

# CS4200-B: Summary & Further Study

**Eelco Visser**



**CS4200 | Compiler Construction | January 14, 2021**

# Grading

## Weights

- Homework: 5%
- Project: 50%
- Exam: 45%

**All grades  $\geq 5.0$**

## Final grade

- weighted average
- $\geq 5.8$

# Exam and Resit

**January 22: Resit CS4200-A**

- 13:30-16:30

**January 29: Exam CS4200-B**

- 13:30-16:30

**April 9: Resit CS4200-B**

- 13:30-16:30

## Topics

- Everything we studied in the lectures
- Example exam questions: homework assignments

## Compiler Components

- What did we study?

## Meta-Linguistic Abstraction

- Another perspective

## Further Study & Research

- Courses and conferences

## Research Challenges

- Including topics for master thesis projects

## Exam Dates

# Compiler Components

# What is a Compiler?

A bunch of components for translating programs



# Compiler Components

## Parser

- Reads in program text, checks that it complies with the syntactic rules of the language, and produces an abstract syntax tree, which represents the underlying (syntactic) structure of the program.

## Type checker

- Consumes an abstract syntax tree and checks that the program complies with the static semantic rules of the language. To do that it needs to perform name analysis, relating uses of names to declarations of names, and checks that the types of arguments of operations are consistent with their specification.

## Optimizer

- Consumes a (typed) abstract syntax tree and applies transformations that improve the program in various dimensions such as execution time, memory consumption, and energy consumption.

## Code generator

- Transforms the (typed, optimized) abstract syntax tree to instructions for a particular computer architecture. (aka instruction selection)

# ChocoPy Compiler

## Syntax definition

- Parser through generation, design of abstract syntax

## Static semantic analysis

- Name analysis
  - Lexical scoping, type-dependent name resolution
- Type checking
  - Class-based object-oriented language with sub-typing

## Desugaring

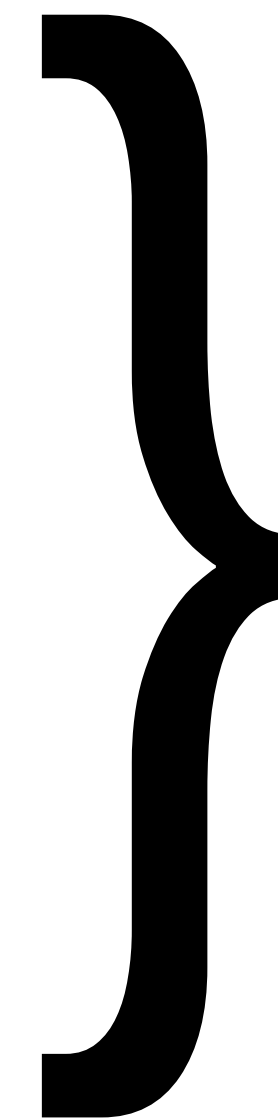
- Simple rewrite rules and strategies

## Code generation

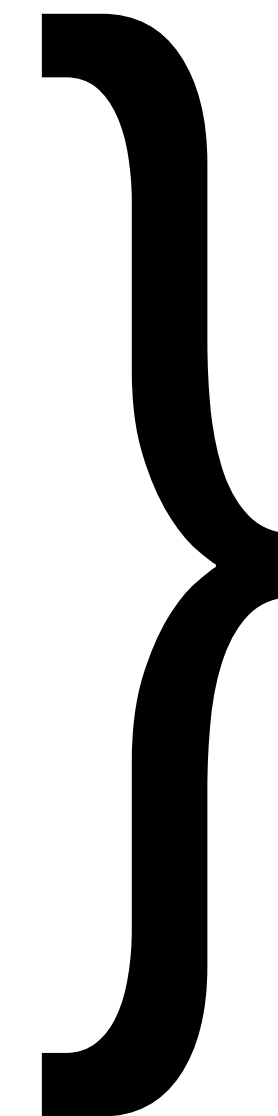
- Generation of Risc V instructions
- AST-to-AST transformation

## Data-flow analysis

- Optimization



CS4200-A



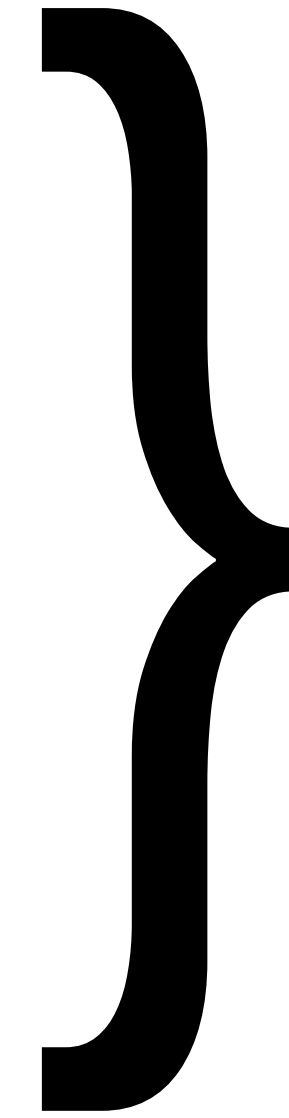
CS4200-B



# Further Study

## More Compiler Components

- Static analyses
- Optimization
- Register allocation
- Code generation for register machines
- Garbage collection



CS4200-B

## Other Object Languages

- Functional programming: first-class functions, laziness
- Domain-specific languages: less direct execution models
- Data (description) languages
- Query languages
- ...

# Meta-Linguistic Abstraction

# Separation of Concerns

## Language design

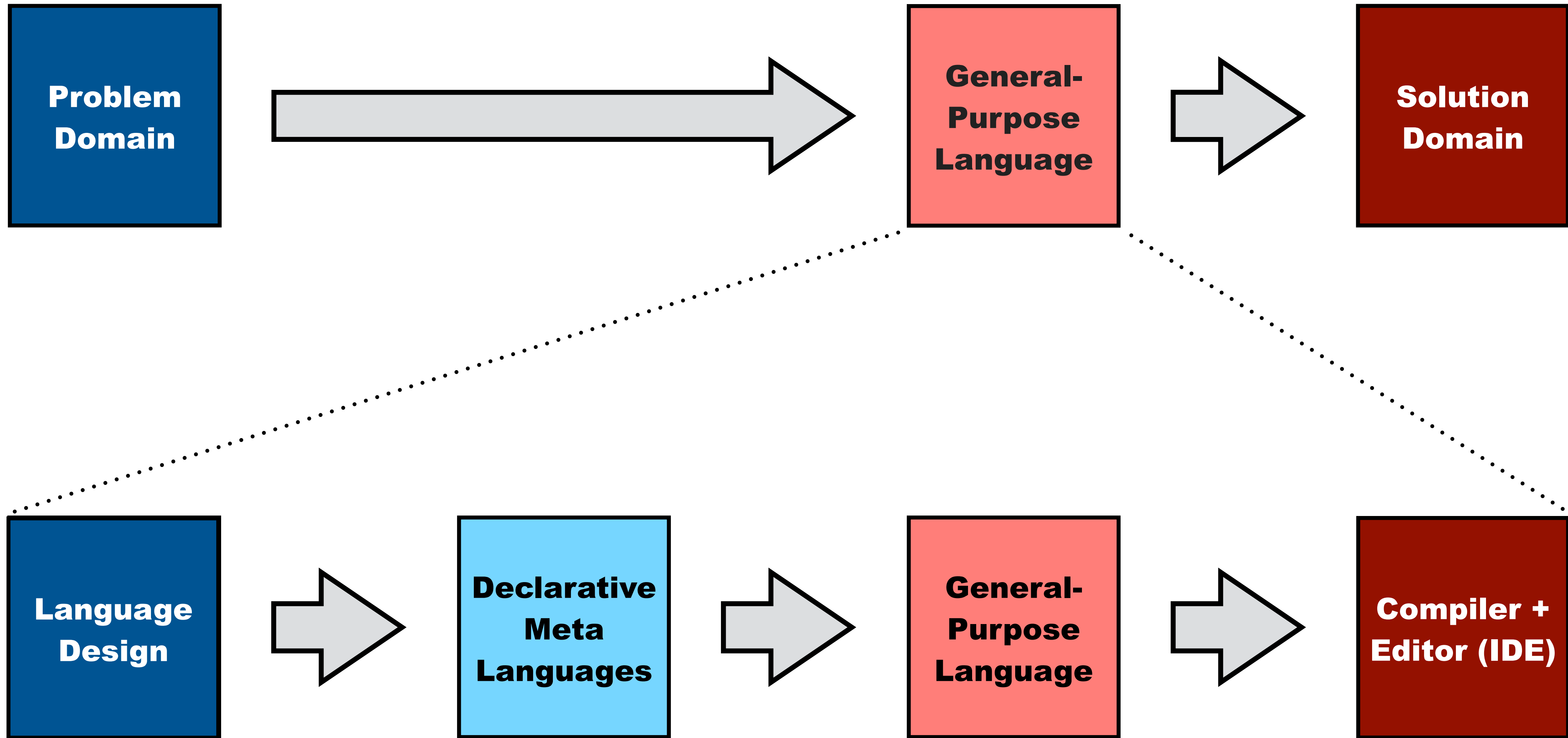
- Define the properties of a language
- Done by a language designer

## Language implementation

- Implement tools that satisfy properties of the language
- Done by a language implementer

## Can we automate the language implementer?

- That is what language workbenches attempt to do



That also applies to the definition of (compilers for) general purpose languages

# Declarative Language Definition

## Objective

- A workbench supporting design and implementation of programming languages

## Approach

- Declarative multi-purpose domain-specific meta-languages

## Meta-Languages

- Languages for defining languages

## Domain-Specific

- Linguistic abstractions for domain of language definition (syntax, names, types, ...)

## Multi-Purpose

- Derivation of interpreters, compilers, rich editors, documentation, and verification from single source

## Declarative

- Focus on what not how; avoid bias to particular purpose in language definition

# Spoofax Meta-Languages

## SDF3: Syntax definition

- context-free grammars + disambiguation + constructors + templates
- derivation of parser, formatter, syntax highlighting, ...

## Statix: Names & Types

- name resolution with scope graphs
- type checking/inference with constraints
- derivation of name & type resolution algorithm

## Stratego: Program Transformation

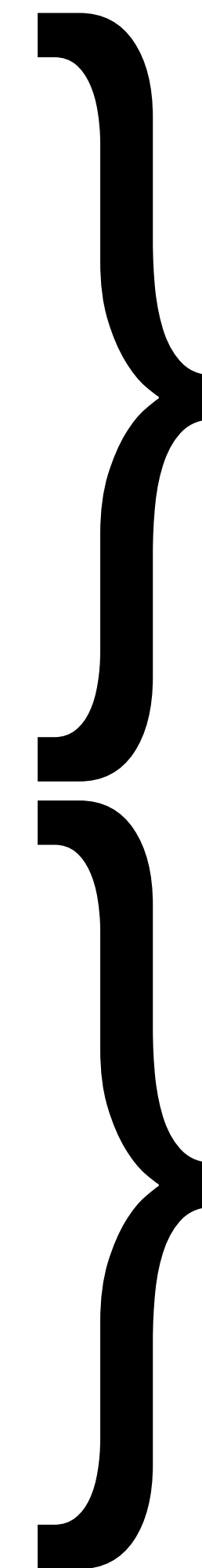
- term rewrite rules with programmable rewriting strategies
- derivation of program transformation system

## FlowSpec: Data-Flow Analysis

- extraction of control-flow graph and specification of data-flow rules
- derivation of data-flow analysis engine

## DynSem: Dynamic Semantics

- specification of operational (natural) semantics
- derivation of interpreter



CS4200-A

CS4200-B

# Course Evaluation

**What do you think of the course, organization, project, etc.?**

# Research Challenges in Compiler Back-Ends



# Transformation

## New Transformation Language

- Operating on richer program model
- Generic (traversal) strategies
- Statically typed
- Type-preserving transformations
  - ▶ intrinsically or extrinsically verified?
- Semantics-preserving transformations
  - ▶ intrinsically or extrinsically verified?

## Transformation Algorithms

- Refactorings, Optimizations

## Code Generation / Instruction Selection

- BURS: Bottom-up rewrite system: Find cheapest / best mapping to code

# Analysis

## FlowSpec

- Declarative specification of data-flow analyses
- Increase expressiveness to cover more analyses
- High-performance execution of analyses
- Incremental execution of analyses (during transformation)

## Data-flow Dependent Static Semantics

- Definite assignment in Java
- Borrow checking in Rust

## Language Engineering

- Better integration in language workbench
- Program model

# Dynamics

## Interpreters from dynamic semantic specification

- Prototype for DynSem [RTA15] applied to Grace [DLS17], Tiger

## Frame-based interpreters

- Scope graph describes memory model [ECOOP16]
- Define in DynSem applied to realistic languages
- Correct-by-construction [POPL18] => dependently-typed DynSem
- Language-independent garbage collection

## Partial evaluation

- Specialize DynSem interpreter to language-specific set of semantic rules
- Specialize language-specific rules to specific program

## Compiler generation

- Derive compiler from dynamic semantics specification

# Verification

## Intrinsically Typed Definitional Interpreters

- For Middle Weight Java (MJ) in Agda [POPL18]
- Extend to more sophisticated type systems (generics, System F)
- Extend to more sophisticated effect systems (continuations, algebraic effects)
- Extend approach to other operations: transformation, code generation

## Extrinsic Proofs

- Generation of extrinsic type soundness proof
- Other properties

## Other Verification Challenges

- Semantics preservation of DSL code generators
- Correctness of meta-DSLs

# Theme: Incremental Compilation

## Make all (meta) language processing incremental

- Effort proportional to size of change

## Modular analysis out of the box

- Static analysis incremental based on (scope graph) dependencies

## Compiler = build system

- Use PIE to glue together language processing pipelines

## In progress

- Incremental parsing (Beta in Spoofax)
- Incremental compilation for Stratego (Beta in Beta)
- Incremental compilation for WebDSL

# Research Challenges in Compiler Front-Ends

# Vision: Language Designer's Workbench

## High-Level Declarative Language Definition

- Human readable / understandable definition
- Serves as reference documentation

## Verification

- Automatically verify properties of language definition
- Type soundness of interpretation
- Type preservation of transformations
- Semantics preservation of transformation

## Implementation

- Generate production quality tools from language definition
- Interpreter, compiler, IDE with refactoring, completion, ...
- Correct-by-construction, high performance



# Syntax

## High-Performance Parsing

- JSGLR2: 2x to 10x speed-up compared to JSGLR
- More speed-up possible?
- Explore effects of different parse table formats (LR, SLR, LALR)

## Error Recovery & Error Messages

- Apply error recovery approach of [TOPLAS12] to JSGLR2
- Generate high quality error messages

## Incremental Parsing

- Re-parse effort proportional to change of program text
- Approach: adapt Graham/Wagner algorithm to SGLR

## Extensible Syntax

- Extend syntax during parsing to support extensible languages



## Code Completion

- Semantic code completion based on static semantics

## Refactoring

- Sound refactoring scripts
- Refactoring based on scope graph program model
- New NWO MasCot project: programming and validating software restructurings

## Live Language Development

- Immediate response after edit of language definition
- Requires: incremental evaluation of all compiler components
- Ongoing work: PIE DSL for interactive software development pipelines

## Language Deployment

- Generate stand-alone language implementation: PIE partial evaluation

## Portable Editors

- Portable editor bindings based on AESI model (Pelsmaecker)
- Case study: bindings for Visual Studio, IntelliJ, LSP

## Web Editors

- Generate language-specific editors for use in web browser
- Architectural questions
  - ▶ All processing client-side? Stateful back-end on server? Scalability?
  - ▶ Performance of Web Assembly (WASM) better than JS?
- Collaborative editing (operational transform)

## Interactive Notebooks

- Combine documents with code in several languages and results of execution

# Statics with Statix

## Specification of type systems with Statix

- Subset of CHR (Constraint Handling Rules) + domain-specific constraints for scope graphs and relations
- Support more advanced type systems
- Structural types, polymorphism (generics), sub-typing [OOPSLA'18]
  - ▶ Better encoding?
  - ▶ Generalization (for parametric polymorphism)?

## Solver

- Parallel solver for multi-module analysis
- Matrix-based name resolution algorithm?
- Correctness wrt resolution calculus?
- Scalability: modular and incremental analysis?

## Substructural Type Systems

- Linear types
- Rust

## Gradual Type Systems

- Gradual type theory: encode calculi and experiment
- Implement existing gradual type checkers
  - Python, TypeScript, Dart, Hack
- Design gradual type system for Stratego

## Dependent Types

- Agda, Idris

## Program Model

- Extend term data model to incorporate scopes and types
- Persistent storage
- Query: retrieve information based on scope graph model
  - ▶ All methods in class A
- Construction
  - ▶ well-formed wrt static semantics

## Random Program Generation

- Generation of well-formed and well-typed programs
- based on syntax + static semantics
- for testing compilers and other language processing tools

# Theme: Error Localization and Diagnosis

## Error Localization

- What program element is responsible for the failure?
- Minimal unsatisfiable core
  - ▶ What is the smallest set of constraints that correspond to failure?

## Error Diagnosis

- Generate good (understandable) explanation of error
- Based on unsat core

# Studying Programming Languages



# Courses

## Software Verification (Q3)

- Learn the basics of mechanised verification with Agda dependently typed programming language

## Web Programming Languages (Q3)

## Language-Based Software Security (Q4)

## Language Engineering Project (Q4)

- Develop a Spoofax language definition for an interesting language

## Seminar Programming Languages (Q1)

- Read and discuss papers from the PL literature

## System Validation (Q1)

- Check properties of (concurrent) software with model checking

## Master Thesis Project in PL group



# Industrial Internships

## Oracle Labs (Zürich)

- Applications of Spoofox: GreenMarl, PGQL
- Other projects (Truffle/Graal)

## Canon (Venlo)

- Designs and manufactures digital printers
- New project to investigate design of DSLs in digital printing domain

## Philips (Best)

- Software restructuring

## Other

- Opportunities for language design and implementation projects at other companies

# Conferences

## ACM Special Interest Group on Programming Languages

- <http://sigplan.org/>

## Key SIGPLAN Conferences

- POPL: Principles of Programming Languages
- PLDI: Programming Language Design and Implementation
- ICFP: International Conference on Functional Programming
- OOPSLA/SPLASH: Systems, Programming Languages, and Applications
- SLE: Software Language Engineering
- GPCE: Generative Programming

## Other Conferences

- ECOOP: European PL conference
- ESOP: European Symposium on Programming

# Summer Schools

## **PLMW: Programming Languages Mentoring Workshop**

- technical sessions on cutting-edge research in programming languages, and mentoring sessions on how to prepare for a research career
- At ICFP, POPL, PLDI, SPLASH

## **OPLSS: Oregon Programming Languages Summer School**

- Foundational work on semantics and type theory
- Advanced program verification techniques
- Experience with applying the theory

## **DSSS: DeepSpec Summer School**

- Formal verification

## **PLISS: Programming Language Implementation Summer School**

- Programming language systems, implementation, analysis

# After the Master

## PhD

- Dive into PL research for four years
- Develop new PL theory, designs, and implementations
- Write research papers and a dissertation
- ~~Present your work at conferences around the world~~

## PL in industry

- Develop compilers, analyses, run-time systems
- Contribute to development of industrial programming languages
  - ▶ Oracle Labs (PGX), Google (Dart), Amazon (Cloud9), Canon (OIL)

# Wanted: PhD Students in PL

## Software Restructuring

- A principled approach to programming refactorings/restructurings
- Application: Transforming C++ code

## Language Engineering

- Static semantics and type checking
- Deriving interpreters, compilers from dynamic semantics

# Wanted: Research Assistants

## Goal

- help with an ongoing research project
- often: programming work

## Research Assistant

- 4 - 8 hours per week (flexible)
- Appointment per project (language)

# Wanted: Java Performance Engineer

## Goal

- Work on speeding up JSGLR2

## Experience

- Java programming, performance engineering

## Research Assistant

- 4 - 8 hours per week (flexible)
- Appointment per project (language)



# Wanted: Grammar Engineer

## Goal

- A collection of high quality syntax definitions for key languages
- Spoofox with `batteries included`
- Speeding up research case studies

## Developing Syntax Definitions

- High quality
- High coverage

## Research Assistant

- 4 - 8 hours per week (flexible)
- Appointment per project (language)



# Wanted: Web Programmer

## Academic Workflow Engineering

- Make university work better with web apps that automate workflows
- Education
  - ▶ WebLab, mystudyplanning, EvaTool
- Research
  - ▶ [conf.researchr.org](http://conf.researchr.org), [researchr.org](http://researchr.org)
- Administration

## Combine with PL research

- Use high-level web PLs (WebDSL, IceDust)
- Contribute to better abstractions for web programming

Except where otherwise noted, this work is licensed under

