My goal as a teacher is to help students not just learn how to solve a problem on their own but also to help them understand how to find or create an interesting problem, how to figure out if a problem can be solved, and finally how to solve a new problem. I believe in learning by example, but I do not believe in learning only by example. I think it is more important to know why we are using a technique than to know how to use that technique. We can drown students in examples, and we can make them masters of different methods by showing them how to solve lots of examples. However, every time a new problem comes up, we are likely to need a teacher. I believe in theory as a way to answer the why question. For example, I would like my students to know why just knowing the sign of $b^2 - 4ac$ is enough to find the number of real solutions in the quadratic equation $ax^2 + bx + c = 0$. If they know this, they might have a chance to find a similar method for finding the number of zeros in the cubic equation $ax^3 + bx^2 + cx + d = 0$. Otherwise, just knowing the quadratic method and solving hundreds of examples for it, won’t help. Admittedly, understanding and answering the why question often requires more maturity in the field, but I believe the ideology should be there.

I believe having an interactive environment is the most fundamental tool in a course, regardless of being an undergraduate or a graduate one. The more interactive the course, the easier it is for students to learn the material. To give a concrete example, when I was a lecturer at the University of Pennsylvania, I used a tool called AutomataTutor for a small part of the course. The tool provides immediate and personalized feedback to students, so they did not have to wait for office hours, nor they have to type in their questions on a forum and wait for others to answer. Everybody found the tool very helpful in understanding the related parts of the course. Another interesting approach to learning is hands-on tools. Not only they are the best way of connecting theory to practice, but they are also instrumental in interdisciplinary courses, like robotics and cyber-physical systems, that often require a solid mathematical background as well as knowledge in control, programming, and sometimes even physics. For example, analyzing stochastic systems involves a few fundamental algorithms that a related computer science course usually covers. However, any practical implementation of these algorithms requires the ability to parse, load, and manipulate systems with millions of states, none of which is covered in any depth at such a course, but they have all been implemented before and are available for free. Another method that facilitates the learning process is collaboration among students. However, not every student feels comfortable collaborating with others. To remedy this problem, I made it possible for them to remain anonymous when they post a question online or answer one. Next in the list is establishing a relation to what students have learned before. That is the main reason that for someone who already knows a few programming languages, learning a new one is usually like reading a novel. Finally, for an undergraduate course, looking at problems as games makes the learning process more joyful. For example, in a computer science course like Introduction to Theory of Computation, many problems can be thought about as games between the student and an imaginary opponent. The student designs a finite number of steps to achieve a goal, and the opponent tries to defeat the student by illustrating flaws in their strategy.

Teaching Experience

I was fortunate enough to be a lecturer in the Computer and Information Science department at the University of Pennsylvania during the time I was a postdoctoral scholar there. I taught Automata, Computability, and Complexity (CIS 262) for one semester to a class of roughly 100 students with six teaching assistants. It gave me a first-hand experience in the teaching process. Besides what I have already mentioned about this class, to encourage collaboration among students, right from the beginning, I announced that for their homework, they could work with any other fellow student they like, as long as everybody writes their solutions and acknowledge any collaboration. I also used Piazza to encourage everyone to participate in the Q&A part of the course. By the end of the semester, 562 posts have been created, 1829 contributions have been made, and 16 minutes was the average response time. Also, on average, every student looked at each post more than twice, and 80% of students have at least one contribution (this includes about 20 students who dropped the course sometime during the semester). Later, I realized to make these numbers even better, I can use an incentive policy like whoever receives an endorsement from a teaching assistant or instructor on Piazza, will receive extra credit. I have also learned from my mistakes. For example, after a few weeks and conversations I had with many students, I found two of them who find homework too challenging. To help them overcome their challenge, I sat with them a few times and without giving up any solution, let them know how they should approach the problems and why some of their ideas are incorrect (I became an interactive tutor for them). Later in the semester, I realized
that this was most likely unfair to some of the other students. A better approach would have been to look at the stats first, and then offer the same service to the bottom 10% of the class. I also realized that even though I love the materials in this course and cherished every minute of it, I need to appear more energetic during a class.

I have also been teaching assistant for one semester in Data Structures and Programming Principles at the University of Illinois, Urbana-Champaign, instructor for one semester in Software Engineering Laboratory at Sharif University of Technology, teaching assistant for four semesters in Data Structures and Algorithms, tutor for one semester also in Data Structures and Algorithms, and teaching assistant for one semester in Software Engineering, all at Amirkabir University of Technology. These classes were in a variety of sizes and styles. The one at UIUC had about 400 students, we were 7 TAs, and about 20 undergraduate students were helping us. The data structure course at AUT had between 60 to 100 students, and there were two TAs. Finally, I was the only TA or instructor in my other teaching experiences, and there where at most ten students involved. My responsibilities in data structure courses included reviewing the content of the course and helping my student where she had difficulties, designing programming problems as well as test cases for those problems, grading homework, exams, and programming problems, and running office hours, labs, as well as Q&A sessions. For the software engineering course, I was helping six students who had a project in a company that I was working part-time at that semester, to have a better understanding of the work process there. Finally, for the software engineering lab, I gave students different documentation of a project that I used to be involved in and helped them to gradually create various unified modeling language diagrams such as use case, activity, sequence, and class diagrams for that project. Looking at different people’s work helped me to better recognize corner cases in which students are more likely to fail. I believe it even led me to become a functional and performance software tester while I was working in the industry. And finally, looking closely at how students think, helped me in being more effective in labs and office hours during the time I was a teaching assistant at UIUC, where students are often stuck in the computer programming problems, and we were expected to spot the issues as quickly as possible.

Courses I Can Teach

Core Courses. Most of my training has been in the CS, and I am ready to teach a variety of undergraduate or graduate courses in the areas of formal verification, programming languages, and software engineering. Courses related to these areas include but are not limited to, An Introduction to Theory of Computation, Program Verification, Data Structure and Software Principles, Formal Software Development Methods, Introduction to Algorithms, Probability and Statistics for Computer Science, and Programming Language Design. With some additional preparation, I would also be able to teach courses like Control Theory, Introduction to Artificial Intelligence, Introduction to Software Testing, and Stochastic Processes and Their Applications.

Formal Verification of Cyber-Physical Systems. I would love to design this special topic course at the graduate level based on my research. A few universities like UIUC, CMU, UPenn, and CU Boulder have realized the need for such an interdisciplinary topic and are already offering it to their graduate students. In this course, students will learn what a cyber-physical system is, why formal verification of these systems is crucial, what are the most critical properties one may want to check about a cyber-physical system, and how we use mathematics to model a cyber-physical system and its properties of interest formally. The rest of the course mostly focuses on different classes of models and their applications, like real-time scheduling, air traffic management, self-driving cars, and medical devices. These classes include those for which we have a solution and those that are undecidable. However, undecidability won’t stop us from attacking systems with more general dynamics. Students will learn about different approaches like automatic abstraction refinement techniques that are primarily designed to handle complex systems. They will also be exposed to various state-of-the-art tools and will use small projects and homework to gain hands-on experience with those tools and find where the strengths and weaknesses of each tool lie.