Application - IBM Fellowship 2018-19

Shashank Srikant

shash@mit.edu

https://shashank-srikant.github.io/

Ph.D. thesis title.

Choices

- Data-driven program analysis - A case for scalably detecting security vulnerabilities
- Towards a comprehensive understanding of developing and understanding computer programs (slightly generic; this can accommodate neuro stuff)

Expected date of graduation. 2021. Started Ph.D. program in Fall, 2017.

Areas of technical interest.

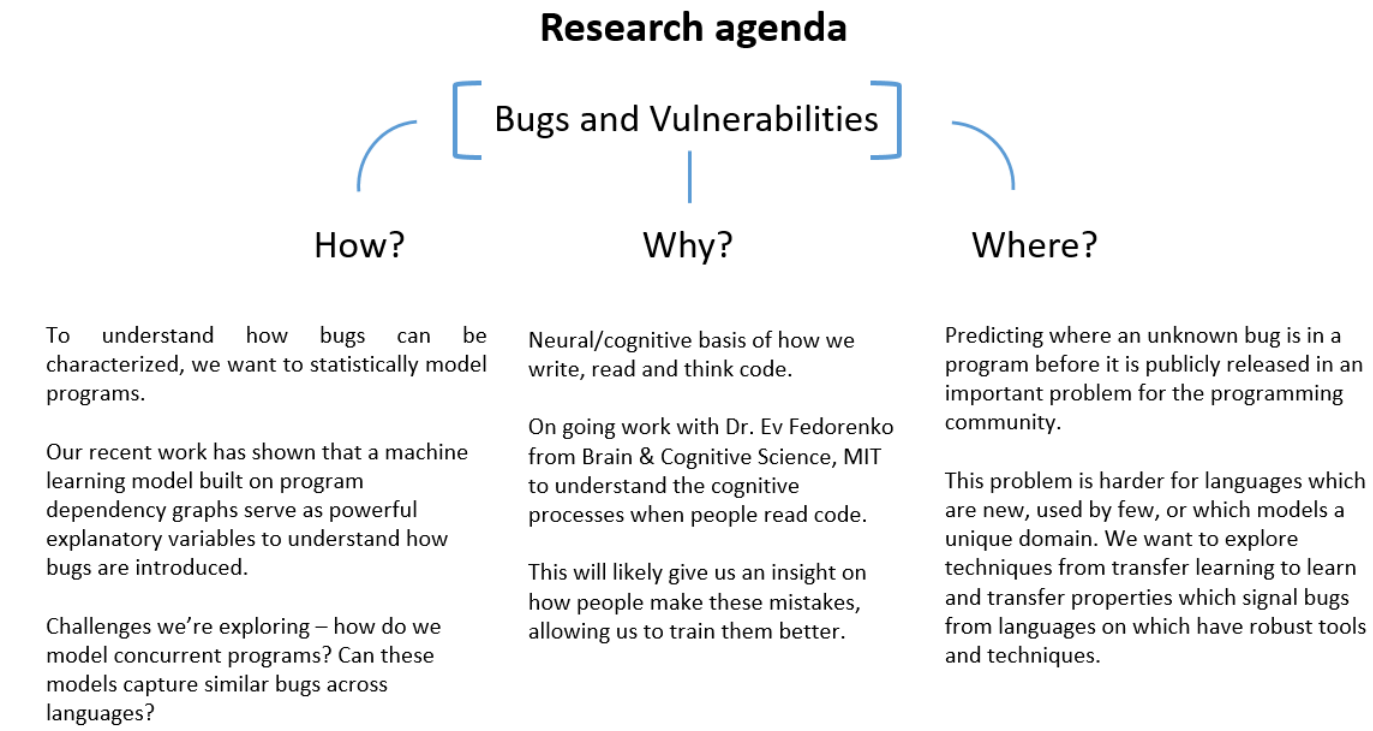
- AI/Cognitive science
- Security
- Blockchain

Primary area of research interest. Machine learning, Program analysis.

Research summary.

The surge of innovations owing to machine learning and data science is a recent phenomenon. Its impact today is mostly on problems in text, images, and sound. I am interested in exploring a rather different, yet equally prevalent domain - computer programs. I'm most excited to build a robust framework to examine computer programs statistically. Having such statistical models can help design a number of very useful technologies like scalably translating one language to another, topic modeling in programs and associating code reviews written in a natural language to constructs in a programming language. I believe these will seem obvious tools to have in the days to come, like Google's Maps and Translate are today. Such models will help us reason about programs better, and will eventually make programming easy to learn, understand and teach.

One specific property that I am interested in understanding from such a statistical viewpoint is the presence of bugs and vulnerabilities in programs. Knowing where a bug is likely present in a program before it is shipped out is of utmost relevance to the entire programming community. State of the art tools and techniques to detect bugs are *soundy*, in that they suffer from precision and recall issues. We see a data-driven model based prediction of such bugs and vulnerabilities as a way of strengthening traditional program analysis techniques which have been developed for years now.



Keeping bug and vulnerability analysis as a central theme, I want to ask three questions related to it - how, where, and why. I want to explore how, in a statistical sense, buggy programs can be characterized; where they are likely to appear in an unseen program; and why we humans really create them in the first place. Each of these questions provide multiple directions of research to explore, all of which promise to have ramifications on the entire programming community. I elaborate on each of these three questions below.

How to characterize bugs. I want to understand how a snippet of buggy code can be characterized - properties the snippet possesses in relation to the rest of the program it is surrounded by, and in relation to other non-buggy programs, which can help uniquely characterize it.

To answer such a question, we need to represent programs in a way that preserves as much of its semantic properties. Such information can stem from various aspects of a source code. From characters appearing in the raw text of the source code, to values which the programs variables take when tested on different inputs, each provides unique information about the programs functionality. The very few related works that have dealt with program representations have approached its design in two broad ways - they either treat programs as a bag of tokens, or they pass programs in a deep network expecting it to learn the right features. I believe these approaches represent extremes. Token and surface level information fails to model complex tasks, while expecting a deep learning model to infer complex variable interactions demands an inordinate magnitude of program samples. In our recent work [**?**], we presented a way to extract semantically rich features which captured complex variable interactions. Using these rich features, we built machine learning models of programs written in Solidity, the language used to define smart contracts on *Ethereum*, and demonstrated its ability to predict four very diverse classes of vulnerabilities. Our engineered features enabled us to build model using a few thousand samples, demonstrating their applicability to low-data settings. We also performed a qualitative analysis of the features selected by these models and showed how their high interpretability allowed for a causal understanding of the accuracy of our models, and helped discover

subtle issues in state of the art analysis tools.

Through this work, and my other previous works on representations[], we show we can characterize program faults. Extending this idea, I am particularly interested in modeling concurrency-related bugs. Most real-world systems today, including contracts written on blockchains, involve concurrency, and are notorious for having bugs which manifest in specific concurrent behavior. Detecting these issues through such a data-driven approach would greatly benefit professional software development.

Preempting where bugs would occur. A constant challenge which security researchers face is discovering the existence of new bugs or unfamiliar variants of known bug-classes in programs. This is exacerbated for languages which are new, used by a few, or which models a unique domain. Informed by our work on designing representations of programs which capture rich semantic information, a promising approach to solving this problem is to learn statistical models of bugs in languages where they are well documented (such as Java, C++), and transfer these models to languages where data is scarce. This poses interesting research questions such as how we would account for differing language properties, how we can truly generalize program properties to match bugs expressed in different languages, etc. This is work we have recently begun investigating.

Why do we create bugs. A fundamental question which supersedes the above two questions is why we create bugs. Is there a cognitive mental model of a problem which we devise, which when incorrect, leads to bugs? The cognitive and neuroscientific aspects of how we write programs has not been addressed by academia. I want to explore what parts of our brain are involved in reading and writing programs. And in understanding this, I also want to specifically study what leads to a misunderstanding in concepts, which results in bugs. To this effect, we have begun a collaborative study with Dr. Evelina Fedorenko, Assistant Professor at Harvard Medical School and Research Affiliate, MIT, in understanding what parts of the brain are responsible for comprehending programs. This study is a first of possibly many which can systematically unravel what regions of the brain are involved when writing programs, and what conditions cause bugs to occur. This line of enquiry however brings with it operational hurdles - in our research so far, we have learned that current imaging infrastructure does not allow people to write programs while having their brains being imaged. This opens up an exciting area of engineering to design apparatus that would allow programming in an MRI, which would enable us to understand these compelling scientific questions.

To summarize, I see my research agenda spanning the general question of why we introduce bugs, and how a data-driven perspective can help detect and discover them scalably.

Other project details on webpage: https://shashank-srikant.github.io/