Teaching Statement

Mukund Raghothaman

I consider teaching to be one of the most rewarding and challenging aspects of life in academia. At the lowest level, I expect my students to leave with knowledge of concrete concepts such as programming abstractions, invariants, and assertions. At a higher level, success as a teacher will only be achieved if I affect the way they think: if students who proceed to careers as software engineers pause to deliberately think about program correctness, undergraduate students continue to graduate school to do research, graduate students come up with elegant and rigorous solutions to research problems, and go on to develop good taste in choosing their own research problems, and if each student leaves the classroom with a sense of having learned something exciting.

Courses on programming languages and formal verification are important in challenging students to think about program correctness. Another reason they are now relevant is that mechanical tools such as ILP, SAT, and SMT solvers are achieving industrial robustness and becoming applicable in diverse areas such as software package managers and in solving scheduling and planning problems. I look forward to exposing students, both at the undergraduate and at the graduate level, to the enormous utility of modern constraint solvers.

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 Software Foundations textbook, primarily through a formalization of information flow security.

Future courses. I am excited to teach undergraduate courses on programming languages, compiler design, the theory of computation, and 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 intend to explore these inter-disciplinary topics in the courses that I design.

Teaching philosophy. My goal will be to make courses challenging while still remaining approachable. 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. An important part of this is to foster in them 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. Therefore, I will 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.

One of the main challenges in computer science education is the large hundred-student classrooms commonly seen in core undergraduate 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 encouraged experimentation and innovative solutions to assignments. Of course, this extreme 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 two graduate students: Manos Koukoutos at EPFL, and Sulekha Kulkarni and Xujie Si at Penn. By jointly authoring papers with them, it was immensely rewarding to witness their evolution from being consumers of knowledge to taking ownership of concepts and synthesizing their own ideas. Especially for undergraduate students, I view research as a way to deepen one’s knowledge in a self-directed manner. As a mentor, I would emphasize the importance of good problem phrasing, healthy irreverence for established dogma, and the 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.