교육 및 연구계획서

전민석

April 3, 2024

My research aims to design domain-specific programming languages (DSLs) and program synthesis algorithms to address challenges in programming language and software engineering domains. In my DSL-based approach, I design a DSL expressive enough to describe solutions of the target problem and develop a corresponding program synthesis algorithm that effectively searches the solutions by leveraging the properties of the DSL. In programming language domain, I have developed DSLs and program synthesis algorithms for generating powerful analysis heuristics in pointer analysis, a key ingredient in compiler optimization. In software engineering, I have designed DSLs for generating effective test cases for firmware (system software) endurance tests. My research has demonstrated that DSL-based approaches are effective in addressing problems and outperform existing state-of-the-art techniques. I believe my DSL-based approach has just started, and there are many challenges and opportunities for improvement. I am planning to address these challenges and expand the applicability of the DSL-based approach to various other domains in my future work.

1 DSLs & Synthesis Algorithms for Effective Static Analysis

The goal of pointer analysis is to approximate the set of memory locations that each variable may point to during the program executions. Pointer analysis is a key ingredient in compiler optimization, and it is also widely used in many other software engineering techniques, including bug detection, security analysis, and program repair. The success of pointer analyzers heavily depends on the quality of their underlying analysis heuristics. Without high-quality analysis heuristics, the analysis becomes imprecise or expensive. Before my research, those heuristics had been manually designed by domain experts. However, manually designing analysis heuristics requires time-consuming and laborious tasks. Even worse, the manually crafted heuristics have shown suboptimal performance. To address this problem, I have designed domain-specific languages that are expressive enough to describe high-quality analysis heuristics and program synthesis algorithms that effectively search high-quality analysis heuristics in the DSLs.

***Disjunctive Model* & *Synthesis Algorithms* [6, 2, 1].** Context sensitivity is one of the most impactful factors in pointer analysis that improves the precision by distinguishing variables and objects in different calling contexts. Applying context sensitivity to all methods, however, is too expensive in practice. Selective context sensitivity aims to address this problem by selectively applying context sensitivity to methods. That is, selective context sensitivity applies context sensitivity to only a subset of methods that are likely to benefit from context sensitivity, while applying context insensitivity to the remaining methods. To effectively apply selective context sensitivity, however, qualified analysis heuristics are required to accurately determine whether each method should be analyzed with context sensitivity.

To address this problem, I designed a domain-specific language, called Disjunctive Model, that describes various selective context sensitivity heuristics. A heuristic in the DSL is a sequence of disjunctive normal form (DNF) formulas consist of user-provided features describing methods that will be analyzed context sensitively. Simultaneously, I designed program synthesis algorithms that automatically generates qualified selective context sensitivity heuristics from the DSL. The experimental results show that the

DSL-based approach automatically generates outstanding analysis heuristics that outperform the existing state-of-the-art manually crafted heuristics.

Context Tunneling [3]. In static analysis, context abstraction is essential as it is impractical to keep all the concrete contexts, and the dominant context abstraction method is k-limited context sensitivity, which keeps only the last k context elements. However, I found the last-k context abstraction has a severe limitation. This approach removes key context elements if they fall outside the last k context elements, significantly degrading the precision of the analysis.

To address this problem, I present context tunneling which enables the k-limited context sensitivity to keep the most important k-context elements instead of the last-k context elements. To apply context tunneling, however, context tunneling heuristics are required to check whether each context element is important. To this end, I adapted the Disjunctive Model and the synthesis algorithm [6] to learn high-quality context tunneling heuristics. The experimental results show that the produced context tunneling heuristics significantly improved the effectiveness of the k-limited context-sensitive analyses.

Obj2CFA [5]. In OOP (Object-Oriented Programming) program analysis, object sensitivity has been established as the dominant context flavor. The superiority of object sensitivity over other context flavors has been reinforced by extensive research. On the other hand, call-site sensitivity has been consistently dismissed because it has shown poor performance in both precision and scalability. However, I found that this superiority of object sensitivity over call-site sensitivity does not hold when context tunneling [3] is included.

In this work, I challenged the commonly accepted knowledge by developing a synthesis algorithm, called Obj2CFA, which transforms a given object sensitivity into a more precise call-site sensitivity. The synthesis algorithm takes a tunneling heuristic of an object sensitivity and produces a corresponding tunneling heuristic for call-site sensitivity. The experimental results show that the transformed call-site sensitivity is significantly more precise and scalable than the given object sensitivity.

Graphick [4]. This work addresses a key burden of the previous DSL-based approach. Disjunctive Model combines user-provided features to describe analysis heuristics; the success of Disjunctive Model heavily depends on the quality of user-provided features. However, designing high-quality features is a nontrivial and laborious task for the users. To remove this burden, I designed a framework Graphick that automatically generates high-quality features. To this end, I designed a domain-specific language, called Feature Language, that describes various features. Subsequently, I developed a synthesis algorithm that automatically generates qualified features from the DSL. The experimental results demonstrate that our approach successfully generates high-quality features, enabling the learning of effective analysis heuristics without user-provided features.

2 DSL & Synthesis Algorithms for Software Testing

In system software testing like write endurance tests, designing test cases has also been done manually by domain experts; it faces the same challenges. It requires laborious efforts, and the manually crafted test cases often yield suboptimal results. To address this problem, I also applied my DSL-based approach to automatically generate effective test cases in system software testing.

ARES [7]. Flash-based storage devices are widely used in practice, including mobile devices, automotives, and PC. However, flash-based storage devices have a finite limit in processing data write requests. Therefore, it is important for manufacturers to rigorously test and accurately provide the maximum data write capacity that their products can reliably endure. To automate the generation of effective test cases for the endurance test, I designed a domain-specific language named Abstract Relative Write Pattern, which effectively reduces the search space of test cases. Then, I developed a synthesis algorithm that automatically generates qualified test cases using the DSL. The experimental results show that the DSL-based approach successfully generates qualified test cases that outperform the existing manually designed test cases used in practice.

3 DSL & Synthesis Algorithms for Explainable Machine Learning

Our approach can also be generalized to develop inherently explainable machine learning methods.

PL4XGL. Many significant real-world problems can be modeled as graph machine learning problems, including fraud detection, program repair, and drug discovery. In such decision-critical applications, it is important to understand why the models made each prediction. However, the dominant neural-network based graph machine learning models (i.e., graph neural networks) are black-box models that do not explain why they made each prediction. Various GNN explanation techniques have been proposed, but they fall short in providing satisfactory explanations because explaining the black-box models is fundamentally challenging. To address this problem, I developed a new inherently explainable machine learning method PL4XGL. The key idea of PL4XGL is to generate inherently explainable graph machine learning models by designing a domain-specific language, called Graph Description Language, and synthesis algorithms that learn models from training graph data. The experimental results show that our DSL-based approach successfully generates inherently explainable graph machine learning models that show competitive accuracy compared to the existing dominant neural network-based models but provide significantly better explanations for the predictions. Based on this work, I will lead DSL-based explainable machine learning.

4 Future Research Plan

I believe that my DSL-based approaches are promising and have a lot of opportunities for improvement, with the potential to extend their applicability to various other domains. In my future work, I will address the existing challenges and limitations in the current methods, and I will also expand the use of the DSL-based approach to other domains.

4.1 Improving the DSL-based Approaches in Static Analysis

The current DSL-based techniques for pointer analysis have several limitations, and I plan to address these limitations.

Generating Combinations of Analysis Heuristics. Practical static analyzers use combinations of various analysis heuristics. This necessitates heuristic design processes to consider the interactions among these heuristics. However, my previous approaches lack consideration of the combinations. They have focused on generating a single analysis heuristic without considering interactions with other heuristics. Recently, I observed that blindly combining the analysis heuristics results in suboptimal performance. To address this problem, I will improve my DSL-based approach to generate combinations of analysis heuristics. To achieve this, I will identify and clarify the properties and relationships between different analysis heuristics. Then, I will design DSLs and synthesis algorithms based on these properties and relationships.

Understanding the Principles Behind the Generated Heuristics. Though effective, the analysis heuristics produced from the DSL-based approaches are currently considered as black-box heuristics that do not explain why they are effective in pointer analysis. Understanding the principles behind the learned heuristics, however, is important as it provides key insights when designing analysis heuristics. Currently, the learned analysis heuristics consist of complex DNF formulas, and this complexity poses a significant challenge in understanding the essence of the learned heuristics. To understand the principle behind the learned heuristics, I plan to design a framework that transform a given heuristic into a simplified one while maintaining its effectiveness. The simplified heuristics will help the understanding of the learned principles. Additionally, this simplification process may lead to the generalization of the learned heuristics, potentially enhancing their performance.

4.2 Generalizing the DSL-based Approaches to Other Domains

I am currently working on extending my DSL-based approach to other domains. Specifically, I am trying to apply my DSL-based approach to fault localization and explainable graph machine learning.

Generalization to Fault Localization. The goal of fault localization is to identify the locations of faults in programs that produce errors. Currently, *Spectrum-Based Fault Localization* (SBFL), which localizes buggy lines using test cases execution information, is the dominant fault localization technique because of its outstanding efficiency. SBFL, however, has a severe limitation in accuracy; it often fails to accurately identify the locations of faults. To address this problem, Idevelop a DSL-based approach to improve the accuracy of SBFL.

Our approach is based on an observation that similar faults have redundantly appeared in the same projects; faults in a previous version of a project are likely to occur again in subsequent versions. Based on this observation, I designed a domain-specific language that describes various fault patterns. Using the DSL, our synthesis algorithm learns fault patterns from the previous versions of a project. These learned fault patterns are then applied to enhance the accuracy of SBFL in the next version of the project. The preliminary results show that our project-aware fault localication technique successfully improves the accuracy of SBFL.

References

[1] Donghoon Jeon, Minseok Jeon, and Hakjoo Oh. A practical algorithm for learning disjunctive abstraction heuristics in static program analysis. *Information and Software Technology*, 135:106564, 2021.

[2] Minseok Jeon, Sehun Jeong, Sungdeok Cha, and Hakjoo Oh. A machine-learning algorithm with disjunctive model for data-driven program analysis. *ACM Trans. Program. Lang. Syst.*, 41(2):13:1–13:41, June 2019.

[3] Minseok Jeon, Sehun Jeong, and Hakjoo Oh. Precise and scalable points-to analysis via data-driven context tunneling. *Proc. ACM Program. Lang.*, 2(OOPSLA):140:1–140:29, October 2018.

[4] Minseok Jeon, Myungho Lee, and Hakjoo Oh. Learning graph-based heuristics for pointer analysis without handcrafting application-specific features. *Proc. ACM Program. Lang.*, 4(OOPSLA), November 2020.

[5] Minseok Jeon and Hakjoo Oh. Return of cfa: Call-site sensitivity can be superior to object sensitivity even for object-oriented programs. *Proc. ACM Program. Lang.*, 6(POPL), jan 2022.

[6] Sehun Jeong, Minseok Jeon, Sungdeok Cha, and Hakjoo Oh. Data-driven context-sensitivity for points-to analysis. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA), 2017.

[7] Jinkook Kim, Minseok Jeon, Sejeong Jang, and Hakjoo Oh. Automating endurance test for flash-based storage devices in samsung electronics. In *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*, 2023.