Lexical and Syntax Analysis in Compiler Design

Vishal Trivedi
Gandhinagar Institute of Technology, Gandhinagar, Gujarat, India

Abstract — This Research paper gives brief information on how the source program gets evaluated in Lexical analysis phase of compiler and Syntax analysis phase of compiler. In addition to that, this paper also explains the concept of Compiler and Phases of Compiler. Mainly this paper concentrates on Lexical analysis and Syntax analysis.

Keywords — Token, Lexeme, Identifier, Operator, Operand, Sentinel, Prefix, Derivation, Kleene closure, Positive closure, Terminal, Production rule, Non-terminal, Sentential.

I. INTRODUCTION

Whenever we create a source code and start the process of evaluating it, computer only shows the output and errors (if occurred). We don't know the actual process behind it. In this research paper, the exact procedure and step by step evaluation of source code in Lexical and Syntax Analysis are explained. In addition to that touched topics are Index Terms, Compilers, Phases of Compiler, Operations on grammar, Lexical analysis, Roll of Scanner, Finite automata, Syntax analysis, Types of Derivation, Ambiguous grammar, Left recursion, Left factoring, Types of Parsing, Top Down Parsing, Bottom Up Parsing, Error Handling.

II. INDEX TERMS

Token refers to sequence of character having a collective meaning. Token describes the class or category of input string. Typical Tokens are Identifiers, Operators, Special symbols, Constants etc. Pattern refers to the set of rules associated with a token. Lexeme refers to the sequence of characters in source code that are matched with the pattern of tokens. Example: int, i, num etc. Sentinel refers to the end of buffer or end of token. Regular expressions used to construct finite automata which is used to Token recognition.

III. COMPILERS

Compiler reads whole program at a time and generate errors (if occurred). Compiler generates intermediate code in order to generate target code. Once the whole program is checked, errors are displayed. Example of compilers are Borland Compiler, Turbo C Compiler. Generated target code is easy to understand after the process of compilation. The process of compilation must be done efficiently. There are mainly two parts of compilation process.

1. Analysis Phase: This phase of compilation process is machineindependent. The main objective of analysis phase is to divide to source code into parts and rearrange these parts into meaningful structure. The meaning of source code is determined and then intermediate code is created from the source program. Analysis phase contains mainly three sub-phases named lexicanalysis, syntaxanalysis and semanticanalysis.

2. Synthesis Phase: This phase of compilation process is machinedependent. The intermediate code is taken and converted into an equivalent target code. Synthesis phase contains mainly three sub-phases named intermediatecode, codeoptimization and codelistgeneration.

IV. PHASESOFCOMPILER

As mentioned above, compiler contains lexical analysis, syntax analysis, semantic analysis, intermediate code, code optimization and code generation phases.
V. OPERATIONS

- Є refers to Empty string.
- Λ or Ø refer to Empty set of string.
- | s | refers to Length of a string.
- Union of L and M written as L U M or L + M refer to \{s | s is in L or s is in M\}.
- Concatenation of L and M written as LM refers to \{st | s is in L and t is in M\}.
- Kleene closure of L written as L* refers to Zero or More occurrences of L.
- Positive closure of L written as L+ refers to One or More occurrences of L.

VI. LEXICAL ANALYSIS

- Lexical Analysis is first phase of compiler.
- Lexical Analysis is also known as Linear Analysis or Scanning.
- First of all, lexical analyzer scans the whole program and divide it into Token. Token refers to the string with meaning. Token describes the class or category of input string. Example: Identifiers, Keywords, Constants etc.
- Sentinel refers to the end of buffer or end of token.
- Pattern refers to set of rules that describes the token.
- Lexemes refers to the sequence of characters in source code that are matched with the pattern of tokens. Example: int, i, num etc.

- There are two pointers in lexical analysis named Lexemepointer and Forwardpointer.
- In order to perform tokenrecognition, RegularExpressions are used to construct Finiteautomata which is separate topic itself.
- Input is sourcecode and output is token.
- Consider an Example:
  
  Input: a=a+b*c*2;
  
  Output: Tokens or tables of tokens

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>a</td>
</tr>
<tr>
<td>+</td>
<td>b</td>
</tr>
<tr>
<td>*</td>
<td>c</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

VII. ROLL OF SCANNER

The lexical analyzer is the first phase of compiler. It’s main task is to read the input characters and produces a sequence of tokens as output that parser uses for syntax analysis.

VIII. FINITE AUTOMATA

We compile a regular expression into a recognizer by constructing a generalized transition diagram called a Finiteautomaton. A finite automata or finite statemachine is a 5-tuple (S, Σ, S₀, F, δ) where S is finite set of states, Σ is finite alphabet of input symbol, S₀ is initial state, F is set of accepting states, δ is a transition function. There are two types of finite automata.

[1] Deterministic finite automata (DFA) :

For each state, DFA has exactly one edge leaving out for each symbol. In the theoryofcomputation, a branch of theoretical computer science, a deterministicfiniteautomaton also known as a deterministicfiniteacceptor.

Deterministicfinitestatemachine(DFSM)is a finite-state machine that accepts and rejects strings of symbols and only produces a unique computation of the automaton for each input string. Deterministic refers to the
uniqueness of the computation.


There are no restrictions on the edges leaving a state. There can be several with the same symbol as label and some edges can be labeled with ε. A nondeterministic finite automaton (NFA) or nondeterministic finite state machine does not need to obey these restrictions. In particular, every DFA is also an NFA. Sometimes the term NFA is used in a narrower sense, referring to a NDFA that is not a DFA.

IX. SYNTAX ANALYSIS

Syntax analysis is also known as syntactical analysis or parsing or hierarchical analysis.

Syntax refers to the arrangement of words and phrases to create well-formed sentences in a language.

Tokens generated by lexical analyzer are grouped together to form a hierarchical structure which is known as syntax tree which is less detailed.

X. TYPES OF DERIVATION

There are mainly two types of derivations which are Leftmost derivation and Rightmost derivation. Let’s consider the grammar with the production $S \rightarrow S + S | S - S | S * S | S / S | (S) | a$

[1] Leftmost derivation:

- A derivation of a string $W$ in a grammar $G$ is a left most derivation if at every step the leftmost non-terminal is replaced.
- Consider string: $a * a - a$

$$
S \rightarrow S - S \\
S \rightarrow S * S \\
a \rightarrow a - a
$$

Equivalent left most derivation tree

[2] Rightmost derivation:

- A derivation of a string $W$ in a grammar $G$ is a right most derivation if at every step the rightmost non-terminal is replaced.
- Consider string: $a - a / a$

$$
S \rightarrow S - S \\
S \rightarrow S / S \\
S \rightarrow S - a \\
a \rightarrow a - a
$$

Equivalent Right most derivation tree

<table>
<thead>
<tr>
<th></th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
</tr>
</tbody>
</table>
XI. AMBIGUOUS GRAMMER

An ambiguous grammar is one that produces more than one leftmost or more than one rightmost derivation for the same sentence. In general, ambiguous grammar can generate more than one parse tree.

S -> S+S
S -> S+S
a + S
a + a + S
a + a + a
a + a + a

XII. LEFT RECURSION

Left hand side of terminal in right hand side of production rule is same as non-terminal on left hand side of production rule. i.e. A -> aA. Left recursion should not be there in grammar or production rule. In order to remove this left recursion, convert it into right recursion.

A -> bA'
A' -> aA'|C

XIII. LEFT FACTORING

Left factoring is kind of same as common prefix, i.e. A -> aB1|aB2|aB3. Left factoring should not be there in grammar or production rule. To remove this left factoring,

A -> aE
E -> B1|B2|B3

XIV. TYPES OF PARSING

There are mainly two types of parsing techniques.

[1] Top Down Parsing

XV. TOP DOWN PARSING

- Root to leaves
- LL Parser
- Left most derivation
- Derivation Process (Sentential)
- Less Complex
- Simple to implement
- Doesn’t work with NFA
- Doesn’t support recursion
- Common prefix not supported
- Applicable to small languages
- i.e. id + id + id

XVI. BOTTOM UP PARSING

- Leaves to root
- LR parser
- Right most derivation
- Reduction process
- High complex
- Complex to implement
- Works with NFA
- Supports recursion
- Common prefix supported
- Applicable to broad class of languages
- i.e. id + id + id
XVII. ERROR HANDLING

Each and every phase of compiler detects errors which must be reported to error handler whose task is to handle the errors so that compilation can proceed. **Lexical errors** contain spelling errors, exceeding length of identifier or numeric constants, appearance of illegal characters etc. **Syntax errors** contains errors in structure, missing operators, missing parenthesis etc. **Semantic errors** contain incompatible types of operands, undeclared variables, not matching of actual arguments with formal arguments etc. There are various strategies to recover the errors which can be implement by analyzers.

![Error Handler Diagram](image)

**REFERENCES**

https://en.wikipedia.org/wiki/Compiler


[7] Dr. Matt Poole and Mr. Christopher Whyley – “Compilers” - Department of Computer Science – University of Wales Swansea, UK


XVIII. CONCLUSION

To conclude this research, source program has to pass and parse from all sections of compilers to be converted into predicted target program. After studying this research paper, one can understand the exact procedure and step by step evaluation of source code in Lexical and Syntax Analysis which contain Index Terms, Compilers, Phases of Compiler, Operations on grammar, Lexical analysis, Roll of Scanner, Finite automata, Syntax analysis, Types of Derivation, Ambiguous grammar, Left recursion, Left factoring, Types of Parsing, Top Down Parsing, Bottom Up Parsing, Error Handling.

ACKNOWLEDGMENT

I am using this opportunity to express my gratitude to everyone who supported me in this research. I am thankful for their aspiring guidance, invaluably constructive criticism and friendly advice during the research. I am sincerely grateful to them for sharing their truthful and illuminating views on a number of issues related to the research work.