CSL101 SML : Recursion and Lists

Abhishek Thakur & S. Arun-Kumar

September 11, 2006

1 Revisiting Recursion

In particular we look at the functions

- **fact(n)** : factorial of n
- **exp(x,n)** : x raised to n
- **fib(n)** : the n^{th} fibonacci number
- **real(n)** : converting a positive integer to a real number

1.1 Recursive Functions

The recursive ML programs were (defining 0^0 as 1)

```
fun fact(n)  = if n=0 then 1
               else n*fact(n-1)
fun exp(x,n) = if n=0 then 1
               else x*exp(x,n-1)
fun fib(n)   = if n=1 then 1
               else if n=2 then 1
               else fib(n-1) + fib(n-2)
fun real(n)  = if n=0 then 0.0
               else 1.0 + real(n-1)
```

1.2 Tail Recursive Functions

The tail recursive ML program for factorial was

```
fun fact1(n,result)  = if n=0 then result
                       else fact1(x-1,x*result)
```

On similar lines, you were asked to define exp1, fib1 and real1

```
fun exp1(x,n,result) = if n=0 then result
                       else exp1(x,n-1,x*result)
fun fib1(n,result1,result) = if n=1 then result+result1
                             else fib1(n-1,result,result+result1)
fun real1(n,result)  = if n=0 then result
                       else real1(n-1,result+1.0)
```

These functions, however, used extra parameters, and it isn't elegant to keep this visible at the top level. So we define another set of functions to hide this fact.

```
fun fact2(n)  = fact1(n,1)
fun exp2(x,n) = exp1(x,n,1)
fun fib2(n)   = fib1(n,0,1)
fun real2(n)  = real1(n,0.0)
```

1.3 Using let .. in .. end

Finally we know we could define fact1 inside fact2

```
fun fact2(n) = let
                   fun fact1(n,result) = ...
               in
                   fact1(n,1)
               end
```

1.4 A Small Test

We shall conclude this section with a small test to check your understanding of the above.

Exercise 1 *Let the* **reverse** *of a positive integer be the digits in reverse order, with any leading* 0*'s removed. i.e. reverse(9876) = 6789, reverse(1010) = 101, and reverse(40000) = 4. You need to define*

1. *A technically complete algorithmic definition for* **reverse** *using only integer operations.*
2. *A recursive function* **reverse(n)**
3. *An equivalent tail-recursive function* **reverse1(n,result)**

TIME : 30 minutes HINT : Use 'div' and 'mod'

Exercise 2 *The empty string is defined as "", and two strings are appended using "^" (the circumflex – the symbol found in the row of the main keyboard containing the numeral* 6*).*

```
- val a = "";
val a = "" : string
- val b = "a"^a;
val b = "a" : string
- val c = "01101";
val c = "01101" : string
- val d = "0101"^"1010";
val d = "01011010" : string
```

Given a positive integer, we require its **binary** *equivalent as a string, i.e. binary(4) = "100", binary(15) = "1111", and binary(27) = "11011". You need to define*

1. *A technically complete recursive function* **binary(n)**
2. *An equivalent tail-recursive function* **binary1(n,result)**

TIME : 30 minutes