Statement of Purpose

I am Siva Somayyajula, a senior mathematics and computer science major at Cornell University, and I am interested in the symbiosis between mathematical reasoning and computation. This idea spans formal methods in software engineering and formalizing mathematics. Simply put, I believe that the future of these disciplines lies in the use of sophisticated tools like proof assistants for correctness and efficiency. As such, I would like to pursue graduate study in the many facets of this interdiscipline—from the underlying theory, which draws from programming languages, logic, and type theory, to its applications. That is, I am also interested in developing proof tools and using them towards writing mathematical proofs and real-world software. These are all key areas of focus in the Principles of Programming Group and the Pure and Applied Logic program, which is why I would like to pursue doctoral studies in computer science at CMU.

Throughout my education, I have always been fascinated by this perspective. As a high school intern at the U.S. Naval Research Laboratory, I had my first taste of "Theory B" implementing verification condition generation for a C code security analysis tool called SecProve. This experience instilled in me an interest in programming logics, which I pursued in college. In fact, during my second semester at Cornell, I started working under the supervision of Prof. Nate Foster on NetKAT, a network programming language based on Kleene Algebra with Tests (KAT). Under his supervision, a team of graduate students and I developed a coalgebraic perspective on an extension to the notion of behavioral equivalence in that language. To elaborate, we designed a language for specifying equivalences on network policies and exploited the automata-theoretic interpretation of KAT to write a procedure deciding these relations. I went on to implement a prototype of this system under the supervision of Prof. Dexter Kozen over the course of my second year (see here). In both projects, I learned to appreciate the beauty of formalism and its power in reasoning about real-world problems. In particular, it is difficult to comprehend, let alone reason about the subtleties of such complex phenomena as the security of system code and the behavior of network policies without elegant abstractions like Hoare logic and KAT. But most importantly, Profs. Foster and Kozen helped me develop an intuition for the deep connections between logic and programs.

This correspondence was finally realized when I enrolled in Prof. Robert Constable's course on applied logic during my second year. He expounded upon type theory, which concretizes the duality between constructive mathematics and computation. I find it difficult to really express how exciting the proofs-as-programs paradigm is—to write programs in terms of proofs while simultaneously reaping the computational content of a proof is sublime! As a result, I co-developed a framework for implementing refinement-style proof assistants for this class, which spurred my interest in both the tooling and practice of automated theorem proving. Due to the immense impact this course had on my career, I am the teaching assistant for it next semester (spring 2018), where I will give lectures on type theory and the applications of proof assistants.

I decided to devote all of my time to studying this phenomenon—in my last few blog posts, I have written at length about the computational role of constructive proof and deriving programs from direct mathematical reasoning. Furthermore, I wanted to learn about all that is ancillary to this concept—during my second year, I took undergraduate and graduate offerings on the theory of programming languages, which describes many aspects of proof automation infrastructure. Then, I secured a position in the REU in Mathematics at Indiana University Bloomington for the summer of 2017 studying homotopy type theory under Prof. Amr Sabry. With a team of graduate students, we formalized connections between a reversible programming language and a type-theoretic model for reversibility in the Agda proof assistant (see here). Prof. Sabry not only showed me the forefront of computer-formalized mathematics, but also the deep foundational questions that arise when reconciling classical with reversible computation. Thus, I am continuing my work with him on formalizing a homotopy-theoretic interpretation of this model, which we hope will elucidate connections between computation and topology.

In finishing my undergraduate studies, I have come to the realization that I would like to focus my career on automated theorem proving. In fact, having conversed with Profs. Harper and Pfenning in the department of computer science and Prof. Avigad in philosophy about graduate opportunities, I feel that CMU is a great institution at which I may both learn about and contribute to this field.

In particular, after completing a self-study of cubical type theory (CuTT) (see here) for a graduate course on constructive type theory taught by Prof. Constable, I contacted Prof. Harper due to his current work on computational higher type theory (CHTT), which builds on the intuition of CuTT. After looking into the resources he sent me about CHTT and the associated RedPRL proof assistant, I am eager to continue studying CHTT and Vladimir Voevodsky's univalent foundations in general. With them, we just might be able to achieve Voevodsky's dream—painlessly verifying all of mathematics.

Furthermore, Prof. Pfenning and I discussed his work towards a concurrent type theory that would unite the expressive power of dependent type theory, the foundation of many proof assistants, with primitives for concurrency. After reading his paper, I am convinced that such proof-theoretic investigations of practical computation concerns will yield powerful tools for software engineers. Thus, I am keen on exploring similar concepts during my graduate career.

Lastly, Prof. Avigad explained to me his interest in formally verifying defense-oriented hybrid systems using the Lean proof assistant as well as the Formal Abstracts project, which aims to utilize proof assistants to verify various mathematical results. The reasons I am eager to pursue such projects are two-fold. First, a *lack* of correctness can be catastrophic, especially in mission-critical software where lives are at risk. As Prof. Constable remarked, it would be unwise to trust a car with your life if its software is not thoroughly tested and verified. But perhaps most importantly, formalization admits new ways of thinking about problems. As I stated earlier, the proofs-as-programs correspondence unifies mathematical and "computational" reasoning.

To close, my studies and experience have brought me to this intersection of mathematics and computer science which I hope to pursue at CMU, given the great faculty and breadth of research opportunities. As Prof. Harper puts it, "if you arrive at an insight that has importance for [mathematical] logic [and programming] languages. . . then you may feel sure that you have elucidated an essential concept of computation—you have made an enduring scientific discovery." With graduate study in these areas, I hope to make such an impact on the way we do math and write software.