Static analysis

UW CSE P 504
Today

- Recap: statistical fault localization
- Static Analysis
  - Motivation
  - Examples
  - Intro to Abstract Interpretation
Recap: statistical fault localization
Recap: statistical fault localization

Program

```java
double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    for(int i=0; i<n; ++i) {
        sum -= nums[i];
    }
    return sum / n;
}
```

Fault localization technique

Statement ranking

```java
double avg(double[] nums) {
    int n = nums.length;
    double sum = 0;
    for(int i=0; i<n; ++i) {
        sum -= nums[i];
    }
    return sum / n;
}
```

Test suite

- Passing tests
- Failing tests

Most suspicious

Least suspicious
Recap: statistical fault localization

Jones et al., Visualization of test information to assist fault localization, ICSE'02

GZoltar
Recap: statistical fault localization

Developer in the loop

● Which granularity is most useful?
  ○ file level
  ○ method level
  ○ statement level

● What context do you need to reason about?
  ○ a file
  ○ a method
  ○ a statement
Recap: statistical fault localization

Developer in the loop

● Which granularity is most useful?
  ○ file level
  ○ method level
  ○ statement level

● What context do you need to reason about?
  ○ a file
  ○ a method
  ○ a statement

● Processing FL output
  ○ How useful is color coding (heatmap) vs. ranking?
  ○ How realistic is “sequential debugging”?
Static Analysis
Static vs. dynamic analysis

Dynamic analysis

- Reason about the program based on some program executions.
- Observe concrete behavior at run time.
- Improve confidence in correctness.
- Unsound* but precise.

* Some static analyses are unsound; dynamic analyses can be sound.
Static vs. dynamic analysis

Dynamic analysis

- Reason about the program based on **some** program *executions*.
- Observe **concrete behavior** at run time.
- Improve confidence in correctness.
- **Unsound*** but *precise*.

```plaintext
[y:=2, x:=2]
y = x++

```
Static vs. dynamic analysis

Dynamic analysis

- Reason about the program based on **some** program *executions*.
- Observe **concrete behavior** at run time.
- Improve confidence in correctness.
- **Unsound** but **precise**.

*Some static analyses are unsound; dynamic analyses can be sound.*
Static vs. dynamic analysis

Dynamic analysis

- Reason about the program based on **some** program executions.
- Observe **concrete behavior** at run time.
- Improve confidence in correctness.
- **Unsound*** but precise.

Static analysis

- Reason about the program **without executing** it.
- Build an **abstraction of run-time states**.
- Reason over **abstract domain**.
- **Prove a property** of the program.
- **Sound*** but imprecise.

* Some static analyses are unsound; dynamic analyses can be sound.
Static vs. dynamic analysis

Dynamic analysis

- Reason about the program based on some program executions.
- Observe concrete behavior at run time.
- Improve confidence in correctness.
- Unsound* but precise.

Static analysis

- Reason about the program without executing it.
- Build an abstraction of run-time states.
- Reason over abstract domain.
- Prove a property of the program.
- Sound* but imprecise.

* Some static analyses are unsound; dynamic analyses can be sound.
Static vs. dynamic analysis

Dynamic analysis
- Reason about the program based on some program executions.
- Observe concrete behavior at run time.
- Improve confidence in correctness.
- Unsound* but precise.

Static analysis
- Reason about the program without executing it.
- Build an abstraction of run-time states.
- Reason over abstract domain.
- Prove a property of the program.
- Sound* but imprecise.

* Some static analyses are unsound; dynamic analyses can be sound.
Static vs. dynamic analysis

Dynamic analysis
- Reason about the program based on some program executions.
- Observe concrete behavior at run time.
- Improve confidence in correctness.
- Unsound* but precise.

Static analysis
- Reason about the program without executing it.
- Build an abstraction of run-time states.
- Reason over abstract domain.
- Prove a property of the program.
- Sound* but imprecise.

* Some static analyses are unsound; dynamic analyses can be sound.
Static vs. dynamic analysis

Dynamic analysis

- Reason about the program based on some program executions.
- Observe concrete behavior at run time.
- Improve confidence in correctness.
- Unsound* but precise.

Static analysis

- Reason about the program without executing it.
- Build an abstraction of run-time states.
- Reason over abstract domain.
- Prove a property of the program.
- Sound* but imprecise.

* Some static analyses are unsound; dynamic analyses can be sound.
Static vs. dynamic analysis

Dynamic analysis
- Reason about the program based on **some** program executions.
- Observe **concrete behavior** at run time.
- Improve confidence in correctness.
- Unsound* but precise.

Static analysis
- Reason about the program **without executing** it.
- Build an **abstraction of run-time states**.
- Reason over **abstract domain**.
- Prove a property of the program.
- Sound* but imprecise.

* Some static analyses are unsound; dynamic analyses can be sound.
<table>
<thead>
<tr>
<th>Static vs. dynamic analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dynamic analysis</strong></td>
</tr>
<tr>
<td>● Concrete domain</td>
</tr>
<tr>
<td>● Precise but unsound</td>
</tr>
<tr>
<td>● Slow if exhaustive</td>
</tr>
</tbody>
</table>
Static vs. dynamic analysis

Dynamic analysis

- Concrete domain
- Precise but unsound
- Slow if exhaustive

Static analysis

- Abstract domain
- Sound but imprecise
- Slow if precise

Concrete domain

```c
int getValue(int a) {
    return (a % 3) * 2;
}
int x = getValue(7);
```

Abstract domain

What possible value(s) does getValue() return?
Static vs. dynamic analysis

Dynamic analysis
- Concrete domain
- Precise but unsound
- Slow if exhaustive

Static analysis
- Abstract domain
- Sound but imprecise
- Slow if precise

Concrete domain
0, 2, 4, 6, 8, 10, ...

Abstract domain
even, odd, anything

What possible value(s) does getValue() return?
Terminology and important concepts

Recall the following terms:
1. Precision vs. Recall (and FP/FN/TP/TN)
2. Soundness vs. Completeness
3. Accuracy vs. Precision

<table>
<thead>
<tr>
<th>Ground Truth</th>
<th>Analysis result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>Pos</td>
</tr>
<tr>
<td></td>
<td>Neg</td>
</tr>
<tr>
<td>Neg</td>
<td></td>
</tr>
</tbody>
</table>

Concrete domain vs. Abstract domain
0, 2, 4, 6, 8, 10, ...

even, odd, anything

```java
int getValue(int a) {
    return (a % 3) * 2;
}
int x = getValue(7);
```
# Terminology and important concepts

1. Precision vs. Recall (and FP/FN/TP/TN)

<table>
<thead>
<tr>
<th>Ground Truth</th>
<th>Analysis result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>Pos</td>
</tr>
<tr>
<td>Neg</td>
<td>Neg</td>
</tr>
</tbody>
</table>

- **Ground Truth**: The true state of the system.
- **Analysis result**: The system's output.
- **Pos**: Positive result.
- **Neg**: Negative result.
- **TP**: True Positive.
- **TN**: True Negative.
- **FP**: False Positive.
- **FN**: False Negative.
Terminology and important concepts

1. Precision vs. Recall (and FP/FN/TP/TN)

```
+---+---+
| TP | FN |
+---+---+
| FP | TN |
```

- **TP** (True Positive): Correctly identified as positive.
- **FN** (False Negative): Missed positive cases.
- **FP** (False Positive): Incorrectly identified as positive.
- **TN** (True Negative): Correctly identified as negative.

Ground Truth vs. Analysis Result: 
- **Pos** (Positive): Identified as positive.
- **Neg** (Negative): Identified as negative.
Terminology and important concepts

1. Precision vs. Recall (and FP/FN/TP/TN)

Precision: \[
\frac{|TP|}{|TP| + |FP|}
\]

Recall: \[
\frac{|TP|}{|TP| + |FN|}
\]
Terminology and important concepts

1. Precision vs. Recall (and FP/FN/TP/TN)
2. Soundness vs. Completeness

![Confusion Matrix]

- TP (True Positive)
- FP (False Positive)
- FN (False Negative)
- TN (True Negative)
Terminology and important concepts

1. Precision vs. Recall (and FP/FN/TP/TN)
2. Soundness vs. Completeness

<table>
<thead>
<tr>
<th>Ground Truth</th>
<th>Analysis result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos</td>
<td>Neg</td>
</tr>
<tr>
<td>TP</td>
<td>FN</td>
</tr>
<tr>
<td>FP</td>
<td>TN</td>
</tr>
</tbody>
</table>

Soundness: no FNs
Completeness: no FPs
Terminology and important concepts

1. Precision vs. Recall (and FP/FN/TP/TN)
2. Soundness vs. Completeness
3. Accuracy vs. Precision

```java
int getValue(int a) {
    return (a % 3) * 2;
}
```

```
int x = getValue(7);
```

Concrete domain

0, 2, 4, 6, 8, 10, ...

Abstract domain

even, odd, anything
Terminology and important concepts

1. Precision vs. Recall (and FP/FN/TP/TN)
2. Soundness vs. Completeness
3. Accuracy vs. Precision

int getValue(int a) {
    return (a % 3) * 2;
}
int x = getValue(7);
Terminology and important concepts

1. Precision vs. Recall (and FP/FN/TP/TN)
2. Soundness vs. Completeness
3. Accuracy vs. Precision

An analysis/measure can be precise and inaccurate at the same time!
Static analysis: applications

Compiler checks and optimizations

- Liveness analysis (register reallocation)
- Reachability analysis (dead code elimination)
- Code motion (while(cond){x = comp(); ...})
Static analysis: code examples

Liveness

```java
public class Liveness {
    public void liveness() {
        int a;
        if (alwaysTrue()) {
            a = 1;
        }
        System.out.println(a);
    }
}
```

Reachability

```java
public void deadCode() {
    return;
    System.out.println("Here!");
}
```
Common static analyses

Live examples
- Definite assignment
- Dead code
- Linter warnings
Challenges to adopting static analysis

- Not integrated into the developer’s workflow.
- Reported issues are not actionable.
- Developers do not trust the results (FPs).
- Fixing an issue is too expensive or risky.
- Developers do not understand the reported issues.
- Issues theoretically possible but don’t manifest in practice.

“Produce less than 10% effective false positives. Developers should feel the check is pointing out an actual issue at least 90% of the time.”

“Lessons from Building Static Analysis Tools at Google”, CACM 2018
Effective false positive

- We consider an issue to be an “effective false positive” if developers did not take positive action after seeing the issue.

- If an analysis incorrectly reports an issue, but developers make the fix anyway to improve code readability or maintainability, that is not an effective false positive.

- If an analysis reports an actual fault, but the developer did not understand the fault and therefore took no action, that is an effective false positive.
Effective false positive: example (mutation testing)

```cpp
text
int RunMe(int a, int b) {
    if (a == b || b == 1) {
        // Changing this 1 line to
        if (a != b || b == 1) {
            // does not cause any test exercising them to fail.
            // Consider adding test cases that fail when the code is mutated to
            // ensure those bugs would be caught.
            // Mutants ran because goranpetrovic is whitelisted
        }
    }
}
```

Petrovic et al., ICSTW’18
Effective false positive: discussion

- **We consider an issue to be an “effective false positive” if developers did not take positive action after seeing the issue.**
- **If an analysis incorrectly reports an issue, but developers make the fix anyway to improve code readability or maintainability, that is not an effective false positive.**
- **If an analysis reports an actual fault, but the developer did not understand the fault and therefore took no action, that is an effective false positive.**

Do you agree with this characterization? Is effective false positive rate an adequate measure?
Abstract Interpretation
Properties of an ideal program analysis

- Soundness
- Completeness
- Termination

```java
int x = 0;
while (!isDone()) {
    x = x + 1;
}
...```

A B C
Properties of an ideal program analysis

- Soundness
- Completeness
- Termination

Abstract interpretation sacrifices completeness (precision)
A first example

Program

```c
x = 0;
y = read_even();
x = y + 1;
y = 2 * x;
x = y - 2;
y = x / 2;
```

Are all statements necessary?
A first example: SSA form

Program
\[
\begin{align*}
    x &= 0; \\
    y &= \text{read\_even}(); \\
    x &= y + 1; \\
    y &= 2 \times x; \\
    x &= y - 2; \\
    y &= x \div 2;
\end{align*}
\]

SSA form
\[
\begin{align*}
    x_1 &= 0; \\
    y_1 &= \text{read\_even}(); \\
    x_2 &= y_1 + 1; \\
    y_2 &= 2 \times x_2; \\
    x_3 &= y_2 - 2; \\
    y_3 &= x_3 \div 2;
\end{align*}
\]

\(X_1\) is never read.
A first example: one concrete execution

Program

```plaintext
x = 0;
y = read_even();
x = y + 1;
y = 2 * x;
x = y - 2;
y = x / 2;
```

Concrete execution

```plaintext
{x=0;  y=undef}
{x=0;  y=8}
{x=9;  y=8}
{x=9;  y=18}
{x=16; y=18}
{x=16; y=8}
```
A first example: symbolic reasoning

Program

```plaintext
x = 0;
y = read_even();
x = y + 1;
y = 2 * x;
x = y - 2;
y = x / 2;
```

SSA form

```plaintext
x_1 = 0;
y_1 = read_even();
x_2 = y_1 + 1;
y_2 = 2 * x_2;
x_3 = y_2 - 2;
y_3 = x_3 / 2;
```

What facts can you deduce about y and x after execution?
A first example: symbolic reasoning

Symbolic reasoning shows simplification potential.
A first example: abstract interpretation

Program

x = 0;
y = read_even();
x = y + 1;
y = 2 * x;
x = y - 2;
y = x / 2;

Abstract domain (even, odd, unk)

{x=e; y=e}

{x=??; y=??}

What’s the abstract type of x and y after (abstract) execution?
A first example: “abstract execution”

Program

x = 0;
y = read_even();
x = y + 1;
y = 2 * x;
x = y - 2;
y = x / 2;

What’s the abstract type of x and y after (abstract) execution?
A first example: “abstract execution”

What’s the abstract type of $x$ and $y$ after (abstract) execution?
A first example: abstract interpretation

Program

```plaintext
x = 0;
y = read_even();
x = y + 1;
y = 2 * x;
x = y - 2;
y = x / 2;
```

Abstract domain (**even**, **odd**, **unk**)

{\(x=\text{e}\); \(y=\text{e}\)}

{\(x=??\); \(y=??\)}

What’s the abstract type of \(x\) and \(y\) after (abstract) execution?
A first example: abstract interpretation

Program

```plaintext
x = 0;
y = read_even();
x = y + 1;
y = 2 * x;
x = y - 2;
y = x / 2;
```

Abstract domain (even, odd, unk)

1. \{x=e; y=e\}  
2. \{x=e; y=u\}

Convince yourself that this is true.
A first example: abstract interpretation

Program

\begin{equation}
\begin{align*}
\text{x} &= 0; \\
\text{y} &= \text{read_even}(); \\
\text{x} &= \text{y} + 1; \\
\text{y} &= 2 \times \text{x}; \\
\text{x} &= \text{y} - 2; \\
\text{y} &= \text{x} / 2;
\end{align*}
\end{equation}

Abstract domain \((\text{even, odd, unk})\)

\begin{equation}
\begin{align*}
\{\text{x}=\text{e}; \ \text{y}=\text{e}\} \\
\{\text{x}=\text{o}; \ \text{y}=\text{e}\} \\
\{\text{x}=\text{o}; \ \text{y}=\text{e}\} \\
\{\text{x}=\text{e}; \ \text{y}=\text{e}\} \\
\{\text{x}=\text{e}; \ \text{y}=\text{u}\}
\end{align*}
\end{equation}

This result is accurate but imprecise.
A first example: abstract interpretation

What abstract domain would allow us to conclude that y is even?