ML Tutorial 2

Polymorphism, Functions, Exceptions
I/O, Modules

Types Review

Primitive types

unit, int, real, char, string, ..., instream, outstream, ...

Composite types

unit, tuples, records

function types

Datatypes

types and n-ary type operators, tagged unions, recursive

nominal type equality

bool, list

user defined: trees, expressions, etc.

Type Abbreviations

types and n-ary type operators

structural type equality

```
type ‘a pair = ‘a * ‘a
```

Type Inference

When defining values (including functions), types do not need to be declared -- they will be inferred by the compiler.

```
- fun f x = x + 1;
val f = fn : int -> int
```

Inconsistencies will be detected as type errors.

```
- if 1<2 then 3 else 4.0;
stdIn:2.1-2.23 Error: types of rules don't agree
  earlier rule(s): bool -> int
  this rule: bool -> real
  in rule:
    false => 4.0
```

Type Inference

In some cases involving record field selections, explicit type annotations (called ascriptions*) may be required*

```
- datatype king = {name: string,
                   born: int,
                   crowned: int,
                   died: int,
                   country: string}

- fun lifetime(k: king) =
=     #died k - #born k;
val lifetime = fn : king -> int

- fun lifetime({born,died,...}: king) =
=     died - born;
val lifetime = fn : king -> int
```

partial record pattern

Polymorphic Types

The most general type is inferred, which may be polymorphic

```
- fun ident x = x;
val ident = fn : ‘a -> ‘a

- fun pair x = (x, x);
val ident = fn : ‘a -> ‘a * ‘a

- fun fst (x, y) = x;
val ident = fn : ‘a * ‘b -> ‘a

- val foo = pair 4.0;
val foo : real * real

- fst foo;
val it = 4.0: real
```

Polymorphic Types

The most general type is inferred, which may be polymorphic

```
- fun ident x = x;
val ident = fn : ‘a -> ‘a

- fun pair x = (x, x);
val ident = fn : ‘a -> ‘a * ‘a

- fun fst (x, y) = x;
val ident = fn : ‘a * ‘b -> ‘a

- val foo = pair 4.0;
val foo : real * real

- fst foo;
val it = 4.0: real
```

type variable

polymorphic type

```
: real -> real * real
```

Polymorphic Data Structures

```
- infixr 5 ::
- datatype ‘a list = nil
                   | :: of ‘a * ‘a list

- fun hd nil = raise Empty
=   | hd (x::_) = x;
val hd = fn : ‘a list -> ‘a

- fun length nil = 0
=   | length (_::xs) = 1 + length xs;
val length = fn : ‘a list -> int

- fun map f nil = nil
=   | map f (x::xs) = f x :: map f xs;
val map = fn : (’a -> ‘b) -> ‘a list -> ‘b list
```

More Pattern Matching

Layered Patterns: x as *pat*

```
(*  merging two sorted lists of ints  *)
fun merge(x, nil) = x
  | merge(nil, y) =
  | merge(l as x::xs, m as y::ys) =
      if x < y then x :: merge(xs,m)
      else if y < x then y :: merge(l,m)
      else x :: merge(xs,ys);
val merge = fn : int list * int list -> int list
```

Note: although < is overloaded, this definition is unambiguously typed with the lists assumed to be int lists because the < operator defaults to the int version (of type `int*int->bool`).

Exceptions

```
- 5 div 0;                              (* primitive failure *)
uncaught exception Div

exception NotFound of string;    (* control structure *)
type ‘a dict = (string * ‘a) list
fun lookup (s,nil) = raise (NotFound s)
  | lookup (s,(a,b)::rest) =
      if s = a then b else lookup (s,rest)
val lookup: string * ‘a dict -> ‘a

val dict = [(”foo”,2), (”bar”,~1)];
val dict: string * int list              (* == int dict *)

val x = lookup(”foo”,dict);
val x = 2 : int

val y = lookup(”moo”,dict);
uncaught exception NotFound

val z = lookup(”moo”,dict) handle NotFound s =>
        (print (“can’t find “^s^”\n”); 0)
can’t find moo
val z = 0 : int
```

References and Assignment

```
type ‘a ref
val ref : ‘a -> ‘a ref
val ! : ‘a ref -> ‘a
val := : ‘a ref * ‘a -> unit

val linenum = ref 0;    (* create updatable ref cell *)
val linenum = ref 0 : int ref

fun newLine () = linenum := !linenum + 1;  (* increment it *)
val newline = fn : unit -> unit

fun lineCount () = !linenum;  (* access ref cell *)
val lineCount = fn : unit -> int

local val x = 1
   in fun new1 () = let val x = x + 1 in x end
  end  (* new1 always returns 2 *)

local val x = ref 1
   in fun new2 () = (x := !x + 1; !x)
  end  (* new2 returns 2, 3, 4, ... on successive calls *)
```

Input/Output

```
structure TextIO : sig

type instream                      (* an input stream *)
type outstream                     (* an output stream *)

val stdIn : instream               (* standard input *)
val stdout : outstream             (* standard output *)
val stdErr : outstream             (* standard error *)

val openIn: string -> instream     (* open file for input *)
val openOut: string -> instream    (* open file for input *)
val openAppend: string -> instream (* open file for appending*)

val closeIn: instream -> unit      (* close input stream *)
val closeOut: instream -> unit     (* close output stream *)

val output: outstream * string -> unit  (* output a string *)

val input: instream -> string      (* input a string *)
val inputLine: instream -> string  (* input a line *)
.....
end
```