Teaching Statement

Di Wang

The opportunity to teach and work with undergraduate and graduate students is one important reason for me to choose an academic career. In my opinion, teaching not only provides the crucial means to communicate and share knowledge with the future generation, but also enjoys the unique advantage of generating novel research ideas.

My research experiences in *the analysis and design of software systems that incorporate randomness* make me qualified to teach courses on *programming languages*, *formal methods*, and *probabilistic programming*. They provide a starting point for interesting graduate-level projects in various and possibly interdisciplinary research areas. My training at CMU as a Ph.D. student has prepared me well for both teaching and mentoring. *I have contributed as a teaching assistant for two courses (one undergraduate-level and the other graduate-level), and as a guest lecturer for an advanced graduate course.* As a teaching assistant, my duties included holding office hours, and grading assignments and exams. In a course about quantitative program analysis, I delivered several guest lectures about cutting-edge research. Moreover, *I have mentored several undergraduate students during their research projects.* I look forward to gaining more experience in teaching and mentoring, as well as sharing knowledge with the students.

Besides teaching and mentoring, I find it enjoyable to develop tools for aiding education in computer science. In computer science, students usually interact with their machines to learn, so it is important to have properly designed tools and interfaces for the education. *I have developed two tools that help students write code and study logic, respectively.* I will keep passionate about creating tools that aid education in computer science in the future.

Teaching Interests

I would enjoy teaching courses in different areas, especially those about programming languages, formal methods, and probabilistic programming. At the graduate level, I would like to teach program semantics, type systems, static analysis, quantitative verification, and more broadly programming randomness in software systems and statistical probabilistic programming. Rather than demonstrating specific technologies, I will present a comprehensive literature review to help students understand the problem formulation, and focus on developing both the practice and theory of those technologies. I believe that it is equally important to help students understand *how* and *why*.

At the undergraduate level, I would be excited to teach programming in different paradigms, compilation, and program logics. Rather than focusing on specific languages or tools, I will use them to illustrate the essence that can guide the students to easily generalize the ideas to other languages and tools. In addition, I feel comfortable teaching classes on theory, mathematical logic, and discrete math. Once I have gained more teaching experience, I would also like to teach more freshman courses such as *introduction to programming*, and *functional and imperative languages*.

I would also like to teach algorithms and data structures for *competitive programming*, e.g., International Collegiate Programming Contest (ICPC). I have been participating in competitive programming since junior high school. I have also taught high-school students competitive programming. From the experience, I understand how hard it is to start with computer science, so I developed some techniques to help beginners study algorithms, such as starting with examples to demonstrate the insights, dividing problem into subparts, and setting up invariants for subparts to compose the solutions to them. I would also like to work as a coach to help train the students on the ICPC team.

Teaching Philosophy and Experience

Being a teaching assistant for undergraduate and graduate courses, as well as working as a guest lecturer, I learned that *it is crucial for an instructor to keep students motivated on the topic, guide students to explore on their own, and encourage interdisciplinarity.* Moreover, I realized that *teaching is not only about guidance and instruction, but also about collaboration and mutual enlightenment, which can indeed benefit research.*

The undergraduate course *Bug Catching: Automated Program Verification* covers developing program logics to prove program correctness and using verification tools to write verified code. When I held office hours, I enjoyed guiding students to verify the correctness of their code, by encouraging them to use theory to guide practice. For

example, verifying code involves annotating the code with loop invariants, and the program logic indicates that when a loop mutates more variables, the invariants would also become more complex, which makes verification more difficult. I guided the students to understand such implications of the theory, and they found out (on their own) how to aid the verification tool via rewriting a loop into smaller pieces. Students had great fun in this and stayed motivated in learning program verification; two of them became the teaching assistants for the course in the succeeding year.

The graduate course *Programming Language Semantics* is about fundamental concepts and mathematical techniques of programming language semantics. The course is a *breadth* course: it is meant to be accessible to all computer science graduate students, and many Ph.D. students from areas other than programming languages would take this course. The most exciting part for me was to *discuss with students from other areas about interdisciplinary research ideas*. For example, a discussion with a student who focuses on machine learning inspired me to think about the formal semantics of statistical probabilistic programs, and led to a research project later.

In the graduate course *Foundations of Quantitative Program Analysis*, I delivered three lectures on the cutting-edge research on resource analysis for probabilistic programs, which is an interdisciplinary topic. I motivated the students with interesting problems (e.g., average-case complexity) and simple examples (e.g., Gambler's Ruin). When I developed the core type system for analyzing the expected cost, I first reviewed the non-probabilistic counterpart, and then guided the students to explore how to patch the existing system for probabilistic analysis. I also demonstrated some incorrect patches, and discussed with students why they were incorrect. In addition, collaborating with one of the students, I even published a paper about resource analysis for probabilistic programs. The success proved that it is beneficial to combine teaching and research, and I will keep this in mind for my future teaching.

Mentoring Philosophy and Experience

My research on the analysis and design of software systems that incorporate randomness provides a basis for advising graduate students who work in different areas such as static analysis, type systems, probabilistic programming, analysis of uncertainty in systems, and design of programming languages. I am also happy to collaborate with colleagues from other research areas to advise students. My goal for advising students is to *help them determine and succeed in their desired career*. I will encourage their own ideas and help them develop a skillset including research, communication, writing, and presentation. I will try my best to help them overcome difficulties, such as lack of ideas for their first project or temporary failure from a rejected paper submission.

I have mentored three undergraduate students for different research projects, which are related to my research. One of them was interested in static analysis for probabilistic programs. We discussed potential research directions and exchanged our ideas, then decided to develop a static analyzer for finite-state probabilistic programs. He proposed to analyze the hardness to obtain an acceptable sample from a probabilistic program. I provided him a comprehensive list of learning materials and related work. We had several follow-up meetings to discuss the problem formulation and possible solutions. I suggested that he try linear programming, but also pointed out other possiblities. He devised a better algorithm to analyze unbounded loops and wrote a paper about his findings. During the research, I successfully motivated him to carry out research in programming languages, and eventually he got admitted to MIT's Ph.D. program.

Tool Development for Education

Besides teaching and mentoring, I also enjoy creating tools and interfaces to aid computer-science education. An undergraduate course at CMU uses Standard ML (SML) to teach functional programming, but there were no handy editors for SML, so I created a plugin for Visual Studio Code that supports precise syntax highlighting and formatting. The plugin is popular at CMU and other universities. Another tool I created is a web-based prover for an undergraduate course which teaches constructive logic. The course is already equipped with a theorem prover, but its logic is fundamentally different from the logic used in the course, and it provides automation functionalities, which may lead the students to rely on automation rather than learn by picking proof rules. I implemented the proof rules for constructive logic and disabled some features that are not suitable for education purpose. The web-based prover helped students to learn the logic in an interactive way, and eased the burden of checking students' assignments.