Principles of Programming Languages, SCS Concentration

Robert Harper, Concentration Director and Advisor
Location: GHC 9229

Amy Weis, Concentration Coordinator
Location: GHC 4115

Principles of Programming Languages Concentration

This concentration is available to SCS students only.

Programming languages play a central role in computer science. All programs are written in a language, and it is obvious that some are better than others, at least for some purposes. The constant demand for new languages reflects the changing demands for constructing reliable and maintainable software systems. Academic research in programming language principles has led to numerous advances in language design, language implementation, and program verification intended to meet these changing expectations through the development of a rigorous theory of programming languages.

Carnegie Mellon is a recognized leader in programming languages, characterized by a strong emphasis on the centrality of type theory, a consolidation of ideas in mathematical semantics, programming logics, and programming language design and implementation. The purpose of the PoPL concentration is to teach the comprehensive view of the field that has been developed here over many decades. Type theory teaches how to define a language, and how to show that it is well-defined, free of internal contradictions. It teaches the mathematical foundations for abstraction and modularity, concepts that are fundamental to building maintainable systems. It teaches how to use a rigorous language definition as the basis for building a compiler that correctly implements the definition, and provides the tools necessary to achieve it. It teaches the logical foundations of program development, how to precisely specify the intended behavior of a program, and how to use machine tools to verify that a program meets those expectations. It gives precise meaning to language concepts, relating them to one another, and distinguishing concepts that are often confused or conflated. It teaches how to specify and verify the resource usage of a program (such as its sequential and parallel time and space complexity) without resorting to a model of how it is implemented on a machine; it supports using actual code, rather than pseudo-code, for defining and analyzing algorithms.

The PoPL concentration is of value to a broad range of students. For the practically minded it will provide the foundation for structuring and validating programs, using type systems or more advanced forms of specification. For the theoretically minded it will provide the foundation for understanding the close relationship between specification and programs on one hand and mathematical conjecture and proof on the other. The elegance of the PoPL lies in their unification of these two perspectives: the theory applies directly to the practice, and the practice informs the theory.

Learning Objectives

The PoPL concentration is characterized by a collection of learning outcomes that it seeks to achieve. These may be summarized by the knowledge that students may expect to gain by concentrating in the area. By their choice of electives each student will choose an emphasis within the area; the required courses ensure that this includes at least the first five objectives:

- Specify the concrete and abstract syntax of a programming language, including a precise specification of the binding and scope of declarations.
- Define the static semantics (compile-time constraints) of a programming language using typing judgments, and how to state and prove that it properly defined.
- Define the dynamics semantics (run-time behavior) of a language using operational and denotational methods.
- Verify rigorously that the statics and dynamics of a language are coherent, a property commonly called type safety.
- Understand the propositions-as-types principle, which relates programs to proofs and specifications to theorems, and know how to apply it in language design and program verification.
- Formulate type and assertion languages for specifying the behavior of a program, and how to verify that a program satisfies such a specification.
- Specify the cost (sequential and parallel time and space complexity) for a program written in a precisely defined language, and how to verify that a given program meets stated cost bounds.
- Use software tools to verify both the properties of languages and the specifications of programs written in well-defined languages.
- Use the static and dynamic semantics of a language to derive a compiler for it that complies with these definitions, and how to use types and verification tools to ensure compiler correctness.
- Relate a language definition to its implementation, both in terms of the run-time structures required, but also to validate abstract cost measures in an implementation.

Curriculum

REQUIREMENTS

This concentration requires one core course plus three electives.

Required Course:

		Units
15-312	Foundations of Programming Languages	12

Electives (complete three of the following):

15-311	Logic and Mechanized Reasoning	9
15-316	Software Foundations of Security and Privacy	9
15-317	Constructive Logic	9
15-413	Advanced Foundations of Programming Languages	12
15-414	Bug Catching: Automated Program Verification	9
15-417	HOT Compilation	12
15-714	Resource Aware Programming Languages	12
In addition to the above list, certain graduate-level Programming Languages course(s) may also qualify as Elective(s) and may be taken with prior permission of the concentration advisor and course instructor(s).		

Students can apply one semester of a senior honors research thesis or research-based independent study in a topic related to this concentration, as approved by the concentration director/advisor, as one of the elective courses for this concentration. This research should be supervised by a member of the Principles of Programming faculty in the Computer Science Department. This research must have a significant communication component, including a paper or technical report, and a poster presentation. Any research course can count for at most 12 units toward the concentration and can count for at most one elective.

Transfer of credit for courses taken outside of Carnegie Mellon University toward this concentration will not be allowed.

Double Counting

15-312 may be double counted towards any major, minor or other concentration being pursued by the student. No other double counting is permitted.

Advising and Management

Participation in this concentration is supervised by the concentration coordinator in cooperation with the students academic advisor, course instructors, and, as appropriate, thesis supervisor. The current advisor is Robert Harper. Content for this concentration will be reviewed yearly by the Principles of Programming faculty in the Computer Science Department.

Students interested in pursuing this concentration should contact Robert Harper for an initial advising consultation.