Swarat Chaudhuri

Teaching statement

The first reason why I want to teach is because I love to listen and talk. The second reason is that I love challenges. It is a challenge to convey, within the short span of a lecture, the essence of a proof technique, algorithm, or programming construct. How does one address the different needs of different students? The best way to explain an algorithm to a good hacker may be to give examples using concrete code, while a student more interested in logic and mathematics may want insights into the algorithm's inductive proof. An explanation necessary for a weak student may be boring to a clever one. How does one condition students to think formally without losing the forest for the trees? These questions are, to me, as exciting as any hard research problem.

There is also the fact that a teacher learns from a class as much as the students. Even the greenest student can unknowingly ask profound research questions, thinking often happens while communicating, and one can never know the basics too well. I also have a couple of personal axes to grind: as a researcher interested in software engineering and programming languages, I have a stake in convincing the next generation of programmers to write code systematically and use tools that reduce programming errors. As someone interested in foundational research, it is important to me to dispel the myth that theoretical computer science is "useless."

How would I go about teaching? The role of a teacher, I believe, is to impart intuitions and critical thinking skills rather than a list of "howto"-s. Whichever class I teach, I would be far happier to leave my students with a big picture and a set of critical thinking skills than a load of details, and I will test them keeping this objective in mind. I would like to evoke in them an awe of the harmony and unity pervading computer science. A finite-state machine is a pattern-matcher as well as an abstraction of a program or system; it is a computer with a fixed-size memory as well as a circuit with latches as well as a logical formula as well as a certain kind of monoid. I would like to convince them that mastering these different views gives them a set of transferable skills that are useful in solving different types of problems. I would like to connect problems of contemporary interest, likely to be appealing to them, to general principles in computer science. I would like to give them a sense of the history of computer science and its connections to the grand tradition of liberal education. To my graduate students, I would like to convey the techniques and insights I have found useful in my own research, and the excitement I find in it.

Like in much else of life, joy and empathy, I feel, are necessary for success in this context. I would like my enjoyment of computer science to be transparent to my students. I would like to be empathetic, because the best teachers I have had were able to "get into my head" and guide my thinking, rather than force their own mental models on me. I would like to know why the eager students are eager, why the bored students are bored. In advanced classes and one-on-one interactions, I would like to be as much of a listener and a moderator as a talker. In larger undergraduate classes, I would like to address different ways of looking at a concept, each of which is likely to appeal to a different set of students, and their different applications. I would also want to encourage tinkering: what would be lost if I omitted a certain step in an algorithm? Why is the order of two operations in a protocol crucial? These excursions will have to be carefully planned, as I would also like to finish my course curriculum. However, I will use them as starting points for class research projects. I recognize that I will need to master several balancing acts—between encouraging imaginative leaps and mathematical rigor, exploration and efficiency, spontaneity and discipline—and will try actively to learn from experience.

At the University of Pennsylvania, I have been a teaching assistant in three courses on formal language and automata theory, at the graduate as well as the undergraduate level. While I did not give lectures in these classes, I held many office hours with students. Such "longitudinal" experience with the same material at different levels has given me an appreciation of teaching and advising students with different needs. I have also given guest lectures in Professor Rajeev Alur's course on logic in computer science. This was a small class and gave me some experience on how to have free-form discussions in a class while covering a planned amount of material. I have also organized two reading seminars, which have given me experience that will hopefully be useful in seminar courses.

In the future, I would be happy to teach undergraduate courses in introductory programming, programming languages, data structures and algorithms, theory of computing, logic in computer science, operating systems, software engineering, and concurrent programming. I would be delighted to teach courses on the history of computing, puzzles and paradoxes in logic, and the connections between computing and the social sciences. I would want to teach graduate courses in areas close to my specialization—for example, theory of programming languages, program analysis and abstract interpretation, concurrency, formal verification (model-checking and theorem proving), formal language theory and its connections with games and logic, and software reliability. I would also like to offer specialized seminars that would, for instance, survey program analysis tools, brainstorm on current topics in program analysis, or read the classic papers of computer science in their original form. I look forward to working in an academic environment that would provide such opportunities.