ParseIT++: Teaching tool for parsing

A thesis submitted

in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

Tushar Gautam

20111071

to the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

July, 2022

CERTIFICATE

It is certified that the work contained in the thesis titled **ParseIT++: Teaching tool for parsing**, by **Tushar Gautam**, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Amey Karkare

Prof Amey Karkare

Department of Computer Science & Engineering

IIT Kanpur

July, 2022

Declaration

This is to certify that the project titled **"ParseIT++: Teaching tool for parsing"** has been authored by me. It presents the research conducted by me under the supervision of **"Prof. Amey Karkare"**.

To the best of my knowledge, it is an original work, both in terms of research content and narrative, and has not been submitted elsewhere, in part or in full, for a degree. Further, due credit has been attributed to the relevant state-of-the-art and collaborations (if any) with appropriate citations and acknowledgments, in line with established norms and practices.

[illegible]

Name: Tushar Gautam

Roll No: 20111071

Programme: Master of Technology

Department of Computer Science & Engineering

Indian Institute of Technology Kanpur

July 2022

ABSTRACT

Name of student: **Tushar Gautam** Roll no: **20111071**

Degree for which submitted: **Master of Technology**

Department: **Computer Science & Engineering**

Thesis title: **ParseIT++: Teaching tool for parsing**

Name of Thesis Supervisor: **Prof Amey Karkare**

Month and year of thesis submission: **July, 2022**

Compiler design is a core course of the undergraduate computer science branch. Compiler courses are very theoretical, and most of the part comprises of parsing techniques. It is also hard to find everyday use of parsing, so it becomes difficult to connect with it simply by gaining theoretical knowledge. For such reasons many teaching tools are built to make the course interactive and practical. However, many of these parsing tools are not widely popular and easily accessible. Mostly these parsing tools only focus on either practical or theoretical aspects. Also these tools do not focus on collecting information about how tools are used.

In this thesis, we present a web-based tool ParseIT++ for teaching parsing techniques. ParseIT++ is an extension of console-based tool ParseIT. It follows a question-answer-based approach for teaching theoretical concepts of parsing along with visualizations for connecting the practical and theoretical aspects of the course. For question-answer-based sections, it provides hints based on the user's solutions. ParseIT++ provides visualizations for Parse Table sections and generates a string for an erroneous entry in the parse table.

As most of the available teaching tools do not focus on data collection, Studies done using these teaching tools are mainly based on feedback or are test-based

studies. But ParseIT++ collects all the activities of the user, and separate studies can be performed based on this data like how users are using the tool, improvements that can be helpful, and comprehensive studies on parsing theory.

To my family and friends

Acknowledgements

I would like to express my gratitude to my thesis supervisor Prof. Dr. Amey Karkare, for letting me work on this project. I am very grateful to him for his support and guidance throughout this thesis work. He has provided me with invaluable input on the tool and also guided me to build an effective tool. I am highly thankful for all the motivation that he has provided to accomplish the thesis work.

I am thankful to Nimisha Agarwal for helping me understand the working of ParseIT and its setup.

I would like to thank Manish Sundriyal and Harsika Diksha for helping me in building the tool and for giving their valuable input on the UI design.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Compiler design is a core course of the Computer Science branch, and it is heavily comprised of parsing techniques theory. Compiler course seems complicated to students because of their theoretical nature, and also, in the early times, not many efforts were made to make this course more interesting and interactive. Many tools, such as YACC, BISON, Flex, etc are used in this course to give practical knowledge of parsing. But these tools are built for professionals who already have knowledge about compiler design. It becomes cumbersome for students to use these tools as they have to understand the parsing concepts and have to learn how to use these tools as well.

Because of these issues, many teaching tools are built for parsing techniques like JFLAP, LR(0) & LL(1), ParseIT, LISA, etc. But, these are not widely popular and accessible. Usually, these tools are used for teaching in the institute where they are built. Most tools focus on visualizations to impart practical aspects of parsing. It also helps students to stay interested in the parsing and understands parsing by seeing the techniques in action. There are a few tools that focus on theoretical aspects of parsing, like ParseIT. These teaching tools lack to impart the combination of theory and practical aspects of parsing.

In this thesis, we have developed a teaching tool that combines both theory and practical aspects of parsing. It helps users with theory using MCQ questions and a hint-based approach, and it also provides visualizations to give a practical view of parsing. We also present a survey done on parsing teaching tools and developed a tool that solves the problems with these tools.

1.1 Objective

Develop a teaching tool for parsing that can solve the problem of teaching theoretical and practical aspects of parsing. The main objective is to build a web-based tool for teaching parsing that generates MCQ questions and, based on user input, generates hint questions. For Parse, the table section provides a puzzle of shuffled parse table and provides visualization of the parse tree for a generated string to excite erroneous entry as a hint. Also, develop a section for parsing and parse table game. The parsing section provides a workspace where users can practice parsing techniques. In the parse table game section, it gamified the detection of erroneous entries in the parse table.

1.2 Motivation

Compiler courses are difficult to understand because of their theoretical nature and no practical exposure. It heavily comprises parsing techniques, and it is difficult to find practical implications of parsing in day-to-day life, so it becomes hard to connect with the theory.

In recent years interactive learning has imparted a significant impact on the world of teaching. It has made it easier for students to connect with concepts and reduced the burden on teachers. There are tools that are built for teaching parsing, but they lack in a few areas. Also, many such tools are not accessible to the wider community, and because of this, people have to build similar tools again. Available teaching tools mainly focus on visualizations and can only provide a practical view of parsing, and there are few tools that focus on theoretical understandings of parsing but lack interactivity and practicality.

Based on the survey done in this thesis, it seems that web-based tools are preferred by users may be because of easy accessibility, shareability, and no extra installations. We developed a web-based tool that helps users with the theoretical and practical aspects of parsing. We extended the ParseIT tool to add visualizations, practice sections, and gamification and increase the interactivity of the tool. One of the reasons to work on ParseIT was that it was built at IIT, Kanpur, so codes and detailed information about the tool was easily available.

1.3 Contribution

In this thesis, we extended the ParseIT tool, which includes the addition of visualizations, GUI features, a parsing section, gamification, and data collection of user activity. We have implemented GUI based parsing teaching tool that also collects the user's activity. We implemented a question-generation algorithm for parsing. MCQ Questions are automatically generated based on context-free input grammar and based on the user's answer, hint questions are generated. This question-answer-based approach is implemented for the First set, Follow set, SLR closure, and SLR goto sections. We have implemented a string generation algorithm for generating strings to excite erroneous cells in the parse table and corrected the minor error in the algorithm. Along with the string parsing algorithm, we implemented a practice section for parsing, we added visualization features and data collection of user's activity.

Chapter 2

Tools study and survey

Teaching tools are has become very popular with the modern learning. There are teaching tools for various courses to keep students more engaged in learning. Many teaching tools for parsing are built but are not accessible to the wider community, because of which people have to build similar tools again. Web based tools seems to be more preferred by students because of no no installation steps, easily accessible and easily sharable. Teaching tools built so far are mainly focuses on visualisations and can only provide practical view over parsing and there are few tools that focuses on theoretical concepts to be taught but lack interactivity and practicality.

2.1 JFLAP

JFLAP [1] project started at Rensselaer Polytechnic Institute as a collection of tools. Later it was moved to Duke University. Users can download it from [2]. JFLAP is a visualization tool and it is not only suitable for teaching parsing but, also provides the features that can be used in other CS courses. It supports different kinds of parsers, from which we experimented with SLR(1), LL(1), CYK, and brute-force parsers. We have listed below the features of JFLAP according to our experience while using the tool:

1. **Installation:** JFLAP is a jar executable application, users do not have to install the application. But for executing JFLAP, users must have Java JDK

and JRE installed in the system.

2. **Supported Parsers:** JFLAP supports a lot of parsers including LL(1), SLR(1), brute-force parser, multiple brute-force parser, user control parser, CYK parser, and multiple CYK parser. We used this tool for LL(1), SLR(1), brute-force parser, and CYK parser.

3. **Grammar Format:** There is a specific format in which it takes the input grammar which is very similar to Backus–Naur form, the format used in classrooms. There are few restrictions on the set of terminals and non-terminals, it treats every small alphabet as a terminal and every capital alphabet as a non-terminal. Users cannot define terminals and non-terminals of more than one alphabet and also each symbol on RHS of a production rule should not be space-separated. 'ϵ' symbol is represented using the 'λ' symbol.

4. **Error Reporting and Messages:** If grammar is not parsable by a parser, JFLAP reports it in different ways for different parsers. In the case of LL(1), it gives a popup reporting that grammar is not LL(1) and lets users proceed to parse table construction. In the case of SLR(1), it does not report but highlights the conflicted cells and users can choose the entry for these cells. While parsing it informs about the steps taken and also highlights the corresponding cells of the parse table. In the CYK parser, if the grammar does not accept any string it simply reports it and does not move to the parsing option. Brute-force parser never reports anything about the grammar and always provides users with the option to parse the string.

5. **Visualizations:** It shows the visualizations of the parse trees. It also shows viable prefixes automaton for SLR(1) parsers. But as the number of states increases, the automaton is not easy to visualize due to clutter. Along with the parse tree, it also shows derivation and the inverted tree.

6. **Auto-String Generation:** It doesn't auto-generate any string for showing

parsing but users have to input a string for which they can see the complete parsing steps, parse tree, and derivation tree.

2.2 ParseIT

ParseIT [3] was developed at the Indian Institute of Technology, Kanpur, and is available at [4]. ParseIT supports LL(1) and SLR(1) parsers and we experimented with both the parsers. It is not a visualization tool but is a console question-answer-based teaching tool. It auto-generates questions for a given grammar for various sections of the parsing process. Depending on the user's inputs it provides auto-generated hint questions to help students with the problems. For the parsing table sections it auto generates a string corresponding to an erroneous entry in the table and shows parsing steps to the users. In LL(1) parser, ParseIT can detect left recursion and non-determinism. In the case of the LL(1) parser, if the grammar is not LL(1) but can be converted to LL(1) by left factoring, it can do so on the user's demand. Based on our experience we have detailed the features of ParseIT as follows:

1. **Installation:** ParseIT is a jar executable application, users do not have to install the application. For executing ParseIT, users must have Java JDK and JRE installed in their system.
2. **Supported Parsers:** ParseIT supports LL(1) and SLR(1) parsers.
3. **Grammar Format:** The grammar format accepted by ParseIT is the same as used in classrooms which is also used in the Dragon book [5]. For using the 'ϵ' symbol in the productions users have to write complete "epsilon" text in place of 'ϵ'.
4. **Error Reporting and Messages:** ParseIT generates hint questions based on user input and does not do any reporting or provide any messages. It generates a string and shows its parsing steps as hints for parse table problems.

ParseIT can detect the non-determinism and left-recursion in the grammar. In case of incompatible grammar, ParseIT simply prints the message in the console and exits for Parsing Moves options.

5. **Visualizations:** ParseIT does not show any visualizations for any option but shows the step-by-step parsing of the string.

6. **Auto-String Generation:** It can auto-generate the input string and show the parsing steps. Users can also input a string for which they have to complete the parsing steps.

2.3 LL(1) and LR(0) Tools

LL(1) and LR(0) tools are web-based tools available on the internet at [6] and [7] respectively. Both the tools show visualization for parse trees while parsing an input string. LR(0) does not generate any parse table and parses the string using LR(0) automaton. An interesting feature of LR(0) is that it can parse sentential forms (a string of terminals and non-terminals of the Grammar) too. Following are the feature details about LL(1) and LR(0) tools based on our experience:

1. **Installation:** LL(1) and LR(0) tools are web-based tools hosted at [6] and [7] respectively. Users only need to have a browser to use these tools.

2. **Supported Parsers:** LL(1) and LR(0) tools support LL(1) and LR(0) parsers respectively.

3. **Grammar Format:** LL(1) and LR(0) tools accept grammar in BNF with some reserved symbols. Each production rule should be in separate lines and symbols should be space-separated. There are a few restrictions on symbols that can be used in grammar. In LL(1), the symbol 'S' is reserved and cannot be used as a terminal or non-terminal. It always appends the grammar with a production rule "S $\rightarrow$ start symbol of the original grammar". In LR(0) symbol

‘S′’ is reserved for augmented production. In both the tools epsilon symbol is represented through empty single quotation marks ”.

4. **Error Reporting and Messages:** In LL(1), users can see the parsing stack and production rule used while building the parse tree. LR(0) tool shows parsing stack and the last action performed, alongside the parse tree.

5. **Visualizations:** Both the tools show the visualizations of parse trees. LR(0) also shows the LR(0) automaton.

6. **Auto-String Generation:** Both the tools cannot generate any string for showing parsing but users have to input a string for which they can see the complete parsing steps and parse tree.

Table 2.1: Summary of the features of the teaching tools

✓ signifies that parser provides the corresponding feature

× signifies that parser does not provides the corresponding feature.

– signifies that corresponding feature is not applicable for the parser

Feature	**JFLAP LL(1)**	**JFLAP SLR(1)**	**LR(0) Tool**	**LL(1) Tool**	**JFLAP Brute-Force**	**JFLAP CYK**	**ParseIT LL(1)**	**ParseIT SLR(1)**
Parse Table	✓	✓	–	✓	–	–	✓	✓
Parsing	×	✓	✓	✓	✓	×	×	×
Visualization	✓	✓	✓	✓	✓	✓	×	×
String Generation	×	×	×	×	×	×	✓	✓

Table 2.2: Parsers supported by the teaching tools

✓: Parser supported ×: Parser not supported

#	**Tool**	**LL(1)**	**LR(0)**	**SLR(1)**	**Brute-force**	**CYK**
1.	JFLAP	✓	×	✓	✓	✓
2.	LR(0)	×	✓	×	×	×
3.	LL(1)	✓	×	×	×	×
4.	ParseIT	✓	×	✓	×	×

2.4 Other tools

There are many tools that are developed for different courses and for different purposes. One example is online judges like SPOJ [8], Codechef [9], Prutor [10], etc. Online judges have become very popular in competitive programming nowadays. There are tools like VISUALGO [11] that visualizes the algorithms to help students with the practical view of algorithms. Many tools for automata courses like [1, 12, 13] are also popular in the computer science community.

Interactive Tutoring System (ITS) [14] was introduced by Rafael del Vado Vírseda for compiler design courses. In this system, teachers can select tools for different phases of the compiler and integrate them together into one environment. These tools can be changed later, and the evaluation process of ITS will not be affected. ITS solves this issue of the integration of various teaching tools with different specifications. Automatic grader checks the students' code in different languages and evaluates and reviews the progress of students.

LISA (Language Implementation System based on Attribute Grammars) [15] Tool for teaching compiler construction was built considering the effectiveness of active and constructive learning. In the LISA tool, students can experiment, visualize and test the techniques used in compiler design. It can be used as an educational as well as a compiler generator tool. It is an IDE in which language can be specified, and programs in the specified language can be compiled and executed. Users can see the compilation process of a program in LISA, and visualization for lexical, syntax, and semantic phase give a better, intuitive, and long-lasting understanding of the compilation process. Through the phases presented in LISA, users can learn about regular expressions, DFA and their implementations, Backus-Naur Form, LL(k) and LR(k) parsers, attribute grammar, and attribute evaluation strategies.

2.5 Survey

We did student survey of JFLAP, LR(0) & LL(1) tools and ParseIT. We asked students who studied compiler courses in their undergraduate to use these tools voluntarily and fill out the google form. Due to time constraints and students could not be available physically, only few students could fill the circulated survey form. For each tool, we asked few questions listed in Table 2.3 with their responses in Table 2.4. Table 2.3 represents all the questions asked in the survey and the questions applicable for each tool. Along with these questions, we also asked for ratings of the tools and reasons if a student didn't use the tool.

Table 2.4 contains the question number for questions available in Table 2.3 and corresponding to each question; all the options are also listed in the table. For each tool, we recorded the number of students (and percentage of students) who selected a particular option, and '−' is used to represent that the option is not applicable for that tool. Some students did not use JFLAP and ParseIT because it requires installation, and are tedious & are time consuming. Based on the survey it can be claimed that students like to use web-based tools as the maximum number(15) of students used LL(0) & LR(1) tools and also reasoned the. The combined use of ParseIT and JFLAP can be effective in teaching, as students found unique features of these tools useful.

2.6 Results of Tools Study & Survey

Based on the observations of tools, we recommend using a combination of tools that can help students with both practical and theoretical aspects of parsing. From our experience, we believe that the combined use of JFLAP and ParseIT can suffice this purpose, where ParseIT's quiz-based sections can help students learn the theory, and JFLAP can help students with the practical knowledge of parsing and visualization.

Due to covid-19 we get very few responses from students. About 71.4% students used LL(1) & LR(0), about 28.6% students used JFLAP and about 23.8% students

Table 2.3: Survey Questions and corresponding tools

✓: Question applicable ×: Question not applicable

#	**Question**	**JFLAP**	**LR(0), LL(1)**	**ParseIT**
1.	How often did you use this tool?	✓	✓	✓
2.	Which section(s) do you find most helpful?	✓	×	✓
3.	Was manual filling of entries useful?	✓	×	×
4.	Was automatically filling of entries useful?	✓	×	×
5.	Did you find conflict detection of this tool useful?	×	✓	×
6.	Was string parsing section beneficial for better understanding of parsing?	×	✓	×
7.	Was “question-based hint” approach helpful in better understanding of syntax analysis?	×	×	✓
8.	Was string generation mechanism for parse table useful?	×	×	✓
9.	Do visualisations help in better understanding of syntax analysis?	✓	✓	×
10.	Did this tool crash while using it?	✓	✓	✓
11.	Do you find this tool useful?	✓	✓	✓

Table 2.4: Student's survey results

count (%) is the number (percentage) of students who selected an option.
– denotes that the option is not applicable for that tool.

Q #	Options	JFLAP count (%)	LR(0) & LL(1) count (%)	ParseIT count (%)
1.	Very Frequent	0 (0%)	1 (6.6%)	1 (20%)
	Frequent	2 (33.3%)	2 (13.3%)	1 (20%)
	Regularly	0 (0%)	3 (20%)	1 (20%)
	Sometimes	4 (66.7%)	9 (60%)	40 (2%)
2.	Build parser	2 (33.3%)	–	–
	Parse	3 (50%)	–	–
	Automatic fill	5 (83.3%)	–	–
	First Set	–	–	2 (40%)
	Follow Set	–	–	3 (60%)
	SLR Closure	–	–	4 (80%)
	SLR Goto	–	–	2 (40%)
	Parse Table	–	–	4 (80%)
	Parsing Moves	–	–	2 (40%)
3.	Yes	5 (83.3%)	–	–
	No	1 (16.7%)	–	–
4.	Yes	5 (83.3%)	–	–
	No	1 (16.7%)	–	–
5.	Yes	–	13 (86.7%)	–
	No	–	2 (13.3%)	–
6.	Yes	–	12 (80%)	–
	No	–	3 (20%)	–
7.	Yes	–	–	5 (100%)
	No	–	–	0 (0%)
8.	Yes	–	–	2 (40%)
	No	–	–	0 (0%)
	Don't know	–	–	3 (60%)
9.	Yes	6 (100%)	13 (86.7%)	–
	No	0 (0%)	2 (13.3%)	–
10.	Yes	1 (16.7%)	14 (93.3%)	0 (0%)
	No	5 (83.3%)	1 (6.7%)	5 (100%)
11.	Yes	6 (100%)	14 (93.3%)	5 (100%)
	No	0 (0%)	1 (6.7%)	0 (0%)

used ParseIT and filled the circulated survey form. Around 23.8% students didn't use any of these tools. On average students give all these tools almost same rating that is 4.3. Based on students' rating of these tools and reasons for not using a tool, we can be conclusive about the results even with low participation of students.

Based on the results of student's survey, we can say that students prefer tools that do not require extra efforts. For example, they prefer web based tools instead of installing a software as is evident from the reasons for not using the tools and the fact that maximum students used web-based tools LL(1) & LR(0). Overall, students found these tools helpful, and will prefer these tools for learning parsing. Survey gives hint that students like unique features of each tool but, it is difficult to substantiate the usefulness of the unique features of these tools because of low participation.

Chapter 3

ParseIT++

ParseIT++ is an extension of ParseIT, and it is a GUI-based tool for teaching parsing. In the background, it executes ParseIT to generate processed data, and also the working mechanism of some of its sections is similar to that of ParseIT. ParseIT is a console-based question-answering tool. It was built in Java. ParseIT provides various sections for users to understand different concepts of parsing. It generates MCQ questions for a given grammar, for which users have to select the answers from available options. It moves to the next question or generates hint questions based on the user's answer. If the user selects all the correct answers, it will move to the next question. But if the user selects the wrong option, it will generate a hint question, and if the user doesn't select one of the correct options, it will generate a different type of hint question. It takes grammar as input and processes it to generate all the data for generating questions. The processed data includes the first set, follow set, the collection of LR(0) items, and parse tables. It provides different sections for users to understand different concepts of parsing. Based on user input, it generates hint questions, and for sections like First Set, Follow Set, SLR Closure, and Goto-I & II, it generates MCQ questions and hints questions. Whereas for Parse Tables sections, it generates MCQ questions and, as a hint, generates a string that will excite the erroneous entry in the table.

3.1 Limitations of ParseIT

ParseIT solves the problem of imparting theoretical knowledge through an automated tool. Such tools reduce the burden on professors and course TAs. However, it lacks some of the features that an automated teaching tool should have. Below are some of the drawbacks that are restricting the capabilities of the tool:

- It is a console-based tool, so all the work is done on the console. Users have to write on the console to answer the questions, which could be a tedious task. It makes it different from using as users have to follow a specific format to answer questions, and user experience is compromised.
- It is a question-answer-based tool and only has MCQ questions to teach theoretical concepts, which makes it difficult to understand practical implications and get a practical view of parsing.
- Any kind of visualization or practical view of parsing is not provided by the tool, which makes it less interactive and cannot make parsing interesting for users.
- It is a Java jar file, and users have to download it and need to have Java installed in their system, and it adds an extra step for users before using it.
- It doesn't collect any kind of data related to how users are using the tool, so it becomes difficult to understand what should be improved in the tool for better user experience and effective learning. In fact, none of the tools we explored collects data related to user activity.

3.2 Improving ParseIT

ParseIT++ is an extension of ParseIT and provides additional features. We improved the tool by eliminating the limitations mentioned in the previous sections.

These improvements can be viewed as the additional features provided by ParseIT++. These features are included to make the tool more interactive and interesting for the students. This results in increasing interest in parsing techniques. These features can also result in long-lasting learning of the concepts. We also added the data collection ability in the tool to collect the user's activity. The below list shows the new features added in ParseIT++:

- **GUI features:** ParseIT was a console-based tool limiting its interactivity and abilities. ParseIT++ is a GUI-based tool that makes it more interactive and results in an improved user experience.
- **Web-based tool:** Web-based tools do not require any cumbersome installations or any other additional work. And also it can be easily shared and operated with a few clicks.
- **Gamification:** We wanted to increase the interactivity of the tool and make it more interesting for the students. It could help in increasing interest in the course and long-lasting learning effects. So we built the tool with an interesting puzzle-oriented structure.
- **Data collection:** Currently, available tools do not collect any user data, so it becomes difficult to know how users are using the tool. And what do they want in the tool? Our tool collects the data of users' activity to gather information to answer these questions.

3.3 Algorithms Implemented

ParseIT++ uses many standard algorithms related to parsing. Most are implemented in ParseIT, like generating the first set, follow set, a canonical collection of LR(0) items, parse tables, etc. We didn't implement those algorithms but directly used ParseIT for that purpose. Some algorithms that we implemented in ParseIT++ are as follows:

3.3.1 Question and Hint Generation

We have implemented the algorithm for generating MCQ questions for various concepts of parsing and also implemented the hint generation algorithm for generating hints depending on the user's solutions. We have implemented the same algorithm as used in ParseIT for Question and hint generation. The algorithm is explained in detail in [3].

3.3.2 String Generation

For generating strings for erroneous entries in the parse table, we are using the same algorithm as used in ParseIT, which is given in [3]. For LL(1), we have made slight modifications as there was a minor error in the algorithm, which results in an infinite loop.

According to the algorithm present in [3], if there are rules like "A $\rightarrow$ α B" & "B $\rightarrow$ γ A" and these rules have the smallest RHS for non-terminals A and B, then this algorithm (terminal-only string generation algorithm) will fall in an infinite loop as it will keep on choosing these production rules one after the other. We handled this issue by marking the production rules that are already selected in the generation of the string. This way, it will not select the same production rule again and avoids the loop.

3.3.3 Parsing Algorithm & Tree Generation

For paring, standard parsing algorithms given in [5] are implemented in ParseIT++. ParseIT++ parses a valid input string and generates the parse tree. Generated parse tree is then visualized. Parse trees are generated using the standard algorithms given in [5].

Chapter 4

ParseIT++ Web-based User Interface

ParseIT++ is a GUI-based tool for teaching parsing based on ParseIT. There are various sections available in ParseIT++, these sections available in ParseIT++ can be divided into four types named question-based sections, parse table sections, parsing sections and parse table game. Question-based sections include the First set, Follow set, SLR Closure, and SLR goto-I & II. For a selected grammar, it generates MCQ questions in these sections. Based on the user's solution, it generates hint questions of two types. This whole mechanism is similar to as of ParseIT. For the Parse Table sections, it gives a shuffled parse table, and users have to unshuffle it. In this section, it generates a string that can excite an erroneous entry in the parse table. It also shows a parse tree for that particular generated string to help users identify the error. Users can also navigate to various steps of generation of that parse tree. These visualizations help users to connect with the practical implications of parsing. In ParseIT++, we added practice sections for LL(1) and SLR(1) parsing where users can practice the string parsing algorithm, and these sections come under parsing sections. It also shows the visualizations of the step-by-step generation of the parse tree of that string. ParseIT++ also provides a parse table game section which is similar to the parse table section. Along with these features, data on user's activity is also collected, which can give insights into how users use the tool, what the issues are and what can be improved to make learning easier and more interesting.

4.1 Tool's Architecture

ParseIT++ can be divided into two parts, front-end, and back-end. The front-end is built in ReactJS, and for back-end is developed in NodeJS. Back-end handles all the data-related work like DB operations and data processing. Front-end mainly has a Home page and four types of sections and the functioning of each section is different. These sections are the Question-based section, Parse Table section, Parsing section, and Parse Table Game section. Each section needs data related to the selected grammar. This data includes the first set, follow set, CLR items, and parse tables corresponding to the selected grammar, and we call this data processed data. This data is generated at the back-end. We are using ParseIT.jar to generate the processed data for a given grammar. At the back-end, we are running the ParseIT.jar file, which generates all the processed data for the given grammar. This processed data is then stored in the database for future use. The front-end requests this data for each section for the selected grammar, and the back-end makes the DB query and returns the requested data.

4.2 Front-end

Front-end of the tool is designed to be very interactive and interesting. It includes different types of the section with which users can interact. Each with different types of interactions, working, and views. It covers sections for the First set, Follows set, SLR(1) closure, SLR(1) goto-I & goto-II, which comes in Question-based sections. It covers the LL(1) Parse table and SLR(1) Parse table, which can be grouped as Parse table sections. In the Parsing sections, LL(1) and SLR(1) Parsing sections are included. It also provides SLR(1) Parse table game section. These sections are

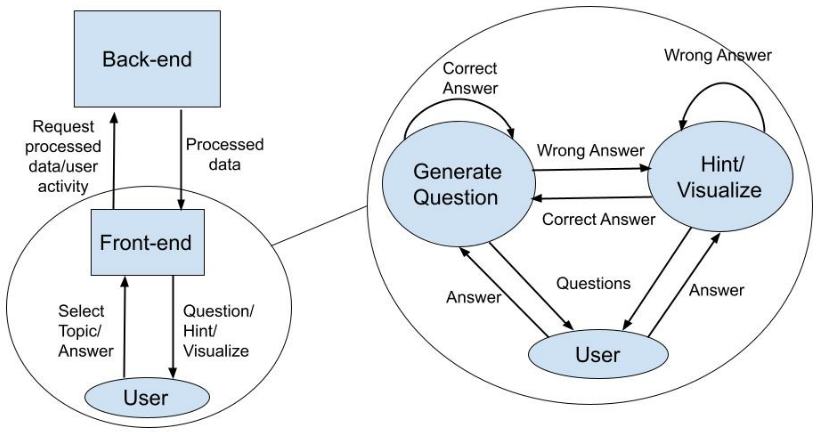


Figure 4.1: Overview of working of front-end & back-end

discussed below in detail.

4.2.1 Home:

On the home page, users first have to get themselves authenticated using their Gmail account. On successful logging in, the home page shows the instructor's grammars and students' grammars. Instructor grammars are those that are added by the admin/instructor and grammars added by students are student's grammar. Instructor's grammars are visible under "Available grammars". A grammar added by the student is only visible to that particular student. On the home page, it provides the option to add grammar. Students can add grammar using that, and added grammar will be visible under "Student's grammars". To add grammar, users have to enter grammar according to the specific format explained in Section 6.1 of Chapter 6. On submitting the grammar, entered grammar is sent to the back-end, where processed data is computed for that grammar and stored in the database.

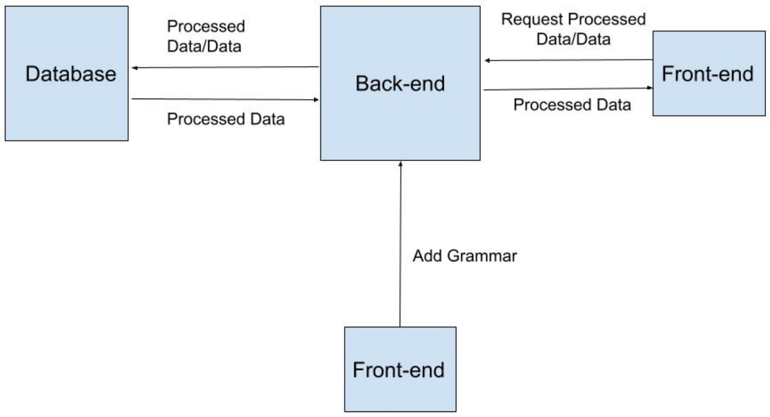


Figure 4.2: Working of addition of grammar

4.2.2 Question-Based Section:

It generates MCQ questions for sections like the First set, Follow set, SLR Closure, SLR goto-I and II. The flow of working of our tool for the Question-based section is that first, it requests the processed data for the selected grammar from the back-end. It generates a question for the selected section using this processed data, then, based on the user's solution, it generates hints and helps students to understand parsing better. For generating a questionnaire for First Set, Follow set, and SLR closure & SLR Goto sections, it needs the first set, follows set, and Canonical collection of LR(0) items, respectively. These sections work in similar ways as that of ParseIT but with GUI features. In these sections, an MCQ question is generated for the selected section, and users have to select the correct option from the available options. If users give a correct answer, it moves to the next question, but if selected options are not correct, it generates hint questions. It generates two types of hint questions:

Type-I: It is generated when users select a wrong option from the available options.

Type-II: It is generated when users misses a correct option from the available options.

Sections included in this are:

- **First Set:** Questions are generated based on First set of the selected grammar.
- **Follow Set:** Questions are generated based on Follow set of the selected grammar.
- **SLR Closure:** Questions are generated for the SLR closure moves for the canonical collection of LR(0) items of the selected grammar.
- **SLR Goto-I:** Questions are generated for the SLR Goto moves for the canonical collection of LR(0) items of the selected grammar.
- **SLR Goto-II:** Questions are generated for the SLR Goto moves for the canonical collection of LR(0) items of the selected grammar. Here questions are asked in a different pattern than that in Goto-I.

This section is similar to that in ParseIT, but in ParseIT, these are console-based, whereas, in ParseIT++, we have an interactive GUI-based interface.

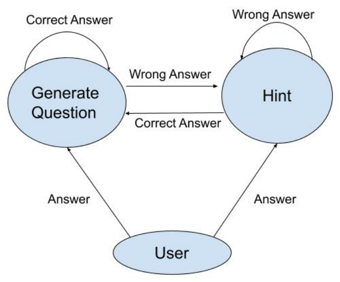


Figure 4.3: Working of question based sections

4.2.3 Parse Table Section:

Parse Table section includes LL(1) and SLR(1) parse table section. In this section, a parse table is given in which entries will be shuffled. Users have to unshuffle the table by swapping the entries in the cells. This section requires a parse table for the selected grammar and then shuffles the entries of the table. This shuffled table is then rendered, and users have to unshuffle it. It generates a string that can help in the detection of erroneous cells in the table. This string is generated by the string generation algorithms given in [3] for LL(1) (with slight modification mentioned in 3.3.2) and SLR(1). Along with this, our tool generates the parse tree for the generated string. Also, it shows a visualization of the generated string. Users can view the generation of the parse tree through the visualizations step-by-step.

This section is available for both SLR, and LL(1) parse tables. It is also a GUI-based section. In these sections, a shuffled parse table is displayed, which users can manipulate. Through the series of manipulation of the table, the user has to unshuffle this parse table. It generates a string that, on parsing, can excite an erroneous cell in the table. It shows the visualization of the parse tree for that generated string. Users can also navigate and see the step-by-step generation of the parse tree.

4.2.4 Parsing Section:

This section is available for SLR and LL(1) parsing and is also a GUI-based section. This section is like a practice space for users where they can practice parsing a string. In this section, an empty stack is given, and users have to fill the stack in a similar fashion as it will be while parsing that string. Users have to input a string to parse, and if the string is not valid, it will be indicated, and if the input string is valid, it allows the user to parse the string. Users are provided with symbols as options that can be pushed into the stack and a POP option to pop symbols out of the stack. Using these options, users have to mimic the actions on the stack. It also

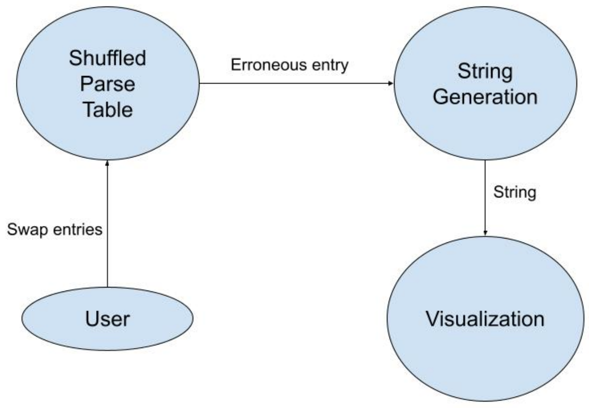


Figure 4.4: Working of parse table sections

generates a parse tree as actions on the stack are performed by the user. If the user tries to perform a wrong action on the stack, it doesn't allow it and alerts the user. The visualization of a parse tree helps students to understand the practical view of parsing. ParseIT++ computes the parse tree and parse stack for the input string and uses them to alert users about their mistakes and to show the visualization of the parse tree.

4.2.5 Parse Table Game Section:

This section is similar to that of the Parse Table section. This section is only available for SLR(1) parse table. A parse table is given in which one non-error entry is swapped with an error entry. Few strings are given from which only one string on parsing can excite the erroneous entry in the table. Now users have to correct the table, and after correcting the table, they are asked to select the string that helped them to detect the error. This section helps students to connect the parse table with the parsing. For this, ParseIT++ first generates the strings to excite the cell for each cell. After generating such strings, for each cell, it finds the strings that do not excite that cell. After doing this for each non-empty cell, it has a list of strings

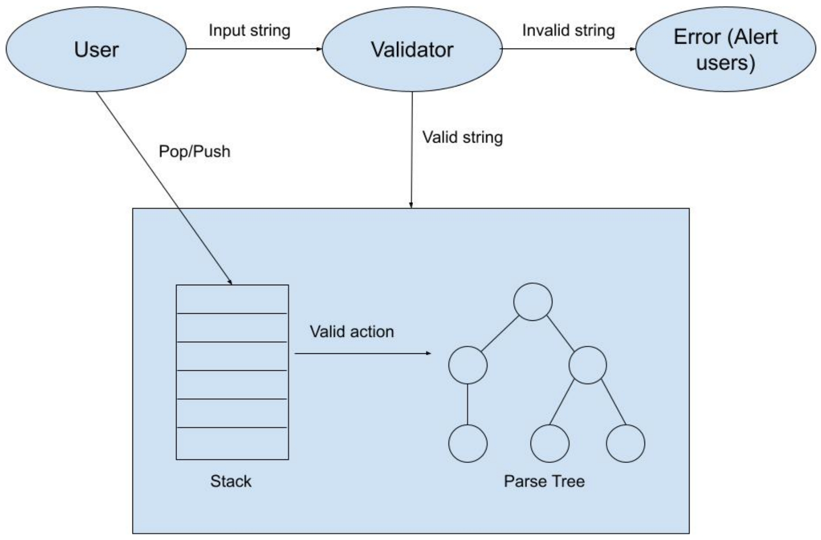


Figure 4.5: Working of parsing section

that excites that cell and does not excite that cell. For a particular cell, this list is then provided to the users, and a parse table with that entry swapped is rendered for the users. Users can perform the same actions as provided for the parse table section to correct the table.

4.3 Back-end

Back-end deals with all the data and database-related stuff. At the back-end, we have created APIs for adding grammar, getting processed data from DB, and storing user activities in DB. When API for adding grammar is hit, it sends the input grammar to the back-end, and there, ParseIT.jar is executed to generate the processed data. This processed data is stored in the database for using it in the future. APIs for getting processed data to return all the processed data requested by the front-end. Front-end sends the grammar ID based on the user's selection of grammar, and the back-end sends the processed data to the front-end. On every action by the user, the user's activity is sent to the back-end through a specific API. All the data collected related to the user's activity is discussed in the next Section 4.3.1.

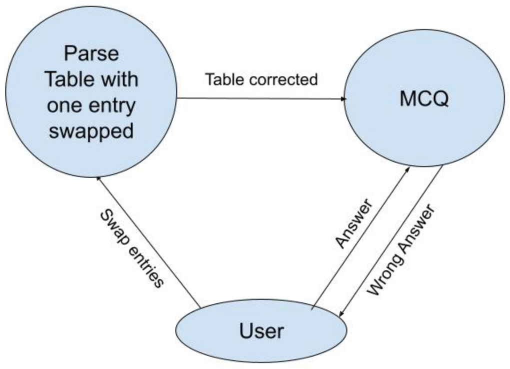


Figure 4.6: Working of parse table game section

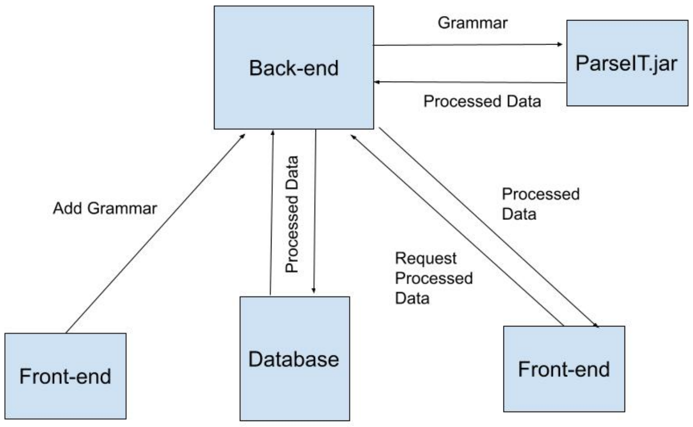


Figure 4.7: Working of Back-end

4.3.1 Database & Data Collection

We are capturing user activities: how users are using this tool, which sections are visited the most by the users, on which grammar students spend more time, etc. All this data is stored in the database. We have four tables in our database named processeddata, userdata, userprofile, and bugreports. For each section, different type of data is collected, but we are storing users' activity for different sections in the same table userdata. There is no specific scheme for the user's data we are collecting; for this reason, we are using a NoSQL database. User's data is stored in the userdata table, and the data fields collected for each section are listed in Tables 4.1, 4.2, 4.3, 4.4, 4.5, 4.6 and 4.7.

First Set

Table 4.1: User data collected for First Set

#	**Field**	**Type**
1.	GrammarID	String
2.	page-name	String
3.	Question	String
4.	options	Array
5.	correct-options	Array
6.	users-selected-options	Array
7.	Question-type	String
8.	is-correct	String
9.	userid	String

Follow Set

Table 4.2: User data collected for Follow Set

#	**Field**	**Type**
1.	GrammarID	String
2.	page-name	String
3.	Question	String
4.	options	Array
5.	correct-options	Array
6.	users-selected-options	Array
7.	Question-type	String
8.	is-correct	String
9.	userid	String

4.4 Potential Studies based on Data Collected

In the previous section, we elaborated on the data collected by ParseIT++. It tries to capture every activity of the user, and it can be used to know how users are using the tool. We can also figure out the improvements required in the tool and additional features that can be added to the tool and can also figure out the

SLR closure section

Table 4.3: User data collected for SLR Closure section

#	Field	Type
1.	GrammarID	String
2.	page-name	String
3.	Question	String
4.	options	Array
5.	correct-options	Array
6.	users-selected-options	Array
7.	Question-type	String
8.	is-correct	String
9.	userid	String

SLR gotos

Table 4.4: User data collected for SLR Goto sections

#	Field	Type
1.	GrammarID	String
2.	page-name	String
3.	Question	String
4.	options	Array
5.	correct-options	Array
6.	users-selected-options	Array
7.	Question-type	String
8.	is-correct	String
9.	userid	String

Parse Table sections

Table 4.5: User data collected for Parse table sections

#	Field	Type
1.	GrammarID	String
2.	page-name	String
3.	prev-row	Number
4.	prev-col	Number
5.	curr-row	Number
6.	curr-col	Number
7.	generated-string	String
8.	is-correct	String
9.	userid	String

Parsing sections

Table 4.6: User data collected for Parsing sections

#	Field	Type
1.	GrammarID	String
2.	page-name	String
3.	input-string	String
4.	is-string-valid	String
5.	stack-action	String
6.	element	String
7.	curr-token	String
8.	is-correct	String
9.	userid	String

SLR Parse Table game section

Table 4.7: User data collected for SLR table game

#	Field	Type
1.	GrammarID	String
2.	page-name	String
3.	prev-row	Number
4.	prev-col	Number
5.	curr-row	Number
6.	curr-col	Number
7.	Question	String
8.	options	Array
9.	correct-options	Array
10.	users-selected-options	Array
11.	is-correct	String
12.	userid	String

information related to the conceptual concepts of parsing. Using this data, we can understand and answer the following questions:

- Which grammar is being used the most?
- Which section is being used the most?
- For a particular grammar which section is used the most?
- For a particular section which grammar is used the most?
- For each user, which grammar is used the most?
- For each user, which section is used the most?
- For each user, for a particular grammar which section is used the most?
- For each user, for a particular section which grammar is used the most?
- How much time do users spend on the tool?
- Percentages for every question asked above.

- It could help to understand how students use the tool, which grammar is difficult to work with, and which section struggles them the most.

Chapter 5

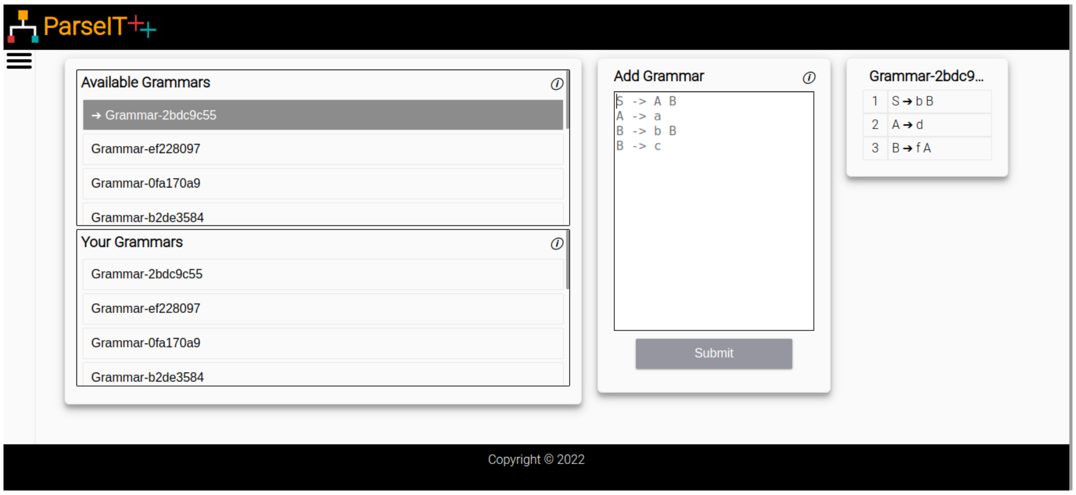
A quick tour of ParseIT++

In this section, we give a quick tour of ParseIT++. The sections below represent the various sections provided by ParseIT++. We show examples of each section for a small grammar. This selected grammar is given below. The actual flow is explained in the dedicated section for each web page.

$$S \rightarrow A\ A$$
$$A \rightarrow a\ A$$
$$A \rightarrow b$$

5.1 Home

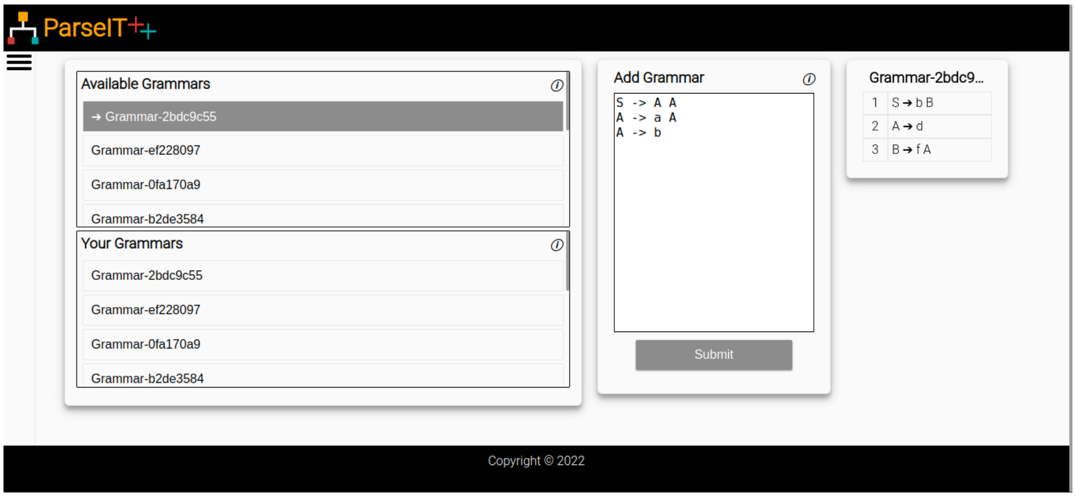
On the home page, all the grammars (Student and instructor grammar) are visible, and users can select any one of them. It also provides the option of adding grammar to users, as can be seen in Figure 5.1 under the “Add Grammar” box. In our case, we are adding above mentioned grammar can be seen in Figure 5.2. After clicking submit button, it sends the grammar to the back-end, and based on the response from the back-end, it shows the message to users. In our case, grammar is successfully added, and it shows the“Grammar Added” message shown in Figure 5.3. Now grammar is visible in available options, and users can select this grammar, as shown in Figure 5.4.



```
S -> A B
A -> a
B -> b B
B -> c
```

1	S → b B
2	A → d
3	B → f A

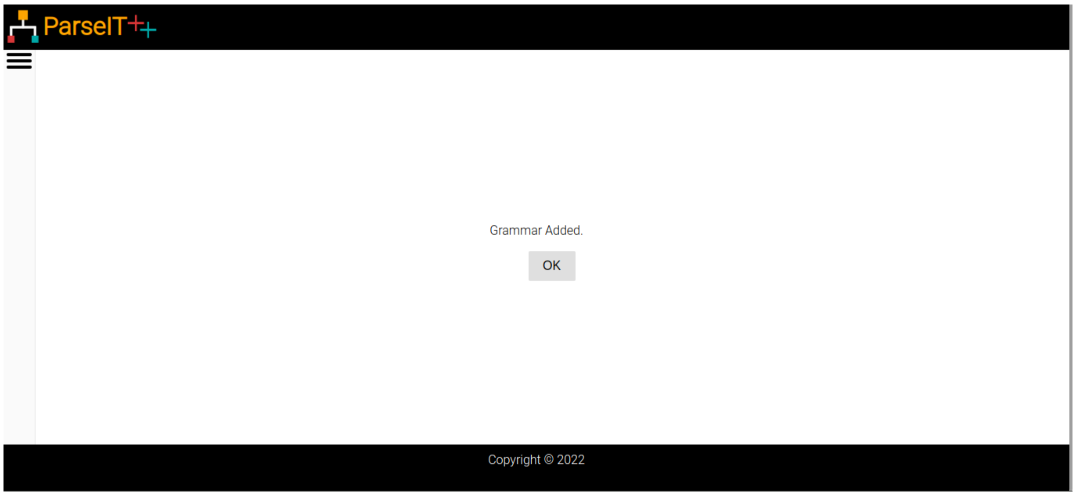
Figure 5.1: Home page



```
S -> A A
A -> a A
A -> b
```

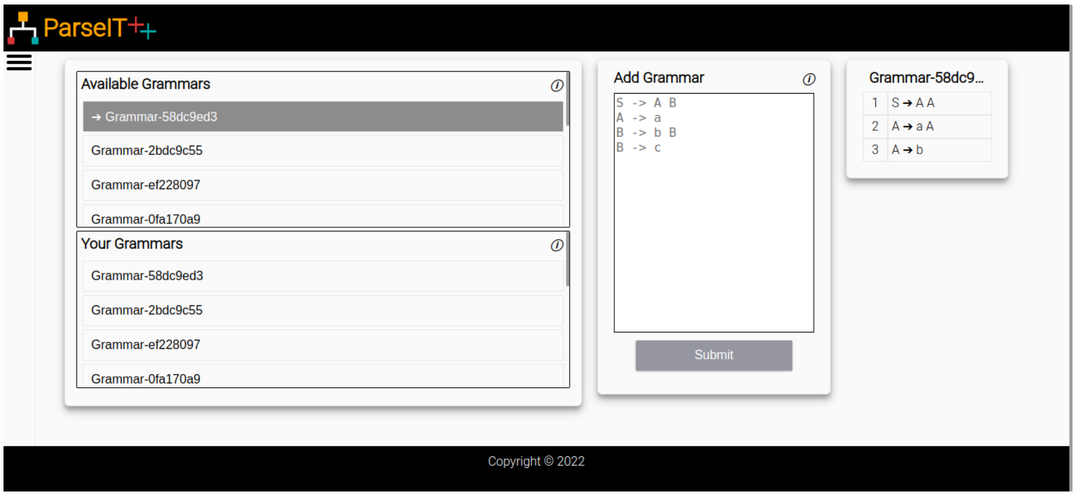
1	S → b B
2	A → d
3	B → f A

Figure 5.2: Adding a grammar



Grammar Added.

Figure 5.3: Success message for addition of grammar



```
S -> A B
A -> a
B -> b B
B -> c
```

1	S ➔ A A
2	A ➔ a A
3	A ➔ b

Figure 5.4: Selection of added grammar

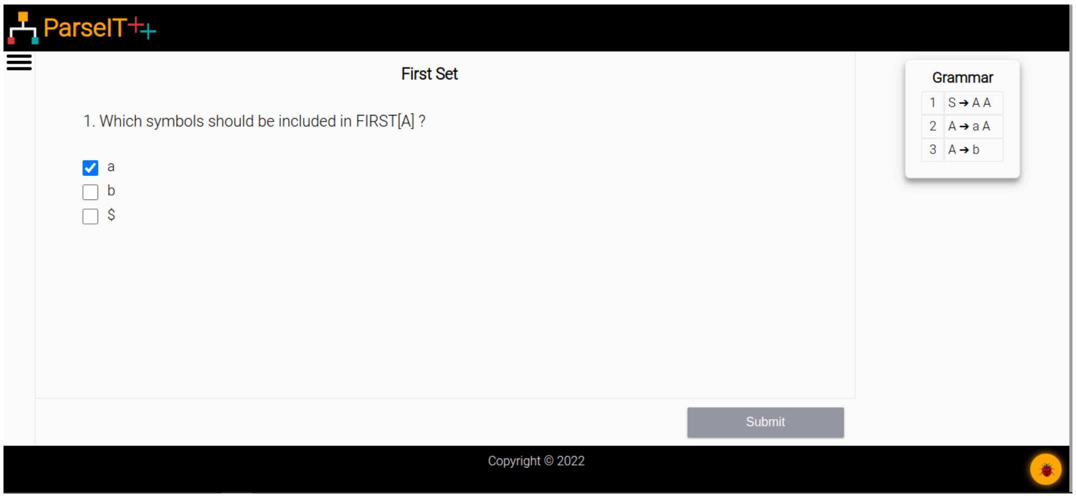
5.2 MCQ-Based Sections

This section provides a question for the first set, and users have to select the options for the available options. For our example grammar, it generates the question "Which symbol should be included in FIRST[A]?" with options 'a', 'b', and '$' as can be seen in Figure 5.5.

Users have to select the correct options from the available options. ParseIT++ generates a Hint question of type-I if users select the wrong options, which is depicted in Figure 5.6. It generates hint questions of type-II for options that are correct but are left by the users, which can be seen in Figure 5.7. If the user selects the wrong options for a hint question, then the same hint question is asked again. In Figure 5.5 user selects the first option, which is 'a', which is correct. If the user selects the correct answers, then the next hint question or questions on the selected sections are asked.

5.3 Parse Table Sections

In this section, users have to unshuffle the given shuffled parse table. For the selected grammar, the shuffled parse table generated by ParseIT++ is shown in Figure 5.8. Users have to unshuffle it by swapping two entries. In Figure 5.8, a string and pare



First Set

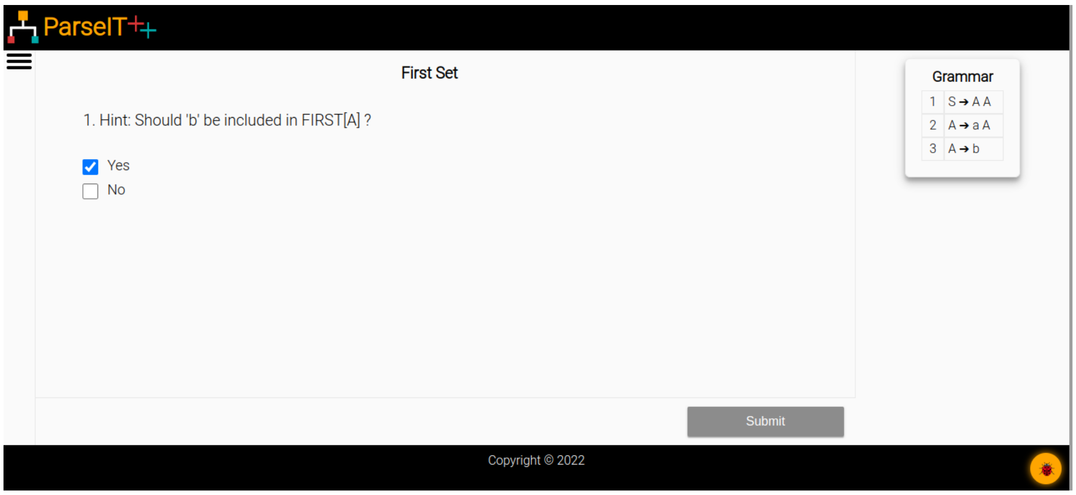
1. Which symbols should be included in FIRST[A] ?

- [✓ a]
- b
- $

Grammar

1	$S \rightarrow A\ A$
2	$A \rightarrow a\ A$
3	$A \rightarrow b$

Figure 5.5: MCQ based section (First Set)



First Set

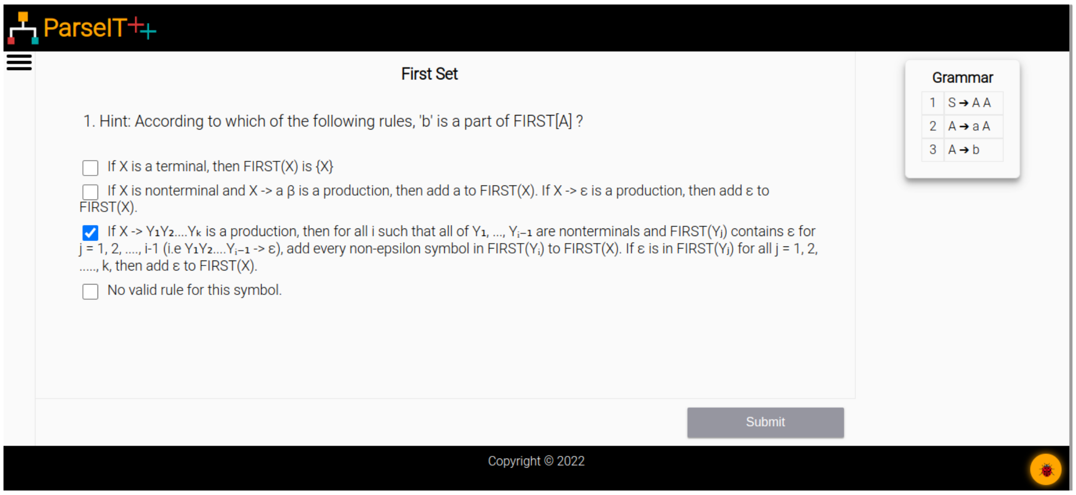
1. Hint: Should 'b' be included in FIRST[A] ?

- [✓ Yes]
- No

Grammar

1	$S \rightarrow A\ A$
2	$A \rightarrow a\ A$
3	$A \rightarrow b$

Figure 5.6: Type-I Hint question for First set



First Set

1. Hint: According to which of the following rules, 'b' is a part of FIRST[A] ?

- If X is a terminal, then FIRST(X) is {X}
- If X is nonterminal and X -> a β is a production, then add a to FIRST(X). If X -> ε is a production, then add ε to FIRST(X).
- [✓ If X -> $Y_1Y_2....Y_k$ is a production, then for all i such that all of $Y_1, ..., Y_{i-1}$ are nonterminals and FIRST(Y_j) contains ε for j = 1, 2, ..., i-1 (i.e $Y_1Y_2....Y_{i-1}$ -> ε), add every non-epsilon symbol in FIRST(Y_i) to FIRST(X). If ε is in FIRST(Y_j) for all j = 1, 2, ..., k, then add ε to FIRST(X).]
- No valid rule for this symbol.

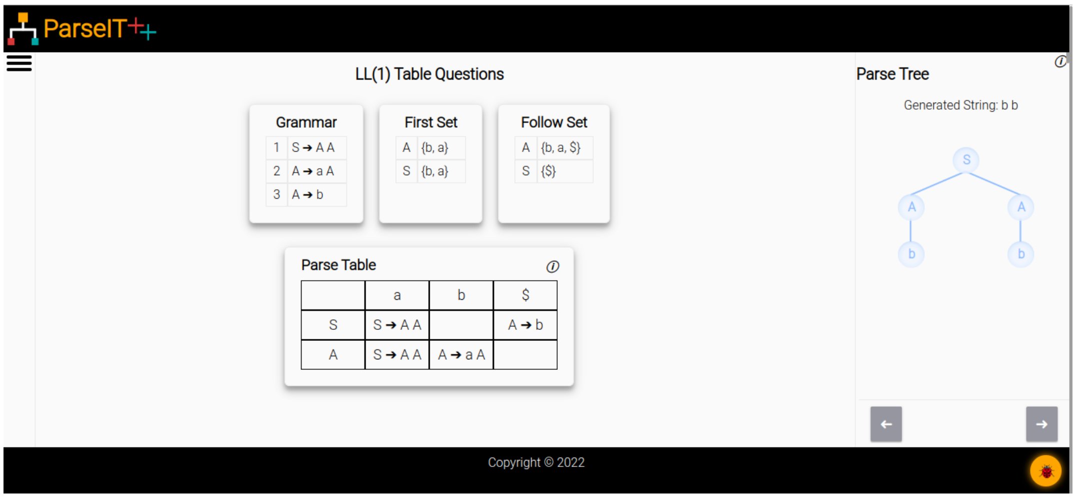
Grammar

1	$S \rightarrow A\ A$
2	$A \rightarrow a\ A$
3	$A \rightarrow b$

Figure 5.7: Type-II Hint question for First set

tree are given; parsing this string will excite the first erroneous entry in the parse table row major-wise. The tree presented below is the parse tree of the generated string, and users can view the construction using the arrow button available at the bottom-left.

Users have to swap entries to unshuffle the table; an entry swap is shown in Figure 5.9. When the user corrects the entry corresponding to which string is generated, ParseIT++ generates another string corresponding to the first erroneous entry in the table. This keeps on till the user completely unshuffles the table, as can be seen in the Figure 5.10



LL(1) Table Questions

Grammar

1	S → A A
2	A → a A
3	A → b

First Set

A	{b, a}
S	{b, a}

Follow Set

A	{b, a, $}
S	{$}

Parse Table

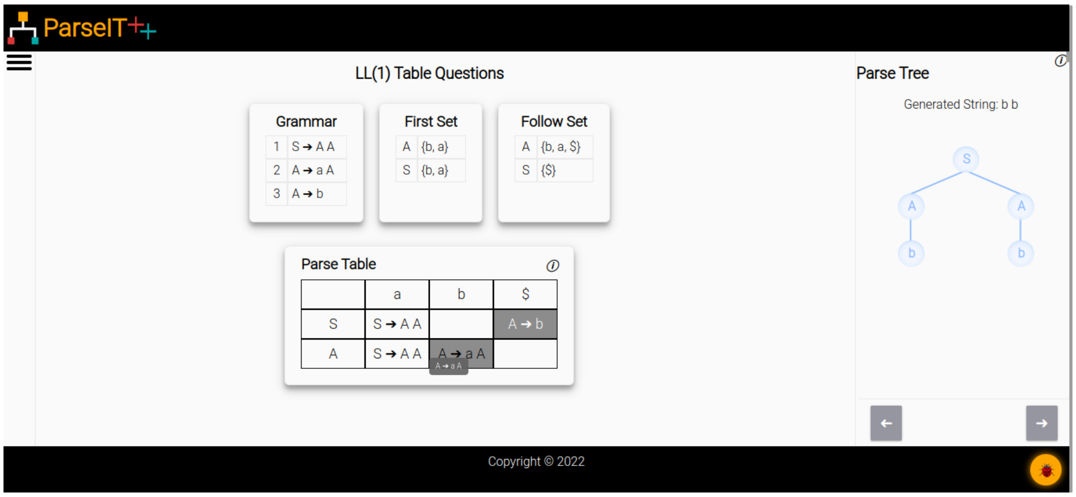
	a	b	$
S	S → A A		A → b
A	S → A A	A → a A	

Parse Tree

Generated String: b b

Copyright © 2022

Figure 5.8: Parse table section for LL(1)



LL(1) Table Questions

Grammar

1	S → A A
2	A → a A
3	A → b

First Set

A	{b, a}
S	{b, a}

Follow Set

A	{b, a, $}
S	{$}

Parse Table

	a	b	$
S	S → A A		A → b
A	S → A A	A → a A	

Parse Tree

Generated String: b b

Copyright © 2022

Figure 5.9: Parse table section showing entry swap

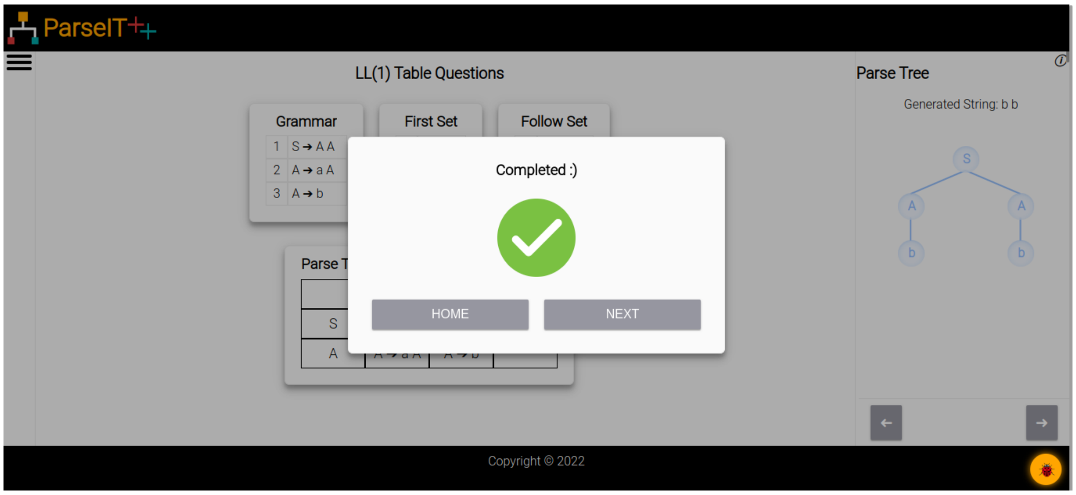


Figure 5.10: Parse table section after shuffling the parse table

5.4 Parsing Sections

In this section, users have to input a string and then have to complete the parsing stack. If the input string is invalid, then it is altered to the user as shown in the Figure 5.11, the user has entered string “a b a”. In case of a valid input string, users are allowed to push/pop operations on the stack.

As shown in the Figure 5.12 user has input string “a b b”. Now users have to perform actions on the provided stack. In Figures 5.13, 5.14, user try to perform incorrect stack operations PUSH and POP respectively and are altered to user by converting buttons’ in reddish colour. On performing correct actions on the stack, a parse tree is built. Building of tree can be seen to the right of parsing stack in the Figures 5.13, 5.14. After parsing the complete string, the complete parse tree is generated, and a pop-up message is shown as visible in Figure, 5.15.

5.5 Parse Table Game Section

In this section also, the user gets a parse table in which they have to perform the actions, but only one entry in the table is misplaced. Users are given some strings from which only one string on parsing can excite that entry. Like in our case it can be seen in Figure 5.16 and strings are “bb”, “abab” and “aabb”. Users have to

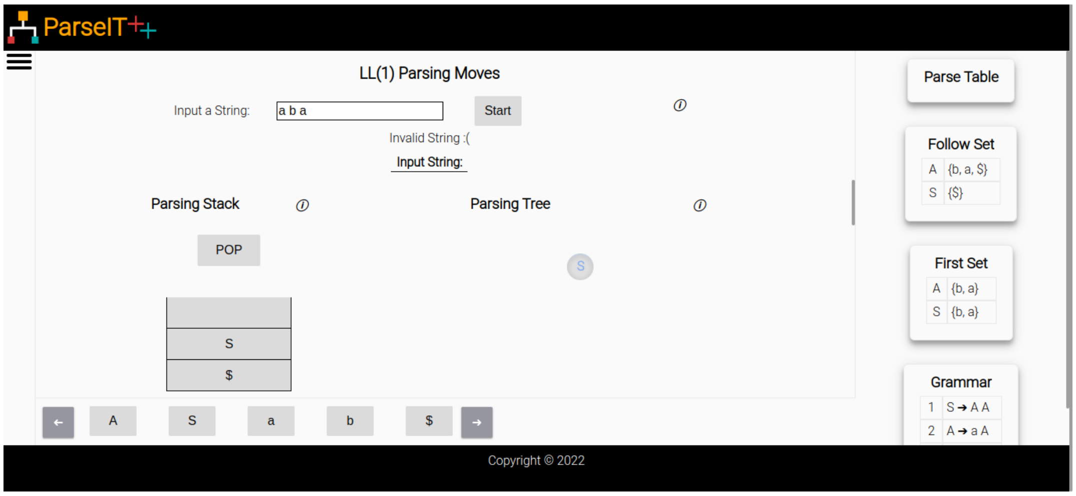


Figure 5.11: Parsing section page for LL(1) with invalid input string

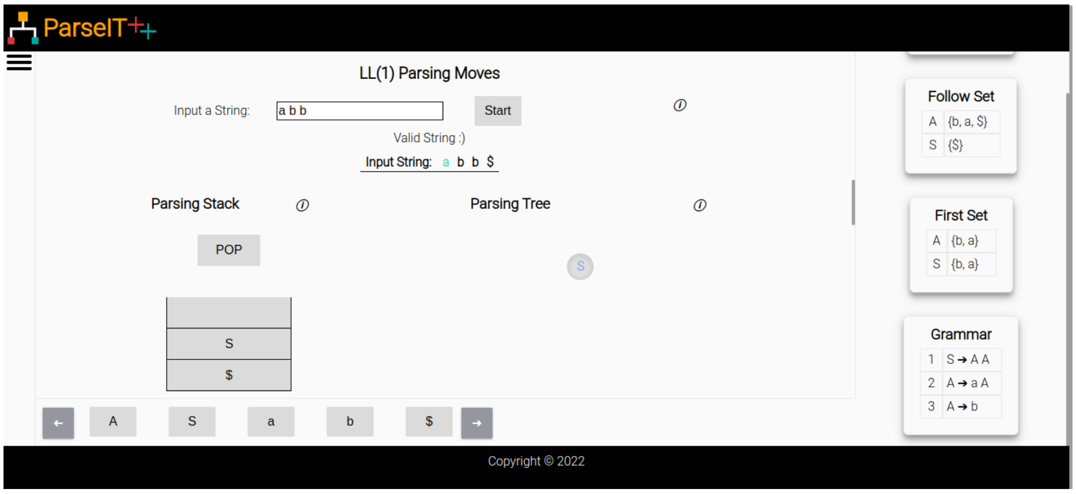


Figure 5.12: Parsing section page for LL(1) with valid input string

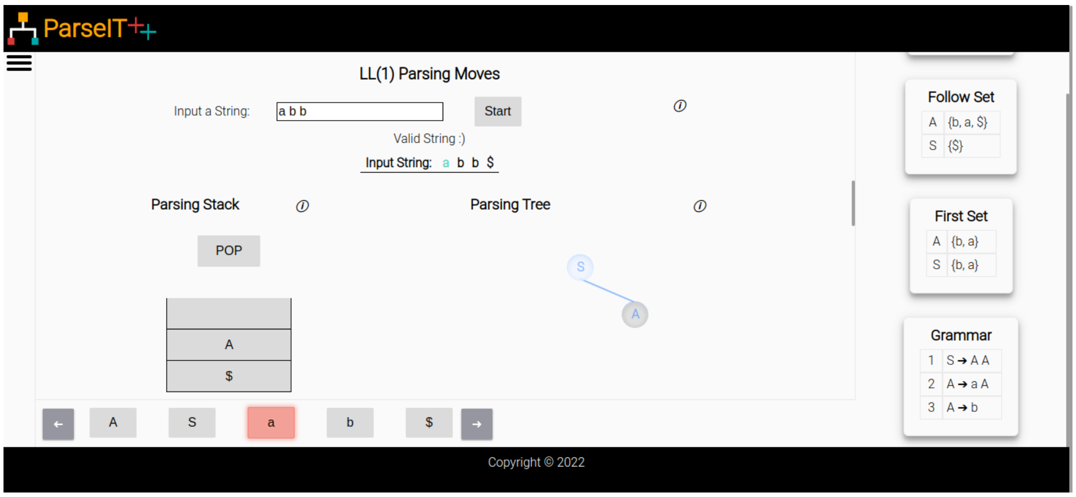
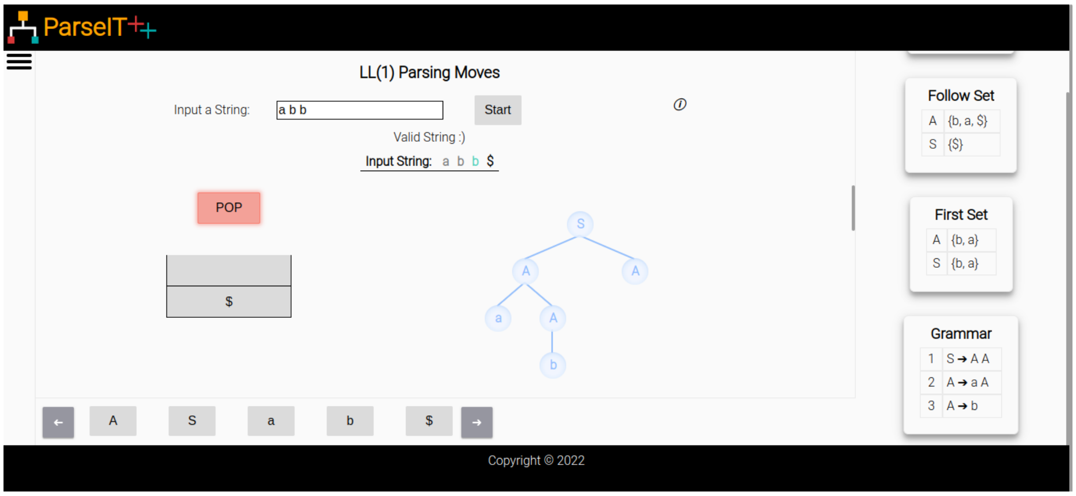


Figure 5.13: Parsing section page showing wrong stack action (Push)



Valid String :)

Input String: a b b $

Follow Set

A	{b, a, $}
S	{$}

First Set

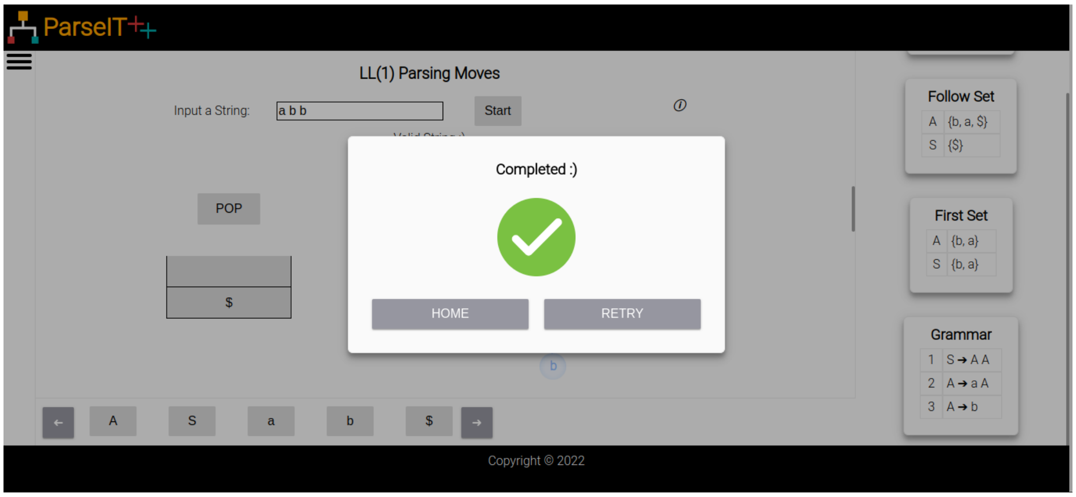
A	{b, a}
S	{b, a}

Grammar

1	S → A A
2	A → a A
3	A → b

Copyright © 2022

Figure 5.14: Parsing section page showing wrong stack action (POP)



Completed :)

Follow Set

A	{b, a, $}
S	{$}

First Set

A	{b, a}
S	{b, a}

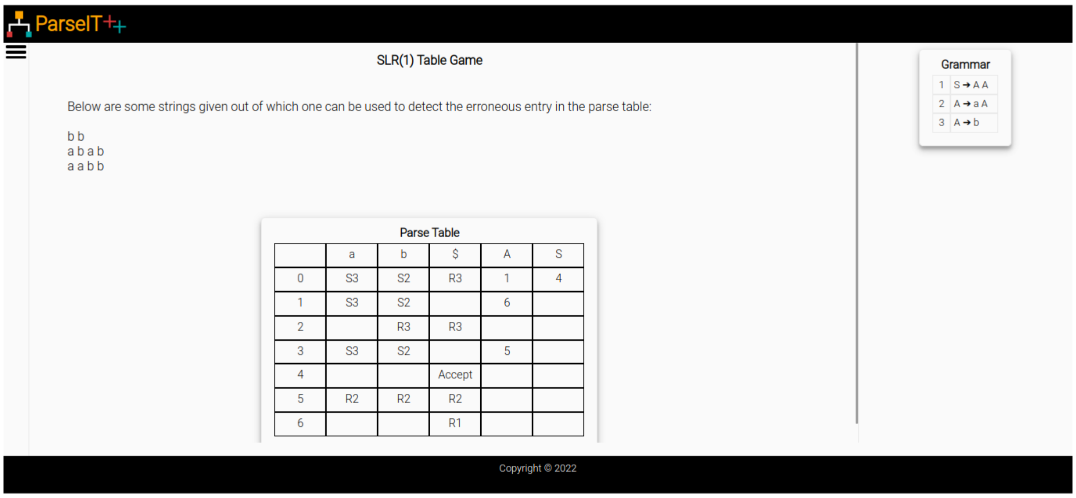
Grammar

1	S → A A
2	A → a A
3	A → b

Copyright © 2022

Figure 5.15: Parsing section after completing string parsing

correct the parse table by swapping the entries similar to that in the parse table section. After correcting the table, users are asked to choose which string can be used to detect that entry, as seen in Figure 5.17. Users have to select the option from the available options and click submit. If the user selects the wrong option, the same question will be asked as in Figure 5.17 user selects string "bb", which is incorrect. After submitting the correct option, users can either replay the game or go to the home page, which can be seen in Figure 5.18.



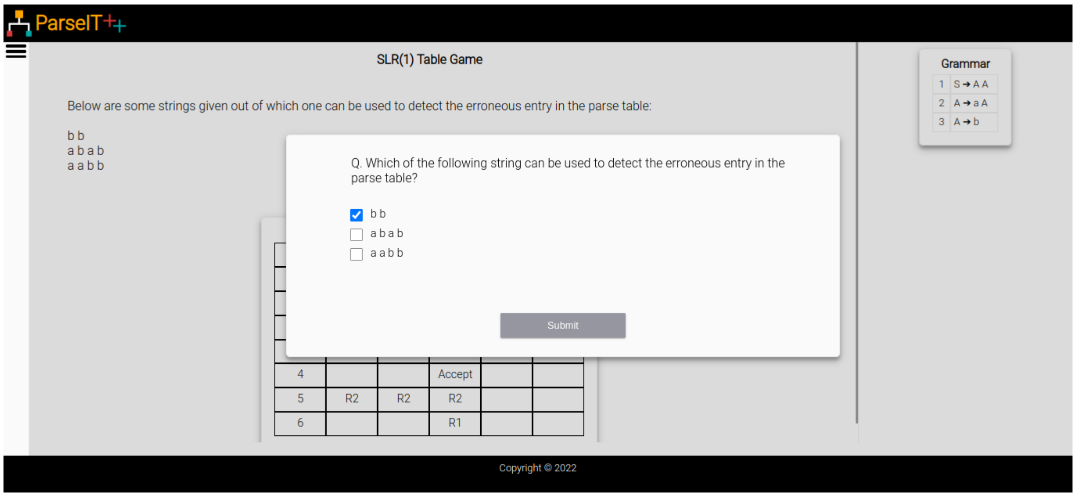
Below are some strings given out of which one can be used to detect the erroneous entry in the parse table:

b b
a b a b
a a b b

1	S → A A
2	A → a A
3	A → b

	a	b	$	A	S
0	S3	S2	R3	1	4
1	S3	S2		6	
2		R3	R3		
3	S3	S2		5	
4			Accept		
5	R2	R2	R2		
6			R1		

Figure 5.16: SLR table game section showing the table and parsing strings



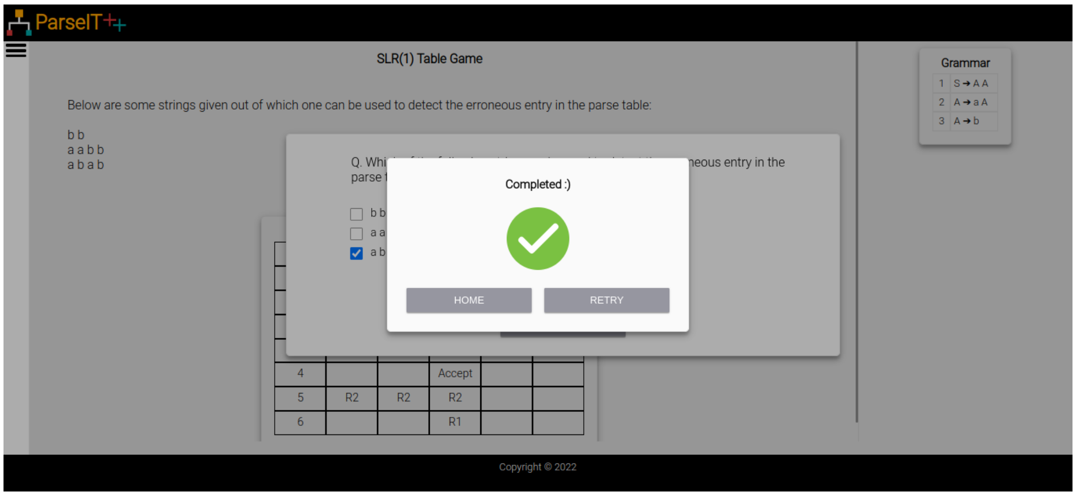
Below are some strings given out of which one can be used to detect the erroneous entry in the parse table:

b b
a b a b
a a b b

Q. Which of the following string can be used to detect the erroneous entry in the parse table?

[✓ b b]
a b a b
a a b b

Figure 5.17: SLR table game section showing the asked question



Below are some strings given out of which one can be used to detect the erroneous entry in the parse table:

b b
a a b b
a b a b

Q. Whi… neous entry in the parse t…

- [ ] b b
- [ ] a a
- [x] a b

Completed :)

1	S → A A
2	A → a A
3	A → b

4			Accept		
5	R2	R2	R2		
6			R1		

Figure 5.18: SLR table game section after submitting correct option

Chapter 6

User Manual

There is no significant difference between student users and admin users. The interface for both types of users is similar. ParseIT++ have 10 sections named First Set, Follow Set, LL(1) Table, LL(1) Parsing, SLR(1) Closure, SLR(1) Goto-I, SLR(1) Goto-II, SLR(1) Table, SLR(1) Parsing, and SLR(1) Table Game. MCQ-based sections like First Set, Follow Set, SLR(1) Closure, SLR(1) Goto-I, and SLR(1) Goto-II have the same interface. Parse Table sections like LL(1) Table and SLR(1) Table have a similar interface, Parsing sections LL(1) parsing and SLR(1) Parsing have similar interface and SLR(1) Table game have very little difference in interface than that of Table sections.

6.1 Home

On the Home page, users can select a grammar on which they want to work or can add grammar. The tool can generate problems for the selected grammar. Users can select a grammar by clicking on the grammar listed on the page. To add grammar, users have to enter the grammar in the provided box on the page in a specific format and click submit. The format for the grammar to be added is as follows:

- Start symbol should be S.
- Symbols should be space separated.

- Non-terminals should be single character capital letter.
- Terminals should be small letters and can be of multiple characters.
- For inserting epsilon production write whole epsilon word for eg. S -> epsilon.
- LHS and RHS should be separated by "->" with a space on each side of "->" for eg. S -> a B.
- Each production rule should be in a separate line.
- Grammar should contain more than one production rule

6.2 MCQ-Based Sections

In these sections, an MCQ question is given. Users have to select all the applicable options from the list of available options and then click on submit button. If the user selects the wrong options, then hint questions will be asked, and users have to answer them correctly. If users leave a correct option, then a different type of hint question will be asked. If users select the wrong options for hint questions, then the same hint question will be asked again. Users can use the grammar given to the right to answer these questions.

6.3 Parse Table Sections

In this section, a shuffled parse table will be given. Users have to unshuffle this parse table by swapping the entries of the parse table. For swapping the entries, select a cell by clicking on it and then click on the cell with which you want to swap. Grammar First set and Follow set are given, which can help users to unshuffle the table. A string will be generated that, on parsing, will excite an erroneous entry in the table. Users can see a parse tree on the left of the generated string. Users can check how this parse tree is built by clicking on the left/right button present at the bottom of the Parse tree.

6.4 Parsing Sections

This section is provided to understand the string parsing and parse tree generation process. In this section, users have to input a valid string and click on the start button, for the invalid string will be indicated. Users have to push/pop from the stack in a similar way as while parsing the input string. Symbol buttons are given on the bottom, and users have to click on these buttons to push a symbol into the stack. Users can use the POP button available just above the stack to pop from the stack. In the Parse Tree area, a parse tree for the input string will be built as users work on the stack.

6.5 Parse Table Game Section

This section is similar to the Parse Table section, with an MCQ question asked at the end. Users can use this section in a similar way as the Parse table section and MCQ question-based section, depending on the applicable scenario.

Chapter 7

Conclusions & Future Work

We have built a web-based system that can be used as a teaching tool for parsing techniques. We have extended the current work (ParseIT) by adding GUI and visualizations features. It can be used to teach parsing techniques in an undergraduate compiler courses. It could be used as a combined tool for teaching theoretical and practical aspects of parsing. The tool is built to be very interactive and can derive student's interest in the compiler courses. We have built this tool with the capability to capture activities of the users.

We could not use this tool in an actual compiler course because of the delay in the tool's completion. Compiler course is offered once a year so it can't be used in classroom for teaching parsing. In future we can use this tool in a classroom for teaching parsing. This could give us data on which separate studies can be performed to gather information about the difficulties in the course, requirements of the students, student's usage of the tool, etc. Currently tool only provides sections for LL(1) and SLR(1), we can extend it to cover more parsing techniques and more theoretical concepts of parsing.

References

[1] Susan H. Rodger et al. “Increasing Engagement in Automata Theory with JFLAP”. In: *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*. SIGCSE ’09. Chattanooga, TN, USA: Association for Computing Machinery, 2009, pp. 403–407. ISBN: 9781605581835. DOI: `10.1145/1508865.1509011`. URL: `https://doi.org/10.1145/1508865.1509011`.

[2] S. H. Rodger. *JFLAP*. last accessed Jan 2022. 2022. URL: `https://www.jflap.org/`.

[3] Amey Karkare and Nimisha Agrawal. “ParseIT: A Tool for Teaching Parsing Techniques”. In: *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. SIGCSE ’16. Memphis, Tennessee, USA: Association for Computing Machinery, 2016, p. 590. ISBN: 9781450336857. DOI: `10.1145/2839509.2850513`. URL: `https://doi.org/10.1145/2839509.2850513`.

[4] Nimisha Agarwal. *ParseIT*. last accessed Jan 2022. 2016. URL: `https://cse.iitk.ac.in/users/nimisha/parseit/index.html`.

[5] Alfred V. Aho et al. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. USA: Addison-Wesley Longman Publishing Co., Inc., 2006. ISBN: 0321486811.

[6] Zak Kincaid and Shaowei Zhu. *LL(1) Tool*. last accessed Jan 2022. 2022. URL: `https://www.cs.princeton.edu/courses/archive/spring20/cos320/LL1/`.

[7] Zak Kincaid and Shaowei Zhu. *LR(0) Tool*. last accessed Jan 2022. 2022. URL: `https://www.cs.princeton.edu/courses/archive/spring20/cos320/LR0/`.

[8] Sphere Research Labs. *SPOJ*. last accessed Jan 2022. 2022. URL: `https://www.spoj.com/`.

[9] Unacademy. *CodeChef*. last accessed Jan 2022. 2022. URL: `https://www.codechef.com/`.

[10] kanpur IIT. *Prutor*. last accessed Jan 2022. 2022. URL: `https://esc101.cse.iitk.ac.in/`.

[11] National University of Singapore. *VISUALGO*. last accessed Jan 2022. 2022. URL: `https://visualgo.net/en`.

[12] Mohamed Hamada. “Web-based Active e-Learning Tools for Automata Theory”. In: *Seventh IEEE International Conference on Advanced Learning Technologies (ICALT 2007)*. 2007, pp. 877–879. DOI: `10.1109/ICALT.2007.283`.

[13] Loris D'Antoni et al. "Automata Tutor v3". In: *Computer Aided Verification.* Ed. by Shuvendu K. Lahiri and Chao Wang. Cham: Springer International Publishing, 2020, pp. 3–14. ISBN: 978-3-030-53291-8.

[14] Rafael del Vado Vírseda. "An Interactive Tutoring System for Learning Language Processing and Compiler Design". In: *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education.* ITiCSE '20. Trondheim, Norway: Association for Computing Machinery, 2020, p. 552. ISBN: 9781450368742. DOI: `10.1145/3341525.3393969`. URL: `https://doi.org/10.1145/3341525.3393969`.

[15] Marjan Mernik and V. Zumer. "An educational tool for teaching compiler construction". In: *Education, IEEE Transactions on* 46 (Mar. 2003), pp. 61–68. DOI: `10.1109/TE.2002.808277`.

[16] Bison. *GNU Bison.* last accessed Jan 2022. 2022. URL: `https://www.gnu.org/software/bison/`.

[17] Scott E Hudson et al. "CUP parser generator for Java, 1997". In: (2006).

[18] Elliot Berk. "JLex: A lexical analyzer generator for Java, 1997". In: (1997).

[19] M. E. Lesk and E. Schmidt. *Lex–a Lexical Analyzer Generator. In UNIX Vol. II: Research System (10th Ed.)* 1990, pp. 375–387. ISBN: 0030475295.

[20] Vern Paxson et al. "Flex–fast lexical analyzer generator". In: *Lawrence Berkeley Laboratory* (1995).

[21] David Beazley. "PLY (Python Lex Yacc)". In: (2022). last accessed Jan 2022. URL: `https://www.dabeaz.com/ply/`.

[22] Alfred V. Aho, Mahadevan Ganapathi, and Steven W. K. Tjiang. "Code Generation Using Tree Matching and Dynamic Programming". In: *ACM Trans. Program. Lang. Syst.* 11.4 (Oct. 1989), pp. 491–516. ISSN: 0164-0925. DOI: `10.1145/69558.75700`. URL: `https://doi.org/10.1145/69558.75700`.

[23] Christopher W. Fraser and Todd A. Proebsting. "Finite-State Code Generation". In: *Proceedings of the ACM SIGPLAN 1999 Conference on Programming Language Design and Implementation.* PLDI '99. 1999, pp. 270–280. ISBN: 1581130945. URL: `https://doi.org/10.1145/301618.301680`.

[24] Christopher W. Fraser, Robert R. Henry, and Todd A. Proebsting. "BURG: Fast Optimal Instruction Selection and Tree Parsing". In: *SIGPLAN Not.* (Apr. 1992), pp. 68–76. URL: `https://doi.org/10.1145/131080.131089`.

[25] K. John Gough. "Bottom-up Tree Rewriting Tool MBURG". In: *SIGPLAN Not.* (Jan. 1996), pp. 28–31. DOI: `10.1145/249094.249110`. URL: `https://doi.org/10.1145/249094.249110`.

[26] T. J. Parr and R. W. Quong. "ANTLR: A Predicated LL(k) Parser Generator". In: *Softw. Pract. Exper.* 25.7 (July 1995), pp. 789–810. ISSN: 0038-0644. DOI: `10.1002/spe.4380250705`. URL: `https://doi.org/10.1002/spe.4380250705`.

[27] Stephen C. Johnson. *Yacc: Yet Another Compiler-Compiler.* Tech. rep. 1979.