Chapter 6

Theory of Computation

Krishna Gaire (Data Engineer @ CGT)

Outlines

6.1 Introduction to finite automata
6.2 Introduction to context free language
6.3 Turing machine

6.4 Introduction of computer graphics
6.5 Two-dimensional transformation
6.6 Three-dimensional transformation

Lecture 2

- Grammar
- Context-Free Languages
- Chomsky Normal Form
- Greibach Normal Form (GNF)
- Backus-Naur Form (BNF)
- Pushdown Automata (PDA)
- Pumping lemma for context free language
- Properties of context free Language.

Grammar

we introduced two different, though equivalent, methods of describing languages: finite automata and regular expressions.
We showed that many languages can be described in this way but that some simple languages, such as {0"1"| n ≥ 0}, cannot.

In this chapter we present context-free grammars, a more powerful method of describing languages. Such grammars can describe certain features that have a recursive structure, which makes them useful in a variety of applications.

Grammar

If a grammar generates the same string in several different ways, we say that the string is derived *ambiguously* in that grammar. If a grammar generates some string ambiguously we say that the grammar is *ambiguous*.

For example, consider grammar G_5:

$$\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle \mid \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle \mid (\langle \text{EXPR} \rangle) \mid \texttt{a}$$

This grammar generates the string a+a×a ambiguously. The following figure shows the two different parse trees.

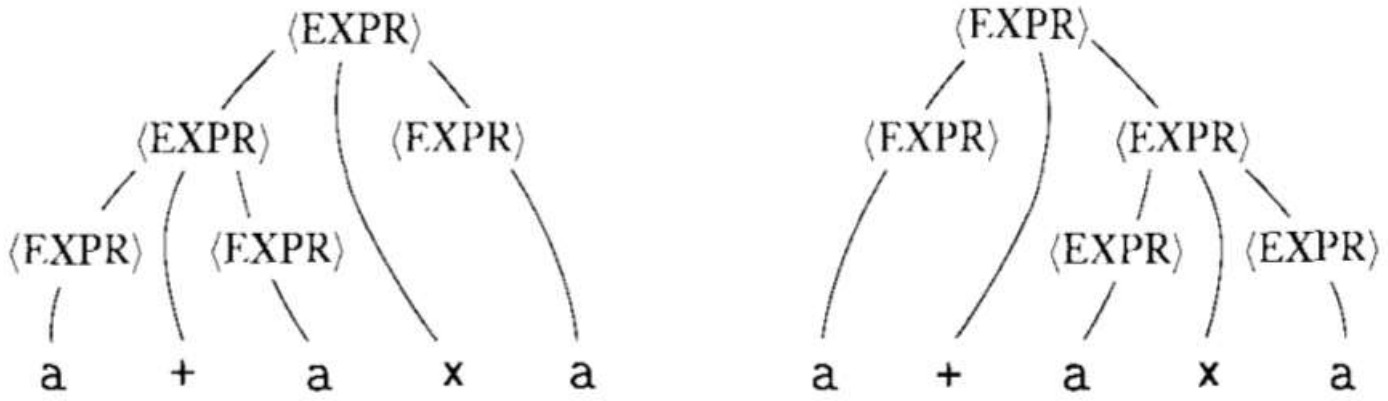


FIGURE 2.6
The two parse trees for the string a+a×a in grammar G_5

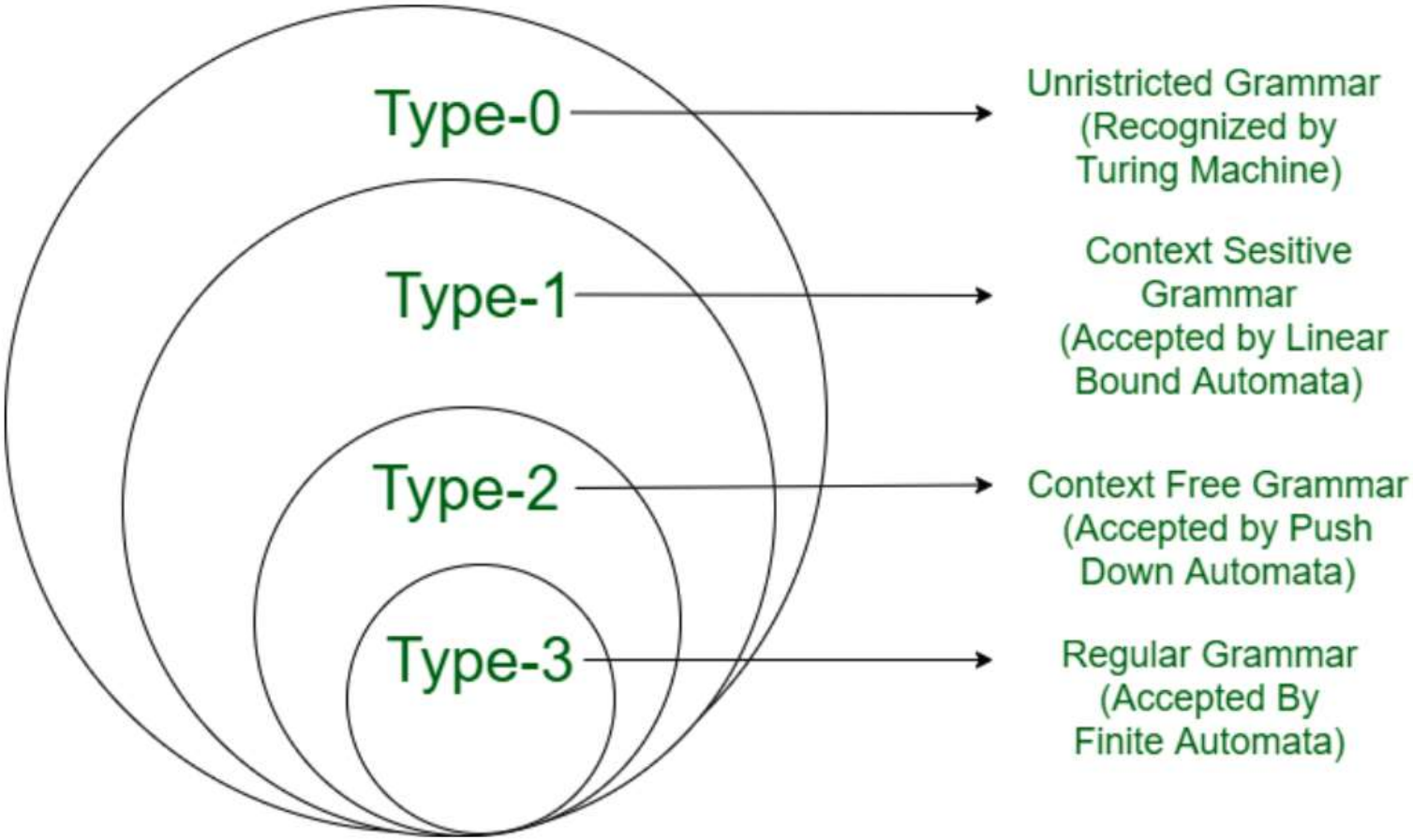
According to **Chomsky hierarchy**, grammar is divided into 4 types as follows:

Type 0 is known as unrestricted grammar.

Type 1 is known as context-sensitive grammar.

Type 2 is known as a context-free grammar.

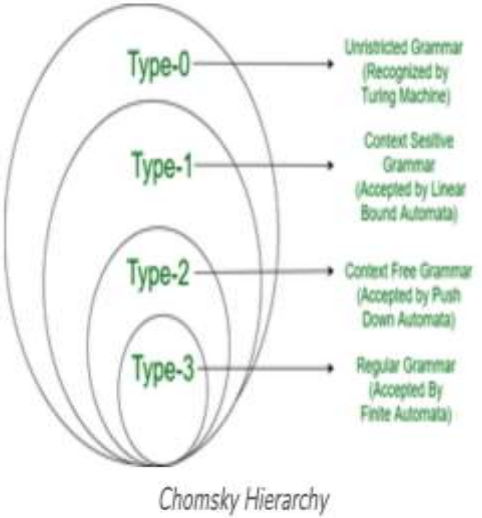
Type 3 Regular Grammar.



Grammar

0. Unrestricted Grammar :

These are the most general type of grammars and have no restrictions on their production rules. They can generate any language that can be recognized by a **Turing machine**, meaning they are as powerful as any computational system.



Chomsky Hierarchy

1. Context Sensitive Grammar :

These grammars have production rules of the form $\alpha\ A\ \beta \rightarrow \alpha\ \gamma\ \beta$ αAβ→αγβ, where A A is a non-terminal, and α α, β β, and γ γ are strings of terminals and/or non-terminals, with γ γ being non-empty.

2. Context Free Grammar :

Type-2 grammars generate context-free languages. The language generated by the grammar is recognized by a Push Down Automata In Type 2:

- First of all, it should be Type 1.
- The left-hand side of production can have only one variable and there is no restriction on β and $|\alpha| = 1$

3. Regular Grammar : Type-3 grammars generate regular languages. These languages are exactly all languages that can be accepted by a finite-state automaton. Type 3 is the most restricted form of grammar.
Type 3 should be in the given form only :

Context-Free Languages

A ***context-free grammar*** is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the ***variables***,
2. Σ is a finite set, disjoint from V, called the ***terminals***,
3. R is a finite set of ***rules***, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable.

We start with an example. Consider the following five (substitution) rules:

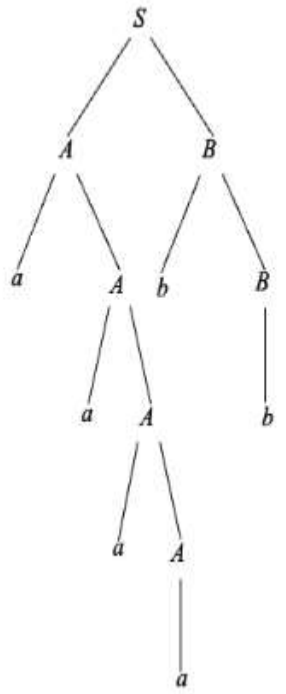
$$S \to AB$$
$$A \to a \quad A \to aA \quad B \to b \quad B \to bB$$

Here, *S*, *A*, and *B* are *variables*, *S* is the *start variable*, and *a* and *b* are *terminals*. We use these rules to derive strings consisting of terminals (i.e., elements of $\{a, b\}^*$), in the following manner:

1. Initialize the *current string* to be the string consisting of the start variable *S*.
2. Take any variable in the current string and take any rule that has this variable on the left-hand side. Then, in the current string, replace this variable by the right-hand side of the rule.
3. Repeat 2. until the current string only contains terminals.

For example, the string *aaaabb* can be derived in the following way:

$$\begin{aligned} S &\Rightarrow AB \\ &\Rightarrow aAB \\ &\Rightarrow aAbB \\ &\Rightarrow aaAbB \\ &\Rightarrow aaaAbB \\ &\Rightarrow aaaabB \\ &\Rightarrow aaaabb \end{aligned}$$



This derivation can also be represented using a ***parse tree,*** as in the figure

Chomsky Normal Form

CNF stands for Chomsky normal form. A CFG(context free grammar) is in CNF(Chomsky normal form) if all production rules satisfy one of the following conditions:

- Start symbol generating ε. For example, A → ε.
- A non-terminal generating two non-terminals. For example, S → AB.
- A non-terminal generating a terminal. For example, S → a.

$$G1 = \{S \rightarrow AB, S \rightarrow c, A \rightarrow a, B \rightarrow b\}$$
$$G2 = \{S \rightarrow aA, A \rightarrow a, B \rightarrow c\}$$

The production rules of Grammar G1 satisfy the rules specified for CNF, so the grammar G1 is in CNF. However, the production rule of Grammar G2 does not satisfy the rules specified for CNF as S → aZ contains terminal followed by non-terminal. So the grammar G2 is not in CNF.

Greibach Normal Form (GNF)

GNF stands for Greibach normal form. A CFG(context free grammar) is in GNF(Greibach normal form) if all the production rules satisfy one of the following conditions:

- A start symbol generating ε. For example, S → ε.
- A non-terminal generating a terminal. For example, A → a.
- A non-terminal generating a terminal whi terminals. For example, S → aASB.

G1 = {S → aAB | aB, A → aA| a, B → bB | b}
G2 = {S → aAB | aB, A → aA | ε, B → bB | ε}

For Example :

The production rules of Grammar G1 satisfy the rules specified for GNF, so the grammar G1 is in GNF. However, the production rule of Grammar G2 does not satisfy the rules specified for GNF as A → ε and B → ε contains ε(only start symbol can generate ε).

So the grammar G2 is not in GNF.

Backus-Naur Form (BNF)

BNF stands for **Backus Naur Form** notation. BNF may be a meta-language (a language that cannot describe another language) for primary languages.

Left side → definition

Shorthand notation for type 2 grammars.

- nonterminals denoted with $\langle\rangle$.
- $\rightarrow$ written as `::=`
- All rules with same nonterminal on left written together with the list of possible right sides separated by `|` .

Example: If we have production rules $A \rightarrow Aa$, $A \rightarrow a$, and $A \rightarrow AB$, we will write

```
<A> ::= <A>a | a | <A><B>
```

Push Down Automata (PDA)

Pushdown automata is a way to implement a CFG in the same way we design DFA for a regular grammar. A DFA can remember a finite amount of information, but a PDA can remember an infinite amount of information with an "**external stack memory**"

The PDA can be defined as a collection of **7 components**:

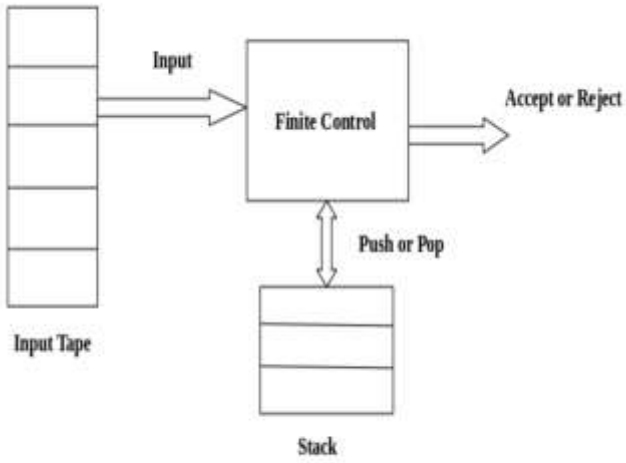


Fig: Pushdown Automata

Q: the finite set of states

∑: the input set

Γ: a stack symbol which can be pushed and popped from the stack

q0: the initial state

Z: a start symbol which is in Γ.

F: a set of final states

δ: mapping function which is used for moving from current state to next state.

Pumping lemma for context free language

Pumping Length (for CFL) is used to prove that a language is not context free

Let L be a context-free language. Then there exists an integer $p \geq 1$, called the pumping length, such that the following holds:

Every string s in L, with $|s| \geq p$, can be written as s = uvxyz, such that
1. $|vy| \geq 1$ (i.e., v and y are not both empty),
2. $|vxy| \leq p$, and
3. $uvixyiz \in L$, for all $i \geq 0$.

Pumping lemma for CFL : 5 Pieces
Pumping lemma for Regular Langauge : 3 Pieces

Properties of context free Language.

1. Union
2. Concentration
3. Transpose
4. Kleenstar
5. Intersection
6. Complement.

Key Points

Finite Automata: Understand the concept of finite automata (FA) and finite state machines. An FA is represented by a **5-tuple** (Q, Σ, δ, q0, F).

DFA: Deterministic Finite Automata - has one unique transition for each symbol and state. Represented by a **5-tuple** (Q, Σ, δ, q0, F), where $\delta: Q \times \Sigma \rightarrow Q$.

NDFA: Non-Deterministic Finite Automata - can have multiple transitions for the same symbol and state. Represented by a 5-tuple (Q, Σ, δ, q0, F), where $\delta: Q \times \Sigma \rightarrow 2^Q$.

Equivalence of DFA and NDFA: Both recognize the same class of languages (regular languages). Any NDFA can be converted to an equivalent DFA.

Minimization of Finite State Machines: Techniques to reduce the number of states in a DFA while preserving the language it recognizes.

Regular Expressions: Representations of regular languages using symbols and operators. E.g., a(b|c)*.

Equivalence of Regular Expressions and Finite Automata: Both describe the same set of languages. A regular expression can be converted to an FA and vice versa.

Pumping Lemma for Regular Languages: A property used to prove that certain languages are not regular. It states that for any regular language, there exists a length p such that any string longer than **p can be split into three parts,** and the middle part can be pumped (repeated) to produce new strings in the language.

Context-Free Grammar (CFG): A type of grammar where each production rule has a single non-terminal on the left-hand side. Represented by a **4-tuple** (V, Σ, R, S).

Derivative Trees: Visual representations of the derivations of strings in a language, showing how a string is generated by the grammar.

Bottom-Up Approach: Building the parse tree from the leaves to the root.

Top-Down Approach: Building the parse tree from the root to the leaves.

Leftmost Derivation: Replacing the leftmost non-terminal first during derivation.

Rightmost Derivation: Replacing the rightmost non-terminal first during derivation.

Language of a Grammar: The set of all strings that can be generated using the grammar.

Parse Tree Construction: Creating a tree that represents the structure of a string according to a grammar.

Ambiguous Grammar: A grammar that can generate the same string in multiple ways, resulting in different parse trees.

Chomsky Normal Form (CNF): A form of CFG where each production rule is either A → BC or A → a, making the grammar simpler and easier to parse.

Greibach Normal Form (GNF): A form of CFG where each production rule is $A \rightarrow a\alpha$, useful for certain parsing algorithms.

Backus-Naur Form (BNF): A notation used to express the grammar of a language in a formal way, often used in programming language specifications.

Pushdown Automata (PDA): A type of automaton that uses a stack to handle context-free languages. Represented by a **7-tuple** $(Q, \Sigma, \Gamma, \delta, q0, Z0, F)$, where Γ is the stack alphabet and $Z0$ is the initial stack symbol.

Pumping Lemma for Context-Free Languages: A property used to prove that certain languages are not context-free. It states that for any context-free language, there exists a length p such that any string longer than p can be **split into five parts**, and certain parts can be pumped to produce new strings in the language.

Closure Properties of Regular Languages: Regular languages are closed under operations like union, intersection, and complementation.

Closure Properties of Context-Free Languages: Context-free languages are closed under operations like union and concatenation but not under intersection and complementation.

Applications: Real-world applications of finite automata and context-free languages in areas like compiler design, formal verification, and natural language processing.

Which of the following is a not a part of 5-tuple finite automata?

a)Input alphabet

b)Transition function

c)Initial State

d)Output Alphabet

Answer: d

Explanation: A FA can be represented as FA= (Q, ∑, δ , q0, F) where Q=Finite Set of States, ∑=Finite Input Alphabet, δ =Transition Function, q0=Initial State, F=Final/Acceptance State).

Finite Automata

1. There are_tuples in finite state machine.

a)4

b)5

c)6

unlimited

Answer: b

Explanation: States, input symbols,initial state,accepting function.

Languages of a automata is

a)If it is accepted by automata

b)If it halts

c)If automata touch final state in its life time

d)All language are language of automata

Answer: a

Explanation: If a string accepted by automata it is called language of automata

Language of finite automata is.

a)Type 0

b)Type 1

c)Type 2

Type 3

Answer: d

Explanation: According to Chomsky classification.

A Language for which no DFA exist is a

a)Regular Language

b)Non-Regular Language

c)May be Regular

d)Cannot be said

Answer: b

Explanation: A language for which there is no existence of a deterministic finite automata is always Non Regular and methods like Pumping Lemma can be used to prove the same.

Production Rule: aAb->agb belongs to which of the following category?

a)Regular Language

b)Context free Language

c)Context Sensitive Language

d)Recursively Ennumerable Language

Answer: c **Explanation:** Context Sensitive Language or Type 1 or Linearly Bounded Non deterministic Language has the production rule where the production is context dependent i.e. aAb->agb.

Which of the following statement is false?

a)Context free language is the subset of context sensitive language

b)Regular language is the subset of context sensitive language

c)Recursively ennumerable language is the super set of regular language

d)Context sensitive language is a subset of context free language

Answer: d

Explanation: Every regular language can be produced by context free grammar and context free language can be produced by context sensitive grammar and so on.

The Grammar can be defined as: G=(V, ∑, p, S) In the given definition, what does S represents?

a)Accepting State

b)Starting Variable

c)Sensitive Grammar

Answer: b

Explanation: G=(V, ∑, p, S), here V=Finite set of variables, ∑= set of terminals, p= finite productions, S= Starting Variable.

The most suitable data structure used to represent the derivations in compiler:

a)Queue

b)Linked List

c)Tree

Hash Tables

Context free grammar is called Type 2 grammar because of hierarchy.

a)Greibach

b)Backus

c)Chomsky

d)None of the mentioned

Explanation: Chomsky hierarchy decide four type of language :Type 3- Regular Language, Type 2-Context free language, Type 1-Context Sensitive Language, Type 0- Unrestricted or Recursively Ennumerable language.

Thank You