SemFix: Program Repair via Semantic Analysis

Ye Wang, PhD student

Department of Computer Science

Virginia Tech

Problem Statement

- Debugging takes much time and effort
- Even after root cause of a bug is identified, fixing bug is non-trivial
- Problem solved by this paper is how to automatically repair bugs

Example

3 usable variables at line 4:
inhibit, up_sep, down_sep

```
1  int is_upward_preferred(int inhibit, int up_sep,
         int down_sep) {
2    int bias;
3    if(inhibit)
4      bias = down_sep;  //fix: bias=up_sep+100
5    else
6      bias = up_sep;
7    if (bias > down_sep)
8      return 1;
9    else
10     return 0;
11 }
```

To be fixed

Constraint

Line	Score	Rank
4	0.75	1
10	0.6	2
3	0.5	3
7	0.5	3
6	0	5
8	0	5

let bias = f(inhibit, up_sep, down_sep)
so that bias > down_sep can pass tests

Test	Inputs			Expected output	Observed output	Status
	inhibit	up_sep	down_sep			
1	1	0	100	0	0	pass
2	1	11	110	1	0	fail
3	0	100	50	1	1	pass
4	1	-20	60	1	0	fail
5	0	0	10	0	0	pass

Synthesize f
(1) try a constant:
cannot satisfy constraint ✗

(2) try to use "+", {v1+c, v1+v2}:
f = up_sep + 100 ✓

Background

- Statistical fault localization
  - Localize root-cause of program failure by exploiting the correlation between execution of faulty statements and program failure
- Component-based program synthesis
  - Generate a program that satisfies all the given input-output pairs.

Approach

- Only generate a repair by altering one statement. The generated fix is always with respect to a given test suite.
  - Generate repair constraint
  - Generate a fix

Approach

- Generate repair constraint
  - The paper focuses on repairs changing the right side of assignments or branch predicates
    - x = $f_{buggy}(\ldots) \rightarrow$ x = $f(\ldots)$
    - if($f_{buggy}(\ldots)$) $\rightarrow$ if($f(\ldots)$)
  - No side effect: f(…) do not modify any program variable

    Definition 1 (Repair Constraint): Given a program P, a test suite T, a repair constraint C of a function f_{buggy} in program P is a constraint over function f such that if $f \models C$, $P[f/f_{buggy}]$ passes all tests in T.

  - Repair constraint C is a conjunction of constraints derived from T. For each test t_i, there is a constraint Ci, $C = \bigwedge_{i=1}^{n} C_i$

Approach

- Generate repair constraint
  - Each C_i is a predicate over the function f
  - To generate C_i, the paper uses symbolic execution in a novel fashion.
  - Traditional symbolic execution takes all input variables as symbolic, while the paper's symbolic execution starts with a concrete input.
  - Execute the program concretely with input t_i to statement s. Denote the program state before executing statement s as ξ_i. Then set the result of function f(…) as symbolic and continue symbolic execution from statement s.

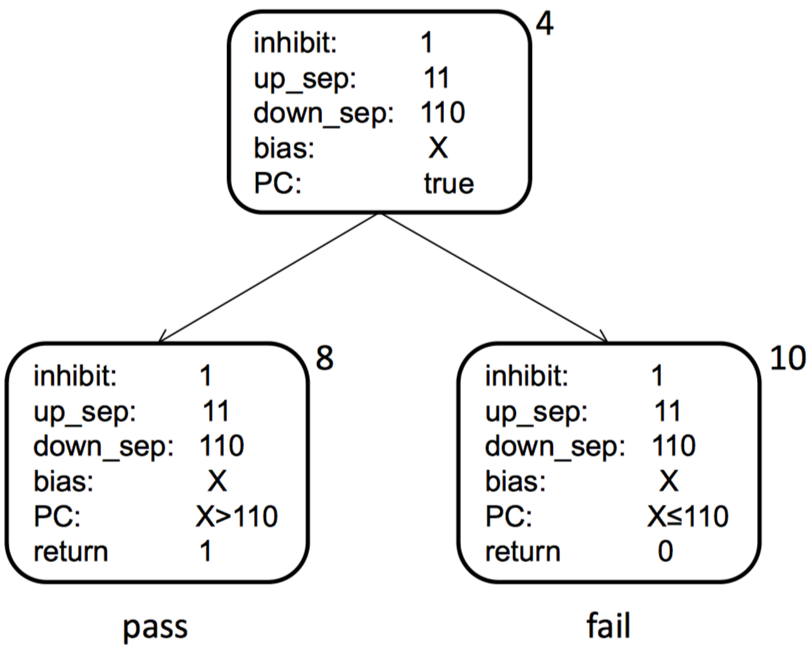
Approach

- Generate repair constraint
  - τ_i: symbolic value assigned to result of function f(…)
  - Symbolic execution explores m paths.
  - For each path π_j, pc_j is the associated path condition, and O_j is the symbolic expression of output
  - $O(t_i)$ is the expected output of program P with input t_i
  - Constraint: $$C_i := (\bigvee_{j=1}^{m} (pc_j \wedge O_j == O(t_i))) \wedge (f(\xi_i) == \tau_i)$$
  - First part means at least one feasible path along which output of program P is the same as the expected output.
  - Second part builds up input-output relationship of function f

Approach

- Generate repair constraint

Test	Inputs			Expected output	Observed output	Status
	inhibit	up_sep	down_sep			
1	1	0	100	0	0	pass
2	1	11	110	1	0	fail
3	0	100	50	1	1	pass
4	1	-20	60	1	0	fail
5	0	0	10	0	0	pass



$$C_i := \left(\bigvee_{j=1}^{m} (pc_j \wedge O_j == O(t_i))\right) \wedge (f(\xi_i) == \tau_i)$$

$(X > 110 \wedge 1 = 1) \vee (X \leq 110 \wedge 1 = 0)$
can be simplified to $X > 110$

$f(1{,}11{,}110) = X$

$f(1{,}11{,}110) > 110$

Approach

- Generate a fix
  - Component based program synthesis
  - Input-output pairs of to-be-synthesis program are encoded into constraints on a set of location variables L, a valuation of which leads to a program that satisfies the given input-output pairs.
  - Constraint $\psi_{func}(L, \alpha, \beta)$ dictates that the synthesized program must produce output β when given input α
  - Input-output pair $\langle \xi_i^k, \tau_i^k \rangle$ is generated when f is hit at the kth time in the execution of program P with input t_i, but $\langle \xi_i^k, \tau_i^k \rangle$ is symbolic in terms of $\{\tau_i^k | 1 \leq k \leq \omega\}$, where ω is number of times f is executed with input t_i

Approach

- Generate a fix
  - $\{\tau_i^k | 1 \leq k \leq \omega\}$ satisfy $(\bigvee_{j=1}^{m}(pc_j \wedge O_j == O(t_i)))$
  - f should satisfy the constraint

$$\theta_i \stackrel{\text{def}}{=\!=} \exists \overrightarrow{\tau_i}, \bigwedge_{k=1}^{w} \phi_{func}(L, \xi_i^k, \tau_i^k) \wedge (\bigvee_{j=1}^{m}(pc_j \wedge O_j == O(t_i)))$$

where $\tau_i := \{\tau_i^k | 1 \leq k \leq w\}$.

  - Conjoin constraints from all tests together with the well-formedness constraint $\psi_{\omega fp}$

$$\theta \stackrel{\text{def}}{=\!=} (\bigwedge_{i=1}^{n} \theta_i) \wedge \psi_{wfp}(L)$$

Approach

- Putting it all together
  - The algorithm takes as inputs a buggy program P, a test suite T and a ranked list of suspicious program statements RC
  - When successful, the algorithm produces a repair, applying which on P makes P pass all tests in the test suite T.

THE CATEGORIZATION OF BASIC COMPONENTS

Level	Conditional Statement	Assign Statement
1	Constants	Constants
2	Comparison ($>$, $\geq$, $=$, $\neq$)	Arithmetic ($+$, $-$)
3	Logic ($\wedge$, $\vee$)	Comparison, Ite
4	Arithmetic ($+$, $-$)	Logic
5	Ite, Array Access	Array Access
6	Arithmetic ($*$)	Arithmetic ($*$)

Evaluation

- Use SemFix to repair seeded defects and real defects in an open source software. The proposed method is also compared with genetic programming based repair techniques.

SUBJECT PROGRAMS FROM SIR REPOSITORY.

Subject Prog.	Size (LOC)	#Versions	Description
Tcas	135	41	air-traffic control program
Schedule	304	9	process scheduler
Schedule2	262	9	process scheduler
Replace	518	29	text processor
Grep	9366	2	text search engine
Total		**90**	

Evaluation

- Intuitively, it is harder to generate a repair to pass more tests
- Repairs generated with small number of tests may not be valid for other tests that are not in test suite.

COMPARING THE SUCCESS RATE BETWEEN SEMFIX (SF) AND GENPROG (GP). X IN [X] ON THE TOP OF EACH COLUMN DENOTES THE NUMBER OF TESTS.

Program	[10] SF/GP	[20] SF/GP	[30] SF/GP	[40] SF/GP	[50] SF/GP
Tcas	38 / 24	38 / 19	35 / 16	34 / 12	34 / 11
Schedule	5 / 1	3 / 1	4 / 1	4 / 0	4 / 0
Schedule2	4 / 4	3 / 2	4 / 2	3 / 3	2 / 1
Replace	7 / 6	7 / 5	8 / 5	7 / 6	6 / 4
Grep	2 / 0	1 / 0	1 / 0	2 / 0	2 / 0
Total	56 / 35	52 / 27	52 / 24	50 / 21	48 / 16

Evaluation

•

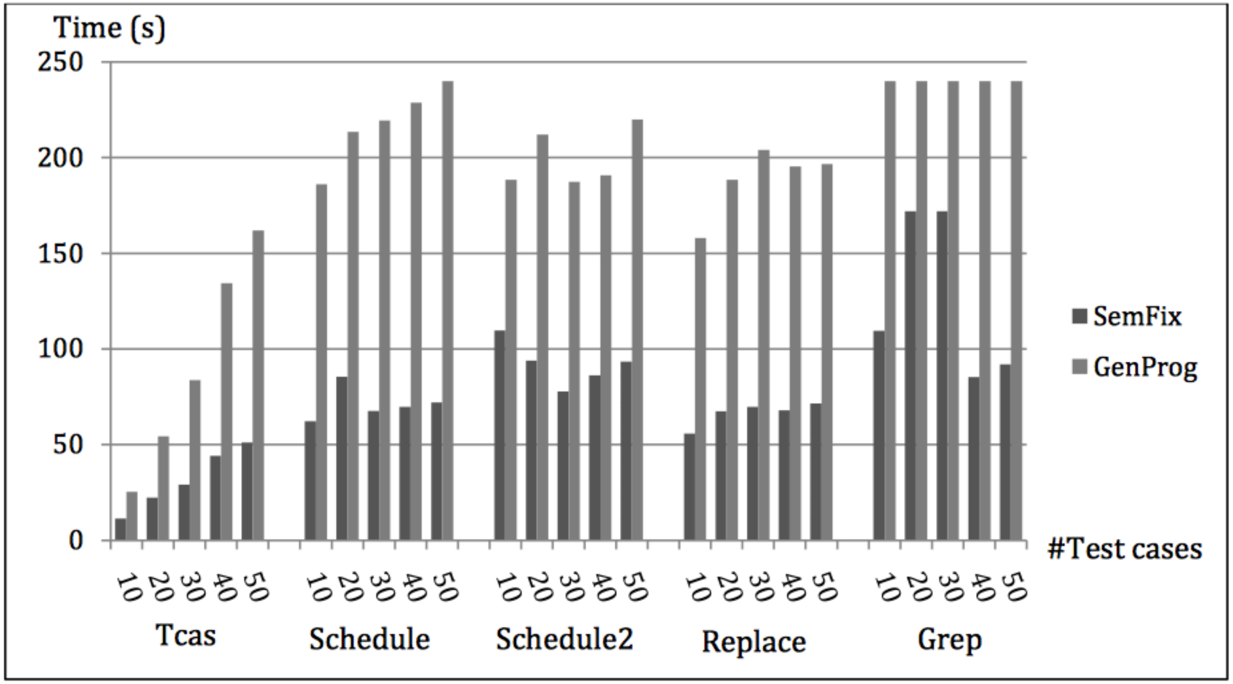


Fig. 4. Comparing the running time between SEMFIX and GenProg.

Evaluation

- Different types of bugs

SEMFIX (SF) VS. GENPROG (GP) IN REPAIRING DIFFERENT CLASS OF BUGS WITH 50 TESTS.

Bug type	Const	Arith	Comp	Logic	Code Missing	Redundant Code	All
Total	14	14	16	10	27	9	90
SemFix	10	6	12	10	5	5	48
GenProg	3	0	5	3	3	2	16

Related Work

- Genetic programming:
  - W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest, "Automatically finding patches using genetic programming," in ICSE, 2009.
  - C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer, "A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each," in ICSE, 2012.
- AutoFix-E and AutoFix-E2 are based on the program contracts in Eiffel programs
- Jobstmann, et. al. uses LTL specifications for finite state programs

Related Work

- Gopinath, et. al. use behavioral specifications and encode the specification constraint on the buggy program into SAT constraint
- Robert and Roderick employ template based repair for linear expressions.
- Dallmeier, et. al. try to generate fixes from object behavior anomalies.
- ClearView follows a similar scheme but works on deployed binary program when high availability is required.

Related Work

- BugFix suggests bug-fix that has been used in a similar debugging situation.
- Debroy and Wong propose to use mutation for program repair.
- PHPRepair focuses on HTML generation errors in PHP programs.
- Instead of fixing a buggy program, program sketching allows a programmer to write a sketch of the implementation idea while leaving the low level details omitted as holes to be automatically filled up by the sketch complier.

Conclusion

- The proposed SemFix is a semantics based program repair tool.
- The *repair constraint*, which is derived from a set of tests, is solved by generating a valid repair.
- SemFix can synthesize a repair even if the repair code does not exist anywhere in the program.

Discussion

- Which is easier, fixing a bug manually or verify the auto-generated bug fix?

Discussion

- Can you apply artificial intelligence (AI) to automatic bug repairing to improve it?
- If yes, how?

Discussion

- If you are a software engineer in an IT company, will you use an automatic bug repairing tool?
- If yes, which cases will you use it in? which cases will you not use it in?

Discussion

- The paper says "the test suite could be large and thus affect the scalability of our technique".
- Do you think selecting a subset of the entire test suite for repair generation is a good idea?

Discussion

- These basic components are used to generate a repairs.
- Do you think they are enough? Should we add something more, like division (/)?

THE CATEGORIZATION OF BASIC COMPONENTS

Level	Conditional Statement	Assign Statement
1	Constants	Constants
2	Comparison ($>, \geq, =, \neq$)	Arithmetic ($+, -$)
3	Logic ($\wedge, \vee$)	Comparison, Ite
4	Arithmetic ($+, -$)	Logic
5	Ite, Array Access	Array Access
6	Arithmetic ($*$)	Arithmetic ($*$)

Discussion

- The proposed method only synthesis an expression.
- Should we use some more complicated logics, like if-condition, for-loop, and while-loop?
- If yes, how will they affect the precision and speed of the bug repairing method?

Discussion

- The proposed method only change one statement.
- Do you think changing more statements is a good idea? Why?

Discussion

- To be honest, no matter how many test cases are used, we can not guarantee the bug fix is right.
- Can the bug repairing method use another constraint, instead of tests?

Discussion

- Can any other research be done based on SemFix?
- If yes, talk about the details.