Context-Free languages

→ Next step from regular languages

→ Useful in parsing programming languages.

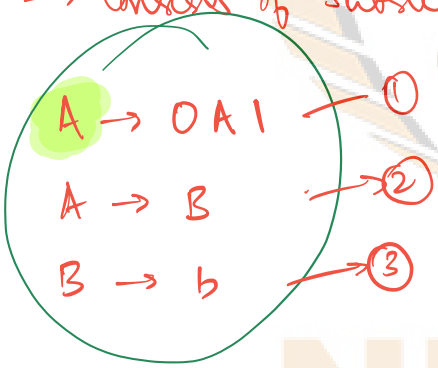
Regular : DFA / NFA Reg. Expressions

Context-Free : PDA Context-Free Grammar

Push Down Automaton

Context-Free Grammars

→ Consists of Substitution rules or productions



Variables : Capital letters
A, B

Terminals : Small letters, numbers, other symbols
0, 1, b.

Example derivations : A → B → b

A → 0A1 → 0B1 → 0b1

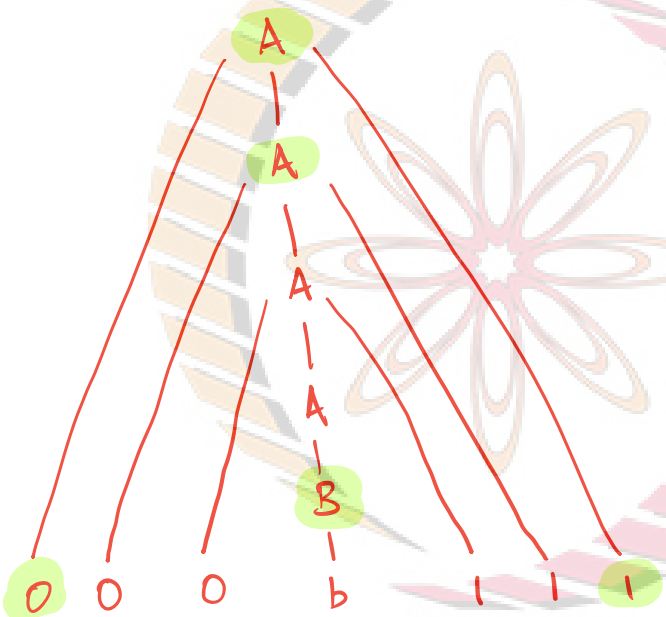
A → 0A1 → 00A11 → 00B11 → 00b11

A → 0A1 → 00A11 → 000A111 → 000B111 → 0000b111

Strings generated : $b, 0b1, 00b11, 000b111, \ldots$

Start variable : On the LHS of the first rule.

→ Start from start variable, repeatedly replace the variables in the resulting strings using the rules. Repeat till no variables remain.



PARSE TREE

How 000b111 is derived from A.

If we write a program, the compiler needs to check for properly nested parentheses.

$$L = \{ w \in \{[,]\}^* \mid w \text{ is properly nested} \}$$

Examples: [[[ ]]], [[ ]][ ], [[[ ]][ ]]

1 1 2 3 3 2 4 4 1

$$\Sigma = \{[,]\}$$

$A = [^*]^*$ $\longrightarrow$ A is regular.

$L \cap A = \{ [^n]^n \mid n \geq 0 \}$ is not regular

Hence L is not regular. We will see that L is context-free.

$B = \{0^n 1^n \mid n \geq 0\}$ $\rightarrow$ Similar to $L \cap A$

$S \rightarrow 0S1$
$S \rightarrow \varepsilon$

$\rightarrow$ Grammar generates B.

$S \rightarrow \varepsilon$,

$S \rightarrow 0S1 \rightarrow 00S11 \rightarrow 0011$

Even natural languages like English follow such a grammar. We have words and we generate sentences in various ways from NOUN, VERB, PREPOSITION, ADJECTIVE etc. (read example in the book, page 101).

Examples: (1) $S \rightarrow (A)$

$A \rightarrow \varepsilon \mid aA \mid ASA$

A can generate ε, or aA, or ASA

Variables: S, A

Terminals: a, (,)

$S \rightarrow (A) \rightarrow (aA) \rightarrow (aaA) \rightarrow (aaASA)$

$\rightarrow (aaSA) \rightarrow (aa(A)A)$

$\rightarrow (aaSA) \rightarrow (aaA)A)$

$\rightarrow (aa())$

(2) $S \rightarrow \varepsilon \mid aSb$

Variables: S

Terminals: a, b

Denotes $\{a^n b^n \mid n \geq 0\}$

$S \rightarrow aSb \rightarrow aaSbb \rightarrow aaaSbbb \rightarrow aaabbb$

$S \rightarrow aSb \rightarrow ab$

Def 2.2: A context-free grammar (CFG) is a 4-tuple (V, Σ, R, S), where

(1) V is a finite set, called variables

(2) Σ is a finite set, disjoint from V, called terminals.

(3) R is a set of rules, each of the form

Variable $\rightarrow$ String of variables and terminals

$A \rightarrow a$, $A \rightarrow aSbAa$

(4) $S \in V$ is the start variable.

A context-free language (CFL) is a language that can be generated from a context-free grammar (CFG).

Yields: $\Rightarrow$ One step by application of a rule

$uAw \Rightarrow uvw$ (using $A \rightarrow v$)

Derives: u derives v, $u \overset{*}{\Rightarrow} v$ if $u = v$ or if there is a sequence $u_1, u_2, \ldots u_k$ such that

$$u = u_1 \Rightarrow u_2 \Rightarrow u_3 \Rightarrow \ldots \Rightarrow u_k = v$$

The language of the grammar G, denoted $L(G)$, is given by

$$L(G) = \{ w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w \}$$

Why context-free? $S \to aSb$

The rules and derivations are not based on the content of the variable. Rules are not of the type

$$aVb \to axb$$
$$bVa \to byb$$

$V \to x$

Rules do not depend on the content. There are context-sensitive languages, though we won't see them in this course.

Example 2.3: $G = \{\{S\}, \{a, b\}, R, S\}$

[ ] [ ]

$R: S \to aSb \mid SS \mid \varepsilon$

If a = [and b =], this yields the set of all properly nested brackets.

$S \to aSb \to aaSbb \to aabb$

$S \to SS \to aSbS \to aaSbbS \to aabbS \to aabbaSb \to aabbab$

[[ ]] [ ]

Designing CFG's

1. If A and B are CFG's, we can construct a CFG that generates the union language by having a rule

New start var. $S \to S_A \mid S_B$

CFL's are closed under union

along with the rules of A and B.

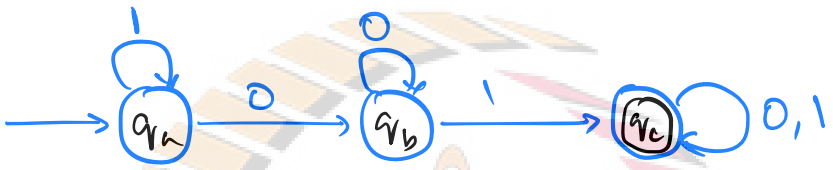
2. All regular languages can be expressed using CFG's.

→ One variable for each state of the CFG.

→ V_0 corresponds to q_0 (start state).

→ $\delta(q_i, a) = q_j$ corresponds to $V_i \to a V_j$

→ For $q_j \in F$, we add $V_j \to \varepsilon$.



Three var: A, B, C.

$$A \to 0B \mid 1A$$
$$B \to 0B \mid 1C$$
$$C \to 0C \mid 1C \mid \varepsilon$$

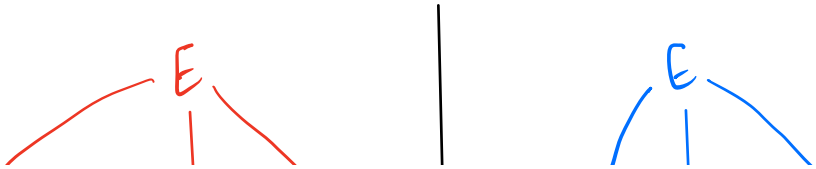
Exercise: Verify that the grammar generates the same language as the DFA.

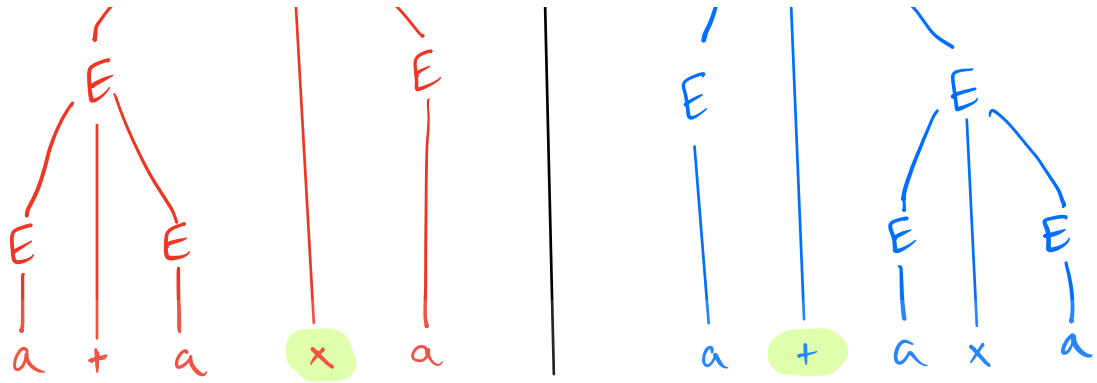
Ambiguity

Var: E

Term: a, +, ×, (,)

$$G: E \to E+E \mid E \times E \mid (E) \mid a$$





When the parser parses 'a+axa' it is not sure how this expression was derived. Does the '+' or 'x' have precedence?

In English: She called the man with an Iphone.

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T \times F \mid F$$
$$F \rightarrow (E) \mid a$$

This grammar generates the same language as above, but is not ambiguous.

Leftmost Derivation: The leftmost variable is expanded first.

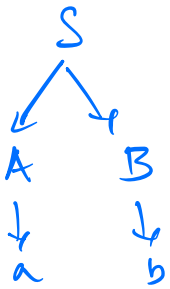
Example: $S \rightarrow AB$,

$A \rightarrow a$

$B \rightarrow b$

$S \rightarrow AB \rightarrow aB \rightarrow ab$

$S \rightarrow AB \rightarrow Ab \rightarrow ab$



Def 2.7: A string w is derived ambiguously in a CFG G if it has two or more distinct leftmost derivations. Grammar G is ambiguous if it generates some string ambiguously.