Una **gramática** es un conjunto de reglas que nos permite generar todas las palabras válidas de un lenguaje. Es como un "recetario" para crear frases correctas.

4.1 Definición Formal:

$$G = (V, T, P, S)$$

Toda gramática tiene **4 componentes**:

1. V **(Variables o No Terminales - NT)** Son símbolos que se pueden sustituir por otras cosas. Se escriben en MAYÚSCULAS.

Ejemplo: S, A, B, E (Expresión), Número, etc.

2. T **(Terminales)** Son símbolos que aparecen en las palabras finales y NO se pueden sustituir. Se escriben en minúsculas o son caracteres específicos.

Ejemplo: a, b, 0, 1, +, -, números, palabras del lenguaje final

3. P **(Producciones o Reglas)** Son las reglas de transformación. Tienen la forma: **Variable → combinación de terminales y/o variables**

4. S **(Símbolo Inicial o Axioma)** Es la variable por la que SIEMPRE empezamos.

Ejemplo Completo Simple:

Queremos crear una gramática para el lenguaje: **L = {a, aa, aaa, aaaa, ...}** (una o más 'a's)

```
G = (V, T, P, S) donde:
V = {S}              // Solo tenemos una variable
T = {a}              // Solo tenemos un terminal
S = S                // Símbolo inicial
P = {                // Reglas:
    S → aS           // S se puede convertir en 'a' seguido de S
    S → a            // S se puede convertir en solo 'a'
}
```

¿Cómo generamos palabras? - Para "a": S → a ✓ - Para "aa": S → aS → aa ✓ - Para "aaa": S → aS → aaS → aaa ✓

4.2 Clasificación de Gramáticas (Jerarquía de Chomsky)

Las gramáticas se clasifican de **menos restrictivas** a **más restrictivas**:

Tipo 0: Sin Restricciones

- **Regla:** x → y, donde x puede ser cualquier combinación (pero al menos un símbolo), y puede ser cualquier cosa (incluso vacío)
- Son las más potentes pero también las más difíciles de analizar

Tipo 1: Sensibles al Contexto

- **Regla:** $\alpha \to \beta$ donde $|\alpha| \leq |\beta|$ (la parte derecha debe ser igual o más larga)
- **Importante:** La sustitución depende del "contexto" (los símbolos alrededor)
- Ejemplo: aSb → aXYb (sustituimos S por XY, pero solo si está entre 'a' y 'b')

Tipo 2: Independientes del Contexto (GIC)

- **Regla:** x → y donde **x debe ser UN SOLO no terminal**
- La sustitución NO depende del contexto
- Ejemplo: S → aSb (sustituimos S sin importar qué hay alrededor)

Tipo 3: Regulares

- Las más restrictivas
- **Reglas lineales:** A → aB, A → Ba, o A → a
- Ejemplo: S → aS, S → a

Relación: Tipo 3 ⊂ Tipo 2 ⊂ Tipo 1 ⊂ Tipo 0

4.3. Gramáticas Regulares (Tipo 3)

Son un subconjunto especial de las GIC.

Lineal por la Derecha:

```
A → xB  (terminal(es) seguido de una variable)
A → x   (solo terminal(es))
```

Ejemplo: Lenguaje de palabras que terminan en 'b':

```
S → aS | bS | b
```

- Genera: b, ab, aab, bb, abb, etc.

Lineal por la Izquierda:

```
A → Bx  (variable seguida de terminal(es))
A → x   (solo terminal(es))
```

Ejemplo:

```
S → Sa | Sb | a
```

- Genera: a, aa, ab, aaa, aab, etc.

Importante: NO se pueden mezclar reglas lineales por la derecha con lineales por la izquierda en la misma gramática regular.

4.4 Lenguaje de una Gramática

$$L(G) = \{w \in T* | S \Rightarrow *w\}$$

Esto significa: El lenguaje generado por G es el conjunto de todas las palabras formadas **solo por terminales** que se pueden derivar desde S.

Ejemplo:

```
S → aSb |
T = {a, b}
```

Derivaciones: - S ⇒ ε → palabra: “” (cadena vacía) - S ⇒ aSb ⇒ ab → palabra: “ab” - S ⇒ aSb ⇒ aaSbb ⇒ aabb → palabra: “aabb” - S ⇒ aSb ⇒ aaSbb ⇒ aaaSbbb ⇒ aaabbb → palabra: “aaabbb”

L(G) = $\{a^n b^n \mid n \geq 0\}$ = {ε, ab, aabb, aaabbb, …}

4.5 Árboles de Derivación

Un **árbol de derivación** es una representación visual de cómo se genera una palabra.

Características:

1. **Raíz:** Símbolo inicial S
2. **Nodos internos:** Variables (no terminales)
3. **Hojas:** Terminales o ε
4. Si un nodo A tiene hijos $X_1, X_2, \ldots, X_k$, entonces existe la regla $A \rightarrow X_1X_2\ldots X_k$

Ejemplo:

Gramática:

```
E → E + E
E → E * E
E → (E)
E → id
```

Para la expresión: **id + id * id**

```
        E
       /|\
      E + E
      |   /|\
     id  E * E
         |   |
        id  id
```

Lectura del árbol (hojas de izquierda a derecha): id + id * id ✓

4.6 Ambigüedad

Una gramática es **ambigua** si una misma palabra puede tener **DOS O MÁS árboles de derivación diferentes**.

Ejemplo Clásico:

Gramática:

```
E → E + E
E → E * E
E → id
```

Para la palabra: **id + id * id**

Árbol 1: (interpretado como (id + id) * id)

```
        E
       /|\
      E * E
     /|\  |
    E + E id
    |   |
   id  id
```

Árbol 2: (interpretado como id + (id * id))

```
        E
       /|\
      E + E
      |  /|\
     id E * E
        |   |
       id  id
```

¡DOS ÁRBOLES DIFERENTES! → Gramática AMBIGUA

¿Por qué es un problema?

En programación, “2 + 3 * 4” debe dar **14** (no 20), por la precedencia de operadores. Una gramática ambigua no respeta esto.

Solución: Crear una gramática no ambigua

```
E → E + T | T
T → T * F | F
F → (E) | id
```

Esta gramática asegura que * tiene más precedencia que +.

4.7 Ejemplos de GIC

Ejemplo 1: Palíndromos sobre {0,1}

```
S → 0S0 | 1S1 | 0 | 1 |
```

Derivaciones: - S ⇒ ε → “” - S ⇒ 0 → “0” - S ⇒ 1 → “1” - S ⇒ 0S0 ⇒ 00 → “00” - S ⇒ 1S1 ⇒ 10S01 ⇒ 10101 → “10101” ✓

L(G) = {palabras que se leen igual de izquierda a derecha que de derecha a izquierda}

Ejemplo 2: L = {0^n1^n | n ≥ 0}

```
S → 0S1 |
```

Derivaciones: - S ⇒ ε → “” - S ⇒ 0S1 ⇒ 01 → “01” - S ⇒ 0S1 ⇒ 00S11 ⇒ 0011 → “0011”

L(G) = {ε, 01, 0011, 000111, …}

4.8 Formas Normales

Las formas normales simplifican las gramáticas para facilitar el análisis.

Pasos para convertir a Forma Normal:

Paso 1: Eliminar producciones ε (excepto S → ε) Ejemplo:

```
Original:
S → AB
A → aA |
B → bB |

Paso 1: Identificar variables anulables (que pueden generar  )
- A es anulable (A →  )
- B es anulable (B →  )
- S es anulable (porque S → AB y tanto A como B son anulables)

Paso 2: Crear nuevas reglas eliminando variables anulables
S → AB | A | B |
A → aA | a
B → bB | b
```

Paso 2: Eliminar producciones unitarias (A → B) Ejemplo:

```
Original:
S → A
A → B | a
B → b

Las producciones unitarias son: S → A y A → B

Eliminándolas:
S → a | b  (sustituyendo lo que generan A y B)
A → b | a
```

```
B → b
```

Paso 3: Eliminar símbolos inútiles **No generadores:** Símbolos que no pueden convertirse en terminales. **No alcanzables:** Símbolos a los que no se puede llegar desde S.

Ejemplo:

```
S → AB | a
A → b
C → c
D → SD

- C no es alcanzable (no hay regla que lo genere desde S)
- D no es generador (genera D infinitamente pero nunca terminales)
- Eliminar C y D
```

4.9 Forma Normal de Chomsky (FNC)

Reglas permitidas:

1. **A → BC** (dos no terminales)
2. **A → a** (un solo terminal)
3. **S → ε** (solo si ε está en el lenguaje)

Ventajas:

- Árbol binario
- Para una palabra de longitud n, se necesitan exactamente **2n - 1** pasos

Ejemplo de conversión:

Original:

```
S → ASA | aB
A → B | S
B → b |
```

En FNC:

```
S → AA  | TB |
A  → SA
A → b | AA  | TB
B → b
T → a
```

4.10 Forma Normal de Greibach (FNG)

Reglas permitidas:

1. **A → aα** donde a es terminal y α es una cadena de variables (puede ser vacía)
2. **S → ε** (solo si ε está en el lenguaje)

Ventajas:

- Para una palabra de longitud n, se necesitan exactamente **n** pasos
- Útil para analizadores sintácticos descendentes

Ejemplo:

```
S → aAB | b
A → aA | a
B → bB | b
```

Todas las producciones empiezan con un terminal