CSE P 504

Advanced topics in Software Systems

Fall 2022

Course introduction

October 03, 2022

Today

- Course overview
- What is Software Engineering
- Static vs. dynamic program analysis
- Small-group brainstorming:
  software testing and debugging challenges

Course overview

The CSE P 504 team

Instructor

- René Just
- Office: CSE2 338
- Office hours: After class and by appointment
- rjust@cs.washington.edu

Teaching assistant

- Hannah Potter
- Office: TBD
- Office hours: by appointment
- hkpotter@cs.washington.edu

Logistics

- CSE2 G10, Mon, 6:30pm – 9:20pm.
- Lectures, discussions, and in-class exercises.
- Course material, schedule, etc. on website: https://homes.cs.washington.edu/~rjust/courses/CSEP504
- Submission of assignments via Canvas: https://canvas.uw.edu

Course overview: the big picture

- **10/03:** Course introduction
- **10/10:** Best practices and version control
- **10/17:** Coverage-based testing
- **10/24:** Mutation-based testing
- **10/31:** Delta debugging
- **11/07:** Invariants and partial oracles
- **11/14:** Statistical fault localization
- **11/21:** Static analysis
- **11/28:** Abstract interpretation
- **12/05:** Formal methods

Each class meeting has two parts: lecture and in-class activity.

Course overview: the big picture

• **10/03:** Course introduction	HW 1
• **10/10:** Best practices and version control	In-class exercise
• **10/17:** Coverage-based testing	In-class exercise
• **10/24:** Mutation-based testing	In-class exercise
• **10/31:** Delta debugging	In-class exercise
• **11/07:** Invariants and partial oracles	In-class exercise
• **11/14:** Statistical fault localization	In-class exercise
• **11/21:** Static analysis	Happy Thanksgiving
• **11/28:** Abstract interpretation	HW 2
• **12/05:** Formal methods	In-class exercise

Questions?

Course overview: in-class exercises

In-class exercises (graded activities) have two parts

1. In-class part: Small-group work on a problem set
2. Take-home part: Reflection and submission of answers

What if I can't attend a class meeting?

- A Zoom option is available for all in-class exercises to facilitate small-group work for remote participants.
- Submissions for in-class exercises are due at the end of the week.

Course overview: grading

- **20%** Homeworks
- **70%** In-class exercises (7 sessions)
- **10%** Participation

Questions?

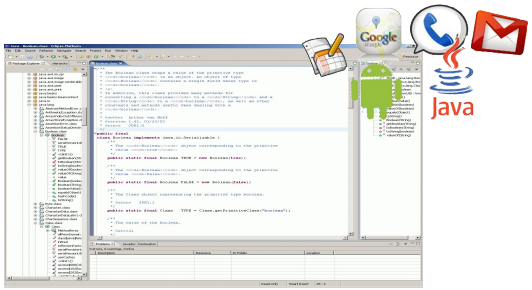
Course overview: expectations

- Programming (and OO) experience.
- Read a few research papers.
- Engage in discussions.
- Have fun!

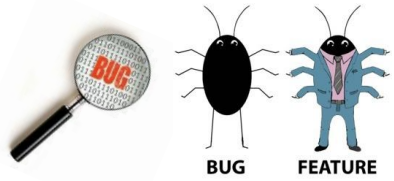
What is Software Engineering

What is Software Engineering?

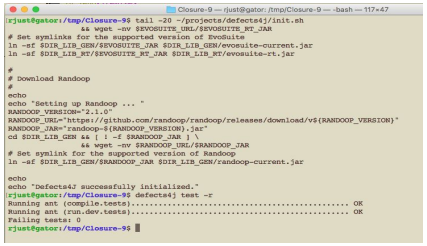
- Developing in an IDE and software ecosystem?



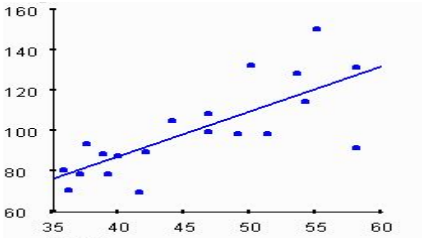
- Testing and debugging a software system?



- Deploying and running a software system?



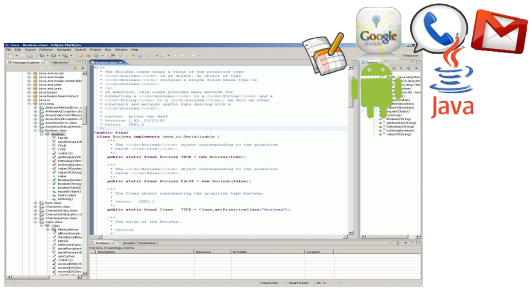
- Empirically evaluating a software system?



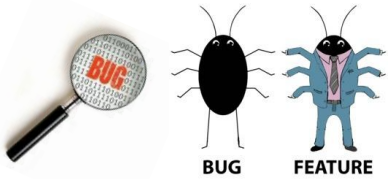
- Writing (design) docs?

What is Software Engineering?

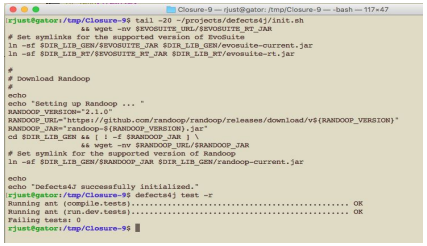
- Developing in an IDE
  and software ecosystem?



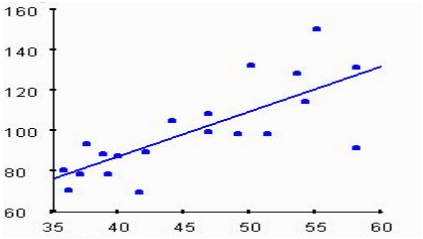
- Testing and debugging a software system?



- Deploying and running
  a software system?



- Empirically evaluating a software system?



- Writing (design) docs?

All of the above and much more!

What is Software Engineering?

More than just writing code

The complete process of specifying, designing, developing, analyzing, deploying, and maintaining a software system.

- Common Software Engineering tasks include:
  - Requirements engineering
  - Specification writing and documentation
  - Software architecture and design
  - Programming
  - Software testing and debugging
  - Refactoring

What is Software Engineering?

More than just writing code

The complete process of specifying, designing, developing, analyzing, deploying, and maintaining a software system.

- Common Software Engineering tasks include:
  - Requirements engineering
  - Specification writing and documentation
  - Software architecture and design
  - Programming
  - Software testing and debugging
  - Refactoring

Just one out of many important tasks!

What is Software Engineering?

More than just writing code

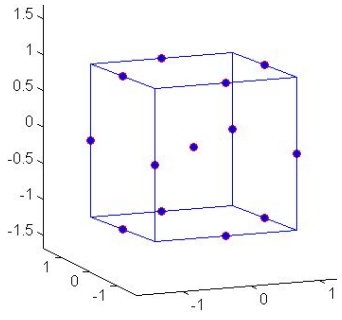
The complete process of specifying, designing, developing, analyzing, deploying, and maintaining a software system.

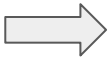
- Common Software Engineering tasks include:
  - Requirements engineering
  - Specification writing and documentation
  - Software architecture and design
  - Programming
  - Software testing and debugging
  - Refactoring

Program analysis is a crucial task in Software Engineering!

The Role of Software Engineering in Research

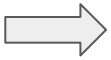
Experimental infrastructure is software, too!





Infrastructure

Design space exploration



1	0.34	0.81
2	0.52	0.32
3	0.21	0.53
4	0.81	0.22
...	...	...

Example (automated debugging)

- 150 configurations, 1000+ benchmarks
- 1-85 hours per execution
- 200,000+ CPU hours (~23 CPU years)

Static vs. dynamic program analysis

What is program analysis?

- (Automatically) analyze the behavior of a program
  - optimize the program or
  - check program's behavior (against its specification)
- Concerned with properties such as
  - Correctness
  - Safety
  - Liveness
  - Performance
- Can be static or dynamic (or both), which affects
  - Computational cost
  - Accuracy and precision

Why do we need program analysis?

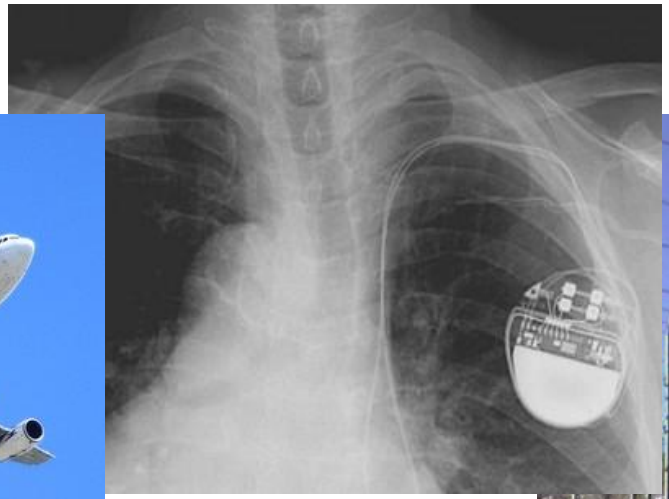
Why do we need program analysis?

- ~15 million lines of code

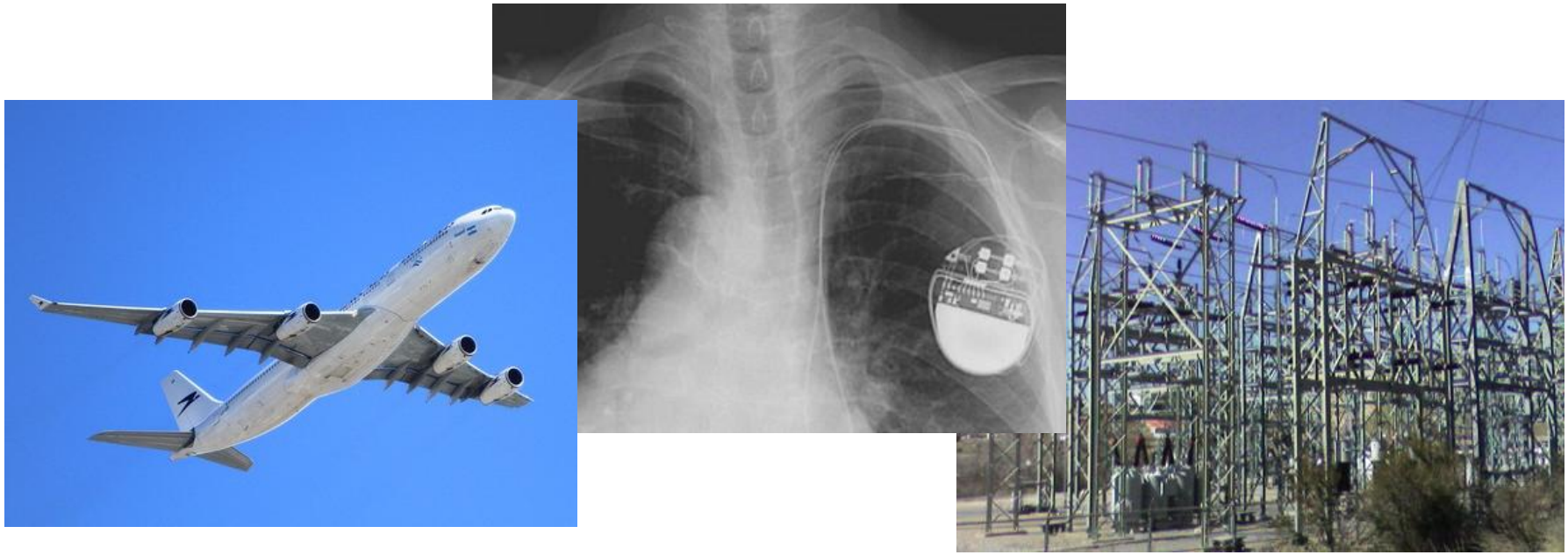
Let’s say 50 lines per page (0.05 mm)

- 300000 pages
- 15 m (49 ft)

Why do we need program analysis?



Why do we need program analysis?



Reliability is critical for many programs

- Increase confidence in program correctness
- Understand the program's behavior
- Prove properties about the program

A first example: code review

Different types of reviews

- Code/design review
- Informal walkthrough
- Formal inspection

A requirement for many (safety-critical) systems.

A first example: code review

Different types of reviews

- Code/design review
- Informal walkthrough
- Formal inspection

```
double foo(double[] d) {
  int n = d.length;
  double s = 0;
  int i = 0;
  while (i<n)
  s = s + d[i];
  i = i + 1;
  double a = s / n;
  return a;
}
```

Let’s do an informal code review.

Anything that could be improved in this (Java) code?

A first example: code review

Different types of reviews

- Code/design review
- Informal walkthrough
- Formal inspection

```
double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;

  int i = 0;
  while (i<n)
    sum = sum + nums[i];
    i = i + 1;

  double avg = sum / n;

  return avg;
}
```

Now, is anything wrong with that code?

A first example: code review

Different types of reviews

- Code/design review
- Informal walkthrough
- Formal inspection

```
double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;

  int i = 0;
  while (i<n)
    sum = sum + nums[i];
    i = i + 1;

  double avg = sum / n;

  return avg;
}
```

```
static OSStatus
SSLVerifySignedServerKeyExchange(...) {
        OSStatus err;
        ...
        if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
                goto fail;
        if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
                goto fail;
        if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
                goto fail;
                goto fail;
        if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
                goto fail;
        err = sslRawVerify(ctx, ctx->peerPubKey, dataToSign, dataToSignLen, signature, signatureLen);
        if(err) {
                sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify returned %d\n", (int)err);
                goto fail;
        }
        fail:
                SSLFreeBuffer(&signedHashes);
                SSLFreeBuffer(&hashCtx);
                return err;
}
```

Anything wrong with that code?

```
static OSStatus
SSLVerifySignedServerKeyExchange(...) {
        OSStatus err;

        if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
                goto fail;
                goto fail;
        if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
                goto fail;
        err = sslRawVerify(ctx, ctx->peerPubKey, dataToSign, dataToSignLen, signature, signatureLen);
        if(err) {
                sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify returned %d\n", (int)err);
                goto fail;
        }
        fail:
                SSLFreeBuffer(&signedHashes);
                SSLFreeBuffer(&hashCtx);
                return err;
}
```

Apple's "goto fail" bug:
a security vulnerability for 2 years!

Anything wrong with that code?

Code review

Pros

- Can be applied at any step in the development process
- Does not require an executable program
- Improves confidence and communication

Cons

- Time-consuming
- Mostly informal
- Not replicable

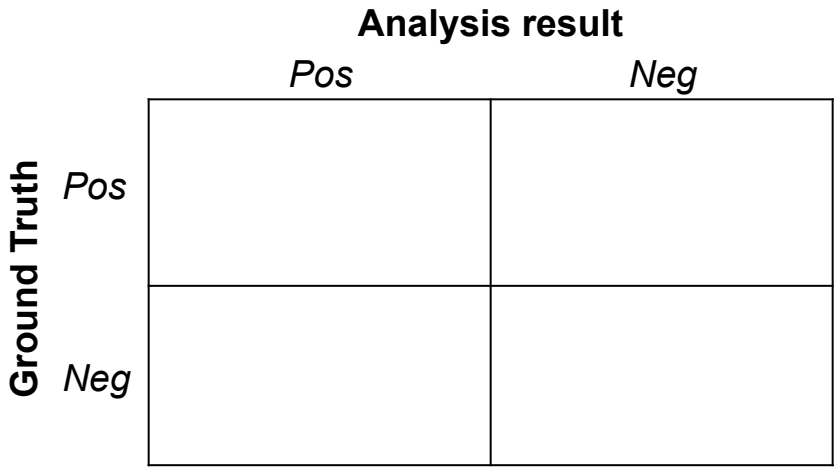
Terminology and important concepts

Let's define the following terms,
in the context of program analysis:

1. Precision vs. Recall (and FP/FN/TP/TN)
2. Soundness vs. Completeness
3. Concrete domain vs. Abstract domain
4. Accuracy vs. Precision (and conservative analysis)

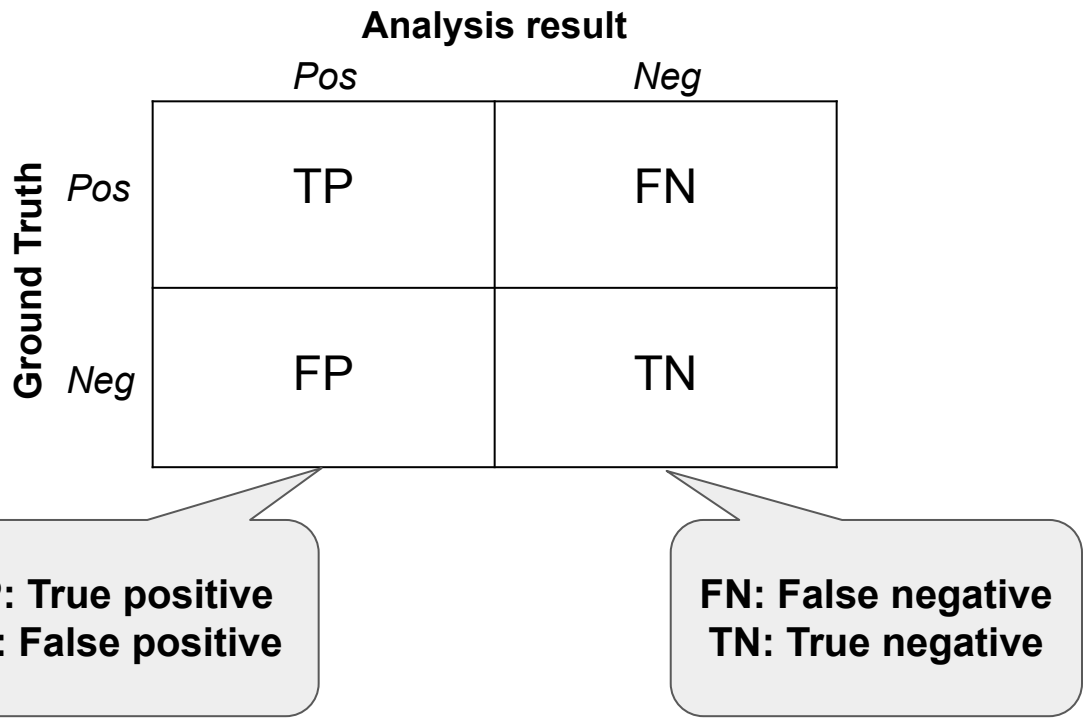
Terminology and important concepts

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



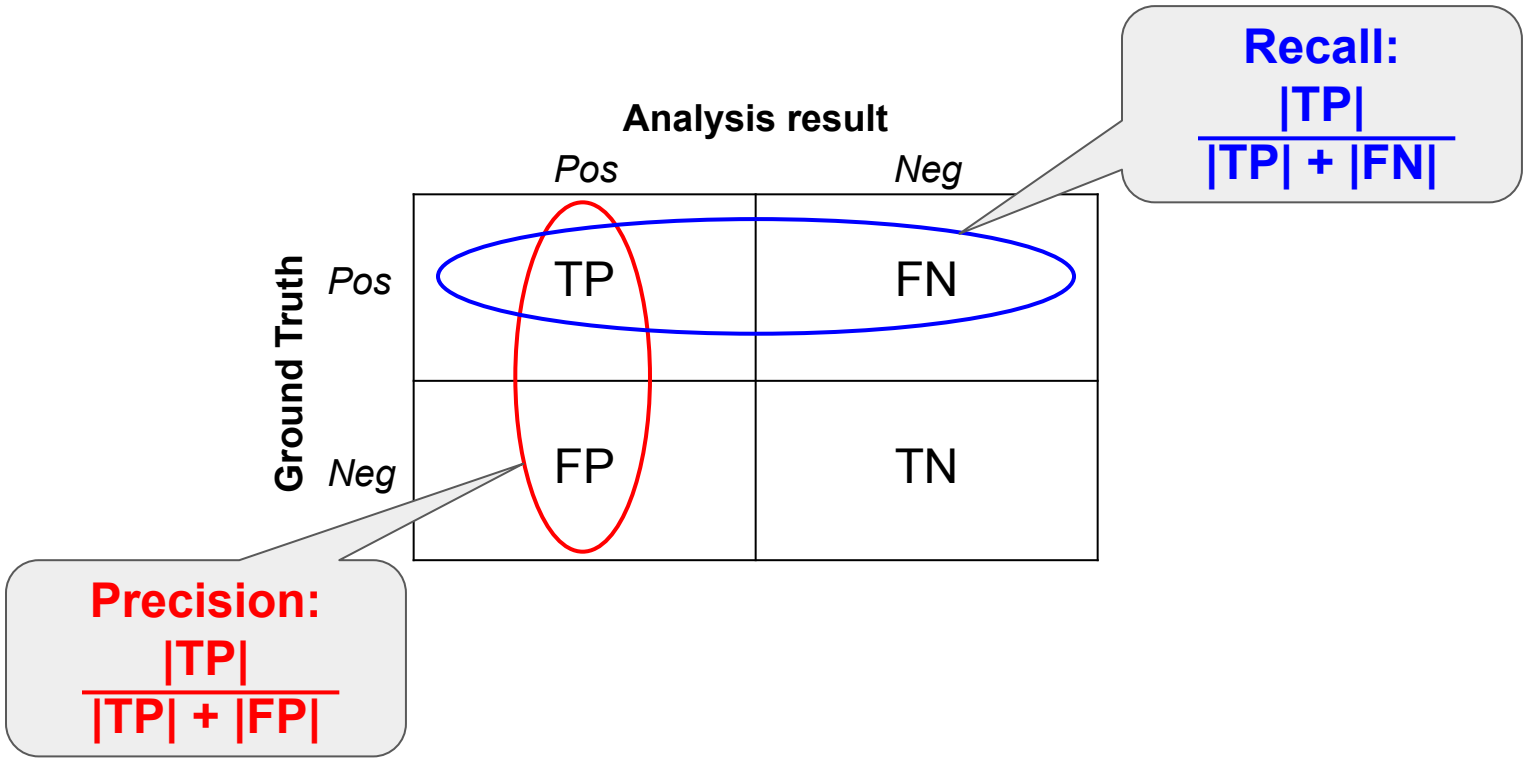
Terminology and important concepts

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



Terminology and important concepts

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



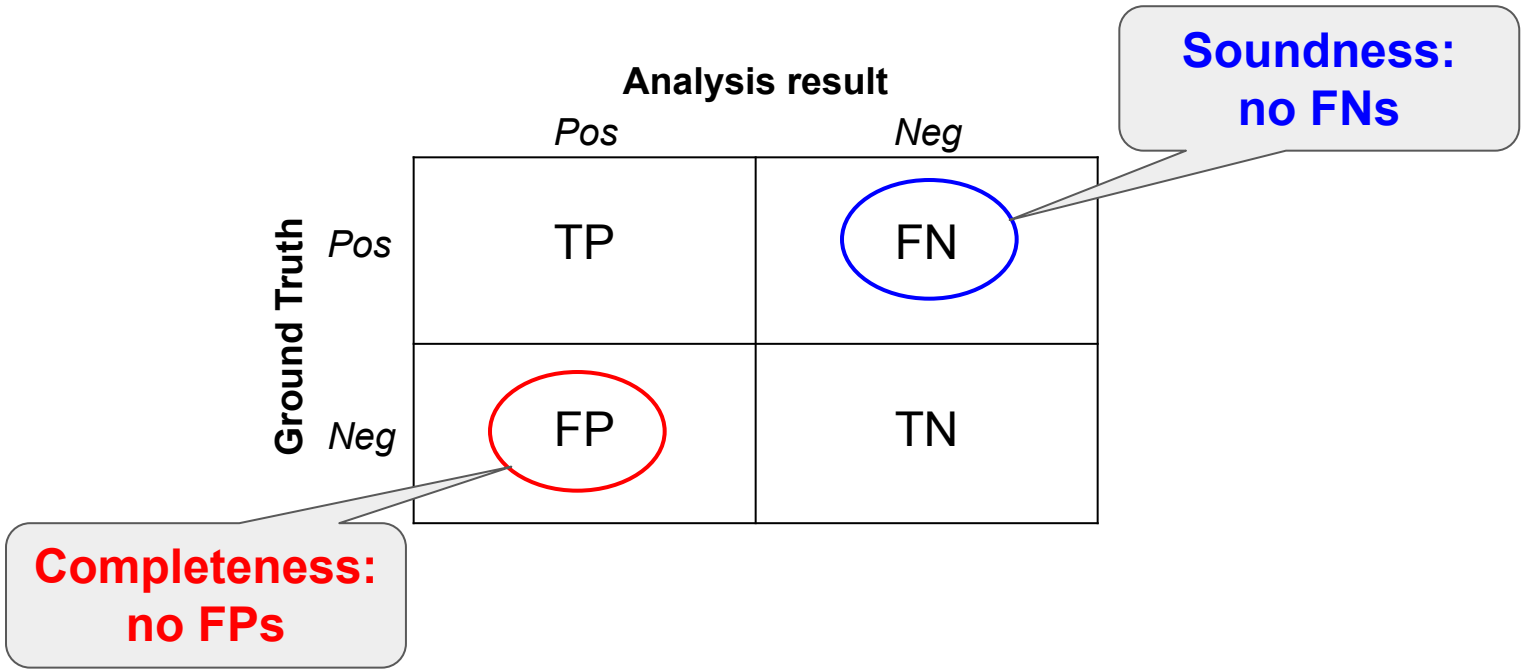
Terminology and important concepts

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. Concrete domain vs. Abstract domain

Concrete domain	**Abstract domain**
0, 1, 2, 3, 4, …	*even, odd*

Terminology and important concepts

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

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

Static vs. dynamic analysis

What are the key differences?

Static vs. dynamic analysis: overview

Static analysis

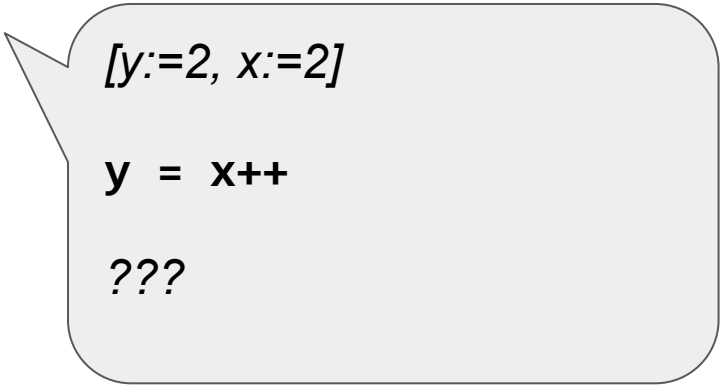
- Reason about the program without executing it.
- Build an abstraction of run-time states.
- Reason over abstract domain.
- Prove a property of the program.
- Sound* but conservative.

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

Static vs. dynamic analysis: overview

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 conservative.

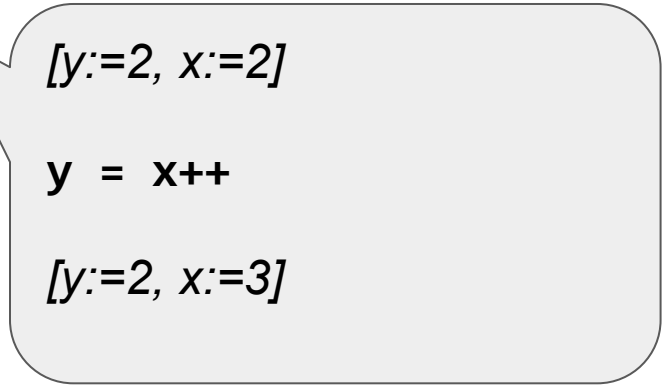


* Some static analyses are unsound; dynamic analyses can be sound.

Static vs. dynamic analysis: overview

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 conservative.

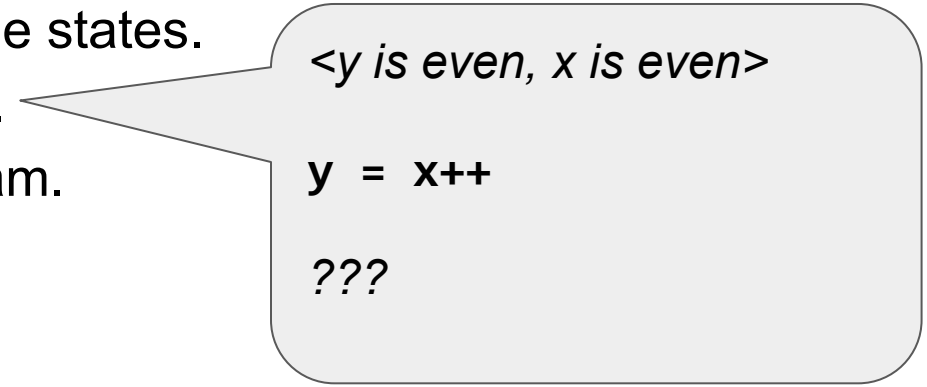


* Some static analyses are unsound; dynamic analyses can be sound.

Static vs. dynamic analysis: overview

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 conservative.

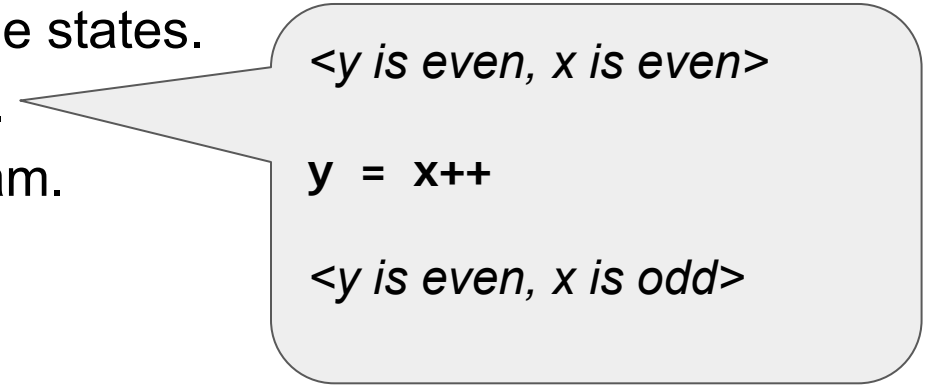


* Some static analyses are unsound; dynamic analyses can be sound.

Static vs. dynamic analysis: overview

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 conservative.

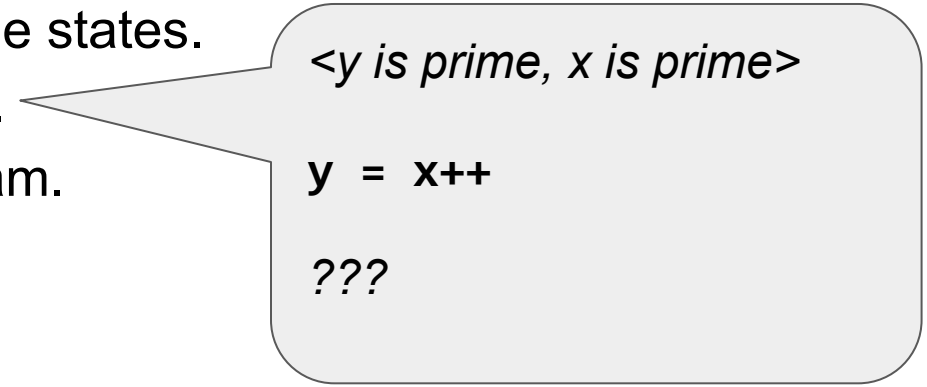


* Some static analyses are unsound; dynamic analyses can be sound.

Static vs. dynamic analysis: overview

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 conservative.

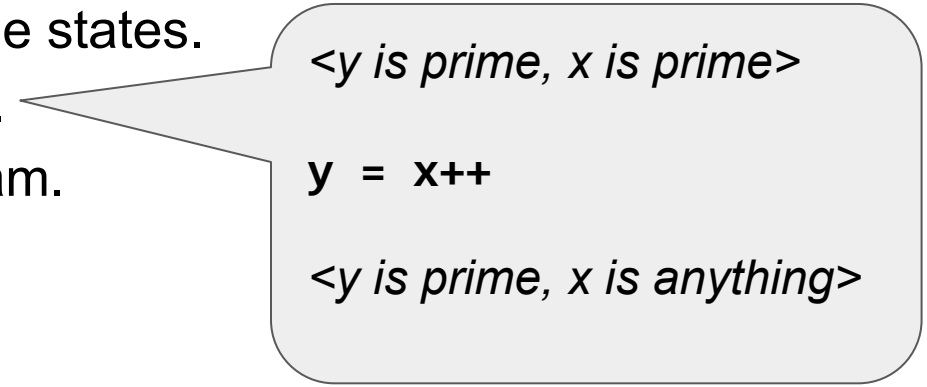


* Some static analyses are unsound; dynamic analyses can be sound.

Static vs. dynamic analysis: overview

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 conservative.

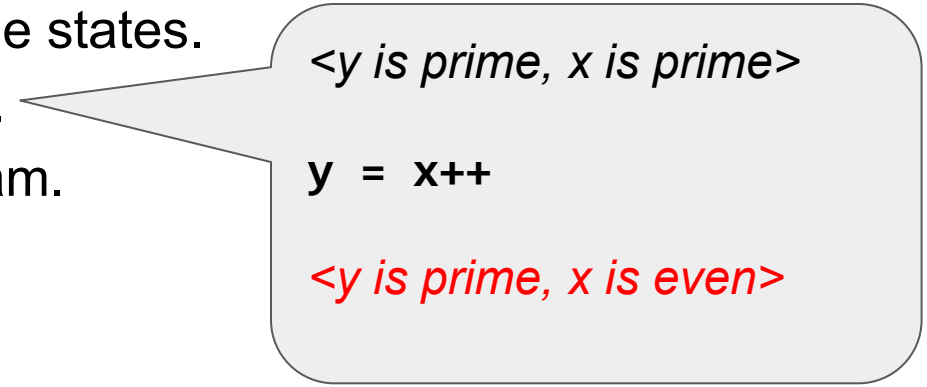


* Some static analyses are unsound; dynamic analyses can be sound.

Static vs. dynamic analysis: overview

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 conservative.



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

Static vs. dynamic analysis: overview

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 conservative.

The statement
"**f returns a non-negative value**"
is weaker (but easier to establish)
than the statement
"**f returns the absolute value of its argument**".

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

Static vs. dynamic analysis: overview

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 conservative.

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 analysis: examples

Type checking (also compiler optimizations)

```
double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;

  int i = 0.0;
  while (i<n) {
    sum = sum + nums[i];
    i = i + 1;
  }
  double avg = sum / n;

  return avg;
}
```

```
double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;

  int i = 0;
  while (i<n) {
    sum = sum + nums[i];
    i = i + 1;
  }
  double avg = sum / n;

  return avg;
}
```

Static analysis: examples

Rule/pattern-based analysis (PMD, Findbugs, etc.).

```
double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;

  int i = 0;
  while (i<n)
    sum = sum + nums[i];
    i = i + 1;

  double avg = sum / n;

  return avg;
}
```

```
double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;

  int i = 0;
  while (i<n) {
    sum = sum + nums[i];
    i = i + 1;
  }
  double avg = sum / n;

  return avg;
}
```

Static analysis: examples

Control-flow and data-flow analysis

```
double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;

  int i = 0;
  while (i<n)
    sum = sum + nums[i];
    i = i + 1;

  double avg = sum / n;

  return avg;
}
```

What is the control flow graph (CFG) for this avg function?

Static analysis: examples

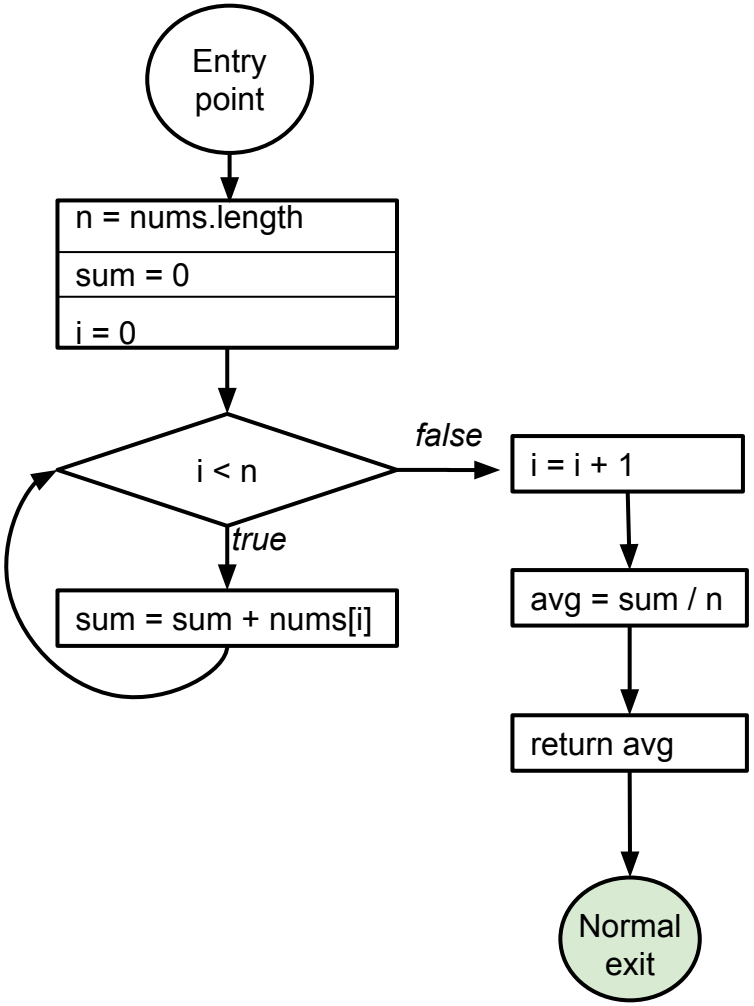
Control-flow and data-flow analysis

```
double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;

  int i = 0;
  while (i<n)
    sum = sum + nums[i];
    i = i + 1;

  double avg = sum / n;

  return avg;
}
```



Static analysis: examples

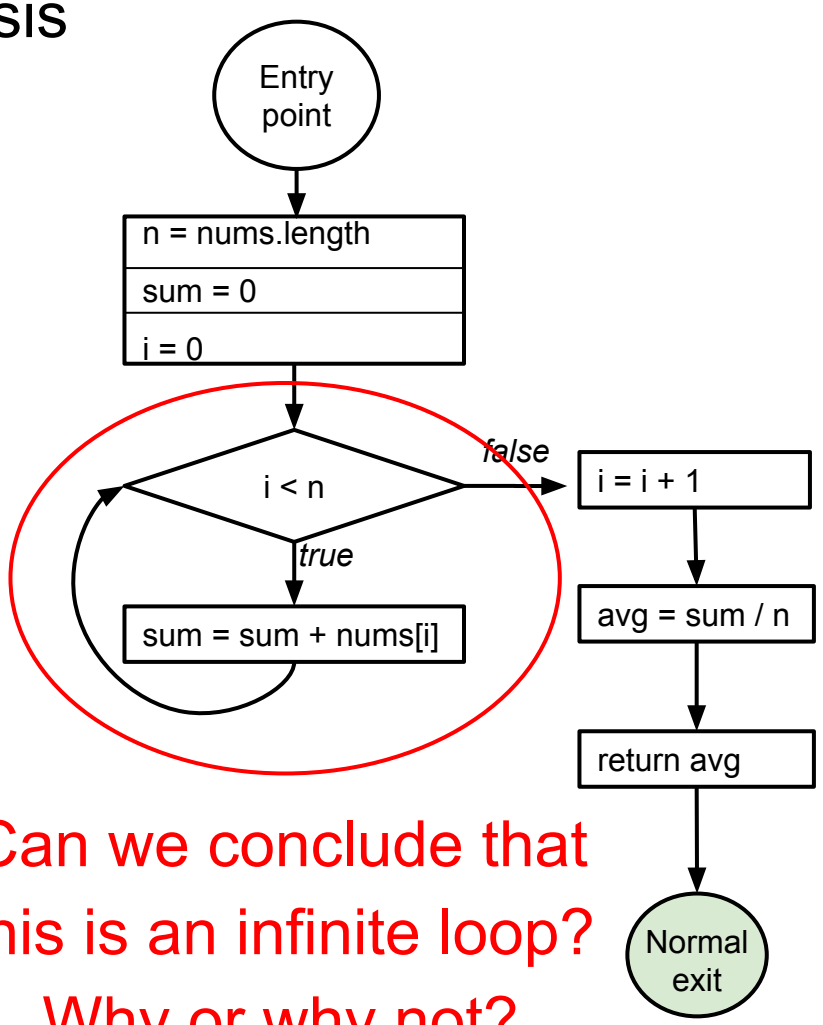
Control-flow and data-flow analysis

```
double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;

  int i = 0;
  while (i<n)
    sum = sum + nums[i];
    i = i + 1;

  double avg = sum / n;

  return avg;
}
```



Can we conclude that this is an infinite loop? Why or why not?

Dynamic analysis: examples

Software testing (also monitoring and profiling)

```
double avg(double[] nums) {
  int n = nums.length;
  double sum = 0;

  int i = 0;
  while (i<n)
    sum = sum + nums[i];
    i = i + 1;

  double avg = sum / n;

  return avg;
}
```

A test for the avg function:

```
@Test
public void testAvg() {
  double nums =
      new double[]{1.0, 2.0, 3.0});
  double actual = Math.avg(nums);
  double expected = 2.0;
  assertEquals(expected,actual,EPS);
}
```

Static vs. dynamic analysis

What are the key challenges?

Static vs. dynamic analysis: challenges

Static analysis: choose good abstractions

- Chosen abstraction **determines cost** (time and space)
- Chosen abstraction **determines precision** (what information is lost)

Dynamic analysis: choose good representatives (tests)

- Chosen tests **determine cost** (time and space)
- Chosen tests **determine accuracy** (what executions are never seen)

Static vs. dynamic analysis: summary

Static analysis

- Abstract domain
- Conservative due to abstraction
- Sound due to conservatism
- Slow if precise

Dynamic analysis

- Concrete execution
- Precise no approximation
- Unsound, does not generalize
- Slow if exhaustive

Small-group brainstorming: software testing and debugging challenges