程序设计技术和方法

Structure and Interpretation of Computer Programs

裘宗燕

北京大学数学学院信息科学系

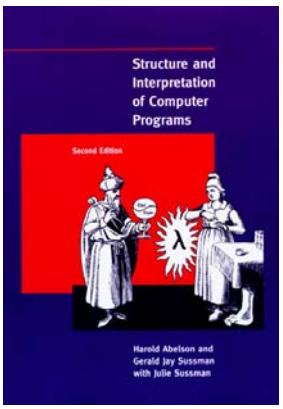
2014.2-2014.6

0. 课程简介

课程教材：

H. Abelson, G. J. Sussman, J. Sussman, Structure and Interpretation of Computer Programs (SICP), MIT

计算机程序的构造和解释，机械工业出版社，2003



课程安排：

- 上课：星期三晚 6 点 40 到 9 点 30，2 教 413
- 上机（具体事项另行安排）：
  - 理科楼 1235
  - 集中上机和辅导时间：星期一晚 7 点到 9 点，第3周开始
  - 每人 50 小时上机时间，可在指定时间去，也可自己安排时间
- 课程辅导
  - 理科一号楼 1480 （我的办公室）
  - 星期三下午 4 点到 5 点半。视情况和需要调整
- 辅导老师：郭炜（负责），刘海洋，徐源盛
  - 作业交给辅导老师（电子邮件）

课程相关材料

- 主页：

  www.math.pku.edu.cn/teachers/qiuzy/progtech/
  - SICP 全文和相关信息，Scheme 手册（R5 有中文翻译）等
  - 发布课程通知，作业和上课幻灯片等
- 讨论用教学网 (course.pku.edu.cn) 本课程讨论板：
  - 欢迎积极参加，提出问题和看法
  - 请只讨论与本课程有关的问题
  - 发贴请尽可能言之有物
  - 给出意义明确的标题，清晰说明问题和/或看法
- 一些同学的教学网帐号下没有邮件地址
  - 请登录并加入邮件地址，以便联系

软件

- 本课程上机用 **Scheme** 语言和 **Racket** 系统
  - **Windows** 界面形式的编辑器和执行环境、调试支持等
  - 系统安装文件可下载
  - 需安装 **SICP** 兼容包，机房将安装 **Racket**
  - 系统网页：**http://racket-lang.org/**
- 也可以用其他编辑器编程后装入系统运行
- 可以用 **MIT Scheme**
  - 系统有联机手册
  - 基本系统是一个交互式解释环境
  - 带一个类似 **emacs** 的编辑环境
  - 网页有简单使用说明

情况和想法

- 信息学院希望开一门用 **SICP** 开的选修课程

  帮助同学进一步加深对程序语言和程序设计中各种问题的认识
- 数学学院原有课程“程序设计技术和方法”

  两方面想法的结合形成了这个供两学院同学选修课。已开过五次

目标：帮助同学从更多角度观察和理解程序和程序设计中的问题：

- 函数式程序设计
- 各种程序组织方式，分解和控制程序复杂性的技术
- 丰富的编程模式
- 对一些基础问题的理解
- 上述各方面与常规（命令式，如**c/c++**等）程序设计的关系/启示等
- **SICP** 这本书提供了许多有用的想法和信息

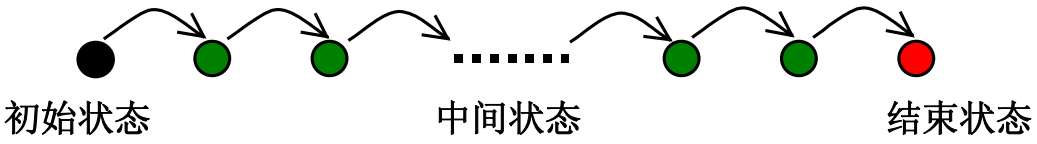
说明式和过程式知识

- 说明式知识（是什么）
  - 有关事实和情况的说明
  - 实例：饭店的菜肴介绍，包括配料成分、色香味说明、照片等
- 过程式知识（怎么做）
  - 有关完成某件工作的一系列步骤（操作）的描述
  - 实例：菜肴的烹制方法和过程，相关操作及执行顺序
- 随着计算机科学技术的发展，过程式知识变得越来越重要
  - 算法：过程式知识的精确化，与适当信息表示形式结合解决问题
  - 二进制编码的普适性，与过程式知识结合，借助于电子计算机作为基本工具，形成了一套具有普适性的全新的问题解决模式
- 软件开发工作就在说明式与过程式知识的结合点上

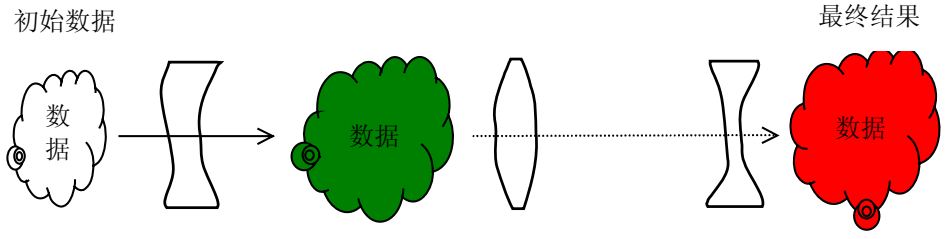
  从说明式知识出发，开发过程式知识

命令式和函数式计算

- 程序设计的主流是命令式 **(imperative)** 程序设计，计算基于一组基本操作，在一个环境里进行。操作效果是改变环境的状态，体现在所创建和修改的状态中：



- 函数式 **(functional)** 程序设计中计算过程被看成是数据的变换，程序的行为就是对数据的一系列变换：



可称为程序（和程序设计）的命令式观点和函数式观点

函数式和命令式程序设计

- （常规）程序以命令方式描述计算
  - 层次较低，接近（容易有效利用）实际硬件（计算机），可能高效
  - 编程时需要关注更多细节，复杂，很容易出错
- 函数式程序设计层次较高
  - 更抽象，屏蔽计算的更多实现细节，程序可能更清晰
  - 与实际硬件距离较远，需要复杂的运行时支持，可能效率较低
- 在编程中命令式思维和函数式思维都有价值

  在不同抽象层次，需灵活使用对程序的命令式思维和函数式思维
- 程序的工作常可分解为一些阶段
  - 每个阶段是对整体数据空间的某种变换（函数式的）
  - 而每个变换又是通过复杂的状态操作实现（命令式的）

Scheme

- 本课程中使用的 **Scheme** 是一种 **Lisp** 方言
- **Lisp** 是函数式语言之祖
  - 许多新语言从中汲取营养
  - 了解 **Scheme** 有利于理解今天和明天的编程语言
- **Scheme** 可以很好表现程序和程序设计中的许多问题
- **Scheme** 能自然地支持许多威力强大的编程技术，许多技术有一般性，可能在其他语言中使用或模拟
- **Scheme** 不是纯函数式语言，为了效率和对计算的控制，提供了命令式特征（状态操作）。学习它有助于理解另一种编程思维，及在两种思维和编程方式间权衡和转换
- 在这里学到的技术可能用到其他地方。课程中将讨论一些技术在常规语言和编程中的应用可能性，相互关系

函数式语言和常规程序设计

- 主流语言已经并将继续从函数式语言中汲取大量营养
  - 基于运行栈的实现技术，递归
  - 动态存储分配，废料回收（垃圾回收）
  - 表处理的问题和技术（**Lisp** 的核心数据结构）
  - 动态操作指派（方法的动态约束，面向对象语言的基础）
  - 虚拟机、字节码、动态编译（**Java, C#** 等）
  - **lambda** 表达式，**… …**
- 学习函数式程序设计（包括 **Lisp** 类语言程序设计），有助于掌握如何在较高抽象层次上思考计算问题和程序问题
- 丰富看程序和程序设计的角度和观点，可能把程序设计中的一些基本问题看得更清楚