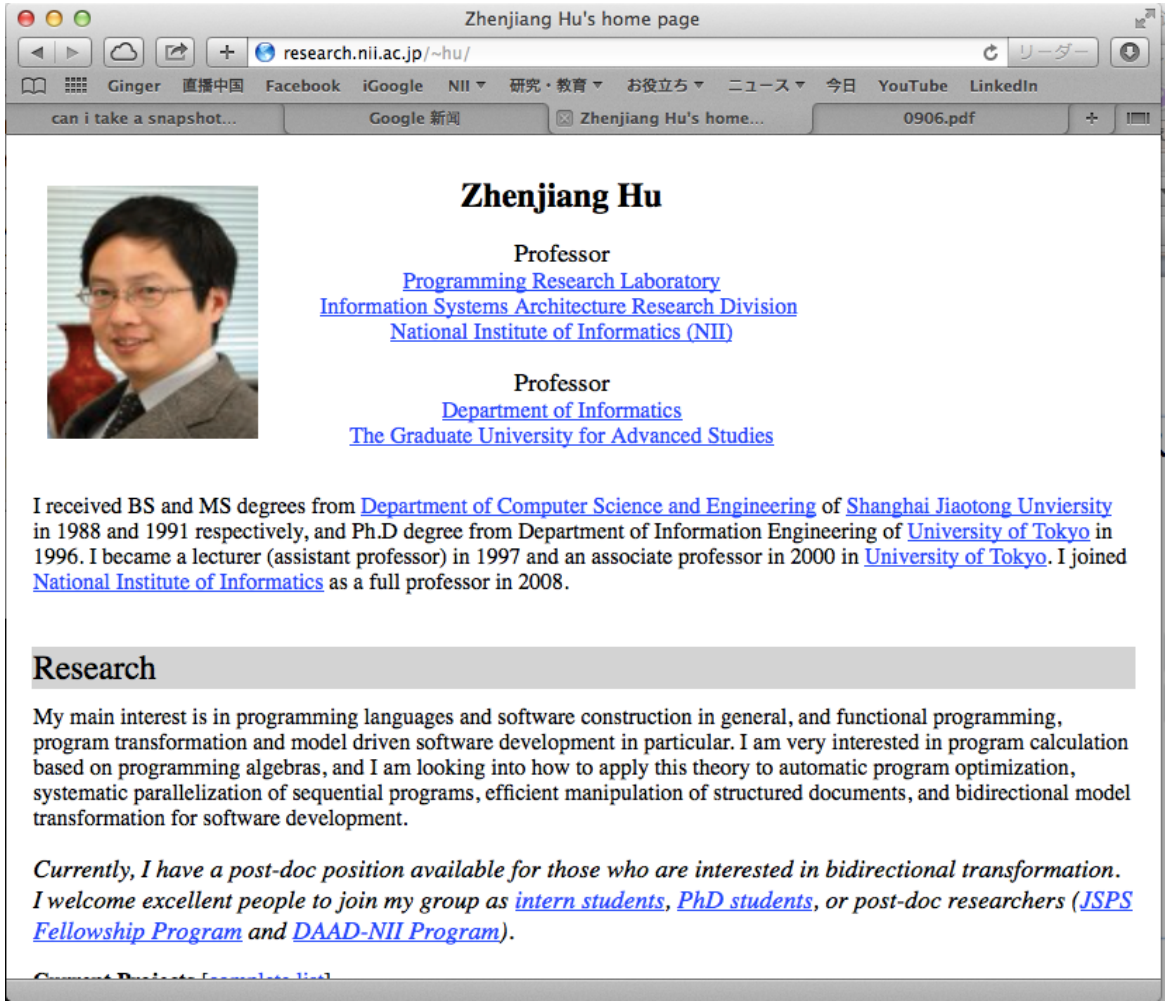
編程語言的設計原理
Design Principles of Programming Languages

Haiyan Zhao, Zhenjiang Hu, Yingfei Xiong
赵海燕、胡振江、熊英飞

Peking University, Spring Term, 2017

Self-Introduction

Zhenjiang Hu



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

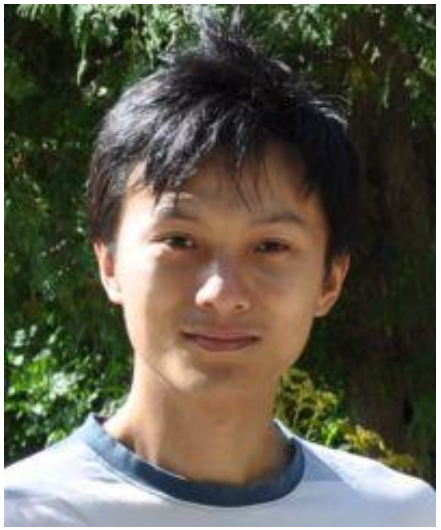
About Prof. Zhao

- 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
  - Fault Localization & Repair

- 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 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:
  - 周钊平，zhouzhaoping@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

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