Statement of Purpose

What first drew me to the field of programming languages, and what has kept me interested since, are the foundational logical ideas that underlie the theory behind programming. I am pursuing a PhD in Computer Science because of my interests in the design and implementation of programming languages, compiler design, type theory, and using these ideas to create safe high-level abstractions for programmers. My interest in University of British Columbia stems from Professor William Bowman's work on type-preserving compilation for dependently typed programming languages and Professor Ron Garcia's work on gradual type systems. Type-preserving compilation is particularly appealing to me because of my experience writing a compiler in my compilers course. As I developed this compiler to compile larger programs, some particular programs would crash unexpectedly. Each construct in the language had an individual translation to the target language; however, as the language grew, these individual translations no longer interacted together sensibly, resulting in generated programs that didn't make sense as a whole program. In pursuing a PhD, I hope to develop the theory that will guarantee correctness of programs throughout compilation, so that programs will no longer have this undefined behavior. My previous experiences with dependently typed languages have shown that this is particularly vital for these programs, as the users of these languages are relying on the correctness of the compiler for the correctness of their proofs.

My experiences in undergraduate research have involved the design and implementation of programming languages, which has given me the appropriate background in both compiler design and type theory. Here in practice, I encountered the same difficulties that came up in my compilers course. Currently, I am working on my undergraduate thesis with Professor Sam Tobin-Hochstadt, creating a partial evaluator for the Racket core language. My research experiences began with work on a compiler called Gibbon, a project started by Professor Ryan Newton at Indiana University. Gibbon transforms programs to operate directly on serialized data, which creates highly-optimized programs in a provably safe way. I extended the language that Gibbon compiles from a first-order language to a higher-order language. To achieve this, I created an intermediate representation (IR) for higher-order programs, and implemented a transformation of this IR to an equivalent IR for first-order programs. This transformation is done through monomorphization and closure-conversion. Monomorphization is specializing polymorphic functions to the types of the arguments from specific call sites, which is done by generating new specialized functions. Closure-conversion converts functions passed as values to data-structures, which is a primary step in attaining a first-order program. From this experience, I learned how to find primary resources relevant for my project, implement these ideas in a new context, and present my ideas in a written report, all while working collaboratively with graduate students.

Further pursuing my research interests, I participated in the Carnegie Mellon University Software Engineering REU program (REU-SE). There, I helped design Obsidian, a programming language for smart contracts, along with a case study to evaluate its design. Obsidian uses a combination of typestate and linear types to create safer smart contracts. Linear types are used to track important assets throughout the program. My case study resulted in the addition of a new language feature: enabling assets to exist at certain states of the program. For example, if a wallet is completely empty, losing it may not be harmful. Conversely, if a wallet is full of money, losing it would be unfortunate. This wallet has two different states:

it either contains money, or it is empty. We shouldn't worry about keeping it safe when it is empty, but we should take special care of it when it has money. This type of situation arose while I developed my case study, which is why we chose to add this feature to Obsidian. I presented this work at the Systems, Programming, Languages and Applications: Software for Humanity (SPLASH) Student Research Competition, where I received second place.

My work on Obsidian has motivated my interest in compiler verification. The Obsidian language uses a type system to detect specific errors in programs before compilation. I found this particularly intriguing, but now I am interested in how types can ensure the correct behavior of programs *after* and throughout compilation to prevent the undefined behavior (and frustration) I experienced in my compilers course and research experiences. As a result, I am interested in Professor William Bowman's work on providing these guarantees throughout compilation of dependently typed languages.

After pursuing my PhD, I plan on working in research as a professor to continue collaborating with and mentoring students. I have had numerous experiences in undergraduate teaching, even creating new videos integrated into courses at Indiana and Northeastern University, and I greatly value the insights I can gain from instruction. I especially appreciate the work that has been done for CPSC 110 at UBC, a course based on *How to Design Programs* (Felleisen et al.), and the material I developed was heavily inspired by the edX component of this course. I also appreciate the insights I have gained from working in collaborative research environments. For these reasons, I am particularly excited for the opportunity to work in the collaborative research and teaching environment at University of British Columbia.