Unit-3

Syntax Analyzer

Menka Patel
Department of Computer Engineering,
UVPCE

Overview of Syntax Analyzer

- Also called Parser
- Generate Parse Tree

Grammar

A grammar is mainly consists of 4 tuples

G: (S, P, N, T)

Where,

S = Start symbol

N = Finite set of Non-terminals

T = Finite set of terminals

P = Finite set of production

$S \in N$, $N \cap T = \emptyset$

Types of Grammar

1. Type-0 (Unrestricted Grammar)
2. Type-1 (Context-Sensitive Grammar)
3. Type-2 (Context-Free Grammar)
4. Type-3 (Regular Grammar)

Type-0

- Unrestricted Grammar
- Most General Grammar
- Phrase Structure
- This grammar class can generate recursively enumerable language
- Turing machine

Rule: $u \rightarrow v$

where, $u \in v^* N v^*$

$v \in v^*$, $v = N \cup T$

Type-1

- Context Sensitive Grammar
- This grammar class can generate context sensitive language
- Linear bounded automata

Rule: $u \rightarrow v$

where, $u \in v^* \, N \, v^*$

$v \in v^+$, $v = N \cup T$

- Restriction is $|u| \leq |v|$
- Length increasing grammar

Type-2

- Context Free Grammar
- This grammar class can generate context free language
- Push down automata

Rule: $u \rightarrow v$

where, $u \in N$

$v \in v^*$, $v = N \cup T$

- $E \rightarrow E + E \mid E * E \mid (E) \mid id$

Type-3

- Regular Grammar
  1) Right linear grammar
  2) Left linear grammar
- Finite automata

Rule: $A \rightarrow aB$

$B \rightarrow b$

Where, $A, B \in N$

$a \in T$

$b \in T \cup \{\epsilon\}$

Type-3

1) Right linear grammar: If all production of a CFG are of the form

   $A \rightarrow aB,\ B \rightarrow b$

   where, $A,B \in N,\ a \in T,\ b \in T \cup \{\epsilon\}$

   grammar is right linear grammar.

2) Left linear grammar: If all production of a CFG are of the form $A \rightarrow Ba,\ B \rightarrow b$

   where, $A,B \in N,\ a \in T,\ b \in T \cup \{\epsilon\}$

   grammar is left linear grammar.

Example

1. Write the left linear and right linear grammar for the regular expression RE = 0 (10)*
2. Identify the type of following grammar:

$A \rightarrow aABc \mid abC$

$CB \rightarrow BC$

$bB \rightarrow bb$

$bC \rightarrow bc$

$cC \rightarrow cc$

Derivation Tree

- Also known as parse tree

Example:

1) What type of string generated by given grammar?

$$S \rightarrow SS \mid (S) \mid \epsilon$$

Check for the string ((())) accept by grammar or not?

Derivation Tree

- Left most derivation
- Right most derivation

Example:

2) Generate left most derivation and right most derivation for the string “id + id + id” using following grammar:

$$E \rightarrow E + E \mid E * E \mid id$$

Restriction on CFG

- Let L be a non-empty context free language, then it can be generated by a context free grammar G with the following properties:

1. Eliminate useless symbol
2. Eliminate ϵ production
3. Eliminate unit production
4. Eliminate cycles

Restriction on CFG

1. Eliminate useless symbol
   - Identify and eliminate non-generating symbol
   - Identify and eliminate unreachable symbol

For Example:

$S \rightarrow AB \mid a$

$A \rightarrow b$

Restriction on CFG

Example:

S → aB | bX

A → BAd | bSX | a

B → aSB | bBX

X → SBD | aBx | ad

Restriction on CFG

2. Eliminate unit production
   - NT → one NT symbol
   - Unit production increases the cost of derivation

Algorithm:

While (there exists a unit production A → B)

{

Select a unit production A → B, such that there exist a production B → α, where α is a terminal.

For (every non-unit production, B → α)

Add production A → α to the grammar

Eliminate A → B from the grammar

}

Restriction on CFG

2. Eliminate unit production

Example:

Consider the CFG and eliminate unit production from given grammar:

$S \rightarrow AB$

$A \rightarrow a$

$B \rightarrow C \mid b$

$C \rightarrow D$

$D \rightarrow E$

$E \rightarrow a$

Restriction on CFG

2. Eliminate unit production

Example:

Consider the CFG and eliminate unit production from given grammar:

$$S \rightarrow A \mid bb$$

$$A \rightarrow B \mid b$$

$$B \rightarrow S \mid a$$

Restriction on CFG

3. Eliminate ϵ production and nullable non terminal
    - Eliminate the production form $A \rightarrow \epsilon$

For Example: Consider the following grammar and remove nullable NT:

$S \rightarrow aA$

$A \rightarrow b \mid \epsilon$

Restriction on CFG

3. Eliminate ϵ production and nullable non terminal
   - Eliminate the production form $A \rightarrow \epsilon$

For Example: Consider the following grammar and remove nullable NT:

$S \rightarrow ABAC$

$A \rightarrow aA \mid \epsilon$

$B \rightarrow bB \mid \epsilon$

$C \rightarrow c$

Restriction on CFG

4. Eliminate Cycle

- Consider an example as follows:

  $A \rightarrow B$

  $B \rightarrow C$

  $C \rightarrow A \mid c$

Restriction on CFG

Example: Eliminate ϵ and unit production from following grammar:

$S \rightarrow A \mid BAabB$

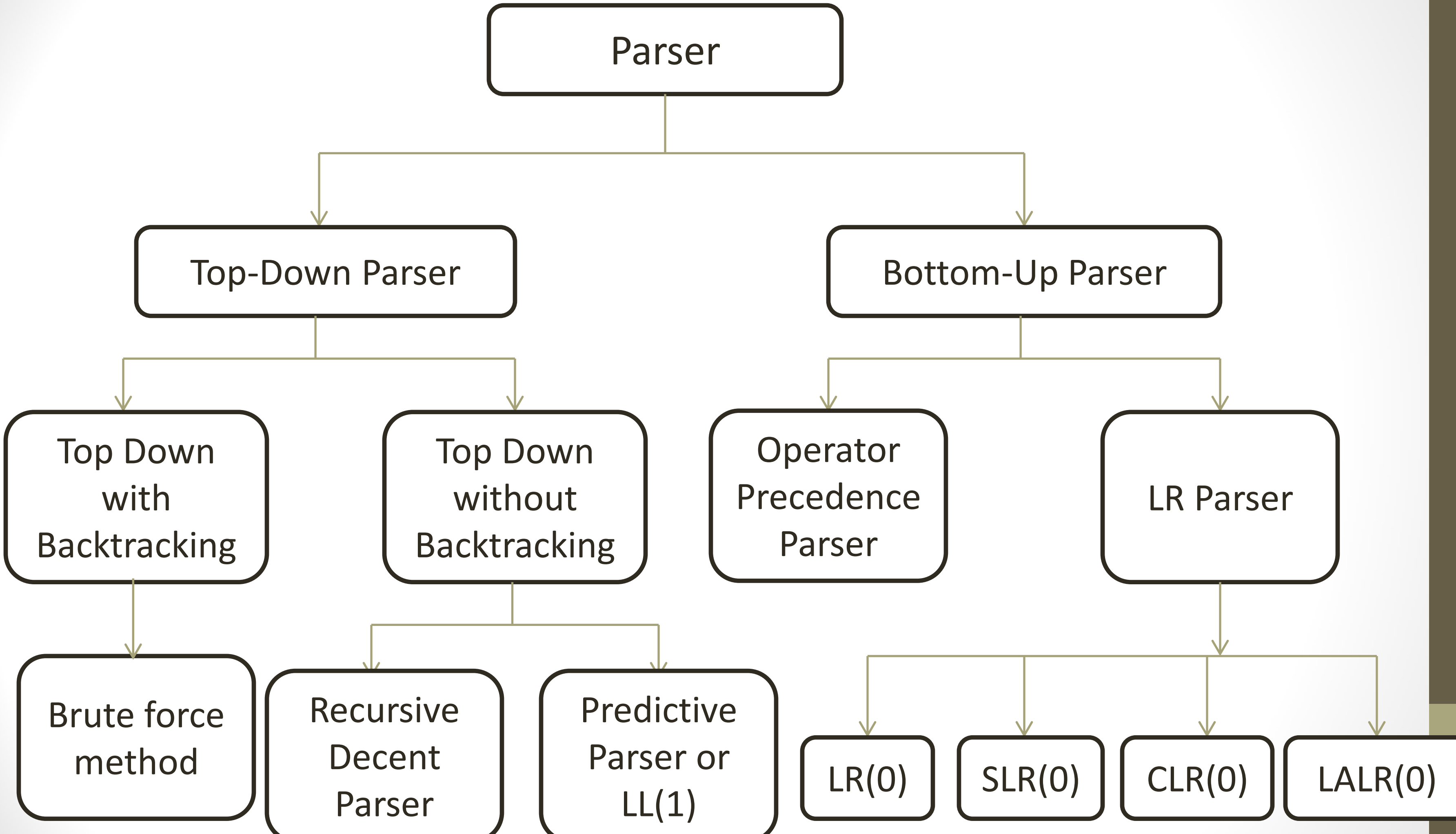
$A \rightarrow abA \mid a$

$B \rightarrow bB \mid \epsilon$

Parser

- A parser for any grammar is a program that takes string W as input and produces either a parse tree for W, if W is a valid sentence of grammar or an error message indicating that W is not a valid sentence of given grammar, as output.

- A goal of parsing is to determine the syntactic validity for a source string.


Parser
Top-Down Parser
Bottom-Up Parser
Top Down with Backtracking
Top Down without Backtracking
Operator Precedence Parser
LR Parser
Brute force method
Recursive Decent Parser
Predictive Parser or LL(1)
LR(0)
SLR(0)
CLR(0)
LALR(0)

Top Down Parsing

- It is attempts to find the left most derivation for the input string W, since string W is scanned by the parser left to right, one symbol/token at a time, and the left most derivation generates the leaves of the parse tree in left-to-right order, which matches the input scan order.
- Derivation of string

Example

S → aTUe

T → Tbc | b

U → d

Derive string ‘abbcde’ from given grammar using Top-Down Parser.

Issues of CFG

- Following are the issues of CFG for the programming languages for top-down parsing:
  1. Left Recursion and Indirect left recursion
  2. Left Factoring
  3. Ambiguity

Issues of CFG

1. Left Recursion
   - If the left most symbol on the right side is the same as the non-terminal on the left side.
   - For Example:

     $A \rightarrow A\alpha \mid \beta$

Example

E → E + T | T

T → T * F | F

F → (E) | I

I → a | b | c

Remove left recursion from given CFG.

Example

$S \rightarrow A$

$A \rightarrow Ad \mid Ae \mid aB \mid aC$

$B \rightarrow bBC \mid f$

$C \rightarrow g$

Remove left recursion from given CFG.