Gramáticas y lenguajes formales. La jerarquía de Chomsky.

Tema 4

Algorítmica y Modelos de Computación

Índice

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 remonta 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 Undecidible Propositions in Principia Mathematica and Relates Systems*". Este trabajo puede considerarse como la mayor realización matemática del siglo XX. El teorema de Gödel sostiene que cualquier teoría matemática contendrá afirmaciones "i**ndecidibles**".

  - 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".

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.

- 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.

- 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

- **Alfabeto:** conjunto finito, no vacío, de elementos que se denominan “letras” o “*símbolos*”.
- Los alfabetos se definen por la **enumeración** de los símbolos que contiene.
- Ejemplos:
  - A1 = {A, B, C, D, E, F, G, ..., Z} → el conjunto de todas las letras mayúsculas
  - A2 = {0,1} → alfabeto binario
  - A3 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} → alfabeto decimal
  - A4 = {(,)} → alfabeto formado por los paréntesis
  - El conjunto de todos los caracteres ASCII
- Convencionalmente se representa con la letra ∑

2.1 Lenguajes formales. 2.1.2. Palabra

- **Palabra** (o cadena): toda *secuencia finita de 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** (o cadena)**:** número de símbolos (letras) que la componen:

  | x | = 4 | y | = 4 | z | = 6
- **Palabra vacía:** aquella cuya <u>longitud es cero</u>. Se representa mediante la letra λ o ε $|\lambda| = 0$
- $\sum^{\mathbf{k}}$: conjunto de palabras (cadenas) de longitud k.
- **Clausura del alfabeto ($\sum^{*}$):** conjunto de todas las palabras $\sum^{*} = U\sum^{k} = \sum^{0} U \sum^{1} U \sum^{2} U \ldots$
- **"Universo del discurso" o "l*enguaje universal*"** sobre el alfabeto $\sum$: **W($\sum$)** conjunto (infinito) de todas las palabras (cadenas) que se pueden formar con las letras de un alfabeto. $W(\sum) = \sum^{*}$
- Ejemplo: un alfabeto con el menor número posible de letras (1).
  - A = { a }
  - En este caso, W(A) = { λ, a, aa, aaa, aaaa, ...}, y contiene un número infinito de elementos.
- La palabra vacía λ pertenece a todos los lenguajes universales de todos los alfabetos.

- **Concatenación** de palabras

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

  Supongamos que $x=A_0A_1..... A_i$ $|x| = i+1$; $y= B_0B_1..... B_j$ $|y| = j+1$

  Se llama concatenación de las palabras x e 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(∑) es otra palabra de W(∑). Si $x \in W(\sum)$ e $y \in W(\sum)$, entonces $xy \in W(\sum)$.
  - *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:** $xy \neq yx$

Potencia de una palabra

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

$$\mathbf{x^i = xxx...xx}$$

$$\underbrace{\text{.....}}_{i}$$

se cumplen las siguientes relaciones

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

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

Para que ambas relaciones se cumplan también para i, j = 0, basta con definir $\mathbf{x^0 = \lambda}$, 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 $x=A_0A_1...A_{n-1}A_n$, se denomina palabra refleja o inversa de x, representado por x^{-1}, a

$$\mathbf{x^{-1} = A_nA_{n-1}.....A_1A_0}$$

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

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

  $\mathbf{L \subset W(\Sigma)}$ $L \subset \Sigma^*$

- 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 (Σ) puede considerarse como un lenguaje: el formado por todas las posibles palabras de una letra.

- Los lenguajes **no se definen por enumeración** de las palabras que pertenecen a dicho lenguaje, **sino por las propiedades que cumplen las palabras** (cadenas) del lenguaje

- Ejemplos de lenguajes para el alfabeto binario Σ = { 0, 1 }
  - *Lenguaje de todas las palabras (cadenas) que constan de n ceros seguidos de n unos para cualquier n ≥ 0: { λ, 01, 0011, 000111, … }*
  - *Lenguaje del conjunto de palabras (cadenas) formadas por el mismo número de ceros que de unos: { λ, 01, 10, 0011, 0101, 1001, … }*
  - *Lenguaje del conjunto de números binarios cuyo valor es un número primo: { 10, 11, 101, 111, 1011, … }*

- Ejemplos de lenguajes para el alfabeto decimal Σ = { 0, 1, …, 8, 9 }
  - *Lenguaje de todas las palabras palíndromas (se leen igual hacia adelante que hacia atrás): { λ, 0, 1, …, 9, 00, …77, 010, 333, 14841, …, 256652, … }*
  - *Lenguaje del conjunto de números decimales cuyo valor es un número par: { 0, 2, 4, 6, 8, 10, 12, 14, …, 100, 102, ….1012, …}*

- **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:

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

- **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 $L \cup L = L$

Concatenación de lenguajes

Consideremos dos lenguajes definidos sobre el mismo alfabeto, L1 y L2. La concatenación o producto de estos lenguajes es el lenguaje

$$\mathbf{L1 \bullet L2 = \{ xy \,/\, x \in L1,\ y \in L2 \}}$$ (lo solemos denotar L1L2)

Las palabras de este lenguaje estarán formadas al concatenar cada una de las palabras del primer lenguaje con otra (todas) del segundo.

- La concatenación de lenguajes con el lenguaje vacío es:

$$\phi \mathbf{L} = \mathbf{L} \phi = \phi$$

Propiedades de esta operación:

- *Operación cerrada*. La concatenación de lenguajes sobre el mismo alfabeto es otro lenguaje sobre ese alfabeto.
- *Propiedad asociativa*. (L1 L2) L3 = L1 (L2 L3)
- *Elemento neutro*. Cualquiera que sea el lenguaje considerado, el lenguaje de la palabra vacía cumple que $\{\lambda\}L = L\{\lambda\} = L$

- **Potencia de un lenguaje**

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

$$L^i = \underbrace{LLL.....L}_{i}$$

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

- **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 $L^0 = \{\lambda\}$. 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

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

$$\Sigma^+ = \Sigma^* - \{\lambda\} = \Sigma^1 \cup \Sigma^2 \cup \ldots \Sigma^\infty$$

- **Cierre o Clausura de un lenguaje**

  Se define el cierre o clausura de un lenguaje L: $L^* = \bigcup_{i=0}^{\infty} L^i$

  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}

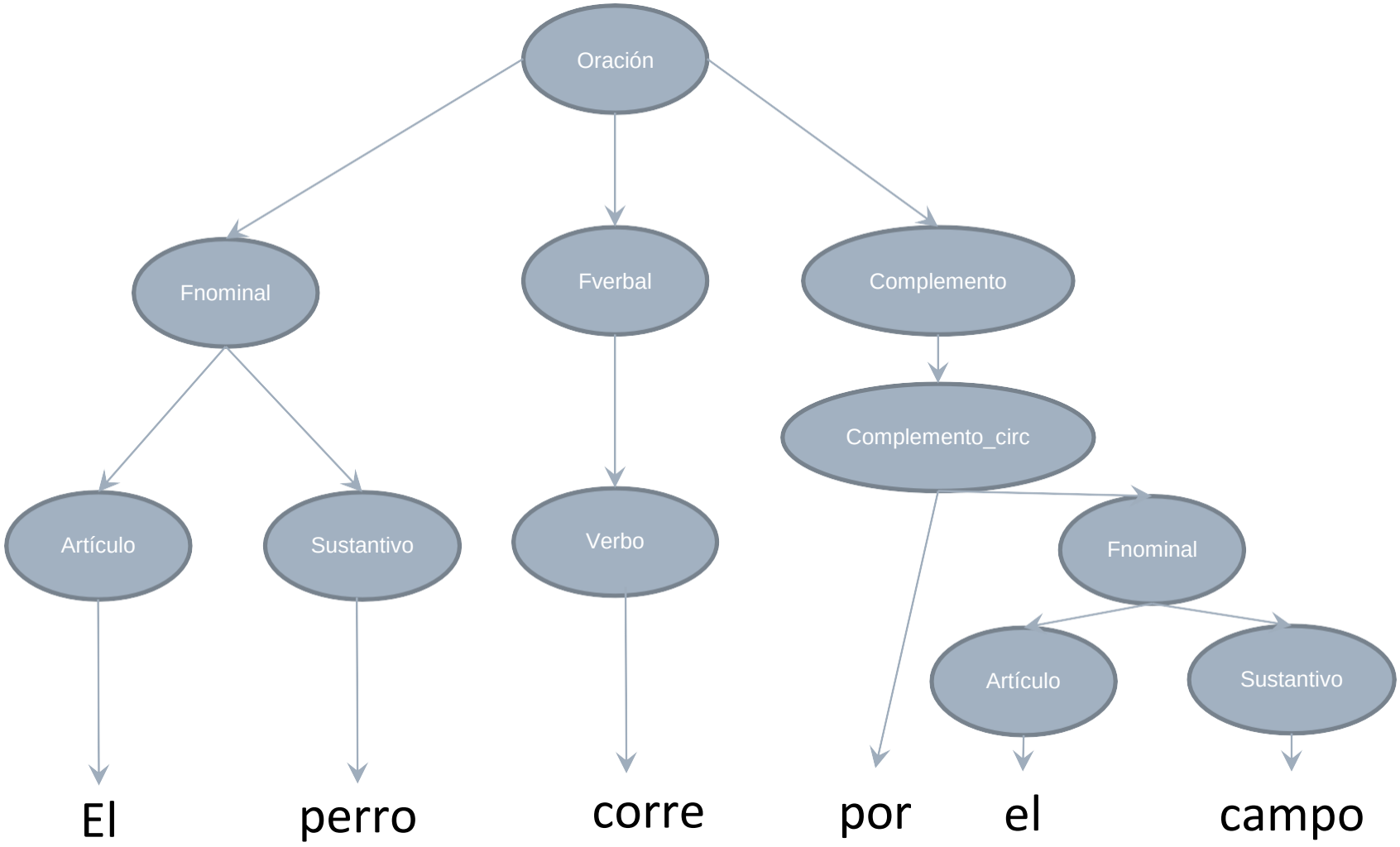
  $L^{-1} = \{ x^{-1} / x \in L \}$ lenguaje que contiene las palabras inversas a las palabras de L

Índice

- 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.
- Las gramáticas permiten, de una manera finita, definir el conjunto de palabras (cadenas) que constituyen un lenguaje.

Gramática del Castellano

Una gramática del castellano como diagrama sintáctico



Ejemplo: consideremos la instrucción **x = y+2*z** del lenguaje definido por:

- conjunto de **reglas** (producciones**)**:

  <instrucción> ::= <asignación>
  <asignación> ::= <identificador>”=”<expresión>
  <expresión> ::= <sumando>
  <expresión> ::= <sumando>”+”<expresión>
  <sumando> ::= <factor>
  <sumando> ::= <factor> “*” <sumando>
  <factor> ::= <identificador>
  <factor> ::= <número>

- Reglas morfológicas:

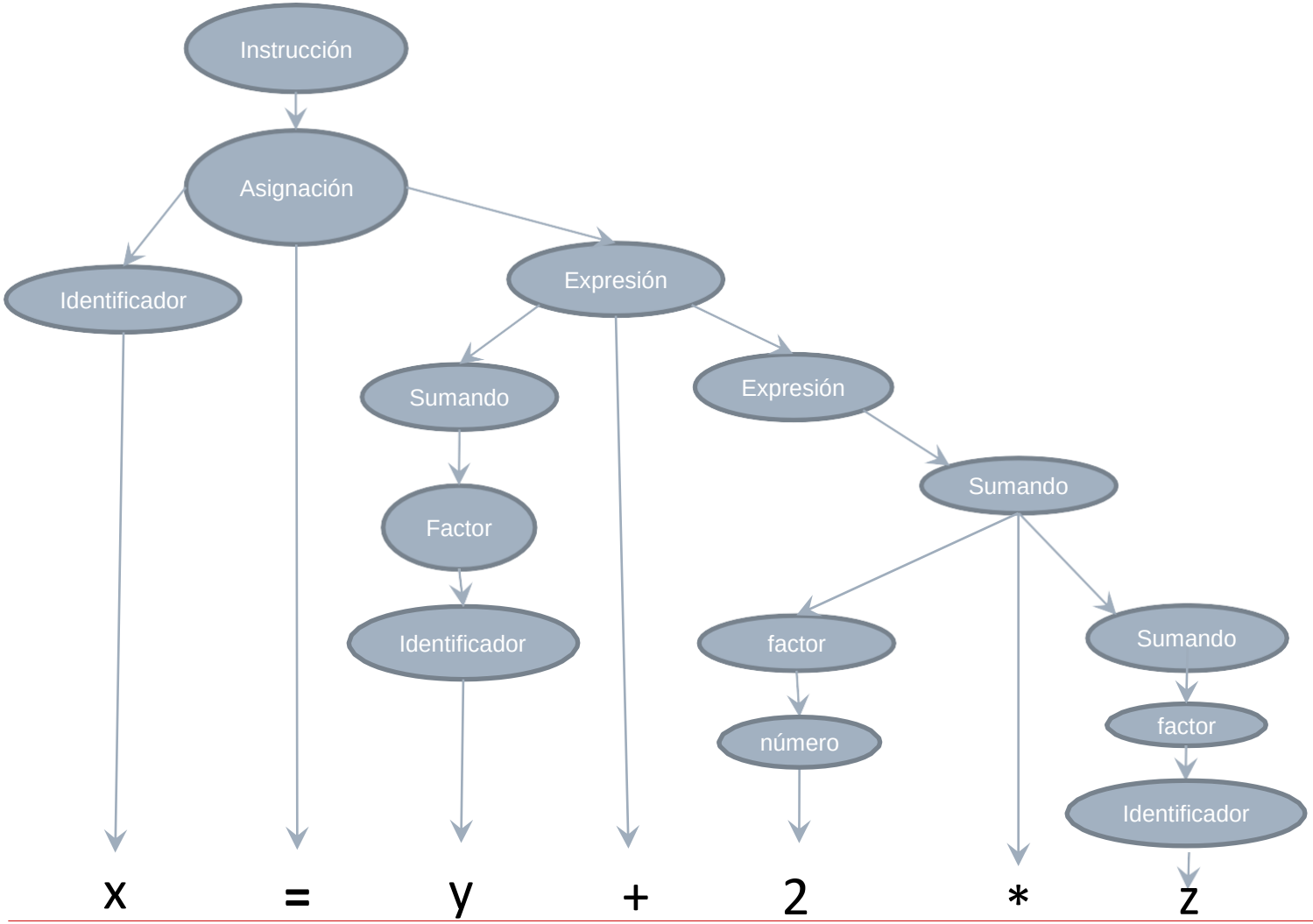
  <identificador> ::= “x”
  <identificador> ::= “y”
  <identificador> ::= “z”
  <número> ::= “2”

- Obtenemos la expresión **x = y+2*z** a partir de <instrucción> así:

<instrucción> → <asignación> → <identificador> = <expresión> → x = <expresión> →
x =<sumando>+<expresión> → x = <factor>+<expresión> → x = <identificador> + <expresión> →
x = y + <expresión> → x = y + <sumando> → x = y + <factor> * <sumando> →
x = y + <número> * <sumando> → x = y + 2 * <sumando> → x = y + 2 * <factor> →
x = y + 2 * <identificador> → 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

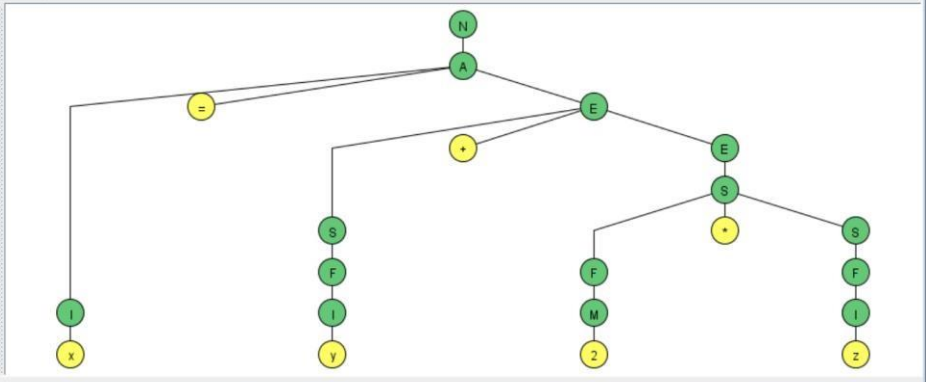
Input: x=y+2*z

String accepted! 294 nodes generated.

Input Field Text Size (For optimization, move one of the window size adjustors 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 adjustors 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
N→A	A
A→I=E	I=E
E→S+E	I=S+E
E→S	I=S+S
S→F*S	I=S+F*S
S→F	I=F+F*S
S→F	I=F+F*F
F→I	I=I+F*F
F→M	I=I+M*F
F→I	I=I+M*I
I→x	x=I+M*I
I→y	x=y+M*I
M→2	x=y+2*I
I→z	x=y+2*z

Derived z from I. Derivations complete.

- Para generar las palabras de una gramática se utilizan las **derivaciones.** Las palabras pueden ser generadas aplicando una única regla o múltiples reglas.
- **Producción o regla (x ::= y)**: es un par ordenado(x, y) con $x \in \Sigma^+$, $y \in \Sigma^*$

  si x forma parte de una palabra v, se puede sustituir x por y en v (permite transformar palabras en otras)
- **Derivación directa: v $\rightarrow$ w** (aplica una única producción/regla)

  aplicación de **una producción (regla)** a una palabra v para convertirla en otra w

  v $\rightarrow$ w si v=zxu , w=zyu (v, w, z, u $\in \Sigma^*$) y existe una producción x := y
  - Se cumple que para cada producción (regla) x::=y existe una derivación directa (haciendo z=u=λ): x $\rightarrow$ y
- **Derivación: v $\rightarrow$ *w** (aplica múltiples producciones/reglas)

  aplicación de una **secuencia de producciones** ($\rightarrow$ ***)** a una palabra v para convertirla en otra w.
- **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 (regla) aplicada a los símbolos más a la izquierda de la palabra.
- **Derivación más a la derecha**: Se utiliza en cada derivación directa la producción (regla) aplicada a los símbolos más a la derecha de la palabra.

- Se llama *gramática formal* a la cuádrupla

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

  - 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 $u ::= v$, donde $u \in \Sigma^+$ y $v \in \Sigma^*$. $\Sigma^+ = W(\Sigma) - \{\lambda\}$ $\Sigma^* = W(\Sigma)$
- Se verifica además que:
  - $\Sigma_T \cap \Sigma_N = \phi$ (símbolo terminal ≠ símbolo no terminal)
  - el alfabeto es $\Sigma = \Sigma_T \cup \Sigma_N$
- La gramática formal permite definir lenguajes formales

- Ejemplo: consideremos la gramática $G = (\Sigma_T, \Sigma_N, S, P)$:

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

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

  $S = N$

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

- 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)

  $P = \{ N::=NC \mid C,\ C::=0 \mid 1 \mid 2 \}$ (notación BNF)

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

  $$\mathbf{S \rightarrow^* x}$$

  a partir del símbolo inicial S podemos llegar a la palabra x aplicando una secuencia de producciones ($\rightarrow^*$)

  - 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 sólo es sentencia: 120

- **Lenguaje generado por una gramática** G = (Σ_T, Σ_N, S, P)

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

Conjunto de todas las sentencias (palabras de símbolos terminales) de la gramática

todas las palabras x compuesta por símbolos terminales que se pueden generar aplicando una secuencia de producciones (derivación) a partir del símbolo inicial S

- Al conjunto L(G) también se le llama lenguaje asociado a G, o lenguaje descrito por G.

- 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).

- Chomsky clasificó las gramáticas en cuatro grandes grupos: G0, G1, G2 y G3. Cada uno de estos grupos incluye las gramáticas del siguiente, de acuerdo con el siguiente esquema:

$$\mathbf{G3 \subset G2 \subset G1 \subset G0}$$

- Jerarquía de Chomsky:

Gramática	Tipo de Lenguaje	Máquina / Dispositivo reconocedor
G0	Sin restricciones o con estructura de frase	Máquina de Turing
G1	Sensible al contexto	Autómata linealmente acotado
G2	Libre de contexto	Autómata con pila
G3	Regular	Autómata Finito

1. Gramáticas tipo 0 (o con estructura de frase)

- Son las que no presentan restricciones en cuanto a las reglas
- Las reglas de producción tienen la forma **xAy ::= v**

  donde **u = xAy** $\in \Sigma^+$, **v** $\in \Sigma^*$, con **x** $\in \Sigma^*$, **y** $\in \Sigma^*$, **A** $\in \Sigma_N$ sin otra restricción

  - En las reglas de producción **u := v**:
    - La parte izquierda no puede ser la palabra vacía. $u \in \Sigma^+$, $\Sigma^+ = W(\Sigma) - \{\lambda\}$
    - En la parte izquierda (u) ha de aparecer algún símbolo no terminal. $u = xAy$ $A \in \Sigma_N$
- Los lenguajes representados por estas gramáticas reciben el nombre de **lenguajes sin restricciones**.

- 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.

  - **Ejemplo1**: sea G = ({a, b}, {A, B, C}, A, P), donde P: G = (Σ_T, Σ_N, S, P)

    A ::= aABC | abC (reglas de producción P en notación BNF)

    CB ::= BC

    bB ::= bb

    bC ::= b

  - Veamos la derivación de la sentencia **aaabbb**:

    $A \rightarrow aABC \rightarrow aaABCBC \rightarrow aaabCBCBC \rightarrow aaabBCCBC \rightarrow aaabbCCBC \rightarrow aaabbCBC \rightarrow aaabbBC \rightarrow aaabbbC \rightarrow aaabbb$

  - 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

    $L(G) = \{ a^n b^n \ / \ n=1, 2,.. \}$ palabras de aes seguidas por el mismo número de bes

- Las gramáticas tipo 0 no tienen por qué ser compresoras

  - **Ejemplo2**: sea G = ({a, b}, {A, B, C}, A, P), donde P: G = (Σ_T, Σ_N, S, P)

    A ::= aABC | aBC (reglas de producción P en notación BNF)
    CB ::= BC
    aB ::= ab, bB ::= bb
    bC ::= bc, cC := cc

  - Ejemplo de derivación:

    $A \rightarrow aABC \rightarrow aaABCBC \rightarrow aaaBCBCBC \rightarrow aaaBBCCBC \rightarrow aaaBBCBCC \rightarrow aaaBBBCCC \rightarrow aaabBBCCC \rightarrow aaabbBCCC \rightarrow aaabbbCCC \rightarrow aaabbbcCC \rightarrow aaabbbccC \rightarrow aaabbbccc$

  - Puede comprobarse que el lenguaje generado por esta gramática es

    $L(G) = \{ a^n b^n c^n / n \geq 1 \}$ palabras de aes seguidas por el mismo número de bes y ces

2. Gramáticas tipo 1 (sensibles al contexto)

- Las reglas de producción de esta gramática tienen la forma **xAy ::= xvy** donde x, y $\in \Sigma^*$, v $\in \Sigma^+$ y A ha de ser un símbolo no terminal ($A \in \Sigma_N$).
- Ya que $v \in \Sigma^+$ no puede ser la palabra vacía, este tipo de gramáticas no pueden tener reglas compresoras. Se admite como excepción la regla **S ::=** λ siendo S el axioma de la gramática. (la palabra vacía pertenece al lenguaje generado por la gramática sólo si contiene esta regla)
- **Definición alternativa**. Las gramáticas tipo 1 son gramáticas crecientes: aquellas en las que el número de símbolos en la parte derecha de una regla es siempre mayor igual que el de la parte izquierda (A ::= B / |A| ≤ |B|).
- Los lenguajes generados por este tipo de gramáticas se denominan "**dependientes del contexto**", ya que la conversión de A en v sólo es posible si se encuentra entre x e y (las cadenas x y representan el contexto en el que se puede aplicar la sustitución de A por v)
- Todas las gramáticas tipo 1 son también de tipo 0, y así, todos los lenguajes dependientes de contexto serán también lenguajes sin restricciones o con estructura de frase.
- Ejemplo : Demostrar que G es tipo 1: G = ({a, b, c}, {S, B, C}, S, P), donde P es: $G = (\Sigma_T, \Sigma_N, S, P)$

  S := aSBc | aBC (reglas de producción P en notación BNF)
  bB := bb
  bC ::= bc
  CB ::= BC
  cC ::= cc
  aB ::= ab

- En principio, esta gramática es de tipo 0 (no es dependiente del contexto) por:
  - la regla **CB ::= BC**; formas de considerarla:
    - Considerando *x* = λ, A = C, *y* = B. Estaría formada la parte izquierda de la producción, pero la derecha será *v*B y sea cual sea *v*, no podrá ser BC
      xAy ::= xvy → λCB ::= λvB → CB ::= vB (es imposible que vB sea BC)
    - Considerando *x* = C, A = B, *y* = λ tenemos formada la parte izquierda de laregla, pero en la derecha tendríamos C*v*, y sea *v* lo que sea no podremos obtener BC
      xAy ::= xvy → CBλ ::= Cvλ → CB ::= Cv (es imposible que Cv sea BC)
    - 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 xAy ::= xvy → λCB ::= λXB → CB ::= XB

  XB ::= XY xAy ::= xvy → XBλ ::= XYλ → XB ::= XY

  XY ::= BY xAy ::= xvy → λXY ::= λBY → XY ::= BY

  BY ::= BC xAy ::= xvy → BYλ ::= BCλ → BY ::= BC

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

3. Gramáticas tipo 2 (libres de contexto)

- Son aquellas cuyas reglas sólo tienen un único símbolo no terminal en la parte izquierda:

  A ::= v donde v $\in \Sigma^+$ (v no puede ser la palabra vacía) y A $\in \Sigma_N$.

  Además podrán contener la 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.**

- Ejemplo1: sea la gramática G = ({a, b}, {S}, S, { S ::= aSb | ab}). $G = (\Sigma_T, \Sigma_N, S, P)$

  La derivación de la palabra aaabbb será: $S \rightarrow aSb \rightarrow aaSbb \rightarrow aaabbb$

  Puede verse que el lenguaje definido por esta gramática es $\{a^n b^n / n \geq 1\}$

- Ejemplo2: sea la gramática G = ({a, b}, {S}, S, { S ::= aSb | λ}). $G = (\Sigma_T, \Sigma_N, S, P)$

  Ejemplo de derivación: $S \rightarrow aSb \rightarrow aaSbb \rightarrow aaaSbbb \rightarrow aaabbb$

  Esta gramática reconoce el lenguaje $L(G) = \{a^n b^n / n \geq 0\}$

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

4. Gramáticas tipo 3 (regulares).:

- Son aquellas cuyas reglas contienen un único símbolo no terminal en la parte izquierda y en su parte derecha un símbolo terminal o un símbolo terminal seguido/precedido de un no terminal.

- 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 ::= λ

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

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

  A ::= a

  A ::= aV

  S ::= λ

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

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

- Ejemplo1: $G = (\Sigma_T, \Sigma_N, S, P)$
  - G1 = ({ 0, 1}, {A, B}, A, { A ::= B1 | 1, B ::= A0}) Gramática lineal por la izquierda que describe el lenguaje L1 = { 1, 101, 10101, ... } = $\{1(01)^n / n = 0, 1, 2, ...\}$
  - G2 = ({ 0, 1}, {A, B}, A, { A ::= 1B | 1, B ::= 0A}) Gramática lineal por la derecha que genera el mismo lenguaje que la gramática anterior.

- Ejemplo2: $G = (\Sigma_T, \Sigma_N, S, P)$
  - G1 = ({ 0, 1}, {S, A, B, C}, S,

    { S ::= 0A | 1B, A ::= 0S | 1C, B ::= 1S | 0C, C ::= 0B | 1A, A ::= 0, B := 1 })
  - Ejemplos de derivación: $S \rightarrow 0A \rightarrow 01C \rightarrow 010B \rightarrow 0101$

    $S \rightarrow 1B \rightarrow 11S \rightarrow 111B \rightarrow 1110C \rightarrow 11101A \rightarrow 111010$
  - Esta gramática reconoce un lenguaje formado por palabras con un número par de 0s y de 1s.

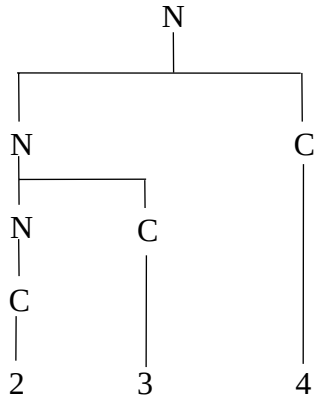
- Las gramáticas regulares tienen la misma capacidad expresiva que las expresiones regulares y pueden ser reconocidas por Autómatas Finitos

- 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** (símbolo inicial) de la gramática
  - Las hojas corresponden a símbolos terminales y los nodos a no terminales
  - **Cada rama** que sale de un **nodo** determinado describe una **derivación directa**. Al aplicar una regla, uno de los símbolos de la parte izquierda de la regla 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** (palabra compuesta por símbolos terminales y no terminales) obtenida por la derivación representada por el árbol.
- El conjunto de *hojas*, leído de izquierda a derecha, forma la **sentencia** (palabra formada por símbolos terminales) generada por la derivación.
- Ejemplo: sea la gramática $G = (\Sigma_T, \Sigma_N, S, P)$

  G = ({0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0}, {N,C}, N,

  {N ::= C | 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:



Proceso inverso

- Para cada derivación existe un único árbol de derivación. Sin embargo, de una misma **sentencia** (palabra *x* / **x** $\in \Sigma_T^*$) pueden obtenerse, a veces, varias derivaciones diferentes.

- Por ejemplo, el mismo árbol anterior puede aplicarse a las siguientes derivaciones:

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

$N \rightarrow NC \rightarrow NCC \rightarrow CCC \rightarrow C3C \rightarrow 23C \rightarrow 234$

$N \rightarrow NC \rightarrow NCC \rightarrow CCC \rightarrow 2CC \rightarrow 2C4 \rightarrow 234$

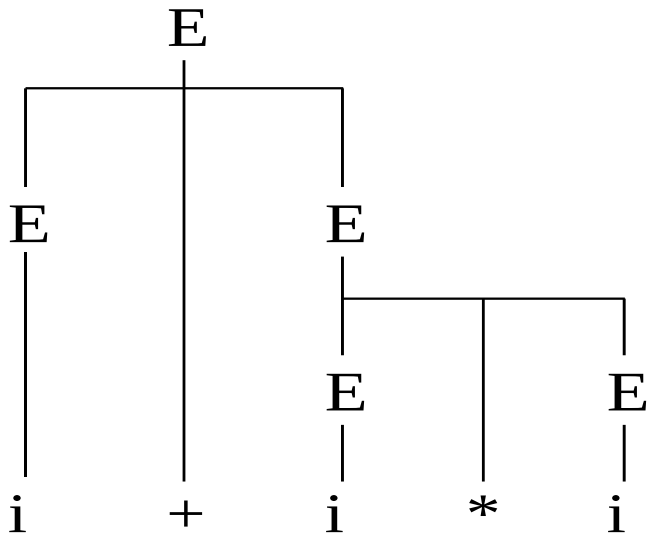
- Se dice que una **sentencia** es ambigüa cuando para una misma sentencia podemos tener varios árboles de derivación diferentes

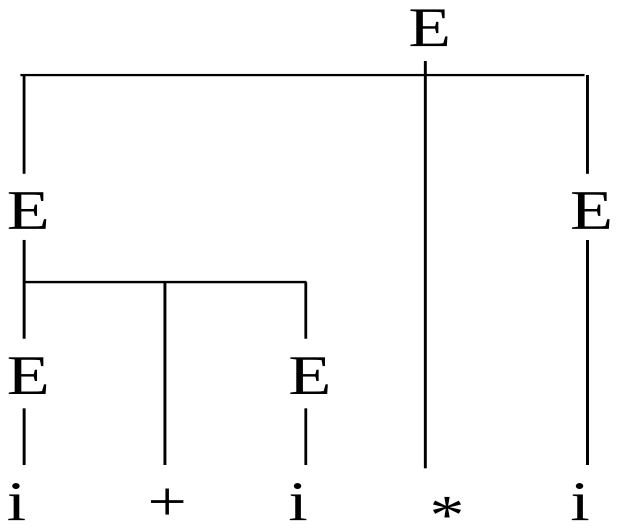
  (Como se vio en el ejemplo anterior, una misma **sentencia** puede obtenerse como resultado de varias derivaciones diferentes, pero a las que les corresponde un único árbol de derivación.)

- Una gramática es ambigua si tiene al menos una **sentencia** ambigua.
- Un lenguaje es ambiguo si existe una gramática ambigua que lo genera.
- Demostrar que una gramática no es ambigua es muy difícil.
- Existen construcciones gramaticales que producen ambigüedad.
- Si se usan estas construcciones es fácil demostrar que una gramática es ambigua

- Construcciones que producen ambigüedad:
  - Reglas de la forma
    E ::= E…..E
  - Gramáticas con ciclos
    S ::= A | a
    A ::= S
  - Conjuntos de reglas con caminos alternativos entre dos puntos
    S ::= A | B
    A ::= B
  - Producciones recursivas en las que la regla no recursiva pueda producir la cadena vacía
    S ::= HRS | s
    H ::= h | λ
    R ::= r | λ
  - Símbolos no terminales que puedan producir la misma cadena y la cadena vacía, si aparecen juntos
    S ::= HR
    H ::= h | λ
    R ::= h | r | λ

- **Ejemplo** : G = (Σ_T, Σ_N, S, P)
  - **G = ({ i, +, *, (,) }, { E }, E, { E ::= E + E | E * E | (E) | i })**
  - Consideremos la **sentencia i + i * i**. Para esta **sentencia** podemos tener los siguientes árboles de derivación → gramática es ambigua



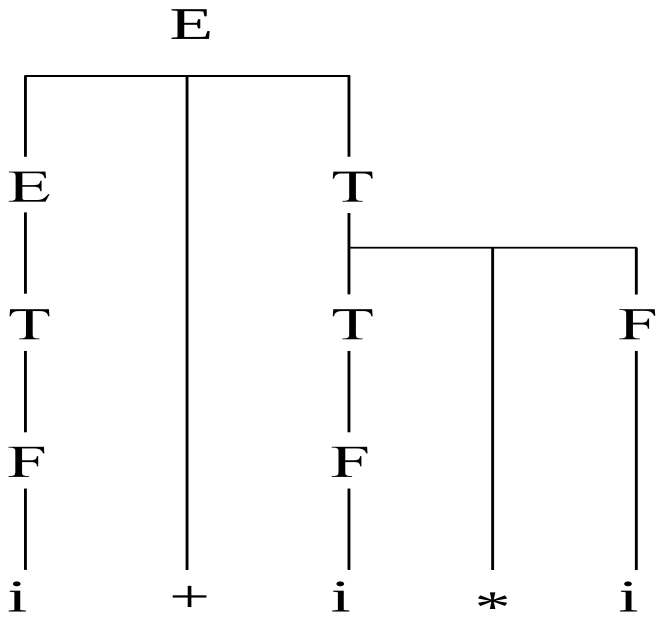


- Esto no quiere decir que el lenguaje sea ambiguo, ya que se puede encontrar una gramática equivalente a la anterior, sin ser ambigua.
- Pero hay lenguajes para los cuales es imposible encontrar gramáticas no ambiguas. Estos lenguajes se denominan **"inherentemente ambiguos"**.

- Ejemplo: gramática equivalente a la anterior sin ser ambigüa: $G = (\Sigma_T, \Sigma_N, S, P)$

  G = ({ i, +, *, (,) }, { E, T, F }, E,
  { E ::= T | E + T, T ::= F | T * F, F ::= (E) | i })

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



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

  $A \rightarrow +xAy$ a partir de un símbolo no terminal podemos obtener una forma sentencial que incluya el mismo símbolo no terminal

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

    $A \rightarrow +Ay$ en alguna recursividad, el símbolo no terminal aparece a la izquierda de la forma sentencial

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

    $A \rightarrow +xA$ en alguna recursividad, el símbolo no terminal aparece a la derecha de la forma sentencial

- Se dice que **una producción (regla) es recursiva** si

  A ::= xAy el símbolo no terminal de la parte izquierda, aparece también en la parte derecha

  - La producción (regla) es recursiva a izquierdas si x = λ (palabra vacía).

    A ::= Ay el símbolo no terminal de la parte izquierda, aparece en primer lugar en la parte derecha

  - Será recursiva a derechas si y =λ (palabra vacía).

    A ::= xA el símbolo no terminal de la parte izquierda, aparece en último lugar en la parte derecha

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

- Eliminación de la recursividad por la izquierda en producciones (reglas) de un mismo símbolo no terminal:

$$A ::= A\alpha_i \; i \le n \mid \beta_j \; j \le m \rightarrow \begin{array}{l} A ::= \beta_j A' \; j \le m \\ A' ::= \alpha_i A' \; i \le n \mid \lambda \end{array}$$

- Algoritmo:

$\forall A \in \Sigma_N$

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

entonces //crear un símbolo nuevo A'

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

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

fsi

f∀

para todo símbolo no terminal A, cualquier producción (regla) Pi recursiva por la izquierda que la use, puede eliminarse creando un nuevo símbolo no terminal A' y sustituyendo la producción (regla) Pi por las dos producciones (reglas) indicadas en la fórmula

- Eliminación de la recursividad por la izquierda en producciones (reglas) de un mismo símbolo no terminal:

$$A ::= A\alpha_i \; i \le n \mid \beta_j \; j \le m \quad \rightarrow \quad \begin{array}{l} A ::= \beta_j A' \; j \le m \\ A' ::= \alpha_i A' \; i \le n \mid \lambda \end{array}$$

- Ejemplo: Eliminar la recursividad por la izquierda de la gramática

G =({ num }, { E,T, F }, E, P={ E::=E+T | E-T | T | F; T::=T*F | T/F | F; F::=num | (E) })

$$\begin{array}{l} E ::= E+T \mid E-T \mid T \mid F \\ T ::= T*F \mid T/F \mid F \\ F ::= num \mid (E) \end{array} \quad \rightarrow \quad \begin{array}{l} E ::= TE' \mid FE' \\ E' ::= +TE' \mid -TE' \mid \lambda \\ T ::= FT' \\ T' ::= *FT' \mid /FT' \mid \lambda \\ F ::= num \mid (E) \end{array}$$

G =({ num }, { E,T, F, E', T' }, E,
P={ E::=TE' | FE'; E'::=+TE' | -TE' | λ; T::=FT'; T'::=*FT' | /FT' | λ; F::=num | (E) })

- Eliminación de la recursividad por la izquierda en más de un paso
  reglas donde la recursividad por la izquierda es indirecta, como por ejemplo:

  $A ::= B\alpha_i \mid \beta_j$; $B::= A \mid \beta_k$ A es indirectamente recursiva por la izquierda a través de B

- Algoritmo:

1. Disponer los Σ_N en algún orden $A_1, A_2, \ldots A_n$
2. Para i:=1 hasta n
   Para j:=1 hasta n
   Si i $\neq$ j entonces
   reemplazar cada regla $A_i ::= A_j \alpha$ por:
   $A_i ::= \delta_1 \alpha \mid \delta_2 \alpha \mid \ldots\ldots \delta_k \alpha$
   donde $A_j ::= \delta_1 \mid \delta_2 \mid \ldots\ldots \delta_k$ son todas las reglas de A_j
   fsi
   eliminar la recursividad por la izquierda de las A_i
   fpara
   fpara

G =({ i,+,*,(,) }, { E,T }, E, P = { E ::=T+E | T*E | i ; T::=E|(E) }) G = (Σ_T, Σ_N, S, P)

1. A_1=E; A_2=T
2. Bucles:
   - i=1 (A_1=E); j=1 (A_1=E). Eliminar recursividad izquierda de A_1=E (E::=Eα. No hay).
   - i=1 (A_1=E); j=2 (A_2=T). Se reemplaza cada E::=Tα. El nuevo P' es

     E ::= E+E | E*E | (E)+E | (E)*E | i

     T::= E | (E)
     - eliminar la recursión izquierda de E ::= E+E | E*E | (E)+E | (E)*E | i quedando P'':

       E ::= (E)+EE' | (E)*EE' | i E'

       E' ::= +EE' | *EE' | λ

       T::= E | (E)
   - i=2 (A_2=T); j=1 (A_1=E). Se reemplaza cada T::=Eα. El nuevo P''' es

     E ::= (E)+EE' | (E)*EE' | i E'

     E' ::= +EE' | *EE' | λ

     T::= (E)+EE' | (E)*EE' | i E' | (E)
     - eliminar la recursión izquierda de T::= (E)+EE' | (E)*EE' | i E' | (E) (no la hay).
   - i=2 (A_2=T); j=2 (A_2=T). Eliminar recursividad izquierda de A_2=T (T::=Tα. No hay)
- El conjunto final de producciones es P'''

- Factorización a izquierdas: reglas de un mismo símbolo no terminal (A) cuya parte derecha contiene una primera parte (β) común a dichas reglas.

$A::=\beta\alpha_i \quad i \le n$	→	$A::= \beta A'$ $A'::= \alpha_i \quad i \le n$

- Algoritmo de factorización a izquierdas:

$\forall A \in \Sigma_N$

Si $A::=\beta\alpha_1 | \beta\alpha_2 | \ldots..|\beta\alpha_n$

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

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

$A::= \beta A'$

$A'::= \alpha_1 | \alpha_2 | \ldots | \alpha_n$

fsi

f$\forall$

- **Ejemplo**: {E ::=E+E | E*E | i }. Se crea E' y quedaría:

E ::= EE' | i

E'::= +E| *E