Introduction to Software Analysis

Introduction

1. Theory and Practice of Software Analysis
   - Lies at the heart of many software development processes
     - Diagnosing bugs
     - Testing
     - Debugging
   - Techniques and skills to enhance your existing programming skills and build better software

Why Take This Course?

1. Bill Gates Quote
   - "We have as many testers as we have developers. And testers spend all their time testing, and developers spend half their time testing. We're more of a testing, a quality software organization than we're a software organization"

- Why Take This Course?
  - Learn methods to improve software quality
    - Reliability, security, performance, etc.
  - Become a better software developer/tester
  - Build specialized tools for software diagnosis and testing

Ariane Rocket Disaster

1. Ariane rocket exploded soon after launching

Ariane Rocket Disaster Post-Mortem

1. Caused due to a numeric overflow error
   - Attempt to fit a 64-bit format data in 16-bit space
2. Cost
   - $100M's for loss of mission
   - Multi-year setback to the Ariane program

Security Vulnerabilities

1. Exploits of errors in programs
   - Widespread problem
     - Moonlight Maze (1998)
     - Code Red (2001)
     - Titan Rain (2003)
     - Stuxnet Worm (2010)
   - Problem is getting worse with the advent of smartphones

What is Program Analysis?

1. Program Analysis
   - Body of work to automatically discover useful facts about programs
   - Broadly classified into three kinds:
     - Dynamic: Class of runtime analyses; discover information by running the program and observing its behavior
     - Static: Class of compile-time analyses; discover information by inspecting the source code or binary code of the program
     - Hybrid: Combine dynamic and static techniques

Dynamic Program Analysis

1. Dynamic program analysis infers facts of a program by monitoring its runs
   - Purify: Array bound checking (C/C++)
   - Valgrind: Memory leak detection (x86 binaries)
   - Eraser: Data race detection (concurrent programs)
   - Daikon: Finding likely invariants
     - Invariant: Program fact true in every run of the program

Static Program Analysis

1. Static program analysis infers facts of a program by inspecting its code
   - Lint, FindBugs, Coverity: Suspicious error patterns
   - Microsoft SLAM: Checking API usage rules
   - Facebook Infer: Memory leak detection
   - ESC/Java: Verifying invariants

Program Invariants

1. An invariant at the end of the program is (z == c) for some constant c. What is c?
   - c == 42

```
int p(int x) { return x * x; }
void main() {
    int z;
    if (getc() == 'a')
        z = p(6) + 6;
    else
        z = p(-7) - 7;

    if (z != 42)
        disaster();
}
```

Discovering Invariants 1

1. Dynamic Invariants
   - A program can have loops or recursion which can lead to arbitrarily many dynamic paths
   - Since dynamic analysis discovers information by running the program a finite number of times, it cannot in general discover information that requires observing an unbounded number of paths
   - Dynamic analysis tool like Daikon can, at best, detect likely invariants
     - Still useful
     - Daikon can rule out entire classes of invariants, even by observing a single run

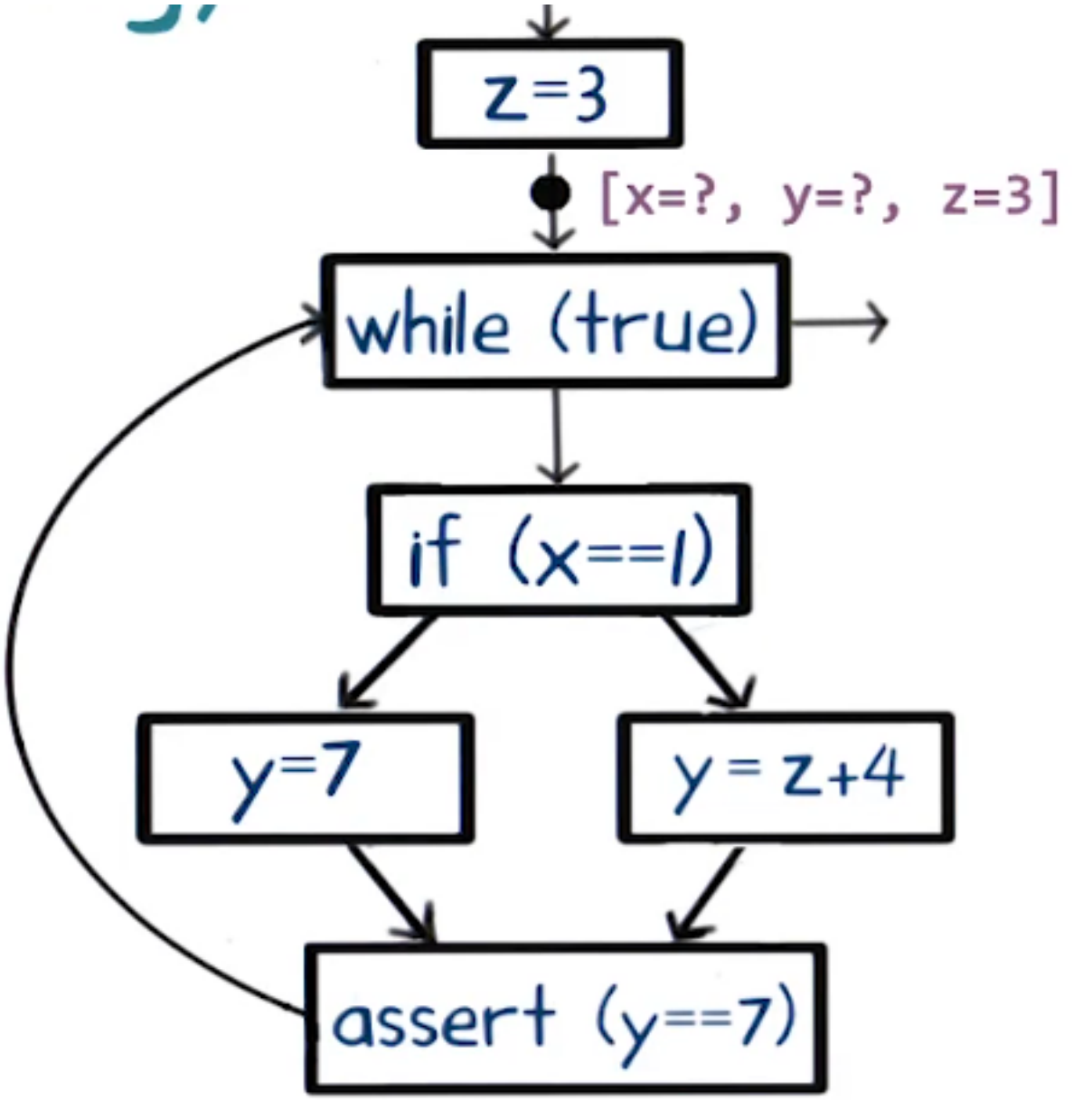
Discovering Invariants 2

1. Dynamic analysis
   - (z == 42) might be an invariant
   - (z == 30) is definitely not an invariant
2. Static analysis
   - (z == 42) is definitely an invariant
   - (z == 30) is definitely not an invariant

Terminology

1. Static analysis

- Typically operates on a suitable intermediate representation of the program
  - Control-flow graph: Summarizes the flow of control in all possible runs of the program
    - Each node represents a unique statement in the program
    - Each edge represents a possible successor
- Abstract state: Static analysis tracks the constant values of the three variables in this program at each program point
- Concrete state: Tracks the actual values in a particular run
- Static analysis operates on abstract states since it isn't running the program
  - Each abstract state summarizes a set of concrete states
  - Ensure termination of the static analysis
- Static analysis sacrifices completeness, but is sound



Control Flow Graph

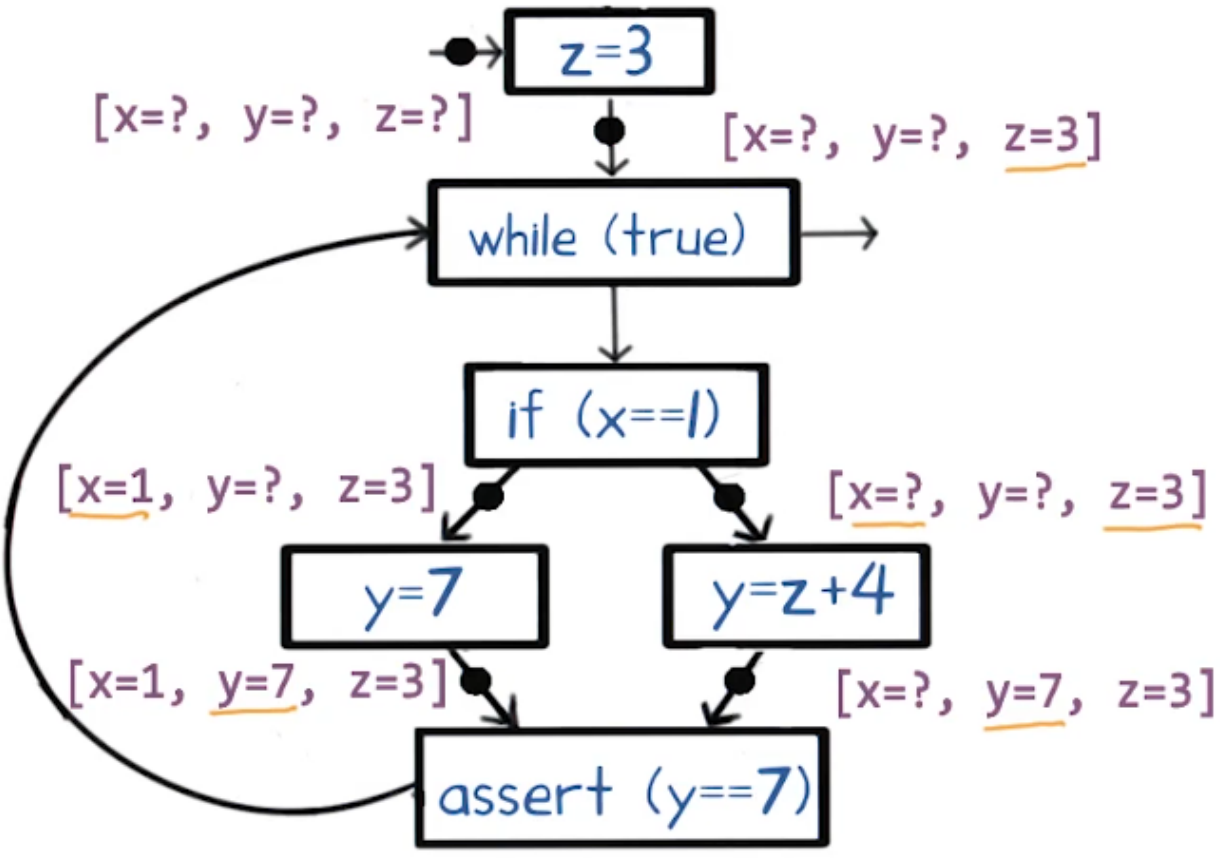
Example Static Analysis Problem

1. Find variables that have a constant value at the given program point

```
void main() {
    z = 3;
    while (true) {
        if (x == 1)
            y = 7;
        else
            y = z + 4;
        assert(y == 7);
    }
}
```

Iterative Approximation 1

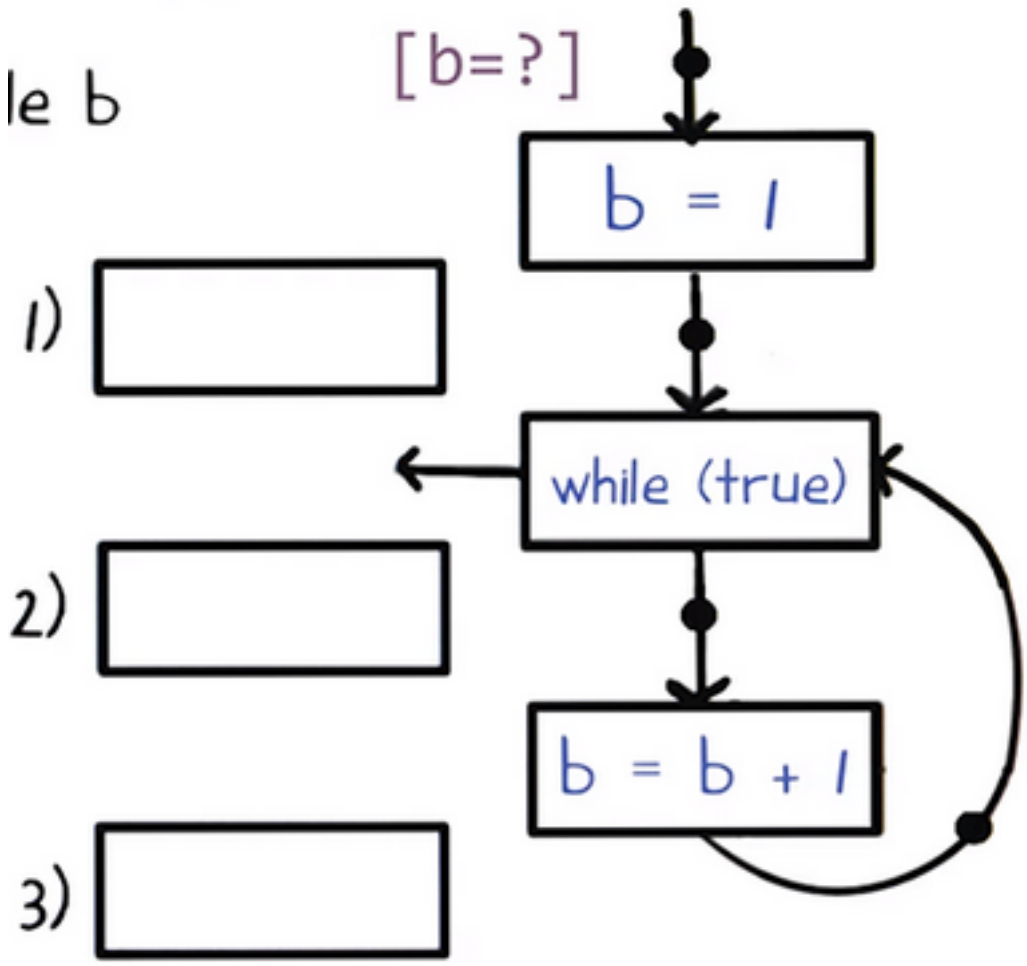
1. Begin with unknown values for x, y, z
   - At each node in the control flow graph, the static analysis updates its knowledge about the values of each variable at each program point
2. Iterative approximation implies that, in general, the analysis might need to visit the same program point multiple times
   - Due to the presence of loop



Iterative Approximation

Iterative Approximation 2

1. Fill in the final value of variable b that the analysis infers at:
   - The loop header
     - b == 1
   - Entry of loop body
     - b == ?
   - Exit of loop body
     - b == ?



Iterative Approximation Quiz

Dynamic vs. Static Analysis

1. Match each box with its corresponding feature
   - A: Unsound (may miss errors)
   - B: Proportional to program's execution time
   - C: Proportional to program's size
   - D: Incomplete (may report spurious errors)
2. Static analysis may produce false positives
3. Dynamic analysis may produce false negatives

	Dynamic	Static
Cost	B	C
Effectiveness	A	D

Undecidability of Program Properties

1. Can program analysis be sound and complete?
   - Not if we want it to terminate!
   - Questions like "is a program point reachable on some input?" are undecidable
   - Designing a program analysis is an art
     - Tradeoff between termination, soundness, and completeness
     - Dictated by consumer

Who Needs Program Analysis?

1. Three primary consumers of program analysis:
   - Compilers
   - Software quality tools
   - Integrated Development Environments (IDEs)

Compilers

1. Compiler: Bridge between high-level languages and architectures
   - Use program analyses to generate efficient code
     - Example: On the example where z == 42, just return 42
       * Runs faster
       * More energy efficient
       * Smaller in size

Software Quality Tools

1. Primary focus of this course
   - Tools for testing, debugging, and verification
   - Use program analysis for:
     - Finding programming errors
     - Proving program invariants
     - Generating test cases
     - Localizing causes of errors

Intergrated Development Environments

1. Use program analysis to help programmers:
   - Understand programs
   - Refactor programs
     - Restructuring a program without changing its external behavior
   - Useful in dealing with large, complex programs
   - Examples: Eclipse and Microsoft Visual Studio

Conclusion

1. Program analysis
   - A process for automatically discovering useful facts about programs
2. Dynamic vs static analysis

- Dynamic works by running the program, static works by inspecting the source code

3. Program invariants
   - Dynamic analysis can discover likely invariants
   - Static analysis can prove invariance
4. Iterative approximation method for static analysis
5. Undecidability: Program analysis cannot ensure termination + soundness + completeness
6. Who needs program analysis?