Higher Order Functions

Abhishek Thakur and S. Arun-Kumar

November 6, 2006

In this session, we shall look at various kinds of higher-order functions and analyze their time/space complexities.

1 Examples

1.1 Lexicographic minimum list

Suppose we were required to find the lexicographically minimum list given a list of integer lists. We proceed in a way similar to finding the minimum tuple from a list of tuples, and add a small function to compare two lists.

```
1  exception EmptyList
2  fun min_list([]) = raise EmptyList
3    | min_list(x::ls) =
4        let
5           fun less (l1   ,[]   ) = false
6             | less ([]   ,l2   ) = true
7             | less (a::l1,b::l2) =
8                       if a<b then true
9                       else if b<a then false
10                           else less(l1,l2)
11          fun min_help([],result) = result
12            | min_help(x::ls,result) =
13                      if less(x,result) then min_help(ls,x)
14                      else min_help(ls,result)
15      in
16          min_help(ls,x)
17      end
```

1.2 Two halves of a list

Suppose we have to divide a list into two equal halves, the first containing $\lceil n/2 \rceil$ elements, and the second $\lfloor n/2 \rfloor$ elements, in the same order. The crux is similar to *rev_helper(ls,result)*, except that we also keep a count of the number of elements seen. The moment we finish looking at the $\lceil n/2 \rceil^{th}$ element, we return *(rev result, ls)*.

```
1 fun halves(ls) =
2       let
```

```
3           fun half_help([]    ,n ,part1) = (rev(part1),[])
4             | half_help(x::ls ,n ,part1) = if n=0 then (rev(part1), x::ls)
5                                            else half_help(ls,n-1,x::part1)
6       in
7           half_help(ls ,(length(ls)+1) div 2 ,[])
8       end
```

2 Higher Order Functions

2.1 Merge Sort

We shall start by revisiting merge-sort, used to sort a list given an ordering on the elements.

```
1  exception NoHalves
2
3  fun merge less [] l2 = l2
4    | merge less l1 [] = l1
5    | merge less (x::l1) (y::l2)  =
6                 if less(x,y) then x::(merge less l1 (y::l2))
7                 else if less(y,x) then y::(merge less (x::l1) l2)
8                      else merge less (x::l1) l2
9
10 fun halves [] = raise NoHalves
11   | halves (ls) =
12       let
13           fun half_help ([],n,first)    = (rev first, [])
14             | half_help (x::ls,n,first) = if n=0 then (rev first, x::ls)
15                                           else half_help (ls, n-1, x::first)
16       in
17           half_help(ls,(length(ls)+1) div 2,[])
18       end
19
20 fun mergesort less [] = []
21   | mergesort less (x::[]) = [x]
22   | mergesort less (x::ls) =
23       let
24              val (half1,half2) = halves(x::ls)
25              val l1 = mergesort less half1
26              val l2 = mergesort less half2
27       in
28              merge less l1 l2
29       end
```

3 Higher Order Functions on Lists

mergesort is a higher order function which takes a comparison function, a list, and sorts the list using the total ordering given by the function.

3.1 Map

Next we look at the function *map*

```
- map;
val it = fn : ('a -> 'b) -> 'a list -> 'b list
```

It takes a function, a list, and applies the function to every element of the list. It is defines as

```
1 fun map f [] = []
2   | map f (x::ls) = (f x) :: (map f ls)
```

3.2 Exercises

Exercise 1 *FOLDR*

Define a function 'foldr' : $(\alpha \star \beta \rightarrow \beta) \rightarrow \beta \rightarrow \alpha$ list $\rightarrow \beta$, which takes a function, an initial value, and folds the list from right to left.

As an example, foldr div 1 [20,12,6,2] = 5, because 5 = 20 div (12 div (6 div (2 div (1))). Here α and β are both 'int'.

Exercise 2 *FOLDL*

Define a corresponding function 'foldl'. What is its type? What would a function such as 'foldl' do? Why is it useful to have such a function? Give some examples of its use.

Exercise 3 *PARTITION*

Define the function 'partition' : $(\alpha \rightarrow$ bool$) \rightarrow \alpha$ list $\rightarrow \alpha$ list $\star \alpha$ list, used to split a list depending on a function from α to bool.

As an example, partition odd [1,2,3,4,5,6] = ([1,3,5],[2,4,6]), where odd : int $\rightarrow$ bool is the usual function to check if an integer is odd.

Exercise 4 *TABULATE*

Define the function 'tabulate' : int $\star$ (int $\rightarrow \alpha) \rightarrow \alpha$ list, which takes a positive integer i, a function f from int to α, and returns the list [f(1),f(2),...,f(i)].

As an example tabulate (3,odd) = [true,false,true], because 1 is odd, 2 is even, and 3 is odd. Similarly tabulate (5,prime) = [false,true,true,false,true], and tabulate (4,inc) = [2,3,4,5].

One can look for various such functions inside the structure 'List', namely 'filter', 'app', 'take', 'drop', etc. Your homework is to code them up in ML in order to become comfortable with higher order functions on lists. In particular try combining various of these functions.

Exercise 5 *Use the function 'filter' to identify the prime numbers by using the method of sieve of Eratostthenes.*