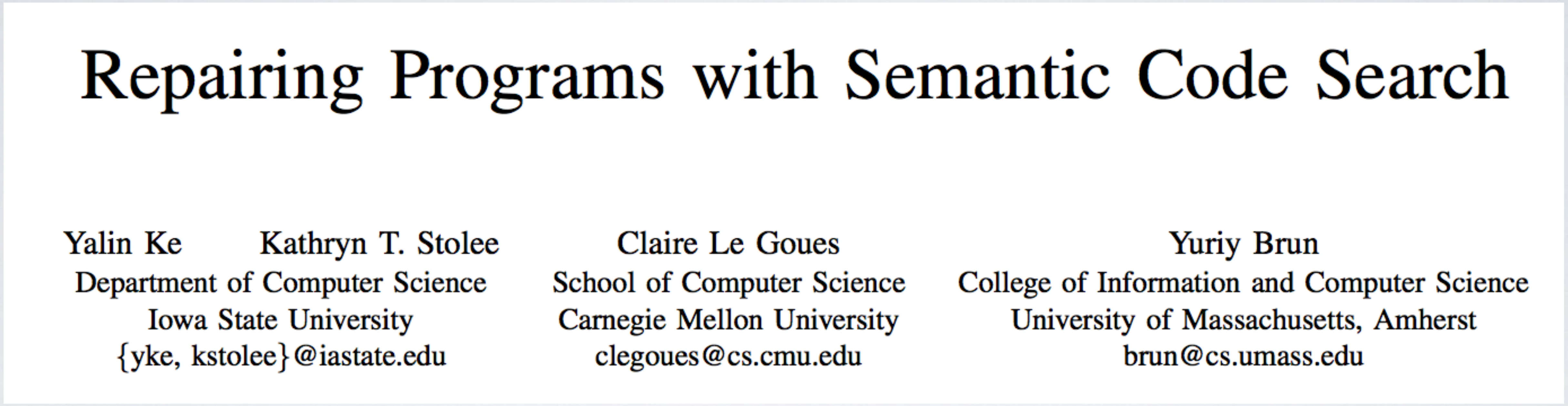


Repairing Programs with Semantic Code Search

Yalin Ke Kathryn T. Stolee
Department of Computer Science
Iowa State University
{yke, kstolee}@iastate.edu

Claire Le Goues
School of Computer Science
Carnegie Mellon University
clegoues@cs.cmu.edu

Yuriy Brun
College of Information and Computer Science
University of Massachusetts, Amherst
brun@cs.umass.edu

Summary By-Fardina Fathmiul Alam
SWE 795, Spring 2017

Repairing Programs with Semantic Code Search

Motivation

- ❑ Automated program repair can potentially reduce debugging costs and improve software quality. But existing resources shortcomings in the quality of automatically generated repairs.
- ❑ The key challenges is to find efficient semantically similar code (but not identical) to defective code and integration of that code into a buggy program.
- ❑ This paper has present a repair technique **SearchRepair** that can do automated program repair using semantic code search.

Repairing Programs with Semantic Code Search

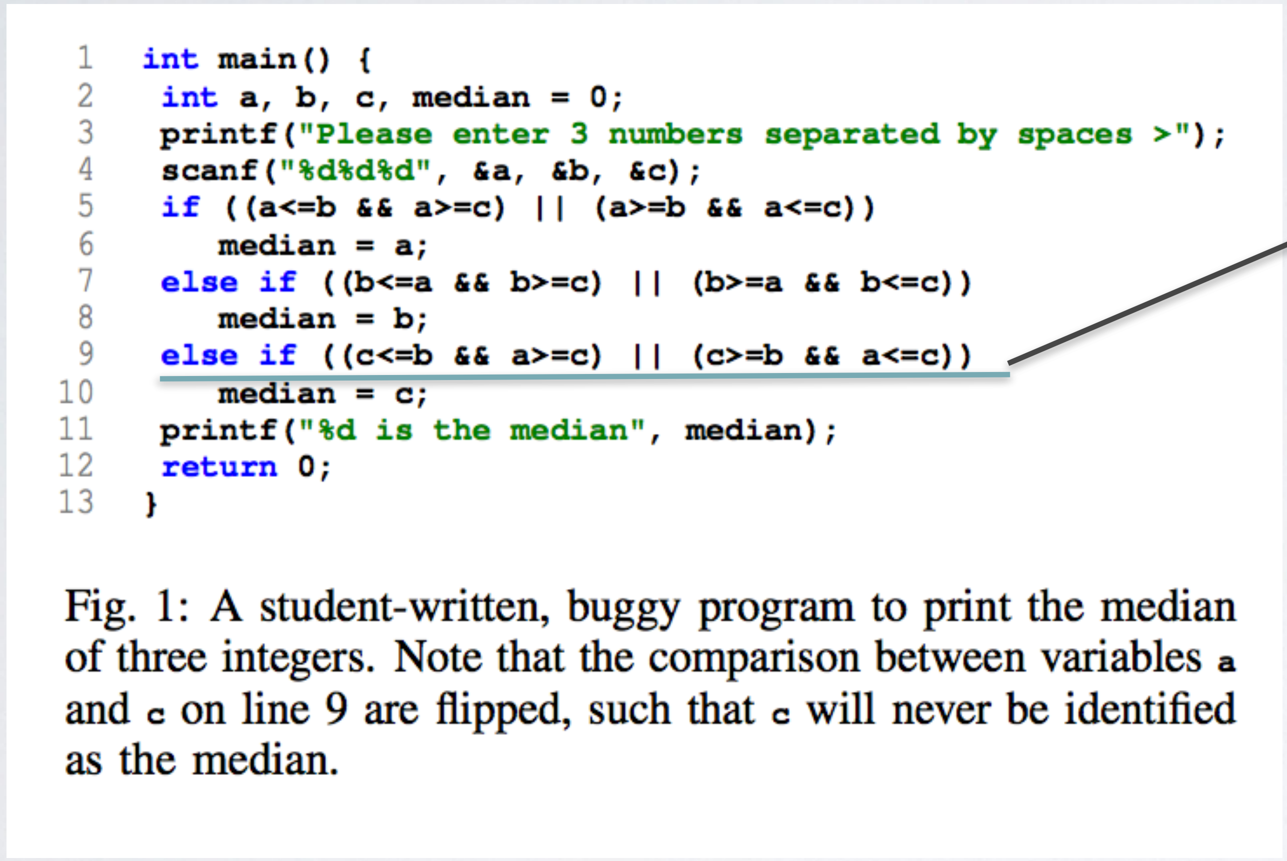
Key Idea

❑ **SearchRepair:**

1. Encodes a large database of human-written code fragments as satisfiability modulo theories (SMT) constraints on their input-output behavior.
2. Localizes a defect to likely buggy program fragments.
3. Constructs, for each fragment, the desired input-output behavior for code to replace those fragments
4. Uses state-of-the-art constraint solvers, to search the database for fragments that satisfy that desired behavior and replacing the likely buggy code with these potential patches.
5. Validates each potential patch to repair the bug against program test suites.

Approach (1/3)

SearchRepair uses fault localization to identify buggy fragments of code.

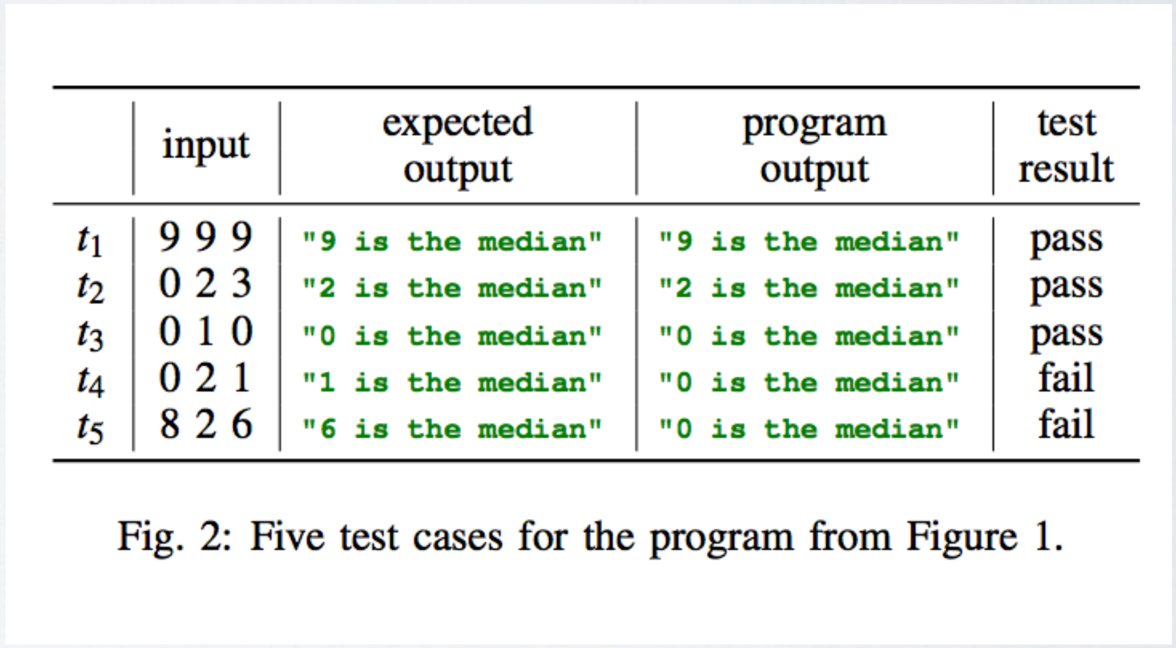


```
 1  int main() {
 2   int a, b, c, median = 0;
 3   printf("Please enter 3 numbers separated by spaces >");
 4   scanf("%d%d%d", &a, &b, &c);
 5   if ((a<=b && a>=c) || (a>=b && a<=c))
 6      median = a;
 7   else if ((b<=a && b>=c) || (b>=a && b<=c))
 8      median = b;
 9   else if ((c<=b && a>=c)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

Potentially buggy

Figures(1): Buggy Program to Print the Median of 3 Numbers



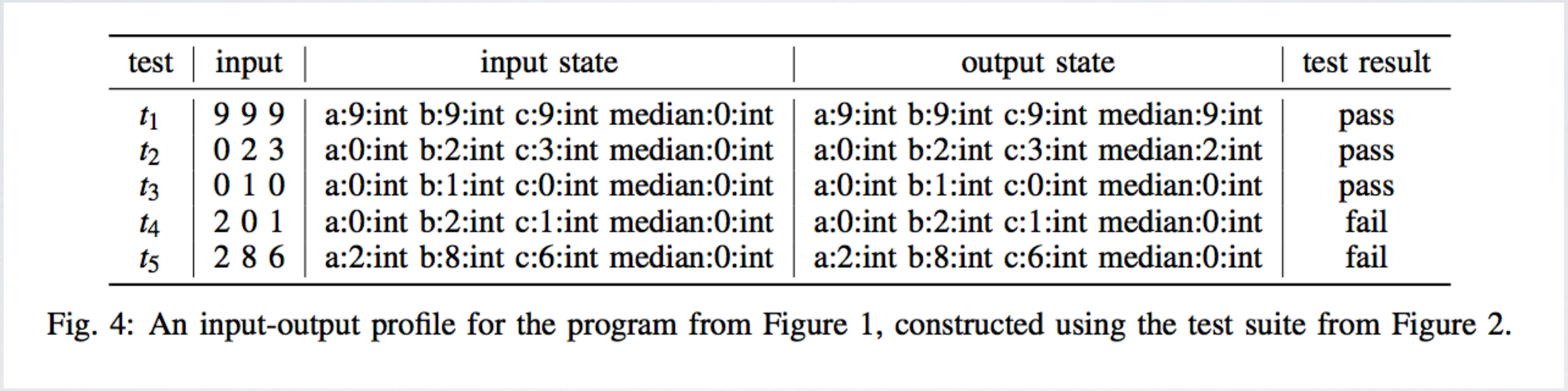
	input	expected output	program output	test result
t_1	9 9 9	"9 is the median"	"9 is the median"	pass
t_2	0 2 3	"2 is the median"	"2 is the median"	pass
t_3	0 1 0	"0 is the median"	"0 is the median"	pass
t_4	0 2 1	"1 is the median"	"0 is the median"	fail
t_5	8 2 6	"6 is the median"	"0 is the median"	fail

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

Figures(2): Five test cases for Figure (1)

Approach (2/3)

For each identified candidate buggy fragment, SearchRepair extracts program state in the form of dynamic variable values over the test cases (called profile).



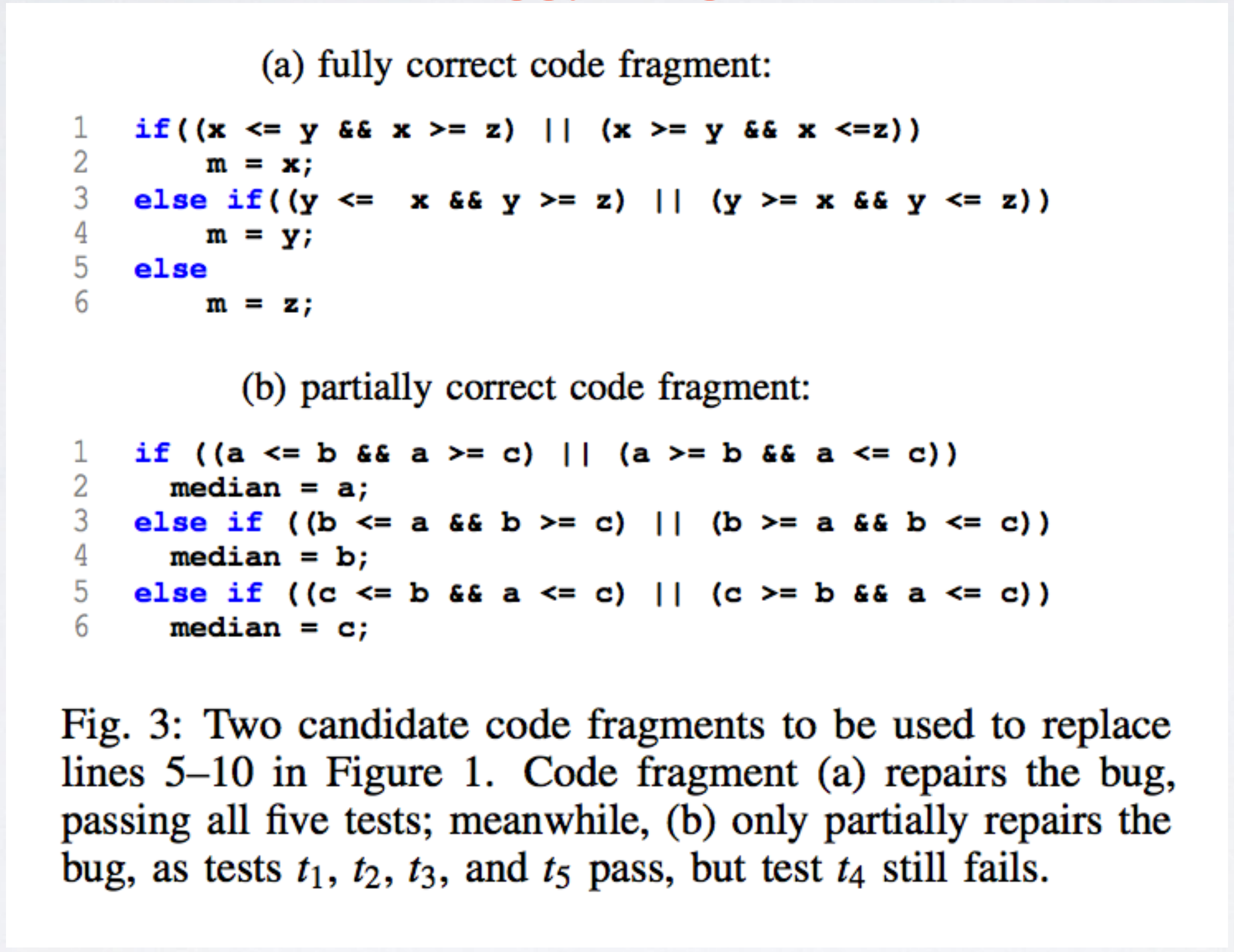
test	input	input state	output state	test result
t_1	9 9 9	a:9:int b:9:int c:9:int median:0:int	a:9:int b:9:int c:9:int median:9:int	pass
t_2	0 2 3	a:0:int b:2:int c:3:int median:0:int	a:0:int b:2:int c:3:int median:2:int	pass
t_3	0 1 0	a:0:int b:1:int c:0:int median:0:int	a:0:int b:1:int c:0:int median:0:int	pass
t_4	2 0 1	a:0:int b:2:int c:1:int median:0:int	a:0:int b:2:int c:1:int median:0:int	fail
t_5	2 8 6	a:2:int b:8:int c:6:int median:0:int	a:2:int b:8:int c:6:int median:0:int	fail

Fig. 4: An input-output profile for the program from Figure 1, constructed using the test suite from Figure 2.

Figures(4): An input-output profile

Approach (3/3)

SearchRepair uses these profiles to search a database of code to find code fragments that can serve as potential patches, replacing the buggy fragments.



(a) fully correct code fragment:

```
if((x <= y && x >= z) || (x >= y && x <=z))
    m = x;
else if((y <=  x && y >= z) || (y >= x && y <= z))
    m = y;
else
    m = z;
```

(b) partially correct code fragment:

```
if ((a <= b && a >= c) || (a >= b && a <= c))
  median = a;
else if ((b <= a && b >= c) || (b >= a && b <= c))
  median = b;
else if ((c <= b && a <= c) || (c >= b && a <= c))
  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.

Figures(3): Two Candidate code for repair buggy program

Experiment Result

- Figure 8 shows a Venn diagram describing the breakdown of which techniques repaired which defects. There are 310 total unique defects the tools were able to repair. Of these, 20 (6.5%) are unique to SearchRepair, SearchRepair can repair 20 of the defects that the other three techniques do not repair.

- 160 (51.6%) can be repaired by at least one other technique but not by SearchRepair, and 130 (41.9%) can be repaired by SearchRepair and at least one other technique.

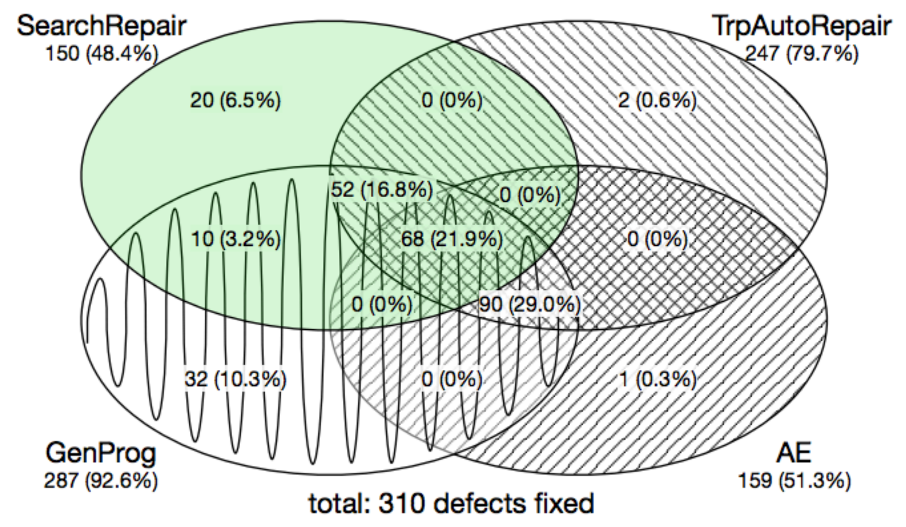
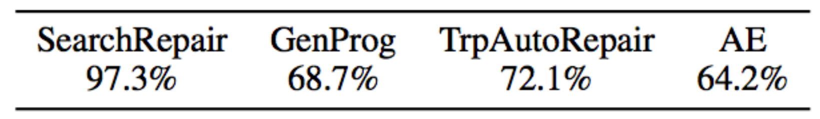


Fig. 8: SearchRepair is the only tool that can repair 20 (6.5%) of the 310 defects repaired by the four repair techniques. The other three repair tools can together repair 160 (51.6%) defects that SearchRepair cannot. The remaining 130 (41.9%) of the defects can be repaired by SearchRepair and at least 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.)



SearchRepair	GenProg	TrpAutoRepair	AE
97.3%	68.7%	72.1%	64.2%

Fig. 9: The quality of the patches produced by the four repair techniques, as measured by the number of independent (not used for patch generation) tests the patched programs pass.

QUESTIONS FOR DISCUSSION

- Overall reactions
- Would you like to use this technique?
- What limitations does this have?