Rohan Padhye - Teaching Statement

Teaching Experience

I have been passionate about teaching ever since I was an undergraduate student in India. I would conduct tutorials for junior college (~12th grade) students on programming topics such as *Web Development* and *Game Development in Java*. Later, as a Master's student at IIT Bombay, I worked as a teaching assistant for three semester-long courses: a lower-division intro CS course, an upper-division course on compilers, and a graduate-level systems lab. I also volunteered as a teaching assistant for two summer schools conducted at IIT Bombay towards training graduate students, lecturers from other institutes in India, as well as industry professionals on topics related to compilers. Finally, I have worked as a Graduate Student Instructor (GSI) at UC Berkeley for two instances of a semester-long upper-division undergraduate class on *compilers*.

Course Material Development

My most significant contribution to education is the **ChocoPy programming language** and associated teaching artifacts [1]. I designed ChocoPy, with advice from Prof. Koushik Sen, in order to overhaul the undergraduate compilers' course at UC Berkeley. In this upper-div class, students first learn to reason about formalisms in programming language design—such as syntax, typing rules, and operational semantics—using a fully specified language as a running example. The students then develop a full compiler for this language in a semester-long project. We now use ChocoPy as this language. ChocoPy is the successor to COOL, which was the language originally designed for teaching this course at Berkeley more than two decades ago. COOL is a language made-up for classroom use; over the years, it was observed that students were not very enthusiastic to reason about the design of or to build a compiler for a language that they did not perceive as "real". In contrast to COOL, ChocoPy is a statically typed subset of Python. Since most students in the class are already familiar with Python, this brings several advantages: (1) students can reason about the syntax, typing rules, and semantics of ChocoPy using their prior knowledge about how Python works, (2) students can extend these formalisms to add features of Python that they use regularly but that ChocoPy does not support by default, (3) students can use the Python interpreter as an oracle alongside the ChocoPy reference compiler which we provide, (4) ChocoPy programs get syntax highlighting for free in any code editor that supports Python, (5) student-developed ChocoPy compilers outperform the standard Python interpreter on non-trivial benchmark programs, which can be extremely rewarding. Further, the artifacts associated with COOL were ageing; for example, the old project targeted MIPS, which most students are no longer familiar with. In the new project, students develop a compiler from ChocoPy to RISC-V.

Apart from designing the ChocoPy language itself, I authored most of its 36-page formal specification document. I also developed several artifacts, including: (1) a complete reference compiler, (2) three modules for the programming assignment with skeleton starter code, (3) an implementation guide, (4) an autograder, and (5) a Web-based ChocoPy IDE that can be powered by the students' own compilers! Additionally, I have provided consultation to the instructor of record and other TAs in creating lecture slides, discussion worksheets, and written assignments to use ChocoPy-based examples and problems.

After the first semester of the ChocoPy-based course (Fall 2018), we received an overwhelmingly positive response from the students through an informal survey and formal course evaluations. Many undergraduate students who completed this class subsequently enrolled for a graduate class on language design and semantics. In Spring 2019, Prof. Paul N. Hilfinger adopted ChocoPy to teach his offering of the same course at Berkeley. I completed teaching the third instantiation of the ChocoPy-based course in Fall 2019. Based on these experiences, I presented a paper on ChocoPy in the education symposium at the SPLASH conference [2], where it was well received. ChocoPy also featured on the front page of Hacker News [3] and in an article by TechRepublic [4]. I have already shared the ChocoPy artifacts with instructors at three other universities and continue to spearhead and maintain the project for re-use by others.

Teaching Philosophy

My approach towards teaching is strongly influenced by the notion of *relatability*. From my own experience, I have found that (a) new material taught in class is engaging when it is relatable to real-world observations, and that (b) the learning process itself is effective if it can be related to previous learning experiences.

In terms of material, the design of ChocoPy was heavily motivated by the desire to introduce projects and artifacts that students were already familiar with (e.g. working with Python and the use of IDEs). As TA for the compilers class, I followed this philosophy when addressing student questions. I would often bring up real-world examples, e.g. “Why do you think Java does X this way?” or “How do you debug your own programs when they don’t compile?”. I found students to be more responsive to such conversations rather than drab dissections of lecture slides or textbook material. When conducting my weekly discussion sections, I always begin with *Trivia of the Week*, which is a factoid that relates course material to the real world. For example, I started the discussion on *parsing* with “What kind of reference grammar does Python have? Let’s go to python.org and find out!” (spoiler: quite unusually, it’s *LL(1)*). I found that such digressions prevent students from viewing course materials as purely a means to a grade.

I have also had some success in implementing this philosophy during focused interactions in office hours. For example, when students were struggling with understanding how context-free grammars work, I found that effective learning strategies differed based on a student’s background. For some students with a programming background, I was able to help them connect the dots by relating production rules in a grammar to recursive function calls; for those with a strong math background, I succeeded in relating the design of new grammars to crafting a mathematical proof by induction. I noticed that focusing on relatability aids students in discovering their own style of reasoning.

Teaching Evaluations

I have been evaluated by 64 students across two compilers courses at Berkeley. For the eight criteria where I was rated on a scale of 1 (least effective) to 5 (most effective), I received a median rating of 5.0 in both semesters and a mean rating of 4.61 and 4.73 (department-wide average was 4.36 and 4.33 respectively). I also received several positive open-ended comments, two of which follow:

“Rohan is the best GSI in interactions across course staff and came out with very well organized projects –– Chocopy was beautifully done. He is very clear in answering questions and has a very good understanding of the material and the ability to communicate concepts across in an articulate manner. Chocopy was a very well done project and I think might have been one of the most satisfying CS course projects I have worked on in all of undergrad.” — *anonymous student from Fall 2018*

“Rohan is the best GSI I have had. A good GSI knows the material they teach and can run through section worksheets. A great GSI, like Rohan, can answer student questions effectively while also adding snippets of relevant knowledge that make attending his discussions special. I am very grateful for having taken this course and for having the opportunity to learn under Rohan. ” — *anonymous student from Fall 2019*

Advising and Mentoring

I am enthusiastic about fostering the intellectual and professional development of the next generation. For example, I participated as a panelist and presenter in the undergraduate research mixer hosted by Berkeley’s IEEE student branch. I have also worked with several undergraduate students—Vasudev Vikram, Connor Prisby, Sirej Dua, and Keyhan Vakil—on projects related to my research area: *fuzz testing*. Further, I have actively counselled junior graduate students on numerous occasions. For example, (a) I organized and hosted the Q&A panel for new admitted PhD students during Berkeley’s visit days, (b) I conducted mock prelim exams for five first-year PhD students, and (c) I served on the department-wide panel for advising mid-stage PhD students on qualification exams and thesis proposals. I have also mentored junior members of our research group such as Abdullah Younis and Azad Salam on navigating their initial years in graduate school.

Teaching Plan

Given my experience with the design of the ChocoPy-based compilers’ course at Berkeley, I am fully prepared to teach an undergraduate course on *programming language design and implementation*. I am also enthusiastic to teach intro CS classes on *programming* and *data structures*, as well as upper-division classes such as *systems security* and *software engineering*. Further, I would be eager to design and teach specialized graduate-level courses that draw upon my research interests, such as: *finding bugs in software automatically*, *program analysis and optimization*, and *virtual machines and managed run-times*.

References

[**1**] ChocoPy Website. https://chocopy.org.

[**2**] Rohan Padhye, Koushik Sen, and Paul N. Hilfinger. ChocoPy: A Programming Language for Compilers Courses. In *2019 ACM SIGPLAN Symposium on SPLASH-E* (SPLASH-E 2019).

[**3**] ChocoPy on HackerNews. https://news.ycombinator.com/item?id=20957420

[**4**] James Sanders. How ChocoPy uses Python and RISC-V to teach compiler creation. *TechRepublic*. https://www.techrepublic.com/article/how-chocopy-uses-python-and-risc-v-to-teach-compiler-creation