Teaching Statement

Oliver Bračevac

Fall 2022

CS is one of the most popular fields of study today. More and more students are enrolling in CS programs every year, and the trend shows no signs of slowing down. I believe that it is equally important to teach both practical (problem solving, programming, and software engineering) and theoretical aspects (algorithms, formal methods, and proofs). My teaching philosophy is to always ground the learning objectives of the course to practically relevant and timely topics, and to use methods that ensure hands-on engagement with the material, especially when it comes to theoretical topics.

Teaching Experience

Instructor for Compilers: Principles and Practice (CS352) In collaboration with Tiark Rompf, I have been co-instructing the undergraduate upper-division course on compilers at Purdue (ca. 80 students). My responsibilities include preparing and holding lectures (50%), designing homeworks and exams, and managing teaching assistants.

We created the current version of the course to address the lack of modernity in most compiler textbooks and teaching methods. For example, traditional classes often dedicate a large amount of time to different parsing algorithms and register allocation methods, while neglecting topics that are more practically relevant and interesting. Our course takes a different approach: we parse simple arithmetic expressions and emit x86 assembly already in the first lecture. We spend the rest of the semester adding features, intermediate languages, and optimizations to this foundation, while always having a working system that students can run. This design is inspired by Ghuloum's "An Incremental Approach to Compiler Construction", scaled up to a much more complete language (a significant subset of Scala) and a more powerful compiler that includes features such as type checking and inference, CPS conversion, and a set of sophisticated optimizations. Student feedback has been positive: while the material is challenging and dense, they find the hands-on focus very rewarding.

Teaching Assistant for Concepts of Programming Languages This course (ca. 80 students) by Mira Mezini at TU Darmstadt teaches senior undergrad and graduate students about the design and implementation of modern programming languages in a language-agnostic fashion. My responsibilities as a teaching assistant consisted of conducting the in-person lab sessions accompanying the lectures, and designing the exercises, homework, and exams. In subsequent editions, I also supervised junior TAs.

The course broadly covers evaluation models, semantics, and implementation techniques of language features sampled from a variety of programming languages (e.g., C, Python, Java, Racket, Haskell, Prolog), as well as type systems and domain-specific languages (DSLs). We approach the theoretical material in a hands-on, implementation-focused manner, i.e., students study language features (e.g., higher-order functions, mutable state, continuations, etc.) by implementing their respective definitional interpreters and type checkers. Students acquire the conceptual tools to understand and conduct programming languages research, and may optionally complement our course with a seminar and semester project to deepen their knowledge and get involved in cutting-edge research projects. We have consistently received positive feedback about the course, seminar and project, which has proven to be a good incubator for bachelor/master theses as well as new PhD students at Mezini's group (myself included).

Team Supervisor for the Software Engineering Project (SEP) I also served as a supervisor for teams of 5-8 students (senior undergrad and graduate) in the SEP at TU Darmstadt, providing support and knowledge about (agile) software development methodologies and project management drawn from my own industry experience. The teams work with industry partners (from small local startups to large corporations, including Daimler AG, Bosch, Deutsche Bahn, and IBM) over the course of two semesters, and learn to plan, manage and implement a professional real-world software project in a self-driven and self-reliant manner. The SEP has been in existence for over 20 years and is renowned among both students and industry partners. Students gain invaluable soft skills and work experiences, and they are often employed by the companies after the project's official end.

Research Mentoring I have supervised bachelor, master, and PhD students. At TU Darmstadt, Fabian Muscariello completed his bachelor thesis on "A Unifying Framework for Complex Event Processing" under my supervision. Also at TU Darmstadt, I supervised Matthias Krebs, (1) in his role as an undergrad research collaborator on our published work on incremental type checking [5], and (2) his master thesis on "Implementing Abstract Dependent Classes with SMT Solving". He is now a PhD student with Mira Mezini. At Purdue, I have mentored several PhD students, sometimes leading to published research outcomes. With Guannan Wei, I have worked on reachability types [2] and generative symbolic execution [4, 3]. With Anxhelo Xhebraj, I have worked on seamless stack allocation with second-class values [1].

Outlook

Introductory Courses I am prepared to teach courses on programming languages (PL), formal methods (FM), mathematically rigorous software engineering (SE), and discrete math. I think it is important to give undergraduate students early exposure and fluency in these topics as a necessary precondition to succeed and innovate, be it in academia or industry. To that end, I would like to emphasize concepts from functional programming in introductory programming courses, such as purity, and strong static typing. From my own experience as student, teacher, and practitioner, functional programming often encourages minimalism, which fortifies the skill to quickly identify the core problem and separate it from non-essential details.

Advanced Courses On top of introductory PL/FM/SE courses, I would like to teach advanced topics to senior undergraduate and graduate students, in the form of lectures, seminars, and labs. One avenue is exploring the deep connection between programming and theorem proving, by means of type theory and dependently-typed functional languages, e.g., Coq and Agda (again, catering to my hands-on philosophy to teaching theory). The course could be based on Pierce et al.'s "Software Foundations" series of books, or Wadler et al.'s "Programming Language Foundations in Agda", and could be complemented by covering automated verification techniques, such as SMT solving, model checking, etc. For more engineering-oriented topics, I intend to dive deep into compilers, metaprogramming, symbolic execution, software testing, and "big data" systems.

Publications

[1] Anxhelo Xhebraj, **Oliver Bračevac**, Guannan Wei, and Tiark Rompf. "What If We Don't Pop the Stack? The Return of 2nd-Class Values". In: ***ECOOP***. Vol. 222. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 15:1–15:29. URL: https://drops.dagstuhl.de/opus/volltexte/2022/16243/.

[2] Yuyan Bao, Guannan Wei, **Oliver Bračevac**, Yuxuan Jiang, Qiyang He, and Tiark Rompf. "Reachability Types: Tracking Aliasing and Separation in Higher-Order Functional Programs". In: *Proc. ACM Program. Lang.* 5.**OOPSLA** (2021), pp. 1–32. URL: https://doi.org/10.1145/3485516.

[3] Guannan Wei, Shangyin Tan, **Oliver Bračevac**, and Tiark Rompf. "LLSC: A Parallel Symbolic Execution Compiler for LLVM IR". In: *ESEC/SIGSOFT* ***FSE***. ACM, 2021, pp. 1495–1499. URL: https://doi.org/10.1145/3468264.3473108.

[4] Guannan Wei, **Oliver Bračevac**, Shangyin Tan, and Tiark Rompf. "Compiling Symbolic Execution with Staging and Algebraic Effects". In: *Proc. ACM Program. Lang.* 4.**OOPSLA** (2020), 164:1–164:33. URL: https://doi.org/10.1145/3428232.

[5] Sebastian Erdweg, **Oliver Bračevac**, Edlira Kuci, Matthias Krebs, and Mira Mezini. "A Co-Contextual Formulation of Type Rules and its Application to Incremental Type Checking". In: ***OOPSLA***. ACM, 2015, pp. 880–897. URL: https://doi.org/10.1145/2814270.2814277.