Teaching Statement

Kiran Gopinathan, April 2025

Teaching, kindling the spark of intellectual curiosity, and guiding the next generation of students on their onward journeys are amongst the most important and rewarding aspects of an academic career. When I started my undergraduate degree, I came from a background without the opportunities of having learnt programming before, and the focus on theory and emphasis on problem solving gave me not only the chance to catch up with my cohort but imbued me with key skills and perspectives that have guided me through my career. As a teacher, I deeply know the diversity of experiences and backgrounds that students can come into a classroom with, and as such I strive to create an inclusive and open teaching environment to facilitate and empower my students to succeed and make the most of their education. As faculty, I would be interested to teach courses across in programming languages, formal methods and software engineering, ranging from introductory to graduate level. My teaching goals are inspire within my students a passion to understand the topics we cover at a deeper level.

Teaching Experience

Over the course of my tenure at the National University of Singapore, I have served as a teaching assistant for several courses at undergraduate and graduate levels, in total dedicating over 400 hours to teaching duties.

I have TA'd two undergraduate-level courses at NUS, one introductory, "CS101E: Introduction to Programming", and one for second and third year students, "CS4215: Programming Languages Implementation". Both courses were large scale, aimed at class sizes ranging between 50 to more than 600 students. As a TA for these courses, I held office hours, ran lab sessions to complement the lectures, and gave feedback on programming assignments. For CS101E, given the diversity of experience within the cohort, with many students having little to no programming experience, I found walking through examples and questions and running them in an live editor helped to answer students' questions. This hands-on process of answering the students questions with a live demo alleviated the frustration many students were having as their first programming experience and helped to instil within them the confidence to experiment and try solving problems themselves.

For CS4215, while students in attendance were more experienced, their course project was implementing a real-world programming language of their choice, and thus involved many students' first experience with production compiler tools such as ANTRL or Yacc, as well as exposure to broader programming languages theory, such as the formalisms of small and big-step semantics, which at times could be overwhelming. To help students with this course, I held weekly lab sessions where I would check in on each student's progress on their project, giving feedback on their work and answering questions on course materials. I adapted my teaching style to reflect the students' experience and encourage independent learning by avoiding directly giving answers and instead guiding students through the respective documentation of the tools they were using, asking students to explain what they had already tried and then giving suggestions on where to look. Over the course of the semester, students gained familiarity in bridging the gap between the conceptual topics presented in their courses and production-scale tools, leaving them well-equipped to deal with real-world problems.

I have also TA'd for multiple graduate courses aimed at masters and PhD students, in particular, "CS5223: Distributed Systems" and "CS5218: Principles of Program Analysis". As a TA for these courses, I held lab sessions and office hours to help students with questions about the course materials and their projects. As these courses were aimed at masters students, the class cohort contained a far broader range of experiences than at the undergraduate level, with some students having little to no programming experience, with others having several years of industry experience. To support this diverse student body, I incorporated a more flexible teaching approach. I began each lab session with a brief review of the prior lecture, explicitly inviting questions—especially the kinds that students might hesitate to ask—thereby creating a space where less experienced students felt comfortable engaging. I also encouraged students to submit questions by email ahead of time, giving a low-pressure channel to raise concerns or handle confusion. Finally, to accommodate the more experienced students, I concluded lab sessions with pointers to more advanced course topics and optional challenge problems, and encouraged interested students to follow up in office hours, helping them to engage with the material at a greater depth without intimidating newer students and leaving others behind.

In addition to my role as a teaching assistant, I have also served as a guest lecturer for multiple graduate-level courses, most notably "CS5232: Formal Specification and Design Techniques", and "CS6217: Topics on Programming Languages & Software Engineering". CS5232 is a master's-level course that introduces students to formal methods and their application in software engineering. As a guest lecturer, I gave a series of seminars on using the Dafny theorem prover, guiding students through the methodology and process of formal specification and the construction of machine-checked proofs of program correctness. My aim was to demystify formal verification for students new to the subject, while building their confidence in using automated reasoning tools to write correct software. In CS6217, a research-oriented seminar course aimed at junior PhD students, I presented material on the key concepts in the design of formal verification tools, such as weakest precondition calculations. These lectures were designed to deepen junior students' understanding of the foundations of program verification and connect them with the state of the art research topics in programming languages and

formal methods. Across both lectures, I designed my lectures to break down formal concepts into approachable steps, using live demonstrations and real-world examples to motivate how formal reasoning is applied in practice.

Teaching Interests

I believe a solid foundation in programming languages and formal reasoning is an essential part of a well-rounded computer science education. These areas not only provide a grounded framework for understanding how programs behave, but also cultivate rigorous thinking and problem solving skills; a tool set invaluable across all areas of computing. I am particularly enthusiastic about teaching courses on the topics of programming languages, software engineering and formal methods at both undergraduate and graduate levels.

At the undergraduate level, I would be particularly interested in teaching courses on compilers, programming languages and software engineering topics. These courses provide students a chance to bridge theory with practice, and I would focus on the topics of language design, parsing, meta-programming and compiler implementation. I am also passionate about teaching foundational logic courses, covering topics such as propositional and first-order logic, and complementing the computer science curricula to provide students with the mathematical tools they will then be able to use to rigorously reason about their programs.

In addition to the above, I would also be eager to develop a course on an introduction to program verification for undergraduates. This course would aim to introduce students to the concepts of formal specification, program semantics, and correctness, providing them a framework and conceptual grounding to be able to reason about the behaviour of their code and write programs with provable guarantees. The teaching goals of this proposed course would be to introduce students to formal reasoning early their education, and provide a rigorous framework for reasoning about programs and their properties that they can use moving forwards.

At the graduate level, I am interested in offering seminar-style courses in formal verification, constructive logic, type theory and program synthesis. These advanced topics provide an excellent venue for students to engage with cutting-edge research and develop a foundation for building trustworthy systems. I hope to offer a research seminar on my own area of expertise, proof maintenance and repair, which introduces students to the challenges of keeping verified artefacts robust in the face of software evolution. Such a course would not only showcase state-of-the-art techniques, but also draw directly from my own research contributions, offering students a unique perspective on the frontier of programming languages and formal methods research.

Mentoring

Mentoring is one of the most treasured parts of academic life to me. I find great fulfilment and purpose in helping students grow into confident and capable researchers, introducing them to the principles of the scientific method, and sharing the excitement of discovering and shaping new research directions.

Throughout my PhD and post-doctorate, much of my graduate mentorship has taken informal forms. At the National University of Singapore, I regularly engaged in research discussions with fellow students in the lab, organised internal seminars, reading clubs and events, and offered guidance on navigating academic processes such as paper reviewing and giving presentations. As the most senior student of my advisor's lab, I aimed to foster a sense of community and support amongst my peers, creating space for collective learning and mutual encouragement. While as a faculty my responsibilities and position in such student spaces will naturally shift, I am committed to maintaining an environment where peer mentorship can flourish and students feel supported.

I have also mentored several undergraduate students over the course of my tenure at NUS. This mentorship has taken two forms: in some cases, I supervised independent research projects in an advisory role, and in others I collaborated hands-on with students on ongoing research. For the former, I held weekly progress meetings with students, discussing their progress, providing advice on how to navigate roadblocks, and helped to guide the project to publishable results. In these meetings I acted as an intermediary between the students and my advisor, providing one-on-one meetings with a faster feedback loop on progress and helping students record and refine their results into a more concise and actionable form. For the latter, I lead several research efforts, working side-by-side with students on shared code repositories, breaking off small and manageable sub-tasks that could be delegated to an undergraduate, and introducing them to the process of active programming languages research in a gentle and manageable way. Several of these students have gone on to pursue graduate study and I take great pride in having played a formative role in their early research careers.

The core of my mentoring philosophy when working with students is grounded in the idea of gradually building students' research independence. When mentoring, I begin by offering well-scoped and approachable problems to help students build momentum and gain confidence. As their confidence grows, I encourage them to engage more deeply with their research by prompting them to decide the next steps and research questions. When giving advice, I focus on encouraging exploration, including being open to the possibility of failure, as a means of building both technical judgement and to help students foster a sense of ownership over their work.

As a future faculty member, I intend to adopt this same approach to mentoring students in my lab. My goal is to create a welcoming and stimulating environment where students feel empowered to take initiative, build confidence in their work, tackle difficult problems and grow into independent researchers.