Elizabeth Dinella: Research Statement

My research interests are in the fields of **Software Engineering** and **Machine Learning**. Broadly speaking, my work focuses on improving software correctness through synergistic machine learning techniques. This is of upmost and timely importance as Large Language Models (LLMs) have shown remarkable results in code generation tasks, but struggle to provide interpretable, faithful, and ultimately, trustworthy responses. Prior to breakthroughs in machine learning, productively writing correct and secure code for software projects has remained a significant challenge. As such, program analysis has been an active research area for many decades producing fruitful techniques based on rules and formal logic. Despite their successes, these symbolic approaches have some noteworthy limitations in accuracy, flexibility, scalability, and ease of use. Ultimately, my research objective is to combine program reasoning tools and language models to achieve **Cooperative Program Reasoning and Neural Modelling**. By integrating these classes of tooling, I seek to address the shortcomings of both LLMs and symbolic program reasoning tools. This research endeavors to establish a symbiotic relationship to capitalize on the strengths of both paradigms.

**Neural Inference of Specifications**

In my PhD research, I showed that inferring specifications of correctness can overcome fundamental roadblocks in program reasoning. **My work has been published in top tier software engineering and machine learning conferences receiving Spotlight and Distinguished Paper awards, over 280 citations, patented, and deployed in industrial systems.** My research contributions are a fundamental shift from the traditional symbolic program reasoning paradigm. A statistical paradigm is desirable as symbolic program analysis tooling suffers across multiple dimensions due to their rigid rule based nature. Since program analysis is fundamentally an undecidable problem, symbolic approaches must carefully balance tradeoffs to achieve an accurate and scalable approach. Often times, such approaches leverage heuristics for program correctness which can sacrifice accuracy. As an alternate approach many analysis tools provide an option for including a human-in-the-loop to achieve a higher degree of accuracy. However, requiring human interaction hinders ease of use and full automation. In general, symbolic approaches must balance a tradeoff between accuracy and including a human-in-the-loop.

My research asks the question: **Can we build program reasoning tools without a human-in-the-loop while maintaining high performance?** In my PhD work, I have shown that the necessary tradeoff between accuracy and human interaction can be eliminated through neural inference of specifications. I have explored this in a diverse set of program reasoning including static bug finding and repair, program merge, and automated test generation.

In the static bug finding and repair domain, I advanced the state-of-the-art for JavaScript programs resulting in a first author spotlight publication: Hoppity [1]. Effective approaches in static bug finding and repair are challenging due to 1) a lack of program specific correctness properties and 2) complex real world programming constructs with potentially unavailable
source code (e.g., API / framework calls). Through an end-to-end neural approach Hoppity addresses both challenges. Firstly, a correctness checking tool can only find bugs it has been specifically engineered to find. For an arbitrary program, there is no clear cut definition of what constitutes an error. Symbolic tools typically default to a set of handwritten universal correctness rules: (e.g., NullPointerException should never occur). These heuristics are not ideal since developers often want to check for program specific rules of correctness: (e.g., "You must be logged in before posting to a blog application"). In contrast, Hoppity learns latent correctness properties through a graph neural network trained on corpora of code commits. Such a statistical method does not rely on rigid correctness rules and is capable of finding program specific functional bugs. In regards to the second challenge, the neural method underlying Hoppity is inherently flexible to complex programming constructs and can scale to projects where the entire source is not available. In an evaluation on 30 real world bugs, Hoppity was able to detect and repair 5 bugs, while the leading symbolic tool was not capable of detecting any due to the aforementioned challenges.

In the program merge domain, I made significant contributions resulting in a first author publication: DeepMerge [3], a 10x improvement over state-of-the-art tooling, and successful development as a Microsoft internal product. In general, program merge suffers the same challenges as static bug finding: a lack of program specific correctness specifications. The most widely used symbolic approach for program merge is the 40 year old diff3 algorithm underlying Git Merge. When a conflict occurs, this tool totally forgoes heuristics for correctness and requires developers to manually construct a resolution. Manual merge conflict resolution is a significant barrier to software development in teams and stalls pipelines to production. In my work on DeepMerge, I contributed the first formulation of merge conflict resolution as a neural modelling problem. When a conflict is detected, DeepMerge performs the resolution task that humans are typically required to perform. To learn the notion of a correct merge resolution, my contributions include a neural encoder-decoder framework and a dataset of merge conflicts and resolutions which it is trained over. As a key innovation to improve performance, I developed a novel input representation unique to program merge. Overall, in an evaluation on real world conflicts, DeepMerge achieved a 10x improvement over state-of-the-art resolution tools. Our follow up work MergeBert [5] improves upon DeepMerge’s accuracy by leveraging a pretrained large language model. My work has been a successful patented tech transfer within Microsoft and will be released as an external tool in the coming year.

In the automated testing domain, my first author work on TOGA [4] received a distinguished paper award at ICSE 2022 for my contributions in neural test oracle generation. Automated testing also suffers from the common program reasoning challenge: a lack of program specific correctness specifications in the form of test oracles. A test oracle is a description of the expected output on a given input. Without effective test oracles, automated testing often gives inaccurate results in the form of both false positives and false negatives. By framing test oracle generation as a neurosymbolic technique leveraging both coverage guided testing tools and pretrained LLMs, TOGA is able to detect a variety of program specific bugs. On a benchmark of real world faults, I significantly advance the state-of-the-art in testing by finding 57 bugs using our inferred specifications in contrast to 20 bugs by the next best approach. By inferring function specific oracles of correctness, our work is capable of finding bugs which are not captured by rigid universal rules of correctness.
Future Work

In my future research, I am excited about broad synergies of program reasoning tools and language models that hold immense potential for enhancing software correctness and programmer productivity. My goals can be categorized as follows: Deep learning to assist program analysis tooling and Symbolic tooling to assist neural models.

Deep learning to assist program analysis: In my PhD work, I have explored this direction, but exciting work still remains. The fundamental challenge of balancing human intervention and accuracy is prevalent in nearly every program reasoning domain. Regardless of domain, neurosymbolic methods for program reasoning have room for improvement. Can we develop general purpose tooling rather than approaches custom for each reasoning domain? Can we infer interpretable specifications of correctness? My early effort toward interpretable specifications of correctness have shown promising results [2]. In the near term, I am excited about working toward a general purpose neural specification inference library to address the scalability and ease of use limitations of program analysis tools.

Symbolic tooling to assist neural models: Neural models, particularly LLMs, have achieved remarkable successes in programming tasks. However, they struggle to understand program semantics, are not robust to semantic preserving transformations, often hallucinate, and tend to provide incorrect yet confident responses. My future research asks, can we exploit symbolic techniques to develop approaches with interpretable outputs? Can we provide formal guarantees on a model’s outputs? This nascent domain will require rigorous experiments to evaluate current approaches on fair benchmarks, observing its failures and successes. I am eager to explore this quickly developing research direction. My early efforts in this direction show promise in exploiting symbolic tooling to address fallbacks in neural modelling for code reasoning tasks.

There is growing interest in both these future research directions from both industry and federal funding programs, acknowledging the importance of improving both program reasoning and neural methods. The NSF Software and Hardware Foundations (SHF) program supports software engineering research and encourages joint synergies in areas such as machine learning. DARPA also provides funding for such research directions. In particular, the Intelligent Generation of Tools for Security (INGOTS) program supports techniques driven by program analysis and artificial intelligence. Lastly, industry programs such as Amazon’s automated reasoning award provide funding for research at the intersection of software engineering and machine learning. My prior research and knowledge in these domains, as well as my experience writing grant proposals and securing funding makes me uniquely positioned to tackle these difficult challenges.

In general, success in my research agenda will improve software development efficiency, quality, and correctness. Grounded in my prior research, I believe there are huge opportunities in integrating program reasoning tools and large language models. Effective work in these directions will lead to more trustworthy, scalable, and accurate neurosymbolic approaches. This fusion of advanced technologies is timely, important, and will have far-reaching implications for diverse industries and domains.
Publications


