编程语言设计原理
Design Principles of Programming Languages

Haiyan Zhao, Zhenjiang Hu

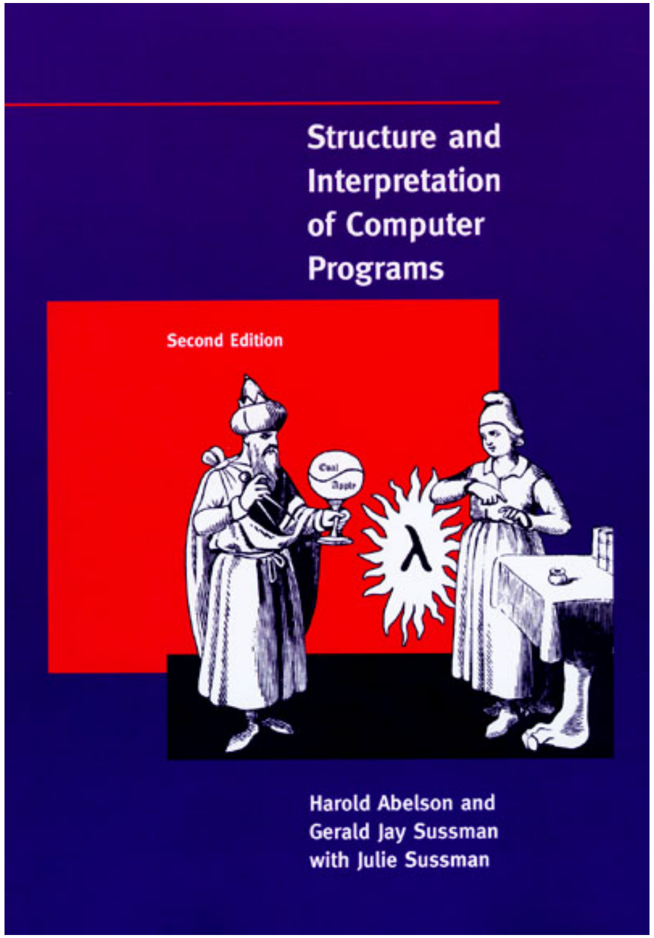
赵海燕，胡振江

Spring Term, 2022

北京大學
PEKING UNIVERSITY

Computer Science = PL Construction ?

- " . . . the technology for coping with large-scale computer systems merges with the technology for building new computer languages, and computer science itself becomes no more (and no less) than the discipline of constructing appropriate descriptive languages"



Types in PL (CS)

1870s	*origins of formal logic*	Frege (1879)
1900s	*formalization of mathematics*	Whitehead and Russell (1910)
1930s	*untyped lambda-calculus*	Church (1941)
1940s	*simply typed lambda-calculus*	Church (1940), Curry and Feys (1958)
1950s	Fortran	Backus (1981)
	Algol-60	Naur et al. (1963)
1960s	*Automath project*	de Bruijn (1980)
	Simula	Birtwistle et al. (1979)
	Curry-Howard correspondence	Howard (1980)
	Algol-68	(van Wijngaarden et al., 1975)
1970s	Pascal	Wirth (1971)
	Martin-Löf type theory	Martin-Löf (1973, 1982)
	System F, F^{ω}	Girard (1972)
	polymorphic lambda-calculus	Reynolds (1974)
	CLU	Liskov et al. (1981)
	polymorphic type inference	Milner (1978), Damas and Milner (1982)
	ML	Gordon, Milner, and Wadsworth (1979)
	intersection types	Coppo and Dezani (1978)
		Coppo, Dezani, and Sallé (1979), Pottinger (1980)
1980s	NuPRL project	Constable et al. (1986)
	subtyping	Reynolds (1980), Cardelli (1984), Mitchell (1984a)
	ADTs as existential types	Mitchell and Plotkin (1988)
	calculus of constructions	Coquand (1985), Coquand and Huet (1988)
	linear logic	Girard (1987) , Girard et al. (1989)
	bounded quantification	Cardelli and Wegner (1985)
		Curien and Ghelli (1992), Cardelli et al. (1994)
	Edinburgh Logical Framework	Harper, Honsell, and Plotkin (1992)
	Forsythe	Reynolds (1988)
	pure type systems	Terlouw (1989), Berardi (1988), Barendregt (1991)
	dependent types and modularity	Burstall and Lampson (1984), MacQueen (1986)
	Quest	Cardelli (1991)
	effect systems	Gifford et al. (1987), Talpin and Jouvelot (1992)
	row variables; extensible records	Wand (1987), Rémy (1989)
		Cardelli and Mitchell (1991)
1990s	higher-order subtyping	Cardelli (1990), Cardelli and Longo (1991)
	typed intermediate languages	Tarditi, Morrisett, et al. (1996)
	object calculus	Abadi and Cardelli (1996)
	translucent types and modularity	Harper and Lillibridge (1994), Leroy (1994)
	typed assembly language	Morrisett et al. (1998)

Self-Introduction

About Me

- 1988: BS, Computer Science, Shanghai Jiaotong Univ.
- 1991: MS, Computer Science, Shanghai Jiaotong Univ.
- 1996: PhD, Information Engineering, Univ. of Tokyo
- 1996: Assistant Professor, Univ. of Tokyo
- 2000: Associate Professor, Univ. of Tokyo
- 2008: Full Professor, National Institute of Informatics
- 2018: Full Professor, Univ. of Tokyo
- 2019: Chair Professor, Peking University

IEEE Fellow, ACM Distinguished Scientist

Member of European Academy

Member of Japanese Engineering Academy

胡振江

- 1988: 上海交通大学 计算机系 本科毕业
- 1996: 日本东京大学 信息工学 博士学位
- 1997: 日本东京大学 工学部 讲师
- 2000: 日本东京大学 工学部 副教授
- 2008: 日本国立信息学研究所 教授
- 2018: 日本东京大学 信息科学技术学院 教授
- 2019: 北京大学 计算机系 讲席教授

日本工学会会士、ACM杰出科学、IEEE Fellow
日本工程院院士、欧洲科学院院士

Research Interests

- Functional Programming (1985-now)
  - **Calculating Efficient Functional Programs**
  - ACM ICFP Steering Committee Co-Chair (2012-2013)
- Algorithmic Languages and Calculi (1992-now)
  - **Parallel programming and Automatic Parallelization**
  - IFIP WG 2.1 Member
- Bidirectional Languages (2003-now)
  - **Bidirectional languages for system/data interoperability**
  - Steering Committee Member of MODELS, ICMT, BX

胡振江

Yanyuan Campus：
Rm 1247, Sci. Blg #1
Changping Campus：
Rm 449, CS Blg.

职称：教授

研究所：软件研究所

研究领域：程序设计语言，函数式语言，软件工程，程序演算

办公电话：86-10-62757974

电子邮件：huzj@pku.edu.cn

个人主页：http://sei.pku.edu.cn/~hu

About Prof. Zhao

- 2003   : PhD, Univ. of Tokyo
- 2003 - : Associate Professor, Peking Univ.

- Research Interest
  - Software engineering
  - Requirements Engineering, Requirements reuse in particular
  - Model transformations
  - Programming Languages

- Contact:
  - Office: Rm. 1809, Science Blg #1
  - Email :   zhhy@sei.pku.edu.cn
  - Phone :   62757670

Teaching Assistant

- Xing Zhang (张星）
- Email： 2001111344@stu.pku.edu.cn

https://zhenjiang888.github.io/PL/

Course Overview

What is this course about?

- Study fundamental (formal) approaches to describing program behaviors that are both precise and abstract.
  - precise so that we can use mathematical tools to formalize and check interesting properties
  - abstract so that properties of interest can be discussed clearly, without getting bogged down in low-level details

What you can get out of this course?

- A more sophisticated perspective on programs, programming languages, and the activity of programming
  - How to view programs and whole languages as formal, mathematical objects
  - How to make and prove rigorous claims about them
  - Detailed study of a range of basic language features

- Powerful tools/techniques for language design, description, and analysis

This course is not about …

- An introduction to programming
- A course on compiler
- A course on functional programming
- A course on language paradigms/styles

All the above are certainly helpful for your deep understanding of this course.

What background is required?

- Basic knowledge on
  - Discrete mathematics: sets, functions, relations, orders
  - Algorithms: list, tree, graph, stack, queue, heap
  - Elementary logics: propositional logic, first-order logic

- Familiar with a programming language and basic knowledge of compiler construction

Textbook

- **Types and Programming Languages**
  - Benjamin Pierce
  - The MIT Press
  - 2002-02-01
  - ISBN: 9780262162098

Let us see how much we can cover in one semester in PKU.

Outline

- Basic operational semantics and proof techniques
- Untyped Lambda calculus
- Simple typed Lambda calculus
- Simple extensions (basic and derived types)
- References
- Exceptions
- Subtyping
- Recursive types
- Polymorphism

Grading

- Activity in class: 20%
- Homework: 40%
- Final (Report/Presentation): 40%

设计一个带类型系统的程序语言，解决实践中的问题，给出基本实现

- 设计一个语言，保证永远不会发生内存/资源泄露。
- 设计一个汇编语言的类型系统
- 设计一个没有停机问题的编程语言
- 设计一个嵌入复杂度表示的类型系统，
  保证编写的程序的复杂度不会高于类型标示的复杂度。
- 设计一个类型系统，使得敏感信息永远不会泄露。
- 设计一个类型系统，使得写出的并行程序没有竞争问题
- 设计一个类型系统，保证所有的浮点计算都满足一定精度要求
- 解决自己研究领域的具体问题

How to study this course?

- Before class: scanning through the chapters to learn and gain feeling about what will be studied

- In class: trying your best to understand the contents and raising hands when you have questions

- After class: doing exercises seriously

★	Quick check	30 seconds to 5 minutes
★★	Easy	≤ 1 hour
★★★	Moderate	≤ 3 hours
★★★★	Challenging	> 3 hours

Chapter 1: Introduction

What is a type system?

What type systems are good for?

Type Systems and Programming Languages

What is a type system (type theory)?

- A type system is a tractable syntactic method for proving the absence of certain (bad) program behaviors by classifying phrases according to the kinds of values they compute.

  - Tools for program reasoning
  - Classification of terms
  - Static approximation
  - Proving the absence rather than presence
  - Fully automatic (and efficient)

What are type systems good for?

- Detecting Errors
  - Many programming errors can be detected early, fixed intermediately and easily.
- Abstraction
  - type systems form the backbone of the module languages: an interface itself can be viewed as "the type of a module."
- Documentation
  - The type declarations in procedure headers and module interfaces constitute a form of (checkable) documentation.
- Language Safety
  - A safe language is one that protects its own abstractions.
- Efficiency
  - Removal of dynamic checking; smart code-generation

Type Systems and Languages Design

- Language design should go hand-in-hand with type system design.
  - Languages without type systems tend to offer features that make type checking difficult or infeasible.
  - Concrete syntax of typed languages tends to be more complicated than that of untyped languages, since type annotations must be taken into account.

> In typed languages the type system itself is often taken as the foundation of the design and the organizing principle in light of which every other aspect of the design is considered.

Homework

- Read Chapters 1 and 2.

- Install OCaml and read “Basics”
  - http://caml.inria.fr/download.en.html
  - http://ocaml.org/learn/tutorials/basics.html