Syntax Analysis

Zakia Zinat Choudhury
Lecturer
Department of Computer Science & Engineering
University of Rajshahi

CONTENTS

Left Factoring

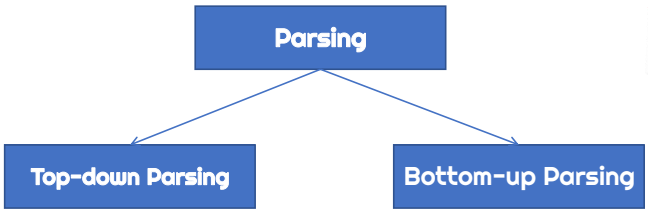
If more than one grammar production rules has a common prefix string, then the top-down parser cannot make a choice as to which of the production it should take to parse the string in hand.

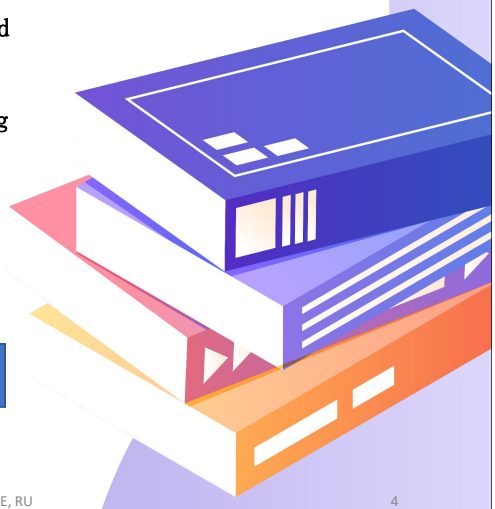
Then it cannot determine which production to follow to parse the string as both productions are starting from the same terminal (or non-terminal). To remove this confusion, a technique is used that is called **left factoring**.

Left factoring transforms the grammar to make it useful for top-down parsers. In this technique, **one production is made for each common prefixes and the rest of the derivation is added by new productions.**

Types of Parsing

Syntax analyzers follow production rules defined by means of context-free grammar. The way the production rules are implemented (derivation) divides parsing into two types : top-down parsing and bottom-up parsing.





FIRST and FOLLOW

An important part of parser table construction is to create first and follow sets. These sets can provide the actual position of any terminal in the derivation.

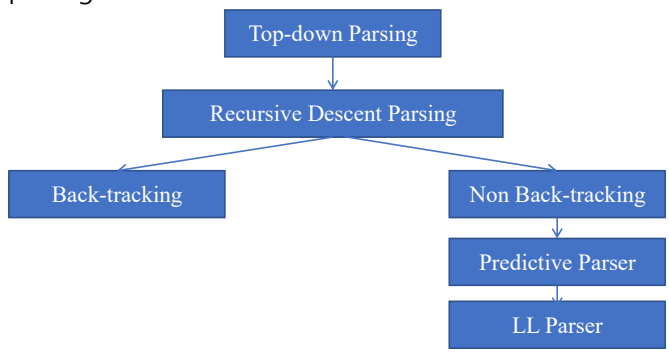
The construction of both top-down and bottom-up parsers is helped by two functions, **FIRST and FOLLOW**, associated with a grammar G. During top-down parsing, FIRST and FOLLOW allow us to choose which production to apply, based on the next input symbol.

Define FIRST(α), where α is any string of grammar symbols, to be the set of terminals that begin strings derived from α. If $\alpha \rightarrow \varepsilon$, then Ɛ is also in FIRST (α).

Define FOLLOW(A), for nonterminal A, to be the set of terminals **a** that can appear immediately to the right of A in some sentential form; that is, the set of terminals a such that there exists a derivation of the form $S \rightarrow \alpha A a \beta$, for some α and β.

Top-down Parsing

When the parser starts constructing the parse tree from the start symbol and then tries to transform the start symbol to the input, it is called top-down parsing.
Equivalently, top-down parsing can be viewed as finding a leftmost derivation for an input string.
The types of top-down parsing are discussed below:



6

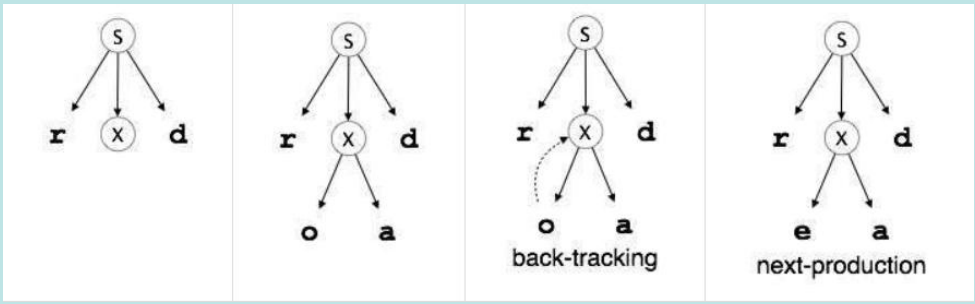
Recursive Descent Parsing

Recursive descent is a top-down parsing technique that constructs the parse tree from the top and the input is read from left to right. It uses procedures for every terminal and non-terminal entity. This parsing technique recursively parses the input to make a parse tree, which may or may not require back-tracking. But the grammar associated with it (if not left factored) cannot avoid back-tracking. A form of recursive-descent parsing that does not require any back-tracking is known as predictive parsing.

This parsing technique is regarded recursive as it uses context-free grammar which is recursive in nature.

Back-tracking

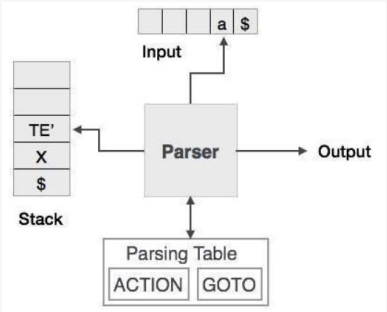
Top- down parsers start from the root node (start symbol) and match the input string against the production rules to replace them (if matched).



Predictive Parser

Predictive parser is a recursive descent parser, which has the capability to predict which production is to be used to replace the input string. The predictive parser does not suffer from backtracking.

To accomplish its tasks, the predictive parser uses a look-ahead pointer, which points to the next input symbols. To make the parser back-tracking free, the predictive parser puts some constraints on the grammar and accepts only a class of grammar known as LL(k) grammar.



Predictive parsing uses a stack and a parsing table to parse the input and generate a parse tree. Both the stack and the input contains an end symbol $ to denote that the stack is empty and the input is consumed. The parser refers to the parsing table to take any decision on the input and stack element combination.

LL Parser

An LL Parser accepts LL grammar. LL grammar is a subset of context-free grammar but with some restrictions to get the simplified version, in order to achieve easy implementation. LL grammar can be implemented by means of both algorithms namely, recursive-descent or table-driven.

LL parser is denoted as LL(k). The first L in LL(k) is parsing the input from **left to right**, the second L in LL(k) stands for **left-most derivation** and k itself represents **the number of look aheads**. Generally k = 1, so LL(k) may also be written as LL(1).

Predictive Parsering Table

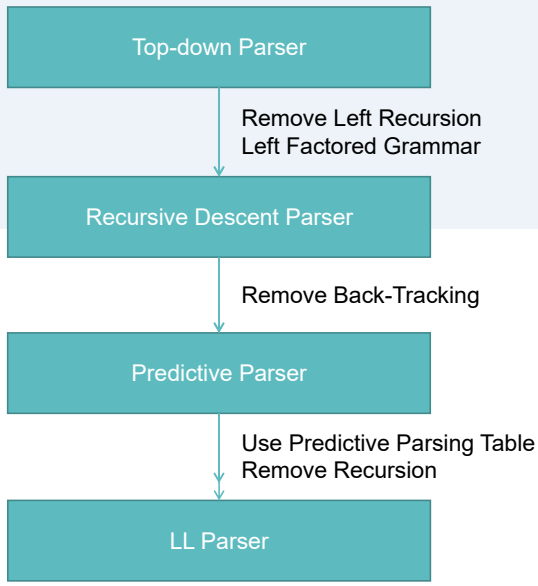
Algorithm of Construction of a predictive parsing table

INPUT: Grammar G.

OUTPUT: Parsing table M.

METHOD: For each production $A \to \alpha$ of the grammar, do the following:

1. For each terminal a in FIRST(A), add $A \to \alpha$ to $M[A, a]$.
2. If ϵ is in FIRST(α), then for each terminal b in FOLLOW(A), add $A \to \alpha$ to $M[A, b]$. If ϵ is in FIRST(α) and \$ is in FOLLOW(A), add $A \to \alpha$ to $M[A, \$]$ as well.




Thank you