Code Inspection

Lecture 4a

Agenda

- Computer Program Analysis
- Code Inspection
- Advantages and Limitations of Code Inspection
- A Critique of Static Analysis

Computer Program Analysis

Computer Program Analysis

- Computer program analysis is the process of automatically analysing the **behaviour** of computer programs.
- The main applications of program analysis are program optimization and program correctness.
- There are **two** main **approaches** in program analysis:
  - **static** program analysis
  - **dynamic** program analysis

Static Analysis

- A couple of definitions:
  - **Static analysis** is the process of **examining the** source **code** of a program with the intention of **discovering** various **weaknesses without** having to actually **execute** it.
  - **Static analysis** is the process of automatically **analysing run-time properties** of a computer program **without executing** it.
- It is done after coding and before executing unit tests. So, it identifies defects before the program runs.

Static Analysis

- Static analysis can be done by a machine to automatically “walk through” the source code and detect noncomplying rules.
- It can also be performed by a person that reviews the code to ensure proper coding standards and conventions are used to construct the program. This is called **Code Review** and is done by a peer developer.
  - Someone other than the developer who wrote the code.

Dynamic Analysis

- Dynamic analysis, on the other hand, **analyses programs by executing** an instrumented program and generating some form of trace.
- It **identifies vulnerabilities** in a **runtime** environment.
- It allows for analysis of applications in which **you do not have access to the actual code**.
  - Performance analysis
  - Security analysis
  - Memory error detection
  - Concurrency errors

Code Inspection

Code Inspection

- **Code inspection**, **static code analysis**, **static analysis** and **static program analysis are terms used interchangeably.**
- As we said before, **code inspection** involves no dynamic execution of the software under test and can detect possible defects in an early stage, **before running the program**.
- Quality attributes that can be the focus of static analysis are:
  - Reliability
  - Maintainability
  - Testability
  - Reusability
  - Portability
  - Efficiency

Code Inspection: Uses

- A (non-exhaustive) list that contains areas where static analysis might be used successfully:
  - **Improper resource management**: Resource leaks of various kinds such as dynamically **allocated memory that is not freed**, files, sockets etc. which are not properly deallocated when no longer used.
  - **Illegal operations**: Things like division by zero, over- or underflow in arithmetic expressions, addressing arrays out of bounds, dereferencing of null pointers, etc.
  - **Dead code and data**: Code and data that cannot be reached or is not used.
  - **Incomplete code**: The use of uninitialized variables, functions with unspecified return values, incomplete branching statements (for example no else branch in conditional statement), non-terminating loops and non-terminating calls.

How does static analysis work?

- Static analysis tools receive **code as input**, **build a model** that represents the program, perform some sort of **analysis** in combination with **knowledge** about what to search for (e.g. security) and finally present the result to the user.

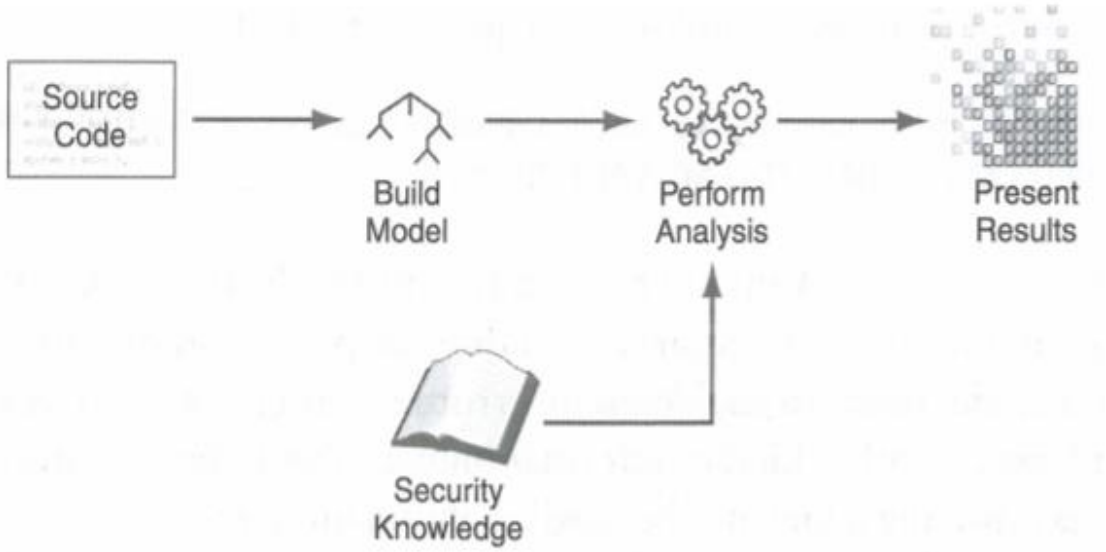


Figure 2.2. Generalized view of the process of a static analysis

How does static analysis work?

- There are several static analysis techniques to build the model of the program. Some of them are:
  1. **Data Flow Analysis**: The program is modelled as a graph. The nodes are the elementary blocks and the edges describe how control might pass from one elementary block to another.

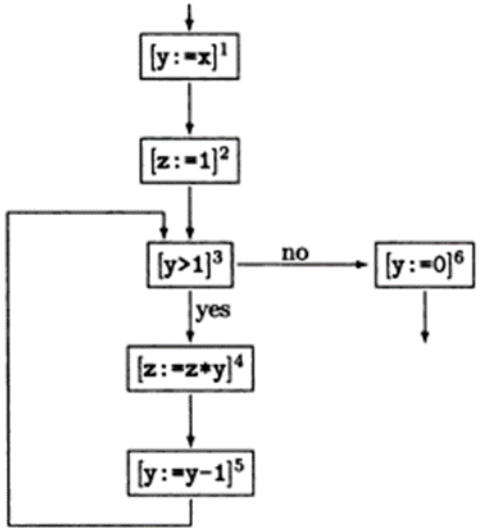
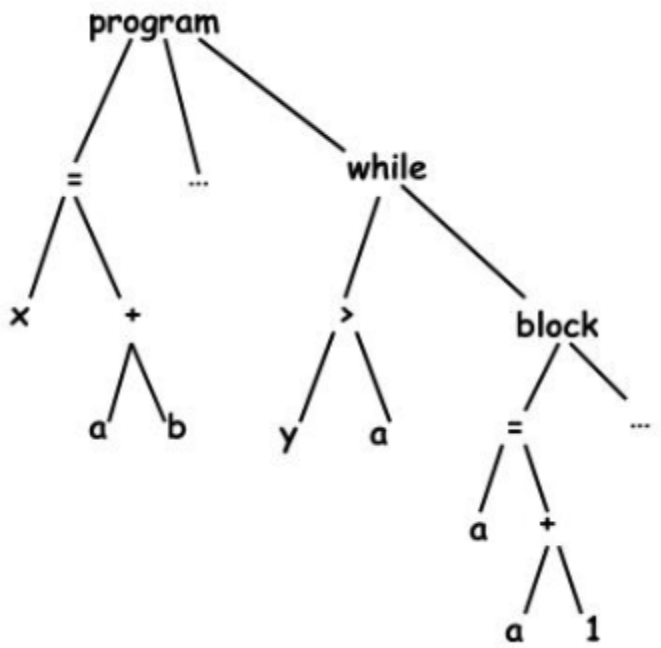


Figure 1.2: Flow graph for the factorial program.

How does static analysis work?

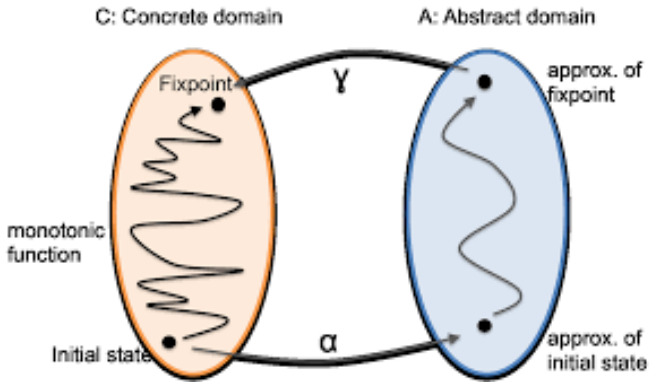
- Other static analysis techniques:
  2. **Syntactic Pattern Matching**: A **parser** takes as input the program source code and outputs a data structure called **abstract syntax tree**.

```
x := a + b;
y := a * b
While (y > a){
  a := a +1;
  x := a + b
}
```



How does static analysis work?

- Other static analysis techniques:
  3. **Abstract Interpretation**: A concrete program, its concrete domain, and semantics operations are replaced by **an approximate program** in some abstract domain and abstract semantic operations.



Advantages and Limitations of Code Inspection

Advantages of Code Inspection

- It can find **weaknesses** in the code at the **exact location**.
- It can be conducted by trained software assurance developers who fully understand the code.
- Source code can be easily understood by other or future developers.
- It allows a quicker turn around for fixes.
- **Weaknesses** are found **earlier** in the development life cycle, reducing the cost to fix.

Advantages of Code Inspection

- Less defects in later tests.
- Unique defects are detected that **cannot or hardly be detected** using dynamic tests.
  - Unreachable code
  - Variable use (undeclared, unused)
  - Uncalled functions
  - Boundary value violations

Limitations of Code Inspection

- It is time consuming if conducted manually.
- Automated tools produce **false positives and false negatives**.
- There are not enough trained personnel to thoroughly conduct static code analysis.
- Automated tools can provide a false sense of security that everything is being addressed.
- Automated tools **only as good as the rules** they are using to scan with.
- It **does not** find **vulnerabilities introduced** in the **runtime** environment.

Comparison of static analysis and testing

Table 1 Comparison of static analysis and testing

Static analysis	Testing
Can be applied without executing the code	Can be applied only by executing the code
Can be applied early in the development process	Is applied late in the development process
Results do not depend on inputs	Results depend on inputs
Results can be generalized for future executions	Results cannot be generalized for future executions
Less costly	Very costly
Very fast process	Slow process
False-positive rate is very high	False-positive rate is very less
Approximations are used	Exact results are used
Cannot be used for functional correctness of program	Can be used for functional correctness of programs

A Critique of Static Analysis

A Critique of Static Analysis

- **Precision**
  - Static analysis can be imprecise because it makes **assumptions or approximations** about the run-time behaviour of a program. Therefore, it suffers a lot from **false positives**.
- **Efficiency**
  - Efficiency is an important attribute as it directly links to the cost of the analysis.
  - Efficiency is also related to precision. Static analysis trades off precision over efficiency; that is, in order to provide analysis results faster, it uses approximation or abstraction mechanisms which lead to less precise results.

A Critique of Static Analysis

- **Coverage**
  - It implies the total number of **valid** execution paths analysed.
  - By its very nature, static analysis provides high coverage as it does not depend on specific input stimuli. Therefore, as compared to its dynamic counterparts, static analysis provides **high coverage** of code for analysis purpose.
- **Scalability**
  - Static analysis **scales well** – can be run on lots of software and can be run repeatedly (as with nightly builds or continuous integration).

Next On

Take Away Points

- What is program analysis?
  - Which are the approaches to it?
- Alternative terms for code inspection
- Which are the areas where code inspection is used?
- Advantages and limitations of code inspection

Further Reading

- Patrik Hellström, “Tools for static code analysis: A survey”
  - Chapter 2: Static Analysis
- Anjana Gosain and Ganga Sharma, “Static Analysis: A Survey of Techniques and Tools”
- Amir Ghahrai, “Static Analysis vs Dynamic Analysis in Software Testing”
  - https://devqa.io/static-analysis-vs-dynamic-analysis-software-testing/
- Fleming Nielson et al., “Principles of Program Analysis”
  - Chapter 1: Introduction

Next Lecture

- More about code inspection