Teaching Statement

Gowtham Kaki

Teaching Preferences

Teaching Computer Science is in general exciting to me. My background makes me an ideal fit to teach graduate and undergraduate courses in Programming Languages, Formal Methods, Software Engineering, Databases, Distributed Systems, and Operating Systems. I would like to contribute to their curriculum drawing from my research experience at the intersection of these diverse fields. Software Engineering, for example, could be enriched with lectures and exercises on applying state-of-the-art program analyses on open-source databases, distributed systems, and web applications readily available on Github. The access to full provenance – commit history, bug reports, their resolutions, and developer comments, makes it possible to compare the results of the analyses on multiple versions of the source code, and determine their effectiveness in detecting the bugs known to be real or false. This would let students understand the shortcomings of the state-of-the-art in addressing the real-world problems, and prepare them for further research in Software Engineering. Likewise, I would extend the curriculum of Systems courses with recent developments in trustworthy sofware systems, such as Unikernels, IronFleet, seL4, and FSCQ File System. A graduate-level Programming Languages course, I believe, would benefit from an emphasis on automated reasoning wherever possible. While I do see the value in teaching students how to write certified programs and meta-theoretic proofs (normalization, type safety etc), I don't think making them labor through most mundane proofs in Coq is the way to go. Instead, I would restructure any such course around proof systems that put automation first, such as F*, Dafny, Lean, and Liquid Haskell.

I also have ideas for new courses that prepare graduate students for inter-disciplinary research with other branches of science and humanities. In particular, I am most excited about a seminar course that explores the possibility of using programs as a means of capturing human knowledge and thought process in the fields of Medicine, Law, and Economics. I am inspired by the recent successes of the pioneering work done by our fellow computer scientists in this direction. Examples include automated drug discovery by Might et al's MediKanren (OOPSLA'19 Keynote-1), Tax loopholes detection through Formal Methods by Lawsky et al (POPL'18 Keynote-3), and inviolable economic "smart" contracts on the emerging Blockchain infrastructure. I think it is worthwhile to institute a course to focus on such diverse applications of Computer Science so as to broaden the perspective of our graduate students as they decide on their thesis topics.

Teaching Methodology

In this section, I shall explain two core principles behind my teaching methodology.

Utility over aesthetics; problems over solutions In my observation, the common motivation for students to enroll in a course is to acquire the skills that they believe will help them in their career in research or industry. It is therefore understandable that if a student fails to see a clear causal relationship between a concept taught in the class and her professional success, she would disengage from the learning process. I believe there has to be a compelling utility of what is taught in the class in the context of students' professional journey. Such problem- and utility-guided approach to teaching is, I believe, more productive than the alternative approaches that stress on solutions, concepts and their aesthetics.

Rigor over breadth A CS instructor should strike a right balance between the breadth of a course curriculum, and the rigor with which the topics are taught in the class. When in quandary, I believe it is better to err on the side of rigor at the expense of breadth for rigor begets clarity, and clarity is a necessary precondition to transform passive subject knowledge into an active skill when an opportunity presents itself. Thus it is imperative to prioritize rigor over breadth in our teaching if we were to produce students who are highly skilled and not merely knowledgeable.

Teaching Experience

As a Visiting Assistant Professor at Purdue, I teach CS307 Software Engineering to undergraduates this (Fall'19) semester, which gives me an opportunity to test some of my ideas and beliefs in practice. The curriculum I designed (with significant inputs from my co-instructor and the previous instructors) includes Agile practices, basic and advanced testing techniques, Symbolic and Concolic execution strategies, a primer on concurrency bugs, and Gradual Typing. The major evaluation component of the course is a semester-long software engineering project, where students develop a large-scale web application in three "sprints". I have therefore structured my instruction to relate every concept taught in the class to real problems developers face in building large-scale web applications today. For instance, the lecture on concurrency bugs includes code examples from web applications, and a demonstration of how they can be exploited by hackers to inflict a financial loss (inspired by Warszawski et al, SIGMOD'17). Likewise the lecture on Gradual Typing includes examples contrasting Javascript from its gradually-typed variant – Typescript, and a demonstration of how the latter benefits frontend web developers. While the formal feedback is still pending, the informal feedback I got from students so far has been extremely encouraging.

Mentorship Experience

During my graduate studies at Purdue, I have had the opportunity to mentor junior graduate students who later became my co-authors. On four of my research papers, my immediate co-author (2nd or 1st) is a junior graduate student for whom it was the first ever publication. Beyond one-to-one mentorship, I have also taken initiatives for collective professional advancement that have stood the test of time. PurPL, a Purdue PL reading group I started in Fall 2013, is still active today[1]. I will be sure to take more such initiatives in the future as I enjoy sharing the excitement of learning and discovery with my fellow researchers.

[1] PurPL matured into a larger research organization at Purdue, but that is largely due to the efforts of others.