How Accurate Is Dynamic Program Slicing? An Empirical Approach to Compute Accuracy Bounds

Siyuan Jiang, Raul Santelices, Haipeng Cai
University of Notre Dame, U.S.A.
E-mail: {sjiang1|rsanteli|hcai}@nd.edu

Mark Grechanik
University of Illinois at Chicago, U.S.A.
E-mail: drmark@uic.edu

***Abstract*—Dynamic program slicing attempts to find runtime dependencies among statements to support security, reliability, and quality tasks such as information-flow analysis, testing, and debugging. However, it is not known *how accurately* dynamic slices identify statements that *really affect* each other. We propose a new approach to estimate the accuracy of dynamic slices. We use this approach to obtain bounds on the accuracy of multiple dynamic slices in Java software. Early results suggest that dynamic slices suffer from some imprecision and, more critically, can have a low recall whose *upper bound* we estimate to be 60% on average.**

I. Introduction

Program dependencies describe how statements affect other statements in programs [1]. Obtaining accurate dependencies is crucial for many tasks in security, reliability, and quality that identify information flows and help developers understand, test, debug, and evolve software. Program slicing [2], in its forward version, is normally used to find all code statements that depend directly or transitively on another statement. Dynamic slicing [3], in particular, is used to find such code for concrete executions to better describe typical program behaviors.

Despite the importance of slicing, however, no work in the literature has assessed how accurately slices reflect *real* dependencies in code. *Semantic* (real) dependencies were defined more than 20 years ago [1] but no method exists to bridge the gap between *computing* such dependencies (an undecidable problem) and their approximations found by slicing. In consequence, we do not know how accurate, in precision and recall, those approximations are. Yet, *the accuracy of program slices affects every technique that relies on them*.

This lack of information about the accuracy in practice of dynamic slices for modeling real dependencies in software creates uncertainty about the effectiveness of existing applications based on such slices. This uncertainty also hinders our ability to rely on dynamic slicing for new applications. Moreover, without understanding all causes of inaccuracies in slicing, further improvements to slicing are hard to achieve.

Various algorithmic improvements have been proposed and empirically shown to reduce somewhat the size of slices and thus, implicitly, increase their *precision*. However, their actual precision remains unknown. But more critically, for dynamic slices, the potential problem of *recall* has yet to be addressed. For many parts of the code, the runtime behavior of a statement s can decide that such code will *not* execute even though by definition [1] this code *is* dynamically dependent on s.

To address this problem, we propose a new approach for estimating the accuracy of dynamic forward slices. Computing semantic dependencies is undecidable, so instead we derive formulas for lower and upper bounds of the precision and recall of dynamic slices based on the relationships among static slices, dynamic slices, and the subset of semantic dependencies found by our sensitivity-analysis tool SENSA [4]. Our results on 50 dynamic slices indicate that *precision* can be at least 78%—relatively high but not perfect. The average *recall*, however, ranges between 25%–60%, which is inadequate for users and tasks that need to find all or most runtime dependencies.

II. Dependencies, Slicing, and SENSA

Dependence between two statements is a relationship such that the behavior of one statement is subject to the behavior of the other. *Data* and *control* dependencies [2] are commonly used to approximate real, *semantic* dependencies [1] which are not computable. If a statement reads from a variable the value that another statement writes to it, the first statement is *data dependent* on the second one. If a statement's execution depends directly on a branching decision at another statement, the first statement is *control dependent* on the second one. A statement s_1 is *semantically dependent* on statement s_2 if, for some program state at s_2, a change can be made to s_2 that affects the values computed at s_1 or the occurrences of s_1.

Program slicing [2] is an analysis technique that extracts the parts of a program related by data and control dependencies to a statement called the "slicing criterion". *Forward dynamic slicing* [3], in particular, finds all statements in the forward transitive closure of the *runtime* data and control dependencies on the slicing criterion for a specific set of executions.

We measure accuracy using standard definitions. The *precision* of a dynamic slice is the fraction of all statements in the slice that are semantically dependent at runtime on the slicing criterion. The *recall* is the fraction of all runtime semantically-dependent statements included in the dynamic slice.

Because the data and control dependencies used to compute slices are approximations of semantic dependencies, slicing can be imprecise. Moreover, dynamic slicing can have imperfect recall not only because of dependencies that only occur in other executions (not addressed in this work) but also because of *dependencies by omission*—statements not executed which would have executed if a change was made to the slicing criterion c and are thus semantically dependent on c [1].

SENSA [4] is a technique that uses sensitivity analysis and execution differencing to identify (a subset of) the statements semantically dependent on other statements. SENSA takes a program, a statement c, and a set of inputs, and reports the set of statements that it finds at runtime to be semantically dependent on c. To do this, SENSA runs each input multiple times.

978-1-4799-5843-6/14 $31.00 © 2014 IEEE
DOI 10.1109/SERE-C.2014.14

Each time, it modifies the values computed at c and records the states and occurrences of all other statements. By comparing this data for different runs, SENSA finds all the statements whose behavior changed due to those modifications. This is a partial but decidable way to find semantic dependencies—SENSA *shows the existence* of many semantic dependencies even though it *cannot prove* the absence of others.

III. BOUNDS OF DYNAMIC SLICES

Our approach for calculating lower and upper bounds of precision and recall of dynamic slices works in two steps. First, for a slicing criterion c and its corresponding dynamic forward slice, we compute the static slice and apply SENSA to c. Second, based on the static slice (*SS*), the dynamic slice (*DS*), and the SENSA result (*SensA*), we compute the lower and upper bounds of recall and the lower bound of precision of *DS*. (The upper bound of precision is 100% as it is undecidable to disprove semantic dependence for false positives.)

Figure 1 shows the relationships between the statement sets *SS*, *DS*, *SensA*, and semantic dependencies (*Sem*). All sets are subsets of *SS*; *SensA* is also a subset of *Sem*. All of these sets are known except for *Sem* which is undecidable. T_S is the intersection of *SensA* and *DS* (true positives found by SENSA).

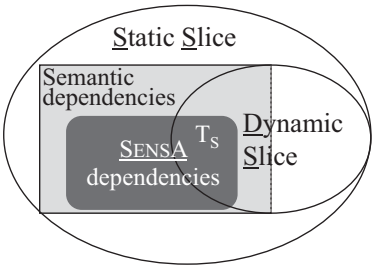

Fig. 1: Relationships between slices and SENSA results

The precision of *DS* is *at least* the fraction intersecting *SensA*. Because we cannot determine semantic dependence for the rest of *DS*, all statements in *DS* might be true positives for 100% upper bound of precision. The lower bound, however, corresponds to the case where T_S contains all true positives:

$$lower(precision) = \frac{|SensA \cap DS|}{|DS|} \quad (1)$$

The recall of *DS* is the fraction of *Sem* that *DS* covers. The lowest recall occurs when all statements in *SS* not in *DS* are semantic dependencies and the intersection of *Sem* and *DS* is exactly T_S. Thus, the lower bound of recall is:

$$lower(recall) = \frac{|SensA \cap DS|}{|SensA \cap DS| + |SS \setminus DS|} \quad (2)$$

On the other end of the spectrum, T_S can equal *DS* so that all statements in *DS* are semantic dependencies and, outside *DS*, only the statements in *SensA* are semantic dependencies. Therefore, the upper bound of recall is:

$$upper(recall) = \frac{|DS|}{|DS| + |SensA \setminus DS|} \quad (3)$$

TABLE I: Average bounds for precision and recall of slices.

Subject name	Subject size	Number of slices	Lower bound of precision	Lower bound of recall	Upper bound of recall
Schedule1	403	10	90.64%	93.41%	94.83%
NanoXML	3523	10	81.64%	23.95%	61.22%
Ant	19047	10	75.72%	3.16%	57.16%
BCEL	34839	10	68.07%	2.73%	29.46%
JMeter	35553	5	96.98%	0.33%	58.81%
PDFBox	59576	5	53.33%	0.04%	55.49%
Average for 50 slices:			**78.25%**	**24.69%**	**59.96%**

IV. RESULTS

We applied our approach to 50 dynamic forward slices on Java subjects of up to 60K lines of code, as shown in Table I. We *randomly* chose 10 slicing criteria for four subjects and 5 for the other two as *these take longer to analyze in our ongoing experiment*. Each row shows the average results per subject. The last row shows the average results for all 50 slices. We omit the upper bound of precision which is always 100%.

The results suggest that, for Ant, BCEL, and PDFBox (three of the largest subjects) there is no guarantee that the precision is greater than 53%, 68%, and 76% respectively. Our hypothesis is that the precision of dynamic slices for larger subjects might be lower than for smaller ones. Also, finding semantic dependencies might be harder in large subjects.

For the lower bounds of recall, a sharp decrease occurs from Schedule1 at 93% to NanoXML at 24% and another considerable decrease occurs from NanoXML to Ant at 3%. This trend continues down the table. We think this phenomenon is caused by increasing differences between the sizes of static slices and dynamic slices in larger subjects.

Meanwhile. the average upper bounds of recall, which are at most 61% and as low as 29% for all subjects but the smallest one, strongly suggest that dynamic slices cannot be trusted to achieve high recalls and thus provide complete results.

Naturally, we need to study many more slices, especially on the larger subjects, to be able to generalize our conclusions. However, our approach has already provided valuable insights.

V. CONCLUSION AND FUTURE WORK

We presented a new approach for assessing *for the first time* the accuracy of dynamic forward slicing with respect to real dependencies. Our initial results reveal some uncertainty on the ability of dynamic slices to model the runtime dependencies needed by important tasks in security and reliability (e.g., information-flow analysis and debugging). More studies on more slices and subjects are needed to explain and characterize this uncertainty and to support the design of new and better techniques that compute dynamic dependencies. Manual inspection and statistical methods will also be added to our approach to narrow down the bounds obtained in this work.

REFERENCES

[1] A. Podgurski and L. A. Clarke, "A formal model of program dependences and its implications for software testing, debugging, and maintenance," *Software Engineering, IEEE Trans. on*, vol. 16, no. 9, pp. 965–979, 1990.

[2] M. Weiser, "Program slicing," *IEEE Trans. on Softw. Eng.*, 10(4):352–357, Jul. 1984.

[3] B. Korel and J. Laski, "Dynamic Program Slicing," *Information Processing Letters*, vol. 29, no. 3, pp. 155–163, 1988.

[4] H. Cai, S. Jiang, Y.-J. Zhang, Y. Zhang, and R. Santelices, "SENSA: Sensitivity Analysis for Quantitative Change-impact Prediction," *Technical Report TR 2013-04*, CSE, U. of Notre Dame, Mar. 2013, 11pp.