Syntax Analysis

(Compilers - Phase 2)

44	Amr Mohamed Fathi
55	Mohamed Ibrahim Shaban
28	Abdel-Rahman Yasser
58	Mohamed El-Sayed Helmi

Module Responsibilities

Syntax Module	Syntax Analyzer Generator	Grammar Extractor	Grammar Parser	Responsible for converting CFG rules from given file into “Class Objects” we can operate on
			Node	Data structure representing symbols (terminals & non-terminals) in CFG rules Ex: S -> Acd | BA A or B or c or d represent Node
			Production	Data structure represting production rule Ex: S -> Acd | BA
			Production Element	Data structure represting combination of symbols in the CFG grammar Ex: S -> Acd | BA Acd represents a Production Element
		Parse Table Generator	First Follow	Responsible for generating first() and flollow() for Derivation Table
		Responsible for generating the Parsing Table which is used to validate user program syntax		
	Syntax Analyzer	Parse Table	Data structure used to store parse table	
		Responsible for parsing user program (validate against given CFG rules) using parse table		
	Utilities	File Reader	Reading Syntax Rules file CFG	

Data Structures

<table>
<tr><td>CFG
(Context-Free
Grammar)</td><td>
<table>
<tr><td>Production</td><td>Data structure represting production rule
Ex: S -> Acd | BA</td></tr>
<tr><td>Production
Element</td><td>Data structure represting combination of symbols in the CFG grammar
Ex: S -> Acd | BA
Acd represents a Production Element</td></tr>
<tr><td>Node</td><td>Data structure representing symbols (terminals & non-terminals) in CFG rules
Ex: S -> Acd | BA
A or B or c or d represent Node</td></tr>
</table>
</td></tr>
<tr><td>Derivation Table</td><td>It’s encapsulation to LL1_Table .
Row represents transitions of a Non_Terminal .
Cell represents Production element ongoing on this transition .</td></tr>
</table>

Algorithms

Kosrajo	Detecting and building FOLLOW set for a strongly connected component of follows in CFG Ex : A follows B , B follows C , C Follows A

Design Walkthrough

<table>
<tr>
<td>CFG Builder</td>
<td>

Reads language rules as .txt file
Then converts CFG rules from given file into “Class Objects” we can operate on

<table>
<tr><td>Grammar</td><td></td></tr>
<tr><td>Production</td><td></td></tr>
<tr><td>Production Element</td><td></td></tr>
<tr><td>Node</td><td></td></tr>
</table>
</td>
</tr>
<tr>
<td>Prepare for LL1</td>
<td>

Using to modules, one for eliminitaing left recursion and another one for applying left fatoring
Left recursion module: We first find indices of Symbols where we need to eliminate left recursion then we handle these indices.
Left factoring module: We first find indices of Symbols where we need to apply left factoring then we handle these indices.

</td>
</tr>
<tr>
<td>Generate LL1 Table</td>
<td>

Use LL1_Table assistant class first_follow to generate first and follow for all Nodes , Production Elements and Productions to be used in Table building .
With previous data provided use it to build DerivationTable that’s used in syntax analysis .

</td>
</tr>
<tr>
<td>Syntax Analysis</td>
<td>

Given user program in form of tokens
We start with stack containing first non-terminal in table (S)
Using derivation table to find which production element “S” will go to which given current Token
Then we replace S with this production and repeat the process
If start of stack == non-terminal we lookup the table
If start of stack == terminal we match terminal with current token and remove current terminal from stack and move on to the next token
If transition goes to “Sync”, we remove top of stack
If transition goes to “Error”, we remove current token and report an error

</td>
</tr>
</table>