Teaching Statement

Prashant Kumar

Teaching Experience

I have always enjoyed teaching and taken every opportunity that came my way. Throughout my academic journey—from my MSc. at the University of Calgary (Canada) to my PhD at Oregon State University (USA), and now as a postdoc at JGU Mainz (Germany)—I have worked as a teaching assistant for eleven Computer Science courses and taught two courses on my own.

At the University of Calgary, I taught Compiler Construction for four consecutive semesters. The course was especially challenging for students as it combined theoretical concepts with progressive implementation assignments, where each assignment built upon the previous one. Seeing students struggle with both aspects, I took initiative. I introduced remedial Haskell tutorials beyond regular hours, maintained an open-door policy for students to seek help anytime by walking into my office, and provided detailed guidance on approaching assignments and debugging code, as most students were new to Haskell. It was demanding but rewarding work. Prof. Robin Cockett recognized the impact, commenting on my "infinite patience in explaining ideas to students" in the recommendation letter I obtained from him when applying for the Postdoc position at JGU (letter attached as a supplementary document) and ensuring I remained the teaching assistant throughout my time there.

I also taught Programming Paradigms for three semesters at the University of Calgary and once at Oregon State University. This course introduces functional and logic programming—paradigms that expose students to fundamental concepts like recursion, algebraic data types, lambda calculus, and unification. While it is a required course in most CS departments, ensuring students encounter these beautiful ideas, it often presented a challenge: these paradigms were typically condensed into brief overviews, leading students to view them as mere variations of imperative programming rather than as distinct approaches to problem-solving. Many initially attempted to debug functional and logic programs using imperative techniques, leading to frustration.

I structured my tutorials to address this gap, helping students develop appropriate mental models for each paradigm. I guided them in adjusting their thinking, making the transition smoother. The effectiveness of this approach showed in student engagement—my tutorials remained well-attended, even at 8 AM on freezing Calgary mornings with negative double-digit temperatures.

Additionally, I worked with Prof. Cockett in Foundations of Functional Programming—an advanced Programming Languages course that delves into advanced features of functional programming, type systems, and language implementation. I helped redesign several assignments to better reinforce key concepts (confirmed in Prof. Cockett's attached letter). Seeking to improve my teaching effectiveness, I invited Prof. Cockett to observe one of my tutorials and provide feedback. His advice to slow down and allow students more time to process new ideas has since shaped my approach to teaching complex materials.

At JGU Mainz, I have independently designed and taught Advanced Topics in Functional Programming twice, developing both theoretical content and practical tutorials. This course explores advanced Haskell features and functional data structures. I built the curriculum from scratch, structuring lectures and designing exercises that help students apply theoretical concepts through hands-on programming. In the second year of offering, the course has seen a surge in the number of students with particularly strong interest from advanced undergraduate students, reflecting both the course's reputation and growing interest in functional programming. Beyond my own course, I actively contribute to

departmental teaching activities, serving as an examiner for students' term papers and presentations in two seminar courses (Program Analysis and Logic Programming) over two semesters.

I have also enhanced my teaching methods I have also enhanced my teaching methods by observing others. At JGU, I sat through two of Prof. Sebastian Erdweg's courses: Program Analysis and Logic Programming. Beyond deepening my understanding of these subjects, I gained valuable insights about teaching. I observed that having students solve examples immediately after introducing new concepts, challenging them to test their understanding, leads to better clarity than deferring exercises to later tutorials. I have since incorporated this approach in my own course.

Student Mentorship

My engagement with students extends beyond teaching courses. At Oregon State University, I led two undergraduate researchers in our research group for a year, working on explainability of rule systems. I also mentored two master's students in the Programming Languages (PL) group. Understanding that PL research can be daunting for newcomers, I helped them with advanced Haskell features and fundamental concepts like type systems. I maintained an open-door policy, encouraging them to bring any PL-related questions. For instance, when they struggled with generating typing rules for advanced language features, I would guide them through the process while ensuring they understood the underlying principles for their future work. One student was working on Abstract Machines—the topic of my MSc. research—and I was able to help her build a strong foundation in this challenging area.

I also co-mentored a total of three high school students with my PhD advisor, Prof. Martin Erwig, through the Apprenticeships in Science and Engineering (ASE) program over two summers. I was tasked to introduce them to functional programming using Haskell and Elm. Their success with functional programming was remarkable, especially given that even advanced undergraduate and graduate students often find this paradigm intimidating.

At JGU Mainz, I supervised two Bachelors' theses: one independently and another in collaboration with my colleague David Klopp. I maintained regular bi-weekly meetings to ensure consistent progress and guidance. I placed particular emphasis on writing, regularly reviewing and helping improve their chapter drafts. For their presentations, I guided them in making complex technical content accessible through examples and clear explanations. These efforts paid off—both students completed their theses successfully with very good grades.

Teaching Approach

My teaching philosophy is built on four key principles that I have developed and refined through years of teaching theoretical computer science. These principles help create an engaging learning environment where students can tackle complex concepts effectively.

Example-Driven Learning: My teaching approach is example-oriented. Through teaching courses with a heavy theoretical component, I have come to realize that abstraction and definitions are quite challenging for most students in the absence of enough examples. Examples help them understand the intuition and retain the theory long term.

Guided Discovery: I believe that computer science is best learned through exploration and guided problem-solving. Too often, students see algorithms and theorems presented as fait accompli, as a polished and static artifact. I structure my courses to guide students through the problem-solving process, including false starts and intuitive but incorrect ideas. After this, when I show them the actual solution, they appreciate it much more. The hope is to keep them engaged in the class as well as give them a head start on the research process.

Interactive Learning Environment: I openly encourage students to challenge me during discussions, and try to keep the classes very conversational where anyone can chime in with their opinions. This

results in a better and more stimulating experience both for students and me. However, this requires work, since only when students are not intimidated do such interactions occur. To create such an atmosphere, I maintain an approachable presence both inside and outside the classroom.

Human Context in Technical Education: Technical education can sometimes strip topics of their historical and human context, reducing breakthroughs and stories of human struggle and ingenuity to dry technical results. In my opinion, the human element makes students care about the subject, sparking a long-term interest in the topic. I still remember how fascinated I was in my proof theory course—and still am—with the stories of Brouwer's and Hilbert's lives and their conflict around the philosophical idea of intuitionism. I make a point of discussing the history of a topic as well as its researchers briefly, and I can sense the fascination in students as well. Once I have that, I can delve into any amount of details without fear of losing their interest when the material becomes complex.

Future Teaching Contributions

At JGU Mainz, I can contribute to teaching of both basic and advanced courses.

At the bachelor's level, I can teach:

- Introduction to Programming
- Introduction to Software Development
- Programming Languages — Drawing from my experience teaching Programming Paradigms at University of Calgary and Oregon State University

At the advanced level, I can teach:

- Program Analysis — Having served as a teaching assistant and attended recent lectures for this course, I can teach it in the future
- Compiler Construction — Drawing from my four semesters of experience as a teaching assistant and my MSc. work on compiler implementation
- Advanced Topics in Functional Programming — Currently teaching this course independently at JGU
- Functional Data Structures and Algorithms — I already cover functional data structures in my current course, but I want to offer a specialized course focusing on formal verification using Coq
- Domain-Specific Languages — Building on my research experience, including two published DSLs, one of which received a best paper award for its innovative approach to game-theoretic matching
- Concurrent Programming with Erlang — Drawing from my research on message-passing languages, I can offer this course in the near future