My research interest lies in automated software engineering, focusing on improving developer productivity and code quality. Specifically, my work centers around program analysis and synthesis, in conjunction with software architecture, security, and formal methods. With the current trend in LLM-based software engineering, we expect a resurgence of formal approaches for verifying generative AI outputs. Building upon my previous and current research, my future work will blend formal methods with ML-supported program analysis and synthesis to generate formally verified results from incomplete specifications.

Previous Research Experience

During my bachelor's and masters studies, I acquired a foundation in software engineering theories. Over fourteen years in the software industry, I gained practical experience and valuable insights into the challenges faced by software engineers. This ranged from delays in the software production process to mismatches between desired specifications of the software and actual implementations. These experiences fueled my passion for developing effective solutions to help programmers create correct and trustworthy software. Throughout my Ph.D., I explored various aspects of automated software engineering, including program analysis and comprehension, repository mining, API recommendation, formal methods, and, significantly, program synthesis.

Software Security: In the area of software security, I focused on identifying legacy security flaws from a software architecture perspective. This involved exploring software quality attribute traceability in legacy systems, software certification, and security tactics detection [3]. I also delved into the use of ML models for detecting security issues in the absence of correct usage of architectural tactics [1].

Program Analysis: Moreover, I concentrated on addressing challenges in code quality and correct implementation through techniques such as dataflow and control-flow analysis. My approach involved inter-procedural analysis, identification of incorrect tactic implementations, and providing API recommendations for fixing these issues [7, 6]. Furthermore, to address an important issue of the state-of-the-art call-graph constructors when facing dynamic features of programming languages, we developed an approach that can create sound call-graphs for object serialization [2].

Program Synthesis: Advancing to program synthesis, I developed an inter-procedural approach that goes beyond providing API recommendations. This approach overcomes the challenge of synthesizing inter-related, and yet discrete, code snippets that implement a scenario and automatically integrating it into an existing code base [4, 5]. This effort was recognized as the first-place award winner at the *ASE21* research competition. This research elaborates the importance of (semi) formal synthesis, even in the era of blooming of ML-based automated programming.

In addition to my Ph.D. research, I undertook two research internships at Google and Palo-Alto Research Center (PARC), contributing to projects related to program analysis and software synthesis, with a focus on ML-based generative models and evolutionary search-based synthesis. These efforts resulted in two inventions (pending patents) and two papers (to be submitted).

Current Research

Expanding my research to formal methods, I believe this domain will become crucial due to the widespread use of ML-based techniques generating unverified outputs. As a postdoctoral researcher, I am currently investigating formally verified pointer analysis and formal-based software compartmentalization. The former aims to perform binary analysis, identify pointers, and formally prove their attributes, while the latter focuses on leveraging formal methods to mitigate CVE exploits via software compartmentalization.

Future Research

Based on my past work, knowledge, and research experience, I plan to pursue three main lines of research in the future.

Formally Verified ML-supported Inter-procedural Program Synthesis: The landscape of automated programming and program synthesis holds great promise, with various branches of the field gaining attention. However, I identified a significant gap in supporting programmers in real-life scenarios. Most existing approaches focus on intra-procedural code synthesis, where a synthesizer generates a block of code based on a given specification. In contrast, real-life programming demands inter-procedural code synthesis capable of constructing interrelated blocks added to different parts of the program for specific use-cases or scenarios.

During my Ph.D. research, I developed an approach to inter-procedurally implement tactic code snippets and seamlessly integrate them into the program; this represents a semi-formal approach. This involved addressing the challenges of real-life programming scenarios. Presently, my focus is on advancing this work with pure formal method-based approaches. I am developing methods that leverage formal techniques to enhance the precision and reliability of code synthesis.

Looking ahead, I envision leveraging the synergy of formal methods and language models in two key directions: (i) generating required specifications from under-specified design/specifications and (ii) establishing a cycle of code generation and formal verification. This involves extracting precise specifications from loosely defined requirements, ensuring a robust foundation for subsequent code generation. Furthermore, I am exploring the integration of language models to enhance the efficiency of the code generation process.

Drawing on the knowledge and experience gained during my research internship at Google, specifically in LLM-based synthesizers, I aim to make this approach fully functional in the next 5 years, bridging the gap between formal methods and practical, real-world software development.

Large-scale Code Repair: Building on my Ph.D. research, I am set to embark on a hybrid solution that merges formal and ML-based approaches for large-scale code repair. This initiative addresses the limitations in existing state-of-the-art code repair methods. These methods often demand explicit and highly specific input specifications, making them less robust in accommodating variations in a given code base.

Continuing from my prior work, I aim to develop a hybrid solution that seamlessly combines the precision of formal methods with the adaptability of ML-based approaches. This endeavor involves identifying sections of code in need of repair, automatically generating the required patches, and seamlessly integrating these patches into the existing codebase. Importantly, this approach extends beyond mere bug identification and repair at the method level, offering a comprehensive solution for large-scale code maintenance and enhancement.

Self-reconfigurable Software: Another direction for my research over the next 5-10 years is the development of a framework enabling self-reconfigurable software systems. While existing efforts focus on separate aspects like API migration, code repair, and ML-based code transition, I believe there's potential to consolidate these endeavors into a unified framework.

My vision is to create a generalized approach that learns from various software evolution examples, offering a holistic solution to software self-reconfiguration. This framework aims to facilitate automatic updates in software systems based on their evolving environment. By integrating insights from API migration, code repair, and ML-based code transition, I anticipate establishing a unified methodology that streamlines software evolution.

This direction aligns with the overarching goal of creating adaptive, self-evolving software systems that can seamlessly and autonomously adapt to changing requirements and environmental conditions. As I delve into this area, I anticipate contributing to the broader landscape of software engineering and paving the way for more resilient and dynamic software systems.

Research Grant Proposal

In addition to contributing to grant proposals for NSF, DARPA, and DoE during my Ph.D., I played a crucial role in submitting a substantial $7M DARPA grant proposal (BAA number HR001123S0039) where I served as the main Principal Investigator (PI). This collaborative effort involved partnerships with other esteemed universities such as the University of Notre Dame and prominent international research institutions like SRI.

My experience in writing grant proposals has provided me with valuable insights into effectively articulating research objectives, methodologies, and expected outcomes. Leading the DARPA proposal showcased my ability to navigate the complexities of interdisciplinary research, fostering synergies between institutions and researchers.

As I transition into the next phase of my academic journey, I am eager to continue contributing to the writing of grant proposals for different funding agencies. Through these proposals, I aim to secure support for innovative research projects, leveraging my expertise in automated software engineering, formal methods, and the integration of ML-based approaches into software development. Collaborating with colleagues and researchers from diverse backgrounds, I seek to lead impactful research projects that address critical challenges in the field.

References

[1] A. Okutan, A. Shokri, V. Koscinski, M. Fazelinia, and M. Mirakhorli, “A novel approach to identify security controls in source code,” *arXiv preprint arXiv:2307.05605*, 2023.

[2] J. Santos, M. Mirakhorli, and A. Shokri, “Sound call graph construction for java object deserialization,” *arXiv preprint arXiv:2311.00943*, 2023.

[3] J. C. Santos, A. Shokri, and M. Mirakhorli, “Towards automated evidence generation for rapid and continuous software certification,” in *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 2020, pp. 287–294.

[4] A. Shokri, “A program synthesis approach for adding architectural tactics to an existing code base,” in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 1388–1390.

[5] ——, “Inter-procedural program synthesis for automatic architectural tactic implementation,” Ph.D. dissertation, Rochester Institute of Technology, 2023.

[6] A. Shokri and M. Mirakhorli, “Arcode: A tool for supporting comprehension and implementation of architectural concerns,” in *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. IEEE, 2021, pp. 485–489.

[7] A. Shokri, J. C. Santos, and M. Mirakhorli, “Arcode: Facilitating the use of application frameworks to implement tactics and patterns,” in *2021 IEEE 18th International Conference on Software Architecture (ICSA)*. IEEE, 2021, pp. 138–149.