Midterm Review
and
Automated Repair Evaluation

Course updates

- Homework 5 in posted
- Midterm this Wednesday, Nov 18, in class
- Final report assignment will be posted soon

Feedback: 2 opportunities

- http://goo.gl/rqRRln

CS 521/621 feedback form (Fall 2015)

Please help me improve the class! I would love to get your feedback. --Yuriy

What do you like MOST about the class?

What do you like LEAST about the class?

If you were in charge, what would you do to improve the class?

Any other comments?

Feedback: 2 opportunities

DTA feedback last 15-minutes today

thanks!

Today's plan

- Midterm review
  - What kinds of questions to expect
  - Examples of questions
  - How to attack the hard questions
  - Topics to be covered
  - Your questions
- new topic: Evaluating automated repair
- DTA feedback

What's the midterm like?

- Some true/false questions
- Some multiple choice questions
- Some reasoning questions

True / False Example

Automatically predicting collaboration conflicts, if applied properly, would eliminate the need for resolving conflicts, which would greatly improve software development productivity.

Multiple Choice Example

Rational Purify can find the following types of bugs (check all that apply):

A. Writing past the end of an array
B. Reading past the end of an array
C. Writing past the end of the first object in an array of objects
D. Null pointer exceptions
E. Using a different method than the developer intended

Reasoning

- Reasoning are the harder questions that require abstraction and application of what you learnt.
- Reasoning questions will largely cover the papers presented in class, and the homework assignments

Reasoning Example

Consider this simple concurrent program.

…

Does it have any races?

Why does CheckSync, from homework 4, report the following race?

…

When Solving Reasoning Problems

- Important to pause for a moment to think about how to proceed.
- Plan your attack and evidence you will use to support your answer.
- You will have scratch paper to use to organize your thoughts (scratch will not be graded).

Come up with an answer, and its support, and write it clearly, concisely in the provided space.

Topics to be covered

- Dynamic analysis
  - Daikon and Purify
- Software development lifecycle
  - ad hoc, code and fix, waterfall, spiral, staged, scrum
- Test generation
  - Korat, Chronicler and BugRedux (field failures), mobile testing and recovery, mutation testing

Topics to be covered

- Automated Bug Fixing
  - redundant methods, Par, staged repair, web app method substitutions, program boosting (crowd),
- Speculative Analysis
  - Quick fix scout, Crystal, CodeHint
- Software security
  - sTile and smart redundancy

Now your chance

…to ask me questions about these topics

- dynamic analysis
- development lifecycle
- test generation
- automated bug fixing
- speculative analysis
- software security

Automated Repair Evaluation

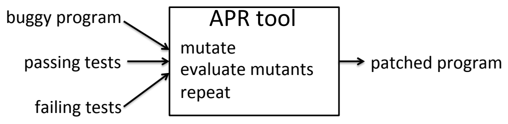
Cobra effect

What do cobras have to do with automated program repair?

repairing python programs?

Automated Program Repair

basic idea:



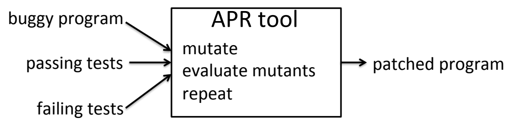
the many repair tools

ClearView [Perkins et al. 2009] GenProg [Weimer et al. 2009]
TDS [Perelman et al. 2014]
AE [Weimer et al. 2013] PAR [Kim et al. 2013]
AutoFix-E [Wei et al. 2010] SemFix [Nguyen et al. 2013]

[Carzaniga et al. 2010] [Carzaniga et al. 2013]
[Forrest et al. 2009] [Jin et al. 2011]
[Coker and Hafiz 2013] [Debroy and Wong 2010]

[Novark et al. 2007] [Demsky et al. 2006] [Lin and Ernst 2004]

Potential problem



the patched program may pass all given tests, but break other functionality

COMPUTE THE MEDIAN OF THREE NUMBERS

```
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<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<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<a))
    result = c;
  return result;
}
```

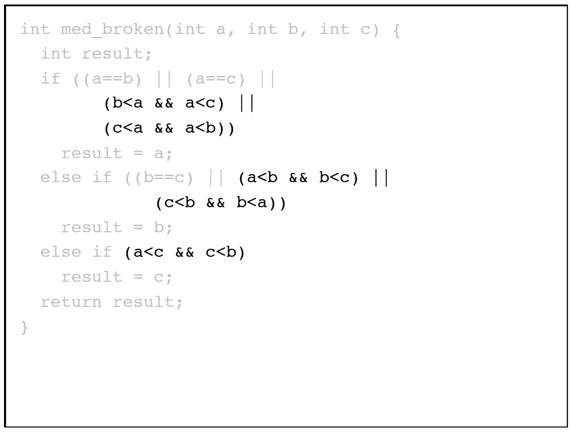
```
int median(int a, int b, int c) {
  int result = 0;
  if ((b<=a && a<=c) ||
      (c<=a && a<=b))
    result = a;
  if ((a<b && b <= c) ||
      (c<=b && b<a))
    result = b;
  if ((a<c && c<b) ||
      (b<c && c<a))
    result = c;
  return result;
}
```

```
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<a))
    result = c;
  return result;
}
```

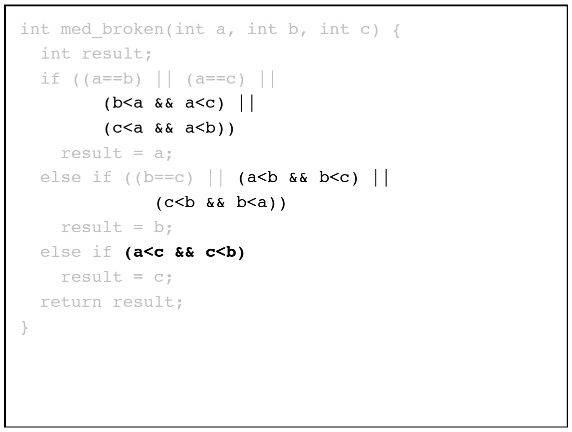
```
int median(int a, int b, int c) {
  int result = 0;
  if ((b<=a && a<=c) ||
      (c<=a && a<=b)
    result = a;
  if ((a<b && b <= c) ||
      (c<=b && b<a))
    result = b;
  if ((a<c && c<b) ||
      (b<c && c<a))
    result = c;
  return result;
}
```

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

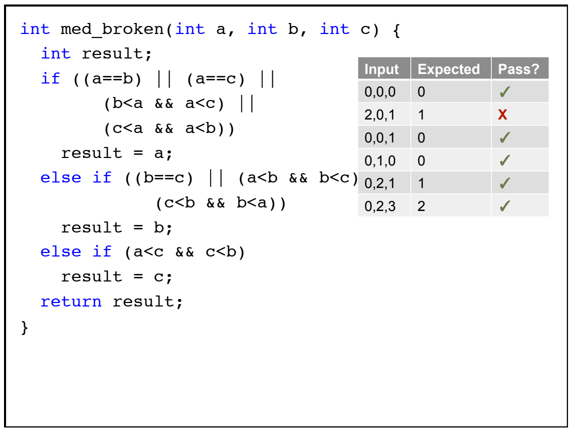
```
int med_broken(int a, int b, int c) {
  int result;
  if ((a==b) || (a==c) ||
        (b<a && a<c) ||
        (c<a && a<b))
    result = a;
  else if ((b==c) || (a<b && b<c) ||
             (c<b && b<a))
    result = b;
  else if (a<c && c<b)
    result = c;
  return result;
}
```



```
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;
}
```

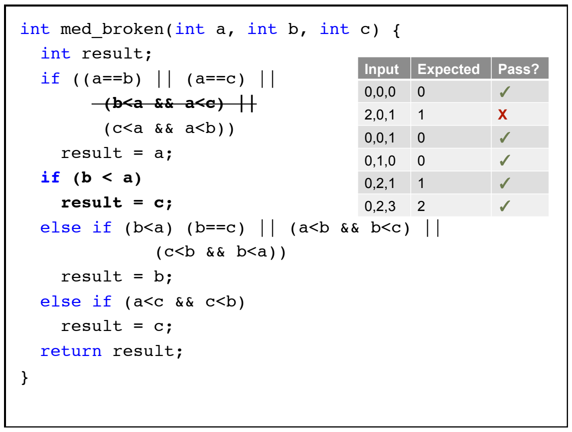


```
int med_broken(int a, int b, int c) {
  int result;
  if ((a==b) || (a==c) ||
        (b<a && a<c) ||
        (c<a && a<b))
    result = a;
  else if ((b==c) || (a<b && b<c) ||
             (c<b && b<a))
    result = b;
  else if (a<c && c<b)
    result = c;
  return result;
}
```



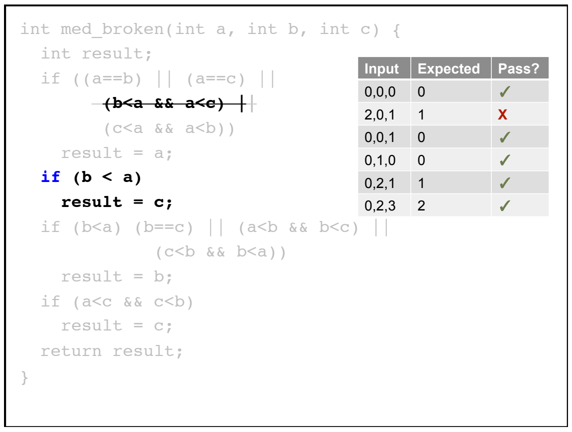
```
int med_broken(int a, int b, int c) {
  int result;
  if ((a==b) || (a==c) ||
        (b<a && a<c) ||
        (c<a && a<b))
    result = a;
  else if ((b==c) || (a<b && b<c) ||
             (c<b && b<a))
    result = b;
  else if (a<c && c<b)
    result = c;
  return result;
}
```

Input	Expected	Pass?
0,0,0	0	✓
2,0,1	1	X
0,0,1	0	✓
0,1,0	0	✓
0,2,1	1	✓
0,2,3	2	✓



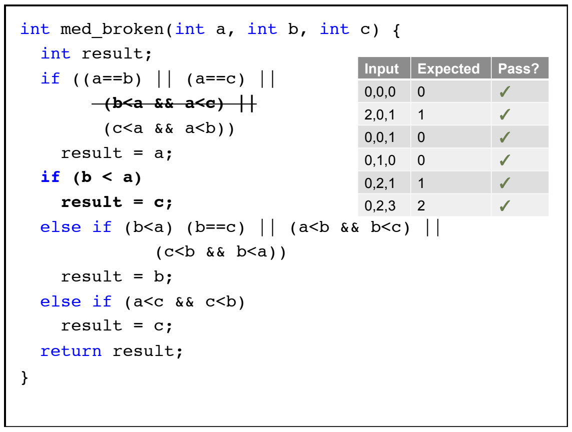
```
int med_broken(int a, int b, int c) {
  int result;
  if ((a==b) || (a==c) ||
       ~~(b<a && a<c) ||~~
        (c<a && a<b))
    result = a;
  if (b < a)
    result = c;
  else if (b<a) (b==c) || (a<b && b<c) ||
             (c<b && b<a))
    result = b;
  else if (a<c && c<b)
    result = c;
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

Input	Expected	Pass?
0,0,0	0	✓
2,0,1	1	X
0,0,1	0	✓
0,1,0	0	✓
0,2,1	1	✓
0,2,3	2	✓



```
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;
  if (b < a)
    result = c;
  if (b<a) (b==c) || (a<b && b<c) ||
             (c<b && b<a))
    result = b;
  if (a<c && c<b)
    result = c;
  return result;
}
```

Input	Expected	Pass?
0,0,0	0	✓
2,0,1	1	X
0,0,1	0	✓
0,1,0	0	✓
0,2,1	1	✓
0,2,3	2	✓



```
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;
  if (b < a)
    result = c;
  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;
}
```

Input	Expected	Pass?
0,0,0	0	✓
2,0,1	1	✓
0,0,1	0	✓
0,1,0	0	✓
0,2,1	1	✓
0,2,3	2	✓

```
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;
  if ((b==c) || (a<b && b<c) ||
             (c<b && b<a))
    result = b;
  if (a<c && c<b)
    result = c;
  return result;
}
```

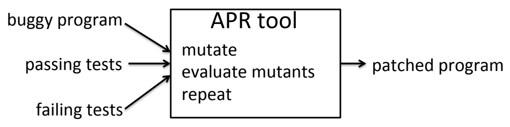
Input	Expected	Pass?
2,6,8	6	✓
2,8,6	6	✓
6,2,8	6	✓
6,8,2	6	✓
8,2,6	6	X
8,6,2	6	✓
9,9,9	9	✓

```
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;
  if (b < a)
    result = c;
  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;
}
```

Input	Expected	Pass?
0,0,0	0	✓
2,0,1	1	✓
0,0,1	0	✓
0,1,0	0	✓
0,2,1	1	✓
0,2,3	2	✓

Input	Expected	Pass?
2,6,8	6	✓
2,8,6	6	✓
6,2,8	6	X
6,8,2	6	✓
8,2,6	6	✓
8,6,2	6	X
9,9,9	9	✓

Potential solution



Use an independent test suite to measure quality of the patch

Focus of prior evaluations

- Most evaluations are interested in whether tools work
  - produce patches
- Some interest in other factors
  - human acceptance of patches [Durieux et al. 2015] [Fry et al. 2012] [Kim et al. 2013]
  - plausibility [Qi et al. 2015]
  - …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

RQ1: What is the base incidence of overfitting?

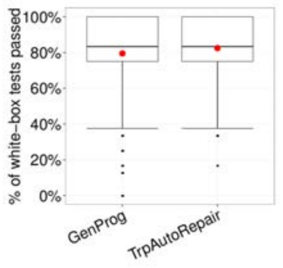
Give a repair tool the buggy program and the black-box test suite, try to repair it, see what fraction of the white-box tests the patches pass.

RQ1: What is the base incidence of overfitting?

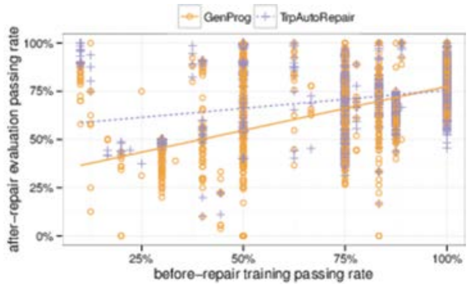
but first, how often can we actually generate patches?

repair tool	patch production %
GenProg	466/778 = 59.9%
TrpAutoRepair	444/778 = 57.1%

RQ1: What is the base incidence of overfitting?

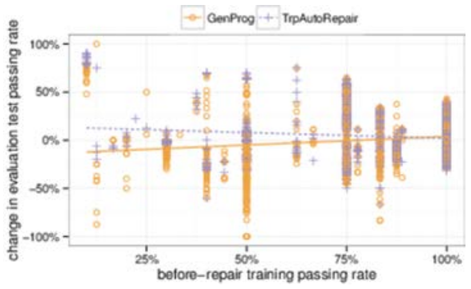


RQ2: What effect do pre-repair test failures have on overfitting?



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

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

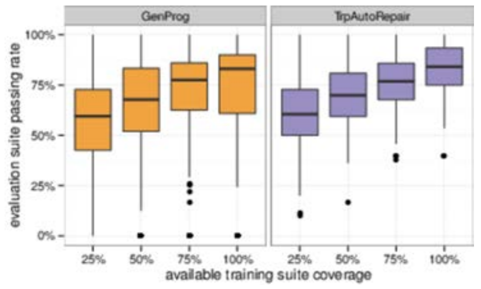


Repair is at best unlikely to improve correctness, at worst likely to worsen it

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

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

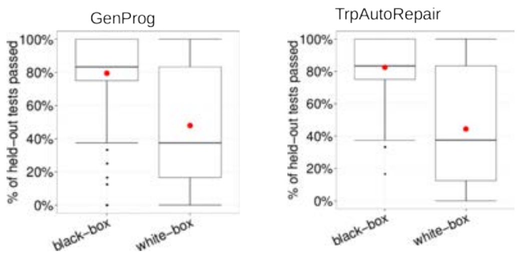


Lower test suite coverage leads to more overfitting

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

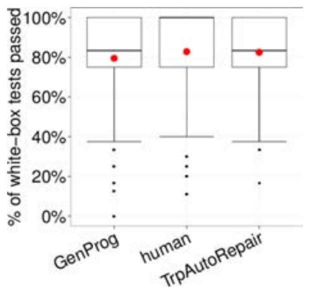
- So far, all experiments have used human-written *black-box* tests to build repairs
- Switch to using KLEE-generated *white-box* tests
- Attempt to repair programs
- If a repair is found, measure correctness of repair
  - this time with *black-box* tests

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

- 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

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.)

Read more about SearchRepair:

http://people.cs.umass.edu/~brun/pubs/pubs/Ke15ase.pdf