Method-level Aggregation

```
class Code{
  static int m1(int x) {
    int y = Math.abs(x);
    if ( y % 2 == 1)
      int s = 1;//buggy
    else
      int s = 1;
    return s;
  }


  static int m2(int x) {
    int s = x + 1;
    return s;
  }
}
```

```
public void t1() {
  int a = Code.m1(-2);
  int b = Code.m2(a);
  assertEquals(2, b);
}
public void t2() {
  int a = Code.m1(2);
  assertEquals(1, a);
}
public void t3() {
  int a = Code.m1(3);
  int b = Code.m2(a);
  assertEquals(0, c);
}
public void t4() {
  int a = Code.m2(5);
  assertEquals(6, a);
}
```

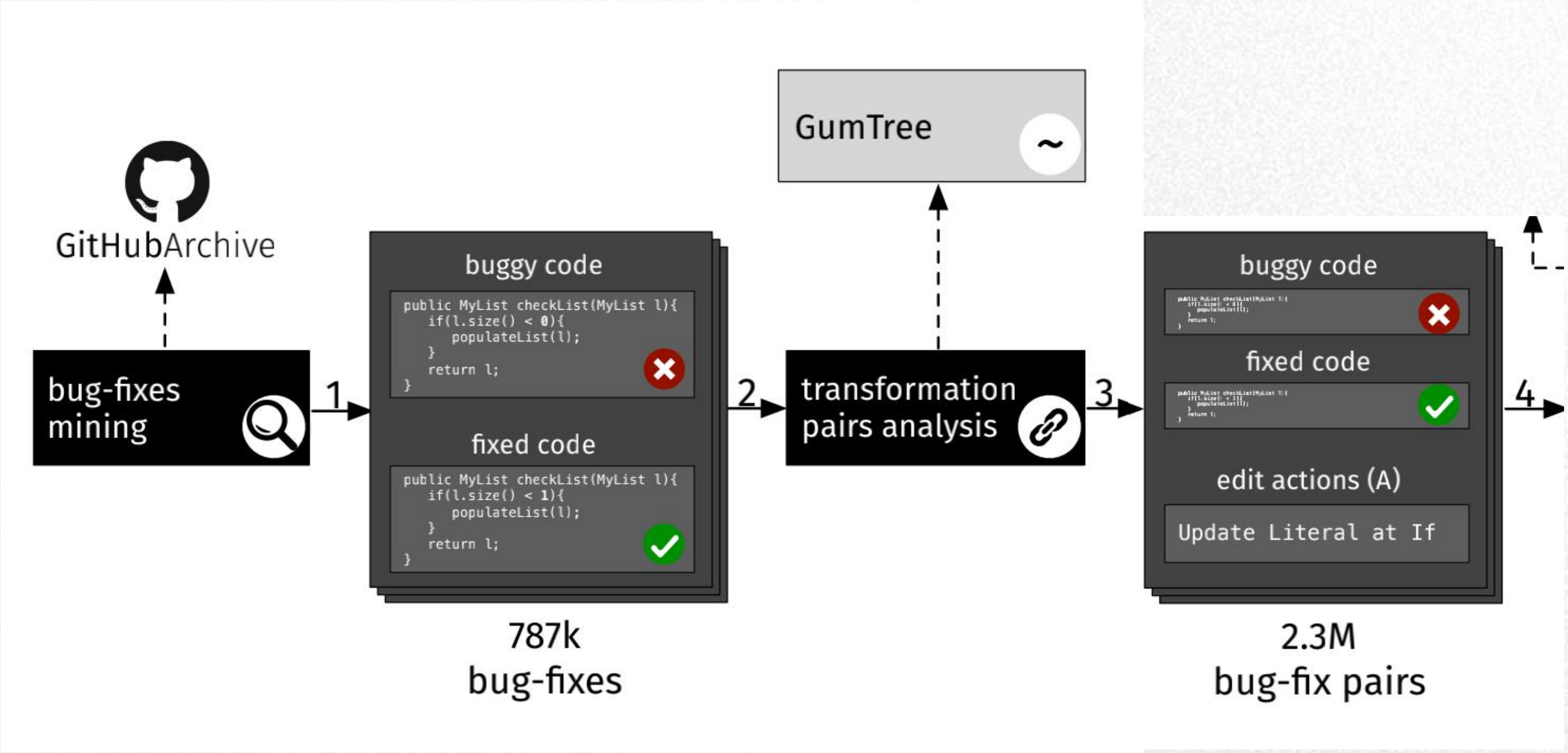
Fig. 2: Example Code Snippet for Method-level Aggregation

TABLE 3: The Process of $PRFL_{MA}$

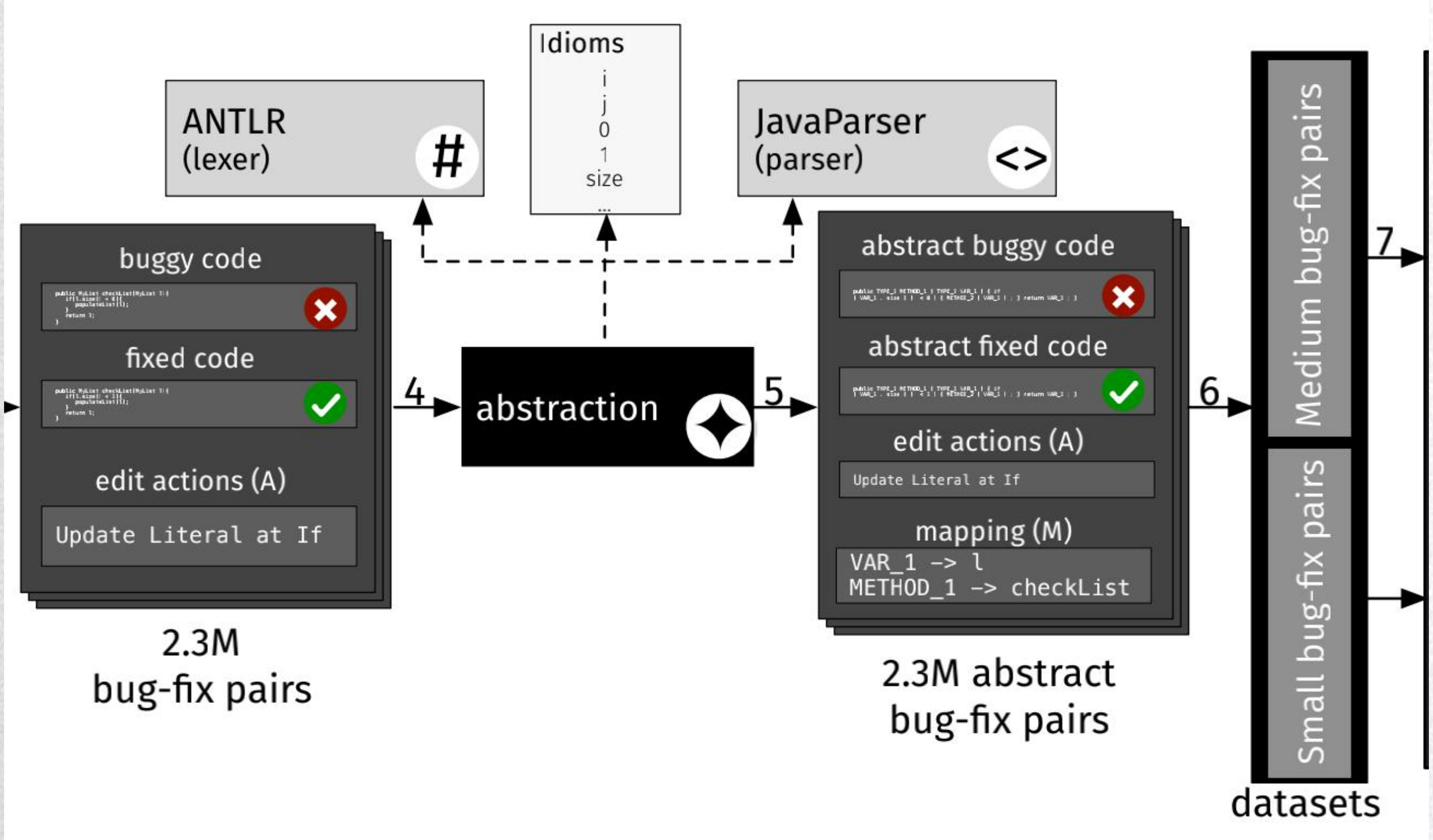
C	Subject	t1	t2	t3	t4	Och.2	Agg.
m1	$static\ int\ m1(int\ x)\{$	1	1	1	0	0.08	1*
s1	$y = Math.abs(x);$	1	1	1	0	0.08	0.08
s2	$if(y \% 2 == 1)$	1	0	0	0	1*	1*
s3	$int\ s = 1; //buggy$	1	0	0	0	1*	1*
s4	$else$	0	1	1	0	0	0
s5	$int\ s = y;$	0	1	1	0	0	0
s6	$return\ s;\}$	1	1	1	0	0.08	0.08
m2	$static\ int\ m2(int\ x)\{$	1	0	1	1	0.08	0.08
s7	$int\ s = x + 1;$	1	0	1	1	0.08	0.08
s8	$return\ s;\}$	1	0	1	1	0.08	0.08
Test case outcome		f	p	p	p	m1/2	m1
Number of candidates		4					

《An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation》	2019	TOSEM
《Precise Learn-to-Rank Fault Localization Using Dynamic and Static Features of Target Programs》	2019	TOSEM
《Improving bug reporting, duplicate detection, and localization》	2017	ICSE
《HMER:A Hybrid Mutation Execution Reduction approach for Mutation-based Fault Localization》	2020	JSS
《Code Complexity and Version History for Enhancing Hybrid Bug Localization》	2021	IEEE Access

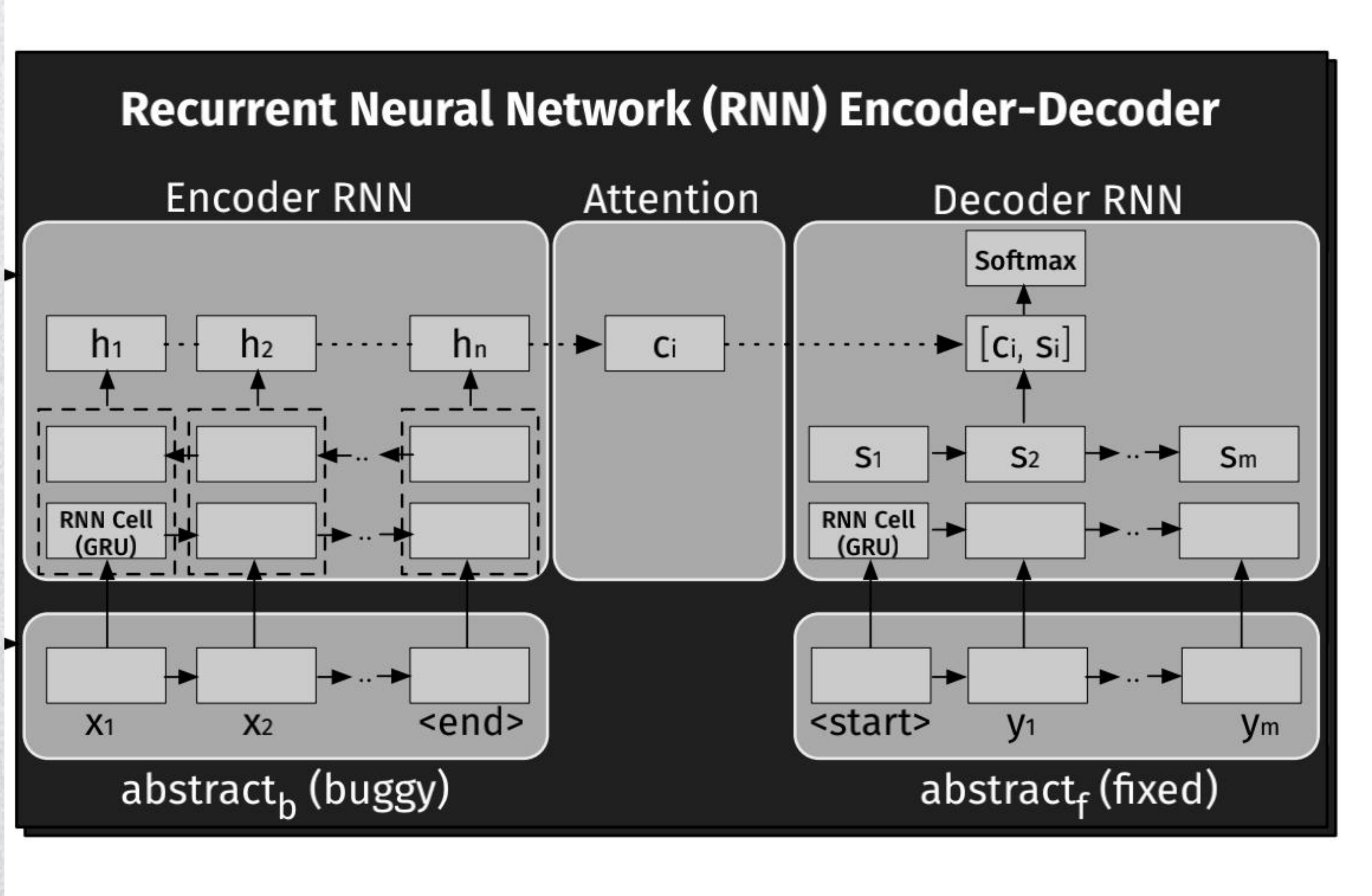
An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation 2019 TOSEM



An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation 2019 TOSEM



An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation 2019 TOSEM



Precise Learn-to-Rank Fault Localization Using Dynamic and Static Features of Target Programs

2019 TOSEM

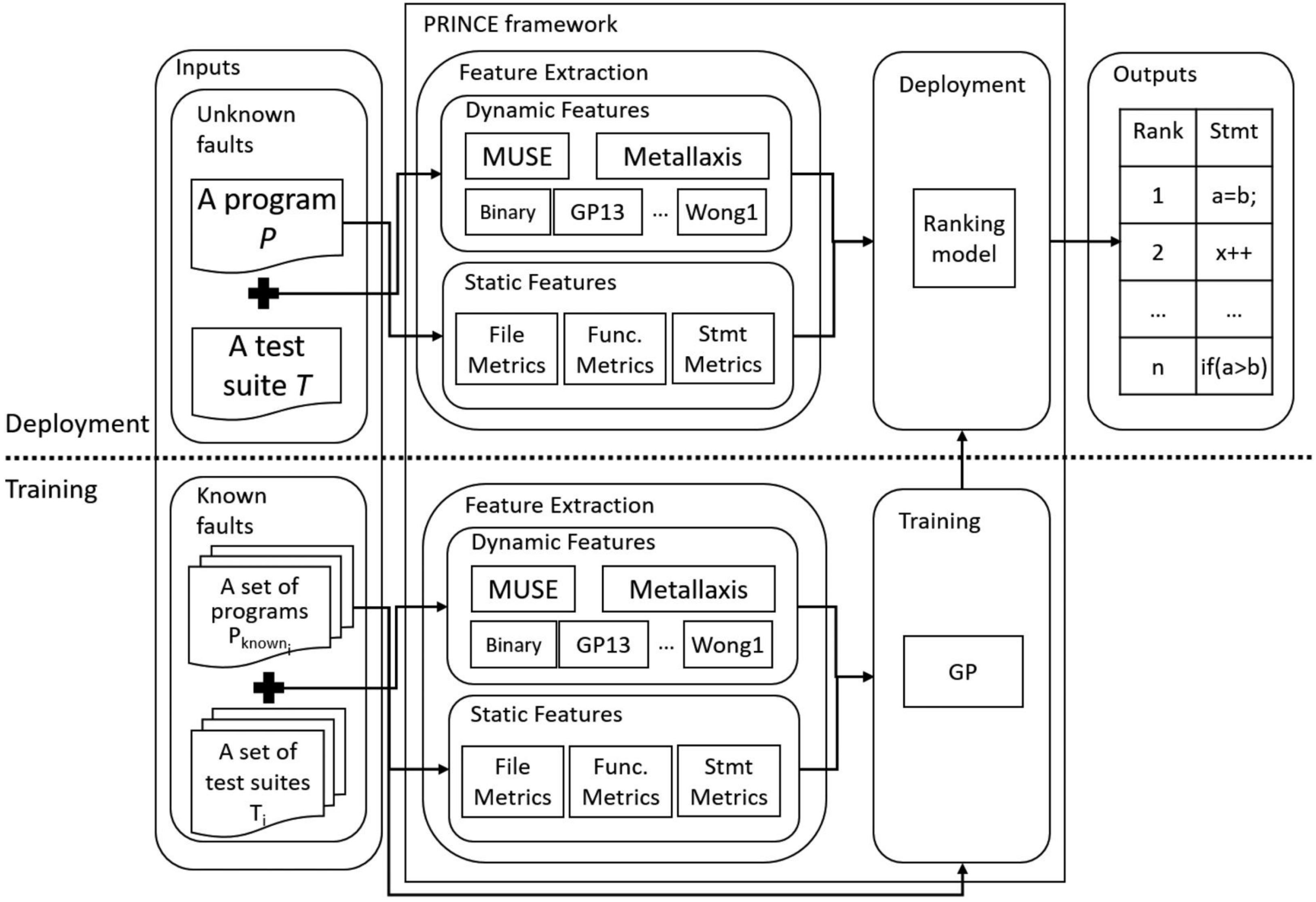


Fig. 1. Overall process of PRINCE.

Precise Learn-to-Rank Fault Localization Using Dynamic and Static Features of Target Programs

2019 TOSEM

Type	Feature group	List of features
Dynamic	MBFL	• Metllaxis: $\max_{m\in mut_{killed}(s)}(kill(m))$, $\max_{m\in mut_{killed}(s)}(\frac{1}{\sqrt{kill(m)}})$, $\max_{m\in mut_{killed}(s)}(\frac{1}{\sqrt{kill(m)+notkill(m)}})$, $\max_{m\in mut_{killed}(s)}(\frac{kill(m)}{\sqrt{(kill(m))(kill(m)+notkill(m))}})$ where - $mut_{killed}(s)$ is a set of killed mutants generated at statement s - $kill(m)$ represents the number of test cases that kill m - $notkill(m)$ represents the number of test cases that do not kill m • MUSE: $\frac{1}{(\|mut(s)\|+1)}$, $\Sigma_{m\in mut(s)}\|p_P(s)\cap f_m\|$, $\Sigma_{m\in mut(s)}\|f_P(s)\cap p_m\|$ $\frac{1}{(\|mut(s)\|+1)(f2p+1)}\times\sum_{m\in mut(s)}(\|f_P(s)\cap p_m\|)$, $\frac{1}{(\|mut(s)\|+1)(p2f+1)}\times\sum_{m\in mut(s)}(\|p_P(s)\cap f_m\|)$, $(\frac{1}{(\|mut(s)\|+1)(f2p+1)}\times\sum_{m\in mut(s)}(\|f_P(s)\cap p_m\|)-$ $\frac{1}{(\|mut(s)\|+1)(p2f+1)}\times\sum_{m\in mut(s)}(\|p_P(s)\cap f_m\|))$ where - $mut(s)$ is # of mutants generated on s - $f_P(s)$ (or $p_P(s)$) is the set of tests that cover s and fail (or pass) on a target program P - f_m (or p_m) is the set of tests that fail (or pass) on a mutant m. - $f2p$ (or $p2f$) is the number of test result changes from fail to pass(or pass to fail) for all mutants of P
	SBFL	• Basic terms: $e_p(s)$, $e_f(s)$, $n_p(s)$, $n_f(s)$ - $e_p(s)$ (or $e_f(s)$) is the the number of passing (or failing) tests that execute s - $n_p(s)$ (or $n_f(s)$) is the the number of passing (or failing) tests that do not execute s • Binary: 0 *if* $0 < n_f(s)$, 1 *if* $0 = n_f(s)$ • GP13: $e_f(s)$, $\frac{1}{2e_p(s)+e_f(s)}$, $\frac{e_f(s)}{2e_p(s)+e_f(s)}$, $e_f(s)+\frac{e_f(s)}{2e_p(s)+e_f(s)}$ • Jaccard: $e_f(s)$, $\frac{1}{e_f(s)+n_f(s)+e_p(s)}$, $\frac{e_f(s)}{e_f(s)+n_f(s)+e_p(s)}$ • Naish1: $n_p(s)$, -1 *if* $0 < n_f(s)$, $n_p(s)$ *if* $0 = n_f(s)$ • Naish2: $e_f(s)$, $e_p(s)$, $\frac{1}{e_p(s)+n_p(s)+1}$, $\frac{e_p(s)}{e_p(s)+n_p(s)+1}$, $e_f(s)-\frac{e_p(s)}{e_p(s)+n_p(s)+1}$ • Ochiai: $e_f(s)$, $\frac{1}{\sqrt{e_f(s)+n_f(s)}}$, $\frac{1}{\sqrt{e_f(s)+e_p(s)}}$, $\frac{e_f(s)}{\sqrt{(e_f(s)+n_f(s))(e_f(s)+e_p(s))}}$ • Russell and Rao: $e_f(s)$, $\frac{1}{e_p(s)+n_p(s)+e_f(s)+n_f(s)}$, $\frac{e_f(s)}{e_p(s)+n_p(s)+e_f(s)+n_f(s)}$ • Wong1: $e_f(s)$

Precise Learn-to-Rank Fault Localization Using Dynamic and Static Features of Target Programs

2019 TOSEM

Static	File	• Fan-in and Fan-out of a file dependency graph • # of defined file scope functions, # of defined file scope variables • # of defined global functions, # of defined global variables • # of defined functions, # of defined variables • Compile time(s), compiler memory usage(KB), # of compiler warnings, LOC
	Function	• Fan-in and Fan-out of a static function call graph • LOC, Cyclomatic complexity, # of parameters • # of global variables a function reads, # of global variables a function writes • # of local variables a function reads, # of local variables a function writes
	Statement	• Length of statements (bytes) • # of operators that a statement uses • # of variables that a statement uses

Improving Bug Reporting, Duplicate Detection,and Localization
2017 ICSE

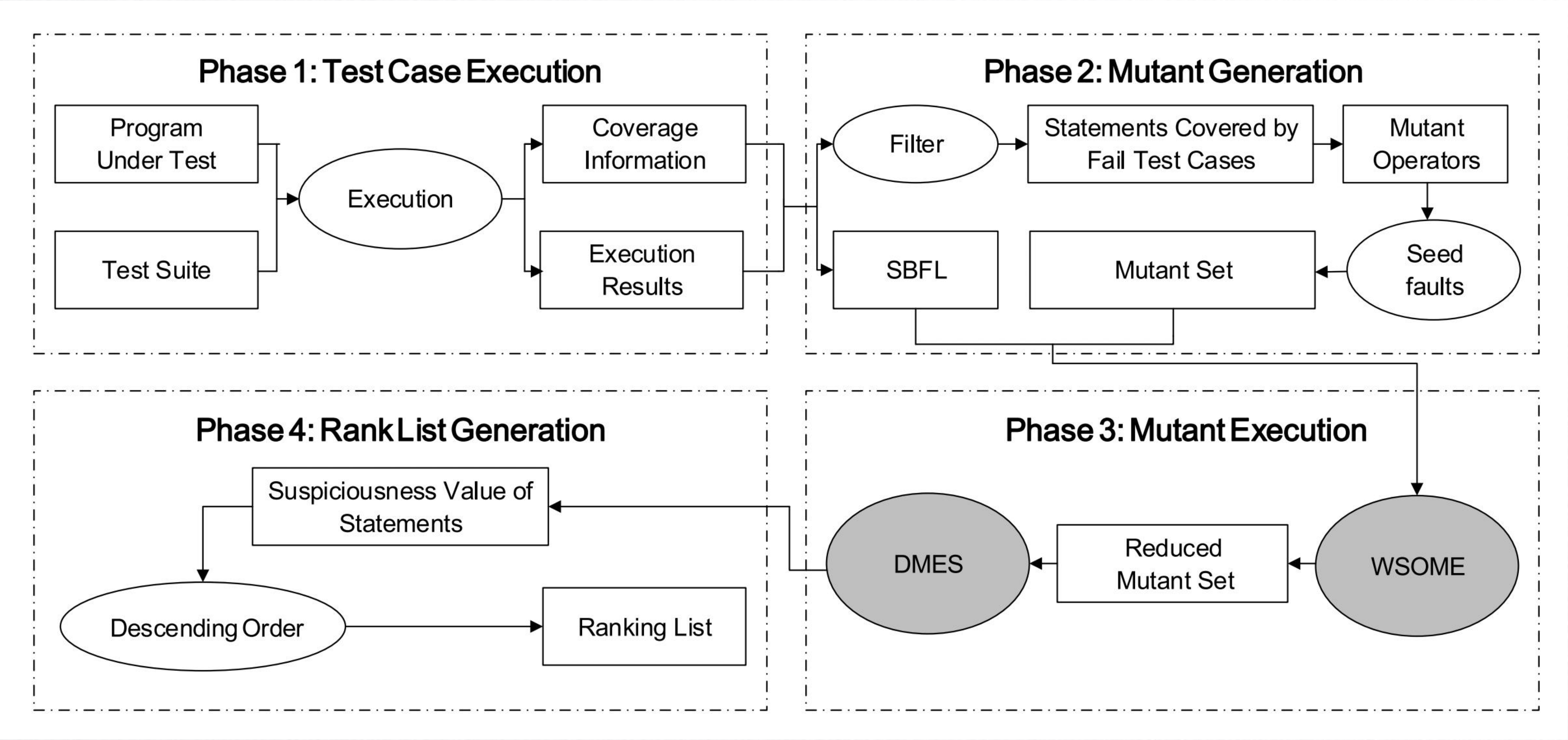
A. Determining the Discourse Used in Bug Descriptions

B. Detecting Missing Information in Bug Reports

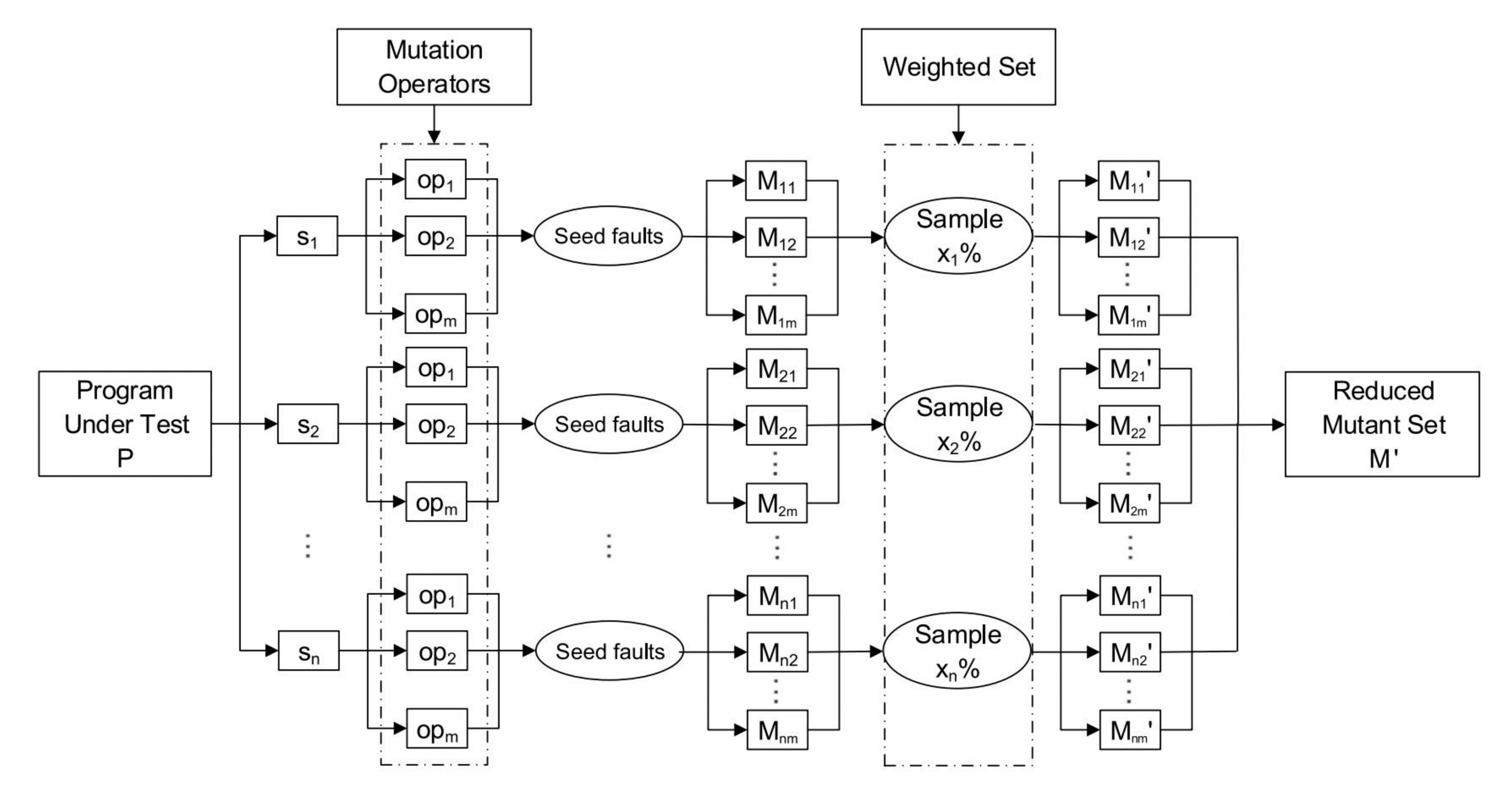
C. Recommending Common Bug Discourse Elements

D. Improving Text Retrieval-based Bug Localization and Duplicate Detection

HMER:A Hybrid Mutation Execution Reduction Approach for Mutation-based Fault Localization 2020 JSS



HMER:A Hybrid Mutation Execution Reduction Approach for Mutation-based Fault Localization 2020 JSS



HMER:A Hybrid Mutation Execution Reduction Approach for Mutation-
...alization 2020 JSS

$$threshold = \begin{cases} a_{kp} + a_{np}, & SusMax = 0, \\ \left\lceil \dfrac{a_{kf}^2}{SusMax^2 * (a_{kp} + a_{np})} - a_{kf} \right\rceil, & SusMax \neq 0. \end{cases}$$

$$\overline{Sus(M)} = \frac{a_{kf}}{\sqrt{(a_{kf} + a_{nf}) * a_{kf}}}$$

C	0	1	1	1	1	0							
R	P	F	P	P	F	P	Suspiciousness of s						
Results of executing T on M(s)							a_{np}	a_{nf}	a_{kp}	a_{kf}	$\overline{Sus}$	*threshold*	*Sus*
m_3	-	*k*	*k*	*n*	*k*	-	3	0	1	2	**1**	2	0.82
m_5	-	*k*	*k*	/	*k*	-	3	0	1	2	**1**	1	0.82
m_1	-	*n*	/	/	*k*	-		1		1	0.71		
m_2	-	*n*	/	/	*k*	-		1		1	0.71		
m_4	-	*n*	/	/	*k*	-		1		1	0.71		
m_6	-	*k*	/	/	*n*	-		1		1	0.71		
m_7	-	*n*	/	/	*n*	-		2		0	0.00		

Fig. 6. A working example of MBFL with DMFS strategy.

Code Complexity and Version History for Enhancing Hybrid Bug Localization

2021 IEEE Access