Software Re-Engineering

Lecture: 12

Dr. Imran Ali
School of Computing- Software Engineering
FAST-National University of Computer & Emerging Sciences, Karachi

Sequence [Todays Agenda]

Content of Lecture

- Syntactic Analysis

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.

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.

Syntactic Analysis

- Syntactic analysis is the process of identifying the grammatical structure of a sentence and the roles of its words.
- It involves understanding how words are related and how they combine to form phrases and clauses, ultimately revealing the sentence's meaning.

Types of Parsing Algorithms used in Syntactic Analysis

- LL parsing:
- Read input left to right, produce leftmost derivation.
- LL parsing is known for its simplicity and ease of implementation.
- LR parsing :
- Read input left to right, produce rightmost derivation.
- LR parsing is more powerful than LL parsing and can handle a larger class of grammars.

Types of Parsing Algorithms used in Syntactic 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.

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.

- <u>Top-Down and Bottom-Up Parsing:</u>
- 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.
- <u>Error Detection:</u>
- 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.

- 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.

Syntactic Analysis

- The breakdown of Syntactic Analysis is:
  1. Identifying Word Classes
  2. Identifying Phrases
  3. Identifying Clauses
  4. Understanding Syntactic Functions

1. Identifying Word Classes:

- The first step is to determine the grammatical category of each word (noun, verb, adjective, etc.).
- In the sentence "The quick brown fox jumps over the lazy dog," "The" is a determiner, "quick" and "lazy" are adjectives, "brown" is an adjective, "fox" and "dog" are nouns, "jumps" is a verb, and "over" is a preposition.

2. Identifying Phrases:

- Next, group words into phrases, which are units of meaning within a sentence.
- In the same sentence, "The quick brown fox" is a noun phrase, "jumps" is a verb phrase, and "over the lazy dog" is a prepositional phrase.

3. Identifying Clauses:

- Clauses are groups of words that contain a subject and a predicate (verb).
- In this example, the entire sentence "The quick brown fox jumps over the lazy dog" can be considered a clause.

4. Understanding Syntactic Functions:

- Syntactic analysis also determines the relationship between different parts of a sentence, like subject, verb, object, etc.
- In our example, "fox" is the subject (performing the action), "jumps" is the verb (action), and "lazy dog" is the object (receiving the action).

Example

- Let's analyze the sentence "Yesterday, John made Him happy."
- <u>Word Classes:</u>
  - Yesterday (adverb), John (noun), made (verb), Him (pronoun), happy (adjective).
- <u>Phrases:</u>
  - "Yesterday" is an adverbial phrase.
  - "John" is a noun phrase.
  - "made Him happy" is a verb phrase.
- <u>Functions:</u>
  - "John" is the subject.
  - "made" is the verb.
  - “Him" is the direct object.
  - "happy" is the complement.
  - "Yesterday" is an adverbial modifier.

Example

- **Lexical Analysis:**
- Perform the Lexical Analysis for the below source code fragment that must include:
  1. Line of Token,
  2. Token number,
  3. Token,
  4. Category.

```
1. int x = 10;
2. double y = 20.5;
3. x = x + 5;
4. if (x > y) {
5.     System.out.println("x is greater");
```

Line No	Token number	Token	Category	Line No	Token number	Token	Category
1	1	int	Keyword	4	17	if	Keyword
1	2	x	Identifier	4	18	(	Separator (Parenthesis Open)
1	3	=	Operator (Assignment)	4	19	x	Identifier
1	4	10	Number (Integer)	4	20	>	Operator (Relational Greater than)
1	5	;	Separator (Semicolon)	4	21	y	Identifier
2	6	double	Keyword	4	22	)	Separator (Parenthesis Close)
2	7	y	Identifier	4	23	{	Separator (Brace Open)
2	8	=	Operator (Assignment)	5	24	System	Identifier (Class name)
2	9	20.5	Number (Floating Point)	5	25	.	Separator (Dot Operator)
2	10	;	Separator	5	26	out	Identifier (Field)
3	11	x	Identifier	5	27	.	Separator (Dot Operator)
3	12	=	Operator	5	28	println	Identifier (Method)
3	13	x	Identifier	5	29	(	Separator (Parenthesis Open)
3	14	+	Operator (Addition)	5	30	"x is greater"	String Literal
3	15	5	Number (Integer)	5	31	)	Separator (Parenthesis Close)
3	16	;	Separator	5	32	;	Separator (Semicolon)

Abstract Syntax Tree (AST) Example

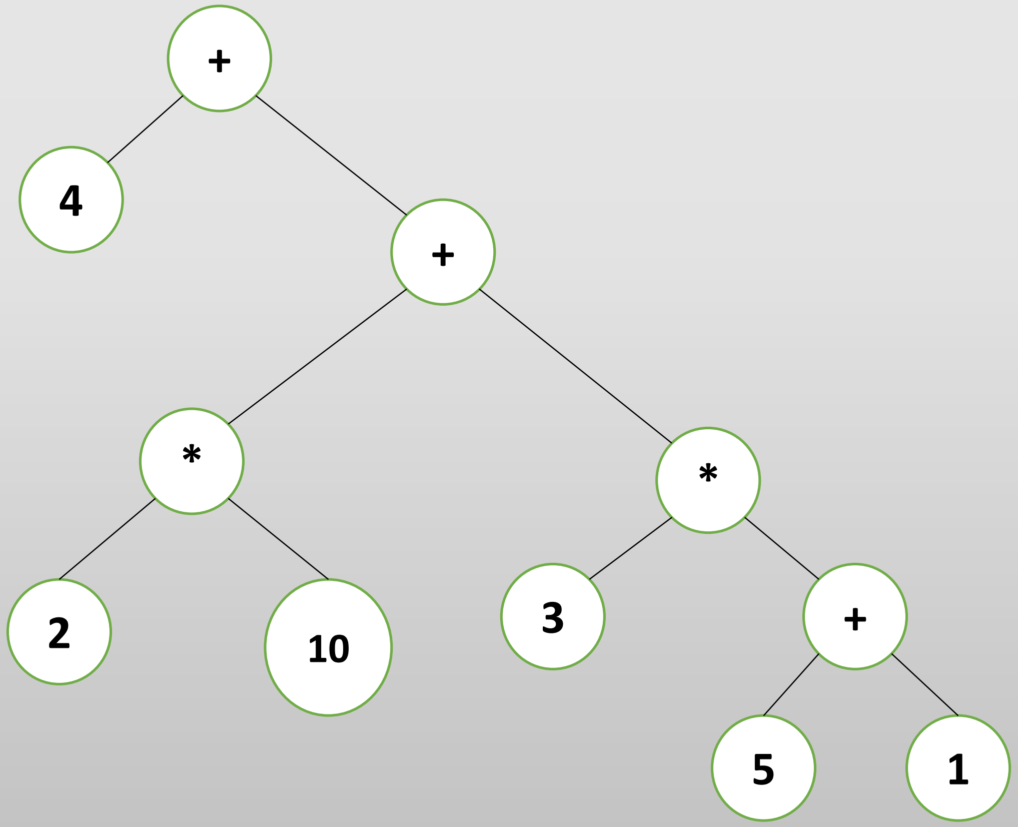
- Consider the given expression and construct an abstract syntax tree such that the leaf nodes represent the numbers while the intermediate nodes represent the operators:

4 + 2 * 10 + 3 * (5 + 1)

- <u>Step-by-step:</u>
- First, remember operator precedence:
- * (multiplication) has higher precedence than + (addition).
- Parentheses () must be evaluated first.
- So the order is:
- (5 + 1) first.
- Then all * multiplications.
- Then the final + additions.

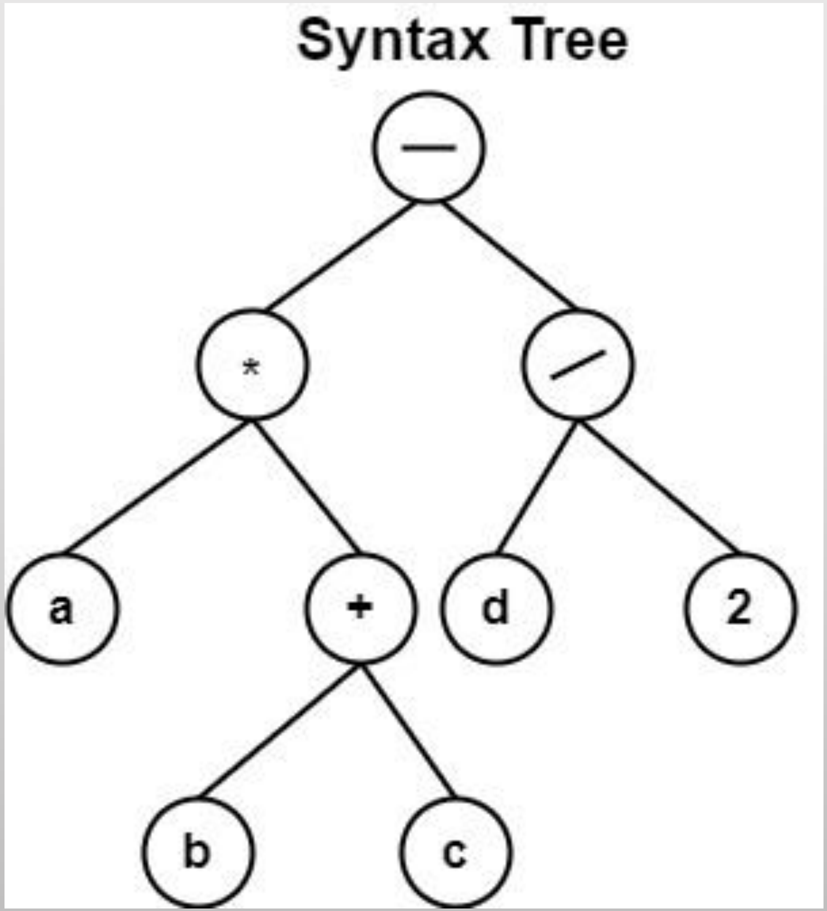
- **Syntax Tree:**

4 + 2 * 10 + 3 * (5 + 1)



Abstract Syntax Tree (AST) Example

a * (b + c) – d /2



- 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.
- 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.

- 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.
- Limited error recovery:
- Syntax analysis algorithms may not be able to recover from errors in source code.

Thank You!