RESEARCH STATEMENT

Teng Zhang (tengz@seas.upenn.edu)

Software systems keep increasing in scale and complexity, requiring ever more effort to design, build, test, and deploy. Although static verification technology has been evolving greatly, it is still extremely hard to realize end-to-end verification of software systems. My research aims to develop theories and systems to improve quality of software systems at different stages of software lifecycle. To achieve this goal, I have conducted research spanning the areas of runtime verification, static verification and model-based development. Research projects I have led and participated in not only make me accumulate the skill set but also equip with me an insight for a long-term goal in this field.

Thesis research on runtime verification

Runtime verification (RV) and runtime enforcement (RE) are promising technologies to improve robustness of software by detecting software faults or attacks and further generating adaptation actions. However, applying them into real-world software, especially large-scale systems, is still a challenging task from multiple perspectives such as trust, expressiveness, usability and performance. In my dissertation work, we proposed a RV-enabled framework for self-adaptive software [6, 5]. The core of this framework is SMEDL, a domain specific language for property specification. The state-machine-based style language for single monitors allows users to describe both high-level temporal properties and low-level properties involving imperative actions conveniently. To support component-based systems naturally, the SMEDL specification can be modeled as a set of connecting monitors. Users can flexibly specify how monitors are deployed to balance performance and overhead. The notion of monitor network provides a powerful and unified way to specify different properties and facilitates flexible deployment of monitors to adapt to a variety of software systems. Dynamic instantiation of monitor instances is suitable for monitoring large data set offline or scalable software systems online. To guarantee the implementation of monitor code strictly follows the intention of the specification, we further formalized the semantics of SMEDL in Coq and refined the relational semantics into executable Haskell programs using Fiat framework [7].

To further improve robustness of the system in reaction to the internal failure or environment change dynamically, we extended the SMEDL framework to support runtime software adaptation based on runtime enforcement. An enforcement mechanism can affect execution of the system, which requires a strict validation for its correctness. One particular issue that has not been addressed is that a poor implementation of an action may interfere with the progress of the application. We presented a framework for reasoning about an implementation of enforcement actions with respect to functional correctness of the target program. We proposed a method to update the proof locally by generating new proof obligations around the instrumentation point and establishing a logical implication between the original proof obligations and ones generated in the instrumented program. Our method can be scaled up to support multiple instrumentation points.

Exploration of usability of software verification tools

Software verification technology has been gaining more attentions in industry. For instance, the bounded model checker CBMC is used to check undefined behaviors at Amazon. Applying deductive techniques to verify functional correctness, on the other hand, is more challenging because it requires a more rigorous and detailed specification which needs to be transformed into logic formula. Moreover, most academic tools only support a subset of the language. Specifying properties using the tool usually requires modification of the source, which is unacceptable. As the first step to bridge the gap between the academic tool and real-world application, I worked on a project to apply Prusti, an existing deductive verification tool for Rust to code base at Amazon. We chose the circular queue of the virtio interface, serial and i8042 keyboard controller in the Firecracker code base as the verification target. After overcoming difficulties on the property design and limitation of the tool, we successfully encoded meaningful properties and detected an off-by-one error in the circular queue and an overflow bug in the keyboard controller. Based on our experience, we proposed suggestions for improvement to the development team of Prusti.

Formal verification based on model transformation

Safety critical systems require more rigorous design and development process. Model-based development has been widely used. During my bachelor and master, I had been working in two projects using model-based technology to verify or simulate behavior of safety critical systems. One was verification of AADL (Architectural Analysis and Design Language) models by transforming into Timed Abstract State Machine (TASM) and UPPAAL timed automata model. I was responsible for implementing the concrete transformation rules from AADL to TASM [2] and TASM to UPPAAL using Atlas Transformation Language (ATL).

The other project is concurrent code generation for synchronous language SIGNAL, a multi-clocked data-flow modeling language for safety critical systems. To fully utilize its feature on describing concurrent behavior, we developed a technique

to generate parallel code automatically from SIGNAL specifications. Based on the information obtained from clock calculus, we proposed a data structure to analyze global data-dependency relations from which concurrent tasks are extracted to help explore parallelism in the original specification. Combined with clock relations, parallel tasks are finally mapped onto the OpenMP structures [1, 3].

Challenges and future directions

This section outlines some interesting challenges in runtime and static verification that I would plan to work on in the foreseeable future to improve quality of software systems at the all stages of life cycle.

1. *Challenge 1: heterogeneous monitoring structure*

   In a cyber-physical system, monitoring requirements include discrete specifications described as DFAs or continuous specifications such as STL formula. We could even use machine-learning-based abnormal detectors as event producers to another monitor. Autonomous systems may need to merge information from multiple sources of monitors or event transducers to make decisions. At the same time, the common interface of SMEDL provides an ideal platform to coordinate heterogeneous monitors to achieve complex monitoring requirements. I would like to study theoretical and engineering challenges brought by heterogeneous monitoring structure based on real-world CPS.

2. *Challenge 2: software adaptation at multiple levels*

   Software adaptation involves both high-level repair process and low-level reflexive enforcement actions. Low-level enforcement actions depend on pre-defined actions when the property violated in single executions, while high-level repair may utilize more information such as sophisticated program profiling from multiple executions to synthesize repair solution. These two types of adaptation could have influence on each other. On one direction, execution of enforcement actions may give hints to the high-level repair. On the other direction, the modified code may require changes to the runtime monitors. I would like to study how to integrate them to achieve software adaptation with good performance.

3. *Challenge 3: overhead-aware deployment of runtime monitors*

   Detecting violation of properties require synchronous instrumentation of monitors, which may lead to large overhead on time and resource usage. SMEDL supports hybrid deployment of monitors to achieve balance between overhead and performance. However, current setting requires the user to decide the architecture manually. In [4], We have proposed a process to decide deployment of a hierarchical monitor structure by analyzing the overhead brought by asynchronous communication and monitoring logic. Based on this initial research, I would like to study a general mechanism or process to achieve optimized deployment of monitors automatically.

4. *Challenge 4: improving usability of deductive verification tools*

   Deductive verification tools for different programming languages, such as Frama-C, Verifast for C and Prusti for Rust have been used in real code base such as FreeRTOS and Firecracker for verification of core data structures. However, limitations on language support and complexity of defining property specifications make them hard to use in the normal development process. I would like to focus on improvement of usability of these tools. One particular interesting topic is to extension of Prusti to support unsafe code.

My approach to research

Conducting research activity is a challenging task. One should have solid knowledge base in general mathematics and engineering and always keep track of the state of the art in the related fields with relatively deep understanding of high-level research challenges, existing methodologies with limitations and new research possibilities. My formal schooling and research experience in multiple fields have laid me a good background to advance research in the area of formal methods.

A research activity usually starts from an interesting topic, which could be a system-level one such as development of a RV framework or a more low-level one such as improving an existing algorithm for monitoring parametric properties. At the system level, I tend to think backwardly from the high-level goal to achieve. For instance, the goal of the RV framework SMEDL is monitor large-scale component-based systems. To achieve this goal, an important research problem is how to design a formalism that can support flexible deployment and dynamically scaling of monitors along with the target system. On the other hand, a more concrete research problem is usually driven by real-world necessities. For instance, when we developed the SMEDL compiler to generate C checker code, the biggest issue is bugs that are caused by either informal semantics or discrepancy between the semantics and the implementation. Since monitor code should belong to the trusted computing base, we decided to study correct-by-construct code generation for SMEDL.

My approach to solve research problems combines deep thinking, bias for action and exchanging idea with other people. I will first try to understand the problem sufficiently by systematically collecting, reading and summarizing related literatures, performing thought experiments and getting my hands dirty if existing tools are available. Once I got

an initial idea, I will refine it by documentation, which is useful for exchanging idea with my advisors and colleagues. After idea exchanges for improvement, I will implement the method and perform experiments for evaluation and comparison. Both the method and implementation are formally documented for reference and future improvement.

I also believe that research can greatly benefit from collaboration and teamwork. Research ideas obtained from sudden inspirations often need to be further improved and validated by exchanging ideas with others. I not only enjoy working with my advisors and colleagues but also have collaborated with many researchers from MIT, IRIT, INRIA, ETH Zrich, BAE, GrammaTech and Amazon AWS. In the future, I will actively put efforts into collaborations with both academia and industry to create more research opportunities.

References

[1] Hu, K., Zhang, T., Yang, Z.: Multi-threaded code generation from signal program to openmp. Frontiers of Computer Science **7**(5), 617–626 (2013)

[2] Hu, K., Zhang, T., Yang, Z., Tsai, W.T.: Exploring aadl verification tool through model transformation. Journal of Systems Architecture **61**(3-4), 141–156 (2015)

[3] Hu, K., Zhang, T., Yang, Z., Tsai, W.T.: Simulation of real-time systems with clock calculus. Simulation Modelling Practice and Theory **51**, 69–86 (2015)

[4] Zhang, T., Eakman, G., Lee, I., Sokolsky, O.: Overhead-aware deployment of runtime monitors. In: International Conference on Runtime Verification. pp. 375–381. Springer (2019)

[5] Zhang, T., Eakman, G., Lee, I., Sokolsky, O.: Flexible monitor deployment for runtime verification of large scale software. In: International Symposium on Leveraging Applications of Formal Methods. pp. 42–50. Springer (2018)

[6] Zhang, T., Gebhard, P., Sokolsky, O.: SMEDL: Combining synchronous and asynchronous monitoring. In: International Conference on Runtime Verification. pp. 482–490. Springer (2016)

[7] Zhang, T., Wiegley, J., Giannakopoulos, T., Eakman, G., Pit-Claudel, C., Lee, I., Sokolsky, O.: Correct-by-construction implementation of runtime monitors using stepwise refinement. In: International Symposium on Dependable Software Engineering: Theories, Tools, and Applications. pp. 31–49. Springer (2018)