ML Lectures (cont.) – Slides III

Fall 2005

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

Type Synonym

We can give existing types new names.

Syntax: `type` *tycon* = *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:

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

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

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

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


Variant Types

Can create union of different types:

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

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

4  -val n2 = r 3.0;
   ??

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

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

9  -sumInts lst;
   ??
```

Recursive Types

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

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

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

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

7  -len z;
   ??
```


Recursive Types (continue...)

Example: a *polymorphic* linked list

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

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

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

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

A binary tree where only leaves have data:

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

8  -fun max (x,y)= if x>y then x else y;
9  -fun depth(L _)=0
10     |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**).

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

3 -val t1=Node(2,Nil);
  ??
4 -val t2=Node(3,Nil);
  ??
5 -val t3=Node(7,Cons(t1,Cons(t2,Nil)));
  ??
6 -val t4=Node(5,Nil);
  ??
7 -val t5=Node(1,Nil);
  ??
8 -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)

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


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

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

6  -numnodeT(t6)
   ??
```