Teaching Statement

Tiark Rompf

January 1, 2020

Software Foundations. At Purdue, I have taught the graduate PL class (CS 565) several times, based on Pierce's Software Foundations book. This has generally been a pleasant experience, and I have become convinced that most theory-oriented classes would benefit from the use of proof assistants like Coq or Agda. There is a class of students who are talented coders but who never got the hang of mathematical reasoning. For these students, *seeing* proof terms represented and manipulated as data structures *in a programming language* is eye-opening, and suddenly they succeed not just in systems courses but also in theory. The positive effect of proof assistants as "personal TA for each student" is also hard to overestimate as classes get larger.

Compilers. In the last few years, my main teaching focus has been on the undergraduate upper-division (CS 352) and graduate (CS 502) compiler classes, which I spent significant time redesigning. I have become frustrated with most compiler textbooks and traditional ways of teaching compilers, which often appear stuck in the 1990s–in the extreme, spending five weeks on different parsing algorithms, five weeks on different ways to do register allocation, and essentially no time on practically relevant and *interesting* topics. In particular the front-to-back organization of traditional compiler classes is problematic, as students get to have a working compiler only at the end of the semester, if at all. In my new course, we parse simple arithmetic expressions and emit x86 assembly in the very first lecture, and we spend the rest of the semester adding features, intermediate languages, and optimizations, while always having an end-to-end runnable system. This design is somewhat inspired by Ghuloum's "An Incremental Approach to Compiler Construction", but 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. Overall this redesign has been a great success in my view. Student feedback reflects that the class, while more work-intensive, is also more interesting now, and students appreciate the rigor and amount of material covered.

Graduate Seminars. I had the opportunity to teach two graduate seminars (CS 590), one on Metaprogramming in 2014 and one on Deep Learning for Symbolic Reasoning in 2018. Both were successful in kick-starting some new research directions with motivated students. I find seminars also a valuable tool to teach general research skills, from scoping a project proposal along Heilmeier's Catechism, through setting realistic milestones, all the way to final deliverables such as code artifacts, experiments, project reports, and presentations. In my 2018 seminar I gamified the process a little: students had to act both as startup founders that pitched their research ideas, and also as venture capitalists, who had to invest a fictitious dollar amount in their peers' projects. A small component of the final grade depended on how well a student's portfolio did in the final IPO, corresponding to the evaluation by myself and the TA. This worked quite well in keeping students engaged and accountable at the same time.