Boolean Satisfiability and Project Discussion

Announcements

- Lab5 due today
  - Fix is to use java version 11.0.11
  - Lab instructions updated

- *Almost* everything is graded!
  - Questions? Complaints? File a regrade request on Gradescope

- Please submit group suggestions by EOD

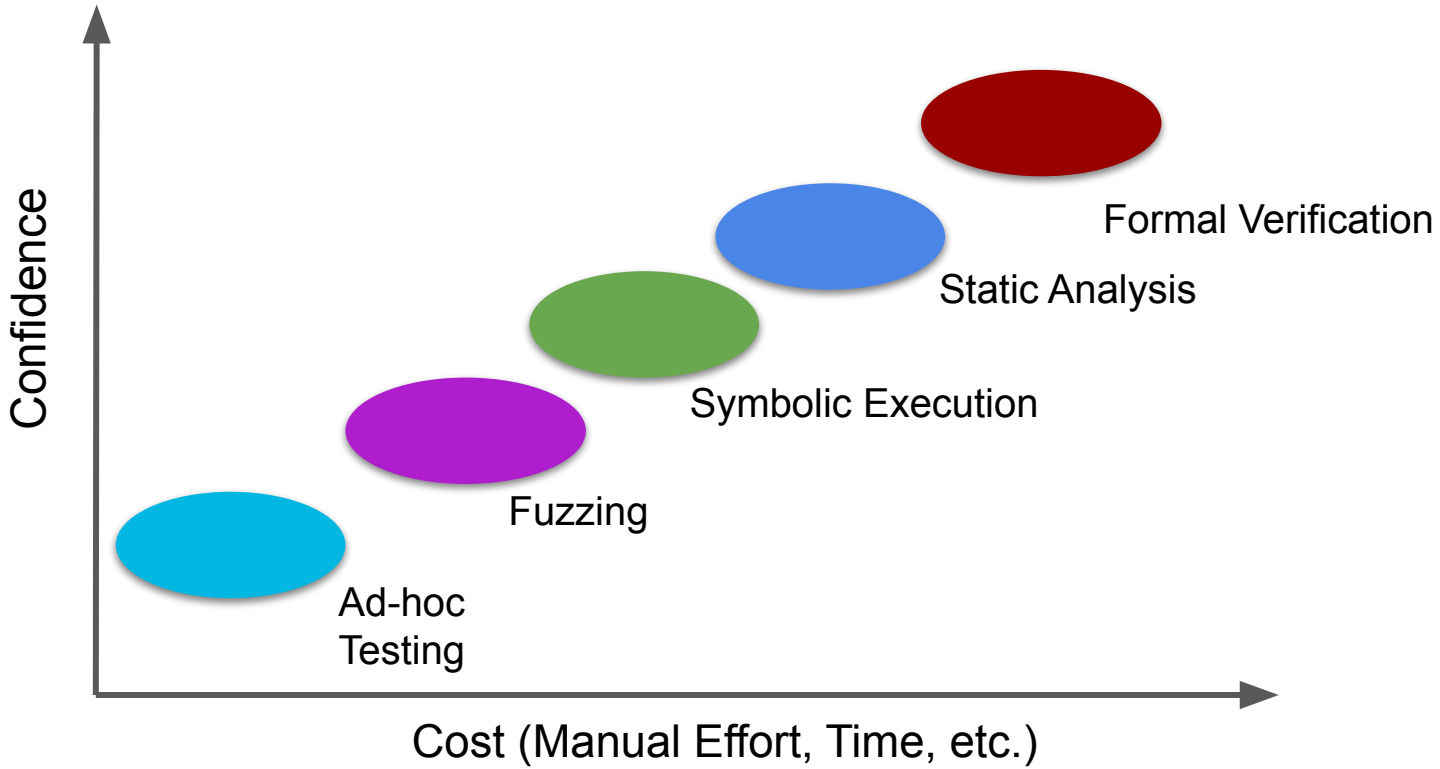
Feedback on Assignments

1. Use correct terminology
   a. “Runs”, “compiles”, “killed”, “survived”
   b. If you don’t know what it means, review slides!

2. Each test contains a prefix and assertion
   a. Consider both when answering!
   b. Description of your bug. Does the trigger test expose a safety or functional property violation?
   c. Randoop Results for your bug. Include the number of tests generated, if any trigger the bug, and the line and branch coverage.
   d. Discuss differences between the Randoop generated tests and the developer written test.

3. Think about what you think *should* happen before running experiments

Landscape of Program Analysis Techniques



Roadmap for rest of the course

Project: Testing

Homework 3: Verification

4 more labs:

Labs 6: fuzzing

Lab 7: formal verification

Lab 8: symbolic execution

Lab 10: static analysis

Syllabus

- Homeworks: 20%
- Labs: 35%
- Project: 40%
- Participation: 5%

Overview

- AFL review
- Project Overview
- Boolean Satisfiability
  - DPLL algorithm

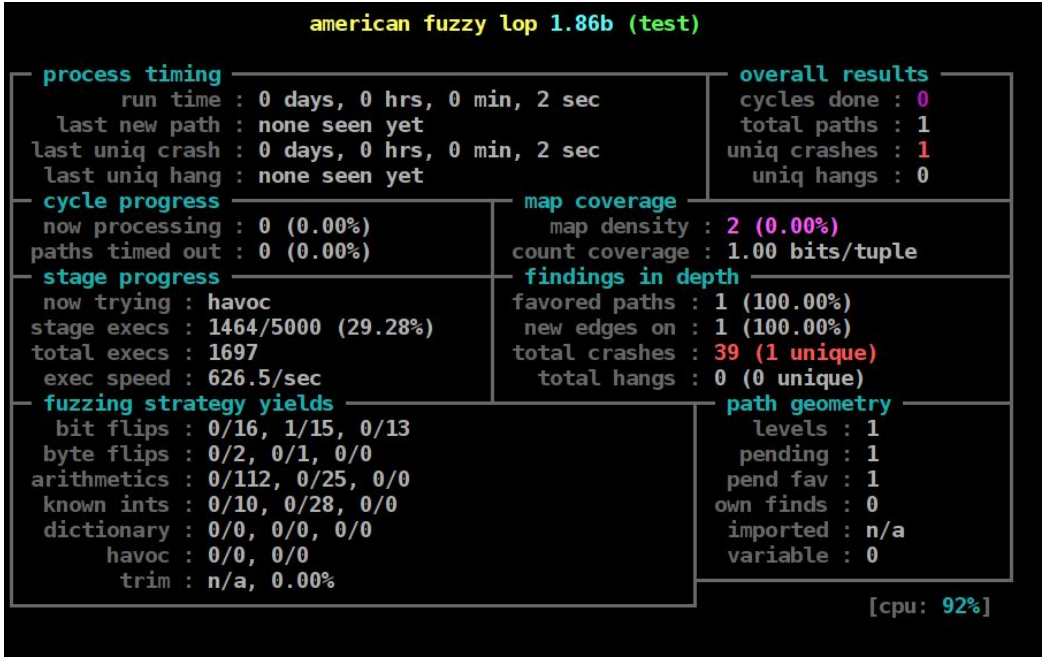
AFL Review

1. How is fuzzing C programs different than randomized testing of Java programs?

2. Generations of fuzzing:
   a. Truly random
   b. Seeds
   c. Feedback guided

3. Constant pools

AFL - Lab Today



Project Discussion

This Course

Learning Objectives:

1. Strong understanding of program analysis landscape
2. Hands on experience with program analysis tools
3. **Ability to perform mature research**

Project Overview

In this project you will run state of the art automated testing tools to attempt to find bugs in a benchmark. You will record the current state of the art as a baseline and attempt to approve upon it. **Test generation is an active and fruitful area of computer science research. This is your opportunity to contribute to it!**

Each group will be assigned a project from the Defects4J benchmark. Your goal will be to study the current state of test generation tools on finding that bug. You will closely analyze each bug and generated tests.

Lab 5 will be helpful for this project!

Project Workflow:

1. Checkout the bug and inspect it
    a. Defects4j dissection is useful for this!
    b. https://program-repair.org/defects4j-dissection/#!/bug/Chart/1

2. Inspect the developer written test
    a. Defects4j dissection is also useful for this!
    b. Understand how it triggers the bug

3. Generate EvoSuite tests
    a. Inspect if they execute the bug
    b. What will the assertions look like?
    c. You will do 3 trials of this

4. If your EvoSuite prefix executed the bug, generate assertions

EvoSuite Review

Algorithm 1 Evolutionary Algorithm

```
1: Initialize population P with random solutions
2: Evaluate fitness of each individual in P
3: while search budget not exhausted do
4:     Select individuals from P based on fitness
5:     Apply crossover to produce new offspring
6:     Apply mutation to offspring
7:     Evaluate fitness of offspring
8:     Add offspring to the population P
9: end while
```

EvoSuite Review

1. How does it initialize the population?
   a. iterative insertions of random statements.

2. How does it avoid generation of invalid test prefixes
   a. leverages typing rules and generates random statements by selecting from a pool.
   b. This pool, is called the **Test Cluster**
   c. Chooses from Test Cluster to insert up to `n` statements
      i. `n` is also randomly generated up to a bound

I wonder if setPlot was in the test cluster for Chart 1...

Test Cluster

Contains three sets

1. Test Methods - all public methods on the target class and superclasses
2. Generators - all constructors and factory methods for relevant types
3. Modifiers - impure methods on the target class

Building blocks of our tests!

EvoSuite Review

Algorithm 1 Evolutionary Algorithm

```
1: Initialize population P with random solutions
2: Evaluate fitness of each individual in P
3: while search budget not exhausted do
4:     Select individuals from P based on fitness
5:     Apply crossover to produce new offspring
6:     Apply mutation to offspring
7:     Evaluate fitness of offspring
8:     Add offspring to the population P
9: end while
```

Let’s explore Chart 1 more....

Can we print the test cluster?

What else would be useful?

1. Initial Population
2. At each iteration:
    a. Parents selected for crossover
    b. The offspring

To print this, use export my_evo_debug=1 before running gen_tests.pl

Assertions

Some of you may get assigned the bug with a prefix that executes the bug!

For Chart 1 example, what would the test look like?

Assertion Generation

To generate assertions, you will use the EditAS2 approach.

combination of the techniques you implemented in HW2.

First, it uses an information retrieval approach based on Jaccard similarity of the prefix. It retrieves the assertion from the most similar prefix in the corpus.

Then, it uses a neural model to perform an edit on that retrieved assertion.

This approach requires access a GPU so you cannot run it directly. Instead, if you think your EvoSuite prefix executes the bug, ask the professor to generate the assertions for you. Perform an analysis on the assertions.

Report

Your report will be written in latex. I've linked a template which you will need to fill in

https://www.overleaf.com/project/67cf01736701be3f79f3d17f

Latex Cheat Sheet

\textbf{} Bold

\texttt{} Code format

\ref{} to refer to a figure, section, or table

\label{} to label a figure (and refer to it later)

Entries in a table are delimited with an &

Use a backslash before special symbols (like %)

Boolean Satisfiability

Boolean logic review

A boolean expression is built from:

1. Variables (can evaluate to TRUE/FALSE)
2. Operators
    a. AND ⋀
    b. OR V
    c. NEGATION ~ / ! / ¬

A formula is said to be *satisfiable* if it can be made TRUE by assigning logical values to its variables.

Boolean Satisfiability Problem

Asks whether there exists an *interpretation* that *satisfies* a given Boolean formula

- UNSAT: `p ∧ !p`
- SAT: `p ∧ !q`

`Interpretation: p -> TRUE, q -> FALSE`

Examples:

SAT or UNSAT? If SAT, give the interpretation

1. $(A \vee B) \wedge (\neg A \vee C)$
2. $(A \vee B) \wedge (\neg A \vee \neg B) \wedge (A \vee \neg B) \wedge (\neg A \vee B)$
3. $(A \vee B) \wedge (\neg B \vee C) \wedge (\neg A \vee \neg C)$
4. $(A \wedge B) \wedge (\neg A \vee C) \wedge (\neg B \vee \neg C)$

What does this have to do with software analysis?

```
if (y > 3 * x + 7) {

    if (x + y - z % 2 == 0) {

        if (z >= x) {

            int res = z / (z - x);

        }

    }

}
```

When will this program crash?

```
z - x == 0  AND
z >= x AND
x + y -z % 2 == 0 AND
y > 3 * x + 7
```

We can encode path constraints as boolean satisfiability problems!

Solving Boolean Satisfiability

Can we write a program to answer SAT / UNSAT?

SAT is the first problem that was proven to be NP-Complete

We can try every possible combination, but it will be slow.

Conjunctive Normal Form (CNF)

Each *clause* is separated by an AND operator

$C1 \wedge \ldots \wedge Cn$

And each *clause* and is of the form:

$Li \vee \ldots \vee Lm$

where each Li is called a literal and is either a boolean variable or a negation of the variable

Conjunctive Normal Form (CNF)

Example 6.A The following is a CNF formula with two clauses, each of which contains two literals:

$$(p \vee \neg r) \wedge (\neg p \vee q)$$

The following formula is *not* in CNF:

$$(p \wedge q) \vee (\neg r)$$

■

DPLL Algorithm

Given a boolean formula in CNF form, DPLL decides UNSAT/SAT and gives an interpretation if its SAT

Alternates between two phases:

1. Deduction
    a. Tries to simplify the formula using the laws of logic
2. Search
    a. Searches for an interpretation

DPLL - Deduction Phase

Boolean Constant Propagation

$$(\ell) \wedge C_2 \wedge \cdots C_n$$

Suppose the first clause consists of a single literal (We call this a *unit clause*).

Any interpretation of the formula must assign ℓ to TRUE.

Simplify the formula by substituting TRUE for all ℓs

Boolean Constant Propagation (BCP) Example

$$(p) \wedge (\neg p \vee r) \wedge (\neg r \vee q)$$

p must be TRUE

$$\begin{aligned} & (\text{true}) \wedge (\neg \text{true} \vee r) \wedge (\neg r \vee q) \\ \equiv\ & (r) \wedge (\neg r \vee q) \end{aligned}$$

r must be TRUE

$$\begin{aligned} & (\text{true}) \wedge (\neg \text{true} \vee q) \\ \equiv\ & (q) \end{aligned}$$

$$\{p \mapsto \text{true}, q \mapsto \text{true}, r \mapsto \text{true}\}$$

Boolean Constant Propagation (BCP) Example 2

$(x) \wedge (p \vee r) \wedge (\neg p \vee q) \wedge (\neg q \vee \neg r)$

Deduction + Search

Once the simplified formula no longer contains unit clauses, we have to go back to the brute force approach....

Iteratively chooses variables and tries to replace them with TRUE or FALSE calling DPLL recursively in the resulting formula

Deduction + Search Example

(x) ∧ (p ∨ r) ∧ (¬p ∨ q) ∧ (¬q ∨ ¬r)

First level of recursion: p -> TRUE

(q) ∧ (¬q ∨ ¬r)

DPLL

Algorithm 1: DPLL

Data: A formula F in CNF form
Result: $I \models F$ or UNSAT

▷ Boolean constant propagation (BCP)
while *there is a unit clause* (ℓ) *in* F **do**
  Let F be $F[\ell \mapsto \text{true}]$

if F *is* true **then return** SAT

▷ Search
for *every variable* p *in* F **do**
  If DPLL$(F[p \mapsto \text{true}])$ is SAT **then return** SAT
  If DPLL$(F[p \mapsto \text{false}])$ is SAT **then return** SAT
return UNSAT
▷ The model I that is returned by DPLL when the input is SAT is maintained implicitly in the sequence of assignments to variables (of the form $[l \mapsto \cdot]$ and $[p \mapsto \cdot]$)

Summary

- Project
  - Send me group suggestions ASAP
  - Groups will be assigned tomorrow
  - Part one due March 31st
  - Start early!!

- Boolean Satisfiability
  - Path conditions can be modelled as SAT formulae
  - Solved with DPLL
  - We can use this to find bugs!

- Lab 6 today
  - Due Sunday March 23rd