

編程語言的設計原理

Design Principles of Programming Languages

Zhenjiang Hu, Haiyan Zhao, Yingfei Xiong

胡振江、赵海燕、熊英飞

Peking University, Spring Term, 2019



Self-Introduction



The screenshot shows a web browser window with the address bar containing "research.nii.ac.jp/~hu/". The page content includes a portrait of Zhenjiang Hu, his title as Professor at the Programming Research Laboratory, Information Systems Architecture Research Division, National Institute of Informatics (NII), and his previous position as Professor at the Department of Informatics, The Graduate University for Advanced Studies. A paragraph describes his education at Shanghai Jiaotong University and the University of Tokyo, and his career path. A "Research" section follows, detailing his interests in programming languages and software construction. At the bottom, there is a note about a post-doc position and a list of links for current projects.

Zhenjiang Hu

Professor
[Programming Research Laboratory](#)
[Information Systems Architecture Research Division](#)
[National Institute of Informatics \(NII\)](#)

Professor
[Department of Informatics](#)
[The Graduate University for Advanced Studies](#)

I received BS and MS degrees from [Department of Computer Science and Engineering](#) of [Shanghai Jiaotong University](#) in 1988 and 1991 respectively, and Ph.D degree from Department of Information Engineering of [University of Tokyo](#) in 1996. I became a lecturer (assistant professor) in 1997 and an associate professor in 2000 in [University of Tokyo](#). I joined [National Institute of Informatics](#) as a full professor in 2008.

Research

My main interest is in programming languages and software construction in general, and functional programming, program transformation and model driven software development in particular. I am very interested in program calculation based on programming algebras, and I am looking into how to apply this theory to automatic program optimization, systematic parallelization of sequential programs, efficient manipulation of structured documents, and bidirectional model transformation for software development.

Currently, I have a post-doc position available for those who are interested in bidirectional transformation. I welcome excellent people to join my group as [intern students](#), [PhD students](#), or post-doc researchers ([JSPS Fellowship Program](#) and [DAAD-NII Program](#)).

[Current Projects](#) [Completed List](#)

<http://www.research.nii.ac.jp/~hu>



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
- 1997: Lecturer, Univ. of Tokyo
- 2000: Associate Professor, Univ. of Tokyo
- 2008: Full Professor, National Institute of Informatics

北京大学海外杰青(2006-2008)

北京大学长江学者教授(2013.12-)



Research Interest

- **Functional Programming**
 - **Calculating Efficient Functional Programs**
 - ACM ICFP 2011 General Co-Chair
 - ACM ICFP Steering Committee Co-Chair (2012-2013)
 - AMC Haskell Symposium Steering Committee Member (2014-)
- **Algorithmic Languages and Calculi**
 - **Parallel programming and Automatic Parallelization**
 - IFIP WG 2.1 Member (IFIP TC 2, Japan Representative)
- **Bidirectional Transformation Languages in SE**
 - **Bidirectional languages for software evolution**
 - Steering Committee Member of BX, ICMT



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

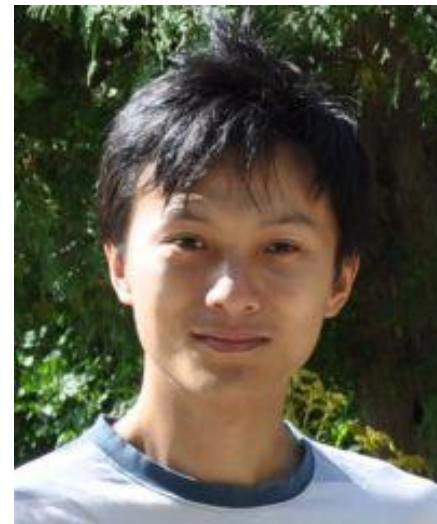


About Prof. Xiong

- 2009: PhD, Univ. of Tokyo
- 2009-2011: Postdoc, Univ. of Waterloo
- 2012: 百人计划研究员, Peking Univ.

- Research Interest
 - Program Analysis
 - Fault Localization & Repair
 - Program Synthesis

- Contact:
 - 理科一号楼1431房间
 - Mail : xiongyf@pku.edu.cn
 - Tel : 62757008



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.

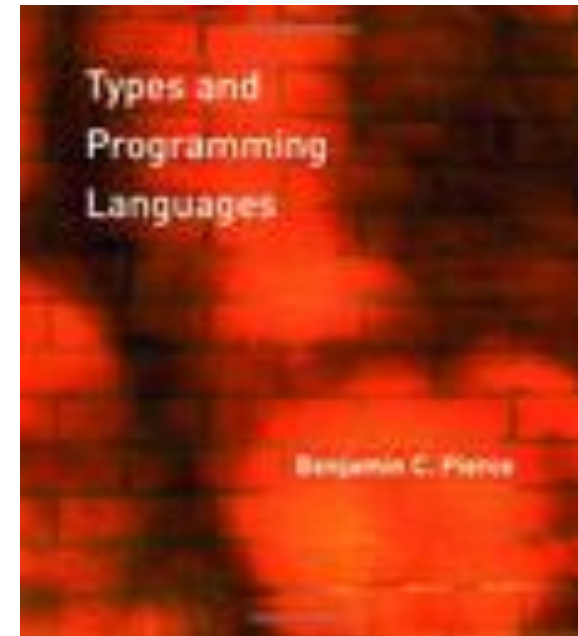
What background is required?

- 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**
- 作者: Benjamin Pierce
- 出版社: The MIT Press
- 出版年: 2002-02-01
- 页数: 648
- 定价: USD 72.00
- 装帧: Hardcover
- ISBN: 9780262162098



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
- 设计一个微分编程语言，保证写出的表达式都是可微分的
- 设计一个概率编程语言，编写概率模型并给出推导
- 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 speaking out 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



Personnel

- Instructors
 - Zhenjiang Hu, Professor, NII/PKU
hu@nii.ac.jp
 - Haiyan Zhao, Associate Professor, PKU
zhhy@sei.pku.edu.cn
 - Yingfei Xiong, Assistant Professor, PKU
xiongyf@pku.edu.cn
- Teaching Assistant:
 - 刘鑫远, liuxinyuan@pku.edu.cn



Information



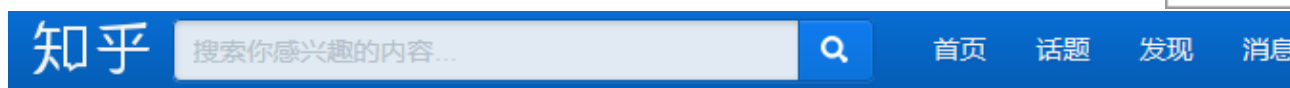
- Course website:

<http://sei.pku.edu.cn/~xiongyf04/DPPL/main.htm>

- Syllabus
- News/Announcements
- Lecture Notes (slides)
- Other useful resources



Recommendation from a student



北京大学

北京大学最好的计算机类课程有哪些？

添加评论 分享 · 邀请回答

举报

查看全部 5 个回答



吴争锴，但是鸽子为什么这么大呢？



王迪、陈牧歌等 18 人赞同

编程语言的设计原理 编程语言的设计原理 授课老师：胡振江 赵海燕 熊英飞

利益相关：熊英飞是我的本科导师

这课用的教材是TAPL, [Types and Programming Languages](#)。书很经典，讲的也比较深，我看了一些其他用这本书的学校稍微讲讲recursive typing后面的一些拓展都不讲。

三位老师本身都是在PL方面的学者，讲课讲的也不错，而且前期还会让同学们现场写代码，不管是有志于进行PL研究 还是 想加深对于编程语言认识的 同学都有帮助。

缺点是后面内容有点赶，而且我认为这课最好还是有个formal的exam比较好。

另外熊老师的软件分析技术[软件分析技术](#) 我看过不少课件，应该也是好课，但我自己没上过。

@熊英飞

编辑于 2016-12-25 收起评论 取消感谢 分享 收藏 · 没有帮助 · 举报 · 作者保留权利



Chapter 1: Introduction

What is a type system?

What type systems are good for?

Type Systems and Programming Languages

Why type system?

- Art vs. Knowledge
 - Art cannot be taught, while knowledge can
 - What people have invented
 - How to interpret them abstractly
 - How to reason their properties formally
- Why formal reasoning important
 - Poorly designed languages widely used
 - Java array flaw
 - PHP, Javascript, etc.
 - Well designed language needs strictly reasoning
 - Devils in details

The three worst programming languages:

<https://medium.com/smalltalk-talk/the-three-worst-programming-languages-b1ec25a232c1#.jdsfib20v>



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

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
- Fully automatic (and efficient)
- Classification of terms
- Static approximation
- Proving the absence rather than pr

Tractable : be finished in short time, often polynomial
 Syntactic: be part of the programming language



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
 - Fully automatic (and efficient)
 - **Classification of terms**
 - Static approximation
 - Proving the absence rather than presence

True, false	Boolean
1, 2, 3, ...	Int
'a', 'b', 'c', ...	Char



W

- Given a property that correct programs should satisfy, does this program satisfy it?
- Based on Rice's theorem, we cannot precisely answer the question on any non-trivial property
- Approximation method 1 (type checking): only determine the program definitely satisfies a property
- Approximation method 2 (testing): only determine the program definitely violates a property
- Can you give a correct program that cannot type-check?
 - Static approximation
 - Proving the absence rather than presence



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 typechecking 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>