Home Assignment #2: Advanced Syntax Analysis in Compiler Design

Part 1: Theoretical Questions

1. Define syntax analysis in compiler design.

Explain how syntax analysis integrates with lexical analysis and semantic analysis in the compilation process. Discuss the importance of syntax analysis in ensuring correct program structure.

2. Compare and contrast LL(k) and LR(k) parsing techniques.

Include their parsing strategies, advantages, disadvantages, and typical use cases. Provide examples illustrating their parsing directions and lookahead mechanisms.

3. Explain the concept of an ambiguous grammar.

Provide an example of an ambiguous grammar and describe how ambiguity can affect parser construction. Suggest at least two techniques to resolve ambiguity in grammars.

4. Describe the role of parsing tables in table-driven parsers.

Explain how parsing tables are constructed for LR(1) parsers and their significance in facilitating deterministic parsing.

5. Discuss the concept of operator precedence and associativity in parsing.

How do parser algorithms handle operator precedence and associativity? Illustrate with an example involving arithmetic expressions.

6. Explain the differences between SLR, CLR(1), and LALR(1) parsers.

Compare their lookahead capacities, table sizes, and typical applications.

Part 2: Practical Tasks – Grammar Design and Parser Implementation

Choose either A or B

A. Grammar Construction and Ambiguity Resolution

1. Design a context-free grammar for the following language constructs, ensuring it is unambiguous:

  - Variable declarations with types (e.g., int a;, float b;)
  - Function definitions with parameters and return types
  - Nested block statements

2. The grammar you designed might be ambiguous. Identify any ambiguity present and modify your grammar to eliminate it, ensuring deterministic parsing.

3. Extend your grammar to include switch-case statements and demonstrate how your parser would handle nested or complex cases.

B. Implementation of a Shift-Reduce Parser with Lookahead

1. Implement an LR(1) shift-reduce parser in your chosen programming language for a small language subset that includes:

- Arithmetic expressions with operator precedence (+, -, , /)
- Variable assignments
- Parenthesized expressions

2. Your parser should:

- Read an input string representing a simple program snippet.
- Use parsing tables to parse the input.
- Report syntax errors with informative messages.
- Generate a parse tree or an abstract syntax tree (AST).

3. Demonstrate your parser with at least three example inputs, including valid and invalid programs, and show the parse result or error messages.

Note:

- You may use existing parser generator tools (e.g., Bison, Yacc, ANTLR) for parts of the implementation, but ensure you understand and explain key steps in your submitted report.

- Focus on demonstrating your understanding of advanced parsing concepts and their practical application in compiler design.

Submission Guidelines

- Submit your written answers to Part 1 in a PDF document – up to 1 pager only.

- Assignment can be done individually, as couples or gorups of 3 max.

- Provide your parser implementation code with proper comments – we'll schedule time to review.

- Include sample input strings and parsing outputs.

- Provide your parsing table diagrams.

- Reflective answers should be concise, well-structured, and demonstrate understanding.