

函数式编程原理实验报告

姓名：江易星

班级：ACM1601

学号：U201614756

指导教师：陆枫、顾琳

2019年05月01日

1.课程内容梳理和总结 3

1.1 lecture1 3

1.2 lecture2 3

1.3 lecture3 3

1.4 lecture4 3

1.5 lecture5 4

1.6 lecture6 4

1.7 lecture7 4

1.8 lecture8 4

1.9 lecture9 5

1.10 lecture10 5

1.11 函数式编程的起源和发展过程 5

1.12 函数式编程语言的家族成员及其简介 6

2.函数式编程的特点和好处 8

2.1函数式编程的特点 8

2.2函数式编程的好处 9

3.课程学习体会和感想 9

4.授课建议（教学方式、教学内容、授课方法） 9

# 1.课程内容梳理和总结

## 1.1 lecture1

（1）介绍了本课程所涉及的基本内容：学时要求，授课方式、参考资料、涉及内容以及课程目标。

（2）介绍了函数式语言的起源及其发展，介绍了几种函数式编程语言。

（3）介绍了函数式语言的基本特点：引用透明性、高阶函数、惰性求值与并行、递归调用及其优化和模式匹配。重点介绍了一般递归调用的过程和尾递归调用的区别，并利用求斐波拉契数列为例，讲解了将一般递归优化为尾递归的方法。

## 1.2 lecture2

主要对ML语言基本语法做了讲解与说明：

（1）ML语言的标准数据类型：unit、int、real、bool、string、list、tuple、record、function。

（2）介绍几个ML的标准函数：标准布尔函数、标准算术运算函数和标准字符串函数。

（3）ML中值的类型、表达式求值和函数求值。

（4）声明，包括函数的声明和值的声明，声明、类型和值的关系以及声明的作用域。

（5）模式的定义、类型的等式判断运算符“=”，模式匹配及其举例。

## 1.3 lecture3

（1）ML代码中的注释格式、代码说明、代码说明的举例以及作用。

（2）程序的正确性证明及其方法：简单归纳法、完全归纳法、结构归纳法、良基归纳法。以及每种归纳法的适用范围及其证明思路。

（3）程序的近似运行时间分析，递推分析及其举例。

## 1.4 lecture4

以排序算法为例，举例ML语言程序的编写、正确性验证以及性能分析：

（1）order类型的介绍以及整数比较函数compare的实现。

（2）判断整数表是否排好序的函数sorted的实现。

（3）插入排序：思想，操作步骤。

（4）将一个整数插入到一个有序表的函数ins的实现以及正确性证明。

（5）插入排序函数isort的两种实现。

（6）归并排序的基本思想及其操作步骤。

（7）表的分割函数split的实现以及正确性证明。

（8）表的归并函数merge的实现。

（9）利用split和merge函数实现归并排序mergesort，及其正确性证明。

（10）ML编程原则、helper函数的作用和功能说明的作用。

## 1.5 lecture5

排序程序的性能分析、数据类型tree以及在tree上实现排序算法：

（1）插入排序程序性能分析。

（2）归并排序程序性能分析。

（3）数据类型tree的定义以及基本术语、模式表示以及模式匹配和书的结构归纳法推导过程。

（4）与树有关的多个函数的实现：树的大小size、树的深度depth，树的中序遍历trav、判断是否为有序树：sorted

（5）list上的函数移植到tree上：ins函数的移植、split函数的移植、merge的移植，树的归并排序Msort

（6）程序的并行执行以及性能评价指标span。

## 1.6 lecture6

（1）类型分析、类型的引用透明性、类型分析的时间

（2）类型规则：数学运算、表达式比较、分支语句、元组、表、函数、应用、声明、let表达式和模式。

（3）规则的应用举例：模式、函数和递归函数。

（4）多态类型、多态类型的应用举例：split、sorting，多态类型的推导。

（5）Option类型的介绍、相等性和等式类型。

## 1.7 lecture7

函数作为值、高阶函数和多态的作用

（1）数据标准化函数norm的思路和实现：返回值为一个函数。

（2）函数norm的扩展使用。

（3）高阶函数以及高阶函数和多态函数的比较。

（4）处理pair和list：list使用map函数处理以及利用map函数求解子集。

（5）联合函数：设计和应用，foldr和foldl。

## 1.8 lecture8

（1）高阶函数的应用：点集数据标准化、求解点集中心以及其他应用。

（2）以排序算法为例，设计通用排序函数解决排序问题：Isort、Msort

（3）任意类型的数据进行比较，比较函数的实现。

（4）二元组的通用比较函数lex，list数据的通用比较函数listlex

（5）函数less、lesseq、sorted、insertion的实现。

（6）柯里函数以及函数的柯里化技巧。

（7）排序函数isortl和isortr，int实例化和代数规则。

（8）高阶函数、多态以及柯里化的好处。

## 1.9 lecture9

柯里化和高阶函数的部分求解、子集问题的延伸、用柯里化思路设计程序代码和说明以及大规模程序设计思路和方法。

（1）排序函数isortl于isortr、int实例化以及代数规则。

（2）高阶函数、多态以及柯里化的好处。

（3）找零问题的解决思想及实现：穷举法、递归法。

（4）优化找零问题的解决，提供如何找零的方案。

（5）进一步优化，利用多态实现不确定的结果。

## 1.10 lecture10

（1）运行时异常的产生、分类以及异常的引入：求值、等价和引用透明性。

（2）异常的声明、作用域和抛出。

（3）以求最大公因数gcd为例，说明异常的相关问题。

（4）异常的处理，以及异常处理的作用域。

（5）分析找零问题。

## 1.11 函数式编程的起源和发展过程

1930年代，普林斯敦(Princeton)大学的四个人Alonzo Church, Alan Turing, John von Neumann、Kurt Godel都对formal system做研究，他们对于实体世界兴趣不大，他们探讨

的问题都是抽象的数学证明题。

他们虽然各自做研究，但他们的问题确有共同点:想要回答关于计算的问题。如果我们具有一部机器，它的运算威力无限，那么我们能用它来解决哪些问题?这些问题能否自动解决?哪些问题不能解决?为什么不能解决?如果两部机器具有不同的设计，可否具有相同的运

算威力?……

Alonzo Church开发出一套formal system，名为lambda calculus。这个系统本质上是一个编程语言，为一部“想象中的机器”所设计的语言。lambda calculus的函数可以接受函数

当作输入和输出。这样的函数用希腊字母λ当作识别，所以这个语言才名为lambda calculuse

利用lambda calculus, Alonzo能够回答上述的许多问题，提出最终的答案。

Alan Turing也在做类似的研究，开发出不同的系统，就是大名鼎鼎的Turing machine，他得到的结论和Alonzo Church类似。后来证实Turing machine和lambda calculus的威力一样强大。

在信息科学的研究领域，资金充裕的美国军方一直都是很重要的推手之一。二次世界大战时，IBM为了帮美国军方解决弹道计算的问题，建立了知名的Mark I计算机。1949年EDVAC计算机诞生，这是第一部采用von Neumann架构的计算机，也是Turning Machine的真实版本。Turing machine领先lambda calculus做出实体机器。

1958年，对lambda calculus相当感兴趣的MIT教授John McCarthy设计出Lisp语言，Lisp实践了lambda calculus，让lambda calculus可以在von Neumann计算机上执行!大家开始注意到Lisp的威力。1973年，MIT的人工智能实验室开发出所谓的Lisp machine硬件，等于是将lambda calculus的机器实践出来了!

函数式编程长期以来没有出现在主流的商业软件世界。欠缺主流语言的支持，函数式编程只能偏安一隅，躲在学术界。而学术界欠缺书籍文章浅显且正确的介绍，这就是为何函数式编程会依然躲在象牙塔中的原因。

一直以来，真正让FP无法被接受的原因可能是“执行效率”。传统上，函数式编程语言的效率确实比命令式编程语言来得差，这在商业系统上是不能忍受的。命令

式语言让我们用贴近冯纽曼架构机器的方式写程序，比较低阶，所以效率会比较高。而函数式编程语言却是使用较高阶的数学抽象，所以效率比较低。

但是这个原因却有了变化。过去这十多年， 我们有了新的衡量标准:“简单”“快速开发”比其它因素都更重要，因为现在软件的复杂度已经到了我们无法忍受的地步了，而IT产业的竞争也比以往激烈许多。想要“简单”、“快速开发”，就要用比较高阶的抽象，因此函数式编程比命令式编程更适合现在的开发环境。这些年来硬件的进步，让我们对于函数式编程的效率不再是大问题;甚至由于编译技术的进步，函数式编程语言的执行速度，现在也已经不再令人难以忍受。另外，多CPU、多核心、超线程(HT)的硬件架构普及，以及分布式运算的流行，几乎就是专为滋养FP繁殖而打造的环境。

## 1.12 函数式编程语言的家族成员及其简介

（1）ML

ML是一个通用的函数式编程语言，它是由爱丁堡大学的Robin Milner及他人在二十世纪七十年代晚期开发的。它的语法是从ISWIM得到的灵感。作为元语言的ML是为了帮助在LCF定理证明机中寻找证明策略而构想出来的。（之前的元语言是pplambda，它联合了一阶逻辑演算和有类型的多态的λ演算）。它使用了Hindley-Milner类型推论算法来推测大多数值的类型，而不需要四处使用注解。

ML一般被归为非纯函数式编程语言，因为它允许副作用和指令式编程。这一点和纯函数式编程语言——例如Haskell——很不一样。

ML特性包括：传值呼叫（Call by value）的求值策略，一级函数，带有垃圾收集的自动内存管理，参数多态，静态数据类型，类型推论，代数数据类型，模式匹配和异常处理。

不像Haskell，ML使用及早求值，也就是说所有的子表达式总是被求值。导致的一个结果是你不能使用无穷表。然而，惰性求值产生的无穷表可以通过使用匿名函数来模拟。

今天在ML家族中有好几种语言：两种主要的方言是Standard ML和Caml，其他的包括F#－针对Microsoft .NET平台的开放研究项目。ML中的思想影响了众多的语言，例如Haskell，Cyclone和Nemerle。

ML的实力大多被用于语言设计和操作（编译器、分析器、定理证明机），但是它作为通用语言也被用于生化，金融系统，和宗谱数据库，一个P2P的客户／服务器程序等等。

（2）LISP

Lisp（历史上拼写为LISP）是具有悠久历史的计算机编程语言家族，有独特和完全括号的前缀符号表示法。起源于公元1958年，是现今第二悠久而仍广泛使用的高端编程语言。只有FORTRAN编程语言比它更早一年。Lisp编程语族已经演变出许多种方言。现代最著名的通用编程语种是Clojure、Common Lisp和Scheme。

Lisp最初创建时受到阿隆佐·邱奇的lambda演算的影响，用来作为计算机程序实用的数学表达。因为是早期的高端编程语言之一，它很快成为人工智能研究中最受欢迎的编程语言。在计算机科学领域，Lisp开创了许多先驱概念，包括：树结构、自动存储器管理、动态类型、条件表达式、高端函数、递归、自主（self-hosting）编译器、读取﹣求值﹣输出循环（英语：Read-Eval-Print Loop，REPL）。

"LISP"名称源自“列表处理器”（英语：LISt Processor）的缩写。列表是Lisp的主要数据结构之一，Lisp编程代码也同样由列表组成。因此，Lisp程序可以把源代码当作数据结构进行操作，而使用其中的宏系统，开发人员可将自己定义的新语法或领域专用的语言，嵌入在Lisp编程中。

代码和数据的可互换性为Lisp提供了立即可识别的语法。所有的Lisp程序代码都写为S-表达式或以括号表示的列表。函数调用或语义形式也同样写成列表，首先是函数或操作符的名称，然后接着是一或多个参数：例如，取三个参数的函数f即为（f arg1 arg2 arg3）。

Lisp语言的主要现代版本包括Common Lisp, Scheme，Racket以及Clojure。1980年代盖伊·史提尔二世编写了Common Lisp试图进行标准化，这个标准被大多数解释器和编译器所接受。还有一种是编辑器Emacs所派生出来的Emacs Lisp（而Emacs正是用Lisp作为扩展语言进行功能扩展）非常流行，并创建了自己的标准。

（3）Haskell

Haskell是一种标准化的，通用的纯函数编程语言，有非限定性语义和强静态类型。它的命名源自美国逻辑学家哈斯凯尔·加里，他在数理逻辑方面上的工作使得函数式编程语言有了广泛的基础。在Haskell中，“函数是第一类对象”。作为一门函数编程语言，主要控制结构是函数。Haskell语言是1990年在编程语言Miranda的基础上标准化的，并且以λ演算为基础发展而来。这也是为什么Haskell语言以希腊字母“λ”（Lambda）作为自己的标志。Haskell具有“证明即程序、命题为类型”的特征。

Haskell是现有的一门开放的、已发布标准的，且有多种实现的语言。支持惰性求值、模式匹配、列表解析、类型类和类型多态。它是一门纯函数编程语言，这意味着大体上，Haskell中的函数没有副作用。Haskell用特定的类型来表达副作用，该类型与函数类型相互独立。纯函数可以操作并返回可执行的副作用的类型，但不能够执行它们，只有用于表达副作用的类型才能执行这些副作用，Haskell以此表达其它语言中的非纯函数。

Haskell拥有一个基于Hindley-Milner类型推论的静态、强类型系统。Haskell在此领域的主要创新就是加入了类型类（type class），原本设想作为重载的主要方式，在之后发现了更多用途。

Haskell的主要实现GHC是个解释器，也是个原生代码编译器。它可以在大多数平台运行，GHC在并发和并行上具有高性能的实现能力，也有丰富的类型系统，如广义代数数据类型和类型族（Type Families）。

单子是一个抽象类型，可以表达不同种类的计算，包括异常处理、非确定性、语法分析以及软件事务内存，其中一个应用是用于表达副作用的类型。单子定义为普通的数据类型，同时Haskell也为其提供了几种语法糖。

Haskell有一个活跃的社区，在线上包仓库Hackage上有丰富的第三方开源库或工具。

（4）Scala

洛桑联邦理工学院的Martin Odersky于2001年基于Funnel的工作开始设计Scala。Funnel是把函数式编程思想和佩特里网相结合的一种编程语言。Odersky之前工作于Generic Java和javac。Java平台的Scala于2003年底/2004年初发布。.NET平台的Scala发布于2004年6月。该语言第二个版本，v2.0，发布于2006年3月。

Scala 2.8的特性包括重写的Scala容器库、命名参数和默认参数、包对象，以及Continuation.[14]

# 2.函数式编程的特点和好处

## 2.1函数式编程的特点

（1）并行。

在函数式编程中，程序员无需对程序修改，程序就可以并发运行。程序运行期间，不会产生死锁现象。原因是通过函数式编程所得到的程序，在程序中不会出现某一数据被同时修改两次及以上的情况，同样的，两个不同的线程就更不用说了。由于函数式编程有这样的优点，导致了程序员完全不用花费精力去考虑增加某个线程带来的并发问题。

（2）不修改状态、无副作用。

在函数式程序语言中，所有的功能的经果就是一个返回值，不存在其他的行为，包括对外部变量的修改。在其他类型的语言中，变量是用来保存状态的。由于函数式编程不修改变量，导致了这些状态不能存在于变量中。那么，函数式编程语言保存状态的方法是使用参数来保存，递归方法是最好的例子。由于采用了递归方法，函数式编程语言在运行速度上相对

于其他语言较慢，所以，速度不够快是函数式编程语言长期不能广泛使用的主要原因。

（3）惰性求值

即表达式不在它被绑定到变量之后就立即求值，而是在该值被取用的时候求值。惰性求值的一个好处是能够建立可计算的无限列表而没有妨碍计算的无限循环或大小问题。例如，可以建立生成无限斐波那契数列表的函数（经常叫做“流”）。第n个斐波那契数的计算仅是从这个无限列表上提取出这个元素，它只要求计算这个列表的前n个成员。

（4）柯里化

链接库往往将函数定义得比较一般化，具有通用性。这样的函数，需要传入比较多的参数。利用柯里化的方式，可以定义出“特殊化”的函数。所谓柯里化，是把接受多个参数的函数变换成接受一个单一参数（最初函数的第一个参数）的函数，并且返回接受余下的参数而且返回结果的新函数的方法。柯里化与部分求值是相关的，但不完全相同。

（5）模式匹配

模式匹配类比于命令式语言中函数的重载，即执行同一功能的函数可以根据输入参数类型和个数的不同做出不同的反应。从而实现分支功能。模式匹配可以让系统自动帮我们进行分支与变量的指定，有了模式匹配，函数式编程可以降低依赖switch/case而且写出来的程序代码也不会像switch/case那样一大块。

（6）递归

函数式编程中没有其他语言中while、for等循环，想要实现类似的功能只能通过递归实现。另外为了防止多重递归调用导致的爆栈等问题，函数式语言中往往会对递归进行优化，变化为等价的尾递归，某些编译器在编译器会对尾递归进行优化，使得每一次调用都使用相同的栈空间。

（7）高阶函数

在函数式编程中，函数也可以视为一个值，所以可以将一个函数作为另外一个函数的参数。一个函数就接收另一个函数作为参数，那么这么函数就是高阶函数。

## 2.2函数式编程的好处

（1）更方便测试

编写可测试的代码是当下软件开发行业的基本要求，代码在交付前要编写测试用例对每一个功能做单元测试。副作用增加了代码测试难度，原因在于当你尝试测试依赖于外部条件的函数，测试用例也必须运行在相同的条件下。纯函数不依赖也不改变外界的状态，只要给定输入参数，返回的结果必定相同。因为没有副作用，所以不必提供特定的单元测试环境、编写单元测试将变得更加简单、轻松、

（2）易于并行编程

编写多线程代码尤其注意资源的互斥访问及死锁的问题。当纯函数在多线程环境执行的时候，不必担心互斥与死锁。纯函数不会去读取或修改外部对象的值，因此不必编写互斥或者读写锁之类的代码来保证资源被有序访问。

（3）开发快速

由于高阶函数的特性，多个函数可以组合成一个更强大的函数。柯里化技术主张用现有函数来构建新函数的理念。这极大地提高了代码的复用程度。以Lisp和C的比较为例，实现相同的功能，C代码的长度是Lisp的7倍到10倍。

# 3.课程学习体会和感想

因为之前已经接触过有关函数式的知识，所以整个课程，感觉非常的简单和熟悉，几乎没有遇到什么困难的地方。老师也非常的和蔼可亲。整个课程的体验很好，很舒服。但是这门课和之前接触到的还是有一定的区别，以前接触的时候，基本上都是在使用高阶函数和一些新型的数据结构来实现一些比较复杂的功能，而且每次都会有复杂度的限制，所以有时候即使相出了解题思路也不一定满足复杂度的要求。这门课的实验内容基本上都是使用ML语言的基本语法和ML语言自带的数据类型类完成一些简单的操作。只在最后完成了“树”这一种数据结构的定义和操作，也没有复杂度的限制。感觉整个实验只在第二次的reverse‘的时候需要考虑一下复杂度，其他时候直接想到思路就可以解决问题了。

我对于函数式编程思想的理解大概是这样的：给定输入x，x就是对这些输入值的一个命名，然后我们通过将一个函数作用到这个x上，这个函数就像数学中的函数一样，x为自变量，我们得到一个中间结果，并将这个结果命名为y1，然后将y1输入到下一个函数，得到下一个中间结果并将这个结果命名为y2……直到最后得到我们需要的结果。因为x、y1、y2都只是这些结果的一个命名，所以他们每一个的内容都不能被修改。我们编程的时候是要做的事情就是实现从x到输出y中间所有的函数。

4.授课建议（教学方式、教学内容、授课方法）

（1）建议老师多讲一些高阶函数的内容，感觉高阶函数的内容比起课程中涉及的内容更加有意思，也更加难理解一些，况且高阶函数是函数式编程的一大特点

（2）实验加入一些高阶函数的内容，并且加一些复杂度的限制，可以使得实验更加有挑战性也更加有趣。

（3）觉得实验内容有点少，建议老师适当增加实验内容。

（4）感觉实验ppt中有些要求表述不是很清楚，在做实验的时候感觉有些迷糊，希望老师可以稍微修改一下实验ppt。