CS383 Software Analysis

Intro to Software Analysis

Outline

1. What is software analysis?
2. This course
   a. Administrative info
   b. Learning objectives
   c. Assignments
3. Intro to Software Analysis
   a. Success / failure stories
   b. Undecidability of program analysis
   c. Landscape of techniques

What is software analysis?

menti.com

code: 5276 2150

What is program analysis?

Suppose you have written a program. What can you know about it? A program analysis tool can *attempt* to provide answers to these questions:

- Does the program meet its specification?
- Does the program terminate?
- Are there logic errors?
- Is there any dead / unreachable code?
- Are there any memory leaks?
- ....

What techniques and/or processes do you use to analyze your code?

code: 5276 2150

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

Administrivia

Course Website:

- Assignments, labs, links to readings, syllabus, etc...

Piazza:

- Asynchronous communication
- Group forming
- Can post anonymously (anonymous just to classmates)
- Answer your peers questions!
- https://piazza.com/class/m61bd6hr6hw7h/

Gradescope:

- Entry code: G3NGKJ
- All assignments submitted here - *not autograded!*

Course Format

- **Lectures**: learning algorithms and techniques for analyzing code automatically
- **Labs**: hands on experience running tools, understanding their limitations + project check-ins
- **Homeworks**: implement tools and algorithms in a mini setting
- **Readings:** blog posts / informative 2-3 pages
- **Project**: in groups. deep case studies and improving upon existing algorithms

Almost all in Java

Syllabus

- Homeworks: 20%
- Labs: 35%
- Project I : 10%
- Project II: 30%
- Participation: 5%

Schedule

- Lecture Mon and Wed
- Lab Park 230 Monday 2:40-4pm
- Project 1:
  - Groups formed by 2/17
  - Due 3/3
- Project 2: (same groups)
  - Due 4/28

Intro to Software Analysis

Software Disaster Stories

Numeric Overflow!

- Guidance system collected 64 bit floating point velocity reading

- Stored in a 16-bit signed integer

- Exploded 40 seconds after takeoff
- Loss of $370 million USD
- *Used to be considered* “costliest software bug”
- Carrying expensive payload

Software Disaster Stories

- Guidance system collected 64 bit floating point velocity reading
  - This is a `double` in Java
  - (2^-1074, 2^1022)

- Stored in a 16-bit signed integer
  - This is a `short` in Java
  - (-2^15, 2^15)

- Previous rocket had slower velocity and the 64-bit floating point value was always in the acceptable 16-bit int range

Software Disaster Stories

Integer Under/Over flow doesn't always result in an exception - depends on the language!

PoWH Coin lost 800k Overnight!

```
contract PoWHCoin {
    uint256 public totalSupply;
    mapping(address => uint256) public balanceOf;

    function buy(uint256 _amount) public {
        balanceOf[msg.sender] += _amount;
    }

     function withdraw(uint256 _amount) public {
        uint256 balanceRem = balanceOf[msg.sender] - _amount;
        require( balanceRem > 0);
        ....
    }
}
```

Software Disaster Stories

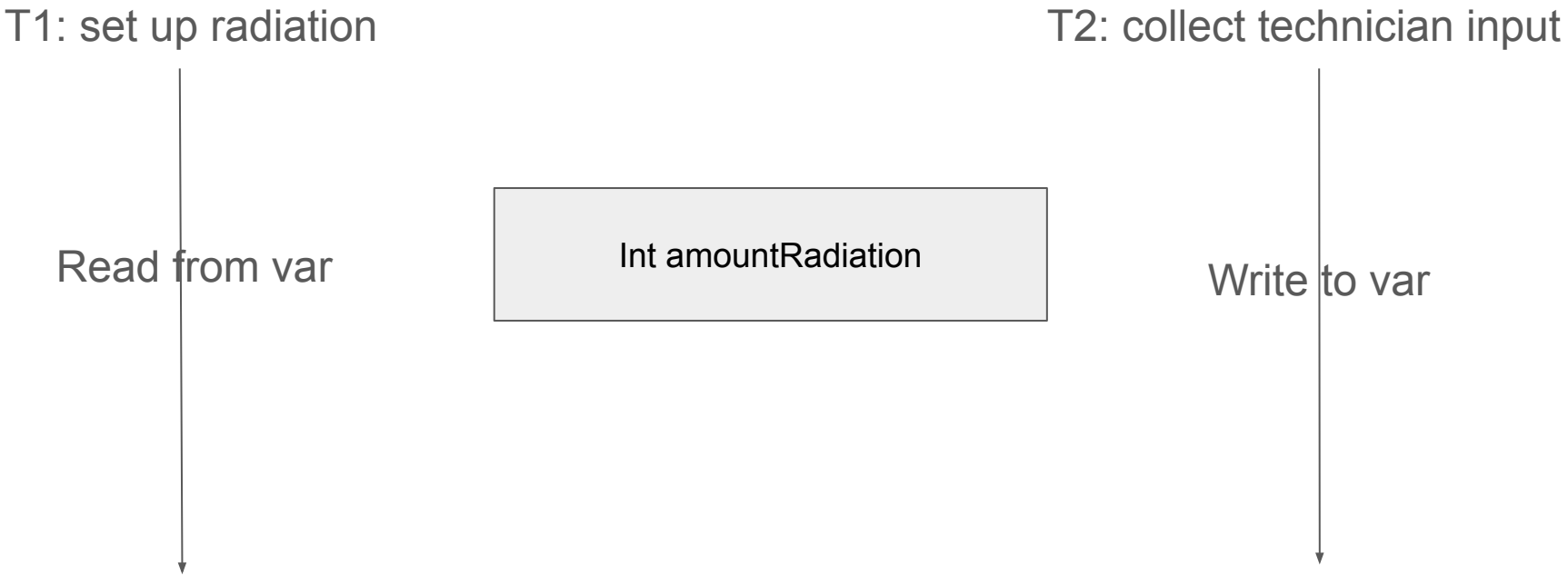
- *AOB* in software update caused “blue screen of death” OS crash
- Downed commercial flights, disrupted banking and healthcare services, and 911 call centers
- Cost Fortune 500 companies $5.4Billion

```
int[] channels = new int[20];

// sensorCode can be from 0-21;
channels[sensorCode]
```

Software Disaster Stories

- Therac-25 computer-controlled radiation therapy machine
- Massively overdosed 6 people between 1985 and 1987
- Race condition! Multiple routines running concurrently which shared a single variable for technician entered settings

Software Disaster Stories



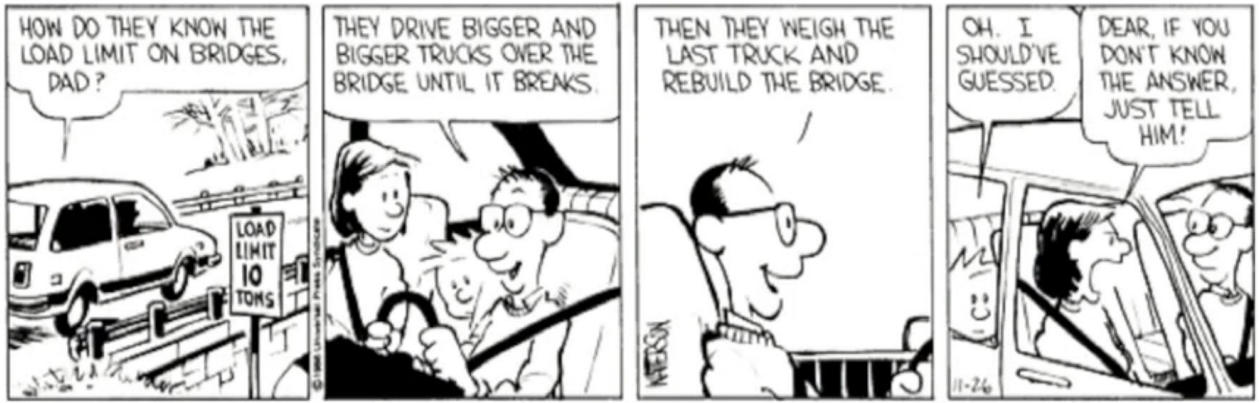
Software Disaster Stories - Many more!

https://en.wikipedia.org/wiki/List_of_software_bugs

Even in non-safety critical systems, software correctness is important!

- Data privacy
- Efficiency and performance
- User trust

Software engineering?



Undecidability of Program Analysis

Program Analysis as a Decision Problem

Given a program P and a property Φ, a program analysis must decide in finite time whether or not P satisfies Φ, returning True if P satisfies Φ and False if P doesn't satisfy Φ.

Example properties:

- Existence of a NPE
- Existence of a Div-by-zero
- Termination
- Postcondition
- Existence of dead code (reachability)
- the value of some expression inside a loop the same in every iteration
- the value of x does not depend on the program input
- lower and upper bounds of the integer variable x
- x and y point to disjoint memory
- ....

Why is this a hard problem?

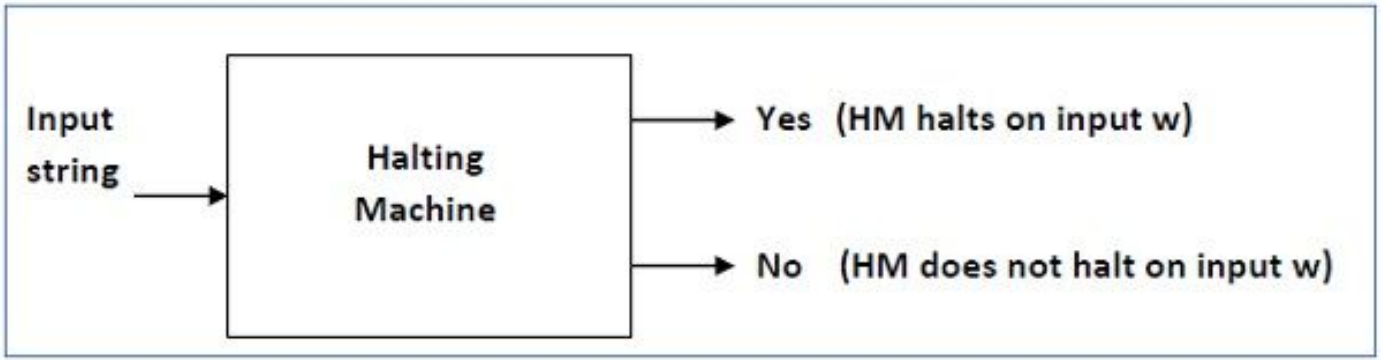
```
public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int a = scanner.nextInt();

        int b = scanner.nextInt();

        int c = scanner.nextInt();

        if (a + b + c != 0 && pow(a, 3) + pow(b, 3) == pow(c, 3)) {

                    // assert false

        }

    }
```

Φ = P will pass the assertion

Halting Problem

Given a computer program and an input, determine whether the program will finish running or continue to run forever.

Proved **undecidable** by Alan Turing in 1936

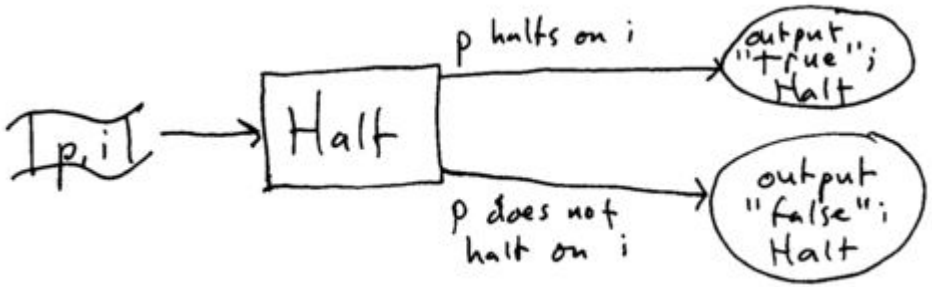


Halting Problem

Proof by contradiction

Assume that we have a program (`halt`) that can solve the halting problem. It takes as input a program **`p`** and an input **`i`**

If **`p`** halts on **`i`**, `then halt` will return true. False otherwise.



Halting Problem

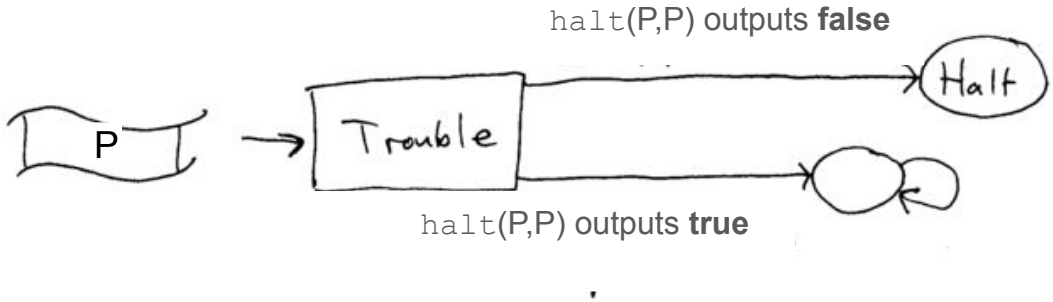
Input `i` could be a program itself!

Let's construct a new program `NEWHALT`:

```
if halt(P,P):
        return true;
else
        return false;
```

And another program `trouble` which does the opposite of `newhalt`:

```
if NEWHALT(P):
        loop forever
else halt;
```



Halting Problem

What happens when we call `trouble` on itself?

`trouble`

```
if NEWHALT(P):
        loop forever
else halt;
```

`NEWHALT`

```
if halt(P,P):
    return true;
else
    return false;
```

- Inside NEWHALT(`trouble`), we call halt(`trouble`, `trouble`).
- If `trouble` halts when fed to `NEWHALT`, halt will return true and `trouble` will loop forever
- If `trouble` doesn't halt when fed to `NEWHALT`, then the call to `trouble` does halt.
- `trouble` can neither halt nor loop forever: **Contradiction!**

Halting Problem

The undecidability of the halting problem has several important consequences. The main consequence is that, in general, it is impossible to predict exactly what a program will do when it is executed. This problem may be stated as follows:

Non Trivial properties of programs are undecidable.

Undecidability of Program Analysis

In summary, it's impossible!

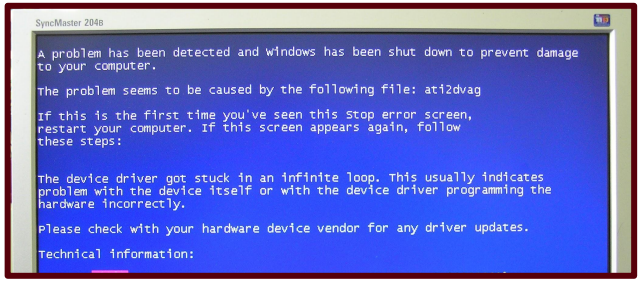
Sound + Complete + Terminating

Soundness - if a bug is present, the analysis will report it.
Completeness - if the analysis reports a bug it is truly a bug.

This is a bit sad, but also motivates the rich diversity of program analysis techniques in theory and practice with their tradeoffs.

This brings us to the landscape of program analysis techniques.

Program analysis success stories

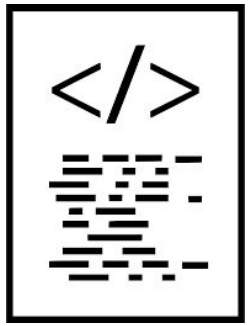


Astrée

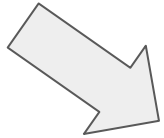
Zoncolan

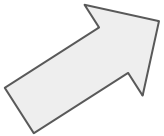
Ingredients of a Program Analysis

Program Implementation

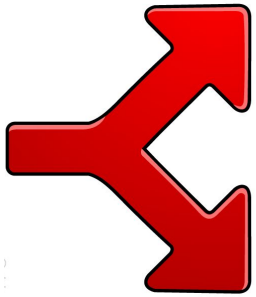


Program Specification

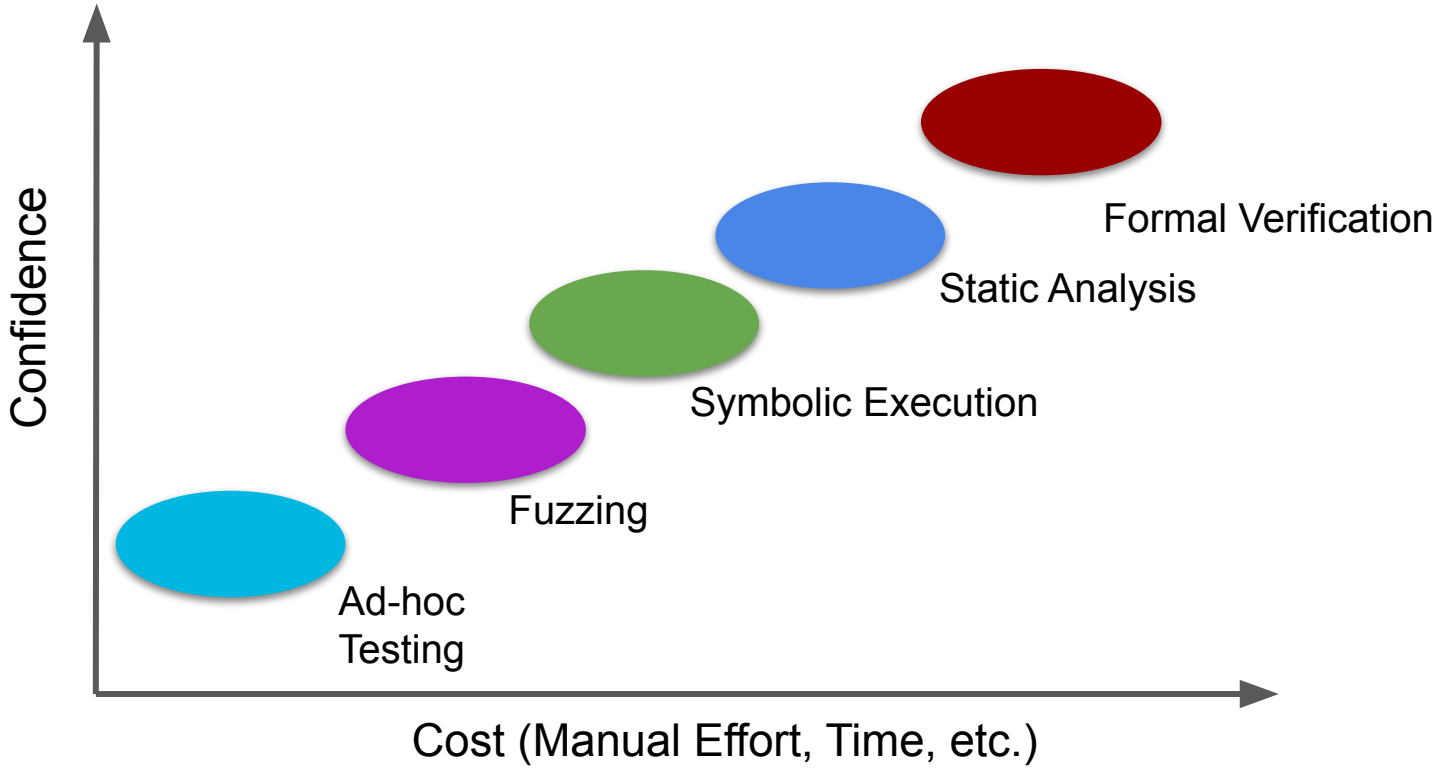




Analysis Algorithm



Landscape of Program Analysis Techniques



Landscape of Program Analysis Techniques

Program analysis can be broadly classified as static or dynamic

Static analysis - type checking (compiler), a human reading the code, chatGPT

Dynamic analysis - unit testing, autograding

Summary

- Software bugs can lead to disaster!
- Unfortunately we can’t automatically predict a program’s behavior
  - Halting problem
- In this class we will learn techniques which balance tradeoffs for effective program analysis algorithms

- Do the reading (on website)
- Sign up for gradescope and piazza