Teaching Statement

Anders Miltner

My fundamental goal in teaching is to provide an inclusive classroom that sets all students up for long-term success. My primary approaches to achieving this goal lie in (1) promoting interest in the lesson material by connecting lesson plans to real-world problems, (2) providing cognitive support for self-learning through large number of worked-out examples, and (3) providing individualized attention for struggling students.

Real-world connections help learning across the board, but are particularly useful for underrepresented minorities. Researchers speculate this disproportionately positive impact is due to exposure. Underrepresented minorities are typically less exposed to concepts in computer science before taking classes, so while real-world connections may be implicitly understood by non-URM students, those implicit connections may not be present for others. By explicitly including real-world impact as part of lesson plans, all students can get excited about what they are learning, not just those who already know why it's important.

A myth in computer science education is that students learn best when they have "discovered" solutions and approaches themselves. However, cognitive load theory has shown this is not the case. By requiring students to learn through discovery, there is greater load on the students. This really is a place where less is more – a student who sees 10 worked out problems and does 10 problems themselves will have greater understanding than the student who does 20 problems themselves. A core tenet of my teaching philosophy is then to provide a large number of worked out examples for students. By having this cognitive support immediately accessible, students can learn concepts in a number of ways: they can learn it from the general theories and algorithms, and they can induce these general principles from examples.

Lessons Learned from Prison Teaching I volunteered during my time in graduate school with Princeton Prison Teaching Initiative (PTI). In this program, I taught math courses and developed a computer science curriculum for incarcerated students. While not everything I learned from PTI is generalizable to college courses outside prison, I came away with many lessons about good course instruction during this experience.

For example, individualized attention is a key component for struggling students, and also I believe the hardest of my approaches to realize. The reality of the world is that there is a fixed amount of time, and providing individualized attention to all students is quite difficult. To provide individualized attention, one must ensure that their general materials are sufficient to ensure that only students struggling at learning specific concepts need individualized attention. But even this is not always enough. During PTI, I had some students where the amount of individualized attention they required negatively impacted the learning of other students. My pod and I made the difficult decision to ask them to drop down to the class they had technically tested out of. Sometimes these difficult decisions are necessary, but I believe they help both the struggling student (who wouldn't have deeply understood the material if he had stayed in) and the general class.

Teaching Interests I am of course willing to fill in and teach any class in the department. Some of the classes I think I would most enjoy teaching include: functional programming, introduction to proofs, discrete math, compilers, and artificial intelligence. I particularly enjoy "mathy" courses – these classes are originally what inspired me to go into research, and I enjoy seeing that inspiration take hold in others.

I also think these courses are overall quite challenging to teach well. For students like me, who find types of formal reasoning and proofs exciting, the classes teach themselves. But I enjoy figuring out ways of inspiring (or at least keeping the attention of) people who don't immediately see the benefit of these types of courses. However, I strongly believe that students who get a background in these types of courses become better computer scientists and even better programmers.

Example Mini-syllabus for Functional Programming Functional programming is a course that often is not taught from principles I agree with. I have seen people use nebulous terms like "beauty" and "simplicity" as justification for teaching functional programming. While I agree that there is a beauty and simplicity to functional programming, I don't believe they are useful principles to serve as a foundation for a class.

Rather, I believe the primary theme in a functional programming class should be around *reasoning*. A fundamental benefit to functional programming lies in the simplicity of its semantics. Because functional programs are immutable, one does not have to talk about memory, pointers, or heaps to describe the behavior of functional programs. This is a theme that I would continually tie back to throughout the course.

The core learning objectives for this course would be for students to:

1. Confidently use and create higher-order functions
2. Confidently use and create data types
3. Confidently use and create modules and functors
4. Understand simple operational semantics
5. Understand simple typing rules
6. Be able to understand the pros and cons of language features
7. Be able to write a simple interpreter
8. Be able to add simple language features of their own design

Initially, I would introduce the class to a toy language for simple arithmetic expressions. This language would serve as a basis for providing students with a simple, initial understanding of formally defined languages.

After this, I would introduce various language features, like higher-order functions, types and modules, and statefulness. These features would each be introduced as a means to solve a specific problem. For example, higher order functions help address code reuse. Statefulness helps address performance issues. Types and typing rules help automate reasoning about code. While learning these concepts, the students homeworks would be a combination of (1) coding using these features, (2) proving theorems about individual programs, and (3) proving theorems about the language as a whole.

Undergraduate Research In addition to classes, I believe it very important to provide students the opportunity to perform research at the undergraduate level. At UT Austin, I helped Ana Brendel as she began a career in research. When Ana began working with me and my collegues, she had no experience in research or programming languages as a whole, though she did have a mathematical background. During her first year of research, I got her up to speed by giving her a basic introduction to some key concepts. With these key concepts, she was able to work on a self-contained subproblem of the work, and follow along with the broad research meetings. After having gotten this initial experience, she is now working on novel, independent work, related to the initial publication.

I very much enjoyed this experience. Watching her grow as a researcher has been fulfilling, and I intend to continue including undergraduates who are interested in research on my projects, and I will likely do so in a similar capacity – (1) get them up to speed with a basic understanding of the required concepts, (2) have them work on a self-contained subproblem while attending broader research meetings and (3) have them work on project of their own.