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). An 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.

CCS CONCEPTS

• **Software and its engineering** → **Dynamic analysis**; *Software testing and debugging*;

ACM Reference Format:
Seongmin Lee, David Binkley, Nicolas Gold, Syed Islam, Jens Krinke, and Shin Yoo. 2018. MOBS: Multi-Operator Observation-Based Slicing using Lexical Approximation of Program Dependence. In *ICSE '18 Companion: 40th International Conference on Software Engineering Companion, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3183440.3194981

1 OBSERVATION-BASED SLICING

ORBS [1] slices a program by iteratively applying a *deletion operator* on its source code. Given a line of source code, l, a *deletion operator* checks whether a set of lines, related to l, can be safely deleted w.r.t. a slicing criterion. If, after deletion, the code either fails to compile or changes the value trajectory of the criterion when executed using the given test suite, the deletion is rejected. Otherwise, ORBS accepts the deletion and moves on to the next line.

The original ORBS implementation [1](W-ORBS), uses a window-deletion operators, Dw, which attempts to delete consecutive source code lines that can only be deleted together. This effectively limits the scalability of ORBS, as it needs to make at least k deletion attempts to delete k lines.

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3194981

2 ORBS WITH LEXICAL SIMILARITY

Our new deletion operators are based on the intuition that, if a line of source code can be deleted with respect to a given slicing criterion, then there may be other lexically similar lines that can also be deleted. We introduce two lexical deletion operators, Dvsm and Dlda, based on two models that are used to represent textual documents as numerical vectors: Vector Space Model (VSM) and Latent Dirichlet Allocation (LDA). Treating each line of source code as a document, these deletion operators choose a set of lines within the given threshold of certain similarity (calculated by their models), to be deleted together.

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 the following features that may yield advantages over the existing W-ORBS in terms of efficiency. First, there is no limit to the number of lines that can be deleted in a single deletion attempt. Second, they can delete non-consecutive lines. Finally, during a single iteration, only one deletion is attempted at each slicing point.

3 MOBS: MULTI-OPERATOR OBSERVATIONAL SLICING

ORBS now has multiple deletion operators at its disposal. Each attempts to delete different parts of the code based on different criteria, and thus brings different results. A method for selecting an appropriate operator is therefore required. We hereby introduce MOBS: Multi-operator Observational Slicing, which selectively applies multiple deletion operators while slicing.

Algorithm 1 presents MOBS. The function InitOperator initializes the deletion operator selection 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

Algorithm 1: MOBS

input : Source program $\mathcal{P} = \{l_1, \ldots, l_n\}$, Slicing criterion $(v, l, \mathcal{I})$, Set of deletion operators $\mathcal{D} = \{\mathrm{D}_1, \ldots, \mathrm{D}_n\}$, Slicing Strategy S, Static Proportion R, Proportion Updater U
output : A slice of $\mathcal{P}$ for $(v, l, \mathcal{I})$

```
O ← Setup(P, v, l)                          ▷ Insert a slicing criterion
V ← Execute(Build(O), I)                    ▷ Obtain the oracle
D ← InitOperator(D, S, R)                   ▷ Set the selection prob.
repeat
    deleted ← False
    for i ← Length(O) to 1 do
        D ← SelectOperator(D)
        O', line_cnt, status ← D(O, V, i, I)        ▷ Delete
        D ← U(D, D, status, line_cnt)               ▷ Update the prob.
        if status = success then
            O, deleted ← O', True                   ▷ Accept the deletion
until ¬deleted
return O
```

Table 1: Comparison of Number of Compilations (C), Number of Test Executions (E), Execution Time (sec) (T), and Number of Deleted Lines (D) between W-ORBS, VSM-ORBS, and LDA-ORBS

Subject	W-ORBS				VSM-ORBS (γ = 0.9)				LDA-ORBS (γ = 0.9)			
	C	E	T	D	C	E	T	D	C	E	T	D
commons-cli	21,707	2,398	33,121	1083	2,525	402	6,148	272	2,191	363	5,680	245
commons-csv	14,645	1,338	27,297	817	1,619	242	4,123	213	1,502	177	3,682	138
guava-escape	6,282	441	10,456	259	741	105	1,825	113	753	98	1,710	106
guava-net	12,511	816	22,202	887	1,715	331	5,749	405	1,650	169	5,499	209

uses pre-defined operator proportions for an entire slice. The proportions are initialized in one of the following 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

We ask the following research questions:

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

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

We use three real world Java projects in our empirical study: `commons-cli` (2,081 NCLOC, 26 test cases) and `commons-csv` (1,504 NCLOC, 13 test cases) from Apache Commons Project, 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` (590 NCLOC, 6 test cases) and `common.net` (1,569 NCLOC, 8 test cases).

The library of deletion operators used by ORBS variants are: W-ORBS with Dw^k where deletion window size k = 1, 2, 3, and 4, VSM-ORBS with Dvsm^γ where threshold γ = 0.6, 0.7, 0.8, and 0.9, and LDA-ORBS with Dlda^γ where threshold γ = 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 1 shows the result of the operator efficiency comparisons between W-ORBS, VSM-ORBS, and LDA-ORBS. 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 2: Statistics on Number of Deleted Lines (μ_{del}), Execution Time (μ_{time}), Seconds per Deletion (μ_{spd}), and Speed Up ratio w.r.t W-ORBS by W-ORBS and MOBS

Criteria	Strategy	μ_{del}	μ_{time}	μ_{spd}	Speedup
commons-cli	ROS-MOBS	1051	20533	19.89	2.76
	FOS-app-MOBS	957	23697	25.32	2.40
	FOS-aff-MOBS	969	21690	22.89	2.62
	FOS-uni-MOBS	951	23653	25.31	2.40
	W-ORBS	1255	56897	46.01	1.00
commons-csv	ROS-MOBS	665	12850	19.86	3.61
	FOS-app-MOBS	618	14862	24.55	3.11
	FOS-aff-MOBS	625	14103	22.97	3.26
	FOS-uni-MOBS	606	13531	22.68	3.39
	W-ORBS	797	46008	58.78	1.00
guava-escape	ROS-MOBS	213	5172	24.75	3.17
	FOS-app-MOBS	195	5146	26.64	3.21
	FOS-aff-MOBS	201	5213	26.55	3.11
	FOS-uni-MOBS	210	5143	24.89	3.17
	W-ORBS	264	16249	63.01	1.00
guava-net	ROS-MOBS	788	11854	15.17	2.67
	FOS-app-MOBS	724	11725	16.23	2.73
	FOS-aff-MOBS	738	12362	16.88	2.55
	FOS-uni-MOBS	730	12702	17.52	2.49
	W-ORBS	917	31645	35.03	1.00

Table 2 shows the average result of the efficiency/effectiveness comparisons between W-ORBS, and MOBS with the four different operator selection strategies. We terminate MOBS 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 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

[1] David Binkley, Nicolas Gold, M. Harman, Syed Islam, Jens Krinke, and Shin Yoo. 2014. ORBS: Language-Independent Program Slicing. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2014)*. 109–120.

[2] David E. Goldberg. 1989. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley, Reading, MA.