CS109A ML Notes for 3/6/96

Maintaining a State

ML dictions learned so far do not allow us to change the value of variables as a side-effect of function evaluation. Here are two ways to do so:

1. *Arrays.* Essentially 1-dimensional arrays as in C.
2. *References.* Something like a pointer to a variable, allowing the value of that variable to change.

Arrays

First, to use arrays at all, you have to open a special module named `Array`. The ML statement to do so is

```
open Array;
```

ML responds with the available functions. The most important:

- `val A = array(n,v);` makes A an array indexed by $0, 1, \ldots n-1$. Initially, each element holds value v.
  - □ We must be able to determine the exact type of v at this moment; e.g., `nil` is not a suitable value, but `nil:int list` is.
- `sub(A,i)` returns the value of `A[i]`.
- `update(A,i,v);` is like `A[i] = v` in C. It returns the unit.

Example: This is an extended example of an ML program to heapify an array.

```
open Array;
    (List of all functions, etc., in Array)
val MAX = 100;
val MAX = 100 : int
val A1 = array(MAX+1,0);
val A1 = [|0,0,0,0,0,0,0,0,0,0,0,...|] : int array
```

The above makes arrays available, sets MAX to the largest number of elements we'll tolerate in a heap, and then creates an array A1 to serve as a heap.

- Note that because A[0] is not used, we need $MAX+1$ elements.

- Initially all elements are 0, and ML can infer that the type of elements is int.

```
fun swap(i,j,A) =
    let
        val Ai = sub(A,i);
        val Aj = sub(A,j);
    in
        (update(A,i,Aj); update(A,j,Ai))
    end;
```

*val swap = fn : int * int array → unit*

The above swaps $A[i]$ with $A[j]$ by:

- Copy $A[i]$ and $A[j]$ into variables Ai and Aj.

- Execute two update statements that set A[i] = Aj and then A[j] = Ai.

  - ☐ A swap using a single temp is possible, mimicking Fig. 5.46, FCS.

```
fun bubbleUp(i,A) =
    if i<=1 then ()
    else
        let
            val i2 = i div 2;
        in
            if (sub(A,i):int) > sub(A,i2) then
                (swap(i,i2,A); bubbleUp(i2,A))
            else ()
        end;
```

*val bubbleUp = fn : int * int array → unit*

$bubbleUp(i, A)$ bubbles $A[i]$ as far forward in the array as it will go.

- If $i = 1$, we are done.

- Otherwise, compare $A[i]$ with $A[i/2]$. If the former is larger, swap and continue bubbling up.

- Otherwise, just return the unit.

```
fun heapify(nil,A,n) = A
|   heapify(x::xs,A,n) = (
        update(A,n,x);
        bubbleUp(n,A);
        heapify(xs,A,n+1)
    );
```
*val heapify = fn : int list * int array → int array*

$heapify(L, A, n)$ inserts a list L of elements into array A, starting at $A[n]$, and heapifies as it goes, assuming A was a heap to begin with.

```
heapify([1,2,3,4,5,6,7,8,9,10], A1, 1);
```
val it = [|0,10,9,6,7,8,2,5,1,4,3,0,...|] : int array

This is a sample call, putting integers 1 through 10 into an empty array $A1$.

References

ML allows a variable to be declared a *reference* to values of some type T.

- There are limits on the acceptable types, but nonfunction types T are OK.

- Unlike other ML types, it is possible to change the value to which a ref variable refers.

- Declare a reference variable by:

```
val s = ref "foo";
```
val s = ref "foo" : string ref

- Change the value referred to by:

```
s := "bar";
```
val it = () : unit

```
s;
```
val s = ref "bar" : string ref

- Access the value of a ref variable by the *dereferencing operator* `!` .

```
!s;
```
val it = "bar" : string

While-Do Loops

An effective way to use ref variables.

- Same idea as while-loops in C.

Example: Assuming module `Array` is open, we can create an array of ten integers and initialize $A[i]$ to i^2 as follows.

```
val i = ref 0;
```
val i = ref 0 : int ref

```
val A = array(10,0);
```
val A = [|0,0,0,0,0,0,0,0,0,0|] : int array

```
while !i < 10 do (
    update(A, !i, (!i)*(!i));
    i := !i + 1
);
```
val it = () : unit

```
A;
```
val it = [|0,1,4,9,16,25,36,49,64,81|] : int array

- You might think that you could use `val` to assign new values to variables, but it is illegal to do so in the expression after a `do` (or in any but a few contexts such as a let-statement).
- The following is *illegal* and results in a syntax error:

```
(* ILLEGAL CODE *)
val A = array(10,0);
val i = 0;
while i < 10 do (
    update(A, i, i*i);
    val i = i + 1
);
```
Error: syntax error found at VAL

Why does ML refuse to provide a feature that "makes sense"?

- Allowing indescriminate assignments in the middle of execution would make it impossible for ML to discover types when it compiles your program.
  - ☐ You would lose the benefits, notably semantic errors could no longer be caught at compile-time.