Measuring and improving quality of automated program repair

Yuriy Brun, UMass Amherst

Ted Smith

Yalin Ke

Manish Motwani

Mauricio Soto

Earl Barr

Prem Devanbu

Claire Le Goues

René Just

Katie Stolee
Automated Program Repair

• Given a software system with a bug
  – (typically) a set of passing and a set of failing tests –
produce a variant of that software system without the bug.

Given a system S that passes tests $T_p$ and fails tests $T_f$, automatically produce $S'$ that passes $T_p$ and $T_f$
Exploration-based approaches

Basic idea:

- Buggy program
- Passing tests
- Failing tests

APR tool
- Mutate / synthesize
- Evaluate candidates
- Repeat

Patched program
the many exploration-based repair tools

The automatic program repair story

• Early papers asked:
  – What fraction of bugs can APR fix?
  – How long does it take APR to fix bugs?
  – How much does it cost for APR to fix bugs?
  – Can humans maintain APR fixes?

The story was, APR produces a patch that passes all tests implies problem solved
Cobra effect
Does exploration-based repair repair?

buggy program

passing tests

failing tests

APR tool

mutate / synthesize
evaluate candidates
repeat

patched program

The patch may break untested or under-tested functionality
How can we know if APR repairs

• Look at the produced patches by hand
  [Qi, Long, Achour, Rinard, ISSTA 2015]
  [Durieux, Martinez, Monperrus, Sommerard, Xuan, 2015]

• Have others look at the produced patches by hand
  [Fry, Landau, Weimer, ISSTA 2012]
  [Kim, Nam, Song, Kim, ICSE 2013]

• Produce patches with test suite T,
  evaluate them on independent test suite T'
  [Brun, Barr, Xiao, Le Goues, Devanbu, 2013]
  [Smith, Barr, Le Goues, Brun, ESEC/FSE 2015]
  – objective
  – repeatable
IntroClass Benchmark

Requires a large set of bugs for programs with 2 independent test suites and the test suites need to be good

- IntroClass: 998 bugs in very small, student-written C programs, with a KLEE-generated test suite, and a human-written test suite.

- [http://repairbenchmarks.cs.umass.edu](http://repairbenchmarks.cs.umass.edu), [TSE 2015]
Do GenProg and TrpAutoRepair patches pass kept-out tests?
More GenProg and TrpAutoRepair findings

- The better the test suite coverage, the better the patch

- APR causes harm to high-quality programs, but is helpful for low-quality programs
- Human-written tests lead to better patches
- Student-written patches also break tests.

More answers and details in “Is the Cure Worse Than the Disease? Overfitting in Automated Program Repair” by Smith, Barr, Le Goues, Brun, ESEC/FSE 2015
Can we **improve** the patch quality?

• Recent work:
  – SPR [Long and Rinard, ESEC/FSE 2015]
  – Prophet [Long and Rinard, POPL 2016]

• Both SPR and Prophet produce more correct patches than GenProg, TrpAutoRepair, AE

• My vision: repair at a higher level
SearchRepair: Use existing code

Replace whole code blocks with code from other projects (e.g., GitHub)

Imagine a program with a buggy sort method:

Option 1
Mutate, synthesize, and tweak the sort method until a set of sorting tests pass

Option 2
Find a method on GitHub that passes the sorting tests

“Repairing Programs with Semantic Code Search” by Ke, Stolee, Le Goues, Brun, ASE 2015
SearchRepair: Use existing code

Replace whole code blocks with code from other projects (e.g., GitHub)
Example: median

```c
int main() {
    int a, b, c, median = 0;
    printf("Please enter 3 numbers separated by spaces >");
    scanf("%d%d%d", &a, &b, &c);
    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;
    printf("%d is the median", median);
    return 0;
}
```

<table>
<thead>
<tr>
<th>test</th>
<th>input</th>
<th>test result</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>9 9 9</td>
<td>✔</td>
</tr>
<tr>
<td>t₂</td>
<td>0 2 3</td>
<td>✔</td>
</tr>
<tr>
<td>t₃</td>
<td>0 1 0</td>
<td>✔</td>
</tr>
<tr>
<td>t₄</td>
<td>2 0 1</td>
<td>✗</td>
</tr>
<tr>
<td>t₅</td>
<td>2 8 6</td>
<td>✗</td>
</tr>
</tbody>
</table>
Encoding

Given snippets of code, automatically compute the SMT constraints between snippet inputs and outputs. Store in DB.

```plaintext
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;
```

vars: LOCAL(int x, int y, int z, int m)

\( p_1 \):
ASSUME[(x <= y && x >= z) || (x >= y && x <= z)]
STMT[m = x]

\( p_2 \):
ASSUME[not((x <= y && x >= z) || (x >= y && x <= z))
&& ((y <= x && y >= z) || (y >= x && y <= z))]
STMT[m = y]

\( p_3 \):
ASSUME[not((x <= y && x >= z) || (x >= y && x <= z))
&& not((y <= x && y >= z) || (y >= x && y <= z))]
STMT[m = z]
Fault localization

Identify the code lines that execute more often on failing tests, the elevate these lines to block level.

```c
int main() {
    int a, b, c, median = 0;
    printf("Please enter 3 numbers separated by spaces >");
    scanf("%d%d%d", &a, &b, &c);
    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;
    printf("%d is the median", median);
    return 0;
}
```
Semantic search and context

Identify input-output behavior on passing tests, and use SMT solver to find satisfying snippets in DB (potential patches).

```c
int main() {  
    int a, b, c, median = 0;
    printf("Please enter 3 numbers separated by spaces >");
    scanf("%d%d%d", &a, &b, &c);
    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<=a && a>=c) || (c>=a && a<=c))
        median = c;
    printf("%d is the median", median);
    return 0;
}
```

```c
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;
```

Barr, Harman, Jia, Marginean, Petke, ISSTA 2015 could enable larger-scale transplantation
Validate potential patches

Rerun tests to select patches that repair the bug.
SearchRepair: Use existing code

- Replace whole code blocks with code from other projects (e.g., GitHub)
# SearchRepair vs. Exploration

% of kept-out tests patches pass

<table>
<thead>
<tr>
<th>Method</th>
<th>% Pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>SearchRepair</td>
<td>97.2%</td>
</tr>
<tr>
<td>GenProg</td>
<td>68.7%</td>
</tr>
<tr>
<td>TRPAutoRepair</td>
<td>72.1%</td>
</tr>
<tr>
<td>AE</td>
<td>64.2%</td>
</tr>
</tbody>
</table>

“Repairing Programs with Semantic Code Search” by Ke, Stolee, Le Goues, Brun, ASE 2015
Contributions

• Repeatable, automated, objective methodology for evaluating automated repair quality
  – including the IntroClass dataset

• SearchRepair: semantic-search-based repair

• A small-scale prototype of SearchRepair, evaluated on IntroClass
  – greatly improves repair quality over GenProg, TrpAutoRepair, and AE