Machine Learning-Guided Adaptive Program Analysis

Hakjoo Oh
Korea University
hakjoo_oh@korea.ac.kr

Kihong Heo
Seoul National University
khheo@ropas.snu.ac.kr

Hongseok Yang
University of Oxford
hongseok.yang@cs.ox.ac.uk

Kwangkeun Yi
Seoul National University
kwang@ropas.snu.ac.kr

1. Overview

In this talk, we present our experiences to use machine learning techniques for automatically adapting a program analysis to a given verification task. Building a cost-effective static analyzer for real-world programs is regarded an art, mainly because of the difficulty in balancing the cost and the precision of an analyzer. An ideal analyzer should be able to adapt to a given analysis task automatically, and avoid using techniques that unnecessarily improve precision and increase analysis cost. However, designing a good adaptation strategy is highly nontrivial, and it requires a large amount of engineering efforts. We aim to automate this procedure by learning a good strategy from an existing codebase.

This talk is based on our recent work [1, 4] and an ongoing project, which are summarized in Sections 2, 3, and 4, respectively.

2. Learning via Bayesian Optimization

In [4], we present an approach for building an adaptive static analyzer, which can learn its adaptation strategy from an existing codebase via Bayesian optimization. In our approach, the analyzer includes a parameterized strategy that decides, for each part of a given program, whether to apply a precision-improving technique to that part or not. This strategy is defined in terms of a function that scores parts of a program. The strategy evaluates parts of a given program using this function, chooses the top k parts for some fixed k, and applies the precision-improving technique to these parts only. The parameter of the strategy controls this entire selection process by being a central component of the scoring function.

The success of such an analyzer depends on finding a good parameter for its adaptation strategy. As typical in other machine learning techniques, this learning part is formulated as an optimization problem: find a parameter that maximizes the number of queries in the codebase which are proved by the analyzer. This is a challenging optimization problem because evaluating its objective function involves running the analyzer over several programs and so it is expensive. We present an (approximate) solver for the problem that uses the powerful Bayesian optimization technique and avoids expensive calls to the program analyzer as much as possible.

Using our approach, we developed partially flow-sensitive and context-sensitive variants of Sparrow [3], a realistic C program analyzer. The experimental results confirm that using an efficient optimization solver such as ours based on Bayesian optimization is crucial for learning a good parameter from an existing codebase; a naive approach for learning simply does not scale. When our partially flow- and context-sensitive analyzer was run with a learnt parameter, it answered the 75% of the buffer-overrun queries that require flow- or context-sensitivity, while increasing the analysis cost only by 3.3x of the flow- and context-insensitive analysis, rather than 40x or more of the fully sensitive version.

3. Learning from Automatically Generated Labeled Data

In [1], we propose a new method for automatically learning a variable-clustering strategy for the Octagon analysis from a given codebase. Relational program analyses track sophisticated relationships among program variables and enable the automatic verification of complex properties of programs. However, the computational costs of various operations of these analyses are high so that vanilla implementations of the analyses do not scale even to moderate-sized programs. One of the most popular optimizations used by practical relational program analyses is variable clustering. Given a program, an analyzer with this optimization forms multiple relatively-small subsets of variables, called variable clusters or clusters. Then, it limits the tracked information to the relationships among variables within each cluster, not across those clusters, increasing the analysis scalability substantially. However, the effectiveness of this technique comes only when a good strategy for clustering is used. Our goal is to automatically learn such a strategy from an existing codebase.

The most important aspect of our learning method is the automatic provision of labeled data. Although the method is essentially an instance of supervised learning, it does not require the common unpleasant involvement of humans in supervised learning, namely, labeling. Our method takes a codebase consisting of typical programs of small-to-medium size, and automatically generates labels for pairs of variables in those programs by using the impact pre-analysis from our previous work [2], which estimates the impact of tracking relationships among variables by Octagon on proving queries in given programs. Because this learning occurs offline, we can bear the cost of the pre-analysis, which is still significantly lower than the cost of the full Octagon analysis. Once labeled data are generated, our method runs an off-the-shelf classification algorithm, such as decision-tree inference, for inferring a classifier for those labeled data. Conceptually, the inferred classifier is a further approximation of the pre-analysis, which gets found automatically from a given codebase.

The experimental results show that our method results in the learning of a cost-effective variable-clustering strategy. We implemented our learning method on top of Sparrow [3] and tested against open source benchmarks. In the experiments, our analysis with the learned variable-clustering strategy scales up to 100KLOC within the two times of the analysis cost of the Interval analysis. This corresponds to the 33x speed-up of the selective relational analysis based on the impact pre-analysis [2] (which was already significantly faster than the original Octagon analysis). The price of speed-up was mere 2% increase of false alarms.

4. Learning with Automatically Generated Features

In the last part of this talk, we introduce our on-going work to automatically generate features used in machine learning models. Although we succeeded to develop machine learning algorithms for adapting program analyses [1, 4], it has an important limitation; the success of the learning approaches crucially depends on the features used, but those features must be manually crafted by human experts. This feature construction process is not only nontrivial and tedious but also hard to reuse and transfer from one analysis to another, preventing the learning approaches from being widely adopted. Our aim is to automate this feature construction process. We will describe the key idea of our approach and present the preliminary experimental results.

References

[1] Kihong Heo, Hakjoo Oh, and Hongseok Yang. Learning a variable-clustering strategy for octagon from labeled data generated by a static analysis. In *SAS*, 2016.

[2] Hakjoo Oh, , Wonchan Lee, Kihong Heo, Hongseok Yang, and Kwangkeun Yi. Selective context-sensitivity guided by impact pre-analysis. In *PLDI*, 2014.

[3] Hakjoo Oh, Kihong Heo, Wonchan Lee, Woosuk Lee, and Kwangkeun Yi. Sparrow. `http://ropas.snu.ac.kr/sparrow`.

[4] Hakjoo Oh, Hongseok Yang, and Kwangkeun Yi. Learning a strategy for adapting a program analysis via Bayesian optimisation. In *OOPSLA*, 2015.