Cyrus Omar Teaching Statement

1 Teaching Experience

I served as the head TA for two courses at CMU. In both cases, the course was entering a period of significant transition, and I was able to substantially influence its development. For my role in these two courses, I was awarded the CMU School of Computer Science's **Alan J. Perlis Graduate Teaching Award** for 2013.

Functional Programming (15-150) I served as the head graduate TA, alongside 14 undergraduate course assistants, for the first full-scale instance of CMU's new freshman core sequence course on the theory and practice of functional programming. I led weekly recitations, held office hours, responded to questions on the course forum, graded code (written in Standard ML), graded handwritten proofs of program correctness, and contributed to the development of the homework assignments, grading rubrics, recitation notes and exams. The students in this course responded positively to my teaching style—my recitations were well-attended and I frequently needed to reserve a separate classroom for my office hours, because 25 or more students (out of about 180) would invariably attend. I also served as a guide to the undergraduate course assistants, often teaching them more advanced concepts while we graded or prepared recitations.

Principles of Programming Languages (15-312) I served as the senior graduate TA, together with another graduate TA, for CMU's undergraduate programming languages course. This course, taken by about 45 juniors and seniors, was taught by Prof. Bob Harper, who had just released his book *Practical Foundations for Programming Languages*. As such, we substantially revamped every homework assignment to follow the developments introduced in this book (which is now in its second edition and being used at an increasing number of universities). I played a leading role in this transition, developing most of the problem statements and reference solutions for the theory problems and contributing significantly to the problem statements and code for programming problems. I also held office hours, developed and presented a 30 minute lecture nearly every week during recitation, responded to questions on the course forum, and helped grade every homework assignment and exam. I also had the opportunity to give one lecture while Bob was away, where I introduced the fundamental concept of substitution in the lambda calculus and led students through the proof of the Substitution Lemma.

The feedback from my students was overwhelmingly positive. Examples from my student evaluations in this course include:

> **"Very patient when one on one, good explanations, helpful. Stays even after office hours technically over to answer questions, very good TA"**
>
> **"Know's what he's talking about and knows how to explain it."**

2 Course Preferences

The experiences described above have left me well prepared to teach undergraduate courses covering **Functional Programming** and **Programming Languages**. I am also prepared to teach the necessary mathematical prerequisities (i.e. discrete mathematics) and core courses in imperative programming—I also followed the development of CMU's new Imperative Programming course.

I am also interested in developing courses suitable for both upper-level undergraduates and graduate students on **Automated Theorem Proving and Program Verification** (I have substantial experience with automated theorem provers in my research), on **Software Security** (which will include coverage of topics that my research has touched on, like cybersecurity, language-based security, and usable security), and on **Human Aspects of Software Development** (I took a course on this topic taught by Brad Myers and Thomas LaToza at CMU, and my course project led to a publication at ICSE 2012, described in my research statement).

At the graduate level, I am well equipped to teach courses in **Type Theory** and **Formal Logic** (both the fundamentals and more advanced courses, e.g. on **Modal Logics** and their many applications). I am also looking forward to leading graduate seminars on various topics in programming languages and user interface design, driven by students' interests and our research goals.

3 Teaching Principles

When I teach, my goal is to help each individual student develop an understanding of, and appreciation for, fundamental principles. By this, I mean fundamental mathematical and scientific principles, of course, but also the fundamental principles of technical communication. For example, I believe that being able to use a shared vocabulary precisely and fluently is of fundamental importance. When a student in a recitation section or in office hours has a question about a function or proof that they are working on, I often ask them to read and explain their code out loud to me "from the top" and, as they do so, I gently help them rephrase their reading until it is clear and correct. This prompts them to organize their thoughts, so that by the time they have finished, many students have figured out the next step on their own. If not, they are better prepared to understand my subsequent guidance. The benefits of this approach extend beyond the classroom, because the very same communication skills help students engage productively with their future coworkers and collaborators.

I strive at the same time to communicate with **patience**, **sensitivity** and **generosity**, and I believe that we must intentionally and thoughtfully cultivate these qualities in our communication, just as we cultivate precision and rigor, if we are to build a computing culture that does not alienate those who are new or struggling with subtle concepts and unfamiliar standards of discourse. Often, these are individuals from under-represented groups who have not previously had access to a supportive technical community. I believe that great universities have a cultural responsibility to lead in this regard.

Finally, I believe that learning technical topics requires repetitive practice with targeted feedback. I plan to incorporate small, frequent homework assignments into every course I teach, and to provide timely and detailed reference solutions and restrospective review sessions. As Hazel evolves, I hope to use it as a platform to deliver even more timely feedback (see Sec. 4 below). I have been closely following research being done by Elena Glassman and others in this regard.

Although I am not primarily a computing education researcher, I do try to read relevant papers and blog posts in this area, e.g. Mark Guzdial's Computing Education Research Blog and Andy Ko's Bits and Behavior. I also closely follow the work of Kathi Fisler and Shriram Krishnamurthi on functional programming in computing education. Finally, I continue to work with Michael Hilton, an early contributor to the Hazel project who is now teaching faculty at CMU.

4 Research in the Classroom

For me, teaching and research go hand-in-hand—I am quite excited about the possibilities of incorporating the artifacts of my ongoing and future research into the classroom (once it is pedagogically appropriate to do so), and on using the feedback from these experiences to help improve my research. In particular, I am working on two relevant artifacts: `typy` and Hazel.

`typy` is a statically typed functional-first language embedded into Python as a library. It was introduced in our paper on semantic fragments at GPCE 2016 (see my research statement). This language could potentially resolve a perennial tension: should we teach using a language that students are likely to see in industry (like Python) or a language with a simpler semantics that corresponds more closely with the mathematics they have learned (like ML/Haskell)? With `typy`, we have strived to make granularly interleaving these choices easy. Given my experience in computational neurobiology, it could be interesting to develop a functional programming course around `typy` targeted at computational and data scientists.

Hazel is the live programming environment that will serve to organize much of my future research (see research statement). I am helping my collaborators at Boulder (Matthew Hammer, Ben Shapiro and David Moon) prepare to teach (1) an introductory functional programming course; and (2) an adapted version of the Bootstrap Data Science course module using Hazel. The first possible benefit of using Hazel in these settings is that it eliminates the possibility of syntax errors. In my experience, syntax errors were particularly confusing for novice programmers. Other structure editors have also been used successfully in educational settings, notably Scratch. The idea would be to start with Hazel and transition later in the term to an existing language with a similar semantics but a textual syntax (e.g. Elm, OCaml or Haskell). Recent research on starting with a structure editor for Elm and transitioning to text later in a student's studies has been promising [Zhang et al., CASCON 2018].

In our POPL 2019 paper, which describes how to reason about and run incomplete programs (programs with holes), we conjectured that Hazel's user interface would be particularly useful for an instructor in a functional programming course by allowing them to incrementally fill out a program during a lecture and point visually to the types and values of intermediate results at all times (i.e. without gaps). It might also be helpful for students by providing them with rapid feedback about program behavior throughout the editing process. We hope to further explore this conjecture with our teaching efforts.

Our ongoing research on palettes in Hazel, described in my research statement, will allow us to integrate graphical elements directly into Hazel programs. In the future, I am interested in using Hazel, together with a collection of domain-specific palettes, to teach programming to artists, designers and musicians. My post-doc advisor, Ravi Chugh, has similar ambitions at Chicago. By integrating UI elements that are already familiar to people in these domains into the programming experience, we hope to smooth the transition to programming. This is one way that I hope to help broaden participation in computing.

I am also excited about the possibility of having students build their own palettes in the context of a course project—the task of designing and building an interesting API together with one or more palettes is just the right size for an undergraduate course project designed to span a few weeks, and the result is usually something interesting. It also exposes students to the basics of code generation, so it is a reasonable kind of starter project for students who may want to go into programming languages research.

5 Advising Experience

At CMU, I was the primary point of contact for three undergraduate research students—Nathan Fulton, Chenglong Wang and Benjamin Chung. All three were able to publish their results and went on to prestigious PhD programs in programming languages (at CMU, Washington and Northeastern, respectively). Nathan and Benjamin contributed to papers that received paper awards (at PSP 2014 and ECOOP 2014, respectively). I also helped mentor a then-first-year graduate student, Darya Kurilova, who also contributed to the award-winning ECOOP 2014 paper.

In the context of the Hazel project, I work with and mentor Ian Voysey, a CMU undergraduate turned independent researcher who has taken the lead on developing mechanized proofs of our core calculi using the Agda proof assistant and earned second author on both POPL papers (POPL 2017 and POPL 2019). Ian is considering applying to graduate school.

At Chicago, I worked together with an undergraduate, Charles Chamberlain, on implementing the mechanism from my ICFP 2018 paper into Reason, which is Facebook's popular new front-end for OCaml. This project was recently released (in November 2018) and has since been picking up substantial external interest. Charles was selected to present this work at the recent OCaml Workshop at ICFP. Charles now works at Jane Street Capital, an investment firm that heavily uses OCaml. I continue to collaborate with Charles, who has developed a significant hobby interest in programming languages research and is now starting to contribute to the Hazel project as well.

Our work on Hazel has received substantial interest on social media (my website provides some examples). As a result, I am frequently approached about contributing to Hazel. My approach has been to invite interested people to the Hazel chat channel (Slack), where they are encouraged to read our papers (and the necessary background), openly ask questions, and discuss their ideas. Two industry programmers who did just that are now first-year graduate students working with us on Hazel. Nick Collins is a PhD student with my post-doc advisor, Ravi Chugh, at Chicago, and David Moon is a PhD student with Matthew Hammer at Boulder. I co-advise both Nick and David. Overall, there are now over 40 people in the Hazel Slack—a mix of academic collaborators and non-academics who are learning more about Hazel as just described. Some of these have started to contribute directly to Hazel. For example, Brandon Kase (who took 15-312 as an undergraduate when I was the head TA) converted major parts of the Hazel implementation to use Coq, and Aida El Kouri, an undergraduate at the University of South Carolina, is starting to contribute new edit actions to Hazel. I am excited about fostering a healthy and supportive technical community around Hazel.