Software Re-Engineering

Lecture: 12

Dr. Syed Muazzam Ali Shah
Department of Software Engineering
National University of Computer &
Emerging Sciences
muazzam.ali@nu.edu.pk

Contact#: *(021) 111-128-128 Ext. 222*
Personal Website: *https://shorturl.at/dfhyN*, ***Official web-page:*** *https://shorturl.at/Xgty3*, ***Google Scholar:*** *https://shorturl.at/ejrv2*

Sequence [**Todays Agenda**]

Content of Lecture

Reverse Engineering – Techniques

- Lexical Analysis
- **Syntactic Analysis**
- Control Flow Analysis
- Data Flow Analysis
- Program Slicing
- Visualization
- Program metrics

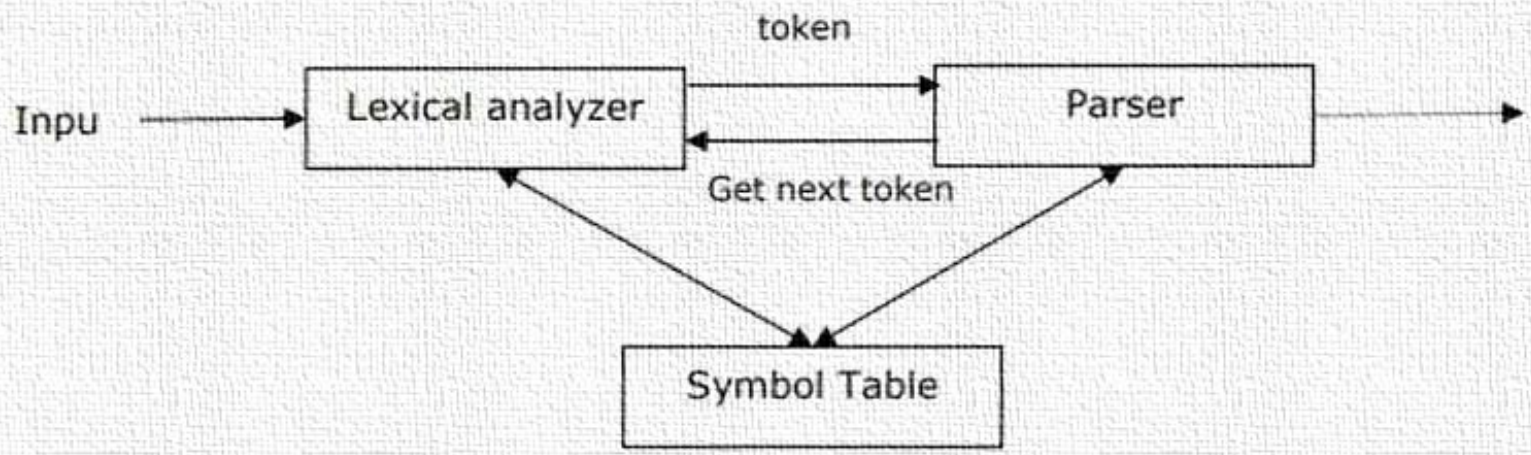
Reverse Engineering – Techniques

Syntactic Analysis

- When an input string (source code or a program in some language) is given to a compiler:
  - The compiler processes it in several phases.
  - Starting from lexical analysis (scans the input and divides it into tokens) to target code generation.

Reverse Engineering – Techniques

Syntactic Analysis



Reverse Engineering – Techniques

Syntactic Analysis

- Syntactic Analysis or Syntax Analysis or Parsing is the second phase, i.e. after lexical analysis.
- It checks the **syntactical structure** of the **given input**, i.e. whether the **given input** is in the **correct syntax** (of the language in which the input has been written) or not.
- It does so by building a data structure, called a **Parse tree** or **Syntax tree**.
- The **parse tree** is constructed by using the **pre-defined Grammar of the language** and the **input string**.
- If the **given input string** can be **produced** with the help of the **syntax tree** (in the derivation process), the **input string** is found to be in the **correct syntax**. if not, the error is reported by the syntax analyzer.

Reverse Engineering – Techniques

Syntactic Analysis

- Syntax/Syntactic analysis, also known as parsing, is a process in compiler design where the compiler checks if the source code follows the grammatical rules of the programming language.
- This is typically the second stage of the compilation process, following lexical analysis.
- The main goal of syntax analysis is to create a parse tree or abstract syntax tree (AST) of the source code:
  - Which is a hierarchical representation of the source code that reflects the grammatical structure of the program.

Reverse Engineering – Techniques

Syntactic Analysis

❖ Types of parsing algorithms used in syntax analysis:

⌗ **LL parsing:**

➢ **Read input left to right, produce leftmost derivation.**

- This is a top-down parsing algorithm that starts with the root of the parse tree and constructs the tree by successively expanding non-terminals.
- LL parsing is known for its simplicity and ease of implementation.

⌗ **LR parsing :**

➢ **Read input left to right, produce rightmost derivation.**

- This is a bottom-up parsing algorithm that starts with the leaves of the parse tree and constructs the tree by successively reducing terminals.
- LR parsing is more powerful than LL parsing and can handle a larger class of grammars.

Reverse Engineering – Techniques

Syntactic Analysis

❖ Types of parsing algorithms used in syntax analysis:

⌗ **LR(1) parsing :**

- This is a variant of LR parsing that uses lookahead to disambiguate the grammar.

⌗ **LALR parsing :**

- This is a variant of LR parsing that uses a reduced set of lookahead symbols to reduce the number of states in the LR parser.

Reverse Engineering – Techniques

Syntactic Analysis

Once the parse tree is constructed, the compiler can perform semantic analysis to check if the source code makes sense and follows the semantics of the programming language.

The parse tree or AST can also be used in the code generation phase of the compiler design to generate intermediate code or machine code.

Reverse Engineering – Techniques

Syntactic Analysis

❖ Features of syntax analysis:

⌗ **Syntax Trees:**

- Syntax analysis creates a syntax tree, which is a hierarchical representation of the code’s structure.
- The tree shows the relationship between the various parts of the code, including statements, expressions, and operators.

⌗ **Context-Free Grammar :**

- Syntax analysis uses context-free grammar to define the syntax of the programming language.
- Context-free grammar is a formal language used to describe the structure of programming languages.

Reverse Engineering – Techniques

Syntactic Analysis

❖ Features of syntax analysis:

⌗ Top-Down and Bottom-Up Parsing :

- Syntax analysis can be performed using two main approaches: top-down parsing and bottom-up parsing.
- Top-down parsing starts from the highest level of the syntax tree and works its way down,
- While bottom-up parsing starts from the lowest level and works its way up.

⌗ Error Detection:

- Syntax analysis is responsible for detecting syntax errors in the code.
- If the code does not conform to the rules of the programming language, the parser will report an error and halt the compilation process.

Reverse Engineering – Techniques

Syntactic Analysis

❖ Features of syntax analysis:

⌗ **Intermediate Code Generation:**

- Syntax analysis generates an intermediate representation of the code, which is used by the subsequent phases of the compiler.
- The intermediate representation is usually a more abstract form of the code, which is easier to work with than the original source code.

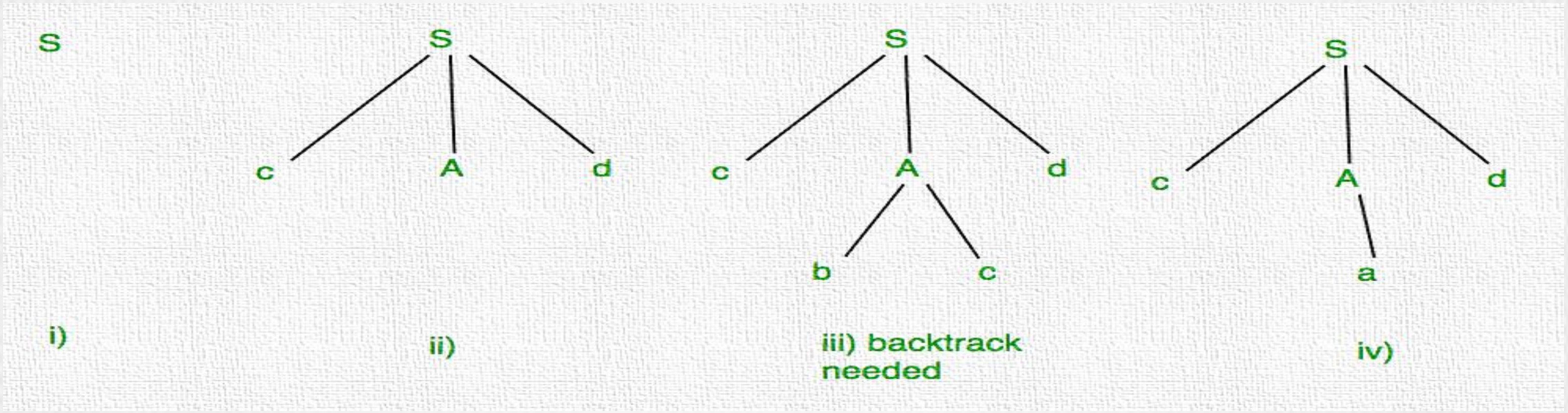
⌗ **Optimization:**

- Syntax analysis can perform basic optimizations on the code, such as removing redundant code and simplifying expressions.

Reverse Engineering – Techniques

Syntactic Analysis

- **Production rules:**
- **The Grammar for a Language consists of Production rules**
  - Example: Suppose Production rules for the Grammar of a language are:
    - S -> cAd
    - A -> bc|a
    - And the input string is "cad".



Reverse Engineering – Techniques

Syntactic Analysis

In step (iii) above, the production rule A->bc was not a suitable one to apply (because the string produced is "cbcd" not "cad"), here the parser needs to backtrack, and apply the next production rule available with A which is shown in step (iv), and the string "cad" is produced.

Thus, the given input can be produced by the given grammar, therefore the input is correct in syntax.

But backtrack was needed to get the correct syntax tree, which is really a complex process to implement.

Reverse Engineering – Techniques

Syntactic Analysis

❖ Advantages

⌗ **Structural validation:**

- Syntax analysis allows the compiler to check if the source code follows the grammatical rules of the programming language, which helps to detect and report errors in the source code.

⌗ **Improved code generation :**

- Syntax analysis can generate a parse tree or abstract syntax tree (AST) of the source code, which can be used in the code generation phase of the compiler design to generate more efficient and optimized code.

Reverse Engineering – Techniques

Syntactic Analysis

- Advantages

- **Easier semantic analysis:**
  - Once the parse tree or AST is constructed, the compiler can perform semantic analysis more easily, as it can rely on the structural information provided by the parse tree or AST.

Reverse Engineering – Techniques

Syntactic Analysis

❖ Disadvantages

⌗ **Complexity :**

- Parsing is a complex process, and the quality of the parser can greatly impact the performance of the resulting code.
- Implementing a parser for a complex programming language can be a challenging task, especially for languages with ambiguous grammars.

⌗ **Reduced performance: :**

- Syntax analysis can add overhead to the compilation process, which can reduce the performance of the compiler.

Reverse Engineering – Techniques

Syntactic Analysis

- Disadvantages

Limited error recovery:

- Syntax analysis algorithms may not be able to recover from errors in the source code, which can lead to incomplete or incorrect parse trees and make it difficult for the compiler to continue the compilation process.

Reverse Engineering – Techniques

Syntactic Analysis

Syntax analysis, also known as parsing, is a crucial stage in the process of compiling a program. Its primary task is to analyze the structure of the input program and check whether it conforms to the grammar rules of the programming language.

This process involves breaking down the input program into a series of tokens and then constructing a parse tree or abstract syntax tree (AST) that represents the hierarchical structure of the program.

Reverse Engineering – Techniques

Syntactic Analysis

❖ Steps:

⌗ **Tokenization:**

- The input program is divided into a sequence of tokens, which are basic building blocks of the programming language, such as identifiers, keywords, operators, and literals.

⌗ **Parsing :**

- The tokens are analyzed according to the grammar rules of the programming language, and a parse tree or AST is constructed that represents the hierarchical structure of the program.

⌗ **Error handling:**

- If the input program contains syntax errors, the syntax analyzer detects and reports them to the user, along with an indication of where the error occurred.

Reverse Engineering – Techniques

Syntactic Analysis

❖ Steps:

⌗ **Symbol table creation:**

- The syntax analyzer creates a symbol table, which is a data structure that stores information about the identifiers used in the program, such as their type, scope, and location.

⌗ **Prerequisite for subsequent phases:**

- The syntax analysis phase is essential for the subsequent stages of the compiler, such as semantic analysis, code generation, and optimization.
- If the syntax analysis is not performed correctly, the compiler may generate incorrect code or fail to compile the program altogether.

Thank You!