函数式编程原理

Lecture 6

参考CMU 15-150 Lecture10-slides.pdf

上节课内容回顾

- tree类型排序算法的并行性能分析 (Span)
- 类型检测 (Type checking)
- 多态特点 (Polymorphism)
- 类型推导 (Type inference)

本节课主要内容

- 函数作为值 (Function as values)
- 高阶函数 (Higher-order functions)
- “多态”的力量(The power of polymorphism)
  - list数据的单独求解问题——map
  - list数据的联合求解问题——foldr，foldl
  - 高阶函数编程实例

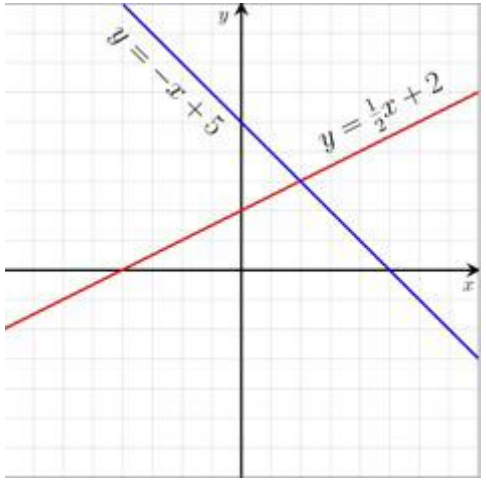
数据标准化问题

数据标准化(归一化)：将不同表征的数据归约到相同的尺度内。其目的在于将各指标统一到同一数量级，以便消除指标之间的量纲影响，进行综合对比评价。

根据求解区间的最小值a和最大值b，求线性函数（形如y=α*x+β (α,β为实数) 的函数）
f : real -> real，使f(a) = ~1.0，f(b) = 1.0

即：norm(a, b) =>* 线性函数f满足： f(a) = ~1.0 and f(b) = 1.0

求解 α, β 使其满足
α*a + β = ~1.0 and α*b + β =1.0




```
fun norm(a, b) = fn x => (2.0 * x - a - b) / (b - a)
- val norm = fn : real * real -> real -> real
```

函数norm的扩展使用

```
fun norm(-2.0,2.0) = fn x => x / 2.0
```

- 对实数对（实数二元组）进行归一化处理：

  利用norm(~2.0, 2.0)，将(1.0,1.5) 处理为 (0.5, 0.75)

```
fun normpair(a, b) =
    fn (x,y) => (norm(a,b) x, norm(a,b) y)
```

- 对实数表中的每个元素进行归一化处理：

  利用norm(~2.0, 2.0) ，将[1.0,1.5,1.8]处理为[0.5, 0.75, 0.9]

```
fun normlist(a,b)= fn [] => []
    | fn x::L =>(norm(a,b) x) :: (normlist(a,b) L)
```

共性问题分析

- 需求：如何将一个函数应用于某种数据结构中的所有元素
  - 批处理：对每个元素执行相同的操作(调用相同的函数)

——数据结构与函数无关

```
fun normpair(a, b) = fn (x,y) => (norm(a,b) x, norm(a,b) y)

fun fibpair(x, y) = (fib x, fib y)

fun factpair(x, y) = (fact x, fact y)
```

对表结构(list) 如何设计？

思路：

```
fiblist(x::L) = fib x :: fiblist L

factlist(x::L) = fact x :: factlist L
```

进一步思考

- 能否设计一个函数，能分别将不同函数应用于某种数据结构(pairs, tuples, lists ……)中的所有元素？
  - 对pairs，设计“多态”函数：

    pair : ('a -> 'b) -> 'a * 'a -> 'b * 'b
  - 对lists，设计“多态”函数：

    map : ('a -> 'b) -> 'a list -> 'b list

——高阶函数（***higher-order*** functions）

多态 vs. 高阶

- 多态（Polymorphism）类型：简化多类型的相同操作
- 高阶（Higher-order）函数：简化多参数的函数操作

（同类型批量数据的不同函数操作）

高阶是函数“多态”应用的一种体现

pair, list, tree……

map, combining…

对pair的处理

```
pair : ('a -> 'b) -> 'a * 'a -> 'b * 'b
(* REQUIRES true *)
(* ENSURES pair f (x, y) = (f x, f y) *)
```

fun pair f = **fn** (x, y) => (f x, f y)  **fun** pair f (x, y) = (f x, f y)

```
pair(norm (~2.0, 2.0)) : real * real -> real * real
```

对list的处理

```
map : ('a -> 'b) -> ('a list -> 'b list)
(* REQUIRES true *)
(* ENSURES For all n ≥0, map f [x1, ..., xn] = [f x1, ..., f xn] *)
fun map f = fn L =>
              case L of
                [ ] => [ ]
              | x::R => (f x) :: (map f R)
```

```
fun map f [ ] = [ ]
  | map f (x::R) = (f x) :: (map f R)
```

使用pair和map可以很容易的将某个函数应用于pairs和lists中的所有数据

- lists of pairs
- pairs of lists
- pairs of pairs
- lists of lists of pairs

```
map (norm(~2.0, 2.0))  : real list -> real list
map (norm(~2.0, 2.0)) [1.0, 1.5. 2.0] : real list
```

map函数应用——求解子集

给定一个list，求解该list的所有子list。

如：sublists [1,2,3] = [[] ,[3],[2],[2,3],[1],[1,3],[1,2],[1,2,3]]

(* sublists : 'a list -> 'a list list *)
(* ENSURES sublists L = a list of all sublists of L *)

算法思想：

(1) 把list分为两部分，第一个元素和剩余元素

(2) 原list的所有子list为：
剩余元素的子list 并上
把第一个元素加入所有剩余元素子list的list

```
fun sublists [ ] = [ [ ] ]
  | sublists (x::R) =
      let
        val S = sublists R
      in
        S @ map ( fn A => x::A) S
      end
```

另一类问题——批量数据的联合求解

- **real** list求和
- **int** list求乘积
- 寻找**int** list中的最小数
- 寻找**real** list中的最大数

多态函数:

('a * 'b -> 'b) -> 'b -> 'a list -> 'b

算法设计思路：编写递归函数

（1）设定一个初值：z : t_2 ;

（2）设计相应功能函数： F : t_1 * t_2 -> t_2

递归应用于list数据 [x_1,...,x_n] : t_1 list

怎么递归应用于list数据？

foldr 与 foldl

('a * 'b -> 'b) -> 'b -> 'a list -> 'b

```
fun foldl F z [ ] = z
  | foldl F z (x::L) = foldl F (F(x, z)) L
fun foldr F z [ ] = z
  | foldr F z (x::L) = F(x, foldr F z L)
```

foldl F z [x_1,...,x_n] = F(x_n, F(x_{n-1},..., F(x_1,z)...))

foldr F z [x_1,...,x_n] = F(x_1, F(x_2, ..., F(x_n, z)...))

foldr (op @) [ ] [[1,2], [ ], [3,4]]

foldl (op @) [ ] [[1,2], [ ], [3,4]]

联合函数的应用

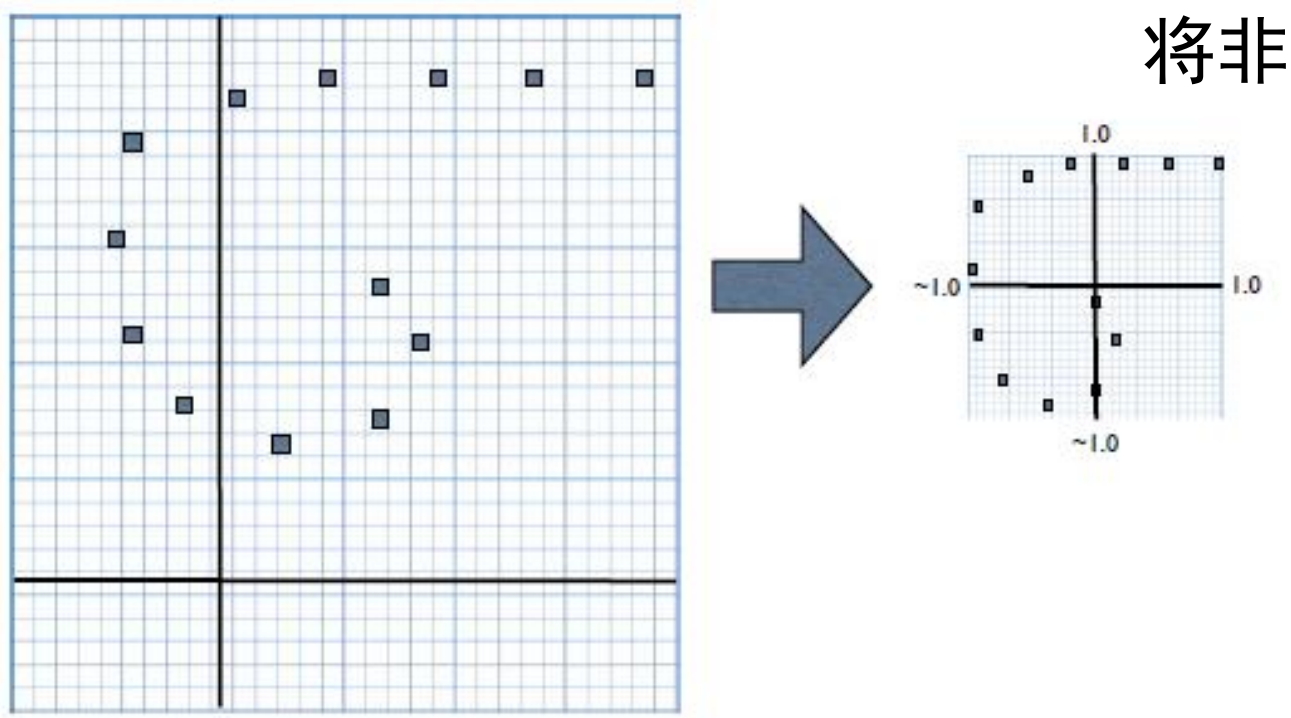
int list求和 (* ENSURES sum L = the sum of the items in L *)

```
sum : int list -> int

fun sum L = foldr (op +) 0 L
```

real list求最大值

```
(* REQUIRES L is a non-empty list          *)
(* ENSURES maxlist L = the largest item in L  *)
fun maxlist (x::R) = foldr Real.max x R

    Real.max : real * real -> real
```

高阶函数应用1——点集数据标准化



将非空、离散的点集标准化至空间：

[~1.0 ... 1.0] X [~1.0 ... 1.0]

求解思路：

1. 分离出x,y;
2. x,y分别norm标准化
    1. 求解x，y的最大/最小值
    2. 每个值norm标准化

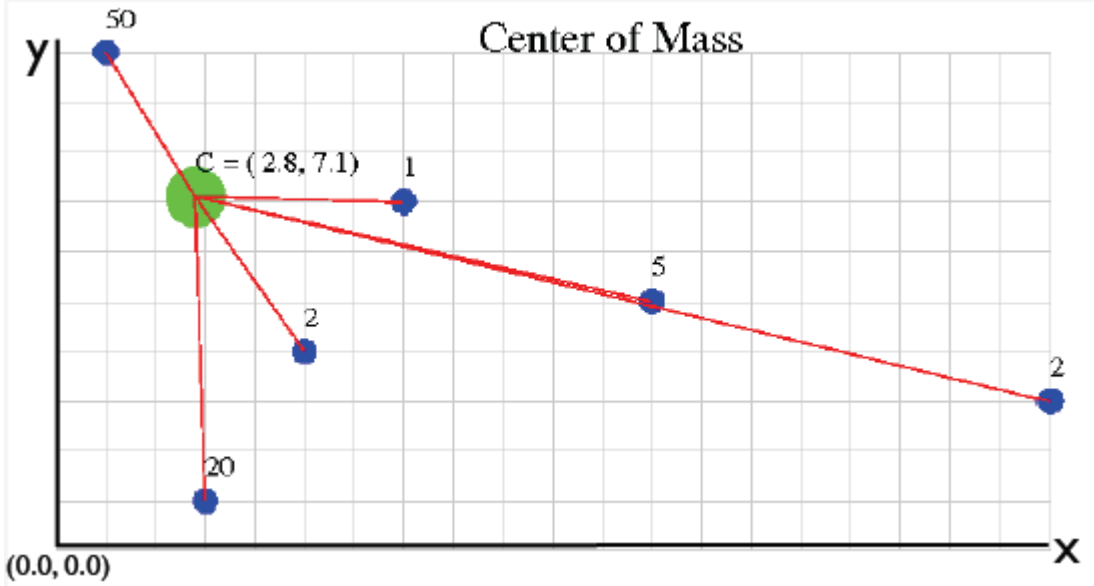
```
fun norm(a, b) = fn x => (2.0 * x - a - b) / (b - a)
```

点集数据标准化——normalize

```
(* normalize : (real * real) list -> (real * real) list *)
(* REQUIRES L is non-empty *)
(* ENSURES normalize L = a list of points in [~1.0...1.0] X [~1.0...1.0] *)

fun normalize (L : (real * real) list) : (real * real) list =
    let
      val xs = map (fn (x,y) => x) L
      val ys = map (fn (x,y) => y) L
      val (xlo, xhi) = (minlist xs, maxlist xs)
      val (ylo, yhi) = (minlist ys, maxlist ys)
    in    map (fn (x,y) => (norm(xlo, xhi) x, norm(ylo, yhi) y)) L
    end
```

```
val (xlo, ylo) = pair minlist (xs, ys)
val (xhi, yhi) = pair maxlist (xs,ys)
```

高阶函数应用2——求解点集中心



给定点集：$[(m_1,(x_1,y_1)),\ldots,(m_n,(x_n,y_n))]$

求解中心点(X, Y): real * real，满足

$$X = (m_1*x_1 + \ldots + m_n*x_n)/M$$

$$Y = (m_1*y_1 + \ldots + m_n*y_n)/M$$

$$M = m_1 + \ldots + m_n$$

给定点集：$[(m_1,(x_1,y_1)),...,(m_n,(x_n,y_n))]$

点集中心的求解

求解中心点(X, Y): real * real，满足

$X = (m_1*x_1 + ... + m_n*x_n)/M$

$Y = (m_1*y_1 + ... + m_n*y_n)/M$

$M = m_1 + ... + m_n$

- 点集的表示：　$[(m_1,(x_1,y_1)),...,(m_n,(x_n,y_n))]$

```
type point = real * real
type body = real * point
```

- 辅助函数：

```
fun add((x1,y1), (x2,y2)):point = (x1+x2, y1+y2)
fun mass (m, (x, y)) = m
fun scale r (m, (x, y))) = (r * m * x, r * m * y)
```

```
fun center (L : body list) : point = let
                                        val M = foldr (op +) 0.0 (map mass L)
                                    In foldr add (0.0, 0.0) (map (scale (1.0/M)) L)
                                    end
```

高阶函数的更多应用

字符串的相关操作：

["all ","your "," base "] → "all your base are belong to us "

foldr (op ^) "are belong to us": string list -> string

[“all ”,“your ”,“base ”] → ["All ","Your ","Base "]

map *capitalize* : string list -> string list

```
explode : string -> char list
implode : char list -> string
Char.toUpper : char -> char
```

```
fun capitalize (s:string) : string =
    let val (x::L) = explode s
    In implode(Char.toUpper x :: L)
    end;
```

通用排序(general sorting)

- Isort, msort : int list -> int list
- Msort : tree -> tree

能否扩展为其他各种数据类型(如int, int*int, string等)的排序?

对公式/表达式进行抽象:

1. 对任意类型的数据，都能够进行比较
2. 对表和树等结构数据进行排序