ThanhVu Nguyen: Research Statement

My work is at the intersection of Software Engineering (SE) and Programming Languages (PL). More specifically, I focus on the core program analysis areas of verification through *invariant analysis* and synthesis through *automatic program repair* (APR).

Highlights Since joining UNL in Fall 2016, I have published 15 refereed papers at top-tier SE/PL journal and conference venues (2 OOPSLA, 3 ICSE, 2 FSE, 1 ASE, 1 PLDI, 1 TACAS, and 5 short conference/workshop papers). I also received **two "Most Influential/Impact" (MIP) awards**: ACM SIGSOFT/IEEE 10-year MIP at the International Conference for Software Engineering (ICSE) and ACM SIGVO 10-year MIP award at the Genetic and Evolutionary Computing Conference for my work on applying genetic programming to repair programs[1]. My APR work creates the widely popular subfield of program repair in SE/PL and has been deployed in the industry and government labs (e.g., used at GrammaTech, Facebook). In particular, Facebook has adopted my repair technique and heuristics to find and fix bugs in their Android and iOS apps.

Currently, I have 3 grants: NSF CRII (sole PI, $175K, 2020–2023) on applying symbolic and dynamic techniques to analyze the Linux kernel's build process; Army Office of the Research (PI at UNL, $550K, my portion $160K, 2019–2021) on predicting and adapting to program failures; and UNL Faculty Seed Award (sole PI, $10K, 2021) on analyzing smart and configurable systems and devices (e.g. Android).

1 Research

Invariant Discovery I develop automatic techniques to discover *program invariant*, which are specifications and properties that are guaranteed to hold at interesting program locations, e.g., post-conditions at function exits and loop invariants at loop entrances. Generated invariants can be used to understand undocumented programs, prove correctness assertions (e.g., Hoare Logic), reason about program termination, resource usage, establish security properties, provide formal documentation, etc.

I have developed the DIG framework, which uses a *data-driven* (i.e., ML) approach to learn useful program invariants in complex programs. To support expressive invariants, e.g., nonlinear polynomial properties often required scientific or cyber-physical systems, DIG interprets nonlinear polynomial formulae as convex geometric objects in high-dimensional space, such as hyperplanes and polyhedra. This representation allows DIG to employ mathematical techniques, e.g., equation solving and convex hull constructions, to dynamically discover conjunctive polynomial invariants. To avoid spurious results, DIG uses the guess-and-check approach [**?**, **?**] to infer invariants: it encodes the semantics of programs as symbolic constraints and use them to generate candidate invariants and also to check or refute, and iteratively refine, those invariants. DIG also employs a custom theorem prover based on k-inductive SMT theorem proving to removes spurious results and produces only true invariants [**?**].

My invariant techniques are used for to infer and analyze many important classes of invariants, e.g., nonlinear numerical relations [**?**, **?**, **?**], array and heap properties [**?**, **?**, **?**], termination proper-

[1]MIP awards are given to the author(s) of the paper from the conference meeting 10 years prior that is judged to have had the most influence on the theory or practice of the field during the 10 years since its original publication. Recipients are invited to give a keynote speaker and given engraved plaques.

ties [?], algebraic specifications for modelling library calls [?], and interactions among configuration options [?, ?, ?]. DIG's invariants are also used for security analysis, e.g., to check the correctness of AES (Advanced Encryption System) [?, ?] and detect potential side-channel complexity attacks [?, ?]. DIG's nonlinear benchmarks are included in SV-COMP (the annual international competition on software verification.

Program Repair. I also work in program synthesis through the form of automatic program repair (APR), which is the problem of modifying a buggy program to meet given specifications. My work on GenProg uses a genetic algorithm to satisfy multiple program repair objectives, e.g., it must pass a given testsuite, is similar to the original buggy program, and (preferably) easy to understand [?, ?, ?]. To reduce search space, GenProg reuses code from non-buggy portions of the program when generating candidate repairs. GenProg is the first APR work to show that computers can automatically repair real-world large programs (million lines of code).

I also developed theoretical work connecting the problems of program synthesis and reachability [?], i.e., test-case generation, are equivalent under certain cases, i.e., one problem is reducible to the other. These results directly connect the two seemingly different fields, enabling the application of existing test-case generation techniques, such as fuzz testing and symbolic execution, to program synthesis and, thus, bug fixing.

Recently, I have applied APR to domain-specific languages. My student and I have developed powerful techniques to localize, analyze, and fix errors in specifications and models written in the Alloy language [?, ?, ?]. We also have applied for patents on the Alloy works.

Impacts. The original DIG work received the Distinguish Paper Award at ICSE'12 and since then has produced more than 15 refereed, top-tier conference and journal papers on invariant technologies built on top of DIG. During a DARPA'16 demo, DIG was used to demonstrate that it can capture precise program runtime complexity and hence detect side-channel attacks[2]. Recently, companies such as GammaTech[3] have integrated DIG in their program analysis tools, raising opportunities for additional industrial adoptions.

GenProg has received multiple awards, e.g., best papers at ICSE, GECCO, ICST, and the ACM SIGEVO "Humie" gold medal, and my APR works have been cited more than 2000 times[4]. In 2019, the original GenProg papers [?, ?] were recognized with a 10-year Most Influential Paper award at ICSE and a 10-year Most Impact Award at GECCO for showing that real-world large programs can be repaired automatically and creating the now widely popular APR subfield in SE. My APR works also have been used in the industry, including GammaTech, MIT Lincoln Lab, etc. In particular, Facebook has adopted the search technique and heuristics in GenProg to automatically and "intelligently" locate and repair bugs in their Android and iOS apps [?, ?].

[2]This demonstration, which shows a rather surprising powerful feature of nonlinear invariants in representing complex program complexity, helps renew the DARPA contract and results in an FSE conference paper [?] in 2017.

[3]GammaTech has a website and videos showing how DIG is used in their project: https://grammatech.gitlab.io/Mnemosyne/docs/muses/.

[4]Google Scholar `https://scholar.google.com/citations?user=TLcVQ-MAAAAJ&hl=en`.

2 Current and Future Research

I have been applying dynamic invariant generation to analyze highly-configurable systems, which have many possible configuration options and behaviors. I am also exploring ways to apply invariant generation and APR to solve problems in emerging domains such as IoT and ML-generated code.

Highly-Configurable Systems I have developed GenTree, a lightweight ML-based, dynamic analysis technique to automatically discover program "interactions", which are invariants showing how configuration option settings affect a program's functionality, performance, etc. GenTree employs an iterative stochastic algorithm that works by running a program under a set of configurations, capturing execution traces, and applies machine learning algorithms on traces learn and refine invariants over configuration settings that map to interested system behaviors [**?**, **?**, **?**]. My current NSF CRII grant is on extending this work to combine static and dynamic techniques to analyze systems such as the Linux kernels with an extremely large number of 2^{13000} configurations.

DNN's I am working with collaborators at UNL and UVA on an invariant generation project to understand and verify ML-generated programs, such as deep neural networks (DNNs). Recent years have witnessed many new exciting works that apply traditional verification techniques to analyze DNNs. Many of these works have suggested that invariants are necessary to solve challenges in analyzing ML-generate code (the same way invariants are used to help analyze and prove programs). I am working on using DIG's invariants to represent complex, nonlinear activation functions and their patterns to help DNNs analysis and understanding (e.g., explainable AI).

IoT Systems I am collaborating with researchers from Michigan State and Facebook on analyzing the interactions in IoT systems, which compose of multiple configurable devices and apps (e.g., smart alarms, doors, phone, watches). I believe that the counterexample-based, iterative framework developed in DIG and GenTree, which use dynamic analysis and *do not* require source code, can improve the scalability and accuracy of IoT analyses. Moreover, my APR works can help locate and even synthesize rules or patches to avoid vulnerabilities and unknown behaviors among the interactions of these complex IoT hardware and devices. My recent UNL Faculty Seed award is on finding and fixing bugs in IoT interactions.

3 Conclusion

More broadly, I am interested in investigating novel techniques to improve software reliability, more specifically in the directions of program analysis, verification, and synthesis/repair. I plan to improve existing and create new program analysis techniques in program repair and invariant inference to help improve the quality of traditional programs as well as new systems in domains such as those in IoT and ML. I believe these works are practical, impactful, and can generate exciting interdisciplinary collaborations and more funding opportunities.