More Course Overview: Models, Tests, Bugs, and Symbols

Some logistics

- Homework 1 has been posted
- Due February 16, by 9 AM on moodle
- Requires running linux or "linux"
  - you all have Edlab access, if you want it

Last time

What did we talk about?

Static analysis

- Using the source code to improve a program
- Manual code reviews and inspections
- Automatic inference of properties, proving

Improve the software quality

Dynamic analysis

- Using the program executions to improve the program
- Manual with debuggers, etc.
- Automatic inference over logged behavior
- Does not need source code or even binaries

Improve the software quality

Any questions?

Areas we will cover in this course

- Static analysis
- Dynamic analysis
- Model checking
- Mutation testing
- Bug localization
- Symbolic execution

areas for your projects

As we go over each topic…

- Think whether this sounds interesting
- Think about what kind of a tool you could make that uses this

- You are all programmers:
  think about things you've done while programming that were hard, and how these kinds of analysis might make it easier

Model checking

- I actually meant:
  - Model checking
  - Model inference
  - Model simulation

Model inference

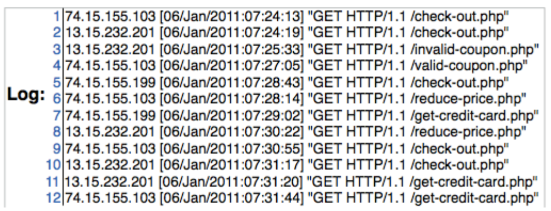
problem:

I have a system (or a log of executions).

I want a small, descriptive model of what the system does.

Model can be used to **understand** the system, **debug**, detect **anomalies**, **document**.

Logs are hard to read



Log:

```
74.15.155.103 [06/Jan/2011:07:24:13] "GET HTTP/1.1 /check-out.php"
13.15.232.201 [06/Jan/2011:07:24:19] "GET HTTP/1.1 /check-out.php"
13.15.232.201 [06/Jan/2011:07:25:33] "GET HTTP/1.1 /invalid-coupon.php"
74.15.155.103 [06/Jan/2011:07:27:05] "GET HTTP/1.1 /valid-coupon.php"
74.15.155.199 [06/Jan/2011:07:28:43] "GET HTTP/1.1 /check-out.php"
74.15.155.103 [06/Jan/2011:07:28:14] "GET HTTP/1.1 /reduce-price.php"
74.15.155.199 [06/Jan/2011:07:29:02] "GET HTTP/1.1 /get-credit-card.php"
13.15.232.201 [06/Jan/2011:07:30:22] "GET HTTP/1.1 /reduce-price.php"
74.15.155.103 [06/Jan/2011:07:30:55] "GET HTTP/1.1 /check-out.php"
13.15.232.201 [06/Jan/2011:07:31:17] "GET HTTP/1.1 /check-out.php"
13.15.232.201 [06/Jan/2011:07:31:20] "GET HTTP/1.1 /get-credit-card.php"
74.15.155.103 [06/Jan/2011:07:31:44] "GET HTTP/1.1 /get-credit-card.php"
```

Model inference

- First, parse out the executions

check-out → valid-coupon → check-out → reduce-price → get-credit-card

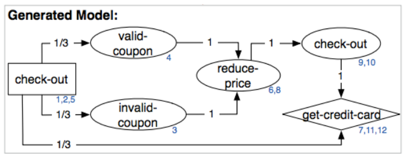
check-out → invalid-coupon → check-out → reduce-price → get-credit-card

check-out → get-credit-card

- …hard to understand

Infer the model

- Magic!



So what's the magic?

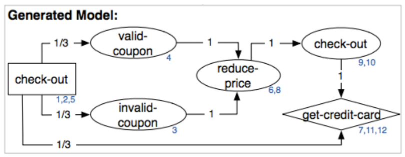
- Lots of ways to do it:
  - Try merging the executions into a small model
  - Mine properties then build a model from the properties alone
  - Use static or dynamic analysis to determine what events can legally take place after others

K-Tails

- let's use k=1 as an example
- merge two states if their name is the same
- (k=2 means merge two states if their name, and all the states to which they have transitions are "the same")
- and so on for larger k

Model checking

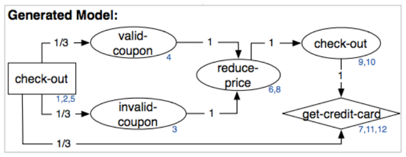
- Given a property and a model, check if the model satisfies that property



- Reduce-price always followed by get-credit-card?

Model simulation

- Given a model, you can generate new executions that have not been observed before!



Mutation testing

- Evaluate the tests
  - not the program!
  - not a type of testing!
  - does not improve a program directly; improves tests!

Mutation

- Take a program
- Create a mutant with one or a few small changes:
  - change a + to a –
  - add/subtract 1 somewhere
  - increment/decrement a loop counter
  - delete a line
  - insert a line from one place in another
- Repeat to create many mutants

Why create mutants?

- Suppose you have a program and a test suite
- All the tests pass
- What does that mean about your program?

1. Program is correct
2. Tests only test parts of the program that are correct and the rest, who knows
3. Tests and program may be written by the same person, using the same *implicit* assumptions

Let's write some tests

```
// returns the factorial of the input n
int factorial (int n) {
  if (n <= 0)
    return 1;
  if (n == 1)
    return 1;
  else
    return n * factorial(n-1);
}
```

OK, so how do we test the tests?

- Run the tests on the main program
- Run the tests on the mutants
  - what if the tests pass?

Mutation testing evaluates the tests

- If a test "kills a mutant" then that's a good test
- If some mutants aren't killed, the test suite is lacking
- Solution: write more tests!

- Is it OK to write more tests until all mutants are killed and then stop?

Consider this mutant

```
// returns the factorial of the input n
int factorial (int n) {
 if (n <= 0)
  return 1;
 if (n == 1)
  return 1;
 else
  return n * factorial(n+1);
}
```

Consider this mutant

```
// returns the factorial of the input n
int factorial (int n) {
  if (n <= 2)
    return 1;
  if (n == 1)
    return 1;
  else
    return n * factorial(n-1);
}
```

Consider this mutant

```
// returns the factorial of the input n
int factorial (int n) {
  if (n == 0)
    return 1;
  if (n == 1)
    return 1;
  else
    return n * factorial(n-1);
}
```

Bug localization

- Narrowing down the most likely place to contain a bug

Failure-inducing input

- This HTML input makes Mozilla crash (segmentation fault).
- Which portion is the failure-inducing one?

```
<td align=left valign=top>
<SELECT NAME="op_sys" MULTIPLE SIZE=7>
<OPTION VALUE="All">All<OPTION VALUE="Windows 3.1">Windows 3.1<OPTION VALUE="Windows 95">Windows 95<OPTION VALUE="Windows
98">Windows 98<OPTION VALUE="Windows ME">Windows ME<OPTION VALUE="Windows 2000">Windows 2000<OPTION VALUE="Windows
NT">Windows NT<OPTION VALUE="Mac System 7">Mac System 7<OPTION VALUE="Mac System 7.5">Mac System 7.5<OPTION VALUE="Mac
System 7.6.1">Mac System 7.6.1<OPTION VALUE="Mac System 8.0">Mac System 8.0<OPTION VALUE="Mac System 8.5">Mac System
8.5<OPTION VALUE="Mac System 8.6">Mac System 8.6<OPTION VALUE="Mac System 9.x">Mac System 9.x<OPTION VALUE="MacOS X">MacOS
X<OPTION VALUE="Linux">Linux<OPTION VALUE="BSDI">BSDI<OPTION VALUE="FreeBSD">FreeBSD<OPTION VALUE="NetBSD">NetBSD<OPTION
VALUE="OpenBSD">OpenBSD<OPTION VALUE="AIX">AIX<OPTION VALUE="BeOS">BeOS<OPTION VALUE="HP-UX">HP-UX<OPTION
VALUE="IRIX">IRIX<OPTION VALUE="Neutrino">Neutrino<OPTION VALUE="OpenVMS">OpenVMS<OPTION VALUE="OS/2">OS/2<OPTION
VALUE="OSF/1">OSF/1<OPTION VALUE="Solaris">Solaris<OPTION VALUE="SunOS">SunOS<OPTION VALUE="other">other</SELECT>
</td>
<td align=left valign=top>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<OPTION VALUE="--">--<OPTION VALUE="P1">P1<OPTION VALUE="P2">P2<OPTION VALUE="P3">P3<OPTION VALUE="P4">P4<OPTION
VALUE="P5">P5</SELECT>
</td>
<td align=left valign=top>
<SELECT NAME="bug_severity" MULTIPLE SIZE=7>
<OPTION VALUE="blocker">blocker<OPTION VALUE="critical">critical<OPTION VALUE="major">major<OPTION
VALUE="normal">normal<OPTION VALUE="minor">minor<OPTION VALUE="trivial">trivial<OPTION VALUE="enhancement">enhancement</SELECT>
</tr>
</table>
```

Thank you to Curino and Giusti for contributing to these slides

Delta Debugging: Try half the input

- Will the program still crash?

```
<td align=left valign=top>
<SELECT NAME="op_sys" MULTIPLE SIZE=7>
<OPTION VALUE="All">All<OPTION VALUE="Windows 3.1">Windows 3.1<OPTION VALUE="Windows 95">Windows 95<OPTION VALUE="Windows
98">Windows 98<OPTION VALUE="Windows ME">Windows ME<OPTION VALUE="Windows 2000">Windows 2000<OPTION VALUE="Windows
NT">Windows NT<OPTION VALUE="Mac System 7">Mac System 7<OPTION VALUE="Mac System 7.5">Mac System 7.5<OPTION VALUE="Mac
System 7.6.1">Mac System 7.6.1<OPTION VALUE="Mac System 8.0">Mac System 8.0<OPTION VALUE="Mac System 8.5">Mac System
8.5<OPTION VALUE="Mac System 8.6">Mac System 8.6<OPTION VALUE="Mac System 9.x">Mac System 9.x<OPTION VALUE="MacOS X">MacOS
X<OPTION VALUE="Linux">Linux<OPTION VALUE="BSDI">BSDI<OPTION VALUE="FreeBSD">FreeBSD<OPTION VALUE="NetBSD">NetBSD<OPTION
VALUE="OpenBSD">OpenBSD<OPTION VALUE="AIX">AIX<OPTION VALUE="BeOS">BeOS<OPTION VALUE="HP-UX">HP-UX<OPTION
VALUE="IRIX">IRIX<OPTION VALUE="Neutrino">Neutrino<OPTION VALUE="OpenVMS">OpenVMS<OPTION VALUE="OS/2">OS/2<OPTION
VALUE="OSF/1">OSF/1<OPTION VALUE="Solaris">Solaris<OPTION VALUE="SunOS">SunOS<OPTION VALUE="other">other</SELECT>
</td>
~~<td align=left valign=top>~~
~~<SELECT NAME="priority" MULTIPLE SIZE=7>~~
~~<OPTION VALUE="--">--<OPTION VALUE="P1">P1<OPTION VALUE="P2">P2<OPTION VALUE="P3">P3<OPTION VALUE="P4">P4<OPTION~~
~~VALUE="P5">P5</SELECT>~~
~~</td>~~
~~<td align=left valign=top>~~
~~<SELECT NAME="bug_severity" MULTIPLE SIZE=7>~~
~~<OPTION VALUE="blocker">blocker<OPTION VALUE="critical">critical<OPTION VALUE="major">major<OPTION~~
~~VALUE="normal">normal<OPTION VALUE="minor">minor<OPTION VALUE="trivial">trivial<OPTION VALUE="enhancement">enhancement</SELECT>~~
~~</tr>~~
</table>
```

Thank you to Curino and Giusti for contributing to these slides

Minimizing via binary search

- 57 test to simplify the 896 line HTML input to the “<SELECT>” tag that causes the crash
- Each character is relevant (as shown from line 20 to 26)
- Only removes deltas from the failing test

```
1  <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✗
2  <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
3  <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
4  <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
5  <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✗
6  <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✗
7  <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
8  <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
9  <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
10 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✗
11 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
12 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
13 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
14 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
15 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
16 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✗
17 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✗
18 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✗
19 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
20 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
21 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
22 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
23 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
24 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
25 <SELECT NAME="priority"␣MULTIPLE␣SIZE=7> ✔
26 <SELECT NAME="priority"␣MULTIPLE␣SIZE=7> ✗
```

Impact analysis

- Run the code on passing test cases
- Run the code on failing test cases
- Keep track of which lines execute

- Lines that executes only on passing test cases are OK. So are lines that execute on both.
- Lines that only execute on failing test cases are suspicious.

What else can you do to localize a bug?

Regressions: suppose a test used to pass and now fails.

- consider the latest changes
- do delta debugging on the changes

Can we automatically fix bugs?

Take a program that passes most test cases and fails one or two, and tweak it

- write (tweak) a very similar program (with minimal change) that passes all the test [see Weimer et al., Automatically Finding Patches Using Genetic Programming, ICSE 2009]

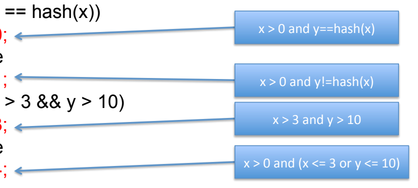
localizing and auto-fixing: great project areas

Symbolic execution

- "Think" about the code, rather than execute it, but execute it anyway. But don't use numbers. Just think about the numbers.

- Clear, right?

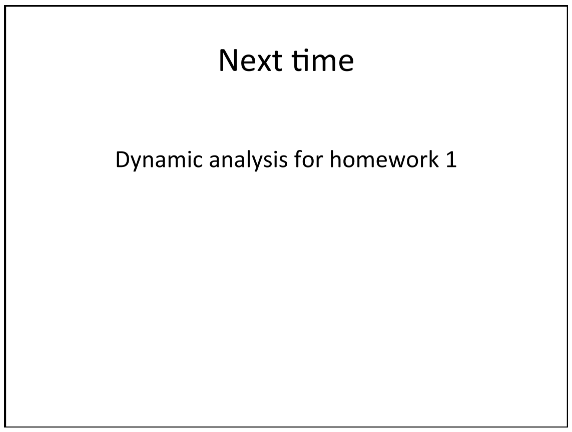
```
void test(int x, int y) {
   if (x > 0) {
     if (y == hash(x))
       S0;
     else
       S1;
     if (x > 3 && y > 10)
       S3;
     else
       S4;
   }
}
```



Thank you to Willem Visser for contributing to these slides

Why symbolic execution?

- A different way to reasoning about the code
- Can determine what parts are reachable and under what conditions
- Can be compared to developers' expectations about those conditions
- Can be used to document
  - For example, "this method can only be called if x>0" or "this method throws an exception is pts == null"



Next time

Dynamic analysis for homework 1