MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

Project Report

on

Parser Visualizer

Submitted By:

Tanishq Soni

0901CS191129

Vishal Prajapati

0901CS191137

Faculty Mentor:

Dr. Anjula Mehto

Assistant Professor, Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE

GWALIOR - 474005 (MP) est. 1957

MAY-JUNE 2022

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

Project Report

on

Parser Visualizer

A project report submitted in partial fulfilment of the requirement for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Submitted by:

Tanishq Soni

0901CS191129

Vishal Prajapati

0901CS191137

Faculty Mentor:

Dr. Anjula Mehto

Assistant Professor, Computer Science and Engineering

Submitted to:

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE

GWALIOR - 474005 (MP) est. 1957

MAY-JUNE 2022

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

CERTIFICATE

This is certified that **Tanishq Soni** (0901CS191129) has submitted the project report titled **Parser Visualizer** under the mentorship of **Dr. Anjula Mehto** in partial fulfilment of the requirement for the award of degree of Bachelor of Technology in Computer Science and Engineering from Madhav Institute of Technology and Science, Gwalior.

Dr. Anjula Mehto
Faculty Mentor
Assistant Professor
Computer Science and Engineering

10/05/2022

Dr. Manish Dixit
Professor and Head,
Computer Science and Engineering

Dr. Manish Dixit
Professor & HOD
Department of CSE
M.I.T.S. Gwalior

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

CERTIFICATE

This is certified that **Vishal Prajapati** (0901CS191137) has submitted the project report titled **Parser Visualizer** under the mentorship of **Dr. Anjula Mehto** in partial fulfilment of the requirement for the award of degree of Bachelor of Technology in Computer Science and Engineering from Madhav Institute of Technology and Science, Gwalior.

Dr. Anjula Mehto

Faculty Mentor
Assistant Professor
Computer Science and Engineering

Dr. Manish Dixit

Professor and Head,
Computer Science and Engineering

Dr. Manish Dixit
Professor & HOD
Department of CSE
M.I.T.S. Gwalior

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR

(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

DECLARATION

I hereby declare that the work being presented in this project report, for the partial fulfilment of requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering at Madhav Institute of Technology & Science, Gwalior is an authenticated and original record of my work under the mentorship of **Dr. Anjula Mehto, Assistant Professor,** Computer Science and Engineering.

I declare that I have not submitted the matter embodied in this report for the award of any degree or diploma anywhere else.

Tanishq Soni
0901CS191129
3rd Year,
Computer Science and Engineering

Vishal Prajapati
0901CS191137
3rd Year,
Computer Science and Engineering

MADHAV INSTITUTE OF TECHNOLOGY & SCIENCE, GWALIOR
(A Govt. Aided UGC Autonomous & NAAC Accredited Institute Affiliated to RGPV, Bhopal)

ACKNOWLEDGEMENT

The full semester project has proved to be pivotal to my career. I am thankful to my institute, **Madhav Institute of Technology and Science** to allow me to continue my disciplinary/interdisciplinary project as a curriculum requirement, under the provisions of the Flexible Curriculum Scheme (based on the AICTE Model Curriculum 2018), approved by the Academic Council of the institute. I extend my gratitude to the Director of the institute, **Dr. R. K. Pandit** and Dean Academics, **Dr. Manjaree Pandit** for this.

I would sincerely like to thank my department, **Department of Computer Science and Engineering, for allowing** me to explore this project. I humbly thank **Dr. Manish Dixit**, Professor and Head, Department of Computer Science and Engineering, for his continued support during the course of this engagement, which eased the process and formalities involved.

I am sincerely thankful to my faculty mentors. I am grateful to the guidance of **Dr. Anjula Mehto, Assistant Professor,** Computer Science and Engineering, for his continued support and guidance throughout the project. I am also very thankful to the faculty and staff of the department.

Tanishq Soni
0901CS191129
3rd Year,
Computer Science and Engineering

Vishal Prajapati
0901CS191137
3rd Year,
Computer Science and Engineering

ABSTRACT

This project is about visualizing different parsers. It is based on JS, HTML, and CSS. Occasionally it makes use of a visualization library d3.js. Main objective of this project is to assist in understanding of parser. It can generate first set, follow set, predictive parsing table, and LL (1) parser to name a few. This can also help one save on time. Repetitive and time-consuming things such as constructing predictive parsing table can be done with project, saving user time and also reducing chance of mistakes.

Keyword: Parser, first set, follow set, Top-down parsing, Bottom-up parsing, Visualization.

सार:

यह परियोजना विभिन्न पार्सर्स की कल्पना करने के बारे में है। यह जेएस, एचटीएमएल और सीएसएस पर आधारित है। कभी-कभी यह विज़ुअलाइज़ेशन लाइब्रेरी **d3.js** का उपयोग करता है। इस परियोजना का मुख्य उद्देश्य पार्सर को समझने में सहायता करना है। यह कुछ नाम रखने के लिए पहला सेट, फॉलो सेट, प्रेडिक्टिव पार्सिंग टेबल और एलएल (1) पार्सर उत्पन्न कर सकता है। यह समय पर बचत करने में भी मदद कर सकता है। दोहराव और समय लेने वाली चीजें जैसे कि भविष्य कहनेवाला पार्सिंग टेबल का निर्माण परियोजना के साथ किया जा सकता है, उपयोगकर्ता के समय की बचत होती है और गलतियों की संभावना भी कम होती है।

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: PROJECT OVERVIEW

1.1 Introduction

This project is based on HTML, JS, and CSS. It does make use of some additional JS library (such as d3.js) though. This is project is about visualizing parsers. It visualizes both top-down, and bottom-up parser. There's also an option to compute first set, and follow set for a given grammar.
JS library d3.js is used for visualizing parse tree, and canonical items graph.

1.2 Objective and Scopes

The main objective of this project is to help visualize parsers, which are tricky to understand. Visualization may help in better understanding parsers.
It can compute a few procedures required for a parser (like, first set and follow set), this can save time as one doesn't have to do redundant things again and again.

1.3 Project Features

1. Evaluating first set and follow set
2. Visualizing LL (1) parser
3. Visualizing Recursive descent parser
4. Visualizing LR (0) parser

1.4 System requirements

This project is based on JS, HTML, and CSS. Most modern browsers are able to run this without any issues. Some browsers such as Internet Explorer 9 won't be able to run it.
Any web browser which supports ES2015 version of JS, CSS version 2.1, and HTML version 5 will be able to run it.
Some part of this project makes use of a JS library named d3.js, since it is a graphical library, it'll require the system to sufficient amount of memory and computation power. Older mobile phones may not be able to run this

CHAPTER 2: PRELIMINARY DESIGN

2.1 Introduction

This chapter contains algorithms used in this project. They are here written in just few lines however they are very lengthy if one has to implement. Especially when an optimized version is expected to written.
Algorithms are the major part of this project. In fact, that's where most time is spent.
Memorization makes program lengthy but at the end it improves algorithm's performance largely due to fact some algorithm here uses recursion.

2.2 Algorithm for Top-down parsing

It constructs parse tree from the top and the input is read from left to right.

2.2.1 First set

First set is required to construct predictive parsing table. First of a symbol X is the set of terminals that begin with all string derived from X.

Algorithm: First (X)

```
first_set    empty
if x is empty return first_set
if x is not a terminal
  insert x in first_set then return first_set
else
   look for left most symbol in each production rules for X
   for each symbol
     if not contain epsilon
        add in first_set
     else
        add first of symbol just next to it in first_set
 return set
```

2.2.2 Follow set

Follow set is required to construct predictive parsing table. Follow of a symbol X is the set of terminals that appear just after right of non-terminal X.
In this algorithm infinite recursion can be easily overlooked. Even with valid input we can make this algorithm call itself forever. We've to make sure with the us of lookup table that this algorithm won't call itself over and over.

2.2.2 Follow set (continued)

Algorithm: Follow (X)

```
follow_set    empty
if X is start symbol
  insert $ in follow_set

for occurrence (y) of X in rhs of every production rules
  if y is non terminal other than epsilon
    insert y in follow_set
  else if y is a terminal
    if follow of y doesn't contain epsilon
      insert follow of y in follow_set
    else
      insert everything other than epsilon in follow_set
      insert follow (rhs) in the follow_set

    insert follow of y in follow_set

return follow_set
```

2.2.3 Predictive parsing table

Predictive parsing table is required by LL (1) parser. Follow set and first set are frequently required by this algorithm to construct the table. It's better to precompute them and cache them for this algorithm to use. We can use hash table to store our computation. Hash table provide us constant time insertion, and access. Only two methods we care about.

This algorithm assumes there is no left recursion in the grammar.

Algorithm: predictieParseTable (G)

```
table # a 2D array
insert each non-terminal in each column
insert each terminal in first column of each row

# Filling the table
for every production rule (pr) in grammar's rules
  if first of pr   rhs doesn't contains an epsilon
    for each non-terminal in first pr   rhs
      fill pr in table, row is pr   rhs and column is non-terminal
      in case of a conflict return false # multiple entries
  else if follow of pr   rhs contains $
      for each non-terminal in follow of pr   rhs
            fill pr in table, row is pr   rhs and column is non-terminal
            in case of a conflict return false # multiple entries
            else
              fill pr in table, row is pr   rhs and column is $
              in case of a conflict return false # multiple entries
                      return table
```

2.2.4 LL (1) parser

Predictive parsing table is required by LL (1) parser. Algorithm expects that there's no conflict in predictive parsing table, i.e., grammar is accepted by LL (1) parser.

It makes use of a stack. If at end both remaining element and stack is empty, then the string is accepted by the parser. Otherwise not.

We've to augment stack with the start symbol, and string that has to be parse with a '$'.

Algorithm: LL_1_parser (string, Table, start_symbol)

```
stack    start_symbol #    represent push
in string append '$' at end

until stack is not empty
 if string is empty
  return false

 if stack top is string top
  pop stack
  remove front of string
  continue loop

 rule    table [stack top] [string top]
 pop stack
 stack    rule's rhs

if string is empty
 return true

return false
```

2.2.5 Recursive descent parser

This parser makes use of backtracking.

Algorithm: recursiveDescentParsing(Grammar, Token)

```
descent_pointer    0
stack    Grammar.start_symbol

while stack is not empty
 if stack.top is a non-terminal
   top    stack.top
   stack.pop
   insert production rule whose rhs is top
   remove that production rule from Grammar
 else
   if stack.top is token[descent_pointer]
     stack.pop
     increment descent_pointer
   else
     backtrack

if descent_pointer is token's size
  return true
return false
```

2.3 Algorithm for Bottom-up parsing

It constructs a parse tree from the bottom and the input is read from right to left.

2.3.1 Constructing canonical items

Canonical items are represented by a directed, labelled graph. We use Breadth First Search for construction of this.

Algorithm: canonicalItems(Grammar)

```
items     empty # directed, labelled graph
insert S'   . StartSymbol in Grammar.production_rules
queue     S'

while queue is not empty
  state     queue.top
  # Construct canonical item
  Look in rhs, for each period
   if on right of period is non-terminal
    insert rule in state with a period on rhs
  queue.pop

  Look in rhs, for each period
    move the period
    push into queue state, right of period

  insert state into items

return items
```

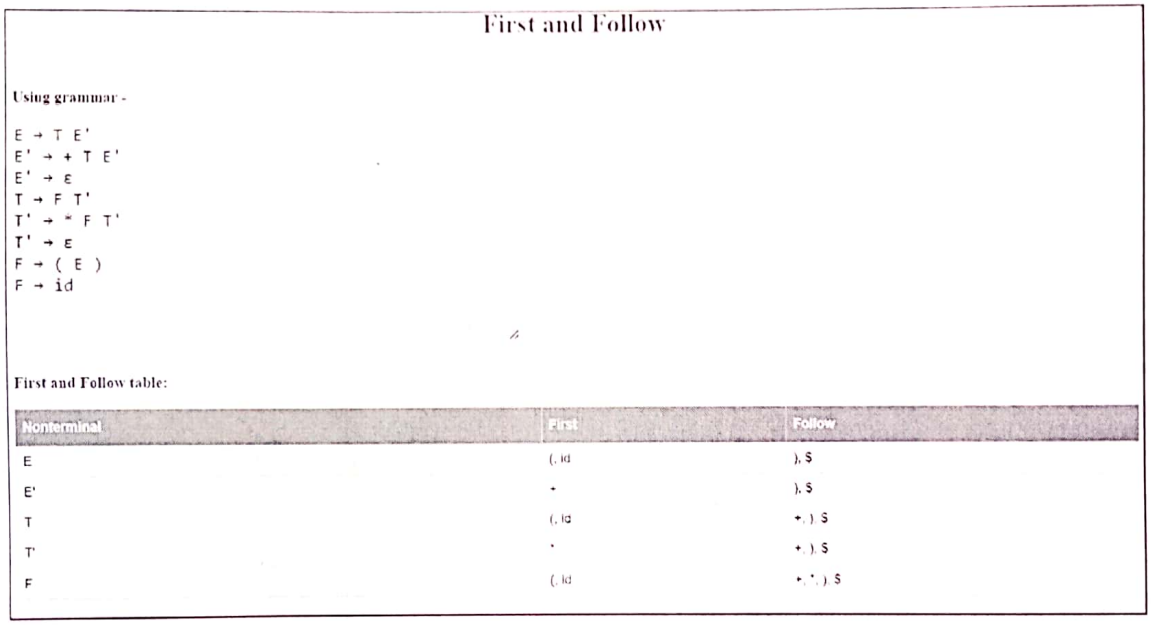
CHAPTER 3: FINAL ANALYSIS

3.1 Results

This project's program is working as expected for valid inputs. However, rarely it doesn't work as anticipated. This happens when wrong input is provided to it (such as incorrect grammar, or grammar having left recursion).

3.2 Result analysis

3.2.1. Computing first and follow set



First and Follow

Using grammar -

```
E → T E'
E' → + T E'
E' → ε
T → F T'
T' → * F T'
T' → ε
F → ( E )
F → id
```

First and Follow table:

Nonterminal	First	Follow
E	(, id	), $
E'	+, ε	), $
T	(, id	+,), $
T'	*, ε	+,), $
F	(, id	+, *,), $

Figure 3.2.1. Web page displaying first set and follow set

This web page is opened when a user input grammar in homepage and click "First and Follow" button. User here has nothing to input, all they can do is see the table. If compute the set for different grammar they'll have to go back to homepage.

First set and follow set are computed as expected for multiple grammar which we've tried. However, for grammar containing left recursion, it won't work. Our algorithm expects grammar without any left recursion. User are expected to remove left recursion if there's any.

3.2.2. LL (1) parser

3.2.2.1 Predictive parsing table

LL(1) parser

Using grammar -

```
E → T E'
E' → + T E'
E' → ε
T → F T'
T' → * F T'
T' → ε
F → ( E )
F → id
```

Predictive Parsing table :

	$	+	*	(	)	id
E				E → T E'		E → T E'
E'	E' → ε	E' → + T E'			E' → ε	
T				T → F T'		T → F T'
T'	T' → ε	T' → ε	T' → * F T'		T' → ε	
F				F → (E)		F → id

Figure 3.2.2.1. Web page showing Predictive parsing table

Predictive parsing table is computed as per the algorithm described in section 4.2.3, it assumes first and follow set are calculated before head.

This is tested for multiple grammar, and it was working for all of them. In case of a conflict, conflicted cell is shown as red.

3.2.2.2 Parsing using LL (1)

Parsing :

Enter tokens seperated by comma : id →

Stack	Remaining	Action
$,E	id $	none
$,E',T	id $	E → T E'
$,E',T',F	id $	T → F T'
$,E',T',id	id $	F → id
$,E',T'	$	Match id
$,E'	$	T' → ε
$	$	E' → ε
		Match $

Figure 3.2.2.2. Web page showing steps taken to parse given string

It'll generate this table as explained in the algorithm in Section 2.2.4.

At the end if stack and remaining is empty, then string is accepted else it's not.

3.2.2.3 Parse tree

Parse tree -

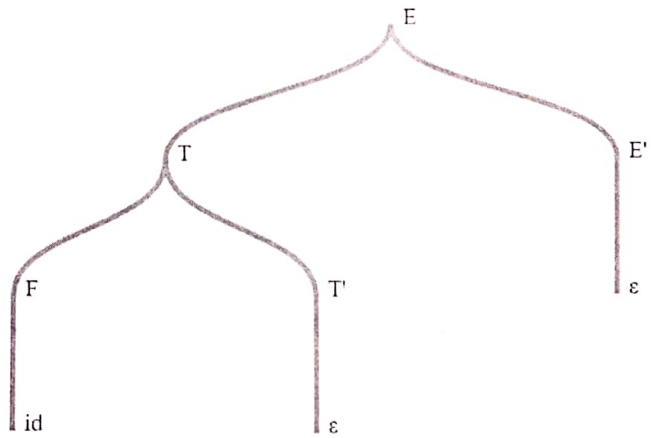


Figure 3.2.2.3. Web page showing parse tree for a given string

Parse tree here is generated using d3.js library.

3.2.3. LR (0) parser

Canonical Items

Figure 3.2.3. Web page showing canonical items

3.3 Problem faced

3.3.1. Tricky algorithms

Algorithms used for almost anything is this project was tricky for us. They are trickier to come up with. Even the naïve ones.

Stack, hash table, array, and graph are some data structures used. Backtracking, recursion, and graph search (DFS) are frequently used. Implementing all of them in a programming language which was completely new to us is somewhat of a challenge.

3.3.2. Lengthy JS programs

JS programs were lengthy. That makes them trickier to maintain and work on.

It was largely due to fact JS was doing so much. It generates html table, draw graphs/tree to name a few.

Every computation is done in JS. HTML, and CSS aren't programming language.

HTML, and CSS aren't heavily used. Front end of this looks very basic due to this.

3.4 Applications

3.4.1. Learning different parsers

With this project one is able to visualize different parsers. They'll be able to understand parser better. This project visualizes LL (1), LR (0), and recursive descent parser.

3.4.2. Saving time

Sometimes repetition just costs us time without us gaining anything from it. Those who're learning LL (1) would find computing the first set and follow set over and over again annoying. They can use this project to compute first and follow set, and save them some time.

They can also do the same about predictive parsing table, and canonical items.

3.4.3. Verifying answers

Users can use this project to verify their answer. They can do so for the first set, follow set, predictive parsing table, and LL (1) parser to name a few.

CHAPTER 4: CONCLUSION

4.1 Conclusion

We've implemented different types of parsers, and added some visualization to it. It'll help one in understanding the working of parsers. It'll save time as one doesn't have to compute first and follow set, predictive parse table, and canonical items.

4.2 Future scope

We can add more types of parsers to this project such as LALR.

We can also add more visualization such as step by step computation as per algorithms.

REFERENCES

1. LL Parsers (https://en.wikipedia.org/wiki/LL_parser)
2. Recursive descent parser (https://en.wikipedia.org/wiki/Recursive_descent_parser)
3. LR Parser (https://en.wikipedia.org/wiki/LR_parser)
4. Javascript references (https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API)
5. d3.js documentation (https://github.com/d3/d3/wiki)
6. HTML references (https://developer.mozilla.org/en-US/docs/Web/HTML/Reference)
7. Aho Alfred V. and Jeffery D Ullman (2002). *Principles of Compiler Design,* Narosa Publishing House