CPSC-354 Report

Gabriel Giancarlo
Chapman University

October 13, 2025

Abstract

This report documents my work on parsing theory and context-free grammars. I learned how to construct derivation trees, understand operator precedence, and analyze grammar ambiguity. The exercises demonstrate the relationship between concrete syntax (strings) and abstract syntax (trees) in programming languages, providing insight into how compilers process source code.

Contents

1 Introduction

Parsing is the process of converting concrete syntax (strings) into abstract syntax (trees). This transformation is fundamental to programming language implementation, as it bridges the gap between human-readable source code and machine-processable representations.

Context-free grammars provide a mathematical framework for describing the syntax of programming languages. They consist of:

- **Terminals:** The actual symbols that appear in strings (operators, keywords, identifiers)
- **Nonterminals:** Abstract categories that represent syntactic constructs
- **Production rules:** Rules that define how nonterminals can be expanded

The key challenge in parsing is dealing with ambiguity - when a single string can be parsed in multiple ways. This is resolved through precedence and associativity rules that guide the parser toward the intended interpretation.

2 Week by Week

2.1 Week 1: Intro to Parsing and Context-Free Grammars

Using the context-free grammar:

$$\text{Exp} \rightarrow \text{Exp } \text{'+'} \text{ Exp1} \tag{1}$$
$$\text{Exp1} \rightarrow \text{Exp1 } \text{'*'} \text{ Exp2} \tag{2}$$
$$\text{Exp2} \rightarrow \text{Integer} \tag{3}$$
$$\text{Exp2} \rightarrow \text{'('} \text{ Exp } \text{')'} \tag{4}$$
$$\text{Exp} \rightarrow \text{Exp1} \tag{5}$$
$$\text{Exp1} \rightarrow \text{Exp2} \tag{6}$$

2.1.1 Problem 1: Derivation Trees

Write out the derivation trees (also called parse trees or concrete syntax trees) for the following strings:

[label=(a)]

1. $2 + 1$
2. $1 + 2 * 3$
3. $1 + (2 * 3)$
4. $(1 + 2) * 3$
5. $1 + 2 * 3 + 4 * 5 + 6$

2.1.2 Problem 2: Unparsable Strings

Why do the following strings not have parse trees (given the context-free grammar above)?

[label=(b)]

1. $2 - 1$
2. $1.0 + 2$
3. $6/3$
4. 8 mod 6

2.1.3 Problem 3: Parse Tree Uniqueness

With the simplified grammar without precedence levels:

$$\text{Exp} \rightarrow \text{Exp } \text{'+'} \text{ Exp} \tag{7}$$
$$\text{Exp} \rightarrow \text{Exp } \text{'*'} \text{ Exp} \tag{8}$$
$$\text{Exp} \rightarrow \text{Integer} \tag{9}$$

How many parse trees can you find for the following expressions?

[label=(c)]

1. $1 + 2 + 3$

2. $1 * 2 * 3 * 4$

Answer the question above using instead the grammar:

$$\text{Exp} \rightarrow \text{Exp } \text{'+'} \text{ Exp1} \tag{10}$$
$$\text{Exp} \rightarrow \text{Exp1} \tag{11}$$
$$\text{Exp1} \rightarrow \text{Exp1 } \text{'*'} \text{ Exp2} \tag{12}$$
$$\text{Exp1} \rightarrow \text{Exp2} \tag{13}$$
$$\text{Exp2} \rightarrow \text{Integer} \tag{14}$$

3 Essay

Working through these parsing exercises was both challenging and enlightening. The most fascinating aspect was seeing how grammar design directly affects the meaning of expressions.

The key insight was understanding how precedence and associativity are encoded in the grammar structure. The grammar with precedence levels uses different nonterminals (Exp, Exp1, Exp2) to enforce a hierarchy where multiplication has higher precedence than addition, and both operators are left-associative.

This was particularly evident in the comparison between the ambiguous grammar and the precedence grammar. The ambiguous grammar allows multiple parse trees for expressions like $1 + 2 + 3$, leading to different interpretations. The precedence grammar eliminates this ambiguity by forcing a specific parsing order.

The derivation tree exercises were especially valuable for understanding the relationship between concrete and abstract syntax. Each tree shows exactly how the parser would interpret the expression, making the precedence rules explicit and visual.

These exercises highlighted several important concepts:

- **Grammar design:** How the structure of production rules affects parsing
- **Precedence:** How to encode operator precedence in grammar rules
- **Associativity:** How to enforce left or right associativity
- **Ambiguity:** The problems that arise when multiple parse trees are possible

Understanding parsing is crucial for programming language design and implementation. These exercises have improved my ability to think about syntax design and to understand how compilers process source code.

4 Evidence of Participation

I completed all the parsing exercises, including:

- **Derivation Trees:** Constructed parse trees for all five expressions, showing how precedence rules affect parsing
- **Unparsable Strings:** Identified why certain strings cannot be parsed with the given grammar
- **Parse Tree Uniqueness:** Analyzed ambiguity in simplified grammars and how precedence grammars eliminate it
- **Mathematical Analysis:** Understood the relationship between grammar structure and parsing behavior

Each exercise was completed with:

- Careful construction of derivation trees
- Step-by-step analysis of parsing processes
- Understanding of precedence and associativity rules
- Recognition of how grammar design affects language semantics

5 Conclusion

These parsing exercises provided valuable insight into the mathematical foundations of programming language syntax. The key lessons learned include:

- Context-free grammars provide a powerful framework for describing syntax
- Grammar design directly affects the meaning of expressions
- Precedence and associativity can be encoded in grammar structure
- Ambiguity is a fundamental problem in parsing that must be resolved
- The relationship between concrete and abstract syntax is crucial for language implementation

Understanding parsing theory is essential for programming language design and compiler construction. These exercises have improved my ability to think about syntax design and to understand how programming languages are processed by compilers.

References

[BNF] John Backus, The Syntax and Semantics of the Proposed International Algebraic Language, 1959.

[Parsing] Alfred Aho, Monica Lam, Ravi Sethi, and Jeffrey Ullman, Compilers: Principles, Techniques, and Tools, Addison-Wesley, 2006.