Teaching Statement

Mukund Raghothaman

I consider teaching to be one of the most challenging and rewarding aspects of life in academia. My role as a teacher has two goals: *first*, to impart a firm grasp of technical concepts such as programming abstractions, invariants, and assertions, and *second*, to provide an intellectual framework for students to understand new challenges they will encounter in the rest of their careers. I will have fulfilled my duty if undergraduates who proceed to the industry as software engineers pause to deliberately think about program correctness; if graduate students develop good taste in choosing research problems and devise elegant and rigorous solutions to them; and if they all leave the classroom with a sense of having learned something exciting.

Courses on programming languages and formal verification are important in introducing students to think rigorously about program correctness. Another reason they are now relevant is that mechanical tools such as SAT, SMT and ILP solvers are becoming sufficiently robust and practical to solve scheduling and planning problems in diverse application areas. I look forward to exposing students, both at the undergraduate and at the graduate level, to the enormous utility of modern constraint solving technology.

Teaching experience. In graduate school, I have been a TA for two courses: an undergraduate course on the theory of computation, and Software Foundations, a graduate-level course on programming languages. My duties included conducting office hours, answering questions and helping in setting and grading assignments and exams. I also contributed material to the open-source Software Foundations textbook, including a chapter formalizing information flow security.

Future courses. I am excited to teach undergraduate courses on programming languages, compiler design, the theory of computation, and on discrete mathematics and logic for computer science. At the graduate level, I will design and teach courses on program synthesis, program analysis, formal verification, and programming language theory. There is exciting potential for cross-fertilizing ideas between the machine learning and programming language communities. Statistical techniques have been used to attack traditional problems in PL research, including software deobfuscation, learning typestate and API usage protocols, and my own research in interactive static analysis. In the other direction, ideas from the PL literature can guide the design of probabilistic programming languages, and inform contemporary challenges such as the quest for explainable AI. I will explore these inter-disciplinary topics in the courses that I design.

Teaching philosophy. Students are capable of far more than they think they are, and I think that it is my role as a teacher to help them achieve their full potential. This involves not only making courses challenging, but also providing students the support and resources they need to excel in them.

An important goal of this pedagogical support is to foster in students the curiosity and confidence to approach new material. It is only by discussing new ideas, theorems, tools, and techniques, and challenging peers on their misconceptions, that familiarity is achieved, and the foreignness of concepts eliminated. I am aware that years of learning about and doing research in the field make it difficult for me to judge the inherent difficulty of ideas when first encountered by students. I will therefore make a conscious effort to provide instructional scaffolding, break down concepts and solicit active, continuous feedback from students to ensure that my classroom style is effective.

Furthermore, creating an accessible classroom requires providing the resources that students from diverse backgrounds, learning abilities and prior experience need to succeed. A common refrain is that course material is either “not directly useful” or “too difficult”. By understanding the backgrounds and goals of my students, I

hope to successfully motivate my coursework and provide appropriate supplemental material and alternative paths so that every student makes tangible progress during their studies with me.

Another important challenge in teaching CS courses is scaling to the large hundred-student classrooms commonly seen in core courses. Within the classroom, I plan to implement interactive learning at scale by embracing recent developments such as video lectures, flipped classrooms, and anonymous response tools such as clickers.

Outside the classroom, providing meaningful feedback for projects, assignments, and exams is central to effective pedagogy. Mechanizing part of the feedback process with automatic grading tools has the potential to reduce some of this feedback load. I witnessed one fascinating approach to this problem in Software Foundations, the graduate programming languages course I attended and for which I was later a TA, where the main problem was the rigorous attention-to-detail demanded by proofs in programming language theory, and to which even the most mathematically-inclined students are usually unaccustomed. Because the entire course was formalized within the interactive theorem prover Coq, students could judge the correctness of their assignments entirely by themselves, by simply getting Coq to print "No more sub-goals". I was struck by how this setup encouraged experimentation and innovative solutions to assignments. Of course, this approach would be more unforgiving than is desirable for most other classes, especially at the undergraduate level. I will also explore various social mechanisms to provide meaningful feedback, including classroom discussions forums and peer-assessment.

Research advising. I look forward to mentoring students as they begin their research careers. As a postdoc, I have advised three graduate students: Manos Koukoutos at EPFL, and Sulekha Kulkarni and Xujie Si at Penn. I found it very gratifying to witness their evolution from being consumers of knowledge to taking ownership of concepts, synthesizing their own ideas, and eventually co-authoring research papers on their chosen topics.

I am also presently mentoring a group of four undergraduate students who, as part of their Senior Design Project, are integrating our group's recent research on continuous program reasoning into the GitHub platform. This is providing them with an opportunity to apply many of the principles they have learned over their years in college and also gives them a first-hand experience of the process of research.

As a mentor, I would emphasize the importance of good problem phrasing, healthy irreverence for established dogma, and rigorous evaluation of proposed solutions. As I look back at my own career in graduate school, I appreciate the freedom that my advisor afforded me through the years, only actively stepping in when I seemed to be stuck on unproductive approaches. I hope to follow a similar style, where my role as an advisor is to perform periodic course-correction and introduce the student to related problems and potential solution strategies while leaving them fully in charge of their research projects.