way; relying on this is bad programming style.

Declaring functions

Conditional expressions

To define a function by cases — where the result depends on the outcome of a test — we employ a conditional expression.

fun sign n
  = if n>0 then 1 else if n=0 then 0 else ~1;
fun absval x
  = if x >= 0.0 then x else ~x;
~"p03"

The boolean infix operators andalso and orelse are not functions, but stand for conditional expressions:

- E1 andalso E2 ≡ if E1 then E2 else false
- E1 orelse E2 ≡ if E1 then true else E2
Tuples

A **tuple** is an ordered, possibly empty, collection of values.

The tuple whose components are \( v_1, \ldots, v_n \) (\( n \geq 0 \)) is written \( (v_1, \ldots, v_n) \).

- A tuple is constructed by an expression of the form \( (E_1, \ldots, E_n) \).

If \( E_1 \) has type \( \tau_1 \), and \( \ldots, E_n \) has type \( \tau_n \) then \( (E_1, \ldots, E_n) \) has type \( \tau_1 \times \cdots \times \tau_n \).

The **empty tuple** is given by \( () \) which is of **unit** type:

```plaintext
- ();
> val it = () : unit
```

- The components of a non-empty tuple can be **selected** (or **projected**).

- With functions, tuples give the effect of multiple arguments and/or results.

Complex numbers

```plaintext
load"Math" ; (* needed in mosml, but not in sml *)
type complex = real * real ;
val origin = ( 0.0 , 0.0 ) : complex ;
fun X( (x,y):complex ) = x ;
fun Y( (x,y):complex ) = y ;
fun norm v = Math.sqrt( X(v)*X(v) + Y(v)*Y(v) ) ;
fun scalevec( r , v ) = ( r*X(v) , r*Y(v) ) ;
fun normal v = scalevec( 1.0/(norm v) , v ) ;
```

- "p04"
> val it = () : unit
> type complex = real * real
> val origin = (0.0, 0.0) : real * real
> val X = fn : real * real -> real
> val Y = fn : real * real -> real
> val norm = fn : real * real -> real
> val scalevec = fn : real * (real * real) -> real * real
> val normal = fn : real * real -> real * real

In ML the keyword \texttt{op} overrides infix status:

\begin{verbatim}
- op xor;
> val it = fn : bool * bool -> bool
- op xor ( true , false ) ;
> val it = true : bool
\end{verbatim}
Fibonacci numbers

\[ F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-2} + F_{n-1} \quad (n \geq 2) \]

\begin{verbatim}
fun nextfib(Fn, Fsuccn) : int * int = (Fsuccn, Fn + Fsuccn);

fun fibpair n = if n = 1 then (0, 1) else nextfib(fibpair(n-1));

> val nextfib = fn : int * int -> int * int
> val fibpair = fn : int -> int * int
\end{verbatim}

Power-of-two test

\begin{verbatim}
fun powoftwo n = (n=1) orelse ((n mod 2 = 0) andalso powoftwo(n div 2));

> val powoftwo = fn : int -> bool
\end{verbatim}

Mutual recursion

Examples

\[ \pi = \frac{1}{4} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots + \frac{1}{4k+1} - \frac{1}{4k+3} + \cdots \]

\begin{verbatim}
fun pos k = if k < 0 then 0.0
  else (if k = 0 then 0.0 else neg(k-1)) + 1.0/real(4*k+1)

and neg k = if k < 0 then 0.0
  else pos(k) - 1.0/real(4*k+3);

> val pos = fn : int -> real
  val neg = fn : int -> real
\end{verbatim}

Parity test

\begin{verbatim}
fun even n = n = 0 orelse odd(n-1)
and odd n = n<>0 andalso (n = 1 orelse even(n-1));

> val even = fn : int -> bool
  val odd = fn : int -> bool
\end{verbatim}
Evaluation of expressions

Execution is the *evaluation* (or *reduction*) of an expression to its value, replacing equals by equals.

**Evaluation of conditionals**

To compute the value of the conditional expression if E then E1 else E2, first compute the value of the expression E. If the value so obtained is *true* then return the value of the computation of the expression E1; otherwise, return the value of the computation of the expression E2.

The evaluation rule in ML is *call-by-value* (or *strict* evaluation).

**Call-by-value evaluation**

To compute the value of \( F(E) \), first compute the value of the expression \( F \) to obtain a function value, say \( f \). Then compute the value of the expression \( E \), say \( v \), to obtain an actual argument for \( f \). Finally compute the value of the expression obtained by substituting the value \( v \) for the formal parameter of the function \( f \) into its body.

**NB:** Most purely functional languages adopt *call-by-name* (or *lazy* evaluation).

The manual evaluation of expressions is helpful when understanding and/or debugging programs.

**Examples**

1. `fun minORmax b = (if b then Int.min else Int.max)(1+3,2);`
   
   `minORmax true`  
   \[ \sim (\text{if true then Int.min else Int.max})(1+3,2) \]  
   \[ \sim (\text{if true then Int.min else Int.max})  
   \]  
   \[ \sim \text{Int.min} \]  
   \[ (1+3,2) \]  
   \[ \sim 4 \]  
   \[ \sim (4,2) \]  
   \[ \sim \text{Int.min}(4,2) \]  
   \[ \sim 2 \]

2. `fact(1-1)`  
   \[ 1 - 1 \sim 0 \]  
   \[ \text{if } 0 = 0 \text{ then } 1 \text{ else } 0 * \text{fact}(0-1) \]  
   \[ 0 = 0 \sim \text{true} \]  
   \[ \sim 1 \]
   \[ \sim 1 \]

For succinctness, the above is typically abbreviated as follows:

`fact(1-1)`  
\[ \sim \text{fact } 0 \]  
\[ \sim \text{if } 0 = 0 \text{ then } 1 \text{ else } 0 * \text{fact}(0-1) \]  
\[ \sim 1 \]
In this vein, thus

\[
\text{fact}(3) \\
\leadsto \text{if } 3 = 0 \text{ then } 1 \text{ else } 3 \times \text{fact}(3-1) \\
\leadsto 3 \times \text{fact}(3-1) \\
\leadsto 3 \times \text{fact}(2) \\
\leadsto 3 \times ( \text{if } 2 = 0 \text{ then } 1 \text{ else } 2 \times \text{fact}(2-1) ) \\
\leadsto 3 \times ( 2 \times \text{fact}(2-1) ) \\
\leadsto 3 \times ( 2 \times \text{fact}(1) ) \\
\leadsto 3 \times ( 2 \times ( \text{if } 1 = 0 \text{ then } 1 \text{ else } 1 \times \text{fact}(1-1) ) ) \\
\leadsto 3 \times ( 2 \times ( 1 \times \text{fact}(1-1) ) ) \\
\leadsto 3 \times ( 2 \times 1 ) \\
\leadsto 3 \times 2 \\
\leadsto 6
\]

\textbf{NB}: Due to call-by-value, one cannot define an ML function \texttt{cond} such that \texttt{cond(E,E1,E1)} is evaluated like the conditional expression \texttt{if E then E1 else E2} for whatever expressions \texttt{E}, \texttt{E1}, \texttt{E2}. 