A Research Journey Driven by Interesting Problems

Greg Morrisett

July 4, 2026

In this lecture, Dr. Greg Morrisett reflected on his decades-long research journey in programming languages, compiler design, type systems, formal verification, and beyond. Rather than presenting a single research topic, he looked back on the ups and downs of his career, showing how each research direction emerged from the limitations of previous work or from unexpected opportunities. Throughout the talk, he also shared the lessons he learned along the way.

1 The Highs and Lows of Language Researcher

While highs and lows represent successes and failures in Morrisett's career, it also comments on his research interests oscillating between high-level languages (ML-like languages, OCaml) and low-level languages.

Some advice that he gave to get through the lows was pursuing things that you value, not being afraid to change direction or topics, getting your hands dirty (from reproving theorems to building prototypes of systems), explaining your work to someone else and giving talks at other universities, and being patient with yourself - some research ideas take years to make an impact on the real world.

1.1 Typed Assembly Language: A Research Cycle

Greg Morrisett began his journey by building a TCP/IP stack in Standard ML. Although the implementation was elegant and highly modular, it performed significantly worse than equivalent C implementations. This was because high-level languages provided little control over data representation. This problem motivated his PhD research on compiling high-level languages efficiently.

The fundamental challenge was polymorphism. Since the concrete type of a polymorphic value is unknown at compile time, the compiler cannot determine its memory representation. He proposed

Compiled By:
Namratha Gangamreddypalli, Jasper Geer, Max Gross,
Abigail Pribisova, and Konami Shu

passing type information at runtime so that polymorphic functions could adapt their behavior accordingly. Although this approach ultimately suffered from runtime overhead, it led to a more important insight. By preserving types throughout the compilation process, intermediate compiler passes could be type checked, making the compiler much easier to debug. This idea eventually evolved into Typed Assembly Language (TAL), where type information is preserved down to the assembly level.

In the TAL project, he demonstrated that types could express low-level properties such as calling conventions, register usage, and memory layouts. This opened up a rich research direction, leading to many projects on low-level type systems and machine-code verification. Eventually, however, it was time to move on and explore something new.

Reflecting on this experience, he remarked that research often follows this pattern. A good idea can open up a rich vein of problems to explore for many years, but eventually that vein runs out, making it natural to pursue new directions.

1.2 Cyclone: Real-World Impact Comes Later

Morrisett then shifted to the development of Cyclone, a system programming language. The project started with Popcorn, a prototype developed by one of his students, which looked like C but compiled down to TAL. Cyclone is a C-like language with safety guarantees, providing region-based memory management and array bounds checking while maintaining good performance. Although the language itself was never widely adopted, many of its ideas later reappeared in Rust. He noted that "really cool research" often influences the real world much later on.

He also reflected that Cyclone involved a great deal of hacking for relatively few papers. He emphasized that building artifacts or research infrastructure can enable future research, while also warned that researchers need to balance the time they spend hacking with the time they spend writing.

1.3 Ynot: Research Takes Unexpected Turns

Morrisett next focused back to higher-level programming languages. He became interested in the methodology of programming with preconditions, postconditions, and theorem proving, leading to the development of Hoare Type Theory. The key idea was to extend Rocq with a monad indexed by preconditions and postconditions, enabling reasoning about effectful programs. This work resulted in the Rocq library Ynot, and the idea was later adopted and converted to reasoning using predicate transformers by F*. Although he had originally expected to build a different version of Standard ML with preconditions and postconditions, the project instead evolved into a Rocq library. Reflecting on this experience, he noted that research often takes unexpected turns, and that academics have the freedom to pursue whichever direction they find most interesting.

1.4 Security

Then, Morrisett grew interested in security. At the time, Google was interested in expanding browser support for running native code, specifically x86 binaries. Javascript, when run in the browser, was very slow and insufficient for running games or visualizations. However, this is quite dangerous from a security standpoint. Their solution was the idea of Software Fault Isolation (SFI), which was a policy on the binary code specifying that it can only execute code or read and write data in particular segments of memory. Google had a checker to make sure that the binary code respected this policy, specifically that code did not read or write or jump outside the specified data region. However, after running a contest to break their checker and twenty teams finding exploits, they realized that the checker had serious bugs. Therefore, Morrisett and his collaborators decided to write a checker and formally prove it correct. First, they built an SFI checker based on regular expressions. Then, they proved the correctness of it with respect to the semantics of x86 binary execution. One challenge concerning the semantics of x86 was dealing with jumps into the middle of instructions. Since the instructions had variable length, one had to guarantee that for every valid parsing, there was no jump outside the segment. Second, they had to build a model of x86 binary execution in Rocq. This took two years to build and an enormous amount of work. However, it did result in a checker that was, one, proven correct, and, two, 20 times faster than Google's original checker.

1.5 Robotic Bees and DNA Computing: Beyond Traditional PL

Morrisett then worked on projects outside of traditional programming languages. NSF had a new program to was interested in tackling the problems resulting from colony collapse disorder. He participated in a project to build robotic bees for crop pollination, focusing on programming swarms of robots. He later also worked on DNA nanotechnology, exploring computation using DNA. He found that working in these new areas was really enjoyable and broadened his perspective. The experience also reinforced his view that the field of programming languages is incredibly broad. PL ideas are particularly well suited to policy and language issues, and they can also be applied across many different domains, including robotics and DNA computing.

1.6 Personal Life and Research

Finally, Morrisett shared his thoughts on balancing research with personal life. One piece of advice he gave was that habits are formed early in one's career. Habits such as exercising, spending time with family, and maintaining friendships, if established early, tend to stay with you throughout your life.

2 Key Takeaways

To summarize, Morrisett presented research as an enjoyable journey. Researchers find a domain, dive deeply into it, work with really smart people, and eventually move on to something new. During those transitions between research areas, it is easy to feel discouraged or burned out. However, these are the times to get your hands dirty, talk to other people, and find new ideas by interacting and socializing. Programming languages is one of the most flexible areas of computer science because PL ideas apply across so many different domains, meaning there are always new and interesting problems to explore.