Upcoming

• Homework 3 due April 18
• Literature review due today April 11

Repairing Automated Repair

Repairing Automated Repair

Cobra Effect

What do cobras have to do with automated program repair?

Automated Program Repair

basic idea:

repairing python programs?
Automated program repair

Program repair techniques

Potential problem

the patched program may pass all given tests, but break other functionality
COMPUTE THE MEDIAN OF THREE NUMBERS

```c
int median(int a, int b, int c) {
    int result;
    if ((b<=a && a<=c) || (c<=a && a<=b))
        result = a;
    if ((a<b && b <= c) || (c<=b && b<a))
        result = b;
    if ((a<c && c<b) || (b<c && c<a))
        result = c;
    return result;
}
```
int median(int a, int b, int c) {
    int result = 0;
    if (b <= a && a <= c) || (c <= a && a <= b)
        result = a;
    if (a < b && b <= c) || (c < b && b <= a)
        result = b;
    if (a < c && c < b) || (b < c && c < b)
        result = c;
    return result;
}

int median(int a, int b, int c) {
    int result = 0;
    if (b <= a && a <= c) || (c <= a && a <= b)
        result = a;
    if (a < b && b <= c) || (c < b && b <= a)
        result = b;
    if (a < c && c < b) || (b < c && c < b)
        result = c;
    return result;
}

int med_broken(int a, int b, int c) {
    int result;
    if ((a == b) || (a == c) || (b < a && a < c) || (c < a && a < b))
        result = a;
    else if ((b == c) || (a < b && b < c) || (c < b && b < a))
        result = b;
    else if (a < c && c < b)
        result = c;
    return result;
}

int med_broken(int a, int b, int c) {
    int result;
    if ((a == b) || (a == c) || (b < a && a < c) || (c < a && a < b))
        result = a;
    else if ((b == c) || (a < b && b < c) || (c < b && b < a))
        result = b;
    else if (a < c && c < b)
        result = c;
    return result;
}

int med_broken(int a, int b, int c) {
    int result;
    if ((a == b) || (a == c) || (b < a && a < c) || (c < a && a < b))
        result = a;
    else if ((b == c) || (a < b && b < c) || (c < b && b < a))
        result = b;
    else if (a < c && c < b)
        result = c;
    return result;
}
```c
int med_broken(int a, int b, int c) {
    int result;
    if (a==b) || (a==c) ||
        (b==b && c==c) ||
        (c==a && b==b))
        result = a;
    else if (b < a) (b==c) || (a<b && b<c) ||
        (c<b && b<a))
        result = b;
    else if (a<c && c<b)
        result = c;
    return result;
}
```

```c
int med_broken(int a, int b, int c) {
    int result;
    if ((a==b) || (a==c) ||
        (b==b && c==c) ||
        (c==a && b==b))
        result = a;
    if (b < a)
        result = c;
    else if (a<c && c<b)
        result = c;
    return result;
}
```

```c
int med_broken(int a, int b, int c) {
    int result;
    if ((a==b) || (a==c) ||
        (b==b && c==c) ||
        (c==a && b==b))
        result = a;
    if (b < a)
        result = c;
    else if (a<c && c<b)
        result = c;
    return result;
}
```

```c
int med_broken(int a, int b, int c) {
    int result;
    if ((a==b) || (a==c) ||
        (b==b && c==c) ||
        (c==a && b==b))
        result = a;
    if (b < a)
        result = c;
    else if (a<c && c<b)
        result = c;
    return result;
}
```
Potential solution

Focus of prior evaluations
- Most evaluations are interested in whether tools work
  - produce patches
- Some interest in other factors
  - human acceptance of patches
  - plausibility
- ...but these don’t fully assess functional correctness
- No evaluations test functional correctness of repair outputs independently of repair inputs

What do we need?
- We need bugs with 2 test suites
  - and the test suites need to be good

Why?
- it’s hard enough to find one good test suite, good luck finding programs with two

Make your own!

http://repairbenchmarks.cs.umass.edu
998 student-written buggy C programs
- simple (very small)
- have 2 test suites
  - white-box (generated by KLEE)
  - black-box (written by instructor)

Some programs fail some wb tests, others bb tests, others, some of both

What do we need?
- We need bugs with 2 test suites

Focus of prior evaluations
- Most evaluations are interested in whether tools work
  - produce patches
- Some interest in other factors
  - human acceptance of patches
  - plausibility
- ...but these don’t fully assess functional correctness
- No evaluations test functional correctness of repair outputs independently of repair inputs

RQ1: What is the base incidence of overfitting?
Give a repair tool the buggy program and the black-box test suite, try to repair it, see what fraction of the white-box tests the patches pass.

RQ1: What is the base incidence of overfitting?
but first, how often can we actually generate patches?

<table>
<thead>
<tr>
<th>repair tool</th>
<th>patch production %</th>
</tr>
</thead>
<tbody>
<tr>
<td>GenProg</td>
<td>466/778 = 59.9%</td>
</tr>
<tr>
<td>TrpAutoRepair</td>
<td>444/778 = 57.1%</td>
</tr>
</tbody>
</table>
RQ1: What is the base incidence of overfitting?

RQ2: What effect do pre-repair test failures have on overfitting?

Programs that fail more tests before repair still fail more tests after repair

Repair is at best unlikely to improve correctness, at worst likely to worsen it

RQ3: What effect does test suite coverage have on overfitting?

• Randomly sample 25%, 50%, and 75% of passing and failing tests for each buggy program
• Attempt to repair programs
  – with each level of test coverage
• If a repair is found, measure correctness of repair

Lower test suite coverage leads to more overfitting

RQ4: What effect does test suite provenance have on overfitting?

• So far, all experiments have used human-written black-box tests to build repairs
• Switch to using KLEE-generated white-box tests
• Attempt to repair programs
• If a repair is found, measure correctness of repair
  – this time with block-box tests
RQ4: What effect does test suite provenance have on overfitting?

Automatically generated tests produced significantly buggier repairs compared to human-written tests.

RQ4: Do tools do better than novices?

Summary of that study

• Overfitting is a real concern
  – median patch for either tool passed only 75% of evaluation suite
• Overfitting is hard to avoid
  – minimization doesn’t help on this dataset
  – N-version voting only works in extreme cases
• Program repair is harder for buggier programs, but likely to break more correct programs
• Novice developers don’t significantly beat repair tools

How well does APR work?

Results on 11 tools, 2,141 bugs:
• Generate patches for 15-213 bugs
• Evaluations overfit to their benchmarks

11 tools generated patches on 15-213 out of 2,141 bugs.
How well does APR work?

- Evaluated 4 techniques
  - GenProg
  - Par
  - TrpAutoRepair
  - SimFix
- Measured patch quality
- Measured what affects patch quality

Quality vs. quantity

When applied to real-world Java code, APR produces patches for 10.6-19.0% of the defects. Less than half (14-46%) of the patches are correct.

Does APR at least improve things a bit?

- SearchRepair, a brand new technique, reduces overfitting to 97.2%.
- Most SearchRepair repairs pass 100% of the held-out test suite. (Select few poor repairs drop the overall rate.)

So is there no hope?

- SearchRepair, a brand new technique, reduces overfitting to 97.2%.
- Most SearchRepair repairs pass 100% of the held-out test suite. (Select few poor repairs drop the overall rate.)

Takeaway: Tests are an imperfect oracle, so APR suffers, producing low-quality patches. Can we find a domain with better oracles?

Read more about SearchRepair: