Parser Generator Performance Analysis

Abstract:

This case study aims to perform a comprehensive analysis of the performance of popular parser generators in the domain of compiler construction. Parser generators play a crucial role in converting source code into a format that can be understood and executed by a computer. This study compares the performance of two widely used parser generators, Yacc and Bison, by implementing parsers for various grammars and evaluating their execution times.

1. Introduction:

Parser generators are essential tools in compiler construction, automating the process of generating parsers from formal grammars. Yacc and Bison are two well-known parser generators that have been used extensively in the development of compilers. However, a thorough performance analysis is necessary to understand how these tools perform under different scenarios.

2. Methodology:

To conduct this performance analysis, we implemented parsers for a set of grammars using both Yacc and Bison. The grammars were carefully chosen to represent a mix of complexities, including ambiguous constructs and various levels of nesting. The goal is to evaluate how these parser generators handle different aspects of language syntax.

3. Implementation:

The grammars chosen for the study cover a range of language constructs, from simple arithmetic expressions to more complex control structures. Each grammar was implemented using both Yacc and Bison, and the resulting parsers were integrated into a simple compiler framework.

5. Performance Metrics:

Key performance metrics were considered in the analysis, including parsing speed, memory usage, and error handling. Execution times were measured for each parser using a standardized set of inputs, and memory consumption was monitored throughout the parsing process. The handling of syntax errors and error recovery mechanisms was also evaluated.

6. Results:

The results of the analysis revealed interesting insights into the performance of Yacc and Bison. While both tools demonstrated efficient parsing for simple grammars, differences emerged in more complex scenarios. Bison, being an extended version of Yacc, showed improved performance in handling ambiguous grammars and exhibited better error recovery mechanisms.

7. Discussion:

The performance differences observed may be attributed to the underlying algorithms employed by each parser generator. Bison's enhancements over Yacc, including better support for ambiguous grammars and advanced error recovery, contribute to its superior performance in certain scenarios. However, it's essential to note that the choice between Yacc and Bison may also depend on factors such as ease of use, documentation, and community support.

8. Conclusion:

This case study provides valuable insights into the performance of parser generators, specifically Yacc and Bison, in the context of compiler construction. The results emphasize the importance of considering the specific requirements of a project when choosing a parser generator. Future work could extend this analysis to include additional parser generators and explore their performance under different optimization settings.

Result and Discussion:

Learning Outcomes: The student should have the ability to

LO1: Identify type of grammar G.
LO2: Define First () and Follow () sets.
LO3: ***Find***First () and Follow () sets for given grammar G.
LO4: ***Apply*** First () and Follow () sets for designing Top Down and Bottom up Parsers

Course Outcomes: Upon completion of the course students will be able to analyze the analysis and synthesis phase of compiler for writhing application programs and construct different parsers for given context free grammars.

Conclusion:

For Faculty Use

Correction Parameters	**Formative Assessment [40%]**	**Timely completion of Practical [40%]**	**Attendance / Learning Attitude [20%]**	
Marks Obtained				