SCIENCE CHINA
Information Sciences

• **Supplementary File** •

Automated Regression Unit Test Generation for Program Merges

Tao Ji[1], Liqian Chen[1*], Xiaoguang Mao[1*], Xin Yi[1] & Jiahong Jiang[2]

[1]*College of Computer, National University of Defense Technology, Changsha* 410073*, China;*
[2]*Beijing Institute of Tracking and Telecommunication Technology, Beijing* 100095*, China*

Appendix A Revised Algorithm

We revise the algorithm of SafeMerge into the following one:

Algorithm A1 Revised algorithm for verifying conflict freedom

1: $\varphi := RelationalPost(O, A, B, M)$
2: $\chi_1 := \forall i.(out_O[i] \neq out_A[i] \Rightarrow out_A[i] = out_M[i])$
3: $\chi_2 := \forall i.(out_O[i] \neq out_B[i] \Rightarrow out_B[i] = out_M[i])$
4: $\chi_3 := \forall i.(out_O[i]{=}out_A[i]{=}out_B[i] \Rightarrow out_O[i] = out_M[i])$
5: **return** $Valid(\varphi \Rightarrow (\chi_1 \wedge \chi_2 \wedge \chi_3))$

Appendix B Transformation

As shown in Algorithm A1, if there exists one input that makes the logic formula `false` (Line 5), we would say the conflict exists. To find the inputs revealing conflicts, we negate the $\chi_1 \wedge \chi_2 \wedge \chi_3$, then we have $\neg\chi_1 \vee \neg\chi_2 \vee \neg\chi_3$, where

$$\neg\chi_1 \triangleq \exists i.(out_O[i] \neq out_A[i] \wedge out_A[i] \neq out_M[i])$$
$$\neg\chi_2 \triangleq \exists i.(out_O[i] \neq out_B[i] \wedge out_B[i] \neq out_M[i])$$
$$\neg\chi_3 \triangleq \exists i.(out_O[i] = out_A[i] = out_B[i] \wedge out_O[i] \neq out_M[i]).$$

Hence, to reveal merge conflicts, we should find the inputs I that make the formula

$$E(O, I) \wedge E(A, I) \wedge E(B, I) \wedge E(M, I) \Rightarrow \neg\chi_1 \vee \neg\chi_2 \vee \neg\chi_3$$

satisfiable, where $E(V, I)$ represents the execution result on the program V by inputs I. In other words, after execution on four different versions with the same inputs I, if $\neg\chi_1 \vee \neg\chi_2 \vee \neg\chi_3$ is satisfiable, we say the inputs I reveal the merge conflict. To simplify the problem of finding the inputs revealing conflicts, we give the following theorem to reformulate $\neg\chi_1 \vee \neg\chi_2 \vee \neg\chi_3$.

Theorem 1. $\neg\chi_1 \vee \neg\chi_2 \vee \neg\chi_3 \Leftrightarrow \neg\chi_1 \vee \neg\chi_2 \vee \neg\chi_3'$, where $\neg\chi_3' \triangleq \exists i.(out_A[i] \neq out_M[i] \wedge out_M[i] \neq out_B[i])$.

Proof. Let $O \triangleq out_O[i], A \triangleq out_A[i], B \triangleq out_B[i], M \triangleq out_M[i]$.

$\exists i.(M \neq A \wedge A \neq O) \vee \exists i.(M \neq B \wedge B \neq O) \vee \exists i.(O = A = B \wedge M \neq O \wedge M \neq A \wedge M \neq B)$
$\Leftrightarrow \exists i.((M \neq A \wedge A \neq O) \vee (M \neq B \wedge B \neq O) \vee (O = A \wedge O = B \wedge M \neq A \wedge M \neq B))$
// $\exists i.P(i) \vee \exists i.Q(i) \Leftrightarrow \exists i.(P(i) \vee Q(i))$ (the axiom after "//" is used for equivalent transformation)
$\Leftrightarrow \exists i.((M{\neq}A{\wedge}A{\neq}O) \vee (M{\neq}A{\wedge}A{\neq}O{\wedge}M{\neq}B{\wedge}B{=}O) \vee (M{\neq}B{\wedge}B{\neq}O) \vee (O{=}A{\wedge}O{=}B{\wedge}M \neq A{\wedge}M \neq B))$
// $P \Leftrightarrow P \vee (P \wedge Q)$
$\Leftrightarrow \exists i.((M{\neq}A{\wedge}A{\neq}O) \vee (M{\neq}B{\wedge}B{\neq}O) \vee (O{=}B{\wedge}M \neq A \wedge M \neq B))$
// $(P \wedge Q) \vee (P \wedge \neg Q) \Leftrightarrow P$
$\Leftrightarrow \exists i.((M \neq A \wedge A \neq O) \vee (M \neq B \wedge B \neq O) \vee (M \neq B \wedge B \neq O \wedge M \neq A) \vee (O = B \wedge M \neq A \wedge M \neq B))$
// $P \Leftrightarrow P \vee (P \wedge Q)$
$\Leftrightarrow \exists i.((M \neq A \wedge A \neq O) \vee (M \neq B \wedge B \neq O) \vee (M \neq A \wedge M \neq B))$
// $(P \wedge Q) \vee (P \wedge \neg Q) \Leftrightarrow P$
$\Leftrightarrow \exists i.(M \neq A \wedge A \neq O) \vee \exists i.(M \neq B \wedge B \neq O) \vee \exists i.(M \neq A \wedge M \neq B)$
// $\exists i.P(i) \vee \exists i.Q(i) \Leftrightarrow \exists i.(P(i) \vee Q(i))$

Hence, we have $\neg\chi_1 \vee \neg\chi_2 \vee \neg\chi_3 \Leftrightarrow \neg\chi \vee \neg\chi_2 \vee \neg\chi_3'$, based on the above equation.

The transformation guarantees that these three cases cover all cases that violate the contract of semantic conflict freedom. Specifically, if (1) one variable's value returned by the merge is different from the values returned by two variants (i.e., $\neg\chi_3'$), or (2) one variable's value returned by one variant is different from that of the original and that of the merge (i.e., $\neg\chi_1$ or $\neg\chi_2$), the merge is not semantic conflict free.

* Corresponding author (email: lqchen@nudt.edu.cn, xgmao@nudt.edu.cn)

Appendix C Algorithm for Selecting UUTs

Given one version v_t together with a set of variants $\mathbb{V}$, after identifying added and changed entities from version pairs $\{\Delta(v_1, v_t)$, $\Delta(v_2, v_t)$, ..., $\Delta(v_n, v_t)\}$, we propose an algorithm for selecting UUTs from v_m that behave differently in all variants, as shown in Algorithm C1. In some cases, the number of candidate methods may be large. For example, if changes on the assignments of the fields are made to the constructor, the other methods which use the changed fields should be analyzed since we have to instantiate these methods before calling them. Hence, one parameter n is used to limit the size of all UUTs and one other parameter d is used to limit the dependency depth explored. The method-level dependency analysis is conducted on the target version to extract the dependency relationships between entities of the whole program (Line 1). We then extract added and changed fields as well as methods (Lines 2-6) by comparing each variant with the target version. During the exploration, we get more directly impacted entities by the already collected entities (Lines 9-17). Then, we compute the intersection between the sets of impacted entities from each version pair as the UUTs (Line 18). If we have a set of UUTs more than the number specified, we return the first n UUTs to generate tests (Line 21). If not, we find more impacted entities by exploring more deeply (Line 8). If we fail to find the common impacted entities during the search (Lines 8-22), we generate test cases for those added and changed methods to ensure the program quality (Lines 23-25).

Algorithm C1 UUTs Selection

Require: the target version v_t with a set of variants $\mathbb{V}$
Ensure: the set of *uuts*

```
1: entity_relations = extract_dependencies(v_t)
2: ce, ie = {} // ce is a list of sets of changed entities, ie is a list of sets of impacted entities
3: for v_i ∈ 𝕍 do
4:    ce[i] = diff(v_i, v_t)
5:    ie[i] = ie[i] ∪ ce[i]
6: end for
7: uuts = ∅
8: for i ∈ [1, d] do
9:    for j ∈ [0, ie.size()) do
10:      de = ∅
11:      for entity ∈ ie[index] do
12:         if entity.depth == i − 1 then
13:            de = de ∪ get_impacted(entity)
14:         end if
15:      end for
16:      ie[j] = ie[j] ∪ de
17:   end for
18:   uuts = intersect_sets(ie)
19:   if uuts.size() > n then
20:      return uuts[0 : n − 1]
21:   end if
22: end for
23: if uuts == ∅ then
24:   uuts = get_all_methods(ce)
25: end if
26: return uuts
```

Appendix D MCon4J

In our experiments, we use eight types of mutation operators implemented in Major to generate mutants. These mutation operators include AOR (Arithmetic Operations Replacement), SOR (Shift Operator Replacement), COR (Conditional Operator Replacement), LOR (Logical Operator Replacement), ROR (Relational Operator Replacement), ORU (Operator Replacement Unary), LVR (Literal Value Replacement) and STD (Statement Deletion).

As shown in Table D1, we list the numbers of merges in each project from Defects4J. The column "DC" means that the mutation branch and the fix branch modify the different classes. The column "SC" means that two branches modify the different methods of the same class. The column "SM" means that two branches modify the same method.

Appendix E Experimental Results

Considering the radom numbers used, we run the detection for three times. As shown in Table E1, we present the detection results for 3-way merges. And as shown in Table E2, we show the detection results for octopus merges.

Table D1 The numbers of 3-way merges and octopus merges.

Project	Bugs	Merges			
		DC	SC	SM	Total
JFreeChart	26	18	6	1	25
Closure	176	168	3	0	171
Commons Lang	65	16	34	11	61
Commons Math	106	61	32	6	99
Mockito	38	6	0	0	6
Joda-Time	27	23	4	0	27
Total	438	292	79	18	389

Table E1 Conflict three-way merges that detected by executing three times with different coverage criteria "Diff-line" and "Multi-criteria". The column "Mutant" shows the mutation operator and its location. "−" means the conflict merge fails to be detected. "⊙" means the conflict merge is detected when generating test cases for the P_f variant (i.e., the bug-fix version). "⊖" means the conflict merge is detected when generating test cases for non-P_f variants. "⊕" means the conflict merge is detected when generating test cases for both P_f and non-P_f variants.

Id	Mutant	Diff-line			Multi-criteria			Id	Mutant	Diff-line			Multi-criteria		
		#1	#2	#3	#1	#2	#3			#1	#2	#3	#1	#2	#3
Chart_4	DC-LVR	⊙	⊙	⊙	⊙	⊙	⊙	Math_16	SC-ROR	−	−	⊙	−	⊙	−
Chart_5	DC-AOR	−	⊙	−	−	−	−	Math_27	DC-COR	⊙	−	⊙	⊙	⊙	−
Chart_8	SC-STD	⊖	⊕	⊖	⊖	⊖	⊖	Math_29	DC-LVR	⊙	⊙	⊙	⊙	⊙	⊙
Chart_11	SC-STD	−	⊕	⊖	⊖	⊙	⊕	Math_32	DC-STD	−	⊙	−	−	−	⊙
Chart_14	DC-STD	−	−	−	−	−	⊙	Math_37	DC-AOR	⊕	−	−	−	−	−
Chart_16	SC-LVR	⊕	⊕	⊕	⊕	⊕	⊕	Math_46	SC-LVR	−	⊙	−	−	⊕	⊕
Chart_21	DC-STD	⊕	⊕	⊕	⊕	⊕	⊕	Math_47	SC-AOR	⊕	⊖	⊖	⊕	⊕	⊕
Chart_24	SC-STD	⊕	⊙	⊕	⊕	⊕	⊕	Math_49	DC-STD	⊙	⊙	⊙	⊙	⊙	⊙
Closure_19	DC-LVR	⊙	⊙	⊙	⊙	⊙	⊙	Math_56	SC-STD	⊙	⊙	⊙	⊙	⊙	⊙
Closure_173	DC-STD	−	⊖	⊖	−	−	−	Math_60	DC-AOR	⊙	⊙	⊙	−	−	⊕
Lang_19	DC-STD	−	−	−	⊙	−	−	Math_63	SC-LVR	⊕	⊙	−	⊕	⊕	⊕
Lang_39	SC-STD	⊙	⊕	⊕	⊕	−	⊕	Math_70	SC-LVR	−	⊙	⊙	⊙	⊙	⊙
Lang_41	SC-STD	⊙	⊙	⊙	⊙	⊙	⊙	Math_71	DC-STD	⊙	⊙	⊙	⊙	⊙	−
Lang_45	DC-ROR	⊕	⊕	⊕	⊕	⊕	⊕	Math_73	SC-ROR	⊖	⊖	⊖	⊖	⊖	⊖
Lang_47	SC-STD	⊙	⊙	⊙	⊙	⊙	⊙	Math_80	SC-AOR	⊙	⊙	−	−	⊙	⊙
Lang_60	SC-ROR	⊕	⊕	⊕	⊕	⊕	⊕	Math_81	SC-STD	⊕	⊙	⊙	⊙	−	⊙
Lang_65	SM-COR	⊕	⊕	⊕	⊕	⊕	⊕	Math_83	DC-LVR	⊙	⊙	⊙	⊙	⊙	⊙
Math_1	DC-AOR	⊙	−	⊙	−	−	⊙	Math_92	SM-LVR	−	⊕	⊕	⊕	⊖	⊕
Math_2	DC-AOR	−	−	⊙	−	−	−	Math_93	SC-ROR	⊙	⊙	−	⊙	⊙	⊙
Math_4	DC-AOR	⊙	⊕	⊕	⊕	⊕	⊕	Math_95	SC-STD	⊕	⊕	⊕	⊕	⊕	⊕
Math_9	SC-STD	−	−	−	⊙	−	−	Math_97	DC-STD	⊙	⊙	−	⊙	−	⊙
Math_10	DC-LVR	⊙	⊙	⊙	−	−	−	Time_9	DC-AOR	⊖	−	⊖	⊖	⊖	⊖
Math_11	DC-LVR	−	⊙	−	−	⊙	⊙								

Table E2 Conflict octopus merges detected by executing three times with the multiple coverage criteria. The column "Mutant" shows the mutation operators used in two constructed branches respectively with the mutant location. The same symbols are explained in the caption of Table E1.

Id	Mutant	#1	#2	#3	Id	Mutant	#1	#2	#3	Id	Mutant	#1	#2	#3
Chart_2	DC-STD-ORU	⊙	⊙	⊙	Closure_97	DC-COR-COR	⊖	⊖	⊖	Math_31	DC-LVR-COR	⊙	–	–
Chart_4	DC-LVR-AOR	⊙	⊙	⊙	Closure_108	DC-ROR-STD	⊖	⊖	⊖	Math_32	DC-STD-STD	⊙	⊙	–
Chart_7	DC-STD-AOR	⊙	⊙	–	Closure_111	DC-ROR-COR	–	–	⊖	Math_37	DC-AOR-AOR	⊙	⊕	⊙
Chart_11	SC-STD-ROR	⊙	⊙	⊙	Closure_138	DC-COR-ROR	⊙	⊙	⊙	Math_39	DC-ROR-STD	–	–	⊖
Chart_14	DC-STD-LVR	–	⊙	–	Closure_140	DC-ROR-LVR	⊖	⊖	⊖	Math_46	SC-LVR-STD	⊙	⊙	⊙
Chart_20	DC-STD-ROR	–	⊕	⊖	Closure_148	DC-STD-STD	⊙	⊙	⊙	Math_47	SC-AOR-COR	⊕	⊕	⊙
Chart_21	DC-STD-LVR	⊙	–	–	Closure_152	DC-LVR-LVR	⊕	⊕	⊕	Math_49	DC-STD-STD	⊙	⊙	⊙
Chart_24	SC-STD-LVR	⊙	⊙	⊙	Closure_165	DC-STD-ROR	⊙	⊙	⊙	Math_52	DC-LVR-LVR	⊕	⊕	⊖
Chart_25	DC-STD-COR	⊖	⊕	⊕	Lang_15	DC-STD-COR	⊕	⊕	⊖	Math_56	SC-STD-STD	⊙	⊙	⊙
Closure_1	DC-ROR-ROR	⊖	⊙	–	Lang_17	DC-LVR-LVR	⊖	⊖	⊖	Math_60	DC-AOR-AOR	⊙	⊙	⊙
Closure_2	DC-COR-COR	⊙	–	⊙	Lang_29	DC-LVR-LVR	⊕	⊕	⊕	Math_63	SC-LVR-LVR	⊙	⊙	⊙
Closure_3	DC-STD-ROR	⊕	⊙	⊙	Lang_34	DC-ROR-STD	–	⊖	–	Math_70	SC-LVR-AOR	⊕	⊙	⊙
Closure_15	DC-ROR-STD	⊖	⊖	⊖	Lang_36	SC-ROR-ROR	⊙	⊙	⊙	Math_71	DC-STD-STD	⊙	–	⊙
Closure_19	DC-LVR-LVR	⊕	⊕	⊕	Lang_37	SC-LVR-ROR	–	⊖	⊖	Math_73	SC-ROR-AOR	–	⊖	–
Closure_24	DC-ROR-COR	–	⊖	⊖	Lang_38	SC-STD-COR	⊙	⊕	⊙	Math_80	SC-AOR-LVR	–	⊙	⊕
Closure_26	DC-ROR-COR	–	⊖	–	Lang_39	SC-STD-COR	⊙	⊙	⊙	Math_81	SC-STD-LVR	⊙	–	⊙
Closure_27	DC-ROR-ROR	⊖	⊖	⊖	Lang_41	SC-STD-COR	⊙	⊙	⊙	Math_83	DC-LVR-AOR	⊙	⊙	⊙
Closure_32	DC-ROR-COR	⊖	⊖	⊖	Lang_45	DC-ROR-COR	⊙	⊙	⊙	Math_85	DC-AOR-STD	⊖	⊖	⊖
Closure_47	DC-SOR-AOR	–	–	⊙	Lang_47	SC-STD-LVR	⊕	⊕	⊕	Math_87	DC-AOR-LVR	–	⊙	–
Closure_55	DC-COR-COR	–	⊖	⊖	Lang_60	SC-ROR-COR	⊙	⊙	⊙	Math_88	DC-STD-LVR	⊖	⊖	⊖
Closure_67	DC-STD-COR	⊙	⊙	⊙	Lang_65	SM-COR-ROR	⊙	⊙	⊙	Math_92	SM-LVR-LVR	⊙	⊙	⊙
Closure_72	DC-COR-LVR	⊖	⊖	⊖	Math_1	DC-AOR-LVR	–	⊙	–	Math_93	SC-ROR-LOR	⊙	⊙	⊙
Closure_78	DC-ROR-COR	–	⊖	⊖	Math_4	DC-AOR-STD	⊙	⊙	⊕	Math_95	SC-STD-STD	⊙	⊙	⊙
Closure_80	DC-STD-ROR	⊙	⊙	⊙	Math_9	SC-STD-ROR	⊖	⊕	⊖	Math_96	SC-COR-LVR	⊕	⊕	⊕
Closure_81	DC-AOR-STD	⊖	⊖	⊖	Math_11	DC-LVR-AOR	⊙	⊙	–	Math_102	DC-AOR-AOR	⊙	⊙	⊙
Closure_84	DC-LVR-LVR	–	⊖	⊖	Math_16	SC-ROR-LVR	⊙	–	–	Math_103	DC-LVR-STD	⊖	–	–
Closure_89	DC-ROR-COR	⊖	–	⊖	Math_25	SC-STD-AOR	–	–	⊖	Math_104	SC-AOR-AOR	⊙	–	–
Closure_95	DC-LVR-LVR	⊖	⊖	⊖	Math_27	DC-COR-ROR	⊙	⊙	⊙	Time_9	DC-AOR-ROR	⊕	⊙	⊙
Closure_96	DC-STD-COR	⊖	–	–	Math_29	DC-LVR-STD	⊙	⊙	⊙	Time_20	DC-STD-STD	–	–	⊖