

編程語言的設計原理

Design Principles of Programming Languages

Haiyan Zhao, Yingfei Xiong, Zhenjiang Hu
趙海燕、熊英飛、胡振江

Peking University, Spring Term, 2014



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
- 1997: Assistant Professor, Univ. of Tokyo
- 1997: Lecturer, Univ. of Tokyo
- 2000: Associate Professor, Univ. of Tokyo
- 2008: Full Professor, NII

北京大学海外杰青(2006-2008)

北京大学長江講座教授(2013.12-)



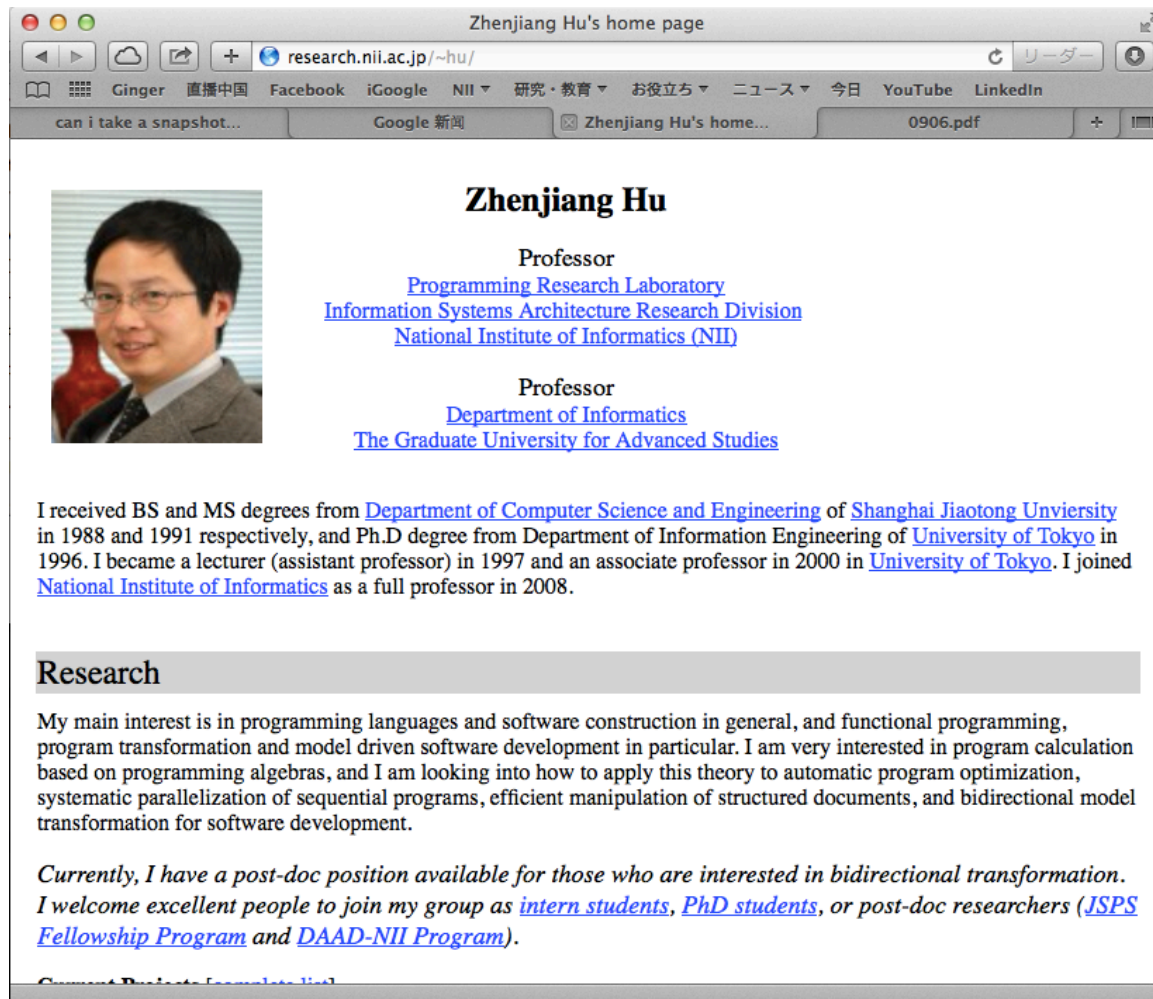
Research Interest



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



More Information



The screenshot shows a web browser window titled "Zhenjiang Hu's home page" with the URL "research.nii.ac.jp/~hu/". The page content includes a profile picture of Zhenjiang Hu, his titles as Professor at the Programming Research Laboratory, Information Systems Architecture Research Division, National Institute of Informatics (NII), and at the Department of Informatics, The Graduate University for Advanced Studies. It also lists his educational background (BS and MS from Shanghai Jiaotong University, Ph.D from University of Tokyo) and his research interests in programming languages and software construction. A section for "Research" describes his main interests and mentions a post-doc position available for those interested in bidirectional transformation.

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: [\[open to link\]](#)

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



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
 - Software Engineering
 - Programming Languages

- 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?

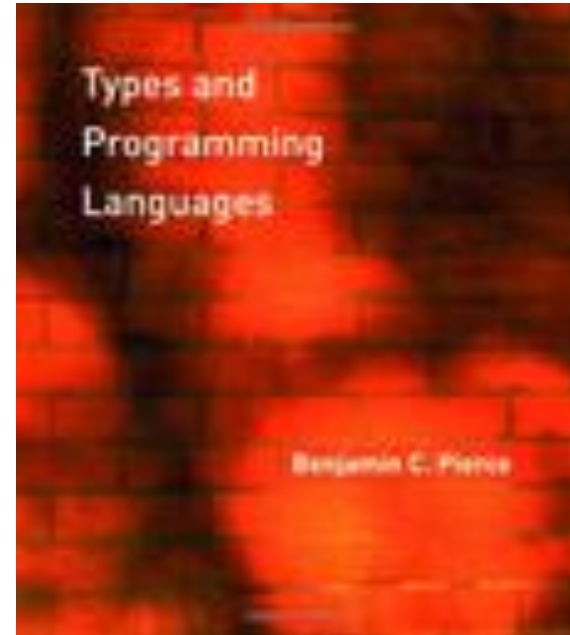


- Basic knowledge on
 - Discrete mathematics: sets, functions, relations, orders
 - Algorithms: list, tree, graph, stack, queue, heap
 - Elementary logics: propositional logic, first-order logic
- Familiar with a programming language and basic knowledge of compiler construction



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



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
- [Higher-order systems]



Grading



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



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



Personnel



- Instructors
 - Haiyan Zhao, Associate Professor, PKU
zhhy@sei.pku.edu.cn
 - Yingfei Xiong, Assistant Professor, PKU
xiongyf@pku.edu.cn
 - Zhenjiang Hu, Professor, NII/PKU
hu@nii.ac.jp
- Teaching Assistant:
 - Jun Li, PhD student, PKU
lijun09@sei.pku.edu.cn



Information



- Course website:
<http://sei.pku.edu.cn/~xiongyf04/DPPL/2014.htm>
 - Syllabus
 - News/Announcements
 - Lecture Notes (slides)
 - Other useful resources



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 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>

