Teaching statement

Tej Chajed

December 2021

I'm excited by the opportunity to teach and advise students, which are both important aspects of being a professor and activities that I enjoy doing.

1 Teaching approach

During my PhD I actively pursued opportunities to teach, including lecturing and curriculum design, both within and outside the classroom. My teaching philosophy arises from my experience teaching in three different settings. I was actively involved in the development and teaching of 6.826 (Principles of Computer Systems).[1] As a Fellow in the EECS Communication Lab I've held over a hundred one-on-one coaching appointments for technical writing tasks, as well as run other projects to help peers with communication. Finally, my research builds upon the Iris concurrency framework and I am enthusiastic about helping teach Iris.

Backward design. When preparing to teach anything, I start by asking, what should the students be able to do after this lecture, assignment, or activity? What are the 2–3 key messages that I would like them to remember? Starting with these goals in mind helps focus teaching on specific goals without inadvertently covering too much material or putting emphasis in the wrong place.

One of my first opportunities to design a curriculum was developing the programming assignments for 6.826, a new class on systems verification. We wanted students to learn fundamental principles rather than the mechanics of the verification infrastructure we were using. As a result, we had students reason about storage systems with relatively little code but intricate reasoning, avoiding a trap of students completing proofs by "brute force" without even understanding the theorem statements. The same objective informed my teaching style in class recitations and office hours. When students get stuck, I get them to articulate their understanding of the problem first before diving into the code. Once they understand what they want to do, I am happy to help more directly with the minutiae of translating intuition to something the proof assistant will accept.

Experience in the Comm Lab more formally introduced me to backward design, which is embodied in both the communication training Fellows receive and in how we approach creating workshops and lectures. I was able to practice the process in creating a lecture for a job search seminar, in a TA training on teaching a class recitation, and in a training for mentors giving feedback on prospective students' grad-school applications. These teaching opportunities have gradually become more challenging — most recently I am co-developing a paper-writing workshop which has required careful thought about what objectives would be suitable in the first place.

Backward design has informed my teaching outside of the classroom as well. Iris is the foundation for my research so I have spent time on teaching Iris to the broader programming languages community. There are many audiences for this kind of teaching, so I've worked on a few very different resources. The first was a blog post and subsequent lecture in 6.826,[2] which was designed to focus on some key ideas in concurrency reasoning that Iris implements. Next, I co-taught the Iris tutorial at POPL 2021,[3] with a more ambitious goal of helping the audience understand Iris paper or use it themselves. Finally, in response to an audience question I developed a separate tutorial on an advanced feature of Iris.[4]

Interactivity. I believe in teaching through practical and hands-on experience. Students learn best, and retain what they have learned, by doing. There is no substitute for getting students to spend time engaging with the material, and hands-on work both demands this kind of engagement and makes it more enjoyable. As a result much of the teaching I do involves code and programming, with well-developed exercises when possible.

In lectures, I try to have some interactive compo-

[1] https://6826.csail.mit.edu/2020/index.html

[2] https://youtu.be/59sPvu9RGy8

[3] https://gitlab.mpi-sws.org/iris/tutorial-popl21

[4] https://github.com/tchajed/iris-simp-lang

nent or demo to break up the lecture. Asking students to discuss a question gets them to think about their answers and gives an opportunity for them to explain to their peers, improving everyone's learning. I gave three lectures in 6.826 that involved walking through a short demo involving code that illustrated the subject of the lecture; before seeing the code I would go through the same ideas on the board to make sure they know what concepts we are emphasizing. Demos are both fun and if done well create an educational resource for the students to examine later.

2 Advising

My main approach to advising is to build students' confidence, focus on learning, and continuously adapt to their current needs. I've developed this approach through experience: I had the privilege to directly mentor six undergraduate and master's students, including helping advise two master's theses. I've also been an informal mentor for many younger grad students in my research group.

Sometimes being a good advisor means putting the student's learning and goals ahead of getting research results. I ask students to propose their own solutions first rather than sharing my ideas so that they have a chance to think things through at their own pace and the satisfaction of solving a problem on their own. To maintain an understanding of the student's code, I like to take on any "boring" work that we both agree will not help the student learn, but which is necessary for them to make progress.

In my field of systems verification, students need experience programming, both systems development and developing machine-checked proofs (which is a similar skill in many ways). I have found it useful to get students started writing proof as quickly as possible, starting with small examples or adapting an existing example. Few students start with verification experience, and most students come in without deep mathematical training; all types of students much better understand verification from working hands-on with proofs in a proof assistant like Coq or a verification language like Dafny.

The ability to select a good problem is a goal of the whole PhD process that needs to be cultivated. At the same time, students require different approaches to advising to succeed, and I plan to adapt my style to each student's needs and stage in their PhD. For example, a common pattern is that students benefit from starting with more concrete, low-risk projects initially; they learn basic research skills and build confidence. Eventually they are able to identify problems they care about on their own, and chart out a plan for the work.

I strongly believe in the importance of feedback, both giving feedback regularly to students and getting feedback from them on how things are going. For each student, I will hold a review each semester and go over their accomplishments and what they would like to focus on going forward — feedback should also include what is going well, to maintain motivation. These reviews are also a chance to assess long-term career goals, internships, and extra-curricular activities. On the flip side, I will actively solicit feedback from students about my advising and anything I could do to improve their experience, learning, and researh.

3 Curriculum ideas

I would be excited to incorporate systems verification into the curriculum. This could start with a standalone seminar on systems verification, with a focus on recent research — my own work has involved several verification tools, so I can introduce students to a breadth of approaches to verification. Verification would inform my perspective on teaching other classes, such getting students to think about specifications in databases, security, operating systems, and networking classes. I'm also prepared to teach any undergrad PL course, including Software Foundations.