Teaching Statement – Gordon Stewart

Teaching and scholarship are inextricably linked. As researchers, the scholarly activities we perform to communicate our work, such as writing and publishing papers, attending conferences, discussing new ideas with colleagues, and giving talks, are all just teaching in disguise—to a wider audience of one’s peers. At the same time, the most engaging and enlightening teaching, for both graduate and undergraduate students, is that which enlivens the lecture hall with the new ideas and excitement that breathe life into research. Teaching to a class—or even just explaining a new idea to a colleague on a whiteboard—can also be an extremely effective method of clarifying one’s own thoughts, which can lead to refinements of research ideas, or even to new research.

I have twice taught recitation sections of introductory computer science, once taught recitations of programming languages theory, and once served as a project advisor in the junior-level software engineering course. My approach in the classroom incorporates research, and my enthusiasm for exploring new ideas in computer science, in a variety of ways. In introductory computer science, I taught recitation sections of approximately 20 students that reinforced material taught in lecture through hands-on exercises, coding examples, and group work. This course—which teaches students the basics of programming in Java, then introduces fundamental algorithms, data structures, and the basics of theoretical computer science—can sometimes be dry for incoming first-years. My precepts kept things fresh by engaging students with exercises that stretched their knowledge and capacity for critical thinking. For example, when doing in-class reviews of programming assignments—which included N-body simulations, Markov models of natural language, and simulations of the plucked string of a guitar—I often had the class step through difficult parts of the code on the blackboard by thinking rigorously about how the state changed at each program step, much as program analyzers “execute” code symbolically. Exercises such as these helped students to develop the “mental debugging” skills that are so useful when programming.

At other levels of the undergraduate and graduate computer science curriculum, I was able to incorporate new ideas and new research more directly into teaching. While serving as a teaching assistant for David Walker’s course in programming languages theory last spring, I developed programming assignments that engaged students by drawing on interesting new research. One of these assignments, which asked students to implement a regular expression engine, eschewed the standard conversion to NFAs—which many of the junior and senior students had already seen in earlier courses—in favor of a less well known but equally interesting technique involving regular expression “derivatives.” The assignment was developed from a recent journal paper that the students were also encouraged to read. In another assignment, which asked students to implement a mini-SAT solver with conflict-driven clause learning (CDCL), I developed the specification of the assignment following a mathematical reformulation of CDCL published just last year at a major conference in programming languages. The resulting assignment encouraged students to think rigorously about the problem in a way that was new and rewarding.

As a faculty member, I will continue to bring to teaching the same commitment and enthusiasm that guide my research. Although I do not yet have experience leading larger courses, I am particularly excited about the chance to teach computer science—especially the introductory sequence—to students who are not yet familiar with the subject. At the same time, I am enthusiastic about teaching graduate and advanced undergraduate students the fundamentals of the subfields of programming languages and software verification, in the form of seminar courses or colloquia.