




PROJECT AND TEAM INFORMATION

Project Title

Interactive Parser Visualizer

Student / Team Information

<p>Team Name:</p> <p>Team id :</p>	<p>PBL - CD</p> <p>CD-VI-T077</p>
<p>Team member 1 (Team Lead)</p> <p>Singh,Jasmeet - 22011017</p> <p>singhjasmeet200526@gmail.com</p>	
<p>Team member 2</p> <p>Negi,Karan - 22021583</p> <p>karanegi076@gmail.com</p>	
<p>Team member 3</p> <p>Sati,Anshul - 220211252</p> <p>satianshul9@gmail.com</p>	

PROPOSAL DESCRIPTION (10 pts)

Motivation (1 pt)

In the field of compiler design, parsing plays a vital role in analyzing the syntactic structure of source code. Understanding how different types of parsers function—especially **top-down** (like LL) and **bottom-up** (like LR, SLR, CLR)—is crucial for both theoretical knowledge and practical implementation of compilers.

This project is motivated by the desire to **deepen our understanding of parsing techniques** by implementing and comparing multiple parsers. By allowing users to input custom grammars and observe each step of the parsing process—such as the generation of FIRST and FOLLOW sets, construction of parsing tables, and real-time stack/input trace—we aim to create a tool that is both **educational and insightful**.

Moreover, our system highlights **the internal decision-making process of parsers**, making abstract concepts like shift/reduce actions and grammar conflicts (e.g., shift/reduce and reduce/reduce) more tangible. By comparing top-down and bottom-up approaches side by side, users can better grasp their differences, advantages, and limitations.

Ultimately, this project serves as an **interactive learning platform** for students and enthusiasts, bridging the gap between theory and practice in compiler construction.

State of the Art / Current solution (1 pt)

Currently, parser implementation is often explored using tools like **YACC**, **Bison**, or **ANTLR**, which generate parsers automatically from grammar specifications. While powerful, these tools abstract away the internal working of parsers—such as how FIRST and FOLLOW sets are computed, how parsing tables are built, or how the input is processed step-by-step using a stack. Moreover, most existing platforms focus on delivering final results rather than teaching the process. They often skip detailed step-by-step visualization, making it difficult for learners to understand **how** and **why** each parsing decision is made. This lack of transparency creates a gap between theory and practical comprehension.

Project Goals and Milestones (2 pts)

Project Goals:

- To build a learning-focused parser simulator that visually demonstrates how a grammar is processed by different types of parsers.
- To help users understand parsing techniques by breaking down complex steps like **FIRST** & **FOLLOW** computation and **parsing table generation** into clear visuals.
- To simulate the entire parsing process including **parsing table lookup, stack movement, and input consumption** in real-time.
- To provide detailed **step-by-step tracing** for every string evaluation—something that's often skipped in traditional tools.
- To bridge the gap between parser theory and real execution, making abstract compiler concepts easier to grasp.
- To allow users to **compare the behavior and results of multiple parsing techniques**, enhancing conceptual clarity through side-by-side evaluation.

Project Milestones

Frontend

- Create a user interface to input grammar and test strings.
- Display FIRST and FOLLOW sets in a readable format.
- Show parsing table with visual indicators for conflicts.
- Animate stack and input buffer updates during parsing.
- Add buttons for step-by-step or full parsing execution.

Backend

- Process grammar input and separate terminals/non-terminals.
- Calculate accurate FIRST and FOLLOW sets including ϵ cases.
- Run parser logic and return detailed parsing steps.
- Build respective parsing tables based on different grammar rules.
- Detect and highlight shift/reduce or reduce/reduce conflicts.

Project Approach (3 pts)

Backend Development (Python):

The backend serves as the core engine of the system and is being developed in Python. It handles all the computational aspects of parsing, including:

- Accepting user-defined grammars and validating them.
- Computing **FIRST** and **FOLLOW** sets, essential for constructing parsing tables.
- Building the **parsing table** for the selected parsing method (starting with CLR).

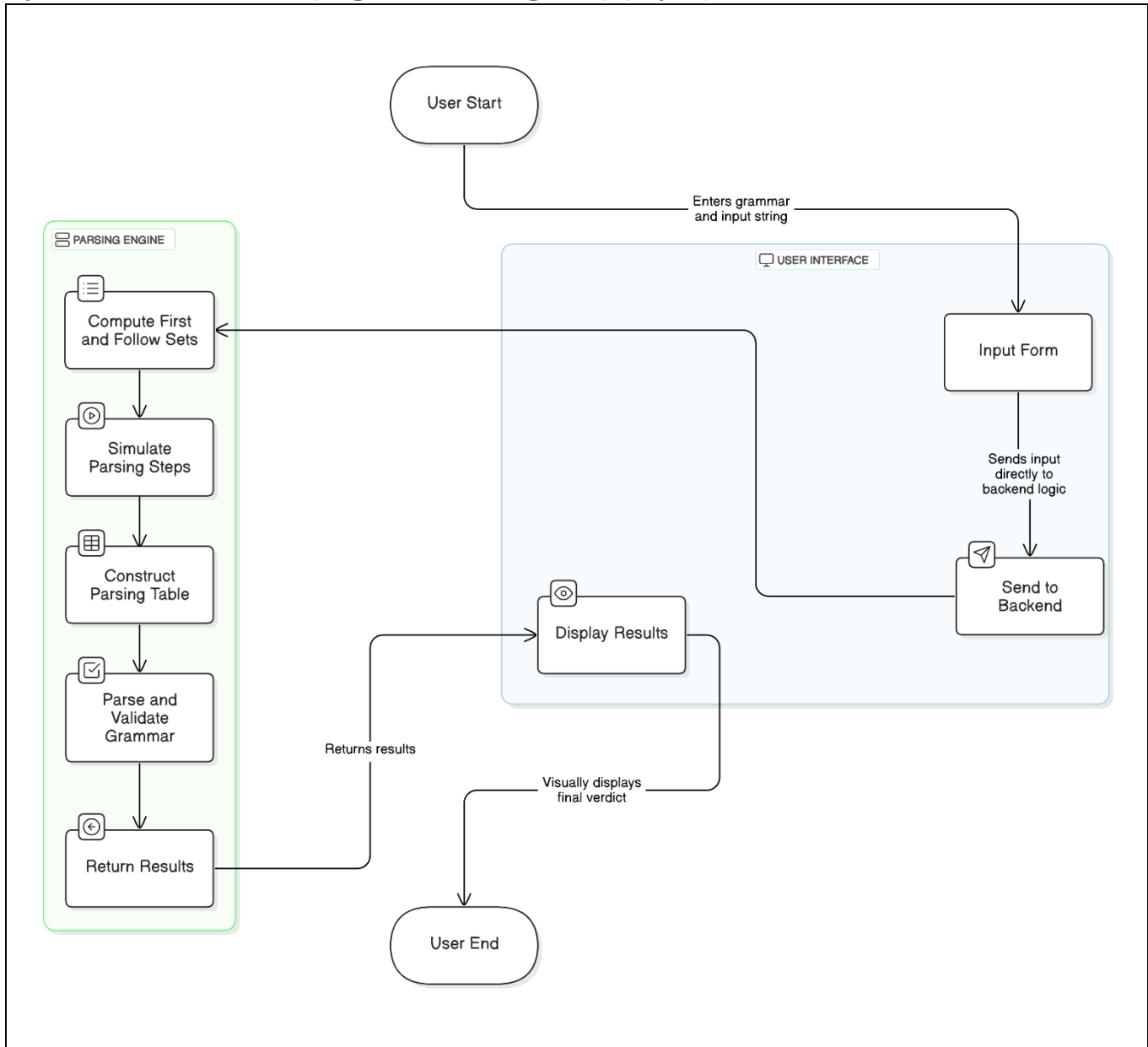
- Simulating the parsing process step-by-step, showing how the input is processed, how the stack evolves, and what actions (shift, reduce, accept) are taken at each step.

Frontend Interface (Planned – React):

The frontend aims to offer a **visually rich and interactive experience** to users. Though ReactJS is the current candidate, the final framework is yet to be confirmed. The frontend will:

- Provide input fields for users to define grammars and test strings.
- Visually display the computed parsing table, highlighting entries and conflicts (like shift/reduce).
- Animate the entire parsing process in real-time, including the stack, input buffer, and current action.
- Offer clear, intuitive visuals to help users understand how each grammar rule is applied during parsing.
- Support parser comparison in future iterations by switching between different parsing modes.

System Architecture (High Level Diagram) (2 pts)



Project Outcome / Deliverables (1 pts)

- A complete parser simulation tool developed using Python for backend logic.
- Interactive frontend allowing users to:
 - Enter custom grammars
 - Generate FIRST and FOLLOW sets
 - View respective Parsing Table
 - Provide input strings for parsing
- Step-by-step visualization of the parsing process:
 - Stack and input trace at each stage
 - Shift, reduce, and accept/reject actions shown clearly

- User-friendly interface to understand internal workings of the parser.
- Designed for educational purposes—helping students visualize and understand parsing concepts practically.
- Parser comparison feature that allows users to compare parsing behavior and results across different strategies, helping understand their strengths and limitations.

Assumptions

- The grammar provided by the user will be context-free and in a correct format (e.g., $S \rightarrow AB$).
- Grammar will use single-character non-terminals (A–Z) and terminals (a–z or symbols).
- No left-recursion or ambiguity handling is implemented for now.
- The tool assumes that users have a basic understanding of grammar rules and parsing

References

- Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2006). Compilers: Principles, Techniques, and Tools (2nd ed.). Pearson Education. : [Compilers Principles Techniques and Tools \(2nd Edition\)](#)
- Torczon, L., & Cooper, K. D. (2011). Engineering a Compiler (2nd ed.). Morgan Kaufmann.: [Engineering A Compiler 2nd Edition by Cooper and Torczon](#)
- Tutorialspoint. Compiler Design - Parsing. Retrieved from: https://www.tutorialspoint.com/compiler_design/index.htm
- GeeksforGeeks. Parser in Compiler Design. Retrieved from: <https://www.geeksforgeeks.org/introduction-of-parsing-ambiguity-and-parsers-set-1/?ref=lbp>
- <https://github.com/gbroques/compiler>