M.S.Ramaiah Institute of Technology
(Autonomous Institute, Affiliated to VTU)
Department of Computer Science and Engineering

# EMULATOR FOR PREDICTIVE PARSING

( 1MS13CS001-Aahan Singh, 1MS13CS004-Abdul Shah , 1MS13CS017-Akhil Raj, 1MS13CS030-Archit Bhatnagar )

## ABSTRACT

The purpose of the project is to develop an **emulator for predictive parsing.** The emulator would take in the grammar as the input and construct the parsing table. For any input string on the grammar, the emulator validates it by parsing in the parsing table. The input consists of a set of grammars which as passed to the JS file and then follows through with the splitting of input into proper strings for each grammar. After left factoring and removing left recursion the first and follow is calculated for each of the grammar strings. The parsing table is finally constructed at the end and printed

## METHODS

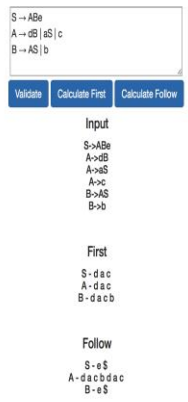
Parsing an entire program is initiated by calling the parser function representing the grammar's start symbol. To implement a predictive parser, we can analyze the grammar to compute the FIRST and FOLLOW sets for each non-terminal symbol. The parser tries all possible choices of production. Any time the parser reaches a point where an error is raised (because the next input token didn't match the one expected), the parser "backtracks" to the most recent nondeterministic choice.

## RESULTS

COMPILER DESIGN

List Of Grammers

| | |
|---|---|
| 1. | $S \rightarrow 0S1 \mid 01$ |
| 2. | $S \rightarrow S(S)S \mid \varepsilon$ |
| 3. | $S \rightarrow A$<br>$A \rightarrow BC \mid DBC$<br>$B \rightarrow Bb \mid \varepsilon$<br>$C \rightarrow c \mid \varepsilon$<br>$D \rightarrow a \mid d$ |
| 4. | $S \rightarrow CC$<br>$C \rightarrow cC \mid d$ |
| 5. | $E \rightarrow E+T \mid T$<br>$T \rightarrow T*F \mid F$<br>$F \rightarrow (E) \mid id$ |
| | $S \rightarrow AaAb \mid BbBa$ |




| | |
|---|---|
| 6. | $S \rightarrow AaAb \mid BbBa$<br>$A \rightarrow \varepsilon$<br>$B \rightarrow \varepsilon$ |
| 7. | $S \rightarrow 0 \mid A$<br>$S \rightarrow AB$<br>$B \rightarrow 1$ |
| 8. | $S \rightarrow S+S \mid S*S \mid a$ |
| 9. | $S \rightarrow (L) \mid a$<br>$L \rightarrow L,S \mid S$ |
| 10. | $S \rightarrow ABe$<br>$A \rightarrow dB \mid aS \mid c$<br>$B \rightarrow AS \mid b$ |

| Table | e | d | a | c | b | $ |
|---|---|---|---|---|---|---|
| S | | S->ABe | S->ABe | S->ABe | | |
| A | | A->dB | A->aS | A->c | | |
| B | | B->AS | B->AS | B->AS | B->b | |

## INTRODUCTION

Recursive descent is a top-down parsing technique that constructs the parse tree from the top and the input is read from left to right. It uses procedures for every terminal and non-terminal entity. This parsing technique recursively parses the input to make a parse tree, which may or may not require back-tracking. A form of recursive-descent parsing that does not require any back-tracking is known as predictive parsing.

## IMPLEMENTATION DISCUSSION

We need to eliminate direct and indirect left recursion and perform left factoring before we pass on the grammar to the Predictive parser. Each grammar production is then passed onto functions to calculate first and follow and then further to the parsing table.

## CONCLUSION

We have read and understood the concepts of Syntax Analysis and in detail the predictive parser(Top Down) and have acquired the knowledge required for designing and making a program that parses an input grammar. We have then deployed this algorithm to create a working example of an emulator and simulates the predictive parsing taking into inputs as the grammar strings and producing the parsing table after calculating the first and follow. We have used javascript to code the back end and displayed it on a web-page using HTML.

## DATA-FLOW DIAGRAMS

**Algorithm for design:**

Input:
string ω
parsing table M for grammar G
Output:
If ω is in L(G) then left-most derivation of ω,
error otherwise.
Initial State : $S on stack (with S being start symbol)
ω$ in the input buffer
SET ip to point the first symbol of ω$.
repeat
let X be the top stack symbol and a the symbol pointed by ip.
if X∈ Vt or $
if X = a
POP X and advance ip.
else
error()
endif
else /* X is non-terminal */
if M[X,a] = X → Y1, Y2,... Yk
POP X
PUSH Yk, Yk-1,... Y1 /* Y1 on top */
Output the production X → Y1, Y2,... Yk
else
error()
endif
endif
until X = $ /* empty stack */

## REFERENCES

[1]A. Aho, R. Sethi, J.D. Ullman, Compilers:
Principles, Techniques, and Tools, Reading, MA: Addison-Wesley, 1986.

[2]J.P. Bennett, Introduction to Compiling Techniques. Berkshire, England: McGraw-Hill, 1990.

[3]www.tutorialspoint.com

[4]www.umd.edu/cse/compiler

[5]www.mit.edu/cse