MOBS: Multi-Operator Observation-Based Slicing using Lexical Approximation of Program Dependence

Seongmin Lee
KAIST
Daejeon, Republic of Korea
bohrok@kaist.ac.kr

David Binkley
Loyola University Maryland
Baltimore, United States
binkley@cs.loyola.edu

Nicolas Gold
University College London
London, United Kingdom
n.gold@ucl.ac.uk

Syed Islam
University of East London
London, United Kingdom
syed.islam@uel.ac.uk

Jens Krinke
University College London
London, United Kingdom
j.krinke@ucl.ac.uk

Shin Yoo
KAIST
Daejeon, Republic of Korea
shin.yoo@kaist.ac.kr

ABSTRACT
Observation-Based Slicing (ORBS) is a recently-introduced program slicing technique based on direct observation of program semantics. Previous ORBS implementations slice a program by iteratively deleting adjacent lines of code. This paper introduces two new deletion operators based on lexical similarity. Furthermore, it presents a generalization of ORBS that can exploit multiple deletion operators: Multi-operator Observation-Based Slicing (MOBS). Empirical evaluation of MOBS using three real world Java projects finds that the use of lexical information improves the efficiency of ORBS: MOBS can delete up to 87% of lines while taking only about 33% of the execution time with respect to the original ORBS implementation.

1 OBSERVATION-BASED SLICING
ORBS [1] slices a program by iteratively attempting a deletion operator on its source code. Given source line \( l \), a deletion operator checks whether a set of lines, related to \( l \), can be safely deleted with respect to the given slicing criterion. If the source code after deletion either fails to compile or changes the value trajectory of the slicing criterion when executed using the given test suite, the deletion is rejected. Otherwise, ORBS accepts the deletion and moves on.

The original ORBS implementation [1] (W-ORBS), uses a window-deletion operators, \( Dw \). which handles consecutive source lines that can only be deleted together. ORBS’s weakness is its scalability; to delete \( k \) lines, ORBS needs at least \( k \) deletion attempts.

2 ORBS WITH LEXICAL SIMILARITY
Our new deletion operators are based on the intuition that if a source line can be deleted with respect to a given slicing criterion, then there are likely other lexically similar lines that can also be deleted. We introduce the two lexical deletion operators: DVSM and Dlda, based on two models which both can represent text documents as numerical vectors: Vector Space Model (VSM) and Latent Dirichlet Allocation (LDA). Each deletion operator chooses a set of lines to be deleted that are beyond the threshold of certain similarity calculated by the model.

To evaluate their effectiveness, we present variations of ORBS that use the newly-designed operators: VSM-ORBS and LDA-ORBS. VSM-ORBS and LDA-ORBS share distinguishing features that may yield advantages over the existing W-ORBS in terms of efficiency.

Algorithm 1: MOBS
\[
\begin{align*}
\text{input} & : \text{Source program } P = \{l_1, \ldots, l_n\}, \text{ Slicing criterion } (v, I, F), \\
& \text{ Set of deletion operators } D = \{D_1, \ldots, D_n\}, \text{ Slicing Strategy } S, \text{ Static Proportion } R, \text{ Proportion Updater } U \\
\text{output} & : \text{A slice of } P \text{ for } (v, I, F) \\
\text{InitOperator} & : \text{Setup } P, v, I \\
V & \leftarrow \text{EXECUTE}(\text{Build}(O), I) \quad \triangleright \text{ Insert a slicing criterion} \\
D & \leftarrow \text{InitOperator}(D, S, R) \quad \triangleright \text{ Obtain the oracle} \\
\text{repeat} & \quad \text{Set the selection prob.} \\
\text{deleted} & \leftarrow \text{False} \\
\text{for} i & \leftarrow \text{LENGTH}(O) \text{ to } 1 \text{ do} \\
D & \leftarrow \text{SELECTOperator}(D) \\
O', \text{line} \_\text{cnt}, \text{status} & \leftarrow D(O, V, i, F) \quad \triangleright \text{ Delete} \\
D & \leftarrow U(D, D, \text{status}, \text{line} \_\text{cnt}) \quad \triangleright \text{ Update the prob.} \\
\text{if} \text{ status } = \text{success} \text{ then} \\
O, \text{deleted} & \leftarrow O', \text{True} \quad \triangleright \text{ Accept the deletion} \\
\text{end} \\
\text{end} \\
\text{until} & \sim \text{deleted} \\
\text{return} & O
\end{align*}
\]

Algorithm 1 presents MOBS. The function InitOperator initializes the deletion operator probabilities. The function SelectOperator chooses a deletion operator to apply at each line using roulette-wheel selection [2] based on operator proportions. Once
chosen, the speculative deletion is the same as that done by ORBS except that MOBS updates the operator proportions using updater, \( U \), which is specific to each operator selection strategy.

There are two kinds of operator selection strategies: Fixed Operator Selection (FOS) and Rolling Operator Selection (ROS). FOS uses pre-defined operator proportions for an entire slice. The proportions are initialized in one of three ways: uniform value, using the number of successful deletions (applicability), using the number of lines deleted (affect). In contrast, ROS updates the proportion after each deletion attempt. The proportion updater \( U \) for ROS changes operator proportions, which have been initialized with a uniform value, based on the result of deletion.

4 EXPERIMENTAL SETUP

Our empirical studies are designed to answer the following research questions:

**RQ1. Lexical Deletion Operators:** How efficient/effective is VSM-ORBS, LDA-ORBS compared to W-ORBS?

**RQ2. MOBS:** How efficient/effective is MOBS compared to W-ORBS?

We use three real world Java projects in our empirical studies: commons-csv, commons-app, and Guava which is a core Java library developed by Google. We choose three slicing criteria for each Apache projects three slicing criteria from each sub-package from Guava we study: common.escape and common.net.

The library of deletion operators used by ORBS variants are:
- W-ORBS: Dw\( ^k \) for deletion window size \( k = 1, 2, 3, \) and 4
- VSM-ORBS: Dvs\( \gamma \) for threshold \( \gamma = 0.6, 0.7, 0.8, \) and 0.9
- LDA-ORBS: Dlda\( ^\gamma \) for threshold \( \gamma = 0.6, 0.7, 0.8, \) and 0.9

MOBS uses all of the aforementioned operators. Due to the stochastic operator selection, we repeat MOBS runs 20 times.

5 RESULTS

Table 2 shows the result of the operator efficiency comparisons between W-ORBS, VSM-ORBS, and LDA-ORBS. The results shows the average of 3 slicing criteria for each subject. Overall, VSM-ORBS and LDA-ORBS delete 35.3% and 26.1% of the number of lines deleted by W-ORBS, respectively. However, VSM-ORBS uses only 12.1% of compilations and 25.0% of executions of W-ORBS, resulting in only 19.7% of the execution time of W-ORBS. Similarly, LDA-ORBS uses 11.4% of compilations, 18.0% of executions, and takes 18.5% of the execution time of W-ORBS.

Table 3 shows the average result of the efficiency/effectiveness comparisons between W-ORBS, and MOBS with the four different operator selection strategies. For all results, MOBS is terminated after the same number of iterations W-ORBS required to terminate.

While all the MOBS variants slices the program more efficiently than W-ORBS, ROS-MOBS performs slightly better than others.

Overall, MOBS deletes about 79% of the lines W-ORBS deletes, using about one third of the execution time W-ORBS requires.

6 CONCLUSION

This paper makes two novel technical contributions. First, we present a generalisation of observational slicing that can exploit multiple deletion operators. Second, we introduce lexical deletion operators that exploit lexical similarities between source code lines to improve the efficiency of ORBS. MOBS is the resulting observational slicer that uses multiple deletion operators including the existing operators and the newly-introduced lexical deletion operators.

The results of our empirical evaluation of MOBS using three real world Java programs suggest that MOBS can significantly improve the efficiency of W-ORBS: it can delete about 79% of the lines deleted by W-ORBS, while taking only about a third of the execution time.

REFERENCES
