My interest in functional programming languages and compilers sprang from my old oboe teacher's philosophy: “Never take no as an answer from an inanimate object.” If a tool was defective or incomplete, I always considered the option of fixing it myself. This philosophy served me well as a developer for multiple open source projects as well as team leader for Scripts, a shared web hosting service used by four thousand members of the MIT community. Eventually, however, I discovered fundamental problems in the programming languages and compilers I was working with. Why didn't the vast numbers of web developers brought up on PHP understand the difference between HTML and plaintext? Why did writing delicate security-sensitive programs boil down to combing through massive checklists of known vulnerabilities by hand? Why was it that most developers deemed advanced optimizing compilers, domain specific languages and other methods of increasing expressiveness too complex to understand and use?

Looking for ways to improve these tools led me to experiment with different programming languages. At Galois, I worked on Cryptol, a compiler and verification system for cryptographic algorithmsMy work was on the translation layer which converted these algorithms into boolean circuits which could then be verified against a reference implementation using an SAT solver. At Microsoft Research Cambridge, as a visiting student from the University of Cambridge, I worked on the new code generator in GHC, improving the correctness and performance and size of the generated code. Code generation is a critical step in the creation of high level programming languages, which require many transformation steps from their original representation to machine executable code.

Through these projects I have discovered more problems which I would like to investigate. For example, working with a SAT solver whose interface was written in C informed me about many of the undocumented invariants that would be much better encoded in a type system. While working on GHC, I learned more about the challenges of ensuring that powerful abstractions also translated into code that consistently performed well. I also have enjoyed learning about work outside my primary interests; for example, at Jane Street over the most recent summer, I enjoyed corresponding with the distributed systems community while creating a novel crash-recovery algorithm for the distributed consensus algorithm Paxos, after discovering that a mechanism which had been published in the literature was incorrect. The pursuit of a doctoral degree will allow me to dig deeper into my primary interests, as well as develop connections between other fields. In the end, my goal is to fuse powerful theory and sound implementation to create programming languages, tools and knowledge to help advance the state of computer science, security and software engineering.