Static loop summarization and its application

Chengpeng Wang

Abstract

Loops are commonly-used control structures in real-world programs and make it convenient for programmers to implement complex functionalities of program. However, the incorrect manipulation inside loops are the cause of many critical vulnerabilities, such as buffer overflow and string fault, and redundant instructions and inefficient scheduling inside the loop can slow down the execution.

Static program analysis, one of classical program analysis techniques, takes source files as inputs and analyzes the behaviors of the program without actual execution. It has been proven useful for bug detection, program verification, and program performance optimization. Despite the effectiveness of static analysis, the loops are still regarded as its Achilles heel. Due to the non-deterministic number of iterations, it is impossible to simulate the execution to obtain all the possible loop states. In order to get the partial or whole effects of the loops, static loop summarization has been proposed to generate loop summaries summarizing relationship of input and output represented by a set of logical constraints, which helps in the understanding of the behaviour of loops.

There have been a large body of literature on loop summarization, most of which focus on numeric loops and single loops. Besides, the string loop and multi-path loops are also the major concerns of researchers. Nevertheless, the previous works have been highly restricted to the form of the loops and can not be applicable to more complex ones, including nested loops and data structure traversal.

In this review, we discuss typical works on static loop summarization and their applications in test case generation, program performance analysis, and loop termination analysis. With the aim of practical loop summarization, our future directions and several concrete problems are discussed.

1 Introduction

Loops are commonly used control structures in real-world programs. Although the flexible usages of loop conditions and nested loops make it convenient for programmers to implement complex program functions, many critical vulnerabilities, such as

buffer overflow and string fault, are caused by the incorrect manipulation in the loops. Meanwhile, redundant instructions and inefficient scheduling inside the loop create the demands of performance analysis and program optimization.

In order to analyze the program with loops, several program analysis techniques, including static program analysis, model checking, and dynamic program analysis to obtain possible states of loops. Static program analysis, which takes source files as inputs aims to over-approximate possible program states without actual execution, is widely used for bug detection, program verification, code refactoring, and performance optimization. Theoretically, compared with dynamic analysis, the results generated by over-approximation can cover all the possible states theoretically, which assures the soundness of the analysis. However, loops are still regarded as Achilles' heel of static analysis. Due to the non-deterministic number of iterations, it is impossible to simulate the execution to obtain all the possible loop states. In order to get the partial or whole effects of the loops, static approaches have been proposed for loop analysis, namely loop unwinding, loop invariant generation and loop summarization.

Loop summarization provides a more accurate and adequate comprehension of loops. It summarizes the relationship between input and output of a loop by a set of constraints [10]. In this review, we will discuss static loop summarization. It provides a more accurate and adequate comprehension of loops and has a wide application in test case generation, program performance analysis, and loop termination analysis.

The contributions of this review include: (1) classify the loops based on the loop structure and data structure; (2) summarize the approaches to summarizing different types of loops and analyze their advantages and disadvantages; (3) propose the possible directions for loop summarization in the future.

The literature review is organized as follows. In section 2, we will give a detailed classification of the loops in real-world programs. In section 3, recent works on static loop summarization will be discussed, which are aimed at different types of loops, including string loops, numeric loops, and multi-path loops. The comparison and the limitation will be given by case studies in section 4. Section 5 will give several examples of the applications based on static loop summarization. The conclusion and the future direction will be discussed in section 6.

2 Loop Classification

To summarize a loop, it is important to classify the loops in the real-world programs and deal with each type specifically. Here we discuss four criterion of loop classification, including the types of data in the loop, linearity of arithmetic operations, the form of branch conditions, and the number of loop paths.

Numeric loops, especially integer loops, are the objectives of most of the static loop analysis [5, 8]. According to the value change of the integer variables in loop conditions, numeric loops can be classified in a more fine granularity. If the value

change of the branch condition variable is constant, the condition is an inductive variable condition. Based on whether the loops contain the non-inductive variable conditions, the loops can be divided into two categories, briefly denoted by IV loops and NIV loops [10]. Compared with IV loops, it is far more difficult to get a precise summaries of the values in NIV loops due to the nondeterministic value change. Another criteria of numeric loops is based on the type of arithmetic transformation in the loops [5]. More specifically, the loops which consist of a sequence of affine operations are called linear loops and the value of the variables in each iteration can be represented precisely in a closed-form. Non-linear loops, in which the semantic of one iteration cannot be represented by a linear transformer, are more complex than linear loops, because the precise values in each iteration might not be easily summarized. Several attempts have been made to summarize the loops with linear operations and inductive variable conditions in the past two decades. The summaries generated for the two types of loops can be represented by a matrix and a group of constraint respectively.

Apart from numeric loops, the loop manipulating particular types of data structures is also frequently used in the real-world programs. For example, the traversals of the array, string, and containers in the C++ STL library are common instances of the loops [6]. Different from the loops manipulating arrays and the strings with a fixed length, the sizes of dynamic data structures, including containers, are not deterministic, and these data structures are accessed and altered in a more sophisticated manner via library functions and APIs. The dynamic memory allocation and the sophisticated semantics in the loops pose a great challenge to the summarizing of these loops.

Moreover, path interleaving in the multi-path loops is another source of the complexity that needs to be considered. In Xie's work [10], two critical pieces of information to identify path interleaving patterns are pointed out, namely the number of iterations and the execution order of each path. Based on these two kinds of information, the patterns of path interleaving are divided into three types, namely sequential, periodic and irregular. The first two of them are regular patterns and the paths can be abstracted precisely in the theory of linear-integer arithmetic.

In [2], a static analysis framework CLAPP is proposed to perform a large-scale empirical study on why and how Android applications take advantages of loops. The actions triggered by APIs upon containers is considered in this work . In [10], the authors conduct a comprehensive investigation of real-world loops and study the distributions of loop classification based on the criterion of path interleaving pattern and the form of branch conditions. It is depicted that there are a considerable amount of loops with inductive conditions and regular patterns of path interleaving, which accounts for 33.87% of those in the real-world programs. The behavior of these loops is predictable, and they are the main objects of current loop summarizations. NIV loops, many of which manipulates complex data structures, take up 66.01% of loops in real-world programs and are still a pain point of loop summarization.

```
1    while (i < n) {
2      sum = sum + i;
3      i++;
4    }
```

Figure 1: Loop for summation

3 Approach

The loop summarization aims to model the effect of the loop in the execution by the constraints with respect to the input and output values of the loop. For example, in Figure 1, the summary of the loop in Figure 1 is formalized by the constraints shown in 1, which provide the relationship of the initial and updated values of the variables.

$$\begin{aligned} i' &= n \\ n' &= n \\ sum' &= sum + (i + i' - 1) * (i' - i)/2 \end{aligned} \tag{1}$$

In the above example, the constraints are in the theory of linear-integer arithmetic. For the loop manipulating data structures and multi-path loops, the form of summaries are more complex. On the one hand, arithmetic theory might be expressive enough to support the modeling of the semantics by the constraints. On the other hand, a large amount of disjunctives can be brought out by the multi-paths if summarizing the loops based on path summaries. In the following three subsections, we will review the loop summarization of numeric loops, string loops and multi-path loops with the example of the loop and the constraints generated in the summarization.

3.1 Numeric Loop Summarization

In recent works, numeric loops are modeled by symbolic matrices with several specific assumptions, such as the assumption that arithmetic operations are the linear combinations of the variables in the loop. In the work [3], the symbolic matrix representing the semantics of the loop is decomposed into Jordan normal form, which can be utilized to the computation acceleration of the closed forms of the variables, which is meaningful to refactor the loops to decrease the time complexity. However, the effectiveness of the approach depends on the assumption that the symbolic matrix can model the effect of the loop precisely, i.e., there are no non-linear operations and branch conditions.

Compared to the approach in [3], the summary technique in [5] provides a more general and fundamental perspective of loop summarization. Firstly, additional numeric quantities, such as the number of memory accesses, are taken into consideration, which makes the summary contain richer information. Secondly, it is proven that the logic for expressing the summaries is decidable, which yields decision procedures for

```
1  #define MAXLINE 50
2  void request(char *input) {
3    char fbuf[MAXLINE + 1];
4    int i = 0, fb = 0, f = 0;
5    while(input[i] != '\0') {
6      if (input[i] != '\n' && f <= MAXLINE)
7        fb++;
8      f++;
9      i++;
10   }
11   fbuf[fb] = '\0';
12 }
```

Figure 2: An example of string loop [11]

verifying the safety and termination of a class of numerical loop over rational numbers. It is shown that the procedure for computing for a class of numeric loops can be used to over-approximate the behavior of arbitrary numerical programs.

Numeric abstract domains, including polyhedral domain and octagon domain, are utilized for over-approximate the possible values of numeric variables in order to obtain the numeric invariants at the entry and exit of the loops, which reflect the relationship of the numeric variables. Although it is natural to handle linear arithmetic branch conditions, the invariants cannot capture the correlation between the input and output of the loop.

3.2 String Loop Summarization

String loop is another common type of loops. In each iteration, the characters in the string can be accessed or overwritten. The length of the string can be changed owing to certain write operations, which change the position of the first ending character. Figure 2 shows a simple example of string loop. The string is accessed by the index i and there is no element changed in the loop. A buffer overflow bug can occurs provided with certain input string if the value of *fb* at the exit of the loop exceeds *MAXLINE*.

One of the challenges in string loop summarization is that the loop conditions are mostly dependent on the content of the string, which demands on more sophisticated theories to support the modeling the semantics by constraints. With the benefit of the development of string solver techniques [4], the solving of string constraints can be supported, which sheds light on the loop summarization of the string loop. In S-Looper [11], the summary of a string loop is summarized by a group of string constraints. Different from most linear numeric loop summarization, S-Looper aims to fit in a more general class of loops, which contains multiple paths. Inspired by the path abstraction technique in [7], S-Looper focuses on the loops which contain string comparisons and access operations, and generates string constraints related to the path

```
1  while (x < n) {
2    if (z > x) x++;
3    else z++;
4  }
```

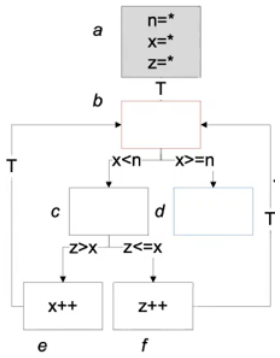

Figure 3: An example of multi-path loop [10]

counters and the content of the substrings. The constraints can be further utilized in program verification and symbolic execution.

For a more general string loop, complex branch conditions make the pattern of path interleaving irregular. Moreover, the change of string content in the iterations can also increase the difficulty of solving the path conditions. Branch conditions and the change of string content in the loop degrade the effectiveness of path counter based summarization techniques.

3.3 Multi-path Loop Summarization

Path interleaving of multi-path loops is another root of difficulty for loop summarization. Figure 3 shows an example of multi-path loop. The loop contains three paths, namely $\pi_1 : b \to c \to b$, $\pi_2 : b \to e \to f \to b$ and $\pi_3 : b \to d$, and the possible path traces include $\pi_1 \to \pi_3$ and $\pi_1 \to \pi_2 \to \cdots \pi_1 \to \pi_3$. In the actual execution, the length of the trace sequences can be infinite, and interleaves in a chaotic manner. This poses a great challenge to establishing a precise and sound abstraction of loop paths.

To the best of our knowledge, Proteus [10] is the first work to consider path interleaving in multi-path loop summarization. It aims to summarize the integer loops with multi-paths. Based on the observation that the loop summary is the disjunctive of the trace summaries, Proteus attempts to summarize all the feasible traces by combining of the summaries of its paths one by one. In order to collect all the feasible traces, Path Dependency Automaton(PDA) is proposed in which the states represent the loop paths and the transitions represent the feasible path interleaving, where the constraints are labeled to indicate the condition of the interleaving. The summaries of the traces

in some certain regular patterns can be obtained based on the transition conditions of PDA, and the loop is summarized by the disjunctive of the trace summaries.

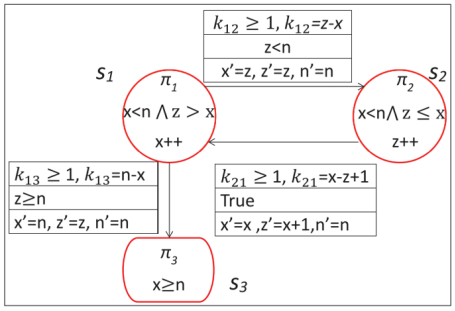

Figure 4: Path Dependency Automaton [10]

Although the evaluation part shows that the loop summaries can be effectively utilized in the loop bound analysis and symbolic execution-based test case generation, there still has several limitations in the cases of complex loop structure or the loops manipulating data structures and suffers the explosion of the disjunctives when the number of the branches is large.

4 Comparison

It is commonly known that a general approach to summarizing of a general loop in any form does not exist. In order to perform precise static loop summarization at best effort, the approaches need to be combined on demands.

According to the forms of the loop, the loop summaries obtained can be divided into two categories. The first category of loop summaries include of string loop summaries in [11] and mltiulti-path loops in [10]. They are a group of the constraints in string theory and linear-integer arithmetic theory, which can not computes the value of loop variables explicitly. The second category is the closed-form summaries of some certain classes of numeric loops, based on which the values in each iteration can be calculated directly. The closed forms of the loops can be regarded as the equivalent or approximated semantics of the loop.

In the following part, we give a brief qualitative comparison of these approaches in terms of precision, efficiency, and capability.

Precision Closed-form summaries can reflect the internal program states in each iteration of the loops while constraint-form summaries cannot. The summaries of numeric loops are more precise than the summaries of string loops and multi-path loops.

Efficiency Constraint solvers are in high demand when integrating constraint-form summaries, and the invocation of solvers can be time-consuming in the case of large-

scale programs. In contrast to constraint-form summaries, it is straightforward to compute the closed-form summaries based on the matrix multiplication algorithm.

Capability String loop summarization technique in [11] can summarize the single loops without path interleaving precisely, and the effectiveness is degraded in the presence of complex path interleaving. By contrast, multi-path loop summarization takes path interleaving into account, and its objectives cover a wider range of loops.

5 Application

5.1 Termination Analysis and Loop Bound Analysis

Loop bound analysis aims to estimate the bound of the loop. The constraint-form loop summaries in [10, 11] are natural to be applied to loop bound analysis by adding up the path counters. The estimated bound can strengthen the ability of program verification technique, such as bounded model checking, to handle the programs in the presence of loops.

Slightly different from loop bound analysis, loop termination analysis is to determine whether the loop can terminate after the finite number of executions [2, 9] It is an important problem to assure system security, and performance problems and denial-of-service attacks can be triggered by non-termination bugs. In [9], Loopster takes advantage of a divide-and-conquer approach to determine the termination of multi-path loops, and the termination of the target loop can be approximated by the termination of each path. Other techniques of loop termination analysis, including ranking functions, are discussed in the part of related work in [9] and compared with loop summarization based approaches.

5.2 Performance analysis

Performance analysis and loop optimization: The performance problems caused by redundant or inefficient computation in loop iterations can be detected based on the loop summarization [1]. In addition, composable and sound transformations can improve the performance of the execution without changing the function of the loops [8]. The redundant traversal of collections is explored in [6], and asymptotic performance bugs can be detected based on the write and traversal footprint. In this work, the footprint is the profile of the loop, which can be regarded as another form of loop summary.

5.3 Test case generation

Current bug detection techniques suffer the problem of path explosion when analyzing the programs with loops. Fortunately, loop summaries abstract the semantics of the loops and approximate the loop states, thus help other program analysis techniques to comprehend the loops better.

Symbolic execution unrolls the loop with a fixed number of iterations, and the whole program state space might not be explored thoroughly due to insufficient number of unrolling. In the work [10], Proteus is integrated into symbolic execution engines, and the loops are regarded as black boxes, of which the semantics are encoded by the loop summaries. This improvement can accelerate symbolic execution engines in the application of test case generation.

6 Conclusion and Future Work

This review has enumerated the recent works on the summarization of different type of loops. According to our survey, it is found that the loop summaries including closed forms and logical constraints provide useful information of loops and have a wide application in loop termination analysis, test case generation and loop performance analysis. Path abstraction based on path counting provides the possibility to summarize the loops with inductive branch conditions, and path dependency provides the insight to extend the path counter based approaches to summarization of general multi-path loops. For numeric loops, the symbolic matrices abstract the linear arithmetic operations effectively and benefit the loop acceleration. There are several empirical studies on the loops in the real-world programs, which discover interesting phenomena about the structure of loops.

However, current loop summarization approaches still face great challenges. **The first challenge is that current approaches can not handle complex data structure in the loop.** For example, the traversal of a linked list or a collection is often involved with the data stored in the node or element, and path interleaving pattern might be more complex. More expressive and specific models are demanded to encode the layout of data structure and operations on it, including manipulation via pointer values, and other APIs.

The second one is complex loop structures, including nested loops and multi-path loops, can not be dealt with by current approaches effectively. The complexity of path interleaving is the cause of difficulty of encoding the effect of statement by constraints.

To achieve a practical loop summarization, there is still a long way to go, but the path is looking considerably brighter with the development of static analysis.

Acknowledgement

The examples and figures are extracted from the original papers.

References

[1] Bihuan Chen, Yang Liu, and Wei Le. Generating performance distributions via probabilistic symbolic execution. In *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*, pages 49–60, 2016.

[2] Yanick Fratantonio, Aravind Machiry, Antonio Bianchi, Christopher Kruegel, and Giovanni Vigna. Clapp: characterizing loops in android applications. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 687–697, 2015.

[3] Bertrand Jeannet, Peter Schrammel, and Sriram Sankaranarayanan. Abstract acceleration of general linear loops. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 529–540. ACM, 2014.

[4] Adam Kiezun, Vijay Ganesh, Shay Artzi, Philip J. Guo, Pieter Hooimeijer, and Michael D. Ernst. HAMPI: A solver for word equations over strings, regular expressions, and context-free grammars. *ACM Trans. Softw. Eng. Methodol.*, 21(4):25:1–25:28, 2012.

[5] Zachary Kincaid, Jason Breck, John Cyphert, and Thomas W. Reps. Closed forms for numerical loops. *PACMPL*, 3(POPL):55:1–55:29, 2019.

[6] Oswaldo Olivo, Isil Dillig, and Calvin Lin. Static detection of asymptotic performance bugs in collection traversals. In David Grove and Steve Blackburn, editors, *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*, pages 369–378. ACM, 2015.

[7] Jan Strejcek and Marek Trtík. Abstracting path conditions. In Mats Per Erik Heimdahl and Zhendong Su, editors, *International Symposium on Software Testing and Analysis, ISSTA 2012, Minneapolis, MN, USA, July 15-20, 2012*, pages 155–165. ACM, 2012.

[8] Kirshanthan Sundararajah and Milind Kulkarni. Composable, sound transformations of nested recursion and loops. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019*, pages 902–917, 2019.

[9] Xiaofei Xie, Bihuan Chen, Liang Zou, Shang-Wei Lin, Yang Liu, and Xiaohong Li. Loopster: static loop termination analysis. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, pages 84–94, 2017.

[10] Xiaofei Xie, Bihuan Chen, Liang Zou, Yang Liu, Wei Le, and Xiaohong Li. Automatic loop summarization via path dependency analysis. *IEEE Trans. Software Eng.*, 45(6):537–557, 2019.

[11] Xiaofei Xie, Yang Liu, Wei Le, Xiaohong Li, and Hongxu Chen. S-looper: automatic summarization for multipath string loops. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015, Baltimore, MD, USA, July 12-17, 2015*, pages 188–198, 2015.