

ML Built-in Functions

Since ML is a functional programming language, many of its built-in functions are concerned with function application to objects and structures.

In ML, built-in functions are curried → they expect their arguments as sequence of objects separated by spaces NOT as a tuple.

The map Function

The map function accepts two parameters: a function and a list of objects. It will apply the given function to each object on the list.

Example:

```
- map (fn x => x + 2) [1,2,3];  
val it = [3,4,5] : int list
```

also works with built-in functions and operators such as the negation function `~ : int -> int`

```
- map ~ [1,2,3];  
val it [~1,~2,~3] : int list
```

The map Function

map can also be applied to a list of structures.

```
- map (fn (a,b) => a + b) [(1,2),(3,4)];  
val it = [3,7] : int list
```

The foldr Function

The foldr function works similar to the map function, but instead of producing a list of values it only produces a single output value.

Syntax:

foldr <binary function> <initial value of output> <list>

Semantics:

```
- foldr f c [x1, x2, ..., xn-1, xn];  
is the same as saying  
- f(x1, f(x2, ..., f(x-1, f(xn, c))...));
```

foldr start at the rightmost object
xn of the list with initial value c

foldr folds a list of values
into a single value starting
with the rightmost element.

The foldr Function

Example:

```
- foldr (fn (a,b) => a+b) 2 [1,2,3];  
  → fn(1,fn(2,fn(3,2)));  
  val it = 8 : int
```

The foldl Function

You guessed it! Works exactly the same as the foldr function except that it start computing at the leftmost element:

```
- foldl f c [x1, x2, ..., xn-1, xn];  
  is the same as saying  
  - f(xn, f(xn-1, ..., f(x2,f(x1,c))...));
```

foldl folds a list of values into a single value starting with the leftmost element.

Example:

```
- foldl (fn (a,b) => a+b) 2 [1,2,3];  
  => fn(3,fn(2,fn(1,2)));  
  val it = 8 : int
```

foldr and foldl

In most cases foldr and foldl will produce the same results, but consider the following:

```
- foldr (fn (a,b) => a^b) "ef" ["ab","cd"];  
=> fn("ab",fn("cd","ef"))  
=> "ab"^( "cd" ^ "ef")  
=> "ab" ^ "cd" ^ "ef"  
=> "abcdef"  
val it = "abcdef" : string
```

```
- foldl (fn (a,b) => a^b) "ef" ["ab","cd"];  
=> fn("cd",fn("ab","ef"))  
=> "cd"^( "ab" ^ "ef")  
=> "cd" ^ "ab" ^ "ef"  
=> "cdabef"  
val it = "cdabef" : string
```

foldr and foldl will only produce the same results if the mapped function is commutative.

Homework and Reminders

Homework for Monday March 6th:

7.2 – use pattern matching

9.6 – use built-in functions (see instructions at the beginning of the problem set)

9.26 – do NOT use built-in functions

Midterm

March 8th – chapters 1 through 9

Review

Week 1

Chapter 1: Programming Languages

features of languages, classes of languages

Chapter 2: Defining Program Syntax

grammars, derivations, formal definition of languages, sentences

Week 2

Chapter 3: Where Syntax Meets Semantics

parse trees as semantics, ambiguous grammars

Chapter 4: Language Systems

structure of IDE/compiler, difference between compiler/interpreter

Week 3

Chapter 5: A First Look At ML

basic expression, tuples, lists

Chapter 6: Types

** a type is a set of values **

Week 4

Chapter 7: A Second Look At ML

patterns

Chapter 8: Polymorphism

overloading, parameter coercion, parametric polymorphism, subtype polymorphism

Week 5

Chapter 9: A Third Look At ML

higher-order programming