Research Statement

As systems grow more intricate, our ability to ensure their correctness and reliability faces significant challenges. Traditional verification methods, rooted in testing and manual inspection, struggle to keep pace with the exponential growth in system complexity and state spaces. My research vision is to bridge this widening gap by developing innovative approaches that leverage the synergy between type theory, modal logic, and automated reasoning. I aim to contribute to a new generation of programming systems that empower developers to specify, verify, and reason about complex software behaviors with unprecedented ease and precision. Central to my approach is addressing the tension between abstract, high-level specifications that promote clarity and composability, and the low-level implementations necessary for efficiency and real-world integration. To tackle these challenges, I propose the following guiding principles:

1. **Reason about domain-specific constraints:** General-purpose verification techniques often struggle with domain-specific properties. Can we design frameworks that capture and reason about domain-specific constraints while retaining general-purpose abstractions?

2. **Build reusable verification infrastructures:** Efficiently mapping high-level specifications to low-level implementations requires deep domain expertise. Can we centralize this expertise by building reusable infrastructures for verification?

3. **Automate reasoning via canonical representations:** Manual proofs are time-consuming and error-prone. Can we design canonical representations of program behaviors that enable automated reasoning and proof generation?

My research has explored these principles across several domains, developing practical frameworks for specifying and verifying complex system behaviors. In the domain of network protocols, I developed a type-preserving compilation strategy for Software-Defined Delay-Tolerant Networks (SDDTNs) [1]. This work introduced NetQIR, an intermediate representation designed to facilitate type-preserving compilation from high-level languages like P4 to a formally verified representation. By leveraging Network Calculus and algebraic methods, we proved key properties of SDDTNs, including packet delivery correctness and delay bounds. The NetQIR type system ensures that well-typed programs inherit guarantees provided by Network Calculus, bridging the gap between theoretical foundations and practical implementations.

In the realm of gradual verification, I contributed to the development of Gradual C0, the first gradual verifier for recursive heap data structures [2, 3]. This work addresses the challenge of incremental specification and verification, allowing developers to smoothly transition between static and dynamic checking. We designed a symbolic execution algorithm capable of handling imprecise specifications, producing minimally sufficient run-time checks for soundness. To evaluate the implementation's correctness, I developed a property-based testing framework that empirically assesses soundness by establishing a truthiness property of equivalence between Gradual C0 and a fully dynamic verifier [3].

Exploring the intersection of formal methods and blockchain technology, I extended our gradual verification approach to smart contracts [4]. This work introduced a prototype for gradually verifying Algorand smart contracts via the pyTEAL language, demonstrating how gradual verification can enhance security, guarantee soundness and flexibility, and optimize resource usage in smart contract interactions.

My research in Kleene Algebra with Tests (KAT) focused on verifying the correctness of parallel composition in hardware description languages [5]. We developed a set of axioms for concurrent

KAT and used them to prove properties of parallel programs. This work provides a foundation for reasoning about concurrent hardware designs using algebraic methods.

These projects demonstrate my ability to apply formal methods across diverse domains, but more importantly, the common thread throughout this work is the development of practical, algebraic/type-based approaches to verification that bridge the gap between high-level specifications and low-level implementations.

Going forward, I am excited to pursue research that builds upon these experiences, focusing on the intersection of type theory, automated reasoning, and practical verification. Specifically, I aim to:

1. **Design new modal type systems:** Building on my experience with temporal logic, I will explore how modal type systems can capture and verify temporal properties of distributed systems and other domains where traditional type systems fall short.

2. **Develop automated reasoning techniques for domain-specific properties:** Extending my work at NASA, I will create new automated reasoning techniques tailored to specific domains like network protocols, concurrent systems, and security-critical applications. This will involve designing domain-specific logics, decision procedures, and proof tactics that can be integrated into existing engineering workflows.

3. **Create reusable verification infrastructures:** Inspired by the Calyx project and my work on Gradual Verification, I will develop modular, reusable verification frameworks that can be adapted to various domains. These infrastructures will aim to centralize verification expertise, making it easier for non-experts to apply formal methods to their specific problems.

4. **Bridge the gap between formal methods and engineering practice:** Addressing the disconnect I observed at NASA, I will work on tools and methodologies that make formal verification more accessible to practicing engineers. This includes the automatic translation of informal requirements to formal properties through automated reasoning.

The next era of computing will require dramatically more reliable and verifiable software systems. I am excited to apply ideas from programming language theory to build tools that make formal verification accessible and practical for a wide range of critical applications.

References

[1] J. Ramos-Davila and A. E. Goodloe. Type-Preserving Compilation for Formally Verified Software Defined Delay-Tolerant Networks.

[2] J. DiVincenzo et al. Gradual C0: Symbolic Execution for Efficient Gradual Verification.

[3] J. Ramos-Davila. Evaluating Soundness of a Gradual Verifier with Property Based Testing.

[4] H. Sun, K. Singh, J. Ramos-Davila, J. Aldrich, and J. DiVincenzo. Gradual Verification for Smart Contracts.

[5] J. Ramos-Davila. Calyx + HardKAT: A Verified IR for Calyx.