# Teoría de la Computación y Lenguajes Formales

## Gramáticas Generativas
## Gramáticas Libres de Contexto (GLC)

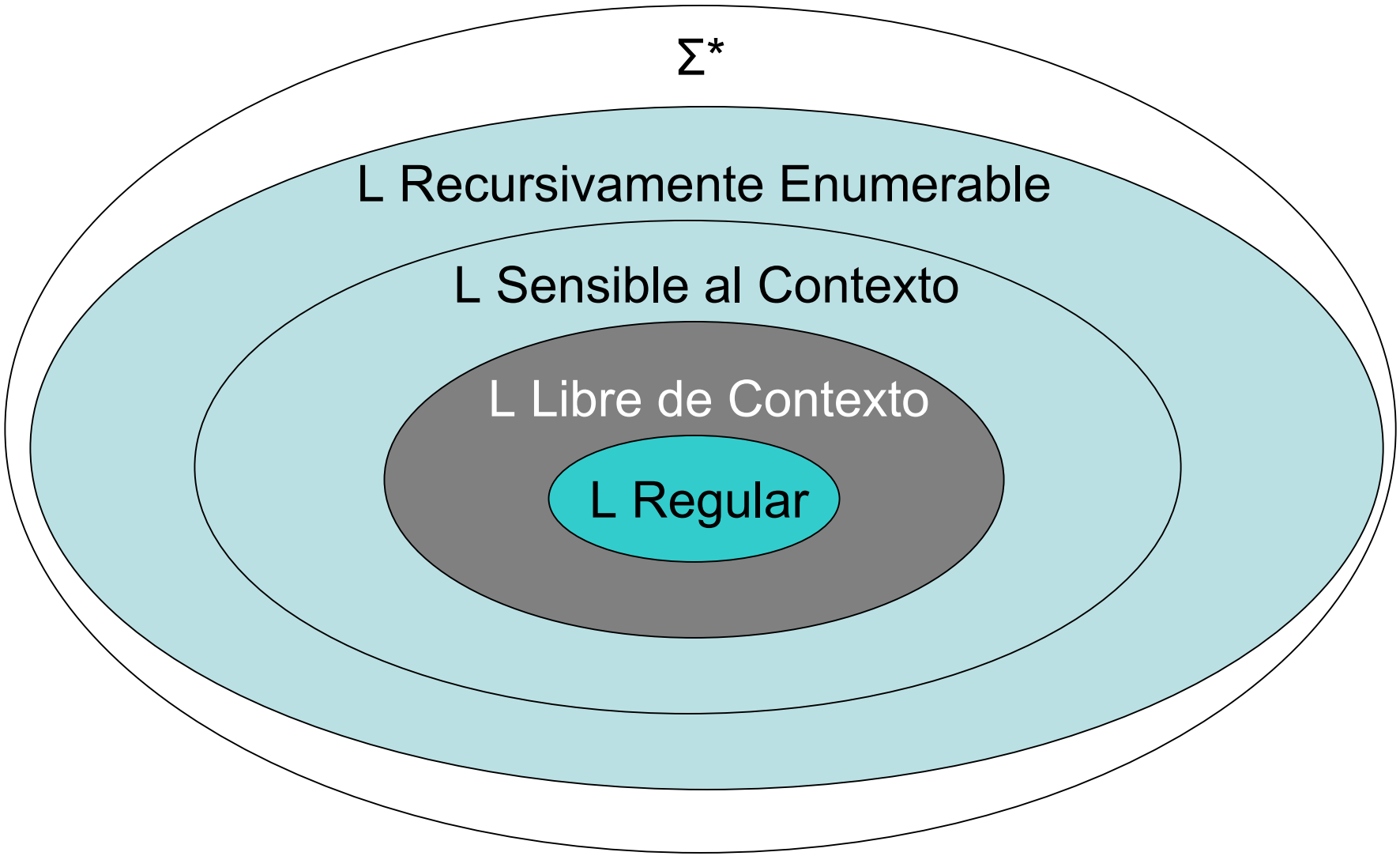
Prof. Hilda Y. Contreras
Departamento de Computación
hyelitza@ula.ve
hildac.teoriadelacomputacion@gmail.com

UNIVERSIDAD DE LOS ANDES

# Contenido

- Gramática Libre de Contexto
- Árbol de derivación.
- Ambigüedad.
- Algoritmos de Simplificación de Gramáticas:
  - Eliminación de Producciones Nulas.
  - Eliminación de Producciones Unitarias.
  - Eliminación de Símbolos y Producciones Inútiles.
- Formas Normales:
  - Forma Normal de Chomsky.
  - Forma Normal de Greibach.
- Parsing
- Ejemplos

# Lenguaje Libre de Contexto



# Lenguaje Libre de Contexto y GLC

- Jerarquía de Chomsky (Lenguaje Regular - Tipo 3)

| Tipo | Lenguaje | Máquina | Gramática |
|---|---|---|---|
| 0 | Recursivamente enumerable | Máquina de Turing | Sin restricciones |
| 1 | Dependiente del Contexto | Autómata linealmente acotado | *Gramática dependiente del contexto*<br>$\alpha A \beta \rightarrow \alpha \gamma \beta$ |
| **2** | **Independiente del Contexto** | **Autómata de Pila** | ***Gramática independiente de contexto***<br>$A \rightarrow \gamma$, $\gamma$ en $(V \cup T)^*$ |
| 3 | Lenguaje Regular | Autómata finito | *Gramática Regular*<br>$A \rightarrow aB$<br>$A \rightarrow a$ |

# Gramática

- Gramática expresa la competencia lingüística. Modelar el comportamiento lingüístico en el proceso de comunicación (comprensión y generación de la lengua)

- Gramática: Especificación rigurosa y explicita de la estructura de un lenguaje (No ambigua, predictiva, aplicativa).

# Gramática Generativa

- En 1957 aparece la denominada “Gramática Generativa” tras la publicación de la obra de Noam Chomsky “Estructuras Sintácticas”.

- En este libro Chomsky expone que una gramática de constituyentes inmediatos **no** es totalmente válida para explicar el mecanismo mediante el cual los hablantes de una lengua son capaces de **producir** y **entender** oraciones.

# Gramáticas Generativas

Una gramática generativa es una cuádrupla **(V, T, P, S)** en la que:

- **V** es un conjunto de variables o símbolos no terminales. Sus elementos se suelen representar con letras mayúsculas.
- **T** es el alfabeto, conjunto de símbolos terminales. Sus elementos se suelen representar con letras minúsculas.
- **P** es un conjunto de pares (α, β), llamados reglas de producción, donde α, β en $(V \cup T)^*$ y α contiene, al menos un símbolo de V. El par (α, β) se suele representar como $\alpha \rightarrow \beta$
- **S** es un elemento de V, llamado símbolo de partida o inicial.

# Paso de Derivación

Dada la Gramática G = (V, T, P, S) y dos palabras α, β en (V U T)* β es derivable a partir de α en un paso (α => β) si y solo si existen palabras $\delta_1$, $\delta_2$ en (V U T)* y una producción $\gamma \rightarrow \eta$ tales que:

$\alpha = \delta_1 \gamma \delta_2$ y $\beta = \delta_1 \eta \delta_2$

la derivación sería:

$\delta_1 \gamma \delta_2 \Rightarrow \delta_1 \eta \delta_2$

# Secuencia de Derivación

Sucesión [1 a ∞] de pasos de derivación

β es derivable de α (α =>* β) si y solo si existe una sucesión de palabras

$\gamma_1, \ldots, \gamma_n$ ( n ≥ 1) tales que:

$\alpha \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \ldots \Rightarrow \gamma_n \Rightarrow \beta$

# Lenguaje Generado

Se define como lenguaje generado por una gramática G = (V, T, P, S) al conjunto de cadenas formadas por símbolos terminales en V y que son derivables a partir del símbolo de partida S.

Es decir,

**L(G) = { w | w en T* y S =>* w }**

# Gramática Libre de Contexto

Una Gramática Libre de contexto es una Gramática Generativa G = (V,T,P,S)

Donde el conjunto P de producciones tiene la forma:

**A -> λ**

donde A en V y λ en (V U T)*

El lenguaje generado L(G) es libre de contexto

# Gramática Libre de Contexto

GLC para el siguiente lenguaje:

$$\mathbf{L_1 = \{ 0^n 1^n \mid n \geq 0 \}}$$

Ver: 0n1n.jff y 0n1n.pl

$$L_1 = \{ \lambda, 01, 0011, 000111, 00001111, \ldots \}$$

$$G = (\{S\},\{0,1\},P,S)$$

P = { S → λ , S → 0S1 } ó { S → λ | 0S1 }

**Derivación:**

**S** => 0**S**1 => 00**S**11 => 000**S**111 => 000λ111 = 000111

S =>* 000111 y esta en $L_1$

# Gramática Libre de Contexto

GLC para el siguiente lenguaje:

$$\mathbf{L_2 = \{ 0^n 1^m \mid n > m,\ n,m \geq 0 \}}$$

Ver: 0n1m-NMayorM.pl y 0n1m-NmayorM.jff

$$L_2 = \{ 0, 000, 001, 00011, 00001, 000011, 0000111, \ldots \}$$

$$G = (\{S\},\{0,1\},P,S)$$

**Derivación:**

P = { S → 0 | 0S | 0S1 }

**S** => 0**S**1 => 00**S**11 => 000**S**11 => 0000**S**11 => 0000**S**11 => 0000011

# Gramática Libre de Contexto

GLC para el lenguajes de todos los binarios palindromes

Ver: palindromo.pl y palindromo.jff

$L_3$ = { 0,1,00,010,11,111,000,11011…}

G = ({S},{0,1},P,S)

**Derivación:**

P = { S → 0 | 00 | 1 | 11 | 0S0 | 1S1 }

**S** => 0**S**0 => 01**S**10 => 010**S**010 => 0101010

S =>* 0101010 esta en $L_3$

# Arbol de derivación

Derivación expresada en forma de árbol:

- Cada nodo del árbol va a contener un símbolo.
- En el nodo raíz se pone el símbolo inicial S.
- Se efectúa una ramificación del árbol por cada producción que se aplique: Si a la variable de un nodo, A, se le aplica una determinada regla $A \rightarrow \alpha$, entonces para cada símbolo que aparezca en α se añade un hijo con el símbolo correspondiente, situados en el orden de izquierda a derecha.
- Este proceso se repite para todo paso de la derivación.
- Si la parte derecha es una cadena vacía, entonces se añade un solo hijo, etiquetado con λ.
- En cada momento, leyendo los nodos de izquierda a derecha se lee la palabra generada.

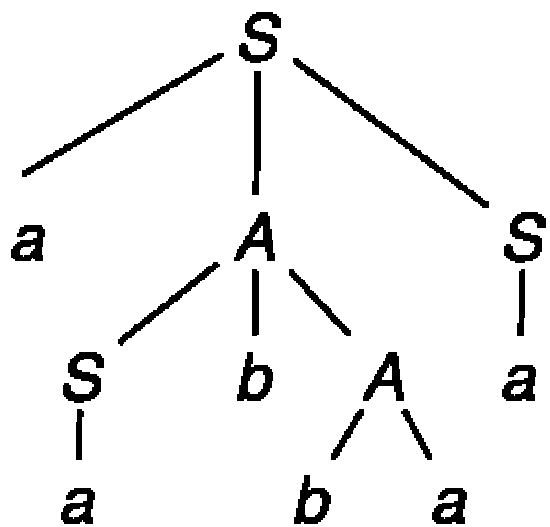
# Arbol de derivación

Por ejemplo:

$S \rightarrow aAS \mid S \rightarrow a, \quad A \rightarrow SbA \mid SS \mid ba$

La cadena “aabbaa” tiene asociado el árbol:

**S** => a**A**S =>
a**S**bAS => aab**A**S
=> aabba**S** =>
aabbaa

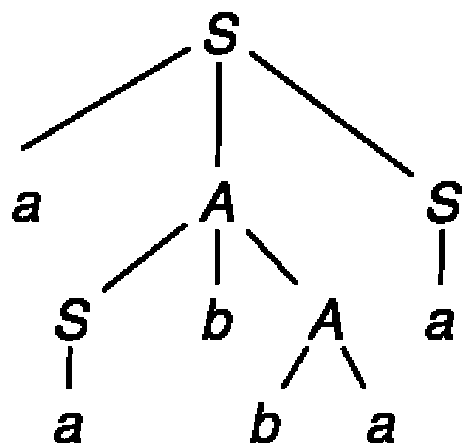


# Arbol de derivación

Un árbol de derivación puede proceder de dos cadenas de derivación distintas:

1. Se llama **derivación por la izquierda** asociada a un árbol a aquella en la que siempre se deriva primero la primera variable (más a la izquierda) que aparece en la palabra.

$S$ => $aAS$ => $aSbAS$ =>
$aabAS$ => $aabbaS$ => $aabbaa$

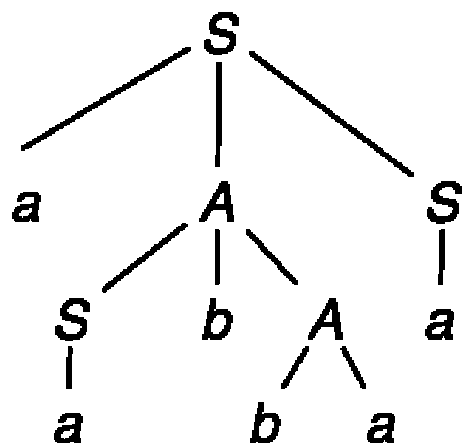


# Arbol de derivación

2. Se llama **derivación por la derecha** asociada a un árbol a aquella en la que siempre se deriva primero la última variable (más a la derecha) que aparece en la palabra.

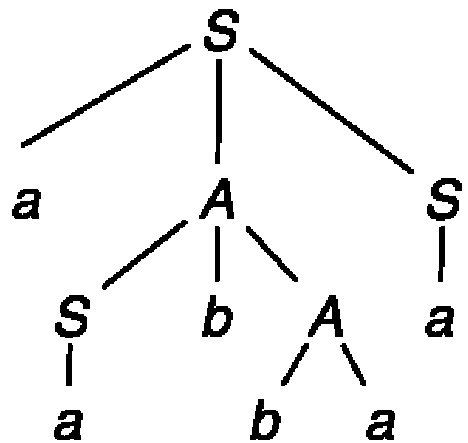
$S \Rightarrow aA\mathbf{S} \Rightarrow a\mathbf{A}a \Rightarrow$

$aSb\mathbf{A}a \Rightarrow a\mathbf{S}bbaa \Rightarrow aabbaa$



# Arbol de derivación

- Por ejemplo para el árbol de la cadena ***aabbaa***




- Derivación por la izquierda:

$S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS \Rightarrow aabbaS \Rightarrow aabbaa$

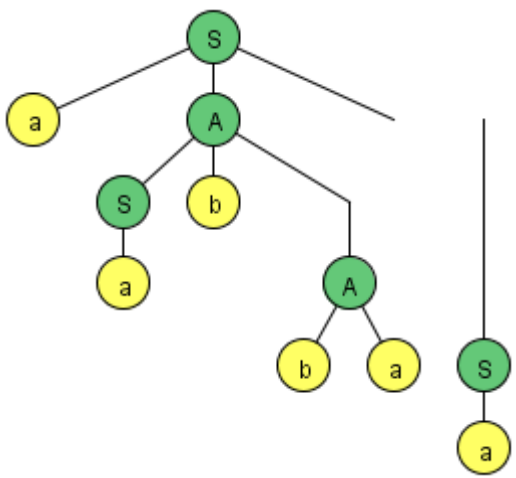
- Derivación por la derecha:

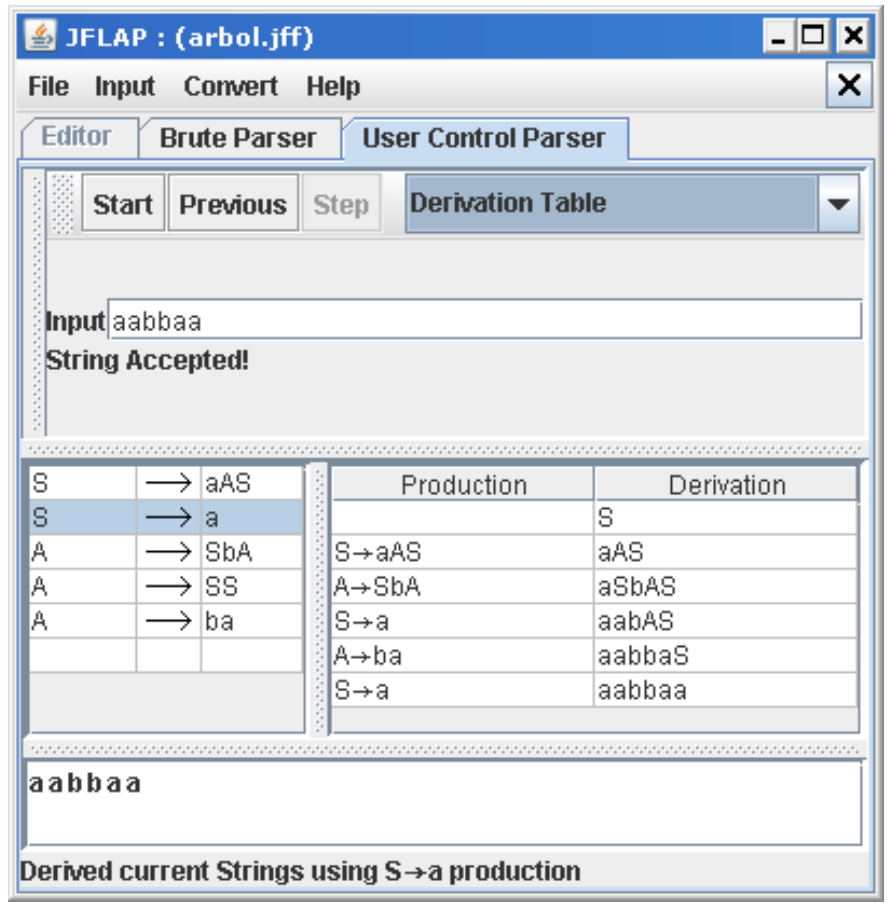
$S \Rightarrow aAS \Rightarrow aAa \Rightarrow aSbAa \Rightarrow aSbbaa \Rightarrow aabbaa$

# Arbol de derivación

- Derivación por la izquierda:

$S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS \Rightarrow aabbaS \Rightarrow aabbaa$

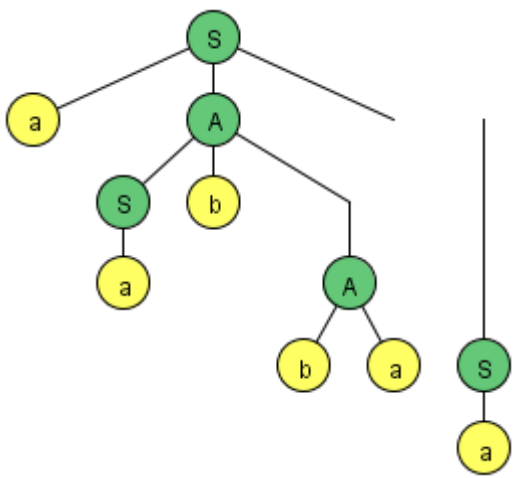


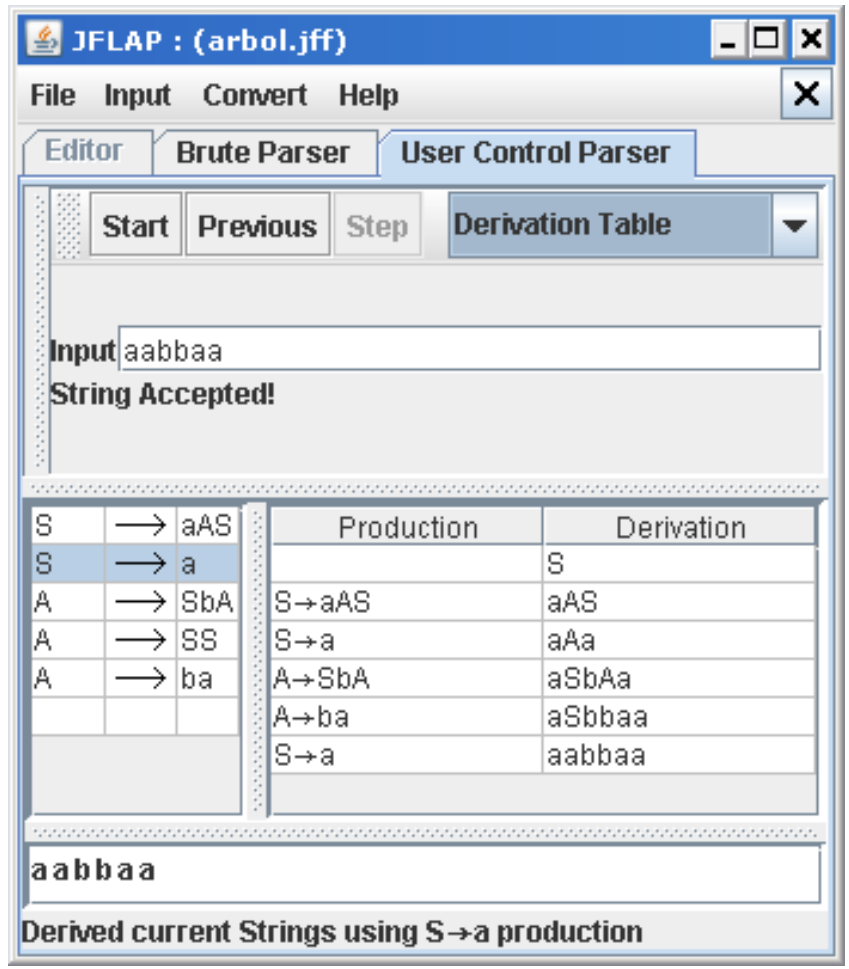



# Arbol de derivación

- Derivación por la derecha:

S => *aA**S*** => *a**A**a* => *aSb**A**a* => *a**S**bbaa* => *aabbaa*



# Gramática ambigua

Una gramática se dice ambigua si existe una palabra con dos árboles de derivación distintos.

Ejemplo: L = { $a^{2+3i}$ | i ≥ 0 }

La gramática G = ({S,A},{a},P,S)

P: S → AA, A → aSa, A → a,

Tiene para la palabra “aaaaa” las siguientes derivaciones por la izquierda:

**S** => **A**A => a**A** => aa**S**a => aa**A**Aa => aaa**A**a => aaaaa

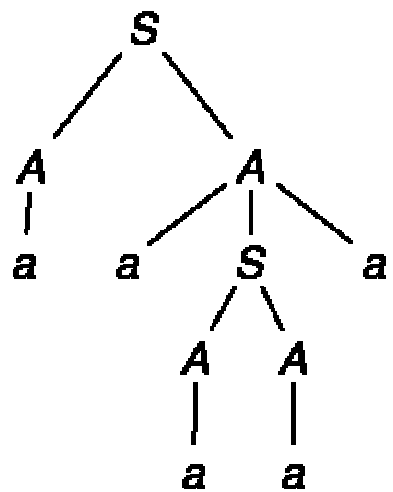
**S** => **A**A => a**S**aA => a**A**AaA => aa**A**aA => aaaa**A** => aaaaa

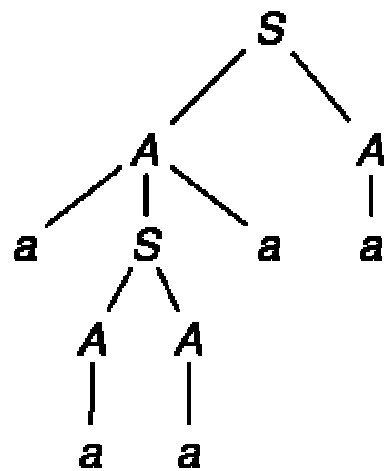
# Gramática ambigua

Ejemplo: Por tanto la gramática S → AA, A → aSa, A → a **es ambigua**, ya que la palabra "aaaaa" tiene 2 árboles de derivación:

**S** => **A**A => a**A** => aa**S**a => aa**A**Aa => aaa**A**a => aaaaa

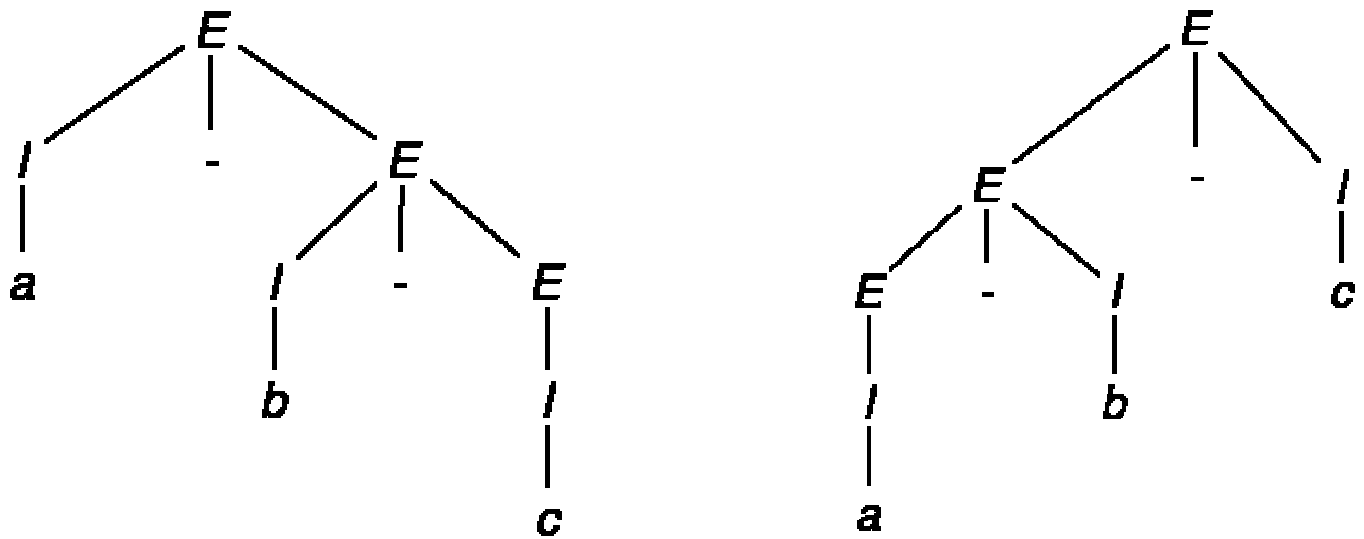
**S** => **A**A => a**S**aA => a**A**AaA => aa**A**aA => aaaa**A** => aaaaa



# Gramática ambigua

Ejemplo: La gramática $E \rightarrow I$, $E \rightarrow I$ - $E$, $E \rightarrow E$ - $I$, $I \rightarrow a \mid b \mid c \mid d$ , es ambigua, ya que la palabra "*a - b - c*" admite dos árboles de derivación distintos:



# Lenguaje inherentemente ambiguo

Un lenguaje libre de contexto es *inherentemente ambiguo* si toda gramática que lo genera es ambigua.

Ejemplo: el lenguaje L = { $a^{2+3i}$ | i ≥ 0 } La gramática G = ({S,A},{a},P,S), P: S → AA, A → aSa, A → a, es ambigua.

- L puede ser generado por otra gramática: $S \rightarrow aa \mid aaU$, $U \rightarrow aaaU \mid aaa$, que no es ambigua. Por tanto el lenguaje no es inherentemente ambiguo

# Simplificación de GLC

- Algoritmos de Simplificación de Gramáticas:
  - Eliminación de Producciones Nulas.
  - Eliminación de Producciones Unitarias.
  - Eliminación de Símbolos y Producciones Inútiles.

# Producción Nula

- Producción nula: $A \rightarrow \lambda$

Idea: dada una gramática *G*, construye una gramática $G_1$ sin producciones nulas y tal que $L(G_1) = L(G) - \{\lambda\}$.

- Si tenemos: $A \rightarrow \lambda$, $D \rightarrow aABc$

D => aABc => aBc, entonces Añadimos $D \rightarrow$ aBc y podemos eliminar $A \rightarrow \lambda$.

- Si también $B \rightarrow \lambda$, entonces habría que añadir: $D \rightarrow aBc$, $D \rightarrow aAc$, $D \rightarrow ac$

# Variables Nulificables

- Variables Nulificables son aquellas variables tales que en algún momento aparece producción: $A \rightarrow \lambda$ o si desde la variables se puede derivar λ: $A \Rightarrow^* \lambda$
- Idea: Identificar dichas variables nulificables, eliminar todas las producciones nulas y a añadir las producciones que compensen esta eliminación.

# Algoritmo

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

Algoritmo (variables nulificables en $N_i$)

i= 0

$N_0$= { A | A → λ una P y A en V }

REPETIR

    i= i +1

    $N_i$= $N_{i-1}$ U { A | A → α en P y α en $N_i$* }

HASTA ($N_i$ = $N_{i-1}$)

# Identificar variables nulificables

- Ejemplo:

$$S \rightarrow ABCBD$$
$$A \rightarrow CD$$
$$B \rightarrow Cb$$
$$C \rightarrow a \mid \lambda$$
$$D \rightarrow bD \mid \lambda$$

- $N_0 = \{C, D\}$
- $N_1 = N_0 \cup \{A\} = \{C, D, A\}$
- $N_2 = N_1 \cup \Phi = \{C, D, A\}$

# Simplificar sin variables nulificables

P'= Si $A \rightarrow X_1 X_2 \ldots X_n$ en P, entonces agregamos a P' las producciones $A \rightarrow \alpha_1 \alpha_2 \ldots \alpha_n$ : (1) Si $X_i$ no esta en $N_i$, $\alpha_i = X_i$ (2) $X_i$ esta en $N_i$ entonces $\alpha_i = X_i$ o $\alpha_i = \lambda$, (3) No todos los $\alpha_i$ son $\lambda$

P: S → ABCBD

A → CD

B → Cb

C → a | λ

D → bD | λ

P' =

S → ABCBD | BCBD | ABCB | ABBD | ABB | BBD | BCB | BB

A → CD | C | D

B → Cb | b

C → a

D → bD | b

$N_i = \{C, D, A\}$

Selfa: selfa-nulificable.txt

# Producción Unitaria

Dada la gramática G = (V,T,P,S)

Es una producción unitaria: A → B, A y B en V (en el cuerpo hay solo una variable)

Definición de derivable de A:

- Si A → B es una producción, entonces B es derivable de A.
- Si C es derivable de A, A → B es una producción y B → C, entonces C es derivable de B.

# Algoritmo: Producción Unitaria

Algoritmo: dada la GLC G = (V, T, P, S)

Precondición = {G no contiene producciones λ}

1. $P_1 = P$
2. Para todos A en V = {B | B es derivable de A}
3. Para cada par (A, B) tal que B es derivable a A y cada producción no unitaria B → α, añadir la producción A → α a $P_1$ sino esta presente ya en $P_1$.
4. Eliminar todas las producciones unitarias de $P_1$.

# Algoritmo: Producción Unitaria

Ejemplo:

$P_1$ = {
S → ABCBD | BCBD | ABCB |
ABBD | ABB | BBD | BCB | BB
A → CD | C | D
B → Cb | b
C → a
D → bD | b }

**Paso 2**:
S = Φ
A= {C, D}
B= Φ
C= Φ
D= Φ

**Paso 3**: Pares
- (A, C) $P_1$ U {A → a}
- (A, D) $P_1$ U {A → bD y A → b}

**Paso 4**: Eliminar producciones unitarias:
- (A, C) $P_1$ - {A → C}
- (A, D) $P_1$ - {A → D}

# Algoritmo: Producción Unitaria

Ejemplo: Selfa: selfa-unitaria.txt

$P_1$ =
S → ABCBD | BCBD | ABCB |
ABBD | ABB | BBD | BCB | BB
**A** → CD | **C** | **D**
B → Cb | b
C → a
D → bD | b

P sin producciones unitarias =
S → ABCBD | BCBD | ABCB |
ABBD | ABB | BBD | BCB | BB
A → CD | bD | a | b
B → Cb | b
C → a
D → bD | b

# Símbolo inútil

Los símbolos útiles son variables V y terminales T que son generadoras de terminales y alcanzables desde S:

Sea G = (V, T, P, S), **X** es un símbolo útil

si $S \Rightarrow^* \alpha \mathbf{X} \beta \Rightarrow^* w$

para α y β en (V U T)* y en w en T*

# Algoritmo: Variable Derivable

VV = VN = { A | A → w en P y w en T* }

MIENTRAS VN ≠ VV HACER

VV = VN

VN = VV U {A | A → α en P y α en (VV U T)*}

// (A → $X_1X_2…X_n$ cada $X_i$ esta en T o en VV)

FINMIENTRAS

V' = VN

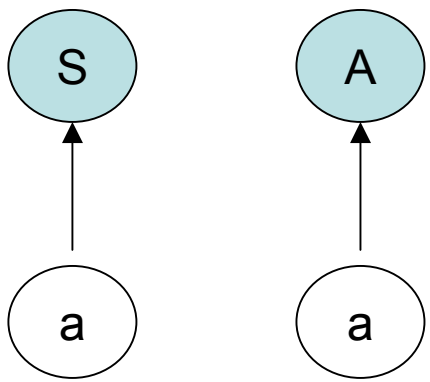
P' = { A → α | A en V' y α en (V' U T)* }

# Algoritmo: Derivable

Ejemplo:

P = S → BA | a
A → a

Inicio:
VN = {S,A}
T = {a}
1. VV ={S,A}
VN = {S, A} U {S,A} = {S,A}




Parada (VV = VN)
V'= {A, S}
P'= S → a
A → a

# Algoritmo: Símbolo alcanzable

VV = TV = TN = Φ; VN = {S}

MIENTRAS (VV≠VN) o (TV≠TN) HACER

VV = VN; TV = TN;

VN = VV U {X | A → $\alpha_1\alpha_2\ldots\ \alpha_n$ en P, X en V,
X en algún $\alpha_i$ y A en VV}

TN = TV U {X | A → $\alpha_1\alpha_2\ldots\ \alpha_n$ en P, X en T,
X en algún $\alpha_i$ y A en VV}

FINMIENTRAS

V'= VN; T'= TN;

P'= { A → α | A en V',α en (V'UT`)*,A→α en P }

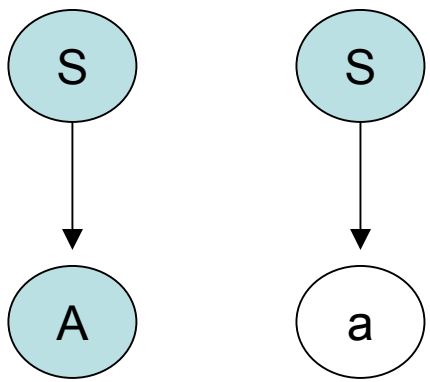
# Algoritmo: Alcanzable

Ejemplo:

P = S → Aa | a
A → a
B → SA

Inicio:
VN = {S}
1. VV ={S}
TV = Φ
VN = {S, A}
TN = {a}

2. VV ={S, A}
TV = {a}
VN = {S, A}
TN = {a}




V’ = {S, A}
T’ = {a}
P’ = S → Aa | a
A → a

# Formas Normales

Formas Normales:

Forma Normal de Chomsky (FNC):

**A → BC**

**A → a**

A,B y C en V y a en T

Forma Normal de Greibach (FNG):

**A → aα**

a en T y α en V*

# Forma Normal de Chomsky

***(Chomsky, 1959)***

***Teorema***: Cualquier GLC sin λ-producciones puede ser generada a una gramática en donde las producciones son de la forma A → BC o A → a

Hay dos operaciones básicas:

–Eliminar terminales en producciones que no sean A → a

–Eliminar producciones con un cuerpo de variables de longitud mayor de 2

# Algoritmo: FNC

Sea $A \rightarrow X_1X_2\ldots X_m$ en P y $m \geq 2$

1. Producciones del tipo $A \rightarrow a$ :

   Si $X_i$ es Terminal y $X_i = a$ entonces agregamos a P' la producción $C_a \rightarrow a$ y reempleamos $X_i$ por $C_a$. $V' = V \cup \{C_a\}$

2. Producciones $A \rightarrow BC$

   Para los casos en que $m \geq 3$, hacemos:

   $A \rightarrow BD_1$; $D_1 \rightarrow B_2D_2;\ldots, D_{m-2} \rightarrow B_{m-1}B_m$

   Agregamos las respectivas variables y producciones $V' = V \cup \{D_1, D_2\ldots, D_{m-2}\}$

# Algoritmo: FNC

Ejemplo: **Selfa:** selfa-FNC.txt

**S --> bA | aB**

**A --> bAA | aS | a**

**B --> aBB | bS | b**

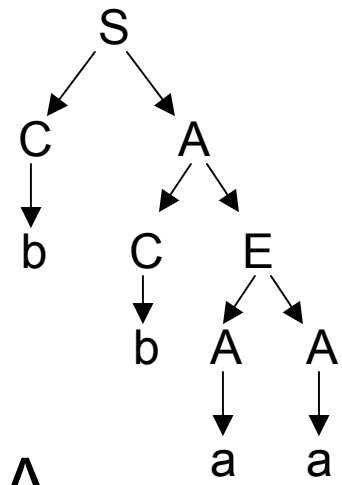
**Paso 1: A→ a**
**S → $C_b$A | $C_a$B**
**A → $C_b$AA | $C_a$S | a**
**B → $C_a$BB | $C_b$S | b**
**$C_a$ → a**
**$C_b$ → b**

**Paso 2: A→ BC**
**S → $C_b$A | $C_a$B**
**A → $C_b$$D_1$ | $C_a$S | a**
**B → $C_a$$D_2$ | $C_b$S | b**
**$C_a$ → a**
**$C_b$ → b**
**$D_1$ → AA**
**$D_2$ → BB**

**G en FNC:**
**S ---> CA | DB**
**A ---> a | CE | DS**
**B ---> DF | CS**
**E ---> AA**
**F ---> BB**
**C ---> b**
**D ---> a**
**B ---> b**

# Forma Normal de Chomsky

Árbol de derivación binario




(GLC original)

S => bA => bbAA => bbaA => bbaa

**(GLC en FNC)**

S => CA => bA => bCE => bbE => bbAA

=> bbaA => bbaa

FNC no parece tener aplicaciones importantes en lingüística natural. Tiene aplicación como una forma eficiente de comprobar si una cadena pertenece a un LLC

Si |w| = k entonces el número de pasos es 2k-1

# Forma Normal de Greibach

***(Sheila Greibach, 1965)***

- ***Teorema***: todo LLC sin λ- producciones pueden ser generado por una GCL en donde las producciones son de forma:

$$A \rightarrow a\alpha,\ a \text{ en } T \text{ y } \alpha \text{ en } V^*$$

FNG genera un símbolo terminal por cada forma sentencial y entonces una cadena de longitud n tiene exactamente n pasos de derivación.

**Actividad**: revisar el algoritmo

# Forma Normal de Greibach

G = ({S},{0,1},P,S)

P = { S → 0 | 00 | 1 | 11 | 0S0 | 1S1 }

FNG

P = { S → 0 | 0A | 1 | 1B | 0SA | 1SB,
A → 0
B → 1 }

S => 0SA => 01BA => 011A => 0011

Selfa: selfa-FNG.txt

# Parsing

Un algoritmo de parsing es un procedimiento que prueba las diferentes maneras de combinar las reglas de una gramática para encontrar una combinación que genere un árbol que represente la estructura de una cadena del lenguaje (Allen, 1987)

# Parsing

Según el procesamiento del lenguaje una GLC tiene: Reglas (A → α, α tiene al menos una variable) y Lexicón (A → a)

%reglas

s --> sn,sv | sv.

sv --> v | v,sn | v, sp | v, sn, sp.

sn --> n | pron | det, n.

sp --> prep, sn.

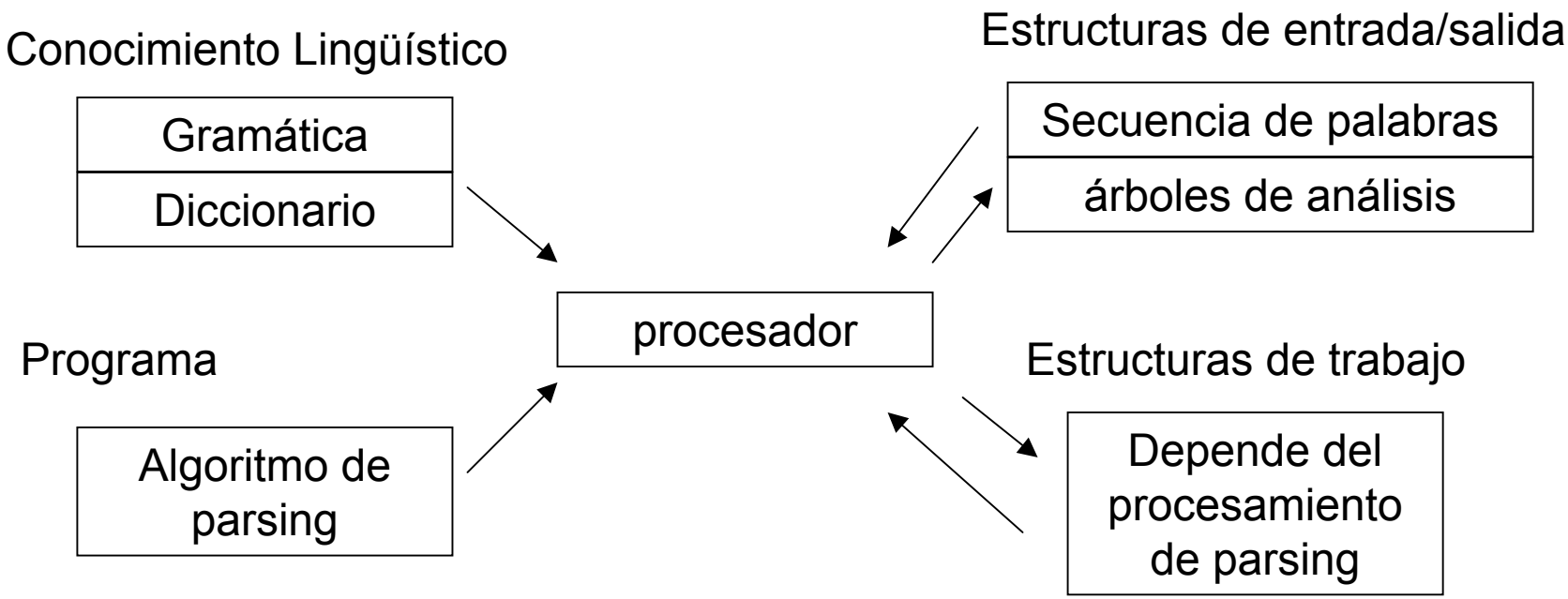
% lexicon

prep --> [a] | [de].
v --> [tiene] | [ama] | [habla].
n --> [ana].
pron --> [yo] | [tu] | [ella] | [nosotros].
det --> [el] | [la] | [una].

- Prolog: pln.pl

# Parsing

## Esquema general de un proceso de parsing (Winograd, 1983)



# Parsing

Los algoritmos de parsing mas utilizados son para GLC:

- Deciden que regla probar y en que orden
- Combinan estructuras de trabajo y diferentes parámetros

Clasificación:

- Análisis descendente/ascendente
- Procesamiento secuencial/paralelo
- Procesamiento determinista o no determinista

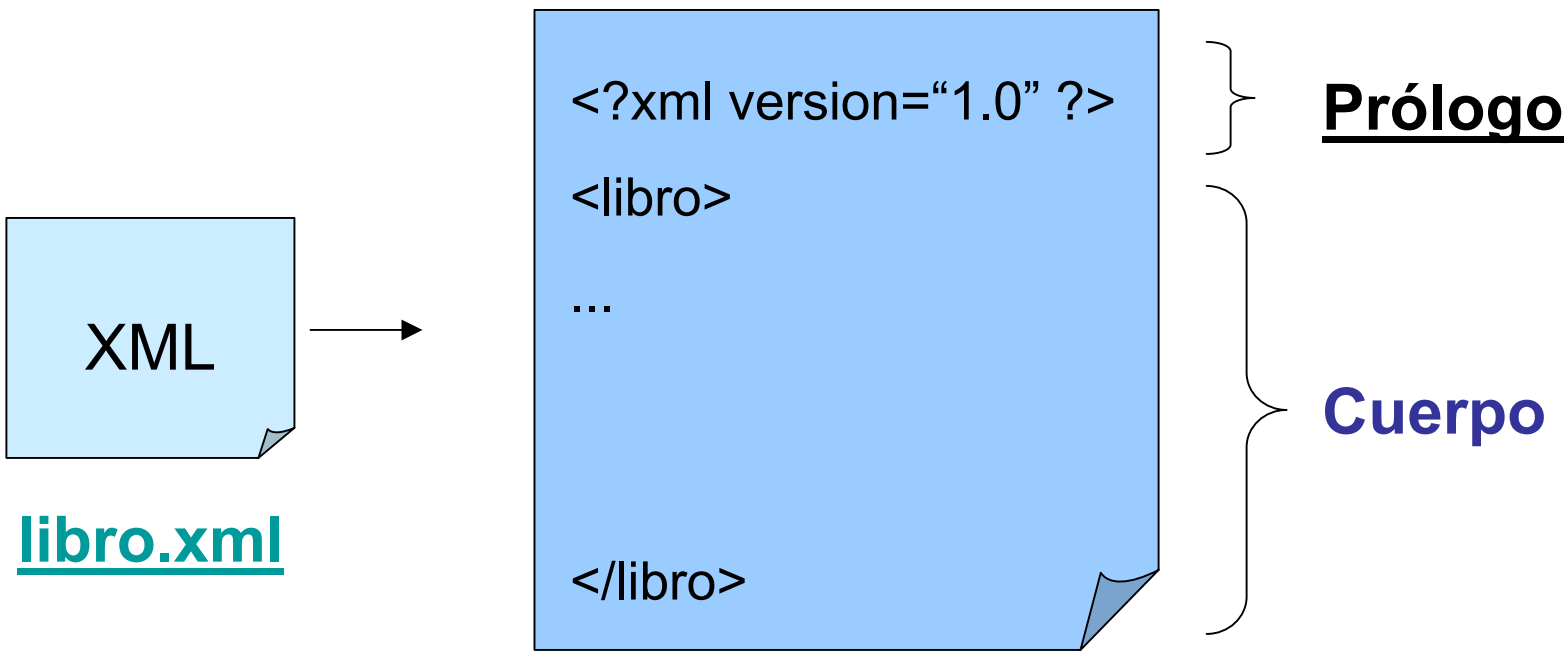
# Parsing

Algunos ejemplos de algoritmos de parsing:

- Backtracking
- Algoritmo de Earley
- Algoritmo de Cocke-Younger-Kasami (CYK)
- Algoritmo de Graham-Harrison-Ruzzo (GHR)
- Recursive-Descent-Parser
- Chart-Parser
- Left-Corner-Parser
- LR-Parser

# Ejemplo de Parsing XML

- Estructura de un XML



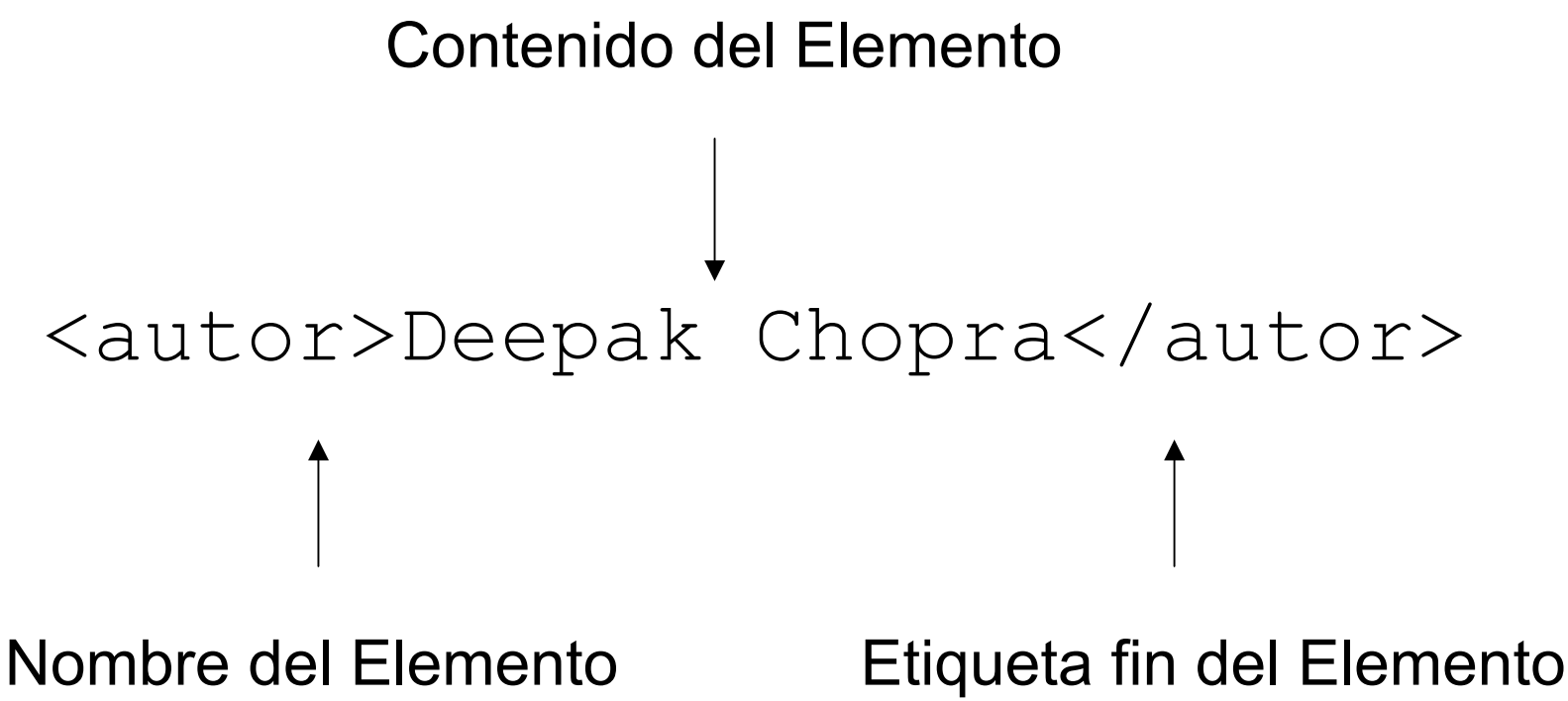
# Ejemplo de Parsing XML

Un Ejemplo de archivo XML: libro.xml

```
<?xml version="1.0" encoding="ISO-
   8859-1"?>
<libro>
   <autor>Deepak Chopra</autor>
   <titulo>El sendero del Mago</titulo>
   <precio moneda="bolivares">60</precio>
</libro>
```

# Reglas sintácticas XML

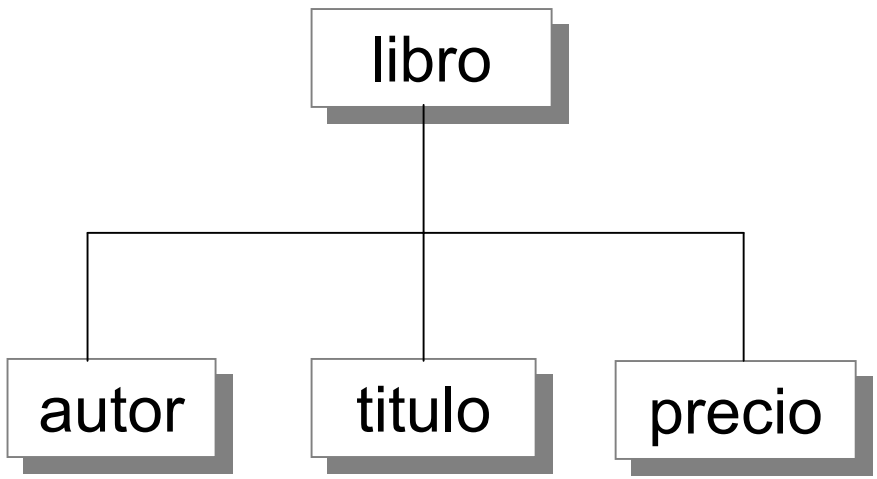
- **Elemento** y Contenido



# Reglas sintácticas XML

## Estructura Jerárquica de Árbol

```
<libro>
<autor>Deepak Chopra</autor>
<titulo>El sendero del Mago</titulo>
<precio moneda="bolivares">60</precio>
</libro>
```



# Técnica de validación XML

Existen 2 formas de definir los elementos que contiene un documento XML a través de reglas gramaticales de los elementos, atributos y entidades:

- **DTD (Document Type Definition)**

  Archivos con extensión .dtd
- **XML Schema**

  Archivos con extensión .xsd

# GLC

GLC G = (V,T,P,S) del cuerpo de un libro en XML:

S → <libro>A</libro>

A → BCDEFGH | BCDEFG

B → <autor>K</autor> | BB

C → <titulo>K</titulo>

D → <isbn>K</isbn>

E → <editorial>K</editorial>

F → <sumario>K</sumario>

G → <precio moneda="K">K</precio>

H → <otro>K</otro>

K → KK | a|b|…|z|0|1…|9|.|!...

# GLC en DTD

Un ejemplo de DTD: libro2.dtd

```
<!ELEMENT libro (autor+, titulo, isbn, editorial, sumario,
      precio, otro?)>
<!ELEMENT autor (#PCDATA)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT editorial (#PCDATA)>
<!ELEMENT sumario (#PCDATA)>
<!ELEMENT precio (#PCDATA)>
<!ATTLIST precio moneda CDATA #REQUIRED>
<!ELEMENT otro (#PCDATA)>
```

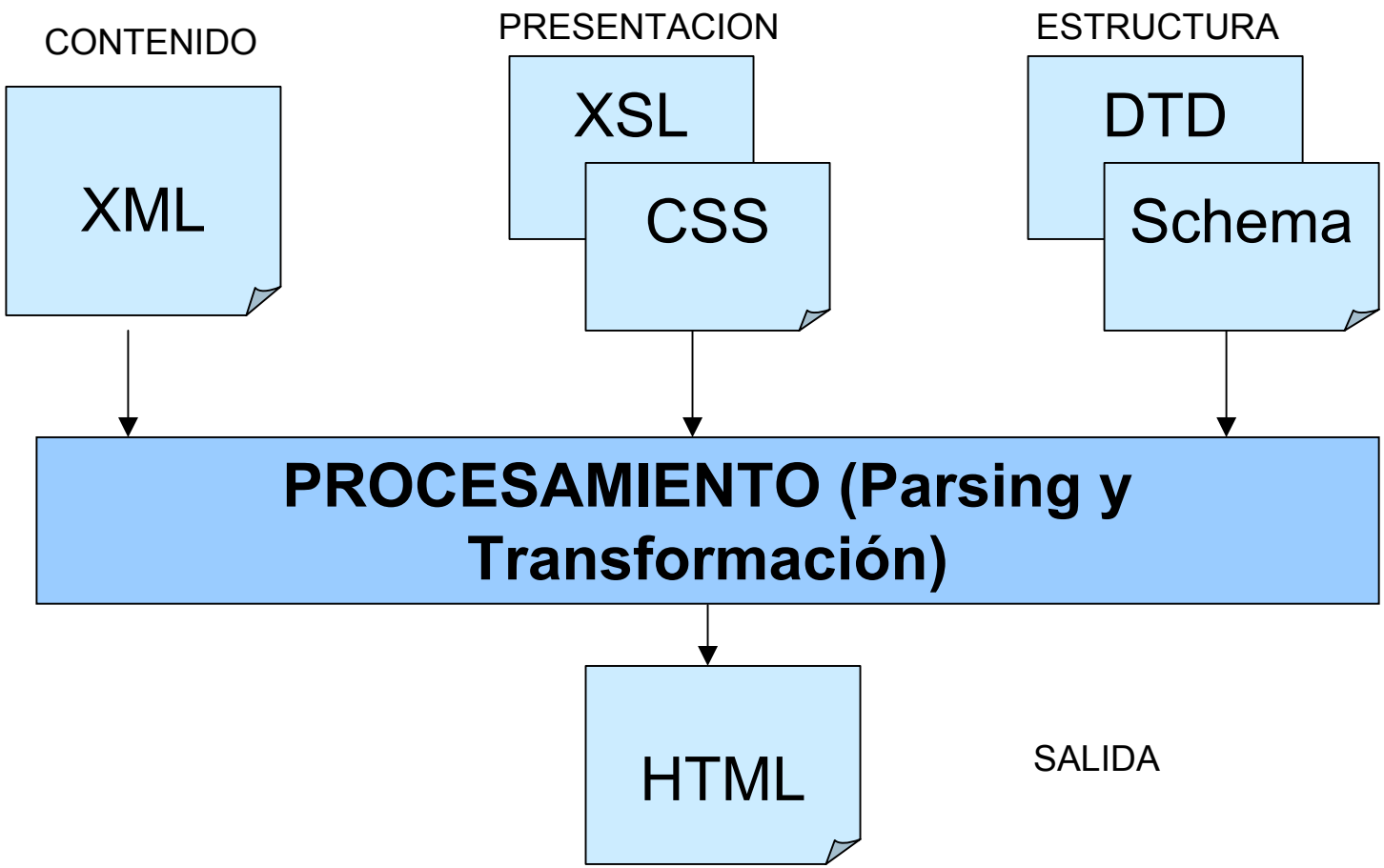
# GLC en XML Schema

## Un ejemplo de XML Schema: libro2.xsd

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<element name="libro">
   <complexType>
    <sequence>
         <xsd:element name="autor" type="xsd:string"/>
         <xsd:element name="titulo" type="xsd:string"/>
         <xsd:element name="isbn" type="xsd:string"/>
         <xsd:element name="editorial" type="xsd:string"/>
         <xsd:element name="sumario" type="xsd:string"/>
         <xsd:element name="precio" type="xsd:string">
              <attribute name="moneda" type="string" use="required"/>
         </xsd:element>
         <xsd:element name="otro" type="xsd:string"/>
    </sequence>
   </complexType>
</element>
</xsd:schema>
```

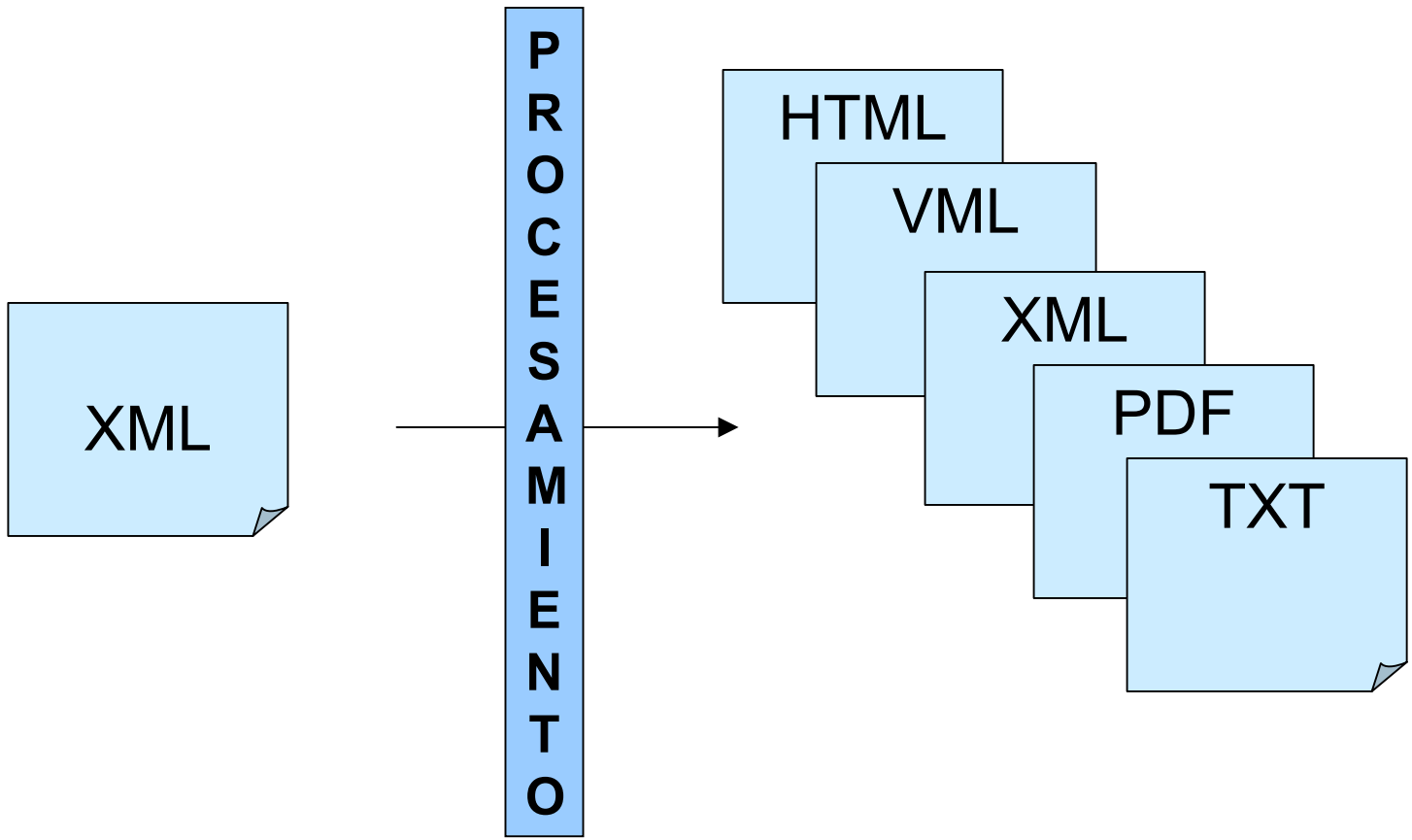
# Parsing XML

Separación de procesamiento, presentación, estructura y contenido:

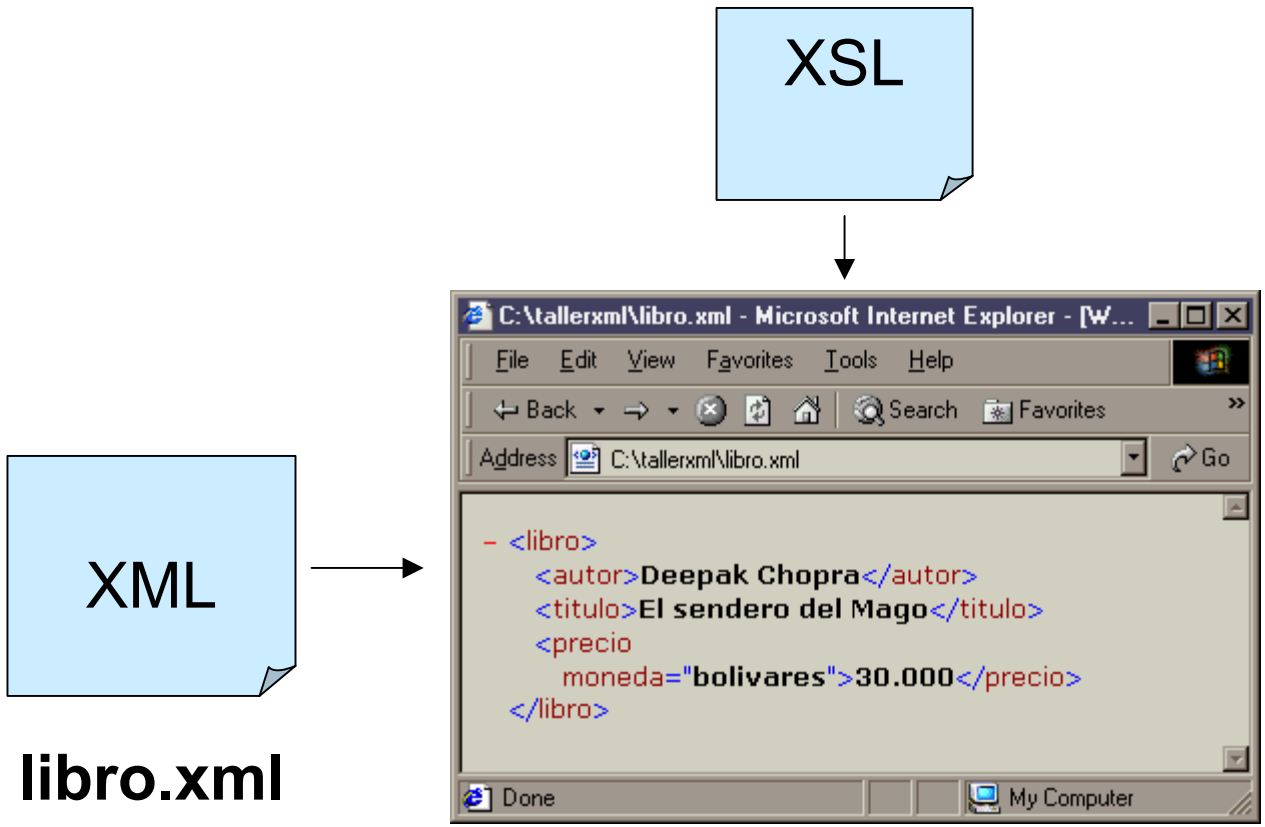


# Parsing y Tranformación

Muchas formas de presentación a partir de un documento XML:



# Parsing XML en la práctica



# Ejemplo de sistemas de Lindenmayer

L-Sistemas:

- Son gramáticas formales de derivación paralela (en oposición a las de Chomsky, secuenciales).
- Creadas para expresar el desarrollo de organismos vivos: gran capacidad expresiva.
- Ver: http://www.destinofractal.com/tutoriales/L-Systems/lsystems.htm

# Chomsky vs. Lindenmayer

Gramáticas de Chomsky:

- Se sigue una secuencia para aplicar las reglas de una en una.
- Los componentes de la gramática se justifican así:
  - Hay símbolos a los que no se puede aplicar reglas porque ya son símbolos finales (*terminales*).
  - Los demás no pueden estar en la expresión final (*no terminales*)
  - El proceso comienza con el símbolo que representa lo que se quiere analizar / construir (*axioma*)
  - La expresión final obtenida (que pertenece al lenguaje generado por la gramática) está compuesta sólo por terminales. Las palabras intermedias no pertenecen a él.

# Chomsky vs. Lindenmayer

Sistema Lindenmayer:

- Se ha aplicado reglas a todos los símbolos de cada palabra simultáneamente.
- Es necesario añadir las siguientes reglas de transformación:
  - Por lo que ya no hay distinción entre símbolos terminales y no terminales y siempre se puede derivar.
  - Todas las palabras (intermedias también) pertenecerían al lenguaje del sistema.

# Sistemas de Lindenmayer

- *Motivación biológica:* Surgen como herramienta para describir el desarrollo de organismos pluricelulares incorporando aspectos: Genéticos, Citológicos y Fisiológicos
- Usada para modelar los procesos de crecimiento de las plantas.
- Los L-sistemas fueron introducidos en 1965 por el biólogo y botánico húngaro Aristid Lindenmayer (1925-1989), profesor de la Universidad de Utrech.

# Sistemas de Lindenmayer

*Definición formal del L-sistema* = (Σ, P, ω)

Donde, (al par (Σ, P) se le llama esquema L)

- Σ es un alfabeto.
- P es un conjunto de reglas de producción del tipo
- A → x, A esta en Σ y x esta en Σ* (puede ser vacía).
- ω el axioma.

# Sistemas de Lindenmayer

Ejemplo:

- Se define un sistema-L que genera cadenas cuyo número de símbolos b es la serie de Fibonacci:

L = {{a,b}, {a → b, b → ab}, a}

Alguna de sus cadenas:

{a, b, ab, bab, abbab, bababbab, ...}

# Sistemas de Lindenmayer

- Las reglas se aplican iterativamente (tantas como sea posible en cada iteración) desde el estado inicial.
- Un L-sistema se dice libres del contexto si las reglas se aplican sobre símbolos individuales, sin considerar sus vecinos. En caso contrario, se llama sensibles al contexto.
- Si hay una regla para cada símbolo, el sistema es determinista (si ademas es libres de contexto, se llama D0L-sistema). Si hay muchas reglas escogidas con cierta probabilidad, se llama L-sistema estocástico

# Sistemas de Lindenmayer

Podemos usar L-sistemas para generar gráficos, asociando por ejemplo, los símbolos a comandos gráficos (comandos de tortuga, similares al lenguaje Logo)

Por ejemplo:

- F  Dibujar una línea hacia delante de longitud predeterminada.
- -  Girar hacia la izquierda el ángulo predeterminado.
- +  Girar hacia la derecha el ángulo predeterminado.

# Sistemas de Lindenmayer

Jflap: tree.jff

Axioma: R ~ ## B

Reglas:

B ➔ [ ~ ## T L - B + + B ]

L ➔ [ { - g + + g % - - g } ]

R ➔ ! @@ R

T ➔ T g

Color = brown

polygonColor = forestGreen