Formal Languages and Grammar Types

Introduction

The set of all strings that can be derived from a grammar is said to be the language generated from that grammar. A language generated by a grammar **G** is a subset formally defined by:

$L(G) = \{W \mid W \in \sum^*, S \Rightarrow_G \mathbf{W}\}$

If **L(G1) = L(G2)**, the Grammar **G1** is equivalent to the Grammar **G2**.

The Chomsky Hierarchy

The Chomsky hierarchy categorizes formal grammars into four types: 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 relationship between different classes of languages is as follows:

Regular Languages $\subset$ Context-Free Languages $\subset$ Context-Sensitive Languages $\subset$ Recursively Enumerable Languages

Each grammar type has a specific scope, with Type 0 being the most powerful and Type 3 being the most restricted.

Type 3 Grammar (Regular 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:

- $X \rightarrow a$
- $X \rightarrow aY$

where:

- $X, Y \in N$ (Non-terminal)
- $a \in T$ (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
```

Regular languages are recognized by finite state automata.

Type 2 Grammar (Context-Free Grammar)

Type-2 grammars generate context-free languages.

What is Context-Free Grammar?

A context-free language is generated by a context-free grammar (CFG). In a CFG, production rules have the form: A → X, Where:

- A is a variable (non-terminal)
- X is any string of terminals or variables

The productions must be in the form:

- A → γ

where:

- A ∈ N (Non-terminal)
- γ ∈ (T ∪ N)* (String of terminals and non-terminals)

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

Properties of Context-Free Languages

The key characteristic of CFLs is that the replacement of A with X is independent of the surrounding context. This property gives CFLs their name—they are "free" of context constraints.

CFLs correspond to pushdown automata (PDAs) in the Chomsky hierarchy, which are more powerful than finite automata but less powerful than linear-bounded automata.

Example:

```
S → X a
X → a
X → aX
X → abc
X → ε
```

Type 1 Grammar (Context-Sensitive Grammar)

Type-1 grammars generate context-sensitive languages.

What is Context-Sensitive Grammars (CSGs)?

A context-sensitive grammar has production rules of the form: αAβ → αXβ, where:

- α, β are strings of terminals and/or variables (can be empty)
- A is a variable
- X is a non-empty string of terminals or variables

The productions must be in the form:

- αAβ → αγβ

where:

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

Properties of Context-Sensitive Languages

Listed below are some of the important properties of context-sensitive languages:

1. **Context Preservation** — The production process maintains the same context (α and β) on both sides, ensuring that the replacement of A with X only occurs within the defined context.
2. **Non-Contracting** — The grammar's property X cannot be empty, ensuring it doesn't reduce string length during derivation. However, the start variable S can generate an empty string if it's part of the language.
3. **Increased Expressive Power** — CSLs can describe patterns that CFLs cannot, such as matching multiple repeated substrings.

Context-sensitive languages extend the concept of CFLs by allowing production rules to depend on the context in which variables appear. This seemingly small change leads to a significant increase in

expressive power.

Example:

```
AB → AbBc
A → bcA
B → b
```

Example of a Language That is Context-Sensitive but Not Context-Free

A classic example of a language that is context-sensitive but not context-free is: L = {a^n b^n c^n | n ≥ 0}

The language is composed of strings with equal numbers of a's, b's, and c's, which cannot be generated by context-free grammars due to their inability to count and ensure equal numbers of three different symbols.

To illustrate the power of context-sensitive grammars, here is a CSG that generates the language L = {a^n b^n c^n | n ≥ 0}:

The production rules are as follows:

```
S → ε (to handle the case n = 0)
S → S'
S' → ABC
AB → BAB
BA → ACA
CA → CB
CB → AB
Bb → Bb
A → a
B → b
C → c
```

This grammar works through a series of transformations:

1. Rule 1 handles the empty string case (n = 0).
2. Rules 2-3 initialize the string with one occurrence of each variable (A, B, C).
3. Rules 4-7 allow for the rearrangement of variables. These rules effectively "bubble" the A's to the left and the C's to the right, maintaining the correct order and equal numbers of each variable.
4. Rule 8 is crucial: it ensures that B's are replaced with lowercase b's only when adjacent to an existing lowercase b. This prevents premature conversion of B's and maintains the structure of the string.
5. Rules 9-11 convert the variables to their corresponding terminals once they are in the correct position.

The grammar maintains the equal numbers of A's, B's, and C's while rearranging them into the correct order. The context-sensitive nature of the rules allows for this precise control over the string's structure.

Type 0 Grammar (Unrestricted 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:

- $\alpha \rightarrow \beta$

where:

- α is a string of terminals and non-terminals 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
```

Construction of a Grammar Generating a Language

Let's consider some languages and convert them into a grammar G which produces those languages.

Example 1

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

Solution: Since L(G) = {a^m b^n | 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
B → bB
```

Derivations:

- 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 })

Example 2

Problem: Suppose, L(G) = {a^m b^n | m > 0 and n ≥ 0}. We have to find out the grammar G which produces L(G).

Solution: Since L(G) = {a^m b^n | m > 0 and n ≥ 0}, the set of strings accepted can be rewritten as: L(G) = {a, aa, ab, aaa, aab, abb, ...}

Here, the start symbol has to take at least one a followed by any number of b including null. To accept the string set {a, aa, ab, aaa, aab, abb, ...}, we have taken the productions:

```
S → aA
A → aA
A → B
B → bB
B → λ
```

Derivations:

- S → aA → aB → aλ → a (Accepted)
- S → aA → aaA → aaB → aaλ → aa (Accepted)
- S → aA → aB → abB → abλ → ab (Accepted)
- S → aA → aaA → aaaA → aaaB → aaaλ → aaa (Accepted)
- S → aA → aaA → aaB → aabB → aabλ → aab (Accepted)
- S → aA → aB → abB → abbB → abbλ → 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 → aA, A → aA | B, B → λ | bB })

Conclusion

Context-sensitive languages offer a significant improvement over context-free languages by considering surrounding context in production rules. This enables the description of complex patterns beyond the reach of CFLs.

The Chomsky hierarchy provides a framework for understanding the power and limitations of different types of grammars and their corresponding automata. Each level in the hierarchy has specific characteristics and applications in computer science and linguistics.

Understanding these grammar types is essential for language processing, compiler design, and theoretical computer science.