Research Statement

Minseok Jeon (전민석)

May 5, 2024

To effectively address a problem, a domain-specific language designed for that problem is essential. Using a suitable language allows us to grasp the essence of the problem and find the appropriate solution. This principle is also applicable in computer science. Each problem in computer science has its own characteristics, and a domain-specific language tailored to these characteristics is necessary to solve the problem effectively.

My research aims to design domain-specific programming languages (DSLs) and program synthesis algorithms to address challenges in various computer science domains, including programming languages, software engineering, and machine learning. Given a problem, I design a DSL that can describe the solution to the target problem. The domain-specific language is carefully designed to capture the essence of the problem. Then, I develop a synthesis algorithm that automatically and effectively searches for solutions within the DSL. The work has been published in top-tier conferences (PLDI 2024, ICST 2023, POPL 2022, OOPSLA 2020, OOPSLA 2018, OOPSLA 2017). In the programming language domain, for example, 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. In machine learning, I used my DSL-based approach to develop an inherently explainable graph machine learning method. I have demonstrated that the DSL-based approach produces outstanding results in various domains. Currently, I am applying the DSL-based approach to fault localization or trying and working on improving the designed DSLs and synthesis algorithms.

As a long-term goal, I will develop a general framework that can automatically generate DSLs for any given problem. Currently, I design DSLs manually for each problem, which is a very challenging and time-consuming task. This is primarily because we do not know the solutions to the problem when we begin designing a DSL. Currently, I repeatedly guess and refine a DSL, then check its effectiveness, which is time-consuming. As a long-term goal, I aim to develop a framework that automates this language designing process. Based on this framework, I will establish the DSL-based approach as a dominant method for addressing most problems in computer science.

1 DSL & Synthesis for Explainable Graph Machine Learning

In my recent work, I applied my DSL-based approach to develop an inherently explainable graph machine learning method, which will be published at PLDI 2024. Currently, I am collaborating with students to further extend this research. Moving forward, I will lead the DSL-based explainable machine learning research, building upon the strong foundation laid by this work.

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 crucial to understand the reason why the models made each prediction. However, the prevalent graph neural networks (GNNs) are typically black-box models that lack transparency in their decision-making processes. Various GNN explanation techniques have been proposed, but they fall short in providing satisfactory explanations, due to the inherent challenges of interpreting black-box models. To address this problem, I developed an 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. Our experimental results demonstrate that this DSL-based approach not only competes in accuracy with existing neural network-based models but also provides significantly clearer explanations for the predictions. Based on this work, I will lead DSL-based explainable machine learning. For example, I am currently working on extending the DSL along with the synthesis algorithm for improving the accuracy of PL4XGL, and trying to use the DSL-based approach to improve the existing neural network-based models.

2 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 crucial component of compiler optimization and is widely used in various software engineering techniques, including bug detection, security analysis, and program repair The effectiveness of pointer analyzers heavily depends on the quality of their underlying analysis heuristics. Without high-quality heuristics, the analysis becomes either imprecise or costly. Before my research, these heuristics had been manually designed by domain experts, a process that is both time-consuming and labor-intensive. 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, significantly enhancing precision by distinguishing between variables and objects in different calling contexts. However, applying context sensitivity to all methods is impractically expensive. Selective context sensitivity addresses this challenge by strategically applying context sensitivity only to methods that are likely to benefit from it, while other methods are analyzed without context sensitivity. 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, which can describe various selective context sensitivity heuristics. A heuristic in this 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 generate effective selective context sensitivity heuristics from the DSL. Experimental results demonstrate that the DSL-based approach automatically generates effective analysis heuristics that outperform the existing state-of-the-art manually crafted heuristics.

Context Tunneling [3]. In static analysis, context abstraction is crucial 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 have identified a significant limitation with this last-k context abstraction method. This approach removes key context elements if they fall outside the last k context elements, significantly degrading the precision of the analysis. This exclusion substantially degrades the precision of the analysis, as important contextual information is lost.

To address the limitations of the conventional k-limited context sensitivity, I present 'context tunneling,' a method that allows for retaining the most important k context elements, rather than merely the last k context elements. To apply context tunneling, however, context tunneling heuristics, checking whether each context element is important, are required. To this end, I adapted the Disjunctive Model and the synthesis algorithm [6] to learn high-quality context tunneling heuristics. The experimental results demonstrate that the heuristics generated by this approach significantly enhance the effectiveness of 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. Conversely, call-site sensitivity has been consistently dismissed because it has shown poor performance in both precision and scalability. However, this research challenge this status quo. I discovered that the perceived superiority of object sensitivity over call-site sensitivity does not hold when context tunneling [3] is applied.

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 generates a corresponding tunneling heuristic for call-site sensitivity. The experimental results demonstrate 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 developed a framework Graphick that automatically generates high-quality features. 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.

3 DSL & Synthesis Algorithms for Software Testing

In system software testing, such as write endurance tests, the design of test cases has been handled manually by domain experts. 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. This method not only reduces the effort required but also improves the quality of the test outcomes.

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.

4 Future Research Plan

I am confident that my DSL-based approach is promising and offers numerous opportunities for further refinement and broadening of their applicability across various domains. In the short term, I am committed to addressing the existing challenges and limitations within our current methods. Concurrently, I plan to expand the use of the DSL-based approach to additional domains. As a long-term goal, I will develop a general framework that can automatically generate DSLs for any given problem. This framework would revolutionize the way we approach and solve complex problems in computer science and beyond.

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 Improving the DSL-based Approaches in Explainable Graph Machine Learning

Currently, I am working on improving the accuracy of PL4XGL and combining it with existing neural network-based approaches.

Improving the Accuracy of* PL4XGL*. The current version of PL4XGL has shown suboptimal accuracy on certain datasets. This problem comes from the current DSL's limitations in describing some key properties in the datasets. To address this problem, I am trying to extend the expressiveness of the DSL to describe these key properties with a student, and I recently observed a promising results that significantly improve the accuracy of PL4XGL.

Combining DSL-based Approach with Nueural Network-based Methods. Another interesting future work is to combine our DSL-based approach with existing neural network-based methods. For instance, I am currently using PL4XGL to generate high-quality features, which are then incorporated into established neural network-based graph machine learning models. Preliminary results are promising; the features produced by PL4XGL have significantly enhanced the accuracy of these models. This hybrid approach not only leverages the strengths of both methodologies but also opens up new possibilities for improving model performance and interpretability in complex machine learning tasks

4.3 Generalizing the DSL-based Approaches to Other Domains

I am currently working on extending my DSL-based approach to other domains. Specifically, I am exploring its potential in fault localization.

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 uses test case execution information to localize buggy lines, is the prevalent technique due to its efficiency. However, SBFL suffers from significant accuracy limitations, often failing to precisely identify fault locations. To address this problem, I am working on developing a DSL-based approach to improve the accuracy of SBFL with a student.

Our technique is based on an observation that similar faults frequently recur within the same projects; faults identified in previous versions of a project are likely to appear again in subsequent versions. Based on this observation, I designed a domain-specific language that describes various fault patterns. Using this DSL, our synthesis algorithm extracts fault patterns from earlier project versions. These patterns are then utilized to improve the accuracy of SBFL for localizing faults in the current version of the project. Preliminary results show that our project-aware fault localization technique significantly enhances the accuracy of SBFL.

4.4 Long-term Goal: Automatic DSL Generation

My long-term goal is to develop a general framework capable of automatically generating domain-specific languages (DSLs) for any given problem. A key challenge in previous research was the manual design of DSLs, which is both time-consuming and prone to errors. An insufficiently expressive DSL may fail to encapsulate solutions effectively, while an overly expressive DSL can complicate the synthesis of solutions. Previously, I engaged in the manual design of DSLs through trial and error. However, my long-term goal is to develop a framework that automates the design of DSLs, thereby overcoming these challenges. By developing a method that can automatically synthesize DSLs tailored to given problems, we can provide effective solutions across various domains in computer science.

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.