程序设计语言原理

THE PRINCIPLES OF PROGRAMMING LANGUAGE

先修课程：C语言、编译原理、离散数学

主讲：吕卫锋
LWF@NLSDE.BUAA.EDU.CN;

诸彤宇
ZHUTONGYU@BUAA.EDU.CN;

程序设计语言原理

一门理论性较强的提高型课程，从更高的层次来理解各种语言机制，指导对计算机语言的学习和程序设计

- 分析并理解各类高级程序设计语言范型与理论模型
- 掌握程序设计语言各主要成分设计中的关键问题、主要步骤、表示法的基本技能
- 学会分析、选择、调合、折中、设计语言的特征

程序设计语言原理

- **教材**:

  程序设计语言原理　麦中凡　吕卫锋　北航出版社

- **参考书**:

- 《程序语言原理（第八版）》，Robert W.Sebesta，机械工业出版社。
- 《程序设计语言：原理与实践（第二版）》，Kenneth C. Louden，电子工业出版社。
- 《程序设计语言：设计与实现（第四版）》，Terrence W. Pratt，Marvin V. Zelkowitz，电子工业出版社。
- 《程序设计语言：概念和结构（第二版）》，Ravi Sethi，机械工业出版社

课程内容

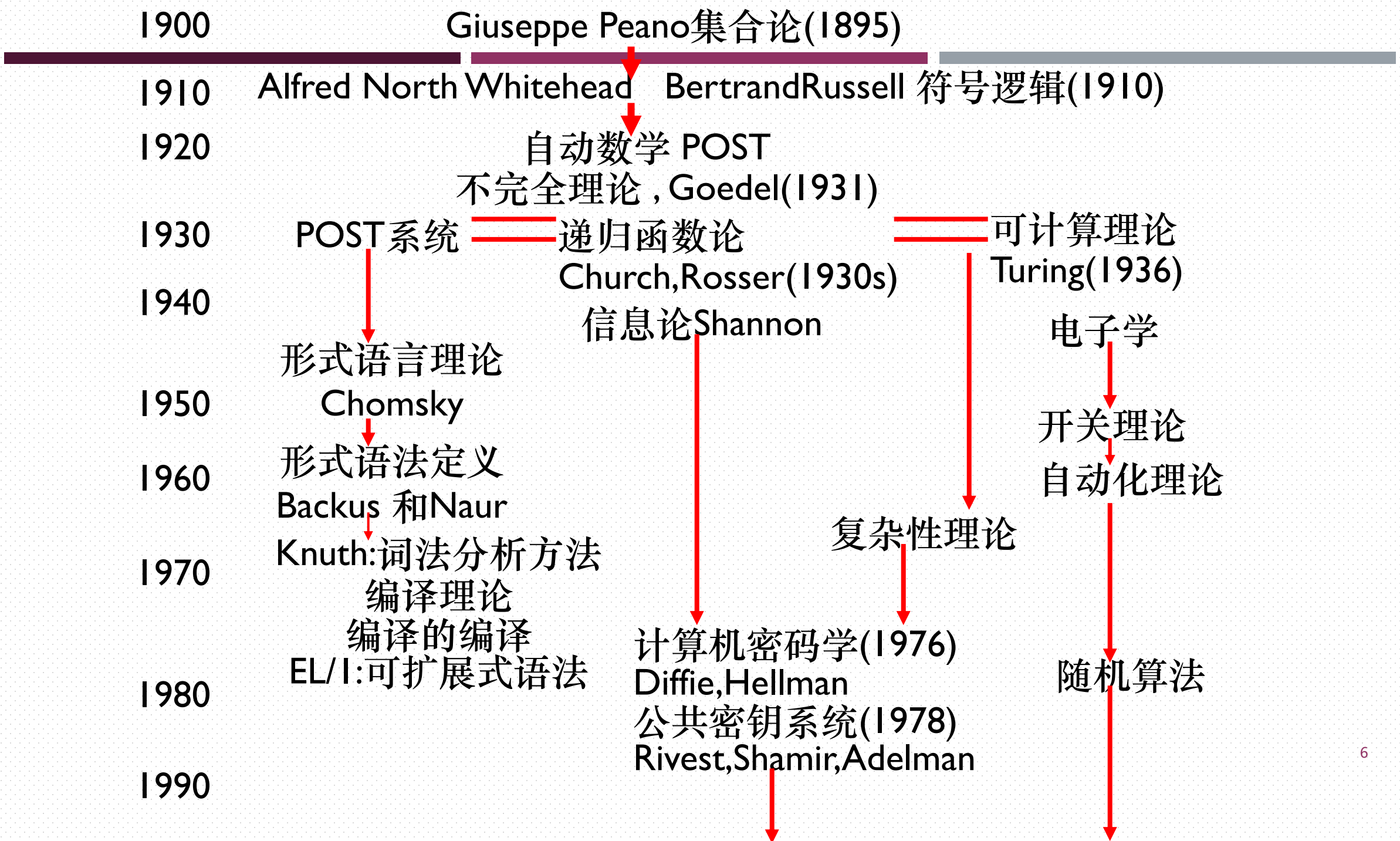
- 第0章　绪论
- 第1章　程序设计语言发展与分类
- 第2章　程序设计语言设计概述
- 第3章　过程式程序设计语言
- 第4章　面向对象程序设计语言
- 第5章　函数式程序设计语言
- 第6章　逻辑式程序设计语言
- 第7章　并发程序设计语言
- 第8章　平台无关程序设计语言
- 第9章　描述性程序设计语言
- 第10章　指称语义的原理与应用

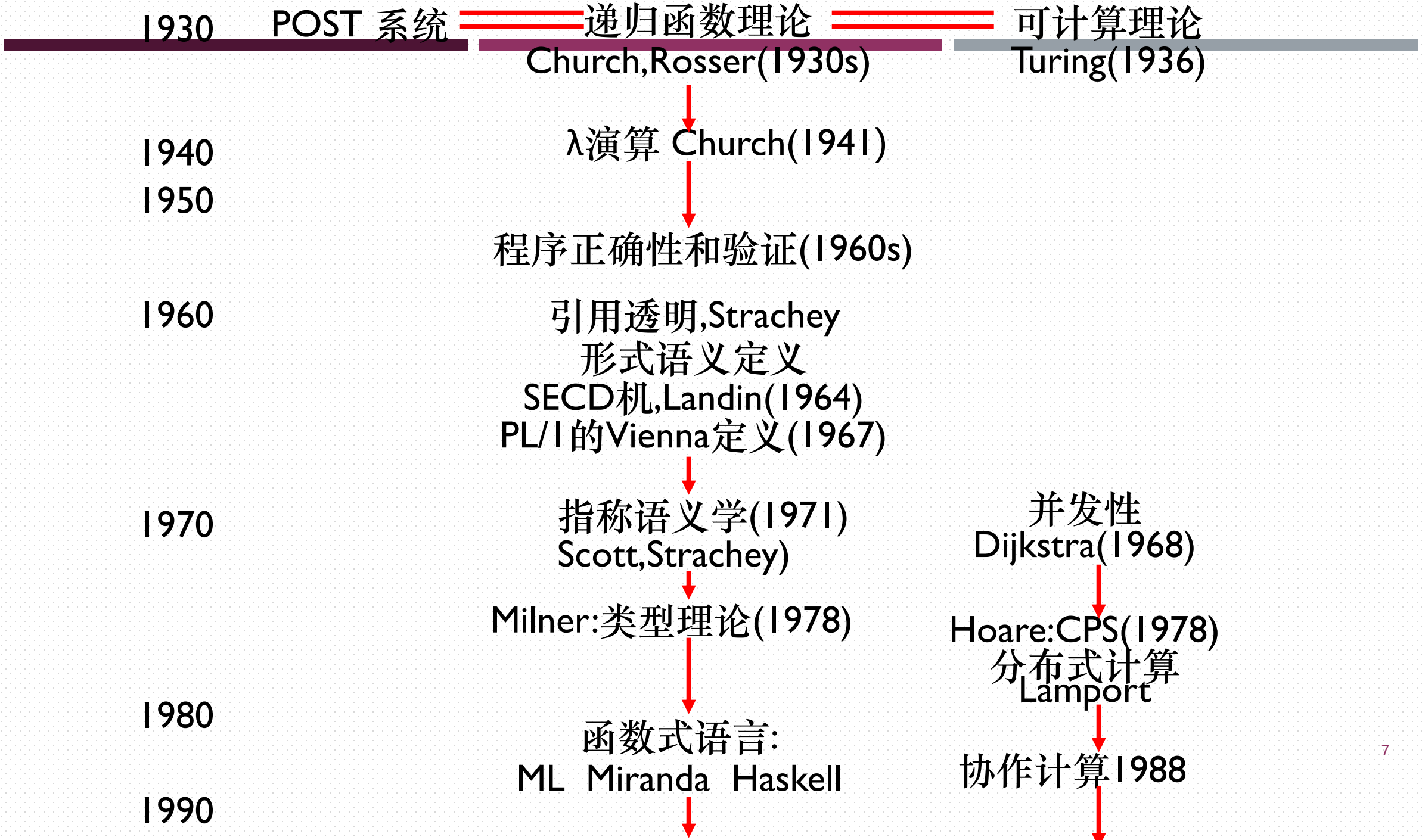
第0章 导 论

- 什么是程序设计语言(PL)
- 为什么研究PL
- 语言规范与处理器
- 本课程内容与要求

0.1 什么是程序设计语言（PL）？

- 人机通信媒体(介)，软件的载体
  - 人工语言　　机器识别，方便人使用
  - 形式语言　　无二义性
  - 必须可执行
- 它是计算机科学与计算机工程的交汇点
  - 计算机科学是在符号学、集合论、离散数学、组合数学基础上发展的
  - 以符号语言表达的软件还要满足正确性、可靠性、安全性、可扩充、可移植、方便性


1900
Giuseppe Peano集合论(1895)
1910
Alfred North Whitehead
BertrandRussell 符号逻辑(1910)
1920
自动数学 POST
不完全理论，Goedel(1931)
1930
POST系统
递归函数论
Church,Rosser(1930s)
可计算理论
Turing(1936)
1940
信息论Shannon
电子学
形式语言理论
Chomsky
1950
开关理论
形式语法定义
1960
自动化理论
Backus 和Naur
Knuth:词法分析方法
复杂性理论
1970
编译理论
编译的编译
计算机密码学(1976)
Diffie,Hellman
公共密钥系统(1978)
Rivest,Shamir,Adelman
随机算法
EL/1:可扩展式语法
1980
1990


1930
POST 系统
递归函数理论
Church,Rosser(1930s)
可计算理论
Turing(1936)
λ演算 Church(1941)
1940
1950
程序正确性和验证(1960s)
1960
引用透明,Strachey
形式语义定义
SECD机,Landin(1964)
PL/I的Vienna定义(1967)
1970
指称语义学(1971)
Scott,Strachey)
并发性
Dijkstra(1968)
Milner:类型理论(1978)
Hoare:CPS(1978)
分布式计算
Lamport
1980
函数式语言:
ML Miranda Haskell
协作计算1988
1990

重要性

程序设计语言的研究和开发处于计算机科学技术发展的中心：

- 计算机理论和方法的研究，许多是由于语言发展的需求
- 许多理论研究成果体现到程序语言的设计中
- 实际应用中最本质的需要常反映到程序语言里，推动语言的演化和发展
- 语言实现的需要是推动计算机体系结构演化的一个重要因素（如RISC）
- 计算机硬件的能力和特征也对程序语言的发展变化有着重要影响（今天和明天，并行性问题）
- 理解程序设计语言，有助于提高对整个计算机科学技术领域的认识

推动语言演化发展的要素：

实际应用的需要，硬件的发展和变化，人们对于程序设计工作的认识发展，实现技术的开发，理论研究的成果

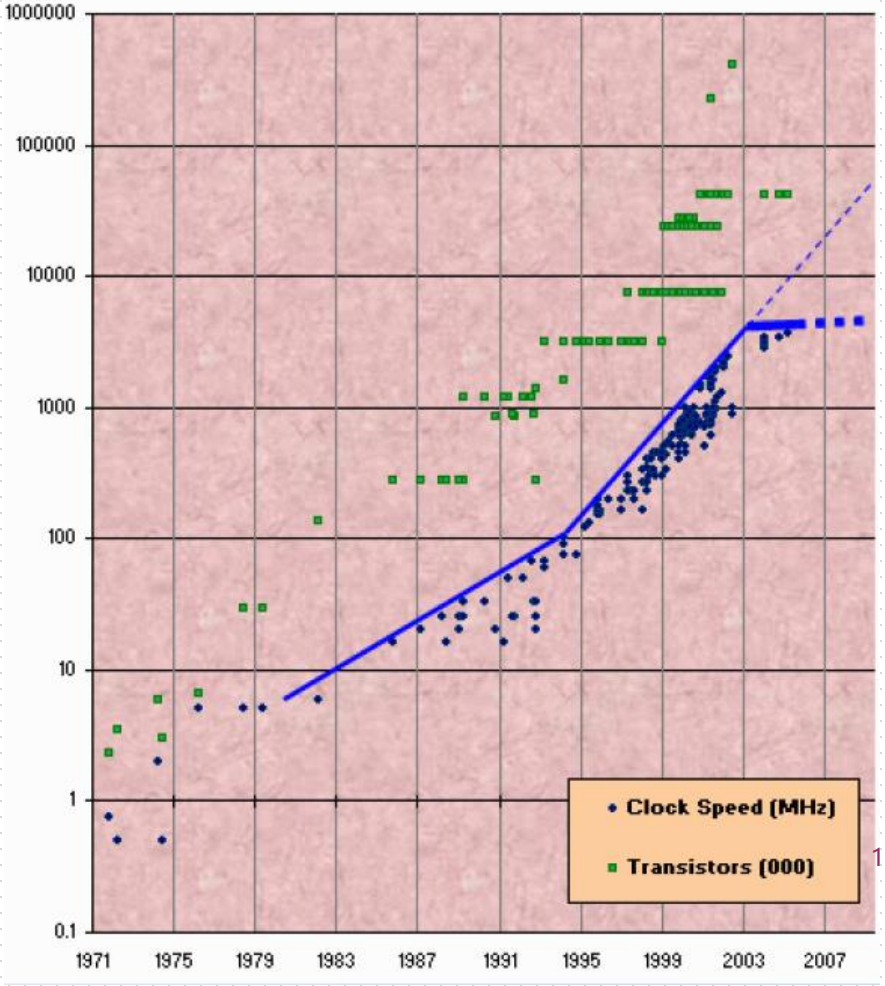
重要性：图灵奖

1966-2019，54届图灵奖，有16届由于与程序语言有关的工作而获奖

- 1966，Alan J.Perlis，早起语言和Algol 60的贡献，图灵奖第一位获奖者
- 1971，John McCarthy，LISP语言，程序语义，程序理论
- 1972，E.W.Dijkstra，Algol编译，结构化程序设计，并发概念和原语，形式化推到，卫式命令等
- 1976，Dana Stewart Scott， Michael Oser Rabin，提出的非确定性机器这一很有价值的概念而获奖
- 1977，John Backus，Fortran语言，FP语言，BNF等
- 1978，Robert Floyd，Algol编译，编译技术，程序优化，归纳断言法和前后断言，程序正确性，编译生成
- 1979，K.E.Iverson,APL语言
- 1980，C.A.R.Hoare，结构化程序设计，case语句，公理语义学，并发程序的理论，CSP等
- 1983，Dennis Ritchie和Thompson，C语言和UNIX
- 1984，Niklaus Wirth，Alogl W, PL360, Pascal, 逐步求精，结构化程序设计，语法图
- 1991，Robin Milner, ML语言，并发理论，CCS
- 1996，Amir Pnueli，时序逻辑和系统验证
- 2001，Ole-Johan Dahl和Kristen Nygaard, Simula语言，OO概念
- 2003，Alan Kay, Smaltalk语言，OO概念、语言和程序设计
- 2005，Peter Naur, Algol 60语言的设计和定义，编译，程序设计的原理和实践
- 2006，Frances Allen，优化编译和并行化
- 2008，Barbara Liskov，数据抽象/OO/容错/分布式计算程序的基础和语言
- 2016，Tim Berners-Lee，制定了互联网的URIs、HTTP、HTML等技术规范

新趋势：并行

- 狭义的摩尔定律已失效，提高主频的趋势已停止
- 并行环境已逐渐成为我们周围最常见计算机的基本结构的一部分
- 如何做并行程序设计的问题变成对每个计算机工作者的挑战
- 程序设计语言也需要反应这方面的需求
- 有关并行语言、程序和程序设计的问题，将在今后很多年里成为程序设计语言研究领域里最重要的问题



新趋势：并行

有关并行系统和并行程序设计的研究已经进行了近40年，但对并行系统和如何设计实现并行系统的认识仍很不成熟：

- 已开发的并行系统（及分布式系统）经常出现意料之外的错误
- 并行系统的开发方法很难使用，开发低效，对开发人员缺乏良好支持
- 描述并发系统的记法形式过于低级和细节，缺乏有效抽象手段
- 并发系统的验证技术不成熟，系统缺乏可靠性的保证

对于上述问题的研究和并发程序开发实践将未来语言的发展影响有重大影响

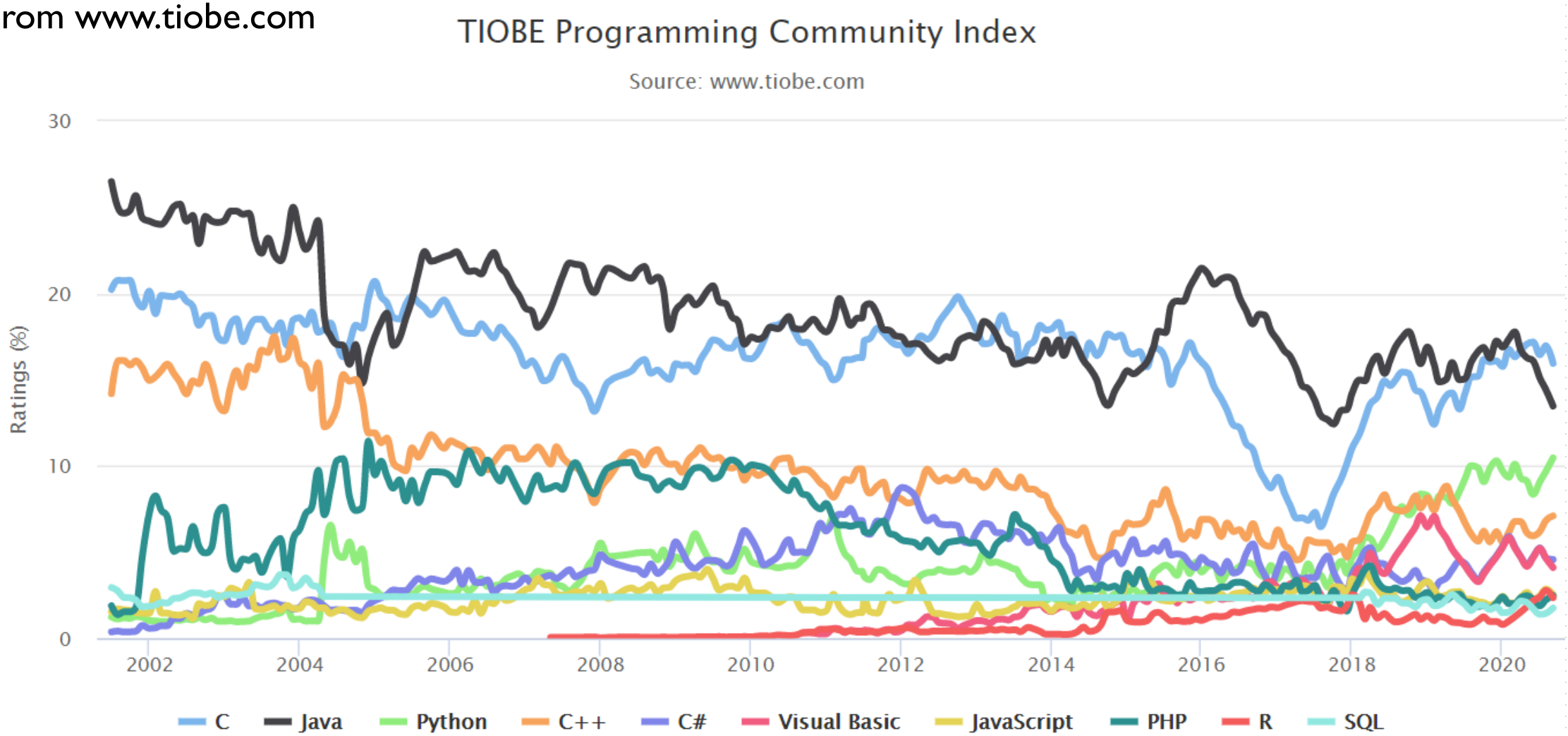
- 许多新语言里加入了并行特征，包括Java、C# 等
- 一些并行理论的研究成果被用于实践，如JCSP
- 人们重新开始重视无状态的程序设计，函数式程序设计（如Erlang 语言受到许多人推崇），提出了一些新想法
- 这方面的理论和实际技术研究将成为很长时间的研究热点

新趋势：脚本语言

- **二十一世纪以来，脚本语言在计算机应用盛行起来，重要实例：**
  - 用于开发**Web** 服务端的**PHP**、**ASP**、**JSP** 等
  - 用于**Web** 客户端网页嵌入应用的**JavaScript** 等
  - 用于更广泛的应用开发的**Perl**、**Python**、**Ruby** 等
  - 其他各种专门用途的脚本语言，如描述图形界面的Tcl/tk
- **与通用程序设计语言相比，通用脚本语言有如下特点：**
  - 丰富的基础数据结构，灵活的使用方式，支持快速的应用开发
  - 基于解释器的执行，或者解释和编译的结合，可以立即看到开发的效果
  - 通常都没有标准化，随着应用的发展变化和很快地扩充
  - 一些语言形成了很好的社团，开发了大量有用的库
- **脚本语言将如何发展？其发展趋势怎样？**

程序设计语言流行程度

From www.tiobe.com



程序设计语言流行程度

- 1、随着机器学习与深度学习的迅速发展，Python首次超越C++，排名第三；
- 2、SQL语言首次挤进前20,跃居第9名；
- 3、Go语言持续上升。

Sep 2019	Sep 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.661%	-0.78%
2	2		C	15.205%	-0.24%
3	3		Python	9.874%	+2.22%
4	4		C++	5.635%	-1.76%
5	6	↑	C#	3.399%	+0.10%
6	5	↓	Visual Basic .NET	3.291%	-2.02%
7	8	↑	JavaScript	2.128%	-0.00%
8	9	↑	SQL	1.944%	-0.12%
9	7	↓	PHP	1.863%	-0.91%
10	10		Objective-C	1.840%	+0.33%
11	34	⇈	Groovy	1.502%	+1.20%
12	14	↑	Assembly language	1.378%	+0.15%
13	11	↓	Delphi/Object Pascal	1.335%	+0.04%
14	16	↑	Go	1.220%	+0.14%
15	12	↓	Ruby	1.211%	-0.08%
16	15	↓	Swift	1.100%	-0.12%
17	20	↑	Visual Basic	1.084%	+0.40%
18	13	⇊	MATLAB	1.062%	-0.21%
19	18	↓	R	1.049%	+0.03%
20	17	↓	Perl	1.049%	-0.02%

程序设计语言流行程度

2020年9月最新排名

C语言重夺榜首！

TIOBE排行榜是根据互联网上有经验的程序员、课程和第三方厂商的数量，并使用搜索引擎（如Google、Bing、Yahoo!）以及Wikipedia、Amazon、YouTube统计出排名数据，只是反映某个编程语言的热门程度，并不能说明一门编程语言好不好，或者一门语言所编写的代码数量多少。

Sep 2020	Sep 2019	Change	Programming Language	Ratings	Change
1	2	↑	C	15.95%	+0.74%
2	1	↓	Java	13.48%	-3.18%
3	3		Python	10.47%	+0.59%
4	4		C++	7.11%	+1.48%
5	5		C#	4.58%	+1.18%
6	6		Visual Basic	4.12%	+0.83%
7	7		JavaScript	2.54%	+0.41%
8	9	↑	PHP	2.49%	+0.62%
9	19	⇈	R	2.37%	+1.33%
10	8	↓	SQL	1.76%	-0.19%
11	14	↑	Go	1.46%	+0.24%
12	16	⇈	Swift	1.38%	+0.28%
13	20	⇈	Perl	1.30%	+0.26%
14	12	↓	Assembly language	1.30%	-0.08%
15	15		Ruby	1.24%	+0.03%
16	18	↑	MATLAB	1.10%	+0.04%
17	11	⇊	Groovy	0.99%	-0.52%
18	33	⇈	Rust	0.92%	+0.55%
19	10	⇊	Objective-C	0.85%	-0.99%
20	24	⇈	Dart	0.77%	+0.13%

长期霸榜语言

Programming Language	2020	2015	2010	2005	2000	1995	1990	1985
Java	1	2	1	2	3	-	-	-
C	2	1	2	1	1	2	1	1
Python	3	7	6	6	20	20	-	-
C++	4	3	3	3	2	1	2	9
C#	5	5	5	7	9	-	-	-
JavaScript	6	8	8	10	6	-	-	-
PHP	7	6	4	5	19	-	-	-
SQL	8	-	-	-	-	-	-	-
Swift	9	16	-	-	-	-	-	-
R	10	12	52	-	-	-	-	-
Lisp	27	24	15	14	8	6	4	2
Fortran	31	25	24	15	15	4	3	5
Ada	33	27	23	17	17	5	9	3
Pascal	241	15	14	22	16	3	10	6

近三年排名前20语言的初创和发行时间

语言	诞生	商用/应用	语言	诞生	商用/应用
JAVA	1995	1996	PL/SQL	1985	1985
C	1972	1973	SAS	1976	1985
PHP	1995	1997	ABAP	1985	1985
C++	1983	1990	D	1999	2007
Visual Basic	1991	1991	Objective-C	1983	1992
Perl	1987	1991	Lua	1993	1993
C#	2000	2002	MATLAB	1970末	1984
Python	1991	2000	Delphi/Object Pascal	1986	1994
JavaScript	1995	1996	Lisp	1958	1962
Ruby	1995	2011	Transact-SQL	1992+	
Delphi	1995	1999	Visual Basic .NET	2001	2002
Pascal	1970	1983	Ada	1980	1983
Lisp/Scheme	1958	1962	Assembly		

Programming Language Hall of Fame

Year	Winner
2019	C
2018	Python
2017	C
2016	Go
2015	Java
2014	JavaScript
2013	Transact-SQL
2012	Objective-C
2011	Objective-C
2010	Python
2009	Go
2008	C
2007	Python
2006	Ruby
2005	Java
2004	PHP
2003	C++

什么是程序设计语言（PL）?续

- 定义：
  - 可以编制软件的
  - 机器可识别
  - 可执行的表示法(或符号)系统

0.2 为什么研究PL

- 人机交互界面永存 过去—现在—未来
- 软硬件技术窗口
- 发展新语言
- 提高软件人员素质
- 通向理论的形式方法
- 通用语言标准化与规范化

0.3 语言规范与处理器

- **PL语言不是软件**
  - 它只是一规范——参考手册（LRM）

    规定符号元素—语法—语义（形式的 非形式的）
  - 按它的规定写出的程序是软件
- **PL翻译器也是软件**
  - 一种语言到另一种语言：翻译器
  - 一种语言到目标码
    - 编译器 先翻译、优化后执行 高效
    - 解释器 即译即执行 低效 灵活

0.4 本课程内容与要求

- **本课程分为三部分**
  - PL的一般概述，形式语法复习　　0-2章
  - 各种PL范型　　3-10章
  - 语义理论　　10-11章
- **要求**
  - 习题作业平时占50%计分
  - 考试占50%计分

0.5 计算学科命名的背景

- 如何认知计算学科，有着不少争论。
  - 1984年7月，美国计算机科学与工程博士单位评审部的领导们，在犹他州召开的会议上对计算认知问题进行了讨论。
  - 这一讨论以及其他类似讨论促使（美国）计算机协会（ACM）与（美国）电气和电子工程师学会计算机分会（IEEE/CS）于1985年春联手组成任务组，
  - 经过近4年的工作，任务组提交了在计算教育史上具有里程碑意义的“计算作为一门学科”（Computing as a Discipline）报告

0.5 计算学科命名的背景

- “计算作为一门学科”报告论证了计算作为一门学科的事实
  - 回答了计算学科长期以来一直争论的一些问题；
  - 并将当时的计算机科学、计算机工程、计算机科学和工程、计算机信息学以及其他类似名称的专业及其研究范畴统称为计算学科。

0.5 计算学科的定义

- “计算”的定义
  - “计算”是从一个符号行ξ得出另一个符号行η的变换；
  - “计算” δ的概念可以用符号简洁地表示如下：

$$\delta: \xi \rightarrow \eta$$

  - 当然，符号ξ和η各自表示了某种信息，因此也可以说，计算是一种信息变换；

0.5 计算学科的定义

- 从“计算”的定义可知，它至少涉及两个方面：
  - 用计算机求解问题的时候，首先需要用适当的数据表示问题，然后再用适当的算法对着这些数据进行变换，进而获得问题的求解结果；
  - 这种所谓的“问题抽象、形式化描述、自动化(计算机化)”的解题思路，实际上就是具有“抽象能力与形式化描述能力” 的“计算机思维”。
  - 因此，就“计算机思维”而言，有两门课是计算机专业的大学生必须学习的：《形式语言与自动机》和《算法设计与分析》

0.5 计算学科的定义

- 计算学科是对描述和变换信息的算法过程进行的系统研究，包括理论、分析、设计、效率、实现和应用等。
- 计算学科包括对计算过程的分析以及计算机的设计和使用。
- 学科的广泛性在下面一段来自美国计算科学鉴定委员会发布的报告摘录中得到强调：
  - 计算学科的研究包括从算法与可计算性的研究到根据可计算硬件和软件的实际实现问题的研究。
- 这样，计算学科包括从总体上对算法和信息处理过程进行的研究，也包括满足给定规格要求的有效而可靠的软硬件设计—它包括所有科目的理论研究、实验方法和工程设计。

0.5　计算学科的根本问题

- 计算学科的根本问题是：
  - “什么能被（有效地）自动进行”。
- 计算学科来源于对算法理论、数理逻辑、计算模型、自动计算机器的研究，并与存储式电子计算机的发明一起，形成于20世纪40年代初期。

0.6 计算机科学与技术体系CC2001

- 一.DS. Discrete Structures

主要内容包括集合论,数理逻辑,近世代数,图论以及组合数学等.

该领域与计算学科各主领域有着紧密的联系,CC2001为了强调它的重要性,特意将它列为计算学科的第一个主领域.该主领域以“抽象”和“理论”两个学科形态出现在计算学科中,它为计算学科各分支领域解决其基本问题提供了强有力的数学工具.

0.6 计算机科学与技术体系CC2001

- 二. PF. Programming Fundamentals

主要内容包括程序设计结构,算法,问题求解和数据结构等.
它考虑的是如何对问题进行抽象.它属于学科抽象形态方面的内容,
并为计算学科各分支领域基本问题的感性认识(抽象)提供方法.

基本问题主要包括:
1.对给定的问题如何进行有效的描述并给出算法?
2.如何正确选择数据结构?
3.如何进行设计,编码,测试和调试程序?

0.6 计算机科学与技术体系CC2001

- 三.AL. Algorithms and Complexity

  主要内容包括算法的复杂度分析,典型的算法策略,分布式算法,并行算法,可计算理论,P类和NP类问题,自动机理论,密码算法以及几何算法等.

  基本问题主要包括:

  1.对于给定的问题类,最好的算法是什么?要求的存储空间和计算时间有多少?空间和时间如何折衷?

  2.访问数据的最好方法是什么?

  3.算法最好和最坏的情况是什么?

  4.算法的平均性能如何?

  5.算法的通用性如何?

0.6 计算机科学与技术体系CC2001

- 四.PL. Programming Languages

主要内容包括程序设计模式,虚拟机,类型系统,执行控制模型,语言翻译系统,程序设计语言的语义学,基于语言的并行构件等.

理论形态的主要内容：包括形式语言和自动机，图灵机(过程式语言的基础)，POST系统(字符串处理语言的基础)，lamda-演算(函数式语言的基础)，形式语义学，谓词逻辑，时态逻辑，近世代数等.

0.6 计算机科学与技术体系CC2001

- 四.PL. Programming Languages

基本问题主要包括:

1.语言(数据类型,操作,控制结构,引进新类型和操作的机制)表示的虚拟机的可能组织结构是什么?

2.语言如何定义机器?机器如何定义语言?

3.什么样的表示法(语义)可以有效地用于描述计算机应该做什么?

0.6 计算机科学与技术体系CC2001

- 五.AR. Architecture and Organization
- 六.OS. Operating Systems
- 七.NC. Net-Centric Computing
- 八.HC. Human-Computer Interaction
- 九.GV. Graphics and Visual Computing
- 十.IS. Intelligent Systems
- 十一.IM. Information Management
- 十二.SE. Software Engineering
- 十三.SP. Social and Professional Issues
- 十四.CN. Computational Science

CS2013的产生背景

- 为顺应计算机科学前沿理论和技术发展以及工业界的需求，每十年，ACM和IEEE-CS会共同发起制定关于“计算机科学”学科的课程大纲
  - 目的是为全球“计算机科学”专业的教学提供最新的课程指导
  - 已分别于1968、1978、1991和2001发布了前期版本
    - 最近的CS2008版本，是临时版
  - 自2001年起，“计算机科学”分为
    - 计算机科学（CS），计算机工程（CE），信息系统（IS），信息工程（IT）以及软件工程（SE）
- 最新完整版于2013发布，命名为CS2013
  - 该版本的制定工作从2010年秋季开始

CS2013的产生背景

- 大帐棚策略（Big Tent）
  - 涵盖包括最新发展在内的计算机科学各领域
  - 起到桥梁作用，关联相关交叉学科
- 控制学时总数
  - 与CS2001相比，学时总数不增加
- 提供真实课程范例
  - 给出包含各知识点的真实范例课程
- 兼容习俗和文化的需要
  - 兼容不同的培养目标、资源和限制条件
  - 兼容不同的学院规模、学院类型以及可利用的资源

CS2013知识点（KNOWLEDGE AREA）与学时安排

Knowledge Area	CS2013 Tier1	CS2013 Tier2	CS2008 Core	CC2001 Core
AL-Algorithms and Complexity	19	9	31	31
AR-Architecture and Organization	0	16	36	36
CN-Computational Science	1	0	0	0
DS-Discrete Structures	37	4	43	43
GV-Graphics and Visual Computing	2	1	3	3
HC-Human-Computer Interaction	4	4	8	8
IAS-Security and Information Assurance	2	6	--	--
IM-Information Management	1	9	11	10
IS-Intelligent Systems	0	10	10	10
NC-Networking and Communication	3	7	15	15
OS-Operating Systems	4	11	18	18
PBD-Platform-based Development	0	0	--	--
PD-Parallel and Distributed Computing	5	10	--	--
PL-Programming Languages	8	20	21	21
SDF-Software Development Fundamentals	42	0	47	38
SE-Software Engineering	6	21	31	31
SF-Systems Fundamentals	18	9	--	--
SP-Social and Professional Issues	11	5	16	16
Total Core Hours	**163**	**142**	**290**	**280**

新增的知识点

内容和学时有较大调整的知识点

重视系统、重视基础

- 分别从软件开发和系统两个层面，强调基础原理与方法策略，学时数高，概括性广
  - SDF. Software Development Fundamentals
    - (42 Core-Tier1 hours，42 total)
    - includes fundamental concepts and skills that could appear in other software-oriented KAs (e.g., programming constructs from Programming Languages, simple algorithm analysis from Algorithms and Complexity, simple development methodologies from Software Engineering)

	Core-Tier1 hours	Includes Electives
SDF/Algorithms and Design	11	N
SDF/Fundamental Programming Concepts	10	N
SDF/Fundamental Data Structures	12	N
SDF/Development Methods	9	N
total	42	

重视系统、重视基础

- SF. Systems Fundamentals
  - (18 core Tier 1, 9 core Tier 2 hours, 27 total)
  - The new Systems Fundamentals KA presents a unified systems perspective and common conceptual foundation for other KAs (notably Architecture and Organization, Network and Communications, Operating Systems, and Parallel and Distributed Algorithms)

	Core-Tier 1 hours	Core-Tier 2 hours
SF/Computational Paradigms	3	
SF/Cross-Layer Communications	3	
SF/State-State Transition-State Machines	6	
SF/System Support for Parallelism	3	
SF/Performance	3	
SF/Resource Allocation and Scheduling		2
SF/Proximity		3
SF/Virtualization and Isolation		2
SF/Reliability through Redundancy		2
total	18	9

强调交叉

- 特别强调，课程与知识点并非一一对应
  - 可将各知识点有机融入不同课程中
  - 一门课程中可以同时涵盖不同知识点的不同层次的知识内容，即可以同时涵盖不同知识点的tier-1 core、 tier-2 core以及elective中的知识内容
- 一个知识点中本身也会涵盖其它知识领域的内容，比如
  - **AL. Algorithms and Complexity**
    - **19 Core-Tier1 hours, 9 Core-Tier2 hours**
  - **SP. Social and Professional Practice**
    - **11 Core-Tier1 hours, 5 Core-Tier2 hours**