On the Accuracy of Forward Dynamic Slicing and its Effects on Software Maintenance

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

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

Haipeng Cai
University of Notre Dame, U.S.A.
E-mail: hcai@nd.edu

Abstract—Dynamic slicing is a practical and popular analysis technique used in various software-engineering tasks. Dynamic slicing is known to be incomplete because it analyzes only a subset of all possible executions of a program. However, it is less known that its results may inaccurately represent the dependencies that occur in those executions. Some researchers have identified this problem and developed extensions such as relevant slicing, which incorporates static information. Yet, dynamic slicing continues to be widely used, even though the extent of its inaccuracy is not well understood, which can affect the benefits of this analysis.

In this paper, we present an approach to assess the accuracy of forward dynamic slices, which are used in software maintenance and evolution tasks. Because finding all actual dependencies is an undecidable problem, our approach instead computes bounds of the precision and recall of forward dynamic slices. Our approach uses sensitivity analysis and execution differencing to find a subset of all program statements that truly depend at runtime on another statement. Using this approach, we studied the accuracy of many forward dynamic slices from a variety of Java applications. Our results show that forward dynamic slicing can have low recall—for dependencies in the analyzed executions—and some potential imprecision. We also conducted a case study that shows how this inaccuracy affects a software maintenance task. To the best of our knowledge, ours is the first work that quantifies the intrinsic limitations of dynamic slicing.

Keywords—dynamic slicing; precision and recall; sensitivity analysis; execution differencing; software maintenance;

I. INTRODUCTION

Program slicing is a popular and widely used approach that identifies the parts of a program that are affected by a particular variable in some program statement (forward slicing) or that affect a particular variable in that statement (backward slicing) [1]–[4]. The statement-variable pair is called the slicing criterion, and the set of statements found by program slicing techniques is called the slice. Dynamic slicing [2], [4], [5] is a practical variant that computes slices for program executions [6]. Forward dynamic slicing (FDS) obtains the statements that are affected by some slicing criterion at runtime. FDS is used in software maintenance and evolution for a variety of tasks that include change-impact analysis [7], [8], testing [9], [10], fault localization [11], [12], comprehensiveness [13], error detection [14], and program merging [15].

Statements that truly affect one another in slices are said to semantically depend on one another [16]. Since static program analysis is undecidable [17], slicing computes many dependencies that cannot be instantiated at runtime. Such spurious dependencies cause the resulting slices to be large and difficult to use [18], [19] and imprecise [15], [20], [21]. On the bright side, static slicing is safe because it includes all semantic dependencies and, thus, has perfect recall.

Dynamic slicing, in contrast, produces slices that are unsafe because the analysis is based on selected executions and, thus, many semantic dependencies may be missed. Nevertheless, dynamic slicing is commonly used as a practical alternative to static slicing because its results are usually smaller and better represent the program behaviors that occur in selected, representative executions. In this paper, we focus on FDS.

Although it is known that dynamic slicing is unsafe by analyzing only a subset of all possible executions, it is less known that its results may inaccurately represent the semantic dependencies that occur in those executions (i.e., not just the dependencies in all other possible executions). For backward dynamic slicing, some researchers have identified this problem and developed (backward) relevant slicing [22], [23] which adds, statically, control decisions that could have affected the slicing criterion. Yet, dynamic slicing continues to be widely used, even though the extent of its inaccuracy is not well understood, which can limit for many tasks the benefits of using this analysis.\footnote{We defer to future work the definition and study of forward relevant slicing.}

Ideally, to study the accuracy of FDS, we would like to compute directly the precision and recall of the resulting slices in terms of dynamic semantic dependencies.\footnote{A dynamic semantic dependence is defined similarly to static semantic dependence [24], but constrained to the selected set of executions.} However, to identify dynamic semantic dependencies, even dynamic ones, we would need to try, for every execution, all possible replacements of the values assigned to the variables at the slicing criterion [24]. This is impractical, if not undecidable.

To overcome this problem, in this paper, we present a novel approach that computes safe bounds instead for the precision and recall of forward dynamic slices. Our main idea is to explore extensively the space of dynamic semantic dependencies by repeatedly changing values assigned at a slicing criterion to discover at least a subset of all dynamic semantic dependencies and thus under-approximate that set for that criterion. To complete the job, our approach also over-approximates the set of semantic dependencies using a (safe) static slicer. Naturally, the tightness of these bounds (but not their safety) depends on the quality of these two analyses.
Our approach uses SENS A [25], a sensitivity-analysis and execution-differencing tool that runs programs multiple times while modifying the slicing criterion C. Because we fix all sources of non-determinism in our experiments (i.e., we ensure the same execution for the same input), the statements whose behavior change in those runs are guaranteed to be dynamically semantically dependent on C. Using this large dependence-space exploration within these slices, the approach approximates from below the precision and recall of forward dynamic slices. In the limit, making all possible changes at C would yield the actual precision and recall.

Using this approach, we conducted an experiment on 100 forward dynamic slices using five nontrivial Java applications and a smaller program. We computed the bounds of precision, recall, and accuracy (F1 score) for all those slices. Our results indicate that, on average for these subjects, the lower and upper bounds of precision are 84% and 100% and the lower and upper bounds of recall are 14% and 53%, respectively. These results suggest that the inaccuracy of FDS can be a real problem for applications of this analysis and inform developers and researchers on what to expect from FDS.

To demonstrate how the (in)accuracy of FDS and its effects on client tasks can be estimated using our approach, we also carried out a case study where we estimated the usefulness of forward dynamic slicing for dynamic impact analysis [7].

This paper makes the following contributions:

• A conceptual analysis of the accuracy of forward dynamic slicing (FDS) and the factors that affect that accuracy.
• An approach for computing lower and upper bounds of precision and recall of forward dynamic slices.
• An empirical evaluation of the accuracy of FDS.
• A case study illustrating the effects of this accuracy.

II. BACKGROUND

In this section, we define semantic (undecidable) and syntactic (computable) dependencies. We also show how program slicing works and how SENS A finds semantic dependencies.

A. Semantic and Syntactic Dependencies

A program statement depends on another statement when the behavior of the latter affects the behavior of the former. Specifically, a statement $s_1$ is semantically dependent on a statement $s_2$ if, for some execution, changing a value computed at $s_2$ changes the values computed at $s_1$ or the execution frequency of $s_1$ [16]. For example, in Figure 1, for input $i=0$, any change in the value of $y$ computed at line 2 changes the evaluation of the condition at line 3 from true to false, preventing the execution of line 4 and causing line 6 to execute. Thus, lines 3, 4, and 6 semantically depend on line 2.

Syntactic (i.e., data and control) dependencies and their transitive closure, obtained via program analysis, are computable over-approximations of semantic dependencies. Factors such as the alias analysis used to compute these dependencies determine the precision of these over-approximations [16]. A statement $s_1$ is data dependent on some other statement $s_2$ if $s_1$ uses (i.e., reads from memory) a value that $s_2$ defines (i.e., writes to memory). We call statement $s_1$ control dependent on another statement $s_2$ if the evaluation of $s_2$ determines whether $s_1$ necessarily executes afterward or not.

The transitive closure of data and control dependencies from a statement $s$ over-approximates the set of semantic dependencies on $s$. Thus, syntactic dependencies are necessary but not sufficient conditions for semantic dependence. In other words, a syntactic dependence among two statements does not imply that they are semantically dependent. However, if it is determined that two statements are semantically dependent on each other, they are also statically syntactically dependent.

Figure 1 helps illustrates the differences between dependence types. Line 2 is data dependent on line 1, and line 3 is data dependent on line 2. Thus, line 3 is transitively data dependent on line 1. However, line 3 is not semantically dependent on line 1 as no change to $x$ in line 1 can affect evaluation of line 3—the behavior of line 3 depends only on line 2. Also, unlike syntactic dependencies, semantic dependencies are not transitive as line 2 is semantically dependent on line 1, and line 3 semantically depends on line 2, but line 3 does not semantically depend on line 1.

B. Dynamic Semantic Dependence

A special case of semantic dependence, which is a property of all possible executions of a program (i.e., a static notion) is a statement $s_1$ that is dynamically semantically dependent on a statement $s_2$. Such a statement $s_1$ is dynamically semantically dependent on $s_2$ if, for a specific set of executions, a change to any value computed at $s_2$ can change the computations at or occurrences of $s_1$ for those executions.

If $s_1$, however, changes its behavior only for changes in $s_2$ for other executions, then $s_1$ is not dynamically semantically dependent on $s_2$. Thus, dynamic semantic dependencies are a subset of semantic dependencies and are those that dynamic slicing, the subject of this paper, tries to compute.

A consequence of this dynamic definition is that a statement $s_1$ that does not execute for an input $I$ can be, nonetheless, dynamically semantically dependent on $s_2$ if a change can be made to $s_2$ such that, for that input $I$, $s_1$ executes. In other words, the non-execution of $s_1$ for that input is determined by control-flow decisions affected by $s_2$ at runtime.

C. Forward Dynamic Slicing (FDS)

Program slicing [1] is a fundamental technique for different software-engineering tasks including software maintenance.
and evolution. Slicing identifies a set of statements that are transitively data or control dependent on a specific statement (for forward slicing) or on which the statement is transitively data or control dependent (for backward slicing). Combining forward slicing and dynamic slicing into FDS, statements are collected that depend on a slicing criterion for a specific execution. For multiple executions, the union of their dynamic slices is a union slice [6]. (In this paper, we continue to use the term dynamic slicing even for multiple executions.)

Consider applying the FDS to the code in Figure 1 with input \( i = 0 \). Lines 1, 2, 3, and 4 are in the forward dynamic slice of line 1 because all those lines are data dependent on line 1 and line 4 is also control dependent on line 3. Lines 5 and 6 are not executed, and, thus, are not in the slice.

D. SENTA and Non-determinism

Using FDS, the SENTA approach identifies statements that are affected by a slicing criterion [25]. Essentially, SENTA performs an extensive exploration of the program execution states by modifying values of different variables in program statements to discover dynamic semantic dependencies. Naturally, SENTA reports only a subset of such dependencies, but this is 100% precise when applied to subjects for which all sources of non-determinism are fixed, as in our studies.

Figure 2 shows the SENTA process. The inputs are program \( P \), slicing criterion (statement) \( s \), and test suite \( T \). Optional inputs include the number of changes \( n \) to apply to \( s \), the change strategy (i.e., random or incremental), and valid value ranges for changes. For each test in \( T \), SENTA runs \( P \) consecutively \( n+1 \) times. First, SENTA executes \( P \) without changing \( s \)—the base run—and then executes \( P \) \( n \) times, each time with a different change to the value computed at \( s \). During execution, SENTA collects, per test, all \( n+1 \) execution traces, including all computed values (primitive and non-primitive) and compares the traces to find the statements that behave differently (different values or occurrences) from the base run.

SENTA offers two strategies to modify the values computed at the slicing criterion (we use both strategies). These are Random and Incremental. Random picks new random values, whereas Incremental picks new values by systematically incrementing and decrementing them. For numerical types, Incremental applies to the slicing criterion changes of +1, -1, +2, -2, and so on. For strings, Incremental increases and decreases their length and Unicode values. For all other non-numerical types, only Random is applicable [25].

Consider again the example of Figure 1 with \( T \) having one input, \( i = 0 \). SENTA first runs example without changes (base run) and then 20 times (default setting) with changes. The base run has execution trace \( \{ 1(1), 2(1), 3(true), 4(false), 7 \} \) which means that line 1 computes 1, line 2 computes 1, line 3 evaluates to true, and so on. If SENTA changes \( x \) to 2 in line 1 in a modified run, the trace is \( \{ 1(2), 2(2), 3(true), 4(true), 5, 7 \} \). Lines 1, 2, 4, and 5 differ in occurrences or values in these traces due to this change, so by definition they are dynamically semantically dependent on line 1 (Section II-B).

III. ESTIMATING THE ACCURACY OF FDS

In this section, we define precision and recall for forward dynamic slicing (FDS) as ideal metrics (undecidable) and we show how to obtain safe bounds for them (computable).

A. Precision and Recall

Based on the definition of dynamic semantic dependence in Section II-B, we analyze FDS using precision and recall, which are defined by Formulas 1 and 2, respectively. In these formulas, \( DSlice \) is the dynamic slice. \( TP_{ds} \) is the set of true positives, and \( FN_{ds} \) is the set of false negatives. Their relationships are depicted in the diagram in Figure 3. Positives are all statements in the dynamic slice and negatives are all other statements. True positives are the statements in the slice that dynamically semantically depend on the slicing criterion \( c \). False positives are all other statements in the slice—those not dynamically semantically dependent on \( c \).

Similarly, the true negatives are all statements that are not in the slice and are also not dynamically semantically dependent on the slicing criterion. Finally, false negative statements dynamically semantically depend on the slicing criterion for the used inputs even though they were not included in the slice. The union of false negatives and true positives is the set of all dynamic semantic dependencies. Precision reflects how correct FDS is when reporting dynamic semantic dependencies, whereas recall reflects how many such dependencies were correctly retrieved by dynamic slicing.

\[
\begin{align*}
\text{precision}_{DSlice} & = \frac{|TP_{ds}|}{|DSlice|} \quad (1) \\
\text{recall}_{DSlice} & = \frac{|TP_{ds}|}{|TP_{ds} \cup FN_{ds}|} \quad (2)
\end{align*}
\]

Our motivation for defining precision and recall of FDS is to understand the differences between its correctly-identified dynamic semantic dependencies, false dependencies, and missed dependencies. In information retrieval, recall is the ratio of the number of relevant items retrieved by a search divided by the total number of existing relevant items, whereas precision...
is the ratio of relevant items retrieved to the total number of items retrieved. Remember from Section II that both the dynamic slice and the set of semantic dependencies are subsets of the static slice. If all retrieved dependencies are in fact true dependencies and no false dependencies are reported, precision is 1. If all identified dependencies are false, precision is 0 and recall is 0. If all dynamic semantic dependencies are in the dynamic slice, then recall is 1 regardless of precision.

B. Accuracy Bounds

Figure 3 illustrates what we wish to obtain. Because computing semantic dependencies is an undecidable problem, we resort to approximations in the form of bounds. Figure 4 shows we can obtain bounds safely by depicting the relationships among dependencies, static and dynamic slices, and SENA results. Interestingly, a dynamic slice is neither a subset nor a superset of the dynamic semantic dependencies because it may include false dependencies and miss true dependencies.

Based on these relationships, we define Formulas 3, 4, and 5 for calculating lower bounds for precision and lower and upper bounds for recall of forward dynamic slices to approximate Formulas 1 and 2. Computing the upper bound of precision is impossible because proving that a statement is not semantically dependent on the slicing criterion. Thus, we always report the upper bound of precision as 100%.

No matter what dependencies are found by SENA as a subset of all semantic dependencies, this subset contains semantic dependencies only (we fix all sources of non-determinism in our studies). This property is important, since it guarantees that the accuracy bounds computed by our approach are true approximations in the limit—the actual values lie within these bounds. Thus, any current limitations of SENA and our slicers would affect only the tightness of the bounds but not the conclusions we can draw from them on FDS accuracy.

1) Lower Bound of Precision: The precision of a dynamic slice (DSlice) is the size of the true positives set divided by the size of the dynamic slice. The intersection of the SENA dependencies (SENA) with the dynamic slice is shown in Figure 4. These dependencies constitute a subset of the true positives for a forward dynamic slice (Figure 3). Therefore, the precision of the dynamic slice is at least the quotient of the size of this intersection and the size of the dynamic slice:

\[
\text{precision}_{\text{DSlice}} \geq \frac{|\text{SENA} \cap \text{DSlice}|}{|\text{DSlice}|} \tag{3}
\]

2) Lower and Upper Bounds of Recall: The recall of a dynamic slice is the size of the true positives set divided by the size of the union of true positives and false negatives. If the false negatives are many, the recall is small. Thus, for our lower bound of recall, we use a lower bound of the true positives and an upper bound of the false negatives.

As already discussed, a lower bound of the true positives is the intersection of SENA and the dynamic slice. For the upper bound of false negatives, any statement in the static slice is potentially semantically dependent on the slicing criterion. Thus, an upper bound of false negatives is the complement of the dynamic slice within the static slice (SSlice).

\[
\text{recall}_{\text{DSlice}} \geq \frac{|\text{SENA} \cap \text{DSlice}|}{|\text{SENA} \cap \text{DSlice}| + |\text{SSlice} \setminus \text{DSlice}|} \tag{4}
\]

Naturally, this lower bound is subject to the precision of the static slicer utilized. A context-insensitive slicer, for example, would cause a larger SSlice and thus a smaller lower bound of recall, but it would not affect the safety of the bound.

The upper bound of recall is obtained from an upper bound of true positives and a lower bound of false negatives. The upper bound of true positives we can use is the size of the whole dynamic slice (finding false positives is undecidable). The lower bound of false negatives is the intersection of SENA (subset of all dynamic semantic dependencies) and the complement of the dynamic slice within the static slice.

\[
\text{recall}_{\text{DSlice}} \leq \frac{|\text{DSlice}|}{|\text{DSlice}| + |\text{SENA} \setminus \text{DSlice}|} \tag{5}
\]
IV. EMPIRICAL EVALUATION

In this section, we present our study of the accuracy of 100 forward dynamic slices in, primarily, nontrivial Java applications using our approach of Section III.

A. Research Questions

Our research questions are the following.

RQ1: How accurate is forward dynamic slicing?

RQ2: How does the accuracy of forward dynamic slicing vary within a subject and across different subjects?

RQ3: Does the accuracy of a dynamic slice correlate with factors such as the size of the slice?

B. Experimental Setup

We carried out our experiment using six open-source Java applications described in Table I. These subjects vary in size, functionality, and complexity. For each application, the table shows its name, a short description, its size in non-comment non-blank lines of Java code (LOC), the size of its test suite, and the number of forward dynamic slices considered.

We obtained BCEL and PDFBox from the Apache project and the rest from the SIR repository [26]. From all possible dynamic slices across these subjects, we randomly selected ten slices for each subject smaller than 10KLOC and twenty slices for each subject. We run the study on a Linux desktop with a 3.40GHz Intel i7 CPU and 32GB of RAM.

We computed the forward dynamic union slices (see Section II-C) using our SENS A [25] tool and the slicers in our DUA FORENSICS analysis system [27]. Both tools analyze Java-bytecode programs using Soot [28], which translates bytecode into an intermediate representation called Jimple. Henceforth, we use Jimple instructions as the units for reporting the sizes of slices and for computing accuracy.

To ensure the determinism of our subjects for SENS A, we inspected their code and also run their test suites multiple times while comparing their execution histories. Whenever we found a difference, we located and determined the code that caused the difference without affecting its semantics.

C. Methodology

The experimental process consists of three steps outlined in Figure 5. The first step is to sample slicing criteria by selecting Jimple instructions at random that are executed by at least one test from the corresponding test suite. The second step is to compute the forward dynamic and static slices for each criterion and apply SENS A to it. Finally, we compute precision and recall bounds for the (union) forward dynamic slice.

In the first step, we discarded some randomly-selected criteria whose dynamic slices sizes were too small (less than 20 statements) and thus uninteresting or for which SENS A does not (directly) support modifications. For example, a conditional statement does not assign values to variables, so there are no assigned variables for SENS A to modify (SENS A can modify, instead, assignments of variables used in the condition). In such cases, we continued the random selection until finding a suitable set of slicing criteria.

The process of deciding whether a sampled statement matches a goal of our experiments as a slicing criterion is shown in Figure 6. First, check whether a statement is directly supported by SENS A (without loss of generality). If supported, check whether the size of the forward (union) dynamic slice from s is at least 20. If so, use s and its (union) dynamic slice for the study.

TABLE I

<table>
<thead>
<tr>
<th>Subject</th>
<th>Short description</th>
<th>Java LOC</th>
<th>Test cases</th>
<th># Forward dynamic slices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schedule1</td>
<td>priority scheduler</td>
<td>301</td>
<td>2,650</td>
<td>10</td>
</tr>
<tr>
<td>NanoXML</td>
<td>lean XML parser</td>
<td>3,521</td>
<td>214</td>
<td>10</td>
</tr>
<tr>
<td>Ant</td>
<td>build tool</td>
<td>19,047</td>
<td>112</td>
<td>20</td>
</tr>
<tr>
<td>BCEL</td>
<td>bytecode library</td>
<td>34,839</td>
<td>75</td>
<td>20</td>
</tr>
<tr>
<td>JMeter</td>
<td>performance tester</td>
<td>35,547</td>
<td>79</td>
<td>20</td>
</tr>
<tr>
<td>PDFBox</td>
<td>PDF library</td>
<td>59,576</td>
<td>32</td>
<td>20</td>
</tr>
</tbody>
</table>

Fig. 5. The process for computing accuracy bounds of (union) forward dynamic slices. Given a slicing criterion s and its forward dynamic slice, run SENS A on s to find a subset of semantic dependencies and the static slice of s. Then, use Formulas 3–5 to obtain the bounds.

Fig. 6. Process for deciding whether a statement is used for our study. First, check whether s is directly supported by SENS A (without loss of generality). If supported, check whether the size of the forward (union) dynamic slice from s is at least 20. If so, use s and its (union) dynamic slice for the study.
scalable than context-sensitive slicing [29]. We also checked that no external dependencies (e.g., via files) affect the results.

We used both the Random and Incremental strategies of SENA to identify as many semantic dependencies as possible. For each strategy, SENA applied 20 modified runs to each subject (the default) [25]. To see whether more runs make a difference, we increased the number to 50 for NanoXML but SENA found no more results. We limited each run to 30 minutes to avoid infinite loops caused by the SENA changes. The tool discards timed-out runs and performs additional runs to reach the required 20 modified runs per strategy.

D. Effect of Slice Sizes

A key property of a slice is its size, which may have an effect on its accuracy. Table II shows the average size, in Jimple lines, of the dynamic and static slices per subject. For the dynamic slices, the table also shows the minimum and maximum sizes. The table indicates that the average size of the dynamic slices ranges across subjects from about 100 to 1,000 Jimple instructions, whereas the average size of the static slices ranges from about 200 to 60,000 Jimple instructions.

Within each subject, the dynamic-slice sizes varied considerably. For example, for BCEI, these sizes range between 65 and 2,523 instructions. In contrast, the static slice sizes do not vary much within each subject. The likely reason for this phenomenon is that the static slices are context insensitive and that, consistently with other studies, those slices include a large fraction of the code for each subject application [18].

E. Results and Analysis

1) RQ1: Overall Accuracy of Forward Dynamic Slicing: Table III presents the average, minimum, maximum, and median of the precision and recall bounds for all subjects. Because the upper bound of precision is always 100% (see Section III-B), we omit this bound. In addition, Table IV shows the bounds for the F1 measure of accuracy, which is the harmonic mean of the precision \( p \) and recall \( r \), computed as \( 2 \times p \times r / (p + r) \). We obtained the lower and upper bounds of the F1 accuracy measure \( p \) dynamic slice using the lower and upper bounds of the precision and recall, respectively.

To illustrate, consider the results for Schedule1. The average lower bound of precision is 91.6%, the average lower bound of recall is 93.2%, the average upper bound of recall is 94.5%, and the average lower and upper bounds of F1 are 92.0% and 97.1%. All averages are over 90%, which indicates that dynamic slicing is accurate for this subject, even though the upper bound of recall is less than 100% indicating that at least 5.5% of the statements that are dynamically semantically dependent on the slicing criteria are missing from the slices.

For all 100 slices, the average lower bound of precision is 83.6%, which suggests the overall precision of forward dynamic slicing is generally high. However, the average upper bound of recall is 52.8%, which means that the recall of forward dynamic slicing can be quite deficient.

For individual subjects, the average lower bounds of precision are relatively high. For only two subjects those values are below 80%, and the median lower bounds are all above 90%. These values show that the majority of the studied dynamic slices have a guaranteed high precision. However, for several dynamic slices, the lower bounds of precision are low—as low as 0.1%. Although the upper bound of precision is still 100% and we do not know where the real precision is within those bounds, these low lower bounds indicate that it cannot be guaranteed that precision is high in all cases.

2) RQ2: Variation of Dynamic Slicing Accuracy: As we just discussed, the precision is generally high and the recall is generally low. However, the bounds of precision and recall vary not only across different subjects, but also within a single subject. In addition to Table III, we provide Figures 7 and 8 for a visual breakdown of our results for precision and recall, respectively. In Figure 7, each range of precision is denoted by a line from its lower bound to 100% (the upper bound). Figure 8 is formatted in the same way, but the upper bounds in it are not all 100%. For example, out of the 10 ranges for Schedule1 in Figure 8, six are very short and near the top and the rest are located mostly between 80% and 90%.

The average precision and recall bounds vary across the subjects, as Table III reveals. First, for the lower bounds of precision, the smallest two subjects (Schedule1 and NanoXML) have bounds over 88%. For the remaining subjects, other than JMeter, the precision lower bounds are below 81%. The exception of JMeter shows that the size of the subject is not the only factor that affects the precision of dynamic slicing. Second, the differences among the average bounds of recall are considerable. The average lower bound of recall for Schedule1 is 93.2%, for NanoXML is 31.1%, and for the rest it is less than 6%. For the average upper bounds of recall, the numbers vary among the larger subjects, from 36.1% in JMeter to 61.3% in Ant. Similar to the case of lower bounds, Schedule1 and NanoXML have greater upper bounds of recall.

The bounds vary not only across the subjects, but also within each subject. Figure 7 shows that, for all subject applications with the exception of Schedule1, there are one or more extreme cases where the lower bounds of precision are quite low in comparison with the other dynamic slices in the same subject. For example, for NanoXML, the lower bound of the precision of the seventh slice is less than 20% whereas the lower bounds of precision for the other slices in NanoXML are greater than 90%. As the size of the subject increases, the number of extreme cases tends to grow. In consequence, despite having an average precision of at least 83.6%, it is possible that dynamic slicing has a rather low precision in some cases.

As for recall, Figure 8 shows that dynamic slicing has
TABLE III

<table>
<thead>
<tr>
<th>Subject Name</th>
<th>Lower bound of precision</th>
<th>Upper bound of recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Average</td>
<td>Min</td>
</tr>
<tr>
<td>Schedule1</td>
<td>91.6%</td>
<td>82.6%</td>
</tr>
<tr>
<td>NanoXML</td>
<td>88.7%</td>
<td>17.4%</td>
</tr>
<tr>
<td>Ant</td>
<td>78.9%</td>
<td>24.7%</td>
</tr>
<tr>
<td>BCEL</td>
<td>74.0%</td>
<td>1.4%</td>
</tr>
<tr>
<td>JMeter</td>
<td>93.7%</td>
<td>60.0%</td>
</tr>
<tr>
<td>PDFBox</td>
<td>81.0%</td>
<td>0.1%</td>
</tr>
<tr>
<td>All slices:</td>
<td>83.6%</td>
<td>0.1%</td>
</tr>
</tbody>
</table>

TABLE IV

<table>
<thead>
<tr>
<th>Subject Name</th>
<th>Lower bound of F1 measure</th>
<th>Upper bound of F1 measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Average</td>
<td>Min</td>
</tr>
<tr>
<td>Schedule1</td>
<td>92.0%</td>
<td>84.6%</td>
</tr>
<tr>
<td>NanoXML</td>
<td>43.1%</td>
<td>0.4%</td>
</tr>
<tr>
<td>Ant</td>
<td>8.4%</td>
<td>0.3%</td>
</tr>
<tr>
<td>BCEL</td>
<td>3.6%</td>
<td>0.1%</td>
</tr>
<tr>
<td>JMeter</td>
<td>6.6%</td>
<td>0.1%</td>
</tr>
<tr>
<td>PDFBox</td>
<td>0.2%</td>
<td>0.0%</td>
</tr>
<tr>
<td>All slices:</td>
<td>16.1</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

Fig. 7. Precision ranges of the dynamic slices. The subjects and their slices are listed on the X axis.

TABLE V

<table>
<thead>
<tr>
<th>Coeff. of precision</th>
<th>Lower bound of recall</th>
<th>Upper bound of recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic</td>
<td>-0.184</td>
<td>0.0673</td>
</tr>
<tr>
<td>Static</td>
<td>0.195</td>
<td>0.0514</td>
</tr>
</tbody>
</table>

a high recall for Schedule1 (at least 78%). For the other subjects, however, many lower bounds of recall are near 0%. For the larger subjects in particular (Ant, BCEL, JMeter, and PDFBox), the lower bounds of recall remain below 20% and most of them are less than 10%, while the upper bounds vary between 10% and 100%. This trend suggests that forward dynamic slicing can miss more semantic dependencies in larger applications, as Table III also indicates.

3) RQ3: Correlation Between Slice Size and Accuracy: It is also important to study factors that might influence accuracy. Therefore, we investigated whether the dynamic slice accuracy statistically correlates with dynamic or static slice size. To that end, we used the Spearman test [30], which is non-parametric (i.e. no assumptions on data distribution). Table V lists the coefficients and P-values of the Spearman rank correlation test. A coefficient of 1 or -1 indicates perfect correlation whereas 0 indicates no correlation. Sub-columns Coeff. list the coefficients and sub-columns P-val. list the P-values. Rows Dynamic and Static show the correlation of the bounds with the forward dynamic and static slice sizes, respectively.

The P-values of the correlations between lower bounds of precision and dynamic and static slice sizes are 0.067 and 0.051, respectively. As they not less than 0.05, we cannot conclude with confidence that lower bounds of precision correlate with slice sizes. The correlation coefficients -0.184 and 0.195 suggest that the lower bound of precision tends to decrease slightly as dynamic slice size increases and tends to increase slightly as the static slice size increases.

For the recall bounds, the P-values are tiny, which strongly
indicates that the bounds of recall correlate with the dynamic and static slice sizes. The signs of the coefficients for the two bounds indicate that recall increases when dynamic slice size increases and decreases when static slice size increases.

These results support two intuitions about recall (or at least its bounds): (1) larger dynamic slices capture more semantic dependencies than smaller slices and (2) larger static slices provide more room for false negatives to occur (see Figure 3).

F. Threats to Validity

The main threat to the validity of our study is the potential looseness of the bounds of precision and recall. Because it is impossible to obtain the actual precision and recall for a slice, those values can lie anywhere within the bounds we computed. Nevertheless, we were able to draw sound and interesting conclusions from the bounds we obtained.

The primary cause for this threat is the uncertainty about the extent of the results of SENA. SENA cannot guarantee that it will find a sizable subset of dynamic semantic dependencies. We countered this threat in three ways. First, we chose applications with nontrivial test suites that exercise a variety of behaviors so that SENA can identify, at least, the most common semantic dependencies. Second, we execute SENA 20 times (and tried it up to 50 times) which we found experimentally to maximize the benefits we can obtain from SENA. Finally, we chose different, independent subject applications to reduce the risk of obtaining skewed results.

An internal threat to validity is the potential existence of errors in our implementation of the slicers and SENA. The possibility of major errors, however, is low because these tools have been extensively tested and used for this and other experiments. Another internal threat is possibility of errors in our use of the slicers and SENA, which we reduced by manually checking many of the results.

The main external threat to validity is our choice of subjects, slicing criteria, and test suites. Naturally, our results might not reflect the accuracy of forward dynamic slices in other subjects. To reduce this threat, we chose a variety of subjects supported by nontrivial test suites. We also sampled a larger number of slices for subjects larger than 10KLOC for our study to increase representativeness within those subjects.

A construct threat to validity is in the test of correlation between slice sizes and accuracy. We used precision and recall bounds to approximate dynamic slice accuracy, although the correlation between bounds and slice sizes might not represent the correlation between actual accuracy and slice sizes. However, correlations with actual accuracy are undecidable.

For conclusion validity, we used the F1 measure for accuracy that regards precision and recall equally. We also sampled slicing criteria randomly. All of this might or might not be representative of what developers typically need or do.

V. Case Study: Impact Analysis

To illustrate how the accuracy of dynamic slicing can affect its applications, we chose a forward dynamic slice from BCEL for dynamic impact analysis. BCEL is an Apache library for manipulating Java bytecode. We used a test case from its test suite. The slicing criterion is part of the creation of a key string for a field constant in method addFieldref, which adds a field to the constant pool (cp_table) of a classfile. This key string is used to check whether the parsed class already has the field constant. The test checks the addition of annotations to a class. It creates a HelloWorld class, adds two annotations to the class, and checks the serialization of that class.

Fig. 9. Code snippet from BCEL for our case study. ConstantPool is a table of constants in a parsed class. Method addFieldref adds a field to ConstantPool if it does not yet contain that field.
shows the slicing criterion (line 3) and surrounding code. For dynamic impact analysis, we manually identified all possible impacts of changing the slicing criterion when running this test, and compared our findings with the forward dynamic slice. For this test, addFieldref is called twice, which adds java.lang.System.in and java.lang.System.out to cp_table. Each time, the slicing criterion generates part of the key string for the table. The two fields have the same class name, so the strings that the slicing criterion generates are the same. Even if the slicing criterion generated different strings, the behavior of adding the fields would not be affected. However, if the slicing criterion assigns null to key, the program crashes immediately at line 4. Hence, there are two possible impacts of changing the slicing criterion: an immediate crash and two key changes in cp_table.

Based on our formulas of Section III-B, the lower bound of precision and the lower and upper bounds of recall of the dynamic slice are 100%, 2.4%, and 33.1%, respectively. Manually, we found that the dynamic slice is 100% precise, although almost 97% of those statements are semantically dependent on the slicing criterion simply because key can be assigned null—something that SENSA also finds. Also, because the SENSA results include all statements executed after line 3, there are no other false negatives that we could find manually. Thus, the actual recall is only 33.1%.

It seems unlikely, however, that a developer would change line 3 to assign null to key. If we discard that possible change in our investigation, we find that the execution does not change except for values in cp_table. To investigate the accuracy of the dynamic slice in that practical case, we made SENSA ignore this potential change and obtained 3%, 0.1%, and 100% for the lower bound of precision and the lower and upper bounds of recall, respectively. Manually, we found that in this case the precision is actually 3% because all statements not found by SENSA as dependent are, in fact, false positives. An example of a false positive is line 6: although key_str is transitively data dependent on line 3, its outcome for this test is always true and cannot change. In contrast, we found no false negatives for this scenario, so the recall is 100%.

This case study shows that the inaccuracy of dynamic slicing can have serious effects on impact analysis.

VI. RELATED WORK

Researchers have addressed the size of slices and practicality of slicing methods [18], [31]–[35] and the relationship between syntactic and semantic aspects of dependence analysis [33], [36]–[38]. That line of work assesses the relative precision of slicing techniques by comparing the sizes of the resulting slices whenever it is appropriate for safe slices. However, such approaches do not measure the actual accuracy of slices. Therefore, the approach we presented in this paper is complementary to the size-based comparison methods.

Empirical studies of program have already been performed to compare typical differences in the sizes of slices produced by different techniques. Binkley et al. [18] studied static program slicing by computing slices of all possible slicing criteria in 43 programs. The study considers five factors in slicing techniques that affect the resulting slice sizes, such as whether a program slicer takes calling context into account. Another study conducted by Binkley et al. [32] addresses the performance of slicing by comparing six optimization techniques in terms of memory usage and computation time.

To address shortcomings of (backward) dynamic slicing in particular, researchers have proposed unions of dynamic slices [6] (which we used in this paper and simply call “dynamic slices”), relevant slices [22], [23] which incorporate potential dependencies on predicates that could have affected the slicing criterion, and statement-deletion based methods such as critical slicing [39] and observation-based slicing [40]. In this paper, we decided to study the more commonly-used, traditional form of (union) dynamic slicing.

For other kinds of program slicing that are not based on statement slicing [33], [35], empirical studies also use slice size as the only metric of slice precision. Androustopoulos et al. [35] carried out an empirical study of dependences in extended finite state machines. In this study, slice sizes are used to demonstrate the properties of dependencies in the studied models. Binkley et al. [33] studied slices based on concept-slicing criteria. Such criteria, which are generated by a technique called concept assignment [41]. Similar to this paper, that work analyzed six subjects to describe the relationship between slicing criteria and slice size. Unlike our work, it did not analyze the bounds of precision and recall.

Regarding slicing and semantics, Mastroeni et al. [36] construct a dependence graph where a dependence can be defined using abstract semantics. In their framework, slices can be computed in terms of abstract semantic dependencies. Other research has also focused on semantics in slicing [38]. That work is purely theoretical and relates the formal concept of semantics to imperative languages by proving the semantic correctness of slicing algorithms in Weiser’s framework.

VII. CONCLUSION AND FUTURE WORK

Forward dynamic slicing is used in many software-engineering tasks. In this paper, we presented a novel approach to estimate the accuracy of forward dynamic slices by computing bounds for their precision and recall using sensitivity analysis and static slicing. We experimented with 100 forward dynamic slices primarily from nontrivial Java applications.

Our results show that forward dynamic slicing suffers from low recall of dynamic semantic dependencies and some imprecision. In particular, almost two thirds of the studied forward dynamic slices miss 68% or more semantic dependencies. In addition, we conducted a case study that illustrates how the imprecision of forward dynamic slicing can affect software maintenance in practice. To the best of our knowledge, ours is the first work that quantifies the limitations of dynamic slicing.

In summary, our results and analysis strongly indicate that the accuracy of dynamic slicing is quite insufficient. Accuracy varies not only across different slicing criteria in the same subject but also across different subjects.
Future improvements of our approach include extending SENS to avoid indirectly analyzing conditional statements. With this and other enhancements to SENS, we will get better bounds of accuracy. We will also incorporate a context-sensitive static slicer to further improve those bounds. Our future research on dynamic-slice accuracy will include three tasks. First, we will use manual inspection to classify statements not reported by SENS as semantically dependent. Although it is impractical to perform this task for all such statements, random samples of those statements would be analyzed by human participants to estimate tighter bounds statistically. Second, we will analyze the accuracy of other types of slicing, such as relevant slicing [22, 23], and critical slicing [39]. Finally, we will further investigate how the inaccuracy of dynamic slices affects the use of those slices in software-maintenance tasks by conducting more case studies.

ACKNOWLEDGMENTS

This work was partially supported by ONR Award N000141410037 to the University of Notre Dame.

REFERENCES