Anti-patterns in Search-Based Program Repair

Paper by: Shin Hwei Tan, Hiroaki Yoshida, Mukul R. Prasad, and Abhik Roychoudhury. 2016.

Presentation by: Melisa Panaccione (40182167)

Overview - Anti-Pattern solution

- Addition to existing APR tools
- More meaningful patches
- Tested with GenProg and SPR

Contents

1 Background

2 Introduction

3 Research Questions

4 Experiments/Results

5 Limitations

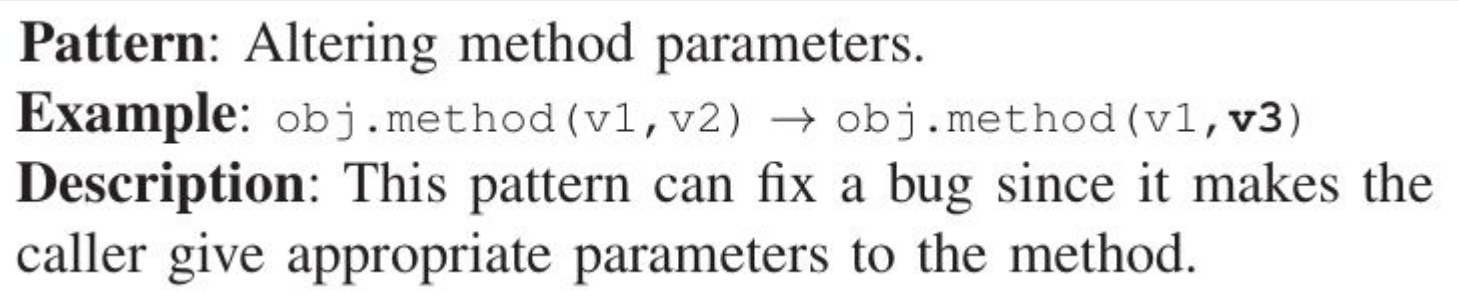
Contents

Background - Test Oracle

- Incomplete specification
- Weak tests (equivalence classes, coverage, etc.)
- Patched program may introduce new errors

Background - Templates

- Avoids unwanted solutions
- More “human” solutions
- Limitation: overfitting



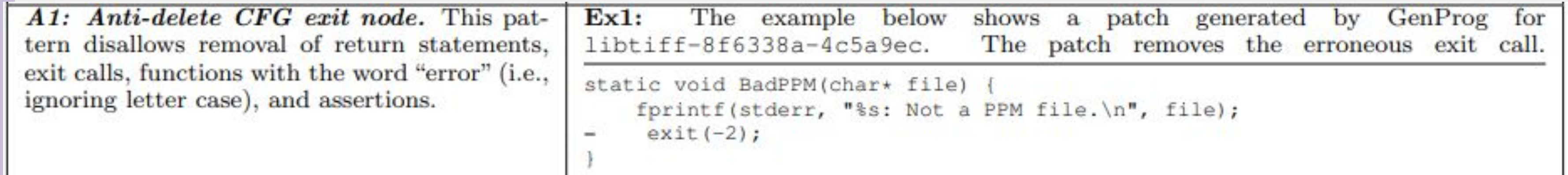
Pattern: Altering method parameters.
Example: `obj.method(v1,v2)` → `obj.method(v1,`**`v3`**`)`
Description: This pattern can fix a bug since it makes the caller give appropriate parameters to the method.

Background

Introduction

- Specify anti-patterns (less overfitting)
- Speeds up process (pruning)
- Better at localizing errors
- Less functionality removal
- Better correctness? Not conclusive

Introduction



A1: Anti-delete CFG exit node. This pattern disallows removal of return statements, exit calls, functions with the word "error" (i.e., ignoring letter case), and assertions.

Ex1: The example below shows a patch generated by GenProg for libtiff-8f6338a-4c5a9ec. The patch removes the erroneous exit call.

```
static void BadPPM(char* file) {
    fprintf(stderr, "%s: Not a PPM file.\n", file);
-     exit(-2);
}
```

Research Questions

Q1) How do anti-patterns affect the quality of patches generated by search-based program repair tools?

Q2) How many nonsensical patches can our anti-patterns eliminate to reduce manual inspection costs?

RQ3) When our modified tools produce the same patch, what is the speedup that we achieve?

RQ4) How does the use of anti-patterns compare to an approach that simply prohibits deletion?

Research Questions

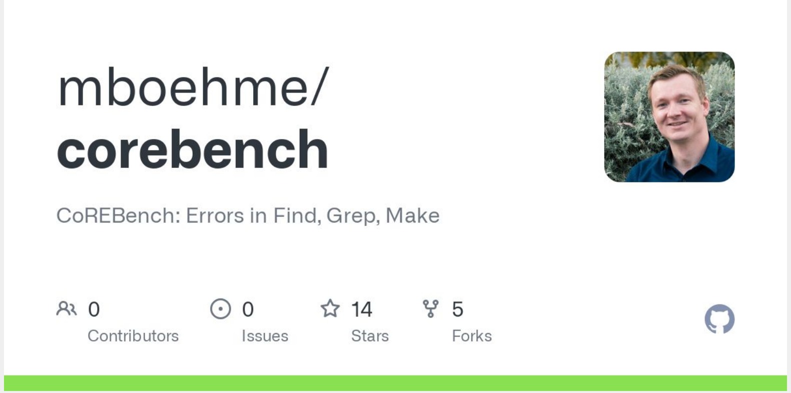
Experiment

- Benchmarks: GenProg and CoREBench
- New tools: mGenProg, mSPR

lindenb/**genprog**

Genetic programming in C

1 Contributor · 0 Issues · 2 Stars · 1 Fork



mboehme/ **corebench**

CoREBench: Errors in Find, Grep, Make

0 Contributors · 0 Issues · 14 Stars · 5 Forks

Experiment

Experiment

Finding out what changes lead to “plausible” patches

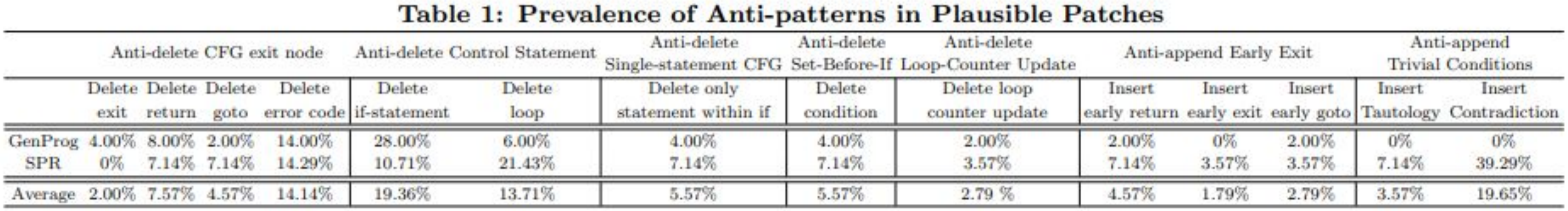


Table 1: Prevalence of Anti-patterns in Plausible Patches

	Anti-delete CFG exit node				Anti-delete Control Statement		Anti-delete Single-statement CFG	Anti-delete Set-Before-If	Anti-delete Loop-Counter Update	Anti-append Early Exit			Anti-append Trivial Conditions	
	Delete exit	Delete return	Delete goto	Delete error code	Delete if-statement	Delete loop	Delete only statement within if	Delete condition	Delete loop counter update	Insert early return	Insert early exit	Insert early goto	Insert Tautology	Insert Contradiction
GenProg	4.00%	8.00%	2.00%	14.00%	28.00%	6.00%	4.00%	4.00%	2.00%	2.00%	0%	2.00%	0%	0%
SPR	0%	7.14%	7.14%	14.29%	10.71%	21.43%	7.14%	7.14%	3.57%	7.14%	3.57%	3.57%	7.14%	39.29%
Average	2.00%	7.57%	4.57%	14.14%	19.36%	13.71%	5.57%	5.57%	2.79 %	4.57%	1.79%	2.79%	3.57%	19.65%

Experiment

Experiment

Set of 7 proposed anti-patterns:

1) Anti-delete CFG exit node
2) Anti-delete Control Statement
3) Anti-delete Single-statement CFG
4) Anti-delete Set-Before-If
5) Anti-delete Loop-Counter Update
6) Anti-append Early Exit
7) Anti-append Trivial Conditions

Experiment - Evaluation

1) Same Patch (original and modified tool)
2) Localizes Correct Line
3) Localizes Correct Function but Incorrect Line
4) Removes Less Functionality
5) No Repair

Experiment - GenProg vs mGenProg

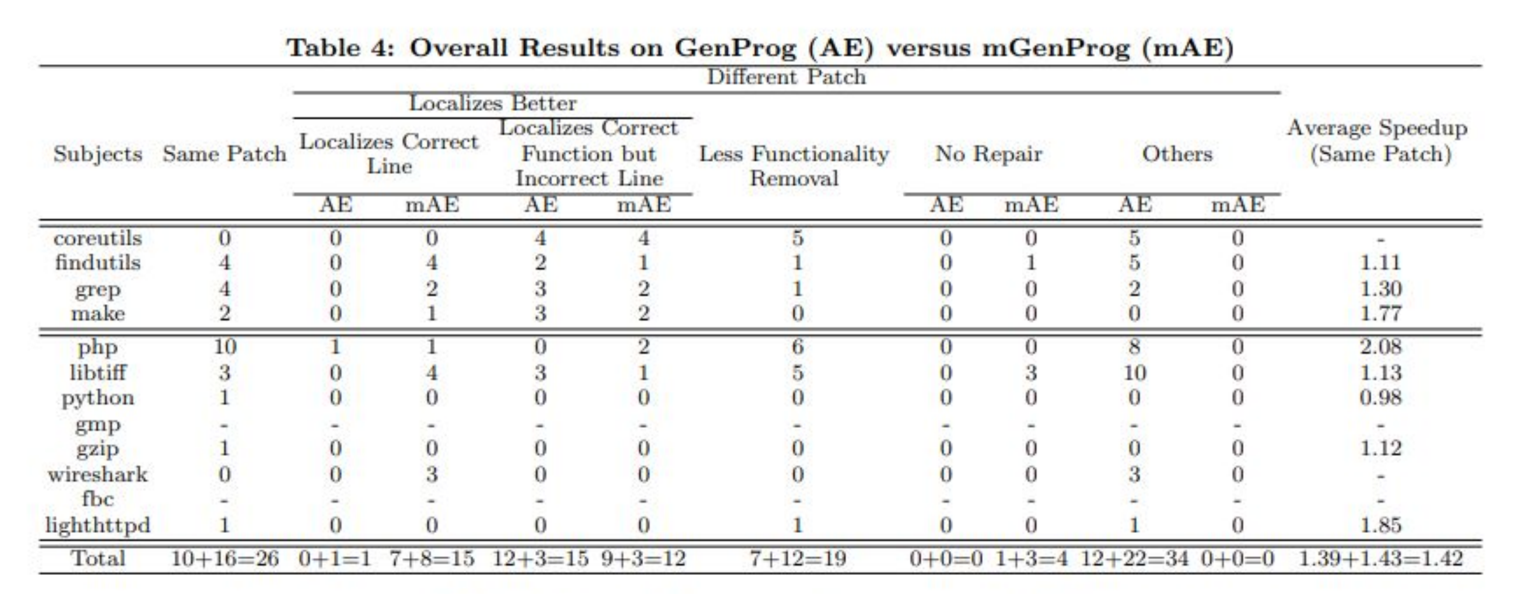


Table 4: Overall Results on GenProg (AE) versus mGenProg (mAE)

		Different Patch									
		Localizes Better									
Subjects	Same Patch	Localizes Correct Line		Localizes Correct Function but Incorrect Line		Less Functionality Removal	No Repair		Others		Average Speedup (Same Patch)
		AE	mAE	AE	mAE		AE	mAE	AE	mAE	
coreutils	0	0	0	4	4	5	0	0	5	0	-
findutils	4	0	4	2	1	1	0	1	5	0	1.11
grep	4	0	2	3	2	1	0	0	2	0	1.30
make	2	0	1	3	2	0	0	0	0	0	1.77
php	10	1	1	0	2	6	0	0	8	0	2.08
libtiff	3	0	4	3	1	5	0	3	10	0	1.13
python	1	0	0	0	0	0	0	0	0	0	0.98
gmp	-	-	-	-	-	-	-	-	-	-	-
gzip	1	0	0	0	0	0	0	0	0	0	1.12
wireshark	0	0	3	0	0	0	0	0	3	0	-
fbc	-	-	-	-	-	-	-	-	-	-	-
lighthttpd	1	0	0	0	0	1	0	0	1	0	1.85
Total	10+16=26	0+1=1	7+8=15	12+3=15	9+3=12	7+12=19	0+0=0	1+3=4	12+22=34	0+0=0	1.39+1.43=1.42

Experiment

Experiment - SPR vs mSPR

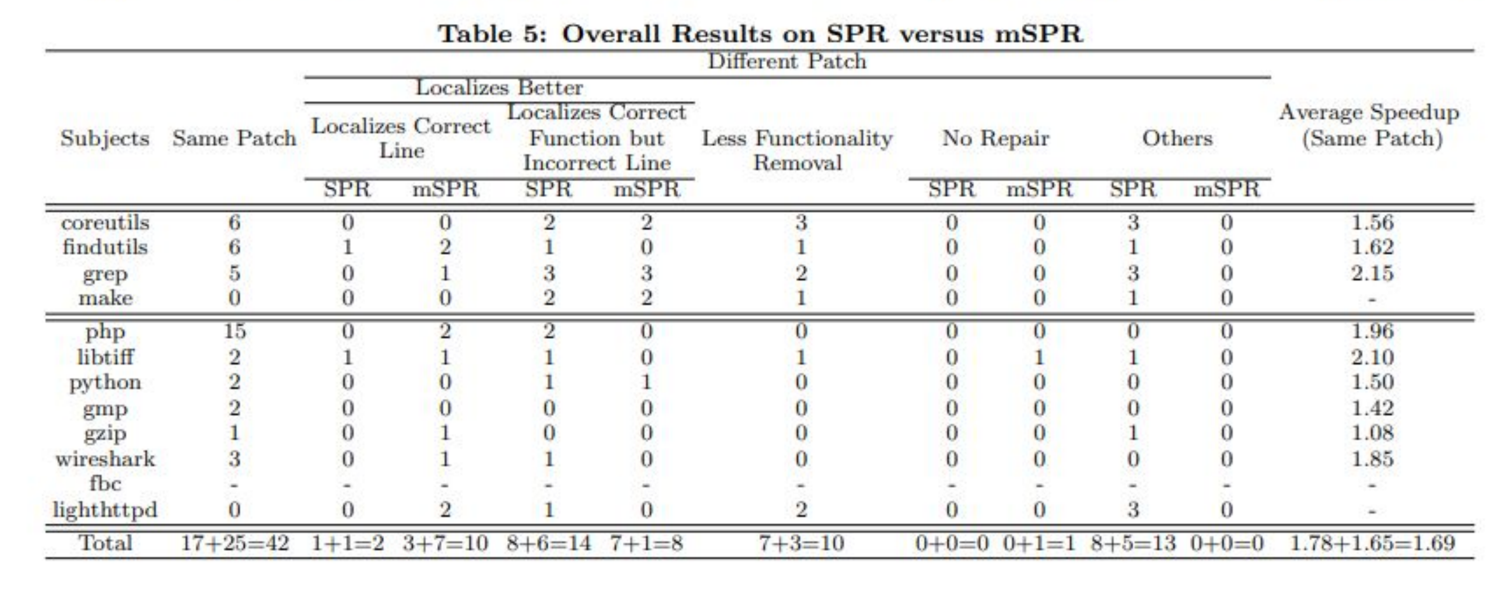


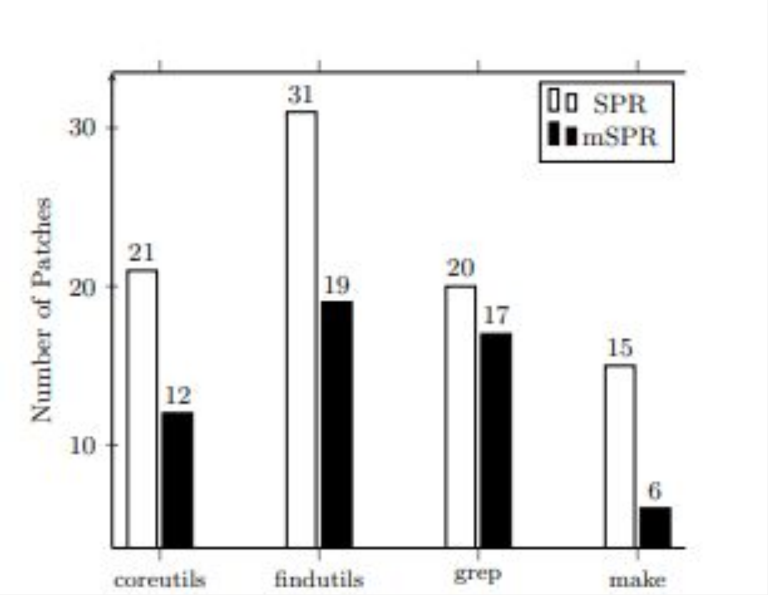
Table 5: Overall Results on SPR versus mSPR

Subjects	Same Patch	Different Patch									Average Speedup (Same Patch)
		Localizes Better									
		Localizes Correct Line		Localizes Correct Function but Incorrect Line		Less Functionality Removal	No Repair		Others		
		SPR	mSPR	SPR	mSPR		SPR	mSPR	SPR	mSPR	
coreutils	6	0	0	2	2	3	0	0	3	0	1.56
findutils	6	1	2	1	0	1	0	0	1	0	1.62
grep	5	0	1	3	3	2	0	0	3	0	2.15
make	0	0	0	2	2	1	0	0	1	0	-
php	15	0	2	2	0	0	0	0	0	0	1.96
libtiff	2	1	1	1	0	1	0	1	1	0	2.10
python	2	0	0	1	1	0	0	0	0	0	1.50
gmp	2	0	0	0	0	0	0	0	0	0	1.42
gzip	1	0	1	0	0	0	0	0	1	0	1.08
wireshark	3	0	1	1	0	0	0	0	0	0	1.85
fbc	-	-	-	-	-	-	-	-	-	-	-
lighthttpd	0	0	2	1	0	2	0	0	3	0	-
Total	17+25=42	1+1=2	3+7=10	8+6=14	7+1=8	7+3=10	0+0=0	0+1=1	8+5=13	0+0=0	1.78+1.65=1.69

Experiment

Result discussion

- Improved localization
- Less functionality removal
- No difference in correctness
- Less patches created
- Speed up
  - 41% GenProg
  - 27% SPR



Limitations

- Generalized anti-patterns
- Weak evaluation of “correctness”

Thank You😊🦄