函数式编程原理

顾琳

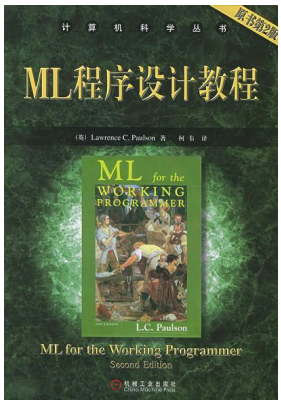
函数式编程原理

- 学时要求： 20+12
- 授课方式：讲授+上机
- 参考资料：CMU 15150
- 涉及内容：
  - 程序正确性/有效性证明
  - 类型,声明,表达式,函数
  - 递归，模式匹配，多态类型检测
  - 高阶函数，惰性求值
  - ……

- 课程目标：
  - 掌握函数式编程方法
  - 掌握程序书写规范，并采用严格的推导方法证明程序的正确性
  - 掌握串/并行程序的性能分析方法
  - 掌握各种数据结构的特点，学会选择合适的数据结构进行功能设计，提高程序效率

参考资料

- 以CMU 15-150课程相关文档为主
  - Lectures、Slice、Labs…
- ML程序设计教程(第2版)，机械工业出版社



感受一下...

程序1:

```
int a, b, r;
void add_abs() {
    scanf("%d %d", &a, &b);
    r = abs(a) + abs(b)
    printf("%d", r);
}
```

程序2:

```
int add_abs(int a, int b) {
    int r = 0;
    r += abs(a);
    r += abs(b);
    return r;
}
```

程序3:

```
int add_abs(int a, int b) {
    return abs(a) + abs(b);
}
```

程序1 是用命令来表示程序, 用命令的顺序执行来表示程序的组合, 不算函数式
程序2 是用函数来表示程序, 但在内部是用命令的顺序执行来实现, 不太函数式
程序3 是用函数来表示程序, 用函数的组合来表达程序的组合, 是完全的函数式编程

什么是函数式编程?

注意我们的判断标准关注的是程序在人(也就是程序员)脑中的状态, 它关心的是某个人在思考问题的时候, 程序是以什么样的概念存在于脑中的 -- 是指令的序列呢, 还是计算呢 -- 而跟程序在机器中执行的状态无关, 毕竟函数式的代码和非函数式的代码最终编译得到的机器码完全有可能是一样的. 所以这里重要的区别在于人如何理解程序, 而非程序如何执行.

所以对于某段代码算不算函数式编程这个问题, 我觉得这跟它用的什么编程语言并没有什么直接的关系 (比如我在程序3中用的同样是非函数式语言), 它跟问题本身有关, 跟具体的实现思路有关, 甚至具体到一个程序员, 它还跟这个程序员如何去解读这段程序有关 (比如当我把程序2中的add_abs看作黑盒的时候, 我忽略它的实现, 而在其它地方把它当做纯函数去使用, 它也完全可能成为函数式代码的一部分).

什么是函数式编程?

函数式编程是一种非常简单的风格

无类型的lambda 演算其实规则很简单 只有三条

1. 变量 (variable)
2. 函数 (lambda-abstraction)
3. 替代 (beta-reduction)

任何支持函数的语言都可以进行函数式风格的编程 注意到与命令式风格不同的是没有赋值，这意味着reason 程序的时候每个变量的值是不变的 不用考虑程序变量随着时间的变化 -- 大大降低了程序的复杂性。

函数式编程的style

1. closure
按照上面的三条规则函数式是first class 的 是可以直接传递作为参数的，

2. 高阶类型推断
因为函数可以作为参数，其类型可以非常复杂 比如下面的函数类型其实非常普遍:

```
val callCC : (('a -> 'b -> 'c) -> ('a -> 'c) -> 'd) -> ('a -> 'c) -> 'd
```

如果没有类型推断，其实很难写对或者理解它的语义

3. tail-call
因为函数式风格没有赋值，也就没有for循环， 要实现循环操作 只能通过递归调用, 比如下面简单的例子:

```
let rec even n = if n = 0 then true else if n = 1 then false else odd (n - 1
and odd n = if n = 1 then true else if n = 0 then false else even (n - 1)
```

函数式编程的好处

```
h(g(f(x))) = (h∗(g∗f))(x) = ((h∗g)∗f)(x)
```

函数式的编程，在我看来，体验上主要就是上面这个表达式。在很多传统的命令式语言里面，你只能按左边这个形式编程。原因有很多，比如很多操作不是表达式，函数不是一等公民，不支持闭包等。

而一个语言如果对函数式支持良好，你就可以很方便的应用右边的形式。

对于函数式风格，你总是在构造一个巨大的表达式（通过组合已有的表达式），然后把数据灌进去，结果就出来了。

而对于命令式风格，你总是在操作数据，等你把数据操作完了，结果就出来了。

进一步的，你还可以操作持有数据的容器。这些容器提供map, filter, reduce, fold之类的高阶方法。它使得你可以在容器上组合函数逻辑。函数天然是可以组合的，天然满足结合律。

函数式编程在使用的时候的特点

1）**你已经再也不知道数据是从哪里来了；**

2）**每一个函数都是为了用小函数组织成更大的函数，函数的参数也是函数，函数返回的也是函数；**

3）**最后得到一个超级牛逼的函数，就等着别人用main函数把数据灌进去了。**