

函数式编程原理

课程报告

**姓 名： 胡沁心 名**

**学 号： U202015360 级**

**班 级： cs2003 级**

**指导教师： 顾琳 名**

**计算机科学与技术学院**

**2021年12月22日**

一、函数式语言家族成员调研

1. LISP：
2. 创始人：McCarthy
3. 特征

LISP语言是一种早期开发的、具有重大意义的自由软件项目。它适用于符号处理、自动推理、硬件描述和超大规模集成电路设计等。特点是，使用表结构来表达非数值计算问题，实现技术简单。LISP语言已成为最有影响，使用十分广泛的人工智能语言。

1. 发展历程

1960年4月，McCarthy以《递回函数的符号表达式以及由机器运算的方式，第一部》为题，于ACM通讯上发表LISP设置。McCarthy的学生Steve Russell根据该论文，以IBM 704于麻省理工学院的计算机运算文中心成功执行了第一版的LISP。

1962年，McCarthy及人工智能小组按LISP 1的编译基础上改良出LISP 1.5版本。

1969年9月，史丹福大学人工智能实验室的Lynn Quam与Whitfield Diffie推出的Stanford LISP 1.6广泛地应用于使用TOPS-10系统的PDP-10计算机系中。Stanford LISP 1.6版本自麻省理工智能项目更新LISP 1.5成MACLISP及BBN科技公司推出的InterLisp成功后，渐被弃置。

自1960代末年至1980年初年，各种更新LISP版本涌现，有源自加大伯克来分校的Franz Lisp、在AutoCAD运行的AutoLISP前身XLISP、犹他大学开展的Standard Lisp及Portable Standard Lisp、专属于Lisp机器上运行的ZetaLisp、源自法国国家信息与自动化研究所的LeLisp、以及MIT人工智能实验室的Gerald Sussman与Guy Steele所开发的Scheme等。

1984年，改良自MacLisp、集各版本大成、跨平台、且被目为事实标准的Common Lisp诞生。至1994年，美国国家标准学会(ANSI)对Common Lisp语言进行了标准化。

自稳定运行的Common Lisp出现起，再有各机构按各自所需而开展后续Lisp，包括1990年来自欧洲用户的EuLisp及自由开源的IsLisp，ACL2等。

1. ML
2. 创始人：Robin Milner
3. 特征

ML可以算一种具备命令式语言特点的函数型语言，或者说面向函数的命令型语言。和Lisp一样，ML具有非常灵活的函数功能。例如一个表达式的值可能就是一个函数，这个函数可以被作为参数传递给另一个函数，或者函数的返回值就是一个函数。同时和Algol类的语言比较接近的是，ML的语法象命令型的，而且用起来象用Algol家族的很多比较新的后代们一样方便。而且ML有并行扩展，可以用来写并行系统；甚至还有面向对象扩展。ML提供了对函数实际参数的模式匹配、垃圾回收、指令式编程、传值调用和柯里化。它被大量的用于编程语言研究之中，并且是全面规定了的和使用形式语义验证了的少数语言之一。它的类型和模式匹配使得它非常适合并且经常用于在其他形式语言上进行操作，比如在编译器构造、自动化定理证明和形式验证中。

1. 发展历程

ML是Robin Milner主管LCF项目时设计的。LCF项目是受Dana Scott给出的一组逻辑原则启发而设立的，致力于开发一种“可计算函数逻辑”(Logic of Computable Functions)。Robin Milner的目标是构造一个方便实用的系统，来自动的或者半自动的证明函数程序中一些有趣的性质。他的LCF项目于1970年在Standford开始，并于1980年代在Edinburge继续进行。期间取得了很多重要进展，并且激发了相关领域的一系列研究工作。

目前ML有两个发展分支：Standard ML和Caml。

1. Haskell
2. 创始人：Haskell Brooks Curry
3. 特征

Haskell是一种标准化的，通用纯函数式编程语言，有非限定性语义和强静态类型。具有“证明即程序、结论公式即程序类型”的特征。

Haskell支持惰性求值、模式匹配、列表解析、类型类和类型多态。它是一门纯函数式编程语言，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，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。

1. 发展历程

1990年定义了Haskell的第一个版本（“Haskell 1.0”）。委员会形成了一系列的语言定义（1.0，1.1，1.2，1.3，1.4）。

1997年底，该系列形成了Haskell 98，旨在定义一个稳定、最小化、可移植的语言版本以及相应的标准库，以用于教学和作为将来扩展的基础。委员会明确欢迎创建各种增加或集成实验性特性的Haskell 98的扩展和变种。

1999年2月，Haskell 98语言标准公布，名为《The Haskell 98 Report》。2003年1月，《Haskell 98 Language and Libraries: The Revised Report》公布。接着，Glasgow Haskell Compiler (GHC)实现了当时的事实标准，Haskell快速发展。

2006年早期，开始了定义Haskell 98标准后续的进程，非正式命名为Haskell Prime。这是个修订语言定义的不断增补的过程，每年产生一个新的修订版。第一个修订版于2009年11月完成、2010年7月发布，称作Haskell 2010。

Haskell 2010加入了外部函数接口允许绑定到其它编程语言，修正了一些语法问题并废除了称为“n加k模式”。引入了语言级编译选项语法扩展，使得在Haskell源代码中可以明确要求一些扩展功能。

1. Erlang
2. 开发公司：爱立信公司
3. 特征：

1.并行程序设计：在语言中，可以借由spawn/\*函数，将特定的函数设置为独立的进程，之后可以做跨进程通信。

2.单次赋值：每个变量只能跟数据绑一次，所以，不像一般程序设计语言的变量可以多次指定为不同的值。单次赋值的好处是状态单纯，使程序容易阅读。

3.及早求值或严格求值 Erlang：基本求值策略为电脑语言中及早求值之特性。而且，可以借由明确使用无参数的λ表达式，将特定函数设置为惰性求值策略。

4.动态数据类型与类型系统：有编译时期的类型检查系统支持。

5.快速失败：在执行时期发生的错误，会由错误位置提交消息，发生错误的进程立刻停止执行。借由进程通讯机制，可以自动传递错误、捕捉错误，使其他进程能够帮助处理错误。

6.代码热更新：由于Erlang是函数语言，可以撰写特定的程序结构，制作即时更换新版函数的机制。

1. 发展历史

该语言在爱立信内部的若干电信设备开发项目中得到应用，并逐步完善；1988年底Erlang成为开源语言(开源对爱立信的直接好处是:Erlang语言本身的维护与发展、相关的教育培训等投入不再由爱立信独自承担;在开源社区的看护下可以使Erlang更好地发展，同时爱立信也可以继续享受该语言的成果)，但并不广为人知;近年来随着基于互联网的电子商务、即时通讯、云计算、大型网络游戏等应用的发展，以及多核处理器的逐渐兴起，Erlang在大规模并发处理能力和高可靠性方面的优势被越来越多的人所重视，其应用范围有逐渐扩大之势。

二、上机实验心得体会

在函数式编程原理这门课程的学习中，我学到了函数式编程语言的发展和使用场景，并具体学习了Standard ML这一门函数式编程语言。这是我第一次接触函数式编程，与之前学的命令式语言完全不同，但要有趣得多，而且在一些特定问题的求解上能够取得非常好的效果。在逐步摸索Standard ML的语法和进行课程实验的过程中，我逐渐熟练了对 list结构、 tree结构的函数的操作和高阶函数的编写与分析，list和 tree结构都是非常贴合递归思想的数据结构。第三次实验的heapify和SwapDown函数是我在这门课中写过的最难写的函数。这两个函数具体的实现功能和思路和数据结构课讲过的小顶堆排序相似但又有不同，堆排序采用for循环，但这次实验需要用函数式的思想。其中SwapDown的功能分为五种情况：1.empty；2.叶子节点；3.只有左子树；4.只有右子树；5.有两个子树。分别进行排序处理。Heapify函数将一棵二叉树不停分割至叶子节点，从叶子节点开始一层层地调用SwapDown返回小顶堆，和其他子树组成更大的二叉树进行下一轮SwapDown，以保证SwapDown的参数是小顶堆。第四次实验让我对高阶函数的编写及其类型的判断更熟悉了。其中的treeFilter函数，我刚开始不知道’a tree option的类型应该如何表示，在参考了GitHub上的代码后才逐渐理解。

三、课程建议和意见

希望老师每节课结束后能布置与课程相关的题目让我们练习一下，比实验的题目更少但是更可以难些，让我们对课程的知识有更深入的思考。尤其是多阶函数的课程，个人认为比较难学，我在写了实验后才能基本理解。