Theory of Computing

CSE - 203

Introduction to Grammars

**Grammar**

A grammar **G** can be formally written as a 4-tuple (N, T, S, P) where −

* **N** or **V*N*** is a set of variables or non-terminal symbols.
* **T** or **∑** is a set of Terminal symbols.
* **S** is a special variable called the Start symbol, S ∈ N
* **P** is Production rules for Terminals and Non-terminals. A production rule has the form α → β, where α and β are strings on V*N* ∪ ∑ and least one symbol of α belongs to VN.

**Example**

Grammar G1 − ({S, A, B}, {a, b}, S, {S → AB, A → a, B → b})

Here,

* **S, A,** and **B** are Non-terminal symbols;
* **a** and **b** are Terminal symbols
* **S** is the Start symbol, S ∈ N
* Productions, **P : S → AB, A → a, B → b**

### Example

If there is a grammar

G: N = {S, A, B} T = {a, b} P = {S → AB, A → a, B → b}

Here **S** produces **AB**, and we can replace **A** by **a**, and **B** by **b**. Here, the only accepted string is **ab**, i.e.,

L(G) = {ab}

## Construction of a Grammar Generating a Language

We’ll consider some languages and convert it into a grammar G which produces those languages.

### Example

Problem − Suppose, L (G) = {am bn | m ≥ 0 and n > 0}. We have to find out the grammar **G** which produces **L(G)**.

***Solution***

Since L(G) = {am bn | m ≥ 0 and n > 0}

the set of strings accepted can be rewritten as −

L(G) = {b, ab,bb, aab, abb, …….}

Here, the start symbol has to take at least one ‘b’ preceded by any number of ‘a’ including null.

To accept the string set {b, ab, bb, aab, abb, …….}, we have taken the productions −

S → aS, S → B, B → b and B → bB

S → B → b (Accepted)

S → B → bB → bb (Accepted)

S → aS → aB → ab (Accepted)

S → aS → aaS → aaB → aab (Accepted)

S → aS → aB → abB → abb (Accepted)

Thus, we can prove every single string in L(G) is accepted by the language generated by the production set.

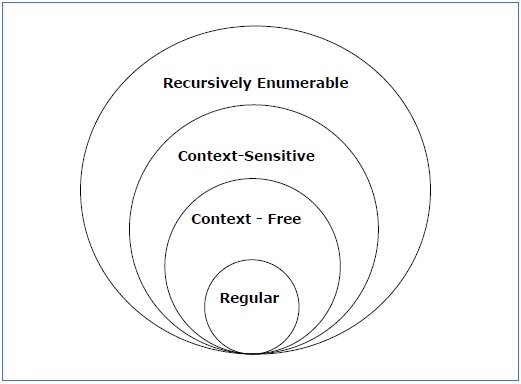
Hence the grammar −

G: ({S, A, B}, {a, b}, S, {S → aS | B, B → b | bB})

According to Noam Chomosky, there are four types of grammars − Type 0, Type 1, Type 2, and Type 3. The following table shows how they differ from each other −

|  |  |  |  |
| --- | --- | --- | --- |
| **Grammar Type** | **Grammar Accepted** | **Language Accepted** | **Automaton** |
| Type 0 | Unrestricted grammar | Recursively enumerable language | Turing Machine |
| Type 1 | Context-sensitive grammar | Context-sensitive language | Linear-bounded automaton |
| Type 2 | Context-free grammar | Context-free language | Pushdown automaton |
| Type 3 | Regular grammar | Regular language | Finite state automaton |

The following illustration shows the scope of each type of grammar −



## Type - 3 Grammar

Type-3 grammars generate regular languages. Type-3 grammars must have a single non-terminal on the left-hand side and a right-hand side consisting of a single terminal or single terminal followed by a single non-terminal.

The productions must be in the form **X → a or X → aY** where **X, Y ∈ N** (Non terminal) and **a ∈ T** (Terminal)

The rule **S → ε** is allowed if **S** does not appear on the right side of any rule.

### Example

X → ε

X → a | aY

Y → b

## Type - 2 Grammar

Type-2 grammars generate context-free languages.

The productions must be in the form **A → γ** where **A ∈ N** (Non terminal) and **γ ∈ (T ∪ N)\*** (String of terminals and non-terminals).

These languages generated by these grammars are be recognized by a non-deterministic pushdown automaton.

### Example

S → X a

X → a

X → aX

X → abc

X → ε

## Type - 1 Grammar

Type-1 grammars generate context-sensitive languages. The productions must be in the form

**α A β → α γ β**

where **A ∈ N** (Non-terminal) and **α, β, γ ∈ (T ∪ N)\*** (Strings of terminals and non-terminals)

The strings **α** and **β** may be empty, but **γ** must be non-empty.

The rule **S → ε** is allowed if S does not appear on the right side of any rule. The languages generated by these grammars are recognized by a linear bounded automaton.

### Example

AB → AbBc

A → bcA

B → b

## Type - 0 Grammar

Type-0 grammars generate recursively enumerable languages. The productions have no restrictions. They are any phase structure grammar including all formal grammars.

They generate the languages that are recognized by a Turing machine.

The productions can be in the form of **α → β** where **α** is a string of terminals and nonterminals with at least one non-terminal and **α** cannot be null. **β** is a string of terminals and non-terminals.

### Example

S → ACaB

Bc → acB

CB → DB

aD → Db

**Pumping Lemma for Regular Grammars**

**Theorem**

Let L be a regular language. Then there exists a constant **‘c’** such that for every string **w** in **L** –

**|w| ≥ c**

We can break **w** into three strings, **w = xyz**, such that −

* |y| > 0
* |xy| ≤ c
* For all k ≥ 0, the string xykz is also in L.

**Applications of Pumping Lemma**

Pumping Lemma is to be applied to show that certain languages are not regular. It should never be used to show a language is regular.

* If **L** is regular, it satisfies Pumping Lemma.
* If **L** does not satisfy Pumping Lemma, it is non-regular.

**Method to prove that a language L is not regular**

* At first, we have to assume that **L** is regular.
* So, the pumping lemma should hold for **L**.
* Use the pumping lemma to obtain a contradiction −
  + Select **w** such that **|w| ≥ c**
  + Select **y** such that **|y| ≥ 1**
  + Select **x** such that **|xy| ≤ c**
  + Assign the remaining string to **z.**
  + Select **k** such that the resulting string is not in **L.**

**Hence L is not regular.**

**Problem**

Prove that **L = {aibi | i ≥ 0}** is not regular.

***Solution*** −

* At first, we assume that **L** is regular and n is the number of states.
* Let w = *anbn*. Thus |w| = 2n ≥ n.
* By pumping lemma, let w = xyz, where |xy| ≤ n.
* Let x = ap, y = aq, and z = arbn, where p + q + r = n, p ≠ 0, q ≠ 0, r ≠ 0. Thus |y| ≠ 0.
* Let k = 2. Then xy2z = apa2qarbn.
* Number of as = (p + 2q + r) = (p + q + r) + q = n + q
* Hence, xy2z = an+q bn. Since q ≠ 0, xy2z is not of the form anbn.
* Thus, xy2z is not in L. Hence L is not regular.

Homework

1. Find a CFG that generates the following languages:
   1. L(G) = { an bm cm d2n | n ≥ 0, m > 0}.
   2. L(G) = { an bm | 0 ≤ n ≤ m ≤ 2n}.
2. Which language generates the grammar G given by the following production rules?
3. *S → aSa | aBa*

*B → bB | b*

1. *S → abScB |* ɛ

*B → bB | b*