# Programming Languages

Nate Nystrom
University of Lugano

Amanj Sherwany
Ilya Yanok

usi-pl-staff@googlegroups.com
http://inf.usi.ch/nystrom/teaching/pl/sp13

# Who are we?

Nate Nystrom

Amanj Sherwany

Ilya Yanok

# About me

1991–1995 Purdue: BS Computer Science, Mathematics

1996–1998 Purdue: MS Computer Science

1998–1999 Hewlett-Packard: compiler engineer

1999–2006 Cornell: PhD Computer Science

2006–2009 IBM Research PL/SE group

2009–2010 Arlington, Texas: Assistant Professor

2011–present University of Lugano: Assistant Professor

# My research

Using programming languages to solve systems problems:

Extensibility

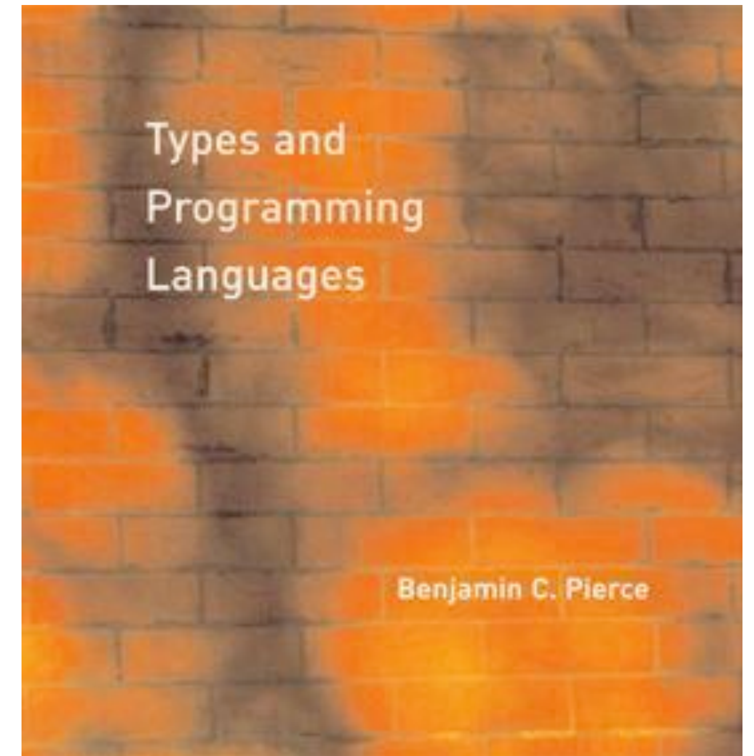
- Polyglot, an extensible compiler framework
  - http://www.cs.cornell.edu/Projects/polyglot
- A framework for Scala compiler plugins

Concurrency and distribution and fault tolerance

- X10, a concurrent OO language for HPC
  - http://www.x10-lang.org
- Firepile, a Scala library for GPU programming
- Languages for reasoning about relaxed consistency

# Book / scribing

*Types and Programming Languages*
By Benjamin C. Pierce




I will **not** follow the book very closely

Most lectures will **not** use slides

Each lecture, one of you will be a **scribe** and take notes for the class.
(This does not mean you shouldn't take your own notes.)

We will post the scribe notes on the web page. You should **edit** your notes for clarity and accuracy. Send the staff your notes within a week of the lecture so we can post them.

# Grading

Assignments 40%

Midterm exam 25%

Final exam 25%

Scribing and participation 10%

# Assignments

Some writing.

Some math.

Some programming, but not too much

About one homework every 7-10 days

Plus occasionally some small exercises due before the next lecture, usually one or two short questions

# Website

http://inf.usi.ch/faculty/nystrom/teaching/pl/sp13/

Everything will be posted there

# Moodle

Exists.

How would you like us to communicate with you?

# Community

Moodle forums are an abomination unto Nuggan.

Join the G+ community for this course.

- Discuss assignments, ask questions there.
- Announcements will go there and to the web page.

Questions for the staff: usi-pl-staff@googlegroups.com

What do *you want* to get out of this course?

What do *I want you* to get out of this course?

# What do *I want you* to get out of this course?

- Become familiar with different programming paradigms
- Understand principles behind programming languages
- Apply these principles to solve “real” problems

# What do *I want you* to get out of this course?

- Become familiar with different programming paradigms
- Understand principles behind programming languages
- Apply these principles to solve “real” problems

# Language features

We'll look at features common across multiple languages

Variables
Functions
Eager and lazy evaluation
Mutable state (assignment)
Exotic control-flow constructs: exceptions, continuations
Typing, subtyping, polymorphism
Objects

# Different paradigms

We'll look at several different programming languages and try to distill them to their essential features

We'll also look at how those features interact

- e.g., parametric polymorphism + subtyping = WTF!?

But, we'll program primarily in **Haskell**

# Why Haskell?

# Get out of your comfort zone

# Learning zone

# Panic zone

# Why Haskell?

Haskell is a **pure** functional language

- No assignment, no loops
- You have to think differently about programs

Haskell is **lazy**

- Think about computation as function composition, not as a sequence of instructions

# Haskell crash course

This Thursday

Bring your computers

# What do *I want you* to get out of this course?

- Become familiar with different programming paradigms
- Understand principles behind programming languages
- Apply these principles to solve “real” problems

# PL principles

Focus is on **semantics** (what do programs mean?)

(Mostly) ignore **syntax** (what do programs look like?)

# Dynamic semantics

How does a program behave?

How is a program evaluated?

We'll experiment with different semantics by implementing **interpreters**

We'll define behavior **formally** with **operational semantics**

Formal semantics lets you state precisely and **prove** properties of programs

# What does this expression do?

```
'1' + '2'
```

# Static semantics

Restrictions on programs to provide (some) correctness guarantees

- e.g., if this program type checks, it won’t core dump

Focus on **type systems**

Some other formal methods (e.g., program verification) are covered in other classes

# What do *I want you* to get out of this course?

- Become familiar with different programming paradigms
- Understand principles behind programming languages
- Apply these principles to solve “real” problems

# Languages are models of dynamic systems

A programming language provides **abstractions** and ways to **compose** these abstractions

The languages you are familiar with are **models** of computer systems

They provide abstractions for data and computation

| | **abstractions** | **compositions** |
|---|---|---|
| **Assembly languages** | addresses, registers, instructions, labels | sequences of instructions |
| **Procedural languages** | booleans, arithmetic, loops, arrays, procedures | sequences of statements, procedure calls |
| **OO languages** | objects, methods, fields, classes | method invocation, inheritance |
| **Functional languages** | first-class functions, algebraic data types | function application, type constructors |

# Languages are models of dynamic systems

But languages can model not just computer systems, but any dynamic system

**General-purpose languages** provide abstractions for modeling computation

**Domain-specific languages** provide abstractions for other domains

# Domain-specific languages

| | **abstractions** | **compositions** |
|---|---|---|
| **SQL** | relations, tuples, queries | joins, selection, projection |
| **make** | files, build rules | dependencies |
| **lex** | characters, strings | sequences, alternation (\|), repetition (*) |
| **yacc** | tokens, nonterminal symbols | grammar rules |
| **OpenFlow** | packets, network flows, channels | matching, actions (drop, forward, etc) |
| **OpenSCAD** | shapes | union, intersection, linear transformations |

# Domain-specific languages

When designing a system, can be useful to **think of the system as a language**

Implement the system as an interpreter for a domain-specific language

A DSL can either:

- be a language **in its own right** or
  - e.g., yacc
- be **embedded** in a general-purpose language as a library
  - e.g., parser combinators

Same issues that arise with general-purpose languages arise with domain-specific languages

- name binding, control-flow, mutation, evaluation order, …
- similar problems => similar solutions

# Homework for Thursday

Do Assignment 0. It’s easy.

- https://docs.google.com/forms/d/1Mers7bsiRrG6_VF9YcYkIEM-AIiIjAtoz90cWHWSt40/viewform

Install ghc (the Glasgow Haskell Compiler)

- 7.4 or later
- http://haskell.org
- Mac users:
  - brew install ghc haskell-platform
  - port install ghc