Static analysis

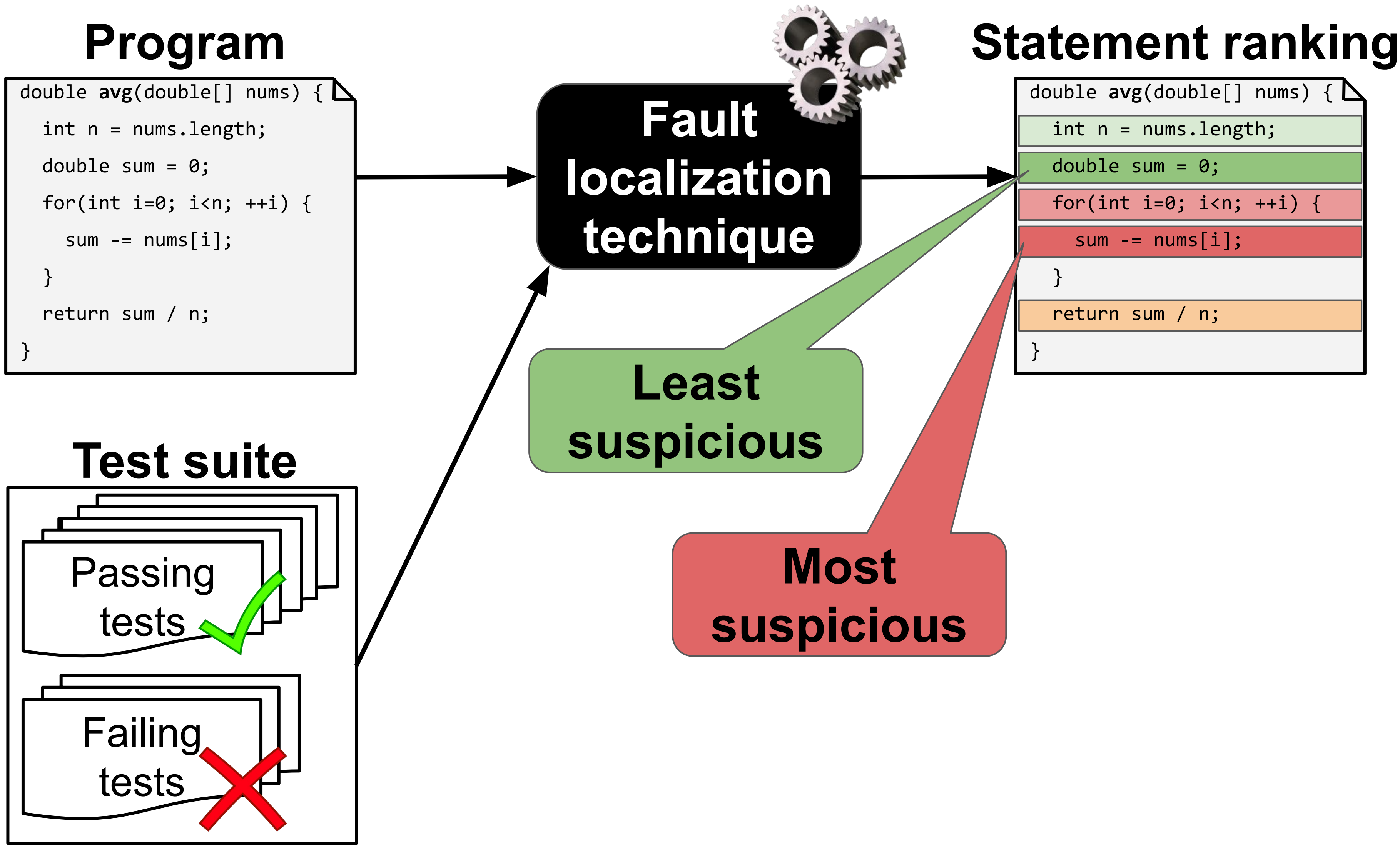
UW CSE P 504

Today

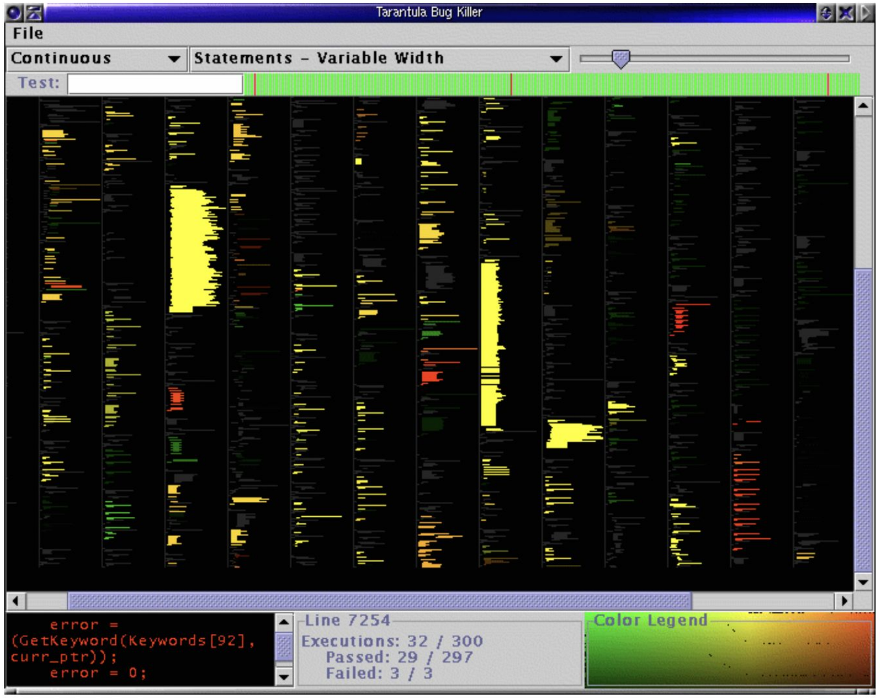
- Recap: statistical fault localization
- Static Analysis
  - Motivation
  - Examples
  - Intro to Abstract Interpretation

Recap: statistical fault localization

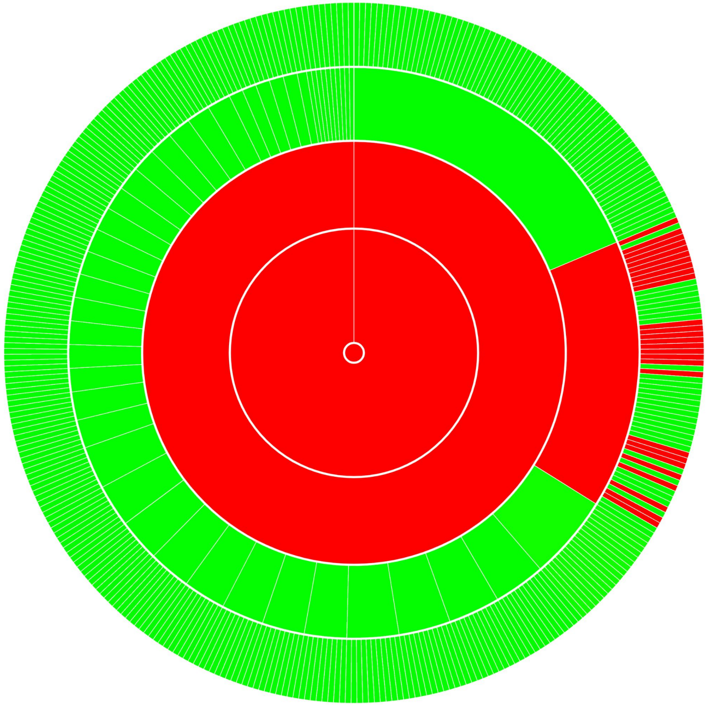
Recap: statistical fault localization



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**.

[y:=2, x:=2]

```
y = x++
```

???

* 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**.

[y:=2, x:=2]

y = x++

[y:=2, x:=3]

* 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**.

[y:=2, x:=2]

```
y = x++
```

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**.

[y:=even, x:=even]

```
y = x++
```

* 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**.

[y:=2, x:=2]

y = x++

[y:=2, x:=3]

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**.

[y:=even, x:=even]

y = x++

???

* 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**.

[y:=2, x:=2]

y = x++

[y:=2, x:=3]

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**.

[y:=even, x:=even]

y = x++

[y:=even, x:=odd]

* 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**.

[y:=2, x:=2]

y = x++

[y:=2, x:=3]

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**.

[y:=prime, x:=prime]

y = x++

???

* 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**.

[y:=2, x:=2]

y = x++

[y:=2, x:=3]

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**.

[y:=prime, x:=prime]

y = x++

[y:=prime, x:=anything]

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

Static vs. dynamic analysis

Dynamic analysis

- Concrete domain
- Precise but unsound
- Slow if exhaustive

Static analysis

- Abstract domain
- Sound but imprecise
- Slow if precise

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

Abstract domain

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

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, ...

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

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

Analysis result

Ground Truth	*Pos*	*Neg*
Pos		
Neg		

Concrete domain **vs.** **Abstract domain**

0, 2, 4, 6, 8, 10, ... *even, odd, anything*

```
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)

		Analysis result	
		Pos	*Neg*
Ground Truth	*Pos*		
	Neg		

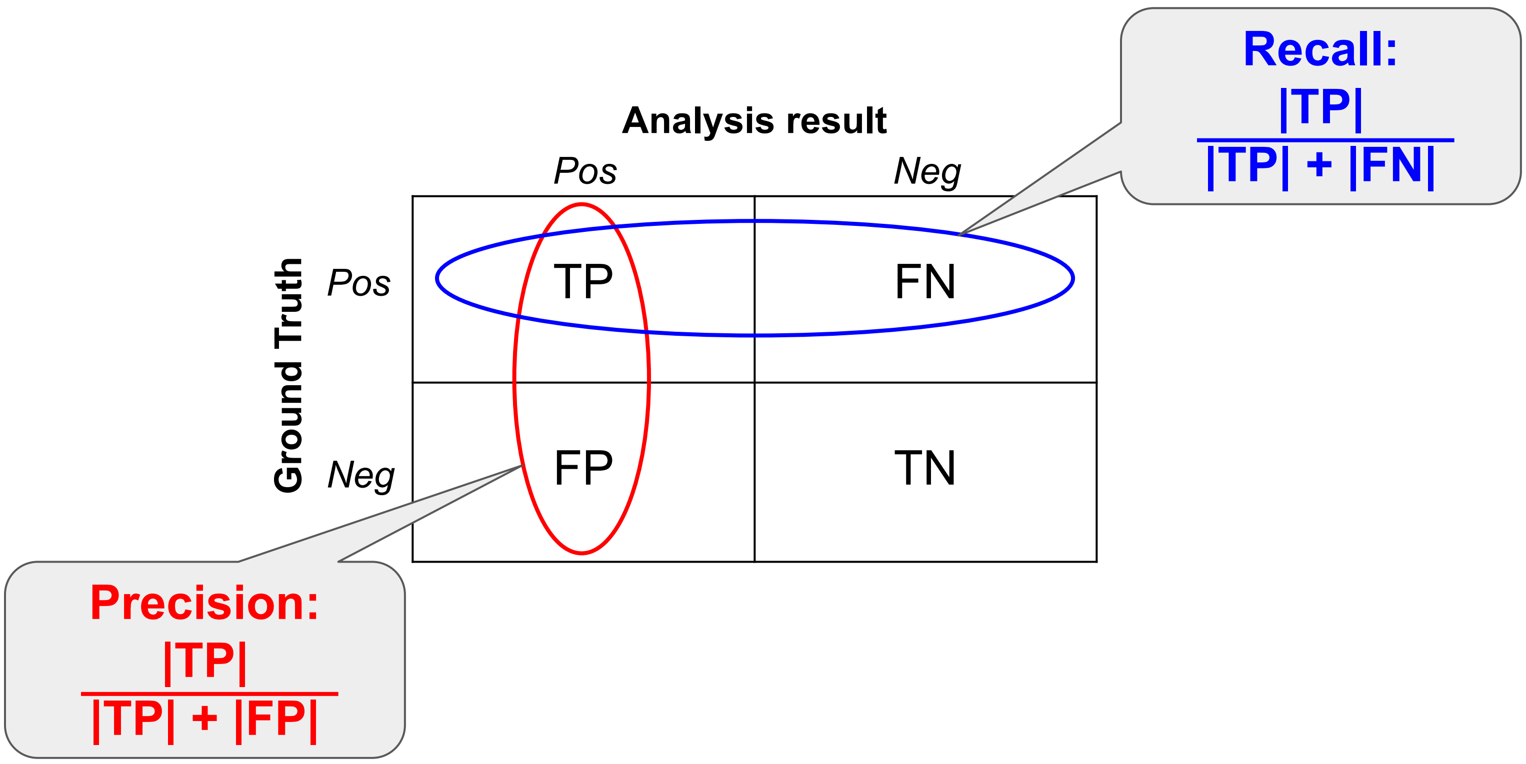
Terminology and important concepts

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

Ground Truth \ Analysis result	*Pos*	*Neg*
Pos	TP	FN
Neg	FP	TN

Terminology and important concepts

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



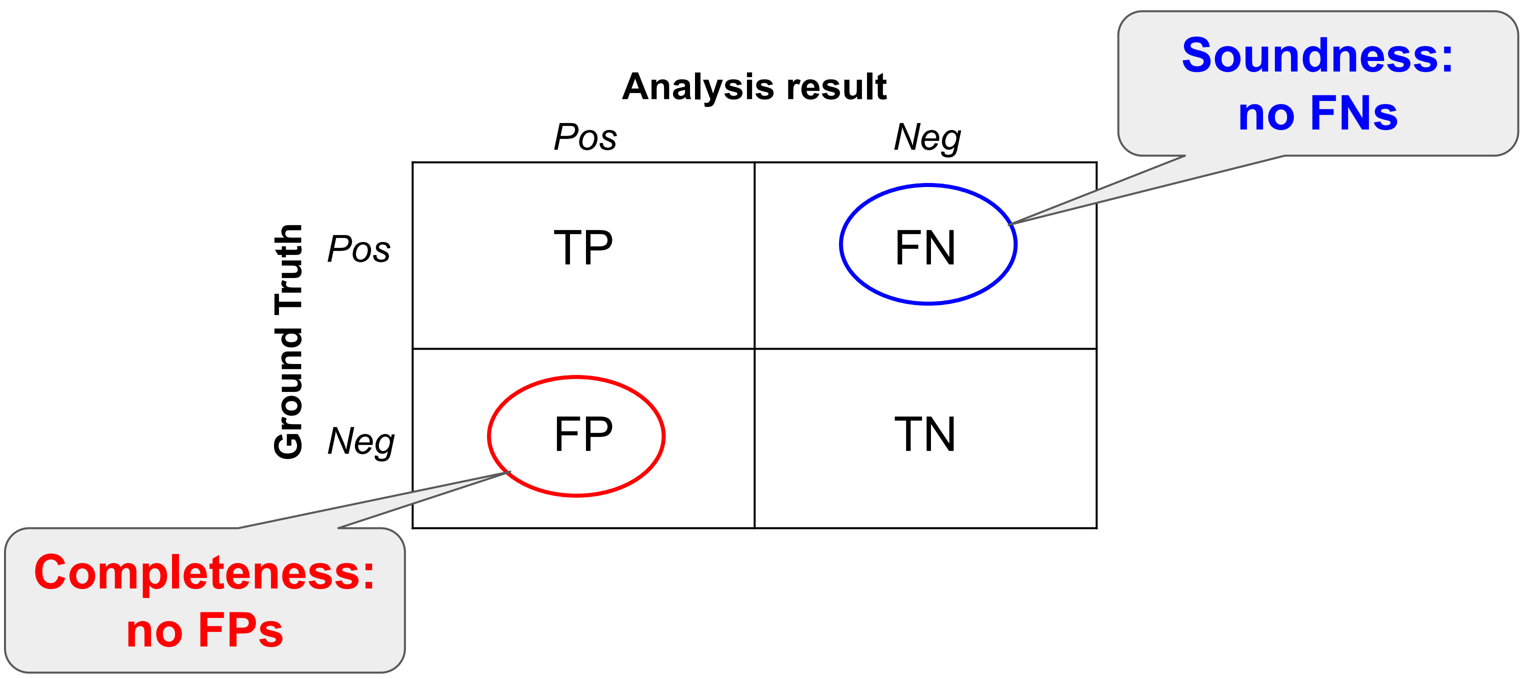
Terminology and important concepts

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

		Analysis result	
		Pos	*Neg*
Ground Truth	*Pos*	TP	FN
	Neg	FP	TN

Terminology and important concepts

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



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);
```

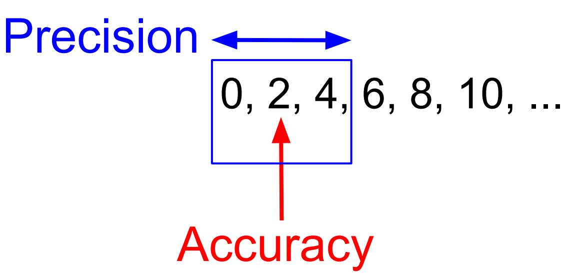
Concrete domain	**Abstract domain**
0, 2, 4, 6, 8, 10, ...	*even, odd, anything*

Terminology and important concepts

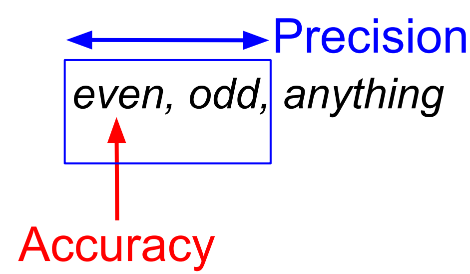
1. Precision vs. Recall (and FP/FN/TP/TN)
2. Soundness vs. Completeness
3. Accuracy vs. Precision

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

Concrete domain



Abstract domain



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);
```

Concrete domain

Precision

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

Accuracy

Abstract domain

Precision

even, odd, anything

Accuracy

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

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

Reachability

```
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."*

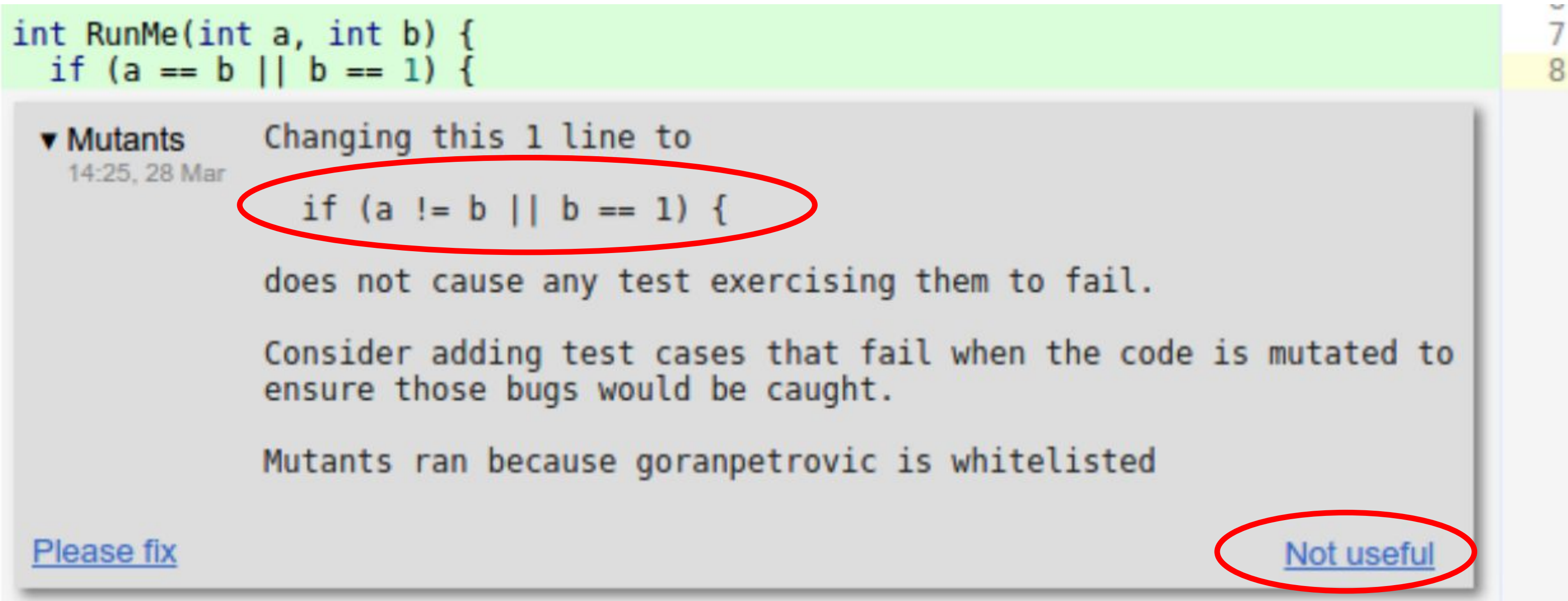
"Lessons from Building Static Analysis Tools at Google", CACM 2018

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.*

"Lessons from Building Static Analysis Tools at Google", CACM 2018

Effective false positive: example (mutation testing)



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.*

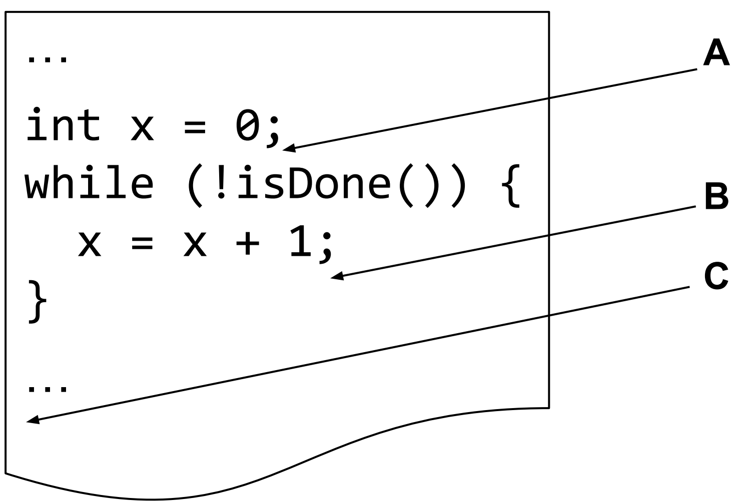
Do you agree with this characterization?

Is effective false positive rate an adequate measure?

Abstract Interpretation

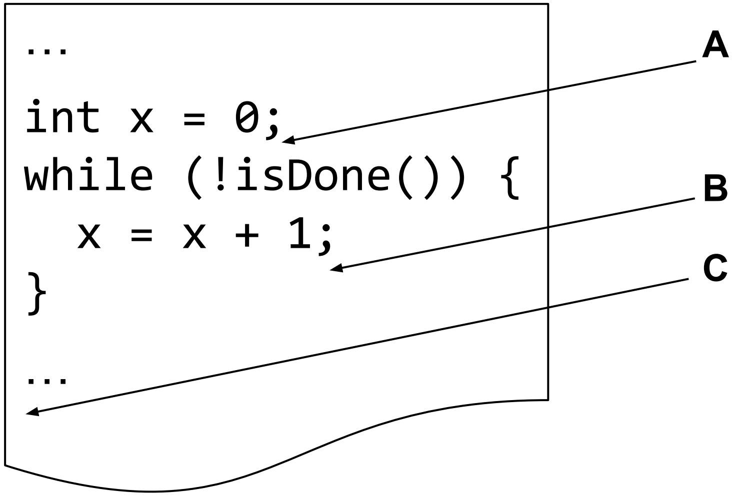
Properties of an ideal program analysis

- Soundness
- Completeness
- Termination



Properties of an ideal program analysis

- Soundness
- Completeness
- Termination



Abstract interpretation sacrifices completeness (precision)

A first example

Program

```
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

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

SSA form

```
x1 = 0;
y1 = read_even();
x2 = y1 + 1;
y2 = 2 * x2;
x3 = y2 - 2;
y3 = x3 / 2;
```

X_1 is never read.

A first example: one concrete execution

Program

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

Concrete execution

```
{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

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

SSA form

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

Program

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

SSA form

$$
\begin{aligned}
x_1 &= 0; \\
y_1 &= \mathrm{read\_even}(); \\
x_2 &= y_1 + 1; \\
y_2 &= 2 * x_2; \\
x_3 &= y_2 - 2; \\
y_3 &= x_3 / 2;
\end{aligned}
$$

$y_3 = x_3 / 2$

$y_3 = (y_2 - 2) / 2$

$y_3 = (2 * x_2 - 2) / 2$

$y_3 = (2 * (y_1 + 1) - 2) / 2$

$y_3 = (2 * y_1 + 2 - 2) / 2$

$y_3 = y_1$

$x_3 = y_1 * 2$

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;
```

+	unk	odd	even
unk			
odd			
even			

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;
```

+	unk	odd	even
unk	unk	unk	unk
odd	unk	even	odd
even	unk	odd	even

What’s the abstract type of x and y after (abstract) execution?

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 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=**e**; y=**u**}

Convince yourself that this is true.

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=o; y=e}
{x=o; y=e}
{x=e; y=e}
{x=e; y=u}
```

This result is accurate but imprecise.

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=o; y=e}
{x=o; y=e}
{x=e; y=e}
{x=e; y=u}
```

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