Poster: Subsuming Mutation Operators

Huan Lin
State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications
Beijing, China
linhuan@bupt.edu.cn

Yawen Wang*
State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications
Beijing, China
wangyawen@bupt.edu.cn

Yunzhan Gong
State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications
Beijing, China
gongyz@bupt.edu.cn

ABSTRACT

Though minimal mutations were widely studied in recent years, the operators that produce these mutants remain largely unknown. This poster develops a coverage-based approach to identify them by defining *Subsuming Mutation Operators* (SMOs), the operators that cover all subsuming ones with minimal number of mutants. We then performed an empirical study on 61,000 mutants of 14 programs to determine SMOs among Proteum operators and investigate their properties. The results show only 17 of 82 operators are SMOs and they are not equally significant. For example, OLRN produces only 0.5% of subsuming mutants while VDTR accounts for almost 20%. A set of operators that are efficient at producing subsuming mutants are identified in our study and presented in this paper.

CCS CONCEPTS

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

KEYWORDS

Minimal mutation, subsuming mutation operators, coverage

ACM Reference Format:
Huan Lin, Yawen Wang, and Yunzhan Gong. 2018. Poster: Subsuming Mutation Operators. 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/10.1145/3183440.3194948

1 INTRODUCTION

Mutation testing has long suffered from the high costs introduced by redundant mutants, which also misleads the testers to overrate test completeness [5]. The recent development of minimal mutations [1] allows researchers to eliminate such redundancy by *subsuming mutants.* Informally, mutant m_1 subsumes m_2 when all the tests that kill m_1 also kill m_2. Since it is undecidable to determine the subsuming mutants, approximated approches have been proposed to identify them, such as [2–4].

*Yawen Wang is the correspondence author.

This paper focuses on the operators that are efficient at producing subsuming mutants by defining *Subsuming Mutation Operators*, or SMOs. Informally, SMOs are the operators that *cover* all subsuming ones with minimal size of mutants. Any tests that kill non-equivalent mutants among SMOs deem to achieve 100% of dominator score. We performed an empirical study on 14 programs to identify the SMOs among Proteum operators. Approches to determine SMOs and their properties are reported in section 2.

2 SUBSUMING MUTATION OPERATORS

The subsuming mutation operators are defined based on mutant subsumption graph (MSG) [3]. MSG is a directed graph, of which node represents a maximal set of *duplicated* mutants, and edge represents subsumption relation. Two mutants are duplicated when they produce the same results against every test. For example, "`x-1<y`" is duplication of "`x<=y`" for integer expressions in "`x<y`".

Given MSG of mutants set M in program. The *subsuming mutants* are those in MSG roots, which subsume all the others in set. We define the *mutants coverage* over subsuming set to measure utility of each operator. Let M_s be mutants of specified operators. The coverage of M_s, denoted as $Cv(M_s)$, is defined as follow.

$$Cv(M_s) = \frac{|\{r_i \in R | \exists m \in M_s \wedge m \in r_i\}|}{|R|} \times 100\% \qquad (1)$$

where R are the MSG roots with non-equivalent mutants.

Clearly, for any tests that are M_s-sufficient, which kill all non-equivalent mutants among M_s, are guaranteed to achieve dominator score that is greater than (or equal with) the coverage of M_s. Let the $DS(T)$ be dominator score of such tests T, there we have:

$$Cv(M_s) \leq DS(T) \leq 1.0 \qquad (2)$$

The mutants coverage defines the *lower-bound* of dominator score for any M_s-sufficient test suite.

According to coverage, we define *Subsuming Mutation Operators* (SMOs) as the best solution of following optimization problem.

$$\text{constraint: } Cv(\bigcup M_{op}) = 100\% \qquad (3)$$

$$\text{minimalize: } \sum |M_{op}|, op \in SMOs \qquad (4)$$

In this definition, SMOs provides two benefits. First, SMOs are the "true" sufficient operators in selective mutation. Any tests that kill all its mutants are guaranteed to achieve 100% of dominator score. Second, SMOs provide the most efficient operators for relevant works to investigate subsuming mutants, which require the minimal efforts in their analysis.

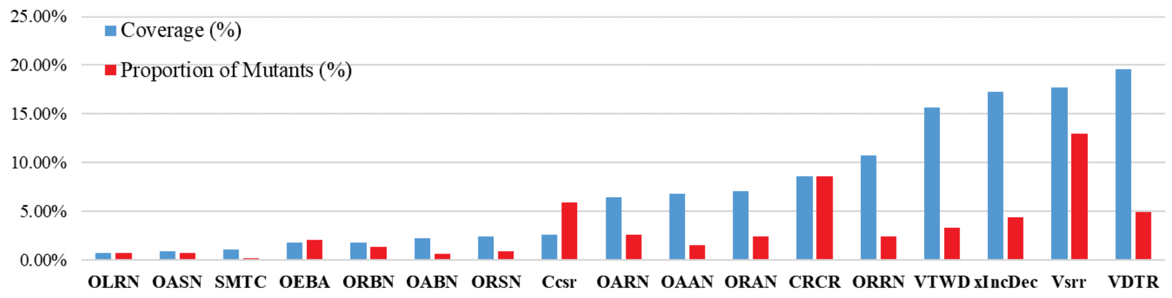

Figure 1: Coverage and Proportion (%) of Mutants Generated by Each of Subsuming Operators Found in Study

Table 1: Subject programs, mutants and operators used

Program	#Mutants	#$Su(M)$ (%)	#Op	#SMOs	#M_{SMO}(%)
bubble	747	14 (1.9%)	34	8 (23%)	367 (49.1%)
calendar	3346	17 (0.5%)	62	9 (14%)	862 (25.7%)
day	2199	46 (2.0%)	58	7 (12%)	358 (16.2%)
insert	1029	7 (0.7%)	35	5 (14%)	192 (18.6%)
mid	1109	26 (2.3%)	43	6 (13%)	277 (24.9%)
minmax	675	18 (2.6%)	31	5 16%	343 (50.8%)
prime	802	8 (1.0%)	54	6 (11%)	127 (15.8%)
profit	2190	15 (0.6%)	43	5 (11%)	266 (12.1%)
triangle	598	33 (5.5%)	27	7 (25%)	329 (55.0%)
replace	23792	161 (0.6%)	65	9 (13%)	6757 (28.4%)
schedule	4146	139 (3.3%)	70	12 (17%)	1205 (29.0%)
schedule2	6552	67 (1.0%)	73	9 (12%)	1592 (24.3%)
tcas	4915	97 (1.9%)	58	8 (13%)	1377 (28.0%)
tot_info	8916	48 (0.3%)	82	6 (7%)	1222 (29.5%)

3 EMPIRICAL STUDY AND RESULTS

We performed an empirical study based on mutants of 14 programs to identify their subsuming operators. Table 1 provides information about the program names, number of (all and subsuming) mutants, the number of (subsuming) operators applied, and proportion of mutants generated by SMOs in study.

The mutation tool `Proteum` is used to generate mutants for each program. The dynamic subsumption approach [3] is used to compute MSG in experiment, which is the basis to determine SMOs.

From table 1, we can find that the number of operators identified as SMOs is trivial, accounting for 7% to 25% of all operators applied in each program. By average, less than 15% of operators used are subsuming. Meanwhile, the table 1 shows that, the number of mutants generated by SMOs identified in each program is less than 30% of all mutations, except three trivial subjects: `bubble`, `minmax` and `triangle`. The low proportion of SMOs' mutants suggests that SMOs can not only guaranteed the high quality of test suite but also reduce the test costs in analysis.

Though the SMOs are different between programs, only 17 of all 82 mutation operators applied in the study are identified as SMOs in some of the programs. These operators are: {OAAN, OABN, OARN, OASN, OEBA, OLRN, ORAN, ORBN, ORRN, ORSN, Ccsr, CRCR, Vsrr, VDTR, VTWD, xIncDec[1], SMTC}.

Figure 1 presents the coverage and proportions of mutants generated by each of the SMOs found in our study. From the figure, we can find that, the coverage of each SMO is not equally significant. The coverage of VDTR, Vsrr, xIncDec, VTWD, ORRN, CRCR, ORAN, OAAN, and OARN are significant. They produce the majority of subsuming mutants in the 14 subject programs. Meanwhile, by eliminating the SMOs with high mutants proportions (red bar in the figure), including Vsrr, CRCR, we identify the {VDTR, xIncDec, VTWD, ORRN, ORAN, OAAN, OARN} as seven SMOs that might be efficient in producing subsuming mutants in the study.

4 CONCLUSION

This paper proposes a coverage-based approach to identify the operators that are sufficient for subsuming mutants in analysis. We define the subsuming mutation operators (SMOs) and provides a coverage-based approach to identify them. We then performed empirical study on 14 trivial programs to identify the SMOs among their mutants. The results show that only a small part of Proteum operators are subsuming, and several useful operators are identified from the subject programs.

ACKNOWLEDGMENTS

The work is supported by National Natural Science Foundation of China under Grant No. U1736110 and No. 61702044.

REFERENCES

[1] P. Ammann, M. E. Delamaro, and J. Offutt. 2014. Establishing Theoretical Minimal Sets of Mutants. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*. 21–30. https://doi.org/10.1109/ICST.2014.13
[2] Gary Kaminski, Paul Ammann, and Jeff Offutt. 2011. Better Predicate Testing. In *Proceedings of the 6th International Workshop on Automation of Software Test (AST '11)*. ACM, New York, NY, USA, 57–63. https://doi.org/10.1145/1982595.1982608
[3] B. Kurtz, P. Ammann, M. E. Delamaro, J. Offutt, and L. Deng. 2014. Mutant Subsumption Graphs. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*. 176–185. https://doi.org/10.1109/ICSTW.2014.20
[4] B. Kurtz, P. Ammann, and J. Offutt. 2015. Static analysis of mutant subsumption. In *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 1–10. https://doi.org/10.1109/ICSTW.2015.7107454
[5] B. Kurtz, P. Ammann, J. Offutt, and M. Kurtz. 2016. Are We There Yet? How Redundant and Equivalent Mutants Affect Determination of Test Completeness. In *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 142–151. https://doi.org/10.1109/ICSTW.2016.41

[1]The xIncDec represents I-DirVarIncDec and I-IndVarIncDec in Proteum.