CSC2/455 Software Analysis and Improvement
Introduction

Sreepathi Pai

URCS

January 16, 2019

Outline

Outline

What does this Python program do?

```
for i in range(n):
    print("boom!")
```

Strategy: Run (or Interpret) the program

- Running the program and observing what it does is a perfectly reasonable way of analyzing a program
  - Maybe run it in a simulator/VM or interpreter
- What problems do you anticipate with this strategy?

Some potential problems

- Too long
- Infinite loop (aka non-termination)
- Number of inputs may be infinite!
  - Behaviour may depend on input

What does this program do?

```
def collatz(n): # n is a positive integer
    while n > 1:
        print(n)
        if n % 2 == 0:
            n = n // 2 # integer division
        else:
            n = 3 * n + 1

    for i in range(n):
        print("boom!")
```

Some runs: n=5

```
5
16
8
4
2
1
boom!
```

Some runs: n=12

```
12
6
3
10
5
16
8
4
2
1
boom!
```

Analyzing this program

- This program will print only 1 `boom!`
- *If* the loop terminates
  - Only if n is always reduced to 1
  - Is it always? ($500 reward!)
- Can we determine if the loop terminates?
  - For any n?
  - For a fixed n?

Undecidability: The Halting Problem

- In general, an algorithm cannot determine if a program will terminate on a given input
- What does this imply for program analysis?
  - End of this course?

Program Analysis

- Program analysis needed for optimization ("making programs faster")
  - Reducing number of operations
  - Substituting cheaper operations
  - Increasing parallelism of operations
- Also need for verification ("security")
  - Will this program crash for any input?
  - Will this program leak memory? (`malloc` but no corresponding `free`)
  - Will this program read another user's files?
  - Can a program be subverted to obtain `root` access?
- Computers everywhere, such questions far more important now!
  - 4.4B people have smartphones!

Mission Impossible?

- No general technique to analyze programs
  - Many different approaches
- We will study many of these
  - Basic
  - Advanced
  - Recent advances

Some things program analysis makes possible

- Fast Javascript
  - pioneered by Google's V8
- Safe in-kernel execution of user-provided code
  - Linux eBPF
  - pioneered by Sun's DTrace
- Safe systems programming languages
  - Rust
- Airplanes in the sky

Outline

People

- Instructor: Dr. Sreepathi Pai
  - E-mail: sree@cs.rochester.edu
  - Office: Wegmans 3409
  - Office Hours: By appointment, but I have an open door policy (10AM–3PM)
- TAs:
  - Alan Chiu
  - Chi Chun Chen

Places

- Class: Hylan 202
  - M,W 1025–1140
- Course Website
  - `https://cs.rochester.edu/~sree/courses/csc-255-455/spring-2019/`
- Blackboard
  - Announcements, Assignments, etc.
- Piazza
  - ?

References

- Four textbooks
  - Cooper and Torczon, *Engineering a Compiler*
  - Aho, Lam, Ullman, Sethi, *Compilers: Principles, Techniques and Tools*
  - Allen and Kennedy, *Optimizinig Compilers for Modern Architectures*
  - Muchnick, *Advanced Compiler Design and Implementation*
  - Research Papers
- This course requires a lot of reading!
  - Books and materials have been placed on reserve
  - Some online, some in Carlson Library
- See Blackboard for information on accessing Reserves

Grading

- Homeworks: 20%
- Assignments: 55% (4)
- Exams: 25% (midterm + final)
- Graduate students should expect to read a lot more, and work on harder problems.

There is no fixed grading curve.
See course website for late submissions policy.

Academic Honesty

- Unless explicitly allowed, you may not show your code to other students
- You may discuss, brainstorm, etc. with your fellow students but all submitted work must be your own
- All help received must be acknowledged in writing when submitting your assignments and homeworks
- All external code you use must be clearly marked as such in your submission
  - Use a comment and provide URL if appropriate
- If in doubt, ask the instructor

All violations of academic honesty will be dealt with strictly as per UR's Academic Honesty Policy.