Repairing Programs With Semantic Code Search

JAN 26TH, 2016

论文下载：https://people.cs.umass.edu/~brun/pubs/pubs/Ke15ase.pdf

- 解决的问题：自动化修补源码当中的问题代码
- 背景：人工寻找和修补代码成本高；有大量的开源项目代码可以重用
- 方法：通过语义相似性从已有代码库中找到正确的代码替代原有项目中的问题代码

SearchRepair——具体方法

- 样例代码

```
1  int main() {
2   int a, b, c, median = 0;
3   printf("Please enter 3 numbers separated by spaces >");
4   scanf("%d%d%d", &a, &b, &c);
5   if ((a<=b && a>=c) || (a>=b && a<=c))
6      median = a;
7   else if ((b<=a && b>=c) || (b>=a && b<=c))
8      median = b;
9   else if ((c<=b && a>=c) || (c>=b && a<=c))
10     median = c;
11  printf("%d is the median", median);
12  return 0;
13 }
```

Fig. 1: A student-written, buggy program to print the median of three integers. Note that the comparison between variables `a` and `c` on line 9 are flipped, such that `c` will never be identified as the median.

- 测试用例

	input	expected output	program output	test result
t_1	9 9 9	"9 is the median"	"9 is the median"	pass
t_2	0 2 3	"2 is the median"	"2 is the median"	pass
t_3	0 1 0	"0 is the median"	"0 is the median"	pass
t_4	0 2 1	"1 is the median"	"0 is the median"	fail
t_5	8 2 6	"6 is the median"	"0 is the median"	fail

Fig. 2: Five test cases for the program from Figure 1.

- 待选代码

(a) fully correct code fragment:

```
1  if((x <= y && x >= z) || (x >= y && x <=z))
2      m = x;
3  else if((y <=  x && y >= z) || (y >= x && y <= z))
4      m = y;
5  else
6      m = z;
```

(b) partially correct code fragment:

```
1  if ((a <= b && a >= c) || (a >= b && a <= c))
2    median = a;
3  else if ((b <= a && b >= c) || (b >= a && b <= c))
4    median = b;
5  else if ((c <= b && a <= c) || (c >= b && a <= c))
6    median = c;
```

Fig. 3: Two candidate code fragments to be used to replace lines 5–10 in Figure 1. Code fragment (a) repairs the bug, passing all five tests; meanwhile, (b) only partially repairs the bug, as tests t_1, t_2, t_3, and t_5 pass, but test t_4 still fails.

- 将备用代码输入、输出符号表达式化，同时找到问题代码
- 用一个正确的测试用例结合备用代码的符号表达式，计算可满足性
- 用更多的测试用例测试正确性

Indexing for Semantic Code Search

- 将备选代码的输入、输出的语义表达式计算出来
- 所以例子代码的符号表达式为

$$\exists x, y, z, m : \texttt{int} \qquad (\gamma)$$

$$(m = x) \wedge ((z \leq x \leq y) \vee (y \leq x \leq z)) \qquad (\phi_1)$$

$$(m = y) \wedge (\neg((z \leq x \leq y) \vee (y \leq x \leq z)) \wedge ((z \leq y \leq x) \vee (x \leq y \leq z))) \qquad (\phi_2)$$

$$(m = z) \wedge (\neg((z \leq x \leq y) \vee (y \leq x \leq z)) \wedge \neg((z \leq y \leq x) \vee (x \leq y \leq z))) \qquad (\phi_3)$$

$$\bigvee_{1}^{n} \gamma \wedge \phi_n$$

Fault Localization

- 使用**Tarantula fault localization technique**
- 通过计算代码语句的“可疑程度”定位可能的错误代码，公式如下

$$suspiciousness(s) = \sqrt{\left(\frac{failed(s)}{total\_failed}\right)\left(\frac{failed(s)}{failed(s) + passed(s)}\right)}$$

- *s*是语句，*total_failed*是总共错误的测试数量，*failed*和*passed*分别代表执行*s*错误和正确的测试数量
- 可疑程度最高语句所在代码块就是即将被修补的代码块

Semantic Search with an Input-Output Profile

- 用一个正确的测试用例结合备选代码的符号表达式，计算可满足性
- 因为正确执行的要求更为严格

- 比如例子中，一个正确测试输入

$$(\exists i, j, k, med : \texttt{int}) \wedge (i = 2) \wedge (j = 3) \wedge (k = 4) \wedge (med = 3) \qquad (Q_{io})$$

- 穷举变量对应关系

$$(med = m) \wedge (((i=x) \wedge (j=y) \wedge (k=z)) \vee ((i=x) \wedge (j=z) \wedge (k=y)) \vee ((i=y) \wedge (j=x) \wedge (k=z)) \vee ((i=y) \wedge (j=z) \wedge (k=x)) \vee ((i=z) \wedge (j=x) \wedge (k=y)) \vee ((i=z) \wedge (j=y) \wedge (k=x))) \qquad (\mathcal{M}_{io})$$

- 验证表达式

$$\gamma \wedge \phi_n \wedge Q_{io} \wedge \mathcal{M}_{io} \qquad (search)$$

Evaluating Patches

- 用其他所有的测试用例测试以上得到的代码正确性

Evaluation

program	defects	instructor tests	KLEE tests	description
checksum	29	6	10	check sum of a string
digits	91	6	10	digits of a number
grade	226	9	9	grade from score
median	168	7	6	median of three numbers
smallest	155	8	8	smallest of four numbers
syllables	109	6	10	count vowels of a string
total	778	42	53	

Fig. 6: The IntroClass dataset, including the number of buggy versions of each assignment and the associated test suite sizes.

program	SearchRepair	AE	GenProg	TrpAutoRepair	total
checksum	0	0	8	0	29
digits	0	17	30	19	91
grade	5	2	2	2	226
median	68	58	108	93	168
smallest	73	71	120	119	155
syllables	4	11	19	14	109
total repaired	150	159	287	247	778

Fig. 7: Number of defects repaired by each technique. The total *column* specifies the total number of defects, and the total *row* specifies the total number of repaired defects.

* 778个defects来自6个玩具程序 * 测试结果比**GenProg**差不少 * 备用库的质量是一个可能的原因，比如这里checksum的实验是0，但用了Linux内核代码作为备用库以后就能出来18

个，其中17个是特有

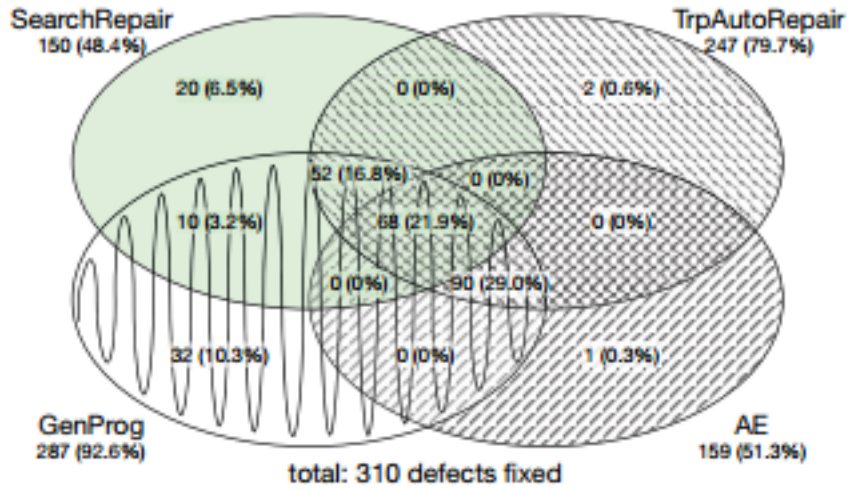

Fig. 8: SearchRepair is the only tool that can repair 20 (6.5%) of the 310 defects repaired by the four repair techniques. The other three repair tools can together repair 160 (51.6%) defects that SearchRepair cannot. The remaining 130 (41.9%) of the defects can be repaired by SearchRepair and at least one other tool. (Not shown in the diagram is that 35 (11.3%) of the defects can be repaired by both GenProg and TrpAutoRepair, and that 0 (0%) of the defects can be repaired by both SearchRepair and AE.)

* 正确率说明这里用已有开源代码作为备选库是很有意义的 *

（**GenProg**是文章三作在ICSE’12的工作）