**GRFP Personal Statement** Ben Greenman

As I was growing up, my parents imposed a strict discipline on my life. Looking back, I can see they wanted the best for me—both had associate degrees and hoped I would earn a bachelor's—but by the time I entered high school I had resolved to move out when I turned eighteen. To support myself, I worked a variety of evening jobs: first at a local restaurant and later on as a janitor. It was during one of these late nights at the restaurant that one customer, a lawyer, told me about his experiences as a student at Cornell University. That night I decided Cornell was my dream school, and applied to the Industrial and Labor Relations program the following year.

To my delight, Cornell accepted me as a guaranteed transfer [3] and I looked forward to arriving as a sophomore. Until then, I spent a terrific freshman year at Hudson Valley Community College, taking a few courses to satisfy Cornell's requirements and also two semesters of calculus and physics. I loved the welcoming, collegial environment at HVCC—my teachers were encouraging and helpful, and my fellow students were friendly and cared about learning. The experience made me all the more excited about going to Cornell.

When I finally arrived in Ithaca, I experienced a culture shock. The only friendly students I met were those trying to recruit me into student government organizations, and teachers cared more about publishing articles than giving quality lectures and assignments. It was a sad time. I felt disengaged, my grades dropped, and my evening work as a janitor left me drained. Thankfully, my search for affordable housing had ended in an apartment of upperclassmen engineers. They encouraged me to try a minor in computer science. Following this advice was one of the best decisions I have ever made.

At the start of my second year at Cornell, I learned about functional programming and immediately fell in love with its brevity and elegance. Functional programming became my hobby, and that Fall I wrote an AI that won a class-wide tournament and earned me a position as a teaching assistant. Good-bye custodial work! I had found my passion. The following summer, I got a job at a New York City startup consisting of two entrepreneurs, one real estate broker, and one programmer. For the rest of my undergraduate career, I worked part-time both as a TA and as a developer for this company. Though I was curious about research, I felt ashamed of my non-standard background and had planned to just continue as a software engineer. Then, a semester before I graduated, Cornell hired Ross Tate and I decided to try approaching him.

**Decidable Subtyping for Object-Oriented Languages** Summer – Fall 2013

Professor Ross Tate is both an academic and the type systems advisor for teams at Red Hat and JetBrains that work on object-oriented languages for managing large software projects. These companies have three main priorities; they want code to be self-documenting, highly reusable, and backed with strong compile-time guarantees. However, the features empowering reusable code in mainstream languages sometimes cause the compiler to diverge. For example, a simple type-safe function for comparing two binary trees will cause the Eclipse and Visual Studio compilers to crash with out-of-memory errors.

Curious to learn if similar companies had encountered this issue, I implemented a source code analyzer to detect cycles in the inheritance hierarchy of Java projects. Next I modified the `javac` compiler to log a warning whenever a class/interfaces used in a cycle appeared as a field, type argument, method parameter, or return type. After running this analysis on over 13 million lines of code taken from 60 open-source Java projects, Ross and I realized we could safely remove the cause of these cycles by formalizing the patterns apparent in this case study. Thus, Ross, myself, and Ph.D student Fabian Mühlböck spent the Fall creating a practical type-checking algorithm that avoids these out-of-memory errors [7]. We presented our findings at PLDI '14 in Edinburgh this past summer.

---

**Conditional Inheritance** Spring – Summer 2014

---

Hooked on the excitement of a successful research project, I continued to study object-oriented type systems the following Spring. With the issue of undecidable type-checking solved, Ross and I hoped to build a mechanism for code reuse called *conditional inheritance*. For example, conditional inheritance empowers a programmer to declare that a list is clonable provided there is a procedure for making copies of each of its elements.

At the time, we thought this extension would be straightforward. Indeed, for a language like Java, such conditions are decidable in our system. The main challenge is in determining which conditions a bounded variable satisfies, and Java's wildcards feature is well-suited to this task. But few languages allow wildcards, so I worked to find a solution compatible with the restrictions of C++ and Scala. Week after week, I presented new designs to Ross only to return to the drawing board when one of us discovered a new corner-case. After four months of iteration, we admitted that any fully backwards-compatible design would be too restrictive to be practically useful. In light of this, I am now building a language and type-checker that allows conditional inheritance, but only on classes and interfaces that are uniquely determined by their parameters. The goal is to test whether this more restricted strategy can still articulate invariants in the Java collections library.

---

**Kleene Coalgebras** Spring 2014

---

Also that Spring, I worked with Professor Dexter Kozen and Ulrik Rassmussen, an exchange student from the University of Copenhagen, on coalgebraic decision procedures. Much of Dr. Kozen's research is on practical applications of Kleene algebra, the language of iteration. Two recent examples are NetKAT [1], a sound and complete network programming language, and KAT+B! [6], a logic for reasoning about mutable boolean variables. These theories provide *algebraic* decision procedures useful for determining when two terminating programs are equivalent. However, practical applications like user interfaces or servers are designed to run forever. A coalgebraic decision procedure allows reasoning about these "endless" programs.

When we began, Dr. Kozen and his collaborators had already developed coalgebraic procedures for KAT (Kleene Algebra with Tests) and NetKAT [5], so we focused on KAT+B!. Together, Ulrik and I studied past work on Kleene algebras and on coalgebras. We then constructed a decision procedure for KAT+B! following similar reasoning as the groups working on KAT and NetKAT had used. This common structure prompted us to investigate

the possibilities for a unified system for deriving the coalgebraic decision procedure of any Kleene algebra with additional equations, but we have not yet reached a conclusion. What I gained from this experience was a finer appreciation of theoretical techniques in programming langauges and of teamwork. Although I found this more abstract research challenging, the small advances I did make inspired me to study even harder.

## Broader Impacts

Teaching has been a part of my life for nearly a decade. For three years in Albany, I taught elementary school students at my church's vacation bible school. While a student at Hudson Valley, I volunteered after school at the local Boys and Girls club. During my first summer in Ithaca, I designed a weeklong tutorial on LEGO Mindstorms robots and taught it to four classes of elementary and middle-school students.

As a student at Cornell, I was isolated and unsuccessful until I began teaching. Becoming a TA changed my life in two ways: it connected me with other students and helped me learn more effectively. A glance at my transcript shows that I nearly failed a course in abstract algebra my second year. But after learning functional programming well enough to teach it, I revisited the material and found algebra so enthralling that I sought out Professor Kozen for a research project. The connections between programming and mathematics are amazing.

For this reason, I am excited to join Matthias Felleisen's research group. Over the past 20 years, he has created and run outreach projects teaching algebra through functional programming [4]. The modern incarnation of this vision is the Bootstrap program [2], which teaches pre-algebra and geometry to middle school students in underprivileged neighborhoods (including Dorchester, Harlem, and East Palo Alto). Already I have plans to volunteer with Bootstrap, and I fully expect that some of my research will draw from and influence this program, just as Dr. Felleisen's research has since 1995.


[1] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. NetKAT: Semantic foundations for networks. In *POPL*, 2014.

[2] Bootstrap. `http://www.bootstrapworld.org`. Accessed: 2014-10-14.

[3] Inside Cornell's guaranteed transfer system. `http://cornellsun.com/blog/2013/09/30/guest-room-inside-cornells-guarantee-transfer-system/`. Accessed: 2014-10-28.

[4] Matthias Felleisen. TeachScheme!: a checkpoint. In *ICFP*, 2010.

[5] Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. A coalgebraic decision procedure for NetKAT. In *POPL*, 2015.

[6] Niels Bjørn Bugge Grathwohl, Dexter Kozen, and Konstantinos Mamouras. KAT + B! In *CSL / LICS*, 2014.

[7] Ben Greenman, Fabian Muehlboeck, and Ross Tate. Getting F-bounded polymorphism into shape. In *PLDI*, 2014.