ML Lectures (cont.)

Winter 2007

Acknowledgments:
1. Standard ML of New Jersey website: www.smlnj.org
2. Programming in Standard ML. by Robert Harper.
3. Concept in Programming Lang. by John C. Mitchell

Type Synonym

We can give existing types new names.

$\boxed{\text{Syntax: } \texttt{type } \textit{tycon} = \textit{ty}}$

tycon becomes an alias (synonym) for the existing type *ty*.

```
1   -type float = real;
    type float = real

2   -type count = int  and  average=real;
    type count = int
3   type average = real
    ??
4   -val f: float = 2.3;
    val f=2.3: float
5   -val i:count = 3;
   val i = 3: count
```

Type Synonym (continue...)

But notice `float`, `real`, and `average` are all of the same base type, i.e. `real`!

```
6   -val a:average = f;
    val a = 2.3: average

7   -val res = a+f;
    val res = 4.6: average
```

Type synonyms make program more readable.

```
8 -type car= {make:string, built:int};
   ??

9 -val c1: car = {make="Toyota", built=2001};
  ??

10 -fun nextModel ({make=n,built=y}:car)= y+1;
   val newxtModel = fn : car -> int

11 - nextModel c1;
   ??
```

User defined datatypes

General Syntax:

```
datatype tycon = cons1 of ty1
                |cons2 of ty2
                ...
                |consn of tyn
```

- Defines a **new** type called `tycon`.
- `tyi`'s are previously defined types..
- `consi`'s are *constructors*. They are used to create a value of `tycon` type

Note: "`of tyi`" is omitted if a constructor does not need any argument (such constructors are called *constants*).

Enumerated Types

When all constructors are constants (no argument).

Example:

```
-datatype color = Red|Blue| Green;
datatype color = Blue | Green | Red

-val c=Red;  (*calling constructor Red*)
val c=Red: color;

-fun colorStr(Red)= "Red"
   | colorStr(Blue)= "Blue"
   | colorStr(Green)= "Green";
val colorStr = fn : ??

-colorStr(c);
val it= ??
```

Variant Types

Can create union of different types:

```
-datatype number = r of real
                 | i of int;
datatype number= i of int | r of real

-val n1 = i 2;
val n1 = i 2 : number

-val n2 = r 3.0;
??

-val lst=[r 2.2, i 3, i 4, r 0.1];
val lst = [r 2.2, i 3, i 4,r 0.1]: ??

-fun sumInts ([])=0
    |sumInts (i x::rest)= x+ sumInts rest
    |sumInts (r x::rest)= sumInts rest;
val sumInts = fn : ??

-sumInts lst;
??
```

Recursive Types

A datatype can be recursive: e.g. **linked list**.

```
-datatype llist= Nil | Node of int*llist;
datatype llist = Nil | Node of int*llist

-val x = Nil;
val x=Nil: ??

-val y = Node (5, Nil);
??
-val z = Node(3, Node(2,Node(1,Nil)));
??
(*computing the length of a linked list*)
-fun len Nil =0
    |len(Node(_,rest))= 1 + len rest;
val len = fn : ??

-len z;
??
```

Recursive Types (continue...)

Example: a *polymorphic* linked list

```
-datatype 'a llist= Nil|Node of 'a*('a llist);

-val x = Nil;
val x=Nil: ??

-val y = Node (5, Nil);
val y = Node (5,Nil) : ??

-val z = Node("Test", Node("B",Nil));
 ???
```

A binary tree where only leaves have data:

```
-datatype 'a tree= L of 'a
                 | N of ('a tree)*('a tree);
-val mytree= N(L(1),N(L(2),L(3)));

-fun max (x,y)= if x>y then x else y;
-fun depth(L _)=0
    |depth(N(ltree,rtree))=
         1+max (depth ltree, depth rtree);
```

Mutual Recursive Types

Want to represent a tree with arbitrary #of branches.

See the diagram first ...

Defining mutually recursive datatypes (using **and**).

```
-datatype tree = Empty| Node of int*forest
     and forest= Nil  | Cons of  tree*forest
datatype tree = Empty | Node of int * forest
datatype forest = Cons of tree * forest | Nil

-val t1=Node(2,Nil);
??
-val t2=Node(3,Nil);
??
-val t3=Node(7,Cons(t1,Cons(t2,Nil)));
??
-val t4=Node(5,Nil);
??
-val t5=Node(1,Nil);
??
-val t6=Node(2,Cons(t5,Cons(t4,Cons(t3,Nil))));
??
```

Mutual Recursive Types: function example...

We want to count how many nodes are in a tree.

solution: `1+` #of nodes in its subtrees (i.e. forest)

```
-fun numnodeT (Empty)=0
    | numnodeT (Node(data,f))= 1+ numnodeF(f)
  and
      numnodeF(Nil) = 0
     |numnodeF(Cons(t,f))= ???


  val numnodeT = fn : tree -> int
  val numnodeF = fn : forest -> int

(* Note that numnodeT and numnodeF are
  mutually recursive.*)

  -numnodeT(t6)
  ??
```