**Programming with Proofs for High-assurance Systems**

**Nikhil Swamy, Microsoft Research**

From the early days of computer science, the potential of mathematical proofs of programs to eradicate large classes of software errors has been widely recognized; however, until recently, few practical applications were feasible. The proof techniques were difficult to apply to full-fledged programming languages, and proofs were hard to scale to large programs. Oftentimes, the proofs that were done came at the cost of other important aspects of software, such as performance. But today, what seemed impossible just a few years ago is becoming a reality. Project Everest, a collaboration between Microsoft Research (MSR) Redmond, MSR Bangalore, MSR Cambridge, INRIA and Carnegie Mellon University produces fully verified software at a nontrivial scale and with runtime performance that meets or exceeds the best unverified software. We outline a few successes so far and present ongoing work towards our vision for making program proof a tool of choice for building and deploying high-assurance software components.

**F*: A Programming Language and Proof Assistant** F* is a proof-oriented programming language under active development at MSR for the past decade. It is at the cutting edge of programming language design, bringing together various strands of research in foundational logics and type theories with automated theorem proving (building on MSR's Z3 theorem prover), metaprogramming, and functional programming. The design and semantics of F* is documented in more than a dozen research papers at POPL, ICFP and PLDI, the flagship ACM PL venues, with this paper providing a good overview. The practical impact of the foundations of F* is low-level, high-performance code formally proven to be safe, functionally correct, secure running in the Windows kernel and Hyper-V, the Linux kernel, Firefox, and several other popular software packages. F* is also in use at several other organizations, including in the financial sector, at several blockchain startups, including Tezos, Concordium, Zen and others. By this measure, ***verified code produced by F* runs in software used by more than a billion users.***

Two libraries produced in F* are worth special mention.

**EverCrypt** is a high-performance library of cryptographic algorithms implemented in C and assembly and verified in F*. All the algorithms it provides meets or beats the performance of existing unverified code, while also providing guarantees of memory safety, functional correctness, side-channel resistance, and cryptographic security. Verifying crypto code has been studied extensively in the research community, especially in the past decade. EverCrypt provides verified cryptographic implementations at scale, integrating more than 120,000 lines of verified C and 80,000 lines of verified assembly code in a single, industrial-grade library: a one-stop shop for high-assurance modern crypto used in secure communication protocols, including, most prominently, TLS, QUIC, Signal, and many others.

**EverParse:** Improper parsing of untrusted data remains a major source of security vulnerabilities. EverParse is a parser generator for ABI-compatible binary message formats that produces efficient low-level parsers with mathematical proofs in F* of safety, correctness and security. We have successfully applied EverParse to several Windows kernel components, particularly at the attack surface between untrusted and trusted components, ensuring that messages in complex, dense, variable-length message formats, designed for performance rather than safety, are properly parsed and handled. Working in the specific domain of binary message formats is a sweet spot. Programming flaws in low-level parsing code can result in severe security vulnerabilities. Meanwhile, by generating verified

code from declarative specifications, we develop decision procedures that allow us to prove generated parsers correct entirely automatically. Push-button proofs for low-level parsing code is likely to make low-level parsing vulnerabilities a thing of the past, reducing the need for post hoc defensive solutions like low-level fuzzing to find parsing flaws.

**Prospects: Targeted Proven Components in High-assurance Systems** Full system proofs remain extremely labor intensive, and even when such proofs are completed, the wholesale replacement of existing systems with proven variants is incompatible with an incremental adoption strategy. Instead, we aim to enhance high-assurance systems with program proofs for their weakest links. Adding EverParse on the attack surface of hypervisors and OS kernels is a case in point. In other projects, including towards building verifiable distributed applications, we are enhancing a system and architecture of verifiable computing by adding formal proofs to the verifier only, the security linchpin entire system. For ElectionGuard, we are providing verified cryptography and binary message processing, with an eye towards verifying the core protocol ensuring auditability of e-voting.

**Prospects: Embedded Domain-specific Languages for Proof Automation** A second theme is the continued development of programming and proof environments tailored to specific applications and programming idioms, e.g., concurrency, asynchrony, message-passing, protocol state machines, etc. Proof automation for specific DSLs is easier to achieve, while being embedded in a common proof assistant (F* in our case) enables DSLs to interoperate and on a shared foundation. Steel, a new DSL embedded in F* for Rust-like safe low-level, concurrent programming with proofs is one such effort. Vale, a DSL for verified assembly programming in F* is another.