

ftae

EMAIL: ft.ae[AT]outlook.com

(๑•๓•)ง

随笔 - 190 文章 - 0 评论 - 4

Coursera课程 Programming Languages, Part A 总结

Coursera

CSE341: Programming Languages

感谢华盛顿大学 [Dan Grossman](#) 老师 以及 [Coursera](#) 。

碎言碎语

- 这只是 Programming Languages 这门课程第一部分，在 Part A 中通过 [Standard ML](#) 这门编程语言把函数式编程的美妙娓娓道来。在 Part B 以及 Part C 中分别会学习 Racket 以及 Ruby 。
- 这是一门什么样的课呢？首先，**它很好玩**，虽然我学过 Scala 学过一些 Scheme，但还是能够感觉到这门语言的简洁和优雅。但是要记住，这门课最重要的不是教你学会 [Standard ML](#) 这门语言，而是函数式编程中的 immutable data、function closures、first-class function、higher-order function，以及这门语言中的 pattern matching、currying and partial application、type inference 等等（当然并不是这门语言所特有的，这些特性在 Scala 中都存在）。不仅仅是这些实际的东西，在 Part A 的 week 4 最后有几乎一小时的视频介绍了这门课程的动机：为什么要学习函数式编程？为什么选择这三门语言？
- 为什么我会去学习这门课呢？首先这是华盛顿大学的一门 CS 课程，在网络上的评价很高，它的课程质量和作业质量得到了保证。再看看这门课程，它一共使用了三门编程语言，既有函数式编程也有面向对象编程，它并不会试图告诉你这门语言的全部，而是专注于语言特性，一些使得这门语言变得有力 and 优雅的语言特性，对于一门陌生的语言，这门课程首先会引导我们看见它、使用它，然后再尽可能地作出解释，而不仅仅是停留在使用的阶段。
- 作业的形式是编程题目，有自动打分系统，如果额外的 Challenge Problems 以及 Extra Practice Problems 都能完成的话，还是要花不少时间的。完成要求的题目并不难，思考最简洁、最优雅的代码才是最重要的。
- 笔记其实是没多少的，为什么呢？课程中给出的 Reading Notes 真是干货满满，值得反复阅读，它提炼了老师的讲课 ppt，不过 20 页的 pdf，内容却充实的很，而又往往前后呼应，情节跌宕起伏，怎么像是说小说一样。举个例子，它的标题是这样的：**By Name vs. By Position, Syntactic Sugar, and The Truth About Tuples**，是不是很容易有种一探究竟的冲动呢。
- 这门课上到现在直观的感受就是：循序渐进。你不用担心它讲的是废话，因为它总能解决你在前面提出的问题（如果认真去学，问题总是很多的），编程中遇到的问题，总能在某一个知识点找到解决方法，理解其中的为什么。
- 当你遇到问题时，几乎只能用英语去表述问题并去 google，当然结果几乎都来自于 Stack Overflow，有时还需要 [The Standard ML Basis Library](#)。
- 并不是说有用的只有我记下的那么多（而且还加工了不少），考虑到阅读材料中的一些内容已经足够精简了。如果对这门课真有兴趣，不妨去上上看。

笔记

1. All values are expressions. Not all expressions are values.

2. SML 中没有赋值语句(assignment statement)，它只有 变量绑定(variable binding) 以及 函数绑定(function binding)。

公告

如有错误，烦请指正，谢谢！

昵称: ftae

园龄: 3年2个月

粉丝: 2

关注: 2

+加关注

搜索

随笔分类

Codeforces(63)

Emacs(1)

OJ(110)

Scala(2)

Scheme(4)

笔记(5)

随笔(3)

题集 & 题解(4)

随笔档案

2018年10月(3)

2018年7月(1)

2018年2月(2)

2018年1月(7)

2017年12月(8)

2017年11月(11)

2017年10月(12)

2017年9月(9)

2017年8月(33)

2017年7月(42)

2017年6月(29)

2017年5月(33)

阅读排行榜

- Scala 中的隐式转换和隐式参数(1887)
- Coursera课程 Programming Languages, Part A 总结(987)
- poj2763 (树链剖分 - 边权) (874)
- hdu6035 (树形DP) (806)
- N皇后问题 (C++实现和函数式编程实现) (730)

推荐排行榜

- Codeforces 888E - Maximum Subsequence(1)
- Scala 中的隐式转换和隐式参数(1)

66

Bindings are immutable. Given `val x = 8+9`; we produce a dynamic environment where `x` maps to 17. In this environment, `x` will always map to 17; there is no "assignment statement" in ML for changing what `x` maps to. That is very useful if you are using `x`. You can have another binding later, say `val x = 19`; , but that just creates a different environment where the later binding for `x` shadows the earlier one. This distinction will be extremely important when we define functions that use variables.

66

an ML program is a sequence of bindings. Each binding adds to the static environment (for type-checking subsequent bindings) and to the dynamic environment (for evaluating subsequent bindings).

3. 它的讲述模式是这样的：先给出SML内置的常见的数据结构或语法，我们可以很容易去使用。再通过介绍核心的东西（数据结构的本质）以及特定的语法，以至于我们可以自己去定义这种数据结构（`list`、`option` 都可以通过 `datatype` 定义出），重要的是核心的语法是极其少的。

定义 `list`：

```
datatype my_int_list = Empty
                    | Cons of int * my_int_list;

fun append_mylist (xs, ys) =
  case xs of
    Empty => ys
  | Cons(x, xs') => Cons(x, append_mylist(xs', ys));

val lst = Cons(1, Cons(2, Empty));

append_mylist(append_mylist(lst, Cons(1, Cons(3, Empty))), lst);
```

在这里 `Empty` 是 `my_int_list` 类型的值，而 `Cons` 是 `int * my_int_list -> my_int_list` 类型的函数。

4. 为什么函数名和参数间会有一个空格呢？看到阅读材料中的代码，心中不由产生疑惑，尤其是当一个函数只有一个参数时，我们可以直接省略其中的括号，例如：

```
fun f x = x + 1;
f(123);
f 123;
```

定义一个函数，接受参数 `x` 返回 `x+1`，原本只以为是内建语法，一个语法糖而已，等到后面看到了模式匹配，才发现并不是这样。为什么不需要括号呢？因为参数本来就是一个整体，即只有一个参数。

```
fun f (x, y, z) = x + y + z;
```

看起来很正常的一个函数，其实它只是在模式匹配下的一种简化，其实它应该是这样的：

```
fun f(t: int * int * int) =
  case t of (x, y, z) => x + y + z;
```

也就是说它其实是接受一个 `tuple` 类型的参数，将对应位置上的值分别绑定到 `x`, `y`, `z` 三个变量上。

也就是说，在 SML 中一个函数只会接受一个参数(one-argument function)。难怪前面说过我们要抛却以前对于 C、Java 的观念来学习这门课。

更准确地说，函数一定会接受一个参数，那么问题来了，无参数函数该怎么定义呢？其实和其它语言类似：

```
fun f () = "HI";
```

这时候必须加上一个括号了，而这个括号其实是一个 `unit` 类型的值。那么对应的，在调用函数的时候，应该这样：

```
f ();
```

括号是一定不能省略的，否则它输出的是这个函数的类型：`val f = fn : unit -> string`。

其实，`datatype unit = ()` 在 SML 中是预定义的，用的也是前面提到的 `datatype`。

5. 匿名函数

如果我们想要将一个函数作为参数去传递，但并不想把它直接绑定到当前环境中，此时就可以使用匿名函数了。

首先我们可以使用简单的语法来自己模拟匿名函数：

```
fun f1 (addone, x) =  
  addone x;  
  
f1 (let fun f x = x + 1 in f end, 11);
```

当然，我们也可以使用关键字去定义匿名函数。

```
f1 (fn x => x + 1, 11)
```

比起简单的关键词定义，那么我们模拟的这种匿名函数是不是就没有任何意义了呢？其实不然，匿名函数最显著的特点是什么呢，匿名，那么问题来了，没有名字，怎么递归呢。想支持递归，还得有个名字。

```
fun f1 (f, x) =  
  f x;  
  
f1 (let fun f x =  
      case x of  
        0 => 1  
      | q => q * f (q - 1)  
    in f end,  
  7)
```

6. Environments and Closures

首先，不得不提 lexical scope。

我们知道，在 SML 中函数就是值，而函数这个值由两部分组成，组成这个函数的代码（即函数本身），以及我们**创建这个函数时的环境(environment)**，当我调用这个函数时，实际上使用的是**创建这个函数时的环境(environment)**，而那个环境里则可以进行一系列的计算，并产生结果。而这个环境和调用这个函数时的环境则是隔绝的，所以我们可以说一个函数构成了 function closure。

```
val x = 1  
fun f y =  
  let  
    val x = y + 1  
  in  
    fn z => x + y + z  
  end  
val x = 3  
val g = f 4  
val t = 5  
val z = g 6
```

z 的值为 15。

```
val x = 1  
fun f y =  
  fn z => x + y + z  
val x = 3  
val g = f 4  
val t = 5  
val z = g 6 (* 15 *)
```

z 的值为 11。

7. Currying and Partial Application

简单来说，就是一个本身具有多个参数的函数（其实仍是一个参数，即一个 tuple），我们把它变成一个只接受第一个参数，并返回一个接受第二个参数的函数，更多的参数类似。例如：

```
fun fold1 (f, acc, xs) =  
  case xs of  
    [] => acc  
  | x::xs' => fold1 (f, f(acc, x), xs')  
  
fun fold2 f = fn acc => fn xs =>  
  case xs of  
    [] => acc  
  | x::xs' => fold2 f (f(acc, x)) xs'  
  
val sum = fn (x, y) => x + y  
val l = [1, 2, 3, 4]  
val res1 = fold1 (sum, 0, l)  
val res2 = (((fold2 sum) 0) 1)  
val res3 = fold2 sum 0 l
```

与 Scheme 相比, SML 看起来清爽很多(如果你以前写过那种结尾带着成吨的右括号的 Scheme 程序的话)。事实上, SML的括号是可选择的, 即使不加括号, 它也有默认的结合规则, 关键字也起到了分割代码的作用。

由于 SML 中存在默认的结合规则, 计算 res3 的表达式和计算 res2 的表达式完全相同。而一个 curried function 的定义也可以简写成:

```
fun fold3 f acc xs =
  case xs of
    [] => acc
  | x::xs' => fold2 f (f(acc, x)) xs'
```

它们的类型:

```
val fold1 = fn : ('a * 'b -> 'a) * 'a * 'b list -> 'a
val fold2 = fn : ('a * 'b -> 'a) -> 'a -> 'b list -> 'a
val fold3 = fn : ('a * 'b -> 'a) -> 'a -> 'b list -> 'a
```

柯里化(Currying)使得我们可以更加灵活的使用函数。

```
fun fold f acc xs =
  case xs of
    [] => acc
  | x::xs' => fold2 f (f(acc, x)) xs'

val is_even = fn (x, y) => x + (if y mod 2 = 0 then 1 else 0)
val count_even = fold is_even 0
val res1 = count_even [1, 2, 3, 4]
val res2 = count_even [2, 3, 4, 10, 11]
```

8. 关于 Value Restriction

参考

 知识共享许可协议

原文链接

本作品由<http://www.cnblogs.com/ftae>采用[知识共享署名-非商业性使用 4.0 国际许可协议](#)进行许可。

分类: 笔记



ftae
关注 - 2
粉丝 - 2

+加关注

0
推荐


0
反对

« 上一篇: [Codeforces 893F - Subtree Minimum Query](#)

» 下一篇: [Codeforces #452 Div2 F](#)

posted @ 2017-12-14 19:17 ftae 阅读(987) 评论(0) 编辑 收藏

[刷新评论](#) [刷新页面](#) [返回顶部](#)

 注册用户登录后才能发表评论, 请 [登录](#) 或 [注册](#), [访问](#) 网站首页。

【推荐】了不起的开发者, 势不可挡的华为, 园子里的品牌专区

【推荐】有道智云周年庆, API服务大放送, 注册即送100元体验金!

【推荐】超50万行VC++源码: 大型组态工控、电力仿真CAD与GIS源码库

【推荐】开放下载! 《长安十二时辰》爆款背后的优酷技术秘籍首次公开



最新 IT 新闻:

· 育碧就《看门狗2》无法领取向玩家致歉 网友: 重新发一份全当无事发生过
· 2020高考成绩开始放榜! 各地分数线陆续公布: 高考查分攻略在此

- 曝蚂蚁集团员工持股约40%：老员工基本都实现财务自由
 - 升级后别后悔！苹果关闭iOS 13.5.1验证 不允许用户降级
 - Jio的崛起之路：印度首富成为互联网“全民公敌”
- » 更多新闻...