
Gramáticas y lenguajes formales. La jerarquía de Chomsky.

Tema 4

Algorítmica y Modelos de Computación

Índice

1. Introducción. Lenguajes, Gramáticas Y Autómatas.
2. Gramáticas y Lenguajes Formales.
 - 2.1. Lenguajes Formales.
 - 2.1.1. Alfabeto
 - 2.1.2. Palabra
 - 2.1.3. Operaciones con palabras
 - 2.1.4. Lenguajes
 - 2.1.5. Operaciones con Lenguajes
 - 2.2. Gramáticas Formales.
 - 2.2.1. Concepto de gramática formal
 - 2.2.2. Tipos de Gramáticas. **Jerarquía de Chomsky**
 - 2.2.3. Árboles de derivación
 - 2.2.4. Ambigüedad
 - 2.2.5. Recursividad
 - 2.2.6. Factorización a izquierdas

1. Introducción.

□ **Informática Teórica :**

- Confluencia de investigaciones sobre los fundamentos de las matemáticas, teoría de máquinas, lingüística, etc.
- Ciencia **multidisciplinar**, apoyada en la observación de que los mismos fenómenos pueden aparecer y explicar áreas completamente distintas y en apariencia sin conexión.

1. Introducción.

- La aparición de estudios sobre Informática Teórica se remontan a primeros de siglo XX. Dentro de los **hitos más importantes** se pueden citar los siguientes :
 - En 1900 **David Hilbert** propone “**el problema de la decisión**”, que trata de descubrir un método general para decidir si una fórmula lógica es verdadera o falsa.
 - Aparición del artículo de **Kurt Gödel**, en 1931, “*On Formally Undecidable Propositions in Principia Mathematica and Related Systems*”. Este trabajo puede considerarse como la mayor realización matemática del siglo XX. El teorema de Gödel sostiene que cualquier teoría matemática contendrá afirmaciones “**indecidibles**”.
 - En 1937, el matemático inglés **Alan Mathison Turing** publica un famoso artículo que desarrolla el teorema de Gödel, y que puede considerarse como el **origen** oficial de la Informática Teórica. En este artículo se introduce el concepto de **máquina de Turing**, una entidad matemática abstracta que formaliza por primera vez el concepto de **algoritmo**.

1. Introducción.

- Paradójicamente, la **máquina de Turing**, mecanismo que formaliza el concepto de algoritmo, que pretende ser lo suficientemente general como para resolver cualquier problema posible (siempre que sea computacionalmente abordable), se introduce para demostrar la validez de los postulados de Gödel. Es decir, Turing demuestra que existen **problemas irresolubles**, inasequibles para cualquier máquina de Turing, y por ende, para cualquier ordenador.
- Al ser la máquina de Turing un modelo ideal de máquina capaz de adoptar una infinidad de estados posibles, resulta obvio pensar que su realización está fuera del alcance práctico. Sin embargo, debido a la gran capacidad de los ordenadores actuales, la máquina de Turing resulta ser un modelo adecuado de su funcionamiento.
- En 1938, se produce una importante aportación a la Informática Teórica, en el terreno de la Ingeniería Eléctrica. El matemático **Claude Elwood Shannon** publica el artículo “*A Symbolic Analysis of Relay and Switching Circuits*”. En él se establecen las bases para la aplicación de la lógica matemática a los circuitos electrónicos. En las décadas siguientes las ideas de Shannon se desarrollaron ampliamente, dando lugar a la formalización de una **teoría sobre máquinas secuenciales y autómatas finitos**.

1. Introducción.

- Desde su nacimiento, la **teoría de autómatas** ha encontrado aplicación en muy diversos campos. Esto se debe a que resulta muy natural considerar, tanto los autómatas como las máquinas secuenciales, sistemas capaces de transmitir (procesar) información. En definitiva, esto es equiparable a cualquier sistema existente en la naturaleza, que **recibe** señales de su entorno, **reacciona** ante ellas y **emite** así nuevas señales al ambiente que le rodea.
- Algunos de los campos donde ha encontrado aplicación la teoría de autómatas son:
 - Teoría de la Comunicación
 - Teoría de Control
 - Lógica de los circuitos secuenciales
 - Ordenadores
 - Teoría lógica de los sistemas evolutivos y auto-reproductivos
 - Reconocimiento de patrones
 - Fisiología del sistema nervioso
 - Traducción automática de lenguajes
 - etc

1. Introducción. Teoría de Lenguajes Formales y Teoría de Autómatas.

- En 1950 se vincula la Informática Teórica con otro terreno con el que habitualmente ni se había relacionado (ni siquiera se consideraba como científico) : la **Lingüística**.
- En este año el lingüista Avram Noam **Chomsky** publica su “Teoría de las Gramáticas Transformacionales”, que establece las bases de la **lingüística matemática**. Proporcionó una herramienta de inestimable ayuda, no sólo en el estudio de los lenguajes naturales, sino también en la **formalización de los lenguajes de ordenador**, que comenzaban a aparecer en estos años.
- La Teoría de Lenguajes Formales resultó tener una sorprendente relación con la Teoría de las Máquinas Abstractas. Los mismos fenómenos aparecían en ambas disciplinas y era posible establecer correspondencias muy claras entre ellas (isomorfismos).
- **Chomsky** clasificará los **lenguajes formales** de acuerdo a una **jerarquía** de cuatro niveles, conteniendo cada uno de todos los siguientes.
 - El lenguaje más general será, pues, de **tipo 0**, y no posee restricción alguna. Este conjunto engloba el conjunto de todos los lenguajes posibles.
 - En el segundo nivel los lenguajes de **tipo 1**, también llamados lenguajes “**sensibles al contexto**”, al permitir que el “papel” de las palabras dependa de la posición en que aparezcan (es decir, del contexto). La mayor parte de los lenguajes de ordenador pertenecen a este tipo.
 - En tercer lugar los lenguajes de **tipo 2**, o lenguajes “**independientes del contexto**”. En ellas el significado de una palabra es independiente del lugar que ocupa en la frase.
 - Finalmente, los lenguajes de tipo 3, o **lenguajes regulares**, son los que presentan una estructura más sencilla.

1. Introducción. Teoría de Lenguajes Formales y Teoría de Autómatas.

- Paralelamente a la jerarquía de lenguajes aparece otra de máquinas abstractas equivalentes, como se observa en el esquema siguiente :

Lenguajes	Máquinas	
Lenguajes Tipo 0	Máquina de Turing	Problemas no enumerables
Lenguajes Tipo 1	Autómata linealmente acotado	Problemas recursivamente enumerables
Lenguajes Tipo 2	Autómata con pila	
Lenguajes Tipo 3	Autómata Finito	Expresiones regulares

- Cada uno de estos tipos de máquinas es capaz de resolver problemas cada vez más complicados, hasta llegar a las máquinas de Turing. Como descubrió Turing, existen una serie de problemas que **NO** son computacionalmente abordables y que reciben el nombre de “**problemas no enumerables**”.

1. Introducción. Lenguajes, Gramáticas Y Autómatas.

Campos fundamentales con influencia:

□ Matemáticas:

- Se considera a Turing como padre de la “Teoría de la Computabilidad” y de la formalización del concepto de algoritmo.

□ Ingeniería Eléctrica:

- La evolución de la Ingeniería Eléctrica en base a las ideas de Shannon, dio lugar a la formalización de una teoría de las máquinas secuenciales y de los autómatas finitos. Los autómatas aparecen como sistemas capaces de transmitir información.

□ Lingüística:

- La “Teoría de las Gramáticas Transformacionales” de Chomsky en 1950 proporcionó una herramienta fundamental para la formalización de los lenguajes de ordenador.

1. Introducción. Lenguajes, Gramáticas Y Autómatas.

- Terminología y algunos conceptos asociados al proceso de traducción.
 - Se establecen los términos de **lenguaje formal** , definidos por **reglas preestablecidas**, y de *lenguaje natural*, no cuentan con reglas gramaticales formales.
 - Así, el estudio de los lenguajes se reduce al **análisis de la estructura de las frases** (Gramática) **y del significado** de las mismas (Semántica).
 - A su vez, la **Gramática** puede **analizar** las formas que toman las **palabras** (Morfología), su combinación para formar **frases** correctas (Sintaxis), y las propiedades del lenguaje hablado (Fonética) (única no aplicable a los lenguajes de ordenador).
 - Para realzar el papel de la gramática en el **proceso de traducción** se indican los componentes básicos de que consta el compilador para un determinado lenguaje de programación: análisis léxico, análisis sintáctico y generación de código.

1. Introducción. Lenguajes, Gramáticas Y Autómatas.

- Se establece un **isomorfismo** entre la Teoría de Lenguajes Formales y la Teoría de Autómatas, estableciendo una conexión entre la clase de lenguajes generados por ciertos tipos de gramáticas y la clase de lenguajes reconocibles por ciertas máquinas. Se detalla la jerarquía de lenguajes de Chomsky y se establecen las siguientes relaciones :

Gramáticas	Lenguajes	Máquinas
Sin restricciones o de Tipo 0	Sin restricciones o de Tipo 0	Máquina de Turing
Sensible al contexto o de Tipo 1	Sensible al contexto o de Tipo 1	Autómata linealmente acotado
Libre de contexto o de Tipo 2	Libre de contexto o de Tipo 2	Autómata con pila
Regular o de Tipo 3	Regular o de Tipo 3	Autómata Finito

- Cada uno de estos tipos/máquinas añade restricciones al tipo/máquina del nivel superior.

Índice

1. Introducción. Lenguajes, Gramáticas Y Autómatas.

2. **Gramáticas y Lenguajes Formales.**

2.1. Lenguajes Formales.

2.1.1. Alfabeto

2.1.2. Palabra

2.1.3. Operaciones con palabras

2.1.4. Lenguajes

2.1.5. Operaciones con Lenguajes

2.2. Gramáticas Formales.

2.2.1. Concepto de gramática formal

2.2.2. Tipos de Gramáticas. **Jerarquía de Chomsky**

2.2.3. Árboles de derivación

2.2.4. Ambigüedad

2.2.5. Recursividad

2.2.6. Factorización a izquierdas

2.1.Lenguajes formales. 2.1.1. Alfabeto

- Se llama **alfabeto** a un conjunto finito, no vacío, cuyos elementos se denominan “letras” o “*símbolos*”. Se definen los alfabetos por la enumeración de los símbolos que contiene.

- Ejemplos :
 - $A1=\{A, B, C, D, E, F, G, \dots, Z\}$
 - $A2=\{0,1\}$
 - $A3=\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
 - $A4=\{(,)\}$

2.1.Lenguajes formales. 2.1.2. Palabra

- Se denomina **palabra** a toda *secuencia finita de letras* formada con los símbolos de un alfabeto.

- Palabras sobre A1 : JOSE, ANA, RREDF, ABACZA

- Palabras sobre A2 : 0 1 11001100 1111

- Palabras sobre A3 : 12 9065 67890

- Palabras sobre A4 : ((()))()()((

Se usarán letras minúsculas para representar las palabras de un alfabeto :

x = JOSE(sobre A1) y = (()) (sobre A4) z = 123456 (sobre A3)

- **Longitud de una palabra:** número de símbolos (letras) que la componen:

| x | = 4 | y | = 4 | z | = 6

- Se define la **palabra vacía** como aquella cuya longitud es cero. Se representa mediante la letra λ o ε .

- Se define “**universo del discurso**” o “**lenguaje universal**” sobre el alfabeto Σ **$W(\Sigma)$** al conjunto de palabras que se pueden formar con las letras de un alfabeto, $W(\Sigma)$ es un conjunto infinito.

- Ejemplo: un alfabeto con el menor número posible de letras (1).

- $A = \{ a \}$

- En este caso, $W(A) = \{ \lambda, a, aa, aaa, aaaa, \dots \}$, y contiene un número infinito de elementos.

- La palabra vacía pertenece a todos los lenguajes universales de todos los alfabetos.

2.1.Lenguajes formales. 2.1.3 Operaciones con palabras

□ **Concatenación** de palabras

Sean dos palabras x, y tales que $x \in W(\Sigma), y \in W(\Sigma)$

Supongamos que $x=A_0A_1..... A_i \quad |x| = i$; $y= B_0B_1..... B_j \quad |y| = j$

Se llama concatenación de las palabras x e y (y se representa por xy) a otra palabra, z , obtenida poniendo las letras de x y a continuación las de y :

$z= A_0A_1..... A_i B_0B_1..... B_j$

Se cumple que: $|z|=|x|+|y|$

□ **propiedades** de la concatenación :

■ *Operación cerrada.* Es decir, la concatenación de dos palabras de $W(A)$ es otra palabra de $W(A)$. Si $x \in W(A)$ e $y \in W(A)$, entonces $xy \in W(A)$.

■ *Propiedad asociativa :* $x(yz)=(xy)z$

■ *Existencia de elemento neutro.* El elemento neutro de esta operación es la palabra vacía λ , tanto por la derecha como por la izquierda. Siendo x una palabra cualquiera, se cumple :

$$x\lambda = \lambda x = x$$

■ **no cumple la propiedad conmutativa**

2.1.Lenguajes formales. 2.1.3 Operaciones con palabras

□ **Potencia de una palabra**

Se denomina potencia i -ésima de una palabra a la concatenación consigo misma i veces.

$$x^i = \underbrace{xxx \dots xx}_i$$

se cumplen las siguientes relaciones

$$x^{i+1} = x^i x = x x^i \quad (i > 0)$$

$$x^i x^j = x^{i+j} \quad (i, j > 0)$$

Para que ambas relaciones se cumplan también para $i, j = 0$, basta con definir $x^0 = \lambda$, cualquiera que sea x .

Ejemplo: $x = ABCD$, entonces

$$x^2 = xx = ABCDABCD$$

$$x^3 = xxx = ABCDABCDABCD$$

■ la longitud de la potencia es $|x^i| = i * |x|$

□ **Reflexión de palabras**

Sea $x = A_0 A_1 \dots A_n$, se denomina palabra refleja o inversa de x , representado por x^{-1} , a

$$x^{-1} = x^r = A_n A_{n-1} \dots A_0$$

está formada por las mismas letras, pero ordenadas de forma inversa.

2.1.Lenguajes formales. 2.1.4. Lenguajes

- Se denomina **lenguaje sobre el alfabeto** Σ a cualquier subconjunto del lenguaje universal $W(\Sigma)$

$$L \subset W(\Sigma)$$

- El conjunto vacío, ϕ , es un subconjunto de $W(\Sigma)$. Este lenguaje no debe confundirse con aquel que contiene únicamente a la palabra vacía. Para diferenciarlos hemos de darnos cuenta de la distinta cardinalidad de ambos conjuntos, ya que

$$C(\phi) = 0$$

$$C(\{\lambda\}) = 1$$

- Estos dos conjuntos serán lenguajes sobre cualquier alfabeto. El alfabeto en sí puede considerarse como un lenguaje : el formado por todas las posibles palabras de una letra.

2.1.Lenguajes formales. 2.1.5. Operaciones con Lenguajes

□ **Unión** de lenguajes

Consideremos dos lenguajes diferentes definidos sobre el mismo alfabeto $L1 \subset W(\Sigma)$ y $L2 \subset W(\Sigma)$

Se denomina unión de ambos lenguajes al lenguaje formado por las palabras de ambos lenguajes :

$$L1 \cup L2 = \{ x / x \in L1 \text{ ó } x \in L2 \}$$

□ **Propiedades** de esta operación :

- *Operación cerrada.* La unión de dos lenguajes definidos sobre el mismo alfabeto será otro lenguaje definido sobre ese alfabeto
- *Propiedad asociativa.* $(L1 \cup L2) \cup L3 = L1 \cup (L2 \cup L3)$
- *Existencia de elemento neutro.* $L \cup \phi = \phi \cup L = L$
- *Propiedad conmutativa.* Se verifica que $L1 \cup L2 = L2 \cup L1$
- *Propiedad de idempotencia.* Se verifica que $L \cup L = L$

2.1.Lenguajes formales. 2.1.5. Operaciones con Lenguajes

□ Concatenación de lenguajes

Consideremos dos lenguajes definidos sobre el mismo alfabeto, L_1 y L_2 . La concatenación o producto de estos lenguajes es el lenguaje

$$L_1 \bullet L_2 = \{ xy / x \in L_1 \text{ y } x \in L_2 \}$$

Las palabras de este lenguaje estarán formadas al concatenar cada una palabra del primero de los lenguajes con otra del segundo.

- La concatenación de lenguajes con el lenguaje vacío es:

$$\phi L = L \phi = \phi$$

□ Propiedades de esta operación :

- *Operación cerrada.* La concatenación de lenguajes sobre el mismo alfabeto es otro lenguaje sobre ese alfabeto.
- *Propiedad asociativa.* $(L_1 L_2) L_3 = L_1 (L_2 L_3)$
- *Elemento neutro.* Cualquiera que sea el lenguaje considerado, el lenguaje de la palabra vacía cumple que $\{\lambda\}L = L\{\lambda\} = L$

2.1.Lenguajes formales. 2.1.5. Operaciones con Lenguajes

□ **Potencia de un lenguaje**

Se define la potencia i -ésima de un lenguaje a la operación de concatenarlo consigo mismo i veces.

$$L^i = \underbrace{LLL \dots L}_i$$

□ Se cumplen las siguientes relaciones

- $L^{i+1} = L^i L = L L^i \quad (i > 0)$
- $L^i L^j = L^{i+j} \quad (i, j > 0)$
- Para que las relaciones se cumplan para $i, j = 0$, se define
 - $L^0 = \{\lambda\}$, cualquiera que sea L

2.1.Lenguajes formales. 2.1.5. Operaciones con Lenguajes

□ Clausura positiva de un lenguaje

Se define la clausura positiva de un lenguaje L :

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

Lenguaje obtenido uniendo el lenguaje con todas sus potencias posibles excepto L^0 . Si L no contiene la palabra vacía, la clausura positiva tampoco.

- Ya que cualquier alfabeto Σ es un lenguaje sobre él mismo (formado por las palabras de longitud 1), al aplicarle esta operación se observa que

$$\Sigma^+ = W(\Sigma) - \{\lambda\}$$

2.1.Lenguajes formales. 2.1.5. Operaciones con Lenguajes

□ Cierre o Clausura de un lenguaje

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

Se define el cierre o clausura de un lenguaje L como :

Lenguaje obtenido uniendo el lenguaje con todas sus potencias posibles, incluso L^0
Todas las clausuras contienen la palabra vacía.

□ Se cumplen las siguientes relaciones:

■ $L^* = L^+ \cup \{\lambda\}$

■ $L^+ = L L^* = L^* L$ (será imposible obtener la palabra vacía)

□ Ya que el alfabeto Σ es un lenguaje sobre sí mismo, al aplicársele esta operación.

$$\Sigma^* = W(\Sigma)$$

□ Se denominará Σ^* al lenguaje universal o “universo del discurso” sobre el alfabeto Σ

□ Reflexión de lenguajes. Se llama lenguaje reflejo o inverso de L, representándose por L^{-1}

$L^{-1} = \{ x^{-1} / x \in L \}$ lenguaje que contiene las palabras inversas a las palabras de L

Índice

1. Introducción. Lenguajes, Gramáticas Y Autómatas.

2. Gramáticas y Lenguajes Formales.

2.1. Lenguajes Formales.

2.1.1. Alfabeto

2.1.2. Palabra

2.1.3. Operaciones con palabras

2.1.4. Lenguajes

2.1.5. Operaciones con Lenguajes

2.2. Gramáticas Formales.

2.2.1. Concepto de gramática formal

2.2.2. Tipos de Gramáticas. Jerarquía de Chomsky

2.2.3. Árboles de derivación

2.2.4. Ambigüedad

2.2.5. Recursividad

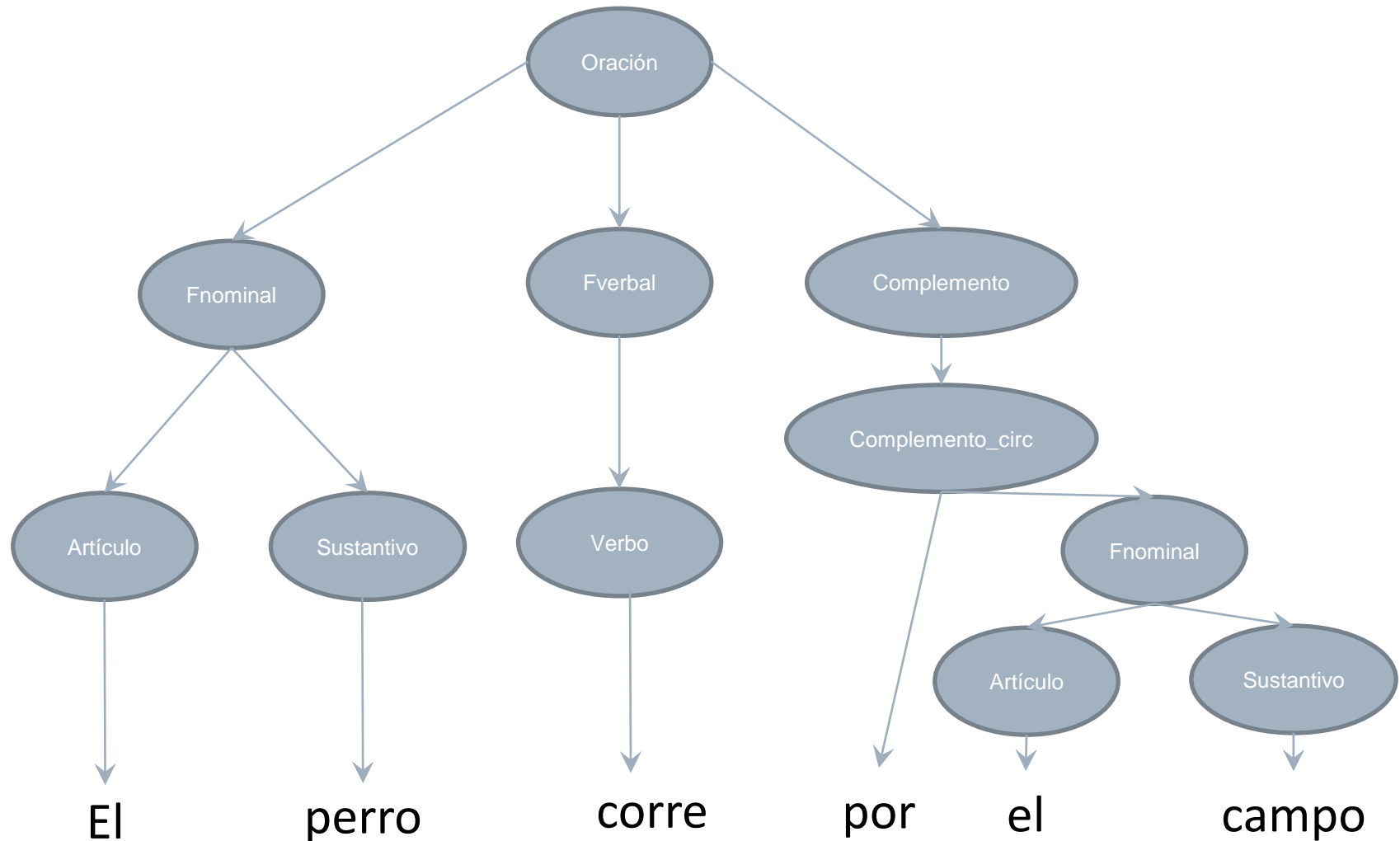
2.2.6. Factorización a izquierdas

2.2. Gramáticas Formales. 2.2.1. Concepto de gramática formal

- Una **gramática** *define la estructura de las frases y de las palabras de un lenguaje.*
- Las gramáticas son un método para la generación de palabras de un lenguaje a partir de un alfabeto.
 - para generar estas palabras se utilizan las derivaciones.
 - se denominan formales porque se centran en los estudios de los lenguajes formales que son aquellos que están definidos a partir de reglas preestablecidas. Para los lenguajes naturales existen otro tipo de gramáticas.

Gramática del Castellano

Una gramática del castellano como diagrama sintáctico



2.2. Gramáticas Formales. 2.2.1. Concepto de gramática formal

Ejemplo: consideremos la instrucción de cualquier lenguaje $x = y+2*z$ con:

■ conjunto de **producciones** :

$\langle \text{instrucción} \rangle ::= \langle \text{asignación} \rangle$

$\langle \text{asignación} \rangle ::= \langle \text{identificador} \rangle "=" \langle \text{expresión} \rangle$

$\langle \text{expresión} \rangle ::= \langle \text{sumando} \rangle$

$\langle \text{expresión} \rangle ::= \langle \text{sumando} \rangle "+" \langle \text{expresión} \rangle$

$\langle \text{sumando} \rangle ::= \langle \text{factor} \rangle$

$\langle \text{sumando} \rangle ::= \langle \text{factor} \rangle "*" \langle \text{sumando} \rangle$

$\langle \text{factor} \rangle ::= \langle \text{identificador} \rangle$

$\langle \text{factor} \rangle ::= \langle \text{número} \rangle$

■ Reglas morfológicas :

$\langle \text{identificador} \rangle ::= "x"$

$\langle \text{identificador} \rangle ::= "y"$

$\langle \text{identificador} \rangle ::= "z"$

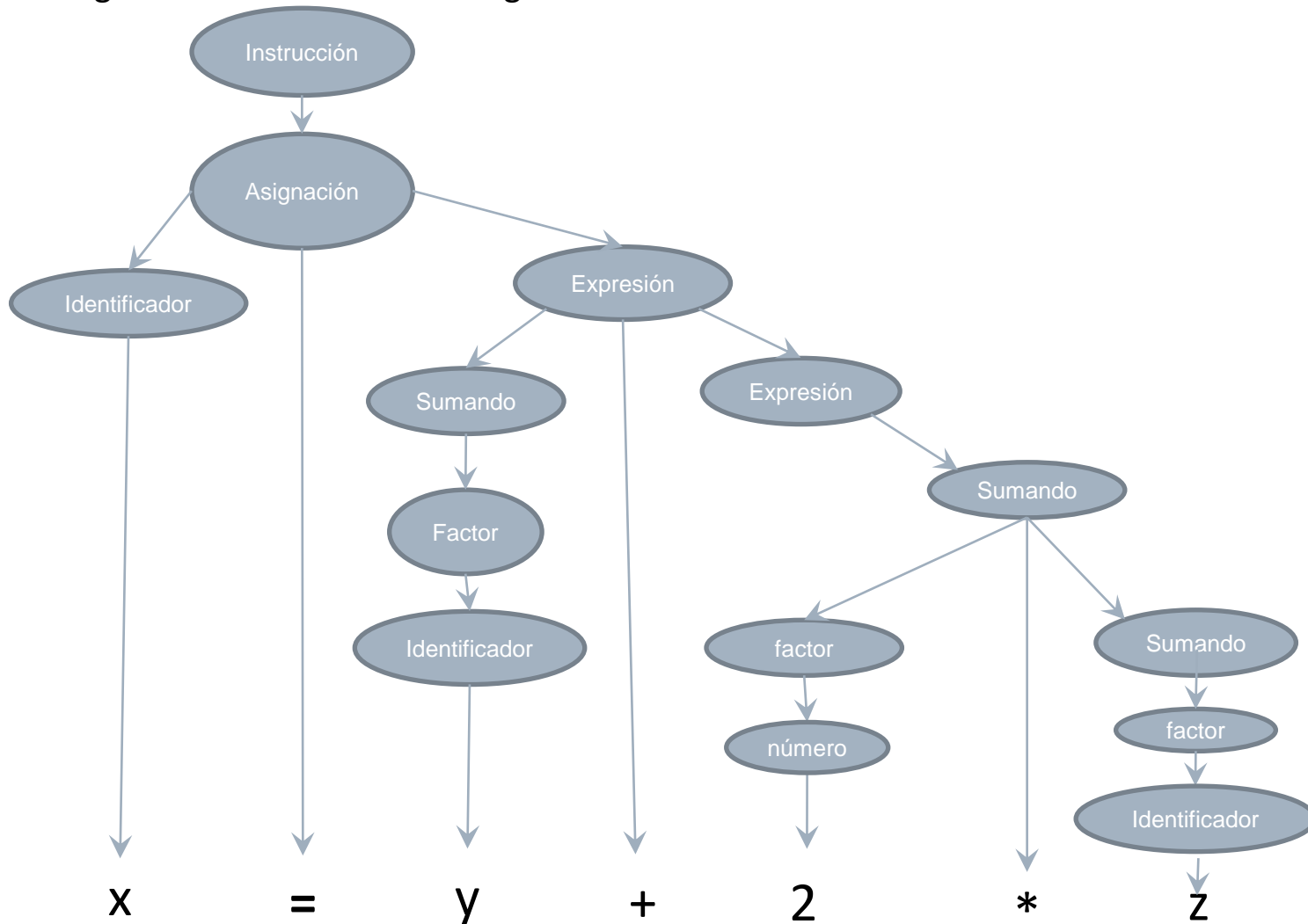
$\langle \text{número} \rangle ::= "2"$

□ Obtenemos la expresión $x = y+2*z$ a partir de $\langle \text{instrucción} \rangle$ así:

$\langle \text{instrucción} \rangle \rightarrow \langle \text{asignación} \rangle \rightarrow \langle \text{identificador} \rangle = \langle \text{expresión} \rangle \rightarrow x = \langle \text{expresión} \rangle \rightarrow x =$
 $\langle \text{sumando} \rangle + \langle \text{expresión} \rangle \rightarrow x = \langle \text{factor} \rangle + \langle \text{expresión} \rangle \rightarrow x = \langle \text{identificador} \rangle +$
 $\langle \text{expresión} \rangle \rightarrow x = y + \langle \text{expresión} \rangle \rightarrow x = y + \langle \text{sumando} \rangle \rightarrow x = y + \langle \text{factor} \rangle * \langle \text{sumando} \rangle \rightarrow x$
 $= y + \langle \text{número} \rangle * \langle \text{sumando} \rangle \rightarrow x = y + 2 * \langle \text{sumando} \rangle \rightarrow x = y + 2 * \langle \text{factor} \rangle \rightarrow x = y + 2 *$
 $\langle \text{identificador} \rangle \rightarrow x = y + 2 * z$

Gramática Formal

Una gramática formal como diagrama sintáctico



JFLAP : (Gej_1.jff)

File Input Test Convert Help

Editor Brute Parser

Start Pause Step Noninverted Tree

Input: $x=y+2*z$
String accepted! 294 nodes generated.

Input Field Text Size (For optimization, move one of the window size adjusters around this window after resizing the text field)

Table Text Size

LHS	RHS
E	→ S
E	→ S+E
S	→ F
S	→ F*S
F	→ I
F	→ M
I	→ x
I	→ y
I	→ z
M	→ 2

Derived z from I. Derivations complete.

JFLAP : (Gej_1.jff)

File Input Test Convert Help

Editor Brute Parser

Start Pause Step Derivation Table

Input: $x=y+2*z$
String accepted! 294 nodes generated.

Input Field Text Size (For optimization, move one of the window size adjusters around this window after resizing the text field)

Table Text Size

LHS	RHS
E	→ S
E	→ S+E
S	→ F
S	→ F*S
F	→ I
F	→ M
I	→ x
I	→ y
I	→ z
M	→ 2

Table Text Size	
N	→ A
A	→ =E
E	→ S+E
E	→ S
S	→ F*S
S	→ F
S	→ F
F	→ I
F	→ M
F	→ I
I	→ x
I	→ y
I	→ z
M	→ 2
I	→ z
N	
A	
=E	
=S+E	
=S+S	
=S+F*S	
=F+F*S	
=F+F*I	
=F+F*I	
=I+M*I	
=I+M*I	
=I+M*I	
x=I+M*I	
x=y+M*I	
x=y+2*I	
x=y+2*z	

Derived z from I. Derivations complete.

2.2. Gramáticas Formales. 2.2.1. Concepto de gramática formal

□ **Producción o regla ($x::=y$):**

es un par ordenado (x,y) con $x,y \in \Sigma^*$, es decir, si se encuentra x como parte de cualquier palabra v se puede sustituir x por y en v , lo que permite transformar palabras en otras

□ **Derivación directa $v \rightarrow w$:**

aplicación de una producción a una palabra v para convertirla en otra w donde

$v=ZXU$ y $w=ZYU$ ($v, w, z, u \in \Sigma^*$)

■ Se cumple que para cada producción $x::=y$ existe una derivación directa (haciendo $z=u=\lambda$): $x \rightarrow y$

□ **Derivación $v \rightarrow^* w$:**

aplicación de una secuencia de producciones a una palabra.

□ **Longitud de la derivación:**

número de derivaciones que hay que aplicar para obtener la palabra.

□ **Derivación más a la izquierda:** Se utiliza en cada derivación directa la producción aplicada a los símbolos más a la izquierda de la palabra.

□ **Derivación más a la derecha:** Se utiliza en cada derivación directa la producción aplicada a los símbolos más a la derecha de la palabra.

2.2. Gramáticas Formales. 2.2.1. Concepto de gramática formal

- Se llama gramática formal a la cuádrupla

$$\mathbf{G} = (\Sigma_T, \Sigma_N, \mathbf{S}, \mathbf{P})$$

- Σ_T , alfabeto de símbolos terminales
- Σ_N , alfabeto de símbolos no terminales
- $S \in \Sigma_N$, es el axioma o símbolo inicial
- P es un conjunto finito de reglas de producción de la forma

$$u ::= v, \text{ donde } u \in \Sigma^+ \text{ y } v \in \Sigma^*.$$

- Se verifica además que:

- $\Sigma_T \cap \Sigma_N = \emptyset$
- el alfabeto es $\Sigma = \Sigma_T \cup \Sigma_N$

2.2. Gramáticas Formales. 2.2.1. Concepto de gramática formal

- Ejemplo : consideremos la gramática

$$\Sigma_T = \{0, 1, 2\}$$

$$\Sigma_N = \{N, C\}$$

$$S = N$$

$$P = \{ N ::= NC, N ::= C, C ::= 0, C ::= 1, C ::= 2 \}$$

- Es posible establecer una notación simplificada para las reglas de producción. Si existen dos reglas de la forma

$$U ::= V$$

$$U ::= W$$

se pueden representar de la forma:

$$U ::= V \mid W$$

Esta forma de representar las reglas de producción recibe el nombre de “**forma normal de Backus**” (o BNF)

2.2. Gramáticas Formales. 2.2.1. Concepto de gramática formal

- Sea $G = (\Sigma_T, \Sigma_N, S, P)$. Una palabra $x \in \Sigma^*$ se denomina **forma sentencial** de G si se verifica que

$$S \rightarrow^* x$$

- Considerando la gramática anterior, las siguientes son formas sentenciales : NCC, NC2, 120

$$S = N \rightarrow NC \rightarrow NCC$$

$$S = N \rightarrow NC \rightarrow NCC \rightarrow NC2$$

$$S = N \rightarrow NC \rightarrow NCC \rightarrow CCC \rightarrow 1CC \rightarrow 12C \rightarrow 120$$

- Si una forma sentencial x cumple que $x \in \Sigma_T^*$ se dice que x es una **sentencia** o instrucción de G . Es decir, las sentencias estarán compuestas únicamente por símbolos terminales.
 - En el ejemplo anterior es sentencia: 120

2.2. Gramáticas Formales. 2.2.1. Concepto de gramática formal

- Sea una gramática $G = (\Sigma_T, \Sigma_N, S, P)$. Se llama lenguaje asociado a la G , o **lenguaje generado por G** , o lenguaje descrito por G , al conjunto :

$$L(G) = \{ x / S \rightarrow^* x \text{ and } x \in \Sigma_T^* \}$$

Conjunto de todas las sentencias de la gramática

- Ya que la teoría de gramáticas formales (Chomsky), junto con la notación BNF, proporciona una forma de describir lenguajes, esta simbología se considera como un metalenguaje (lenguaje para describir lenguajes).

2.2. Gramáticas Formales. 2.2.2. Tipos de Gramáticas. Jerarquía de Chomsky

- Chomsky clasificó las gramáticas en cuatro grandes grupos : G_0 , G_1 , G_2 y G_3 . Cada uno de estos grupos incluye las gramáticas del siguiente, de acuerdo con el siguiente esquema:

$$G_3 \subset G_2 \subset G_1 \subset G_0$$

1. Gramáticas tipo 0

- Las reglas de producción tienen la forma $u ::= v$ donde $u \in \Sigma^+$, $v \in \Sigma^*$, $u = xAy$ con $x \in \Sigma^*$, $y \in \Sigma^*$, $A \in \Sigma_N$ sin otra restricción
 - En las reglas de producción:
 - La parte izquierda no puede ser la palabra vacía.
 - En la parte izquierda (u) ha de aparecer algún símbolo no terminal.
- Los lenguajes representados por estas gramáticas reciben el nombre de **lenguajes sin restricciones**.

2.2. Gramáticas Formales. 2.2.2. Tipos de Gramáticas. Jerarquía de Chomsky

- Ya que v puede ser la palabra vacía, se sigue que en estas reglas podemos encontrar situaciones en que la parte derecha sea más corta que la izquierda. Las reglas en que ocurre esto se denominan **compresoras**. Una gramática que contenga al menos una regla compresora se denomina gramática compresora.
- En las gramáticas compresoras, las derivaciones pueden ser decrecientes, ya que la longitud de las palabras puede disminuir en cada uno de los pasos de derivación.
 - **Ejemplo:** sea $G = (\{a, b\}, \{A, B, C\}, A, P)$, donde P :
 $A ::= aABC \mid abC$
 $CB ::= BC$
 $bB ::= bb$
 $bC ::= b$
 - Veamos la derivación de la sentencia **aaabbb** :
 $A \rightarrow a(A)BC \rightarrow aa(A)BCBC \rightarrow aaab(CB)CBC \rightarrow aaa(bB)CCBC \rightarrow aaab(bC)CBC \rightarrow$
 $aaab(bC)BC \rightarrow aaab(bB)C \rightarrow aaab(bC) \rightarrow aaabbb$
 - Se observa que la gramática es compresora debido a la presencia de la regla $bC ::= b$.
 - Puede comprobarse que el lenguaje generado por esta gramática es $\{a^n b^n \mid n=1,2,\dots\}$

2. Gramáticas tipo 1

Las reglas de producción de esta gramática tienen la forma $\mathbf{xAy} ::= \mathbf{xvy}$ donde $x, y \in \Sigma^*$, $v \in \Sigma^+$ y A ha de ser un símbolo no terminal. (A puede transformarse en v sólo si aparece en el contexto definido por x e y)

- Ya que v no puede ser la palabra vacía, se deduce de aquí que este tipo de gramáticas no pueden tener reglas compresoras. Se admite una excepción en la regla $\mathbf{S} ::= \lambda$ (siendo S el axioma de la gramática). Como consecuencia se tiene que la palabra vacía pertenece al lenguaje generado por la gramática sólo si contiene esta regla.
- Los lenguajes generados por este tipo de gramáticas se denominan “**dependientes del contexto**”.
- Evidentemente todas las gramáticas de tipo 1 son también de tipo 0, y así, todos los lenguajes dependientes de contexto serán también lenguajes sin restricciones.
- Ejemplo : Demostrar que G es tipo 1: $G = (\{S, B, C\}, \{a, b, c\}, S, P)$, donde P es:
 $S ::= aSBc \mid aBC$
 $bB ::= bb$
 $bC ::= bc$
 $CB ::= BC$
 $cC ::= cc$
 $aB ::= ab$

2.2. Gramáticas Formales. 2.2.2. Tipos de Gramáticas. Jerarquía de Chomsky

- En principio, esta gramática es de tipo 0, no es dependiente del contexto por:
 - la regla **CB ::= BC**; formas de considerarla:
 - Considerando $x = \lambda$, $A = C$, $y = B$. Estaría formada la parte izquierda de la producción, pero la derecha será vB y sea cual sea v , no podrá ser BC
 - Considerando $x = C$, $A = B$, $y = \lambda$. Así tendríamos formada la parte izquierda de la regla, pero en la derecha tendríamos Cv , y sea v lo que sea no podremos obtener CB .
 - Ya no es posible hacer ninguna otra descomposición, por lo que esta regla no pertenece al esquema de reglas visto para las gramáticas de estructura de frases.
- Sin embargo la regla **CB ::= BC** puede descomponerse en las cuatro reglas siguientes, que permiten obtener las mismas derivaciones con más pasos, pero ajustándose a las condiciones exigidas para que la gramática sea dependiente del contexto .

$CB ::= XB$

$XB ::= XY$

$XY ::= BY$

$BY ::= BC$

- La gramática resultante, tendrá 3 reglas de producción más y dos símbolos adicionales (X , Y) en el alfabeto de símbolos no terminales.

2.2. Gramáticas Formales. 2.2.2. Tipos de Gramáticas. Jerarquía de Chomsky

3. Gramáticas tipo 2

- Las reglas de estas gramáticas se ajustan al siguiente esquema

A ::= v donde $v \in \Sigma^+$, y $A \in \Sigma_N$. Además podrán contener regla

S ::= λ

- Los lenguajes generados por este tipo de gramáticas se denominan **independientes de contexto**, ya que la conversión de A en v puede realizarse independientemente del contexto en que aparezca A.
- La mayor parte de los lenguajes de programación de ordenadores pueden describirse mediante gramáticas de este tipo.
- Ejemplo : sea la gramática $G = (\{a, b\}, \{S\}, S, \{ S ::= aSb \mid ab \})$.

Es una gramática de tipo 2. La derivación de la palabra aaabbb será:

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

Puede verse que el lenguaje definido por esta gramática es $\{a^n b^n \mid n=1, 2, \dots\}$

- Un mismo lenguaje puede generarse por muchas gramáticas diferentes. Sin embargo, una gramática determinada describe siempre un lenguaje único.

2.2. Gramáticas Formales. 2.2.2. Tipos de Gramáticas. Jerarquía de Chomsky

4. Gramáticas tipo 3. Estas gramáticas se clasifican en los dos grupos siguientes:

- **Gramáticas lineales por la izquierda**, cuyas reglas de producción pueden tener una de las formas siguientes:

$$A ::= a$$

$$A ::= Va$$

$$S ::= \lambda$$

donde $a \in \Sigma_T$, $A, V \in \Sigma_N$, y S es el axioma de la gramática.

- **Gramáticas lineales por la derecha**, cuyas reglas de producción tendrán la forma:

$$A ::= a$$

$$A ::= aV$$

$$S ::= \lambda$$

donde $a \in \Sigma_T$, $A, V \in \Sigma_N$, y S es el axioma de la gramática.

- Los lenguajes representados por este tipo de gramáticas se denominan **lenguajes regulares**.

- Ejemplos:

- $G1 = (\{0, 1\}, \{A, B\}, A, \{A ::= B1 \mid 1, B ::= A0\})$ Gramática lineal por la izquierda que describe el lenguaje $L1 = \{1, 101, 10101, \dots\} = \{1(01)^n \mid n = 0, 1, 2, \dots\}$

- $G2 = (\{0, 1\}, \{A, B\}, A, \{A ::= 1B \mid 1, B ::= 0A\})$ Gramática lineal derecha que genera el mismo lenguaje que la gramática anterior.

2.2. Gramáticas Formales. 2.2.3. Árboles de derivación

- A toda derivación de una gramática de tipo 1, 2 ó 3 le corresponde un árbol de derivación. Este árbol se construye así :
 - La **raíz** del árbol corresponde al **axioma** de la gramática
 - Una **derivación directa** se representa por un **conjunto de ramas** que salen de un **nodo** determinado. Al aplicar una regla, uno de los símbolos de la parte izquierda de la producción queda sustituido por la palabra de la parte derecha.
- A lo largo del proceso de construcción del árbol, los nodos finales de cada paso, leídos de izquierda a derecha, forman la forma sentencial obtenida por la derivación representada por el árbol.
- Será rama *terminal* aquella dirigida hacia un símbolo terminal. Este nodo se llama *hoja*. El conjunto de *hojas*, leído de izquierda a derecha, forma la sentencia generada por la derivación.

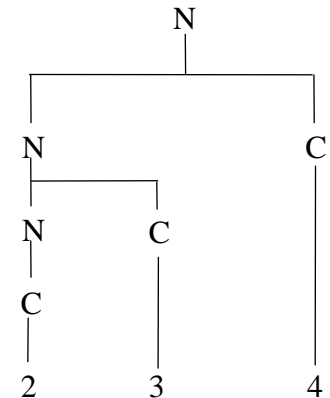
- Ejemplo : sea la gramática

$G = (\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}, \{N, C\}, N,$
 $\{N ::= C \mid NC, C ::= 0|1|2|3|4|5|6|7|8|9\})$

- Consideraremos la derivación :

$N \rightarrow NC \rightarrow NCC \rightarrow CCC \rightarrow 2CC \rightarrow 23C \rightarrow 234$

- El árbol de derivación correspondiente es:



2.2. Gramáticas Formales. 2.2.3. Árboles de derivación

□ Proceso inverso

- Para cada derivación existe un único árbol de derivación. Sin embargo, de una misma sentencia pueden obtenerse, a veces, varias derivaciones diferentes.
- Por ejemplo, el mismo árbol anterior puede aplicarse a las siguientes derivaciones :

$$N \rightarrow NC \rightarrow NCC \rightarrow CCC \rightarrow 2CC \rightarrow 23C \rightarrow 234$$
$$N \rightarrow NC \rightarrow NCC \rightarrow CCC \rightarrow C3C \rightarrow 23C \rightarrow 234$$
$$N \rightarrow NC \rightarrow NCC \rightarrow CCC \rightarrow 2CC \rightarrow 2C4 \rightarrow 234$$

2.2. Gramáticas Formales. 2.2.4. Ambigüedad

- Se dice que una sentencia es ambigua cuando para una misma sentencia podemos tener varios árboles de derivación diferentes.

(Como se vio en el ejemplo anterior, una misma sentencia puede obtenerse como resultado de varias derivaciones diferentes, pero a las que les corresponde un único árbol de derivación.)

- Se dice que una gramática es ambigua si tiene al menos una sentencia ambigua.
- Se dice que un lenguaje es ambiguo si existe una gramática ambigua que lo genera.

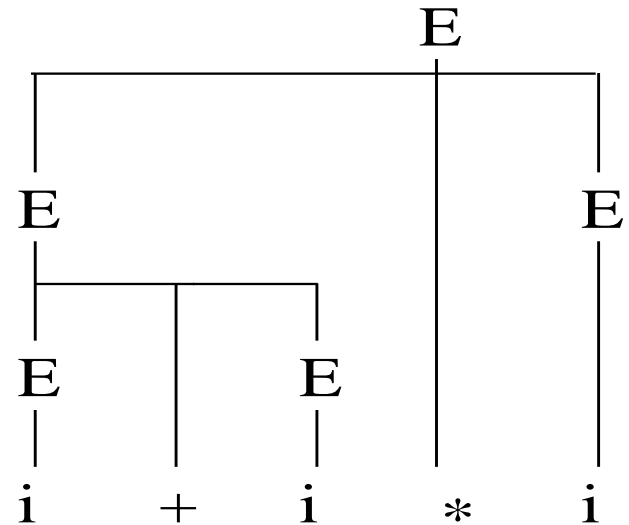
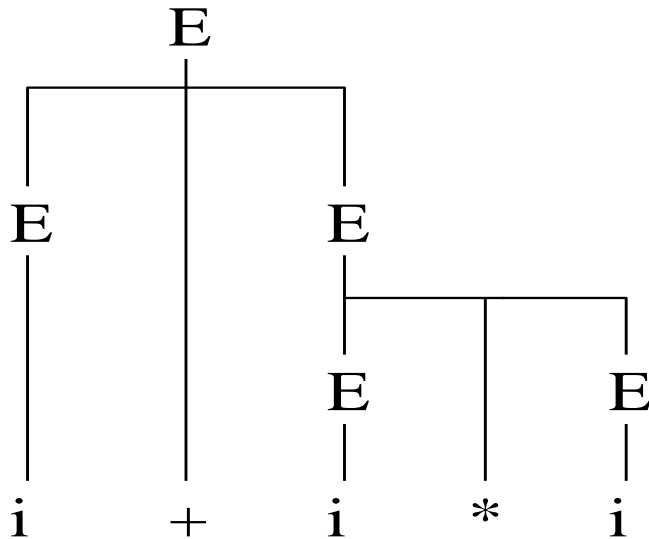
- **Ejemplo :**

- $G = (\{i, +, *, (,)\}, \{E\}, E, E ::= E + E \mid E * E \mid (E) \mid i)$

- Consideremos la sentencia $i + i * i$. Para esta sentencia podemos tener los siguientes árboles de derivación :

2.2. Gramáticas Formales. 2.2.4. Ambigüedad

- árboles de derivación para la sentencia $i + i * i$



- Esto no quiere decir que el lenguaje sea ambiguo, ya que se puede encontrar una gramática equivalente a la anterior, sin ser ambigua.
- Pero hay lenguajes para los cuales es imposible encontrar gramáticas no ambiguas. Estos lenguajes se denominan “inherentemente ambiguos”.

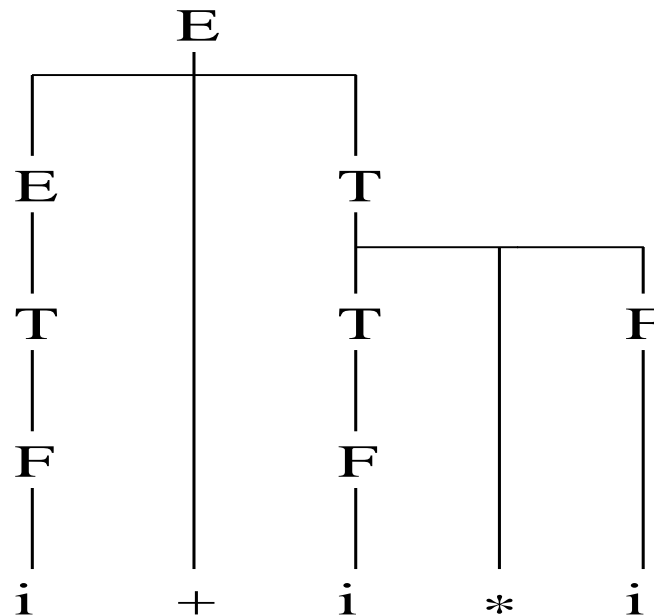
2.2. Gramáticas Formales. 2.2.4. Ambigüedad

- Ejemplo: gramática equivalente a la anterior sin ser ambigua:

$G = (\{i, +, *, (,)\}, \{E, T, F\}, E,$

$E ::= T \mid E + T, T ::= F \mid T * F, F ::= (E) \mid i \})$

- Existe un sólo árbol de derivación para la sentencia **$i+i*i$**



2.2. Gramáticas Formales. 2.2.5. Recursividad

- Una **gramática G** se llama **recursiva** en A, $A \in \Sigma_N$, si

$A \rightarrow +xAy$

- Si x es la palabra vacía, se dice que la gramática es recursiva a izquierdas

$A \rightarrow +Ay$

- Si y es la palabra vacía, se dice que la gramática es recursiva a derechas

$A \rightarrow +xA$

- Se dice que **una producción es recursiva** si

$A ::= xAy$

- La producción es recursiva a izquierdas si $x = \lambda$.

$A ::= Ay$

- Será recursiva a derechas si $y = \lambda$.

$A ::= xA$

- Si un lenguaje es infinito, la gramática que lo representa ha de ser recursiva.

2.2. Gramáticas Formales. 2.2.5. Recursividad

- ❑ Eliminación de la recursividad por la izquierda en producciones de un mismo símbolo no terminal:

\forall $A \in \Sigma_N$

Si $P1 = (A ::= A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_m)$ donde β_i no comienza por A

entonces //crear un símbolo nuevo A'

$\Sigma_N = \Sigma_N \cup \{A'\};$

$P = (P - P1) \cup \{A ::= \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_m A' ; A' ::= \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_n A' \mid \lambda\}$

fsi

\forall

2.2. Gramáticas Formales. 2.2.5. Recursividad

□ Eliminación de la recursividad por la izquierda en más de un paso

1. Disponer los Σ_N en algún orden A_1, A_2, \dots, A_n

2. Para $i:=1$ hasta n

Para $j:=1$ hasta n

Si $i \neq j$ entonces

 reemplazar cada producción $A_i ::= A_j \theta$ por:

$$A_i ::= \delta_1 \theta \mid \delta_2 \theta \mid \dots \mid \delta_k \theta$$

 donde $A_j ::= \delta_1 \mid \delta_2 \mid \dots \mid \delta_k$ son todas las reglas de A_j

fsi

 Eliminar la recursividad por la izquierda de las A_i

fpara

fpara

2.2. Gramáticas Formales. 2.2.5. Recursividad. Ejemplo.

$G = (\{i, +, *, (,)\}, \{E, T\}, E, P = \{E ::= T + E \mid T * E \mid i ; T ::= E \mid (E)\})$

1. $A_1 = E; A_2 = T$

2. Bucles:

□ $i=1 (A_1=E); j=1 (A_1=E)$. Se reemplazarían $E ::= E\alpha$. No hay.

□ $i=1 (A_1=E); j=2 (A_2=T)$. Se reemplazan $E ::= T\alpha$. El nuevo P' es

$E ::= E + E \mid E * E \mid (E) + E \mid (E) * E \mid i$

$T ::= E \mid (E)$

□ Se elimina la recursión en E quedando P'' :

$E ::= (E) + EE' \mid (E) * EE' \mid i E'$

$E' ::= +EE' \mid *EE' \mid \lambda$

$T ::= E \mid (E)$

□ $i=2 (A_2=T); j=1 (A_1=E)$. Se reemplazan $T ::= E\alpha$. El nuevo P''' es

$E ::= (E) + EE' \mid (E) * EE' \mid i E'$

$E' ::= +EE' \mid *EE' \mid \lambda$

$T ::= (E) + EE' \mid (E) * EE' \mid i E' \mid (E)$

□ Se eliminaría la recursión en T si la hubiera.

□ $i=2 (A_2=T); j=2 (A_2=T)$. Se reemplazarían $T ::= T\alpha$. No hay

□ El conjunto final de producciones es P'''

2.2. Gramáticas Formales. 2.2.6. Factorización a izquierdas

- Se puede factorizar a izquierdas cuando en una gramática aparecen producciones de un mismo símbolo no terminal en cuya parte derecha la primera parte es común.
- Algoritmo de factorización a izquierdas:

$\forall A \in \Sigma_N$

Si $A ::= \beta\alpha_1 \mid \beta\alpha_2 \mid \dots \mid \beta\alpha_n$

entonces //crear un símbolo nuevo A' y cambiar las producciones por:

$\Sigma_N = \Sigma_N \cup \{A'\};$

$A ::= \beta A'$

$A' ::= \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$

fsi

f \forall

- **Ejemplo:** $\{E ::= E+E \mid E^*E \mid i\}$. Se crea E' y quedaría:

$E ::= EE' \mid i$

$E' ::= +E \mid *E$