Sintaxis y Semántica del Lenguaje

Wilo Carpio Cáceres

2013

A la memoria de Roberto, mi amado padre !

En la década del 40, nace el homo digitalis . . .

El exponencial crecimiento de Internet y la descomunal proliferación de las redes sociales que conforman el sistema nervioso de la actual e-cultura, donde multiplican diversos formatos de comunicación, los cuales se remontan a los albores de la cultura humana, cuando en la Grecia del siglo VII AC, nace la primera globalización del lenguaje, producida por la explosión comercial, la aparición de la moneda y la internacionalización de los mercados de aquel entonces.
El contacto con otras culturas y el interés por venderles, provocaba en aquellos comerciantes la necesidad conocer con quién estaban tratando, cuales eran sus valores y convicciones propias, adoptando sus costumbres en el modo de usar sus formas de comunicación. Está antigua globalización generó una suerte de reingeniería del lenguaje, que vigorizó los nacionalismos, revitalizó las identidades de grupos étnicos, los sentimientos religiosos, los diversos fundamentalismos y una renovada presencia del espiritualismo.

El tiempo pasó, ahora es inexorable e indispensable contar con capacidad de acceso al actual formato de la comunicación humana basado en información digitalizada de carácter científico, técnico y humanístico de componentes culturales, donde interactúan como naturales administradores, solo algunas pocas empresas transnacionales y/o naciones privilegiadas, creadoras y dueñas de las tecnologías, desarrolladas sobre sus diversas redes de comunicación, tales como la WEB, o como Internet, para ofrecer, vender, consumir, utilizar información tanto para imponer sus productos comerciales, como para trasmitir sus productos culturales y por ende, para implantar sus proyectos.

Tal estructura digitalizada va transformando en realidad palpable la mutación de aquel lejano homo erectus hacia el nuevo formato humano del homo digitalis, quién como morador artífice de la aldea global, es el único habitante dotado de la potencialidad necesaria para lograr, por un lado la supervivencia menos traumática en la sociedad digitalizada y por otro, para ser el paladín estructurador de esta e-sociedad buscadora de la coexistencia más digna de cada componente de la raza humana.

En el ámbito globalizado resulta cada vez más difícil "aferrarse" a algo razonablemente estable por su validez y permanencia en el tiempo, pues pareciera que estamos migrando a una sociedad nihilista donde existe la "creencia" de que todos los valores no tienen sentido, que nada puede ser totalmente conocido ni comunicado, puesto que el hombre no logrará conocer la verdad cuando se sumerge en la virtualidad de la e-sociedad, donde nada es establecido, todo es cuestionado, todo lo que se consideraba "cierto" o "tradicional" ha sido derribado, invadido por la cultura posmodernista, que dió origen a nuevo ser humano, al hedonista e individualista "hombre postmoderno", en concreto al “homo digitalis”.

En la e-sociedad, los componentes de la e-cultura, no solo carecen de toda regla absoluta en el cual estén de acuerdo sus miembros, sino que cada componente, crece según los intereses de estrategias políticas patrimoniales de los países que administran los e-systems con los que manejan los gobiernos de los demás países consumidores.

Personalmente, anhelo que la proliferación de la información digital como herramientas de apertura de las fronteras del mundo, debería imponer un profundo análisis de los valores sociales y económicos, de acuerdo a la mayor demanda de transparencia e idoneidad, destacando la real importancia del lenguaje como bien, sobre todo lo referido a la calidad de la vida humana y su comportamiento.

$\theta, \rho\ \Phi\ \Sigma \in L_1 \neq w^2 \geq \leq \forall \Leftrightarrow > < \cap \cup \exists\ \subseteq \notin\ \delta$

GRAMÁTICAS FORMALES

Wilo Carpio Cáceres

2013

TEORÍA DE LAS GRAMATICAS FORMALES

Siguiendo al lingüista **Noah Chomsky**, la gramática describe la estructura de las lenguas, explicando sus oraciones que se entienden e interpretan, mediante la "**Gramática Universal**", o modelo del conocimiento lingüístico o competencia lingüística, que supone el conocimiento innato e inconsciente, propio de la persona, que le permite producir y comprender las oraciones de su lengua, aun que nunca las haya escuchado.

Tal competencia lingüística supone un **conocimiento innato y el lenguaje como capacidad configurada genéticamente**; tesis demostrada por la capacidad humana de sus módulos cerebrales especializados innatos y por el descubrimiento de un gen cuyas mutaciones producen deficiencias específicas en la función cerebral del lenguaje. Gracias a esto es posible elaborar gramáticas para generar todas las oraciones gramaticales y eliminar las agramaticales de cualquier lengua.

Para Chomsky existen reglas gramaticales universales y otras muchas específicas de cada lengua, que permiten que los elementos que forman una oración puedan ordenarse de varias maneras. Ejemplo: 'Messi metio el gol' y 'Este gol ha sido metido por Messi'. Según esto **existe una gramatica**:

- **TRANSFORMACIONAL**: Dispone unidades semánticas subyacentes, que mediante reglas las transforma en elementos de una oración, reconocible o interpretable.
- **GENERATIVA**: Genera todas las oraciones aceptables.

ROLES GRAMATICALES

La gramática como técnica para **formalizar un lenguaje**, **define**:

- **FORMATOS**: Validos de combinación de símbolos de su alfabeto.
- **NORMAS**: Para generar el correspondiente lenguaje formal.
- **REGLAS**: Formadoras de las palabras componentes del lenguaje.

Ejemplo: Para estructurar frases en lenguaje español

Con las REGLAS	Se generan las frases
< Frase > → < Sujeto > < Predicado > < Signo > < Sujeto > → < Articulo > < Sustantivo > < Predicado>→< Verbo > < Objeto > < Objeto> →<Preposición><Sujeto>| λ |<Advervio> < Articulo> → **| la | el | λ | los** < Verbo> → **| enseña | ama | estudia** < Sustantivo> →**| Facultad | λ | Tucuman | novio** <Adverbio>→ **| mucho | tarde | λ | medida|tecnología** < Signo > → **. | ; | ,**	<Frase>→**< Sujeto >** < Predicado ><Signo> →**<Articulo ><Sustantivo >**<Predicado><Signo> →<Articulo><Sustantivo>**<Verbo><Objeto>**<Signo> →<Articulo><Sustantivo><Verbo>**<Advervio>**<Signo> → **la Facultad enseña tecnología.** →<Articulo><Sustantivo><Verbo>**<Advervio>**<Signo> → **el novio ama mucho.**

Chomsky clasifica las gramaticas, en cuatro grupos

Tipo	Grupo	Gramatica	Lenguaje	Automata
0	G_0	**Sin restricción**	L_0: **Recursivo enumerable** Nivel pragmático	Maquina de Turing MT
1	G_1	**Dependiente del contexto**	L_1: **Dependiente de contexto** Nivel Semántico	Autómata Linealmente Acotado ALA ó ALL
2	G_2	**Independiente del contexto**	L_2: **Independiente de contexto** Nivel Sintáctico	Autómata de Pila AP
3	G_3	**Regular**	L_3: **Regular** Nivel Léxico	Autómata Finito AF

La jerarquía implica que si L_1 contiene al conjunto L_0, en general, se define: $L_3 \subseteq L_2 \subseteq L_1 \subseteq L_0$

GRAMATICA ESTRUCTURADA POR FRASES

DEFINICION	ELEMENTOS:	CONTENIDO:
Es la **cuatrupla** (4-Tupla) $G=(\Sigma_T,\Sigma_N,S,P)$	• Σ_T:Alfabeto de símbolos terminales	Cada símbolo terminal conforma el formato no variable o final de la palabra
	• Σ_N:Alfabeto de símbolos no terminales	Cada símbolo no terminal es una **variable** que representa un estado intermedio de formación de la palabra, bajo la condición: $\Sigma=\Sigma_T\cup\Sigma_N$ y $\Sigma_T\cap\Sigma_N=\phi$
	• **S**: Axioma o símbolo inicial o de la gramática	Símbolo de inicio para generar cada palabra del lenguaje. Cumple con : $S \in \Sigma_N$
	• **P**: Reglas de Sustitución o Produccion	Conjunto finito de transformaciones para alcanzar el estado de palabra formada por solo símbolos terminales.Para esto: A partir del axioma **S** se detecta cada símbolo no terminal y se efectua la producción para transformarla en terminal.
Sus formatos son:	a. **PARES de VALORES**: La gramática $G = (\Sigma_T, \Sigma_N, S, P) = (\{ 0, 1 \}, \{ M, N \}, M, P)$, cuyas producciones pueden ser representados en forma: ▪ **NORMAL**: P = {(**M** ::= 1N1), (**M** ::= 0N0),(**N** ::= M),(**N** ::= 1), (**N** ::= 0), (**N** ::= λ) } ▪ **SIMPLIFICADO**: **M** ::= 1N1 | 0N0 **N** ::= M | 1 | 0 | λ	
	b. **FORMA SENTENCIAL**: Tiene la forma: $S\rightarrow^{*}w$, $w\in\Sigma_T$; y **w** es de **forma sentencial** con derivaciones desde el axioma **S** hasta la palabra final y **todos sus símbolos** $\in\Sigma_T$. Ejemplo: En la gramática anterior: **La cadena:** • **010**: Es sentencia (M → 0N0 → **010**) Sus símbolos $\in \Sigma_T$ • **1M1**: No es sentencia (M → 1N1 →**1M1**) Porque **M** no es terminal. • **1001**: Es sentencia (M → 1N1 →1M1 → 10N01 → **1001**) Sus símbolos $\in \Sigma_T$ • **1011101**: Es sentencia (M → 1N1 →1M1 → 10N01 → **1011101**) Sus símbolos $\in \Sigma_T$.	

Ejemplo: Si aplicamos los conceptos anteriores y usando paréntesis angulares < > para las estructuras sintácticas, una **gramática estructurada por frases,** podríamos definirla como:

$G = (\Sigma_T, \Sigma_N, S, P)$ =
({ el, la, los, Facultad, Tucuman, novios, crece, avanza, aman, para, con, sin, mucho, tarde, . , medida **},**
{<Frase><Sujeto>< Predicado><Signo><Articulo><Sustantivo><Verbo><Objeto>< Preposición >< Advervio >**},**
<Frase>,P)

Sus producciones P pueden ser:

<Frase > → < Sujeto > < Predicado > < Signo >
<Sujeto > → < Articulo > < Sustantivo >
<Predicado > → < Verbo > < Objeto >
<Objeto >→<Preposición><Sujeto>|λ|<Advervio>
<Articulo > → | la | el | λ | los
<Sustantivo > → | Facultad | λ | Tucuman | novios
<Verbo > → | crece | λ | avanza | aman
<Preposición > → | para | con | λ | sin
<Adverbio > → | mucho | tarde | λ | medida
<Signo > → **.** | **;** | **,**

Puede generarse la frases:

<Frase>→ **< Sujeto >** <Predicado> < Signo>
→ **< Articulo > < Sustantivo >** < Predicado > <Signo>
→<Articulo><Sustantivo> **<Verbo> <Objeto>** < Signo>
→<Articulo><Sustantivo><Verbo>**<Advervio>**< Signo>
→ **La Facultad crece mucho.**
→ **Los novios aman tarde.**

→<Articulo><Sustantivo>**<Verbo><Objeto>**<Signo>
→<Articulo><Sustantivo>**<Verbo>**<Preposición><Sujeto><signo>
→<Articulo>< Sustantivo>**<Verbo>**<Preposición><Articulo>
<Sustantivo><Punto>→ **La Facultad aman sin la Tucuman.**

La incoherencia de la última frase muestra que la gramatica planteada solo enfoca la sintaxis y no así la semántica de las frases.

LENGUAJE $L_{(G)}$ generado por la GRAMATICA FORMAL

Conjunto de palabras o cadenas generadas por derivaciones, a partir del axioma de la gramática.

$$L_{(G)} = \{ w | S \rightarrow^* w, w \in \Sigma_T^* \}$$

- **S** : **Axioma**: Símbolo desde el que se hacen derivaciones para generar la palabra del lenguaje.
- $S \rightarrow^* w \in \Sigma_T^*$: **Derivaciones posibles** $\rightarrow^*$ de la **cadena** w a partir del **axioma** S hasta que los símbolos de la cadena w pertenezcan al **alfabeto de símbolos terminales** Σ_T.
- Σ_T^*:**Alfabeto de Símbolos Terminales**, incluida la palabra vacía

Ejemplo: La gramática $G_1 = (\Sigma_T, \Sigma_N, S, P) = (\{ 1 \}, \{ A, B \}, A, P)$ con las

Producciones		Derivaciones
Normal	Simplificado	$A \Rightarrow B \Rightarrow 1$ $A \Rightarrow B \Rightarrow 1\,B \Rightarrow 11$ $A \Rightarrow B \Rightarrow 1\,B \Rightarrow 1\,1\,B \Rightarrow 111$
P = {(A ::= 1), (A ::= B), (B ::= 1), (B ::= 1B)}	A ::= 1 | B B ::= 1B | 1	

Genera el lenguaje: $L(G_1) = (\Sigma_T, \Sigma_N, S, P) = \{ 1^n \mid n \geq 1 \} = \{ 1, 11, 111, \ldots \}$

Ejemplo: La gramática $G_2 = (\Sigma_T, \Sigma_N, S, P) = (\{ 1 \}, \{ A, B, C \}, A, P)$ con las

Producciones		Derivaciones
Normal	Simplificado	$A \Rightarrow B \Rightarrow 1$ $A \Rightarrow C \Rightarrow B \Rightarrow 1\,B \Rightarrow 1\,1$ $A \Rightarrow 1C1 \Rightarrow 1\,B\,1 \Rightarrow 1\,1\,1$
P = { (A ::= 1), (A ::= C), (A::= 1C1),(B ::=1),(B ::= 1B)}	A ::= 1 | C |1C1 C ::= 1| B B::= 1 | 1B	

Genera el lenguaje: $L(G_2) = (\Sigma_T, \Sigma_N, S, P) = \{ 1^n \mid n \geq 1 \} = \{ 1, 11, 111, \ldots \}$

Notemos que el mismo lenguaje: $L(G_2) = \{ 1^n \mid n \geq 1 \} = \{ 1, 11, 111, \ldots \}$ puede ser generado por gramáticas distintas

Ejemplo: La gramática $G = (\Sigma_T, \Sigma_N, S, P) = (\{ 0,1 \}, \{ A, B \}, A, P)$ con las

Producciones		Derivaciones
Normal	Simplificado	$A \Rightarrow 1B1 \Rightarrow 1\,\lambda\,1 \Rightarrow 1\,1$ $A \Rightarrow 1B1 \Rightarrow 101$ $A \Rightarrow 1B1 \Rightarrow 111$
P = { (A ::= 1B1), (A ::= 0B0), (B::= A), (B::=1), (B::=0), (B ::= λ)}	A ::= 1B1 | 0B0 B ::= A | 1 | 0 | λ	

Genera el lenguaje: $L(G) = (\Sigma_T, \Sigma_N, S, P) = \{ w^n \mid w = w^{-1} \} = \{ 11, 101, 111, \ldots \}$ palíndromos

Ejemplo: La gramática: $G_1 = (\Sigma_T, \Sigma_N, S, P) = (\{ a, b \}, \{ M, N, S \}, S, P)$, para las:

Producciones	Derivaciones
S::=MNS | λ NM ::= MN NS::=b Nb::=bb Mb::= ab Ma ::= aa	**S** → MNS→Mb→**ab** → a^1b^1 **S**→MNS→MNMNS→MMNNS→MMNb→MMbb → Mabb→**aabb** → $a^2\,b^2$

Genera el lenguaje: $L(G) = \{ a^n\,b^n \mid n \geq 1 \} = \{ a^1\,b^1, a^2\,b^2, a^3\,b^3, \ldots \}$

Ejemplo: La gramática: $G_1 = (\Sigma_T, \Sigma_N, S, P) = (\{ a, b, c \}, \{ S, B, C \}, S, P)$, para las:

Producciones	Derivaciones
S → aSBC | aBC CB→BC aB→ab bB→bb bC→bc cC→cc	S → aBC → abC → abc = $a^1b^1c^1$ S → aSBC → aaBCBC → aabCBC → aabBCC → aabbCC→ aabbcC → aabbcc = $a^2b^2c^2$ S→aSBC→aaSBCBC→aaaBCBCBC→aaaBBCCBC→aaaBBCBCC→aaaBBBCCC→aaabBBCCC→aaabbBCCC→aaabbbCCC→aaabbbcCC→aaabbbccC→aaabbbccc = $a^3b^3c^3$

Genera el lenguaje: $L(G) = \{ a^n b^n C^n \mid n \geq 1 \} = \{ a^1b^1c^1, a^2b^2c^2, a^3b^3c^3, \ldots, a^n\,b^n c^n \}$

EQUIVALENCIA DE GRAMATICAS:

La gramatica G_1 es equivalente a otra G_2, cuando ambas generan el mismo lenguaje $L_{(G)}$.

Ejemplo: Las gramáticas:

- $G_1 = (\Sigma_T, \Sigma_N, S, P) = (\{ 0,1 \}, \{ A, B, S \}, S, P)$ con las

Producciones	Derivaciones
P = {(S::= A0), (A0::= 1B1), (1A::= AB0), (B ::= λ), (B::= 1), (B::= 0)}	S → A0→ 1B1→ 1λ1 → 11 S → A0 → 1B1→ 101 S → A0→ 1B1→ 111
Genera el lenguaje: $L(G_1) = \{ 11, 101, 111, . . \}$	

- $G_2 = (\Sigma_T, \Sigma_N, S, P) = (\{ 0,1 \}, \{ A, B \}, A, P)$ con las

Producciones	Derivaciones
P ={ (A ::= 1B1), (A ::= 11), (1B1 ::= 101), (1B1 ::= 111)}	A → 11 A → 1B1→ 101 A → 1B1→ 111
Genera el lenguaje: $L(G_2) = \{ 11, 101, 111, . . \}$	

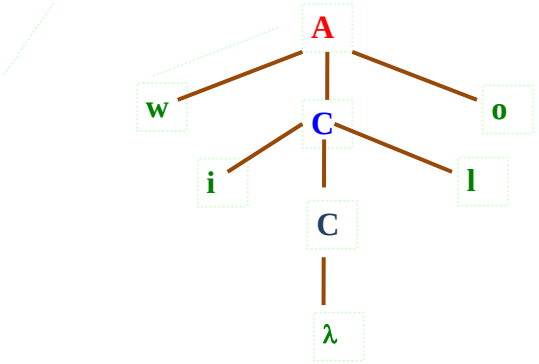
Luego: G_1 y G_2 serán equivalentes por que $L(G_1) = L(G_2)$

ARBOL SINTACTICO DE DERIVACION

Estructura gráfica que representa el conjunto de producciones de una derivación de una cadena.

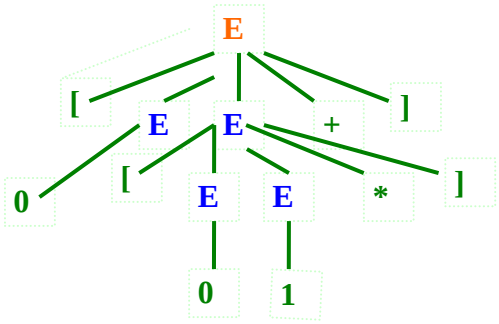
Sus componentes son:

Componentes	Representado por	Ejemplo
• **RAIZ**	• El **axioma**	La gramática **G =(Σ_T, Σ_N, S, P) = ({w, i, l, o }, { A, C }, A, P)** Producciones **P**: **A** ::= w **C** o **C** ::= i **C** l | **λ** Genera: **A**⇒w **C** o ⇒ wi **C** lo⇒ wi **λ**lo⇒ **wilo**
• **HOJAS**	• Símbolos **terminales** de **Σ_T**	
• **NODOS**	• Símbolos **no terminales** de **Σ_N**	
DERIVACIONES	• **Nodos sucesivos** de símbolos terminales y no terminales a partir del nodo raíz	



Ejemplo:
La gramática **G = (Σ_T, Σ_N, S, P) = ({ 0, 1, [,], +,* }, { E }, E, P)**, con las producciones
E ::= [E E +] | [E E *] | 0 | 1,

Pueden generar la palabra **[0 [0 1 *] +]**

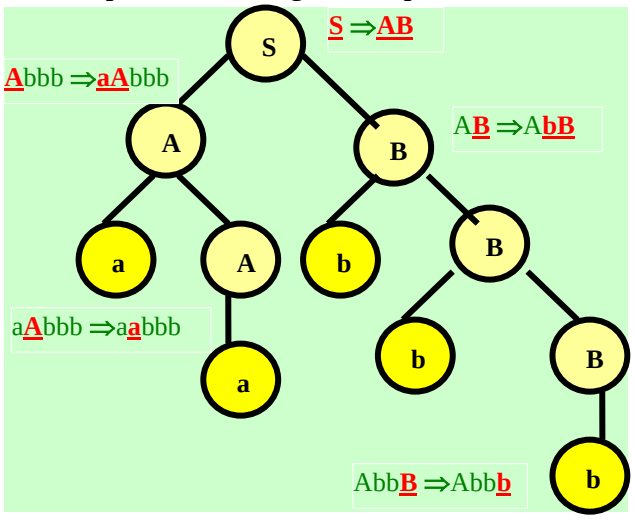


Ejemplo: La gramática **G = (Σ_T, Σ_N, S, P) = ({ a, b }, { S }, S, { S** ::= a **S** b \| **λ})**, genera la cadena **a a b b b** del lenguaje **LIC = { a^n b^m \| n, m ≥ 0 }**, aplicando las siguientes producciones:

S → A B
A → a A | a
B → b B | b

Las derivaciones posibles son:

- **S** ⇒**AB**
- A**B**⇒ A**bB**
- Ab**B**⇒ Ab**bB**
- Abb**B**⇒ Abb**b**
- **A**bbb⇒ **aA**bbb
- a**A**bbb⇒ a**a**bbb



Notar que para la cadena **a a b b b** de este árbol, además de la anterior derivación:

- **S ⇒AB⇒ AbB⇒ AbbB⇒ Abbb⇒ aAbbb⇒ aabbb** existen las siguientes derivaciones:
- **S ⇒AB⇒ aAB⇒ aaB⇒ aabB⇒ aabbB⇒ aabbb**
- **S ⇒AB⇒ aAB⇒ aAbB⇒ aAbbB⇒ aAbbb⇒ aabbb**

Por tanto:

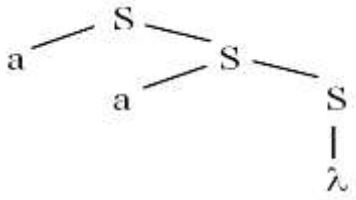
- **Derivación:** Es la secuencia de pasos que generan una palabra **x ∈ L(G)** a partir del axioma **S**.
- A cada derivación de una palabra producida por una gramática corresponde un árbol de la derivación, cuya raíz es la variable inicial o axioma.
- Los hijos de cada nodo son los elementos de la parte derecha de la regla de producción que se aplica en cada paso de la derivación.

Ejemplo: La palabra **x = a^2** del lenguaje **L = $\{a\}^2$**:
Puede ser generado mediante las producciones:

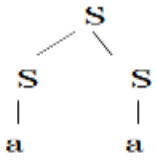
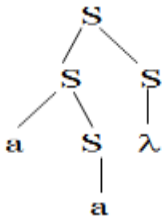
- P = { Regla1: **S → a S** | Regla2: **S → λ**}

$$S \overset{1}{\Rightarrow} a\ S \overset{1}{\Rightarrow} a\ a\ S \overset{2}{\Rightarrow} a\ a$$

El árbol es:



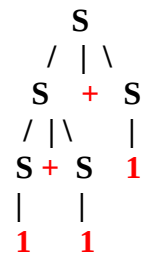
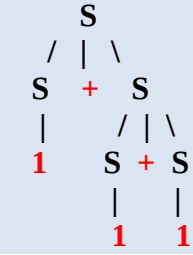
- **P = { 1: S→SS | 2: S→a | 3: S→λ }.**

Derivación	Arbol 1	Arbol 2
$S \overset{1}{\Rightarrow} S\ S \overset{2}{\Rightarrow} a\ S \overset{2}{\Rightarrow} a\ a$		
$S \overset{1}{\Rightarrow} S\ S \overset{1}{\Rightarrow} S\ S\ S \overset{2}{\Rightarrow} a\ S\ S \overset{2}{\Rightarrow} a\ a\ S \overset{3}{\Rightarrow} a\ a$		

- La 2da derivación muestra derivaciones que pueden efectuarse con diferentes longitudes de cadenas con el símbolo auxiliar, contando con la regla de producción 3 para eliminarlo.
- Para una misma longitud de cadena, se puede eliminar cualquiera de las tres S, lo que daría lugar a derivaciones diferentes.
- Un mismo árbol de derivación puede representar varias derivaciones diferentes. Ejemplo el árbol 1 también representa a la derivación

$$S \overset{1}{\Rightarrow} S\ S \overset{2}{\Rightarrow} S\ a \overset{2}{\Rightarrow} a\ a$$

Algunos analizadores, determinan el orden en el que el código será ejecutado; por ello, importa la derivación por la izquierda y por la derecha:

DERIVACION	Con gramatica: **S → S + S | S → 1**, de la cadena: "**1 + 1 + 1**"	ÁRBOL SINTÁCTICO
IZQUIERDA:	**S→S+S (1)** **S→S+S+S (1)** **S→1+S+S (2)** **S→1+1+S (2)** **S→1+1+1 (2)** **{{{1}S + {1}S}S + {1}S}S** Donde {...}S subcadena perteneciente a S	
DERECHA:	**S→ S + S (1)** **S→ 1 + S (2)** **S→ 1 + S + S (1)** **S→ 1 + 1 + S (2)** **S→ 1 + 1 + 1 (2)**	

AMBIGÜEDAD DE ÁRBOLES DE DERIVACIÓN

La gramática es ambigua si el lenguaje que define tiene alguna cadena que tenga más de un árbol de análisis sintáctico.

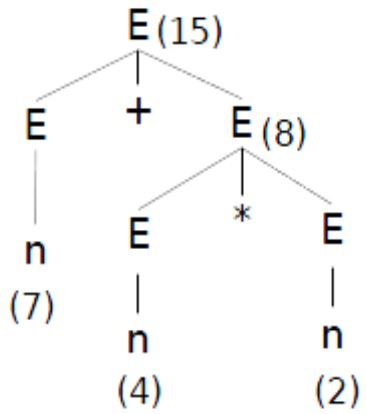
- No es posible construir analizadores eficientes para gramáticas ambiguas.
- Con ciertas restricciones a la gramática se podrá afirmar que no es ambigua.

Ejemplo:

Sea una gramática cuyas reglas de producción son las siguientes:

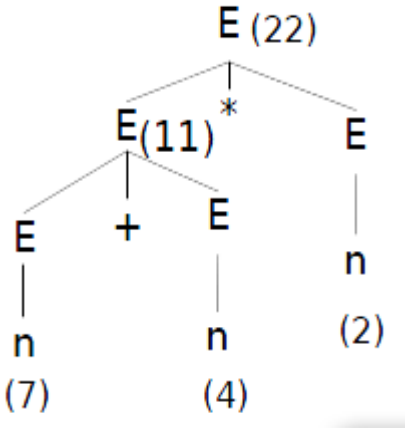
E → E + E
E → E * E
E → n
E → (E)

Se pide el o los árboles sintácticos de "7+4*2"



Gramática
E → E + E
E → E * E
E → n
E → (E)

Producción
E → E + E
→ n + E
→ n + E * E
→ n + n * E
→ n + n * n

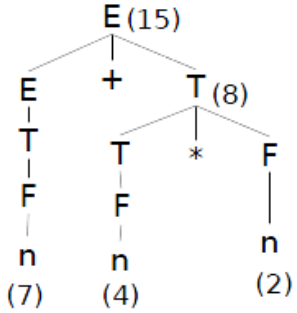


Gramática
E → E + E
E → E * E
E → n
E → (E)

Producción
E → E * E
→ E + E * E
→ n + E * E
→ n + n * E
→ n + n * n

Solución a la ambigüedad → modificar las reglas de producción.
Distinguiendo factor y término (producto de dos factores)

E → E + T | T
T → T * F | F
F → (E) | n
Árbol sintáctico de "7+4*2"



ÁRBOL TOTAL DE UN LENGUAJE

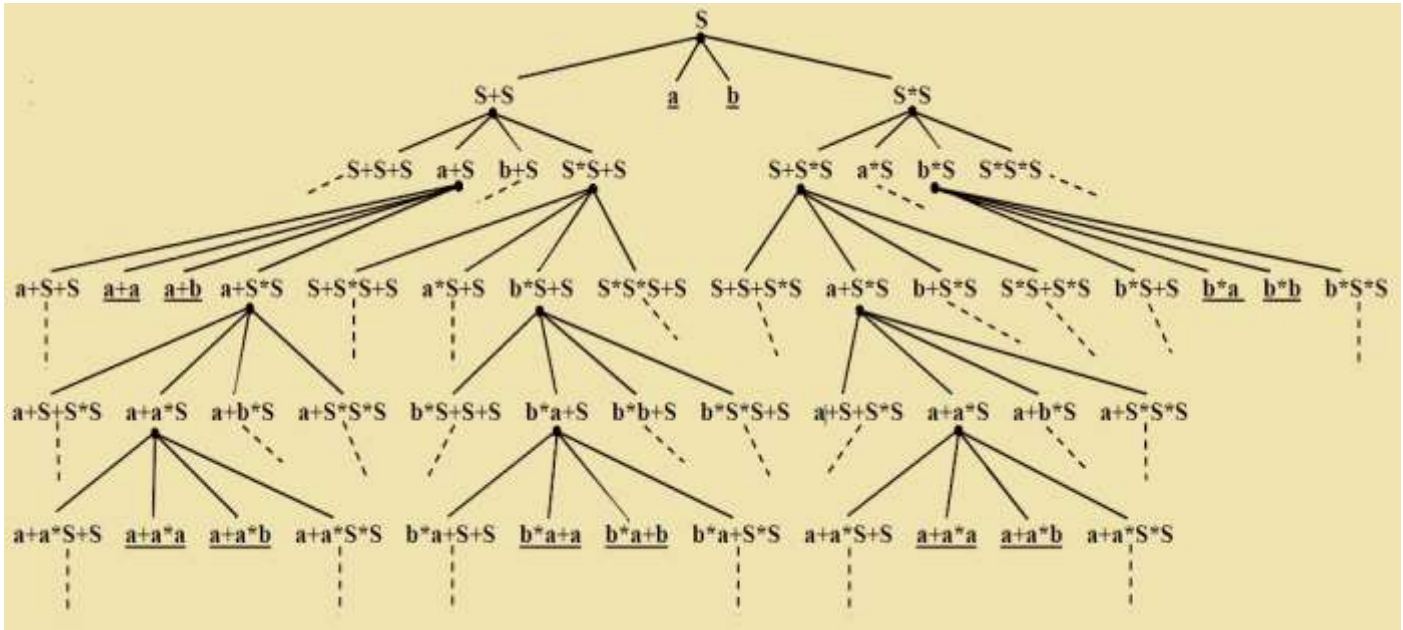
El conjunto de todas las secuencias que se puede obtener a partir de una gramática de tipo 2 o 3, ya sean sentencias (secuencias de terminales) o formas senténciales (secuencias de terminales y/o no-terminales), se pueden representar mediante un árbol del LIC o LR. También es posible plantear este árbol para tipo 1, pero en este caso puede haber ramificaciones que no conduzcan a ninguna sentencia.
Un árbol total del lenguaje generado por una GIC = { Σ_N, Σ_T, S, P }, posee las propiedades:

1) Cada nodo tiene una etiqueta.
2) La raíz tiene como etiqueta al axioma S.
3) La etiqueta de nodos que no son hojas debe estar en $((\Sigma_N \cup \Sigma_T)^* - \Sigma_T^*)$, y las de hojas en L(GIC).
4) Si un nodo "n" tiene etiqueta "α", y los nodos n_1, ... , n_m son sus hijos, con etiquetas β_1, ... , β_m respectivamente, entonces $\alpha \Rightarrow \beta_1 \mid ... \mid \beta_m$ o sea que para obtener los β_i se debe aplicar a α una sola regla de P.

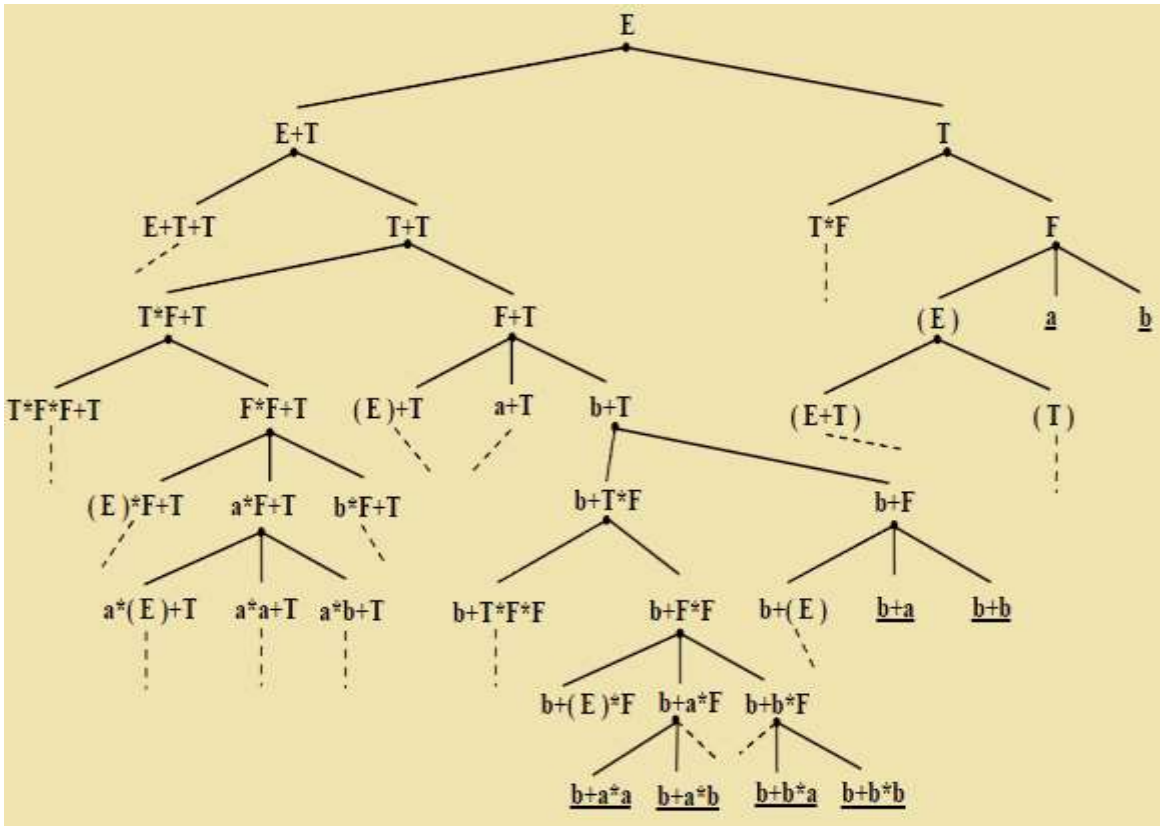
ÁRBOL TOTAL del lenguaje contiene a todas las palabras de dicho lenguaje. Luego si un árbol total es finito, el lenguaje correspondiente también lo será; si es infinito, en consecuencia su lenguaje será infinito y si no tiene hojas entonces el lenguaje asociado será ϕ.
Cada camino desde la raíz (axioma) hasta una hoja (palabra del lenguaje) representa el árbol de derivación de dicha palabra.
Para construir este árbol se segue un criterio fijo de derivación, por ejemplo "más a la izquierda", a fin de no aplicar combinaciones repetidas de reglas que llevarían a tener hojas repetidas. Al seguir este criterio y resultan hojas repetidas, o sea que hay varios caminos o derivaciones más a la izquierda o árboles de derivación para esa hoja, implica que la palabra es ambigua y por ende la GIC es ambigua.

Ejemplo:

$$P \begin{cases} S \rightarrow S + S \mid S * S \\ S \rightarrow a \mid b \end{cases}$$



$$P \begin{cases} E \rightarrow E + T \mid T \\ T \rightarrow T * F \mid F \\ F \rightarrow (E) \mid a \mid b \end{cases}$$

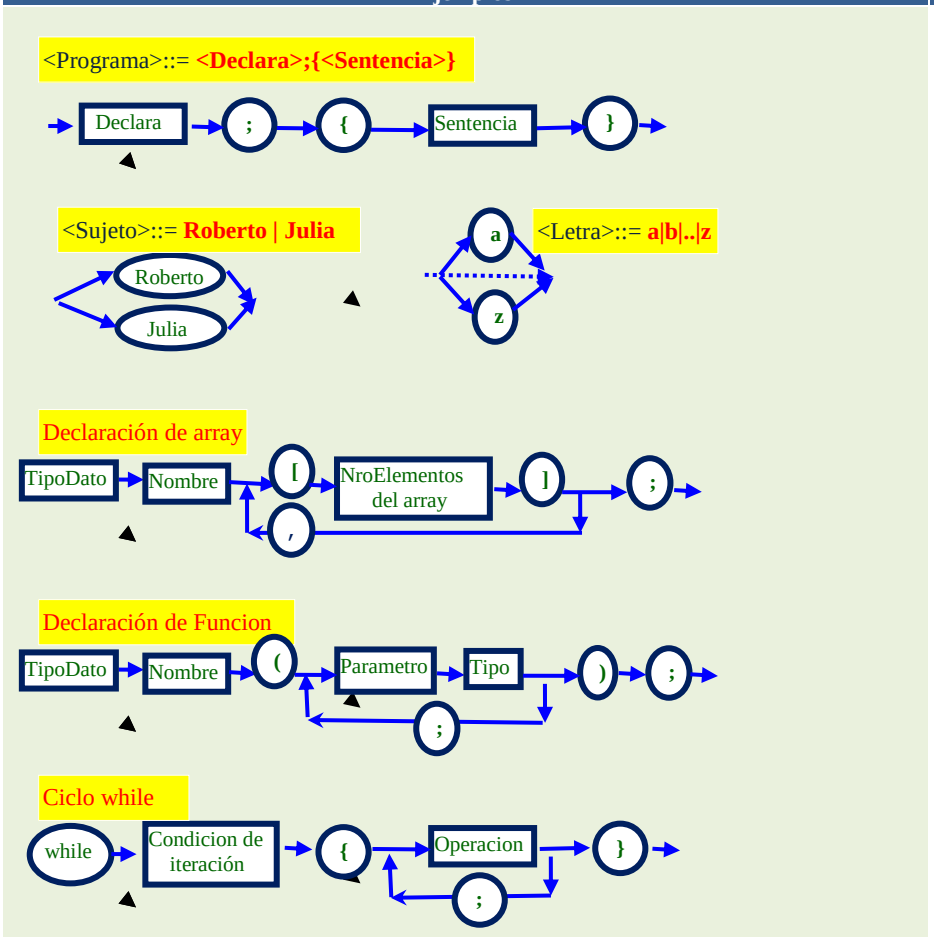


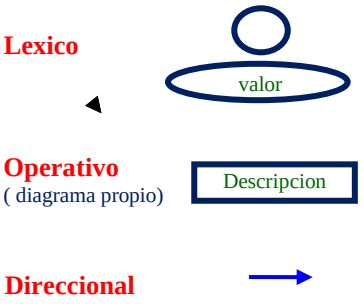
Representación de la SINTAXIS del lenguaje: Notación BNF: **B**ackus **N**aur **F**orm o **B**ackus **N**ormal **F**orm

Concepto	Componentes
Describe la sintaxis de gramáticas libres de contexto, con formatos que permiten descripciones exactas, como en los lenguajes de programación. La producción **BNF** usa un **metalenguaje** compuesta por **metasímbolos**, **formato** y **reglas**	**Metasímbolos**: • < > ::= | { } [ ] ( )
Ejemplo: Los siguientes BNF indican: • **<dirección postal>::=<nombre><dirección><apartado postal>** Dirección postal : nombre, seguido por la dirección y un apartado postal. • **<personal> ::= <nombre> | <inicial> "."** Personal es un nombre o inicial seguido(a) por un punto. • **<nombre>::=<apellido>[<trato>]<EOL>|<personal><nombre>** Nombre es un **apellido** seguido opcionalmente por un trato (Ing., Jr., Sr., Dr.) y un salto de línea (EOL: EndOfLine), o bien una parte **personal** seguida por un **nombre**. • **<dirección>::=[<dpto>]<númeroCasa><nombreCalle> <EOL>** Dirección es especificación opcional del departamento, seguido del número de casa, seguido por el nombre de calle, seguido por un salto de línea.	• **::= Definición**: El esquema de la derecha desarrolla al elemento de la izquierda **Ejemplo reglas**: **< digito > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9**
Ejemplo: Ciertas producciones pueden tener el formato: **<oración> ::= <sujeto> <predicado>** **<sujeto> ::= Roberto | Julia** **<predicado> ::= <verbo> <adverbio>** **<verbo> ::= maneja | corre** **<adverbio> ::= lento | rápido | mal**	• **< > No terminal**: Entre paréntesis angulares se representan los simbolos de Σ_N. **Ejemplo** **<oración>::=<sujeto><predicado>**
Ejemplo: En algún lenguaje de programación: **<identificador>::= <caracter> <sigue> | <simbolo><sigue>** **<caracter> ::= <letra> | <digito>** **<sigue> ::= <caracter> | <simbolo>** **<letra> ::= a | b | . . . | z** **<digito> ::= 1 | 2 | . . . | 0** **<simbolo> ::= , | ; | . | = |**	• **| Alternativa**: Se puede elegir únicamente uno de los elementos que separa. **Ejemplo** **<adverbio> ::= lento | rápido | mal**
Ejemplo: En algún lenguaje de programación: **<programa>::= <declara> ; {<sentencia>}** **<declara>::= <tipo> <nombre> <simbolo>** **<sentencia>::= <estructura> <accion>** **<tipo>::= char|boolean|int|float** **<nombre>::= Legajo|Alumno|Nota|Domicilio** **<estructura>::= while|do|for|if|case** **<simbolo> ::= [|] | , | ; | . | = |** **<accion>::=<condicion><operacion><simbolo>**	• **{ } Recursión**: La regla recursiva **C→αC|λ** representada en llaves: **C::={α}** **Ejemplo** **<programa> ::= <declara>;{<sentencia>}**
	• **[ ] Opción**: Los elementos que incluyen pueden utilizarse o no. **Ejemplo** **<docente> ::= [<trato>]<nombre><apellido>** **<trato> ::= Ing.|Dr.|Sr.|Sra.**
	• **( ) Agrupación**. Sirven para agrupar los elementos que incluyen **Ejemplo** **<declara>::=<tipo><nom>(<parametro><tipo>)**

Representación de la SINTAXIS del lenguaje **DIAGRAMA DE SINTAXIS:**
Representa gráficamente el nivel sintáctico de un lenguaje de programación.

Ejemplos	Componentes





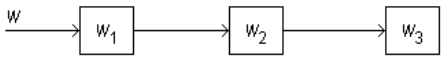
Ejemplo. Un identificador debe estar compuesto por letras y dígitos y debe comenzar con una letra, como muestra la siguiente gramática, con producciones dadas en BNF, con identificadores:

- G = (V, S, identificador, →)
- N = {identificador, resto, dígito, letra}
- S = {a, b, c, . . . , z, 0, 1, 2, 3, . . . , 9}, V = N U S
    1. <identificador> ::= <letra> | <letra> <resto>
    2. <resto> ::= <letra> | <dígito> | <letra> <resto> | <dígito> <resto>
    3. <resto> ::= a | b | c . . . | z
    4. <dígito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6| 7 | 8 | 9

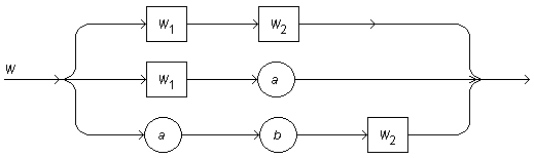
Observa que las producciones "resto → letra resto" y "resto →dígito resto" que aparecen en el enunciado 2 BNF, son recursivas y normales.

El diagrama de sintaxis usado para desplegar las producciones de gramáticas independientes de contexto y consiste en **una imagen de las producciones que permite ver las sustituciones en forma dinámica,** es decir, **verlas como un movimiento a través del diagrama**.

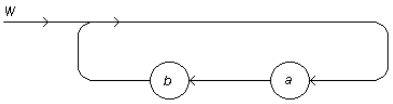
Los diagramas siguientes resultan de traducir conjuntos de producciones típicos, que son, por lo general, todas las producciones que aparecen en el lado derecho de algún enunciado **BNF** (**f**orma **B**ackus-**N**aur).



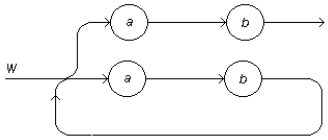
a) <w> ::= <w1> <w2> <w3>



b) <w> ::= <w1><w2> | <w1>a | bc<w2>



c) <w> ::= ab<w>.



d) <w> ::= ab | ab<w>.

FACTORIZACION A IZQUIERDAS:

Al diseñar una gramática, si se presentan producciones del mismo símbolo no terminal, en cuya parte derecha, la primera parte sea común; al compilarse se presentan dificultades que se eliminan mediante el siguiente algoritmo:

Por cada **A** $\in \Sigma_N$ Si $A ::= \beta\,\alpha_1 \mid \beta\,\alpha_2$ Se cambian esas producciones por $A ::= \beta\,A'$ y $A' ::= \alpha_1 | \alpha_2$

Ejemplo: En una gramática de un lenguaje de programación de alto nivel

Dado $\mathbf{G_{IC}} = (\Sigma_T, \Sigma_N, S, P)$ **=** ({If, Then, Else, Repeat, Until, Forever},{**S,C**},S,P**),** con las producciones:

- S::= If C Then S Else S | If C Then S | Repeat S Until C | Repeat S Forever

La factorización por izquierdas genera dos nuevos símbolos no terminales A' y S' y las producciones:

- S ::= If C Then SA' | Repeat SS'
- A' ::= Else S | λ
- S' ::= Until C | Forever

RECURSIVIDAD DE LENGUAJES: Define sentencias complicadas con un número pequeño de sencillas reglas de producción, así para expresar la estructura de un tren formado por una locomotora y un número cualquiera de vagones

tren → locomotora
tren → locomotora vagón
tren → locomotora vagón vagón

Necesitaríamos tantas reglas de derivación como número de vagones posibles tenga el tren, desventaja que podemos ev-itar si usamos un par de sentencias recursivas de la siguiente manera:

- Definimos la regla base no recursiva, definiendo el concepto inicial: tren → locomotora
- Definimos una o más reglas recursivas que permiten el crecimiento ilimitado de la estructura partiendo del concepto inicial anterior. tren → tren vagón

La gramática es recursiva, si permite derivar cadenas donde vuelven a aparecer el símbolo no terminal de la parte izquierda de la regla de derivación. **A** => α**A**β. Cuando aparecen derivaciones como A => **α**A ó A => A**α** se denominan recursividad izquierda y derecha, respectivamente.

La recursividad de un lenguaje se puede definir por un proceso que abarca tres etapas, como podemos apreciar en el siguiente ejemplo, que permite determinar los números pares:

Proceso Recursivo	Ejemplo		
Etapa	**Definición**	**Reglas de recursividad:**	**Probar que 8 es par**
1. Especificar algunos **objetos primarios** del conjunto	PAR = {Conjunto de números enteros divisibles por 2}	El número 2 es PAR.	El número 2 pertenece a PAR
2. A partir de los objetos primarios, **diseñar reglas** para generar más objetos del conjunto	PAR = { x = 2n | n = 1, 2, 3,...}	Si el número x pertenece a PAR, entonces x + 2, también es PAR	✓ 2 + 2 = 4 pertenece a PAR ✓ 4 + 2 = 6 pertenece a PAR ✓ 6 + 2 = 8 pertenece a PAR
3. Solo los objetos construidos de este modo perteneceran al conjunto	El conjunto PAR está definido por 3	Los elementos del conjunto PAR son los producidos por las dos reglas anteriores	

Representación de la SEMANTICA del lenguaje: ESQUEMA DE TRADUCCION: (ET)

Definición	Ejemplo
Es una G_{IC}, en cuyas partes derechas de sus reglas de producción pueden insertarse fragmentos de código de programa de **ACCIONES SEMÁNTICAS**, que actúan, calculan y modifican los atributos asociados con los nodos del árbol sintáctico. ○ La aplicación de la **regla de sintaxis** implica efectuar una **acción semántica** (Entre llaves). **Regla de Sintaxis** → **Acción Semantica** **N → β** : **{fragmento de codigo}** La acción semántica se ejecuta anticipando la definición del valor del identificador de variable, su terminación permanece pendiente hasta que la ejecución de una acción semántica le asigne un valor. Para aplicar un ET, construir el árbol sintáctico y luego aplicar acciones empotradas en reglas en orden de recorrido primero-profundo. Ej: En la regla **A→αβ** el nodo asociado con una producción: deducimos un fragmento de código: **A → α {action}β** La acción **{action}** se ejecuta después de todas las acciones asociadas con el recorrido del subárbol de **α** y antes que todas las acciones asociadas con el recorrido del subárbol **β**. Este ET recibe como entrada una expresión en infijo y produce como salida su traducción a postfijo para expresiones aritméticas con sólo restas de números: Las variables sintácticas en la regla de producción se indexan, para distinguir de que nodo del árbol de análisis estamos hablando. Para el atributo del nodo usamos indexación tipo hash, donde: • **VAL** es atributo de nodos tipo **NUM** denota su valor numérico y para accederlo escribimos **$NUM{VAL}**. • **$expr{TRA}** denota el atributo "traducción" de los nodos de tipo **expr**.	**Formato del Esquema de Traducción:** **Regla de Sintaxis** → **Acción Semantica** **N → β** : **{fragmento de codigo}** **Ejemplo:** T→ char : {$T_{.Tipo}$=4} T→ boolean : {$T_{.Tipo}$=3} A→ λ : {$A_{.Rango}$=vacio} P→ D;{S} : {Flap=0} S→ W : {$S_{.Tipo}$= $W_{.Tipo}$, $S_{.Bifu}$= $W_{.Bifu}$} S→ I : {$S_{.Tipo}$= $I_{.Tipo}$, $S_{.Bifu}$= $I_{.Bifu}$} . . . . : . . . . S→while Tdo{SF : {Si($T_{.Tipo}$==3)entonces $W_{.Tipo}$= $S_{.Tipo}$ sino $W_{.Tipo}$= error.t $W_{.Bifu}$= $S_{.bufu}$, Flap= Flap +1} I→if T then {S}R : {Si($T_{.Tipo}$==3)^($T_{.Tipo}$==ok) ^($T_{.Tipo}$==ok) entonces $I_{.Tipo}$= ok sino $I_{.Tipo}$= error.t Si($S_{.Bifu}$==ok)^($R_{.Bifu}$==ok) entonces $I_{.Bifu}$= ok sino $I_{.Bifu}$= error.b} R→else {S} : {$R_{.Tipo}$= $S_{.Tipo}$, $R_{.Bifu}$= $S_{.Bifu}$ } A→ λ : {$R_{.Tipo}$= ok, $R_{.Bifu}$= ok } R→} : {Flap = Flap -1} **expr→expr$_1$ –NUM** **{$expr{TRA}=$expr[1]{TRA}." ".$NUM{VAL}."-"}** **expr→NUM** **{ $expr{TRA}= $NUM{VAL} }**
Al evaluar la acción, además de usar **variables bandera** cuyos valores finales indican si la sentencia es correcta semánticamente, suele **comprobarse**: ▪ **Tipos**: Verifica la compatibilidad entre el tipo de dato y su operador.	• **La derecha e izquierda de la asignación deben ser de tipos compatibles** • **La condición del bucle DO debe ser tipo boolean.**
▪ **Flujo de control**: Controla que la acción de bifurcaciones o selecciones simples o múltiples de flujos operativos, tengan definidos los direccionamientos correspondientes.	• **El comando BREAK debe estar dentro del ámbito del bucle DO**

ESQUEMA DE TRADUCCIÓN:

El esquema de traducción es una gramática que asocia a los símbolos de la gramática atributos, caracterizados por identificadores o nombre y un tipo o clase (En el ejemplo se inserto acciones, en código Perl/Python/C, en medio de las partes derechas).

El orden en que se evalúan los fragmentos de código es el recorrido primero-profundo del árbol de análisis sintáctico. Específicamente, considerando a las acciones como hijos-hoja del nodo, el recorrido que realiza un esquema de traducción es:

```
1     sub esquema_de_traduccion {
2       my $node = shift;
3
4       for my $child ($node->children) { # de izquierda a derecha
5         if ($child->isa('ACTION') {
6           $child->execute;
7         }
8         else { esquema_de_traduccion($child) }
9       }
10    }
```

El bucle de la línea 4 recorre a los hijos de izquierda a derecha y para que un esquema de traducción funcione, es condición la producción aumentada con acciones, de la forma

$$A \rightarrow X_1 \ldots X_j \ \text{action}\{ (\$A\{b\}, \$X^1\{c\} \ldots X^n\{d\}) \} \ X_{j+1} \ldots X_n$$

Debe ocurrir que los atributos evaluados en la acción insertada después de X_j dependan de atributos y variables que fueron computadas durante la visita de los hermanos izquierdos o de sus ancestros. En particular no deberían depender de atributos asociados con las variables $X_{j+1} \ldots X_n$. Ello no significa que no sea correcto evaluar atributos de $X_{j+1} \ldots X_n$ en esa acción.

Representación de la SEMANTICA del lenguaje

SISTEMAS CANONICOS:

Conceptos	Ejemplo
El sistema canónico es el conjunto de argumentos de la REGLA ó CANON para definir un lenguaje formal en nivel sintáctico como semántico. Donde: • **Axioma:** Es la regla sin premisa • **Terminal:** Símbolo léxico • **Variable:** Símbolo reemplazable. • **Termino:** Serie de terminales y variables concatenados • **Predicado:** Denominación del conjunto de términos. • **Grado:** Cantidad de términos del predicado. • **Remarca:** Término seguido de predicado y cada **premisa** o conclusión del canon es remarca.	Generación de palíndromos ▪ Alfabeto: **A**={**a,b,c**} : **a,b,c** terminales ▪ Axiomas: **a, b, c, aa, bb, cc** ▪ Producciones: **$** → **a$a** donde **$** es una variable **$** → **b$b** donde **b$b** es un término. **$** → **c$c**

En el ámbito de la inteligencia artificial:
- **La inteligencia** se asocia con "regularidades" y
- **El comportamiento** inteligente se asocia a ejecutar **sistemas de reglas o producciónes**.
  Se supone que en el proceso de memoria humano, **la memoria** a:
  - **Corto plazo** usa las deducciones intermedias y
  - **Largo plazo** usa las **producciones ó reglas**.

PROPIEDADES DE LAS REGLAS:
- **MODULARIDAD**: La regla define un pequeño independiente pedazo de conocimiento
- **INCREMENTALIDAD**: Nuevas reglas pueden añadirse a la base de conocimiento relativamente independiente de las demás
- **MODIFICABILIDAD**: Por la modularidad, las reglas viejas pueden ser modificadas
- **TRANSPARENCIA**: Habilidad de explicar sus decisiones y soluciones

UN SISTEMA DE PRODUCCIÓN tiene:
- UN CONJUNTO DE REGLAS: Base de conocimiento.
- UN INTERPRETE DE REGLAS: O máquina de inferencia, que decide la regla aplicar, controla la actividad del sistema.
- UNA MEMORIA DE TRABAJO: Que guarda los datos, metas, y resultados intermedios

Las reglas de producción manipulan estructuras de símbolos, como listas o vectores.
Se tiene un conjunto:
- **N** de nombres de objetos en el dominio
- **P** de propiedades que representan atributos de los objetos
- **V** de valores que los atributos
  Suele usarse una tripleta: (objeto atributo valor).

Las reglas pueden ser: **P1, . .Pm →Q1, . .Qn**. Que significa:

IF las condiciones **P1** y **P2** y . . . **Pm** se cumplen
THEN realiza las acciones **Q1** y **Q2** . . .y **Qn**.

Ejemplo:

IF Animal es_un carnivoro AND
Animal color cafe AND
Animal tiene rayas
THEN Animal es tigre

CHOMSKY: JERARQUIA GRAMATICAL: Gramatica es la 4-Tupla: $G = (\Sigma_T, \Sigma_N, S, P)$

ELEMENTO	Concepto:
Σ_T:Alfabeto símbolo **Terminales**	Conforman el formato no variable o final de la palabra
Σ_N:Alfabeto de. **No terminales**	Conforman las **variable** que representa un estado intermedio de formación de la palabra: $\Sigma=\Sigma_T\cup\Sigma_N$ y $\Sigma_T\cap\Sigma_N=\phi$
S Símbolo inicial: **Axioma**	Para generar cada palabra del lenguaje: $S \in \Sigma_N$
P Regla Sustitución: **Produccion**	Conjunto finito de transformaciones para alcanzar el estado final de la palabra formada. A partir del axioma **S** se detecta cada símbolo no terminal y se efectua la producción para transformarla en terminal.

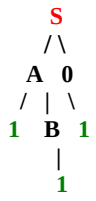
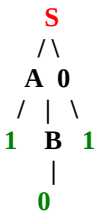
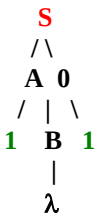
Producción para la gramatica	Descripción
G_0: **IRRESTRICTA** G_I **P** $=\{(u::=v) \mid u = xAy, u\in\Sigma^+, v,x,y\in\Sigma^*, A\in\Sigma_N\}$	La parte **izquierda tiene al menos un símbolo no terminal (Σ_N);** en la derecha no plantea restricciones. Su formato es del tipo: $\alpha\rightarrow\beta$, donde: ▪ α: Cadena no vacía de no terminales y/o terminales. ▪ β: Cadena cualquiera de terminales y/o no terminales: La parte: **Izquierda** no puede ser la palabra vacía λ, así no hay producciones: $\lambda\rightarrow\beta$. **Derecha** puede ser cadena vacía λ, así hay producciones de la forma $\beta\rightarrow\lambda$ **Ejemplo: G_I: $G_0 = (\Sigma_T, \Sigma_N, S, P) = (\{0,1\}, \{A, B, S\}, S, P)$** Producciones **P**: **S**::=A0 **A**0::=1B1 1**A**::=AB0 **B**::=λ| 1 | 0 Derivacion: S→A0→1B1→**111**, S→A0→1B1→**101**, S→A0→1B1→1λ1→**11** Genera el lenguaje: $L_I = \{ w^n \mid w = w^{-1} \} = \{111, 101, 11, \ldots\}$,
G_1: **DEPENDIENTE del CONTEXTO** G_{DC} **P**$=\{(S::=\lambda)$ ó $(xAy::=xvy) \mid x,y\in\Sigma^*, A\in\Sigma_N, v\in\Sigma_T{}^+\}$	La **longitud de parte derecha no puede ser menor que la parte izquierda**. Su formato es: α **A** $\beta\rightarrow\ \alpha\gamma\beta$ donde: **A** Símbolo no terminal ▪ α, β y γ: Cadenas de terminales y no terminales donde el **contexto**: $\alpha, \beta\in(\Sigma_N\cup\Sigma)^*$ y $\lvert\alpha\rvert\leq\lvert\beta\rvert$ α y β pueden ser vacíos, pero $\gamma\neq\lambda$. ▪ Admite la **regla compresora** en el axioma: $S ::= \lambda$; **Ejemplo: G_{DC}: $G_1 = (\Sigma_T, \Sigma_N, S, P) = (\{0,1\}, \{B, S\}, S, P)$** Producciones **P**: **S** ::= 1B1 | 11; **1B1**::=101 |111 Derivacion: S→**11**, S→1B1 → **101**, S→1B1→ **111** Genera el lenguaje: $L_I = \{ w^n \mid w = w^{-1} \} = \{111, 101, 11, \ldots\}$
G_2: **INDEPENDIENTE del CONTEXTO** G_{LC} **P** $=\{(S ::= \lambda)$ **ó** $(A ::= v) \mid A \in \Sigma_N, v \in \Sigma_T{}^+\}$	✓ La **longitud de la izquierda de la producción debe ser 1 y puede tener un solo símbolo no terminal.** ✓ La derecha permite pares del producto Σ_N x Σ^* $(\Sigma_N \cup \lambda)$, así puede tener cualquier numero de símbolos terminales o no terminales. ✓ No considera contexto del símbolo a sustituir. La regla de producción es de forma: **A → v**, donde **A** : Símbolo no terminal y **v** : Cadena de terminales y/o no terminales. **Ejemplo: G_{DC}: $G_1 = (\Sigma_T, \Sigma_N, S, P) = (\{0,1\}, \{A, B\}, A, P)$** Producciones **P**: **A** ::= 1B1 | 11; **B** ::= 1 | 0 Derivacion: A→11; A→1B1→**101**; A→1B1→ **111** Genera el lenguaje: $L_I = \{ w^n \mid w = w^{-1} \} = \{111, 101, 11, \ldots\}$,
G_3:**REGULAR** o **LINEAL** ▪ **Por DERECHA** G_{RED}: $P \subseteq \Sigma_N x \Sigma_T{}^*(\Sigma_N\cup\lambda)$ **P** = { (S::=λ) ó (A::= a**B**) ó (A::=a)|A,B∈Σ_N, a∈Σ_T+} ▪ **Por IZQUIERDA** G_{REI}:$P \subseteq \Sigma_N x(\Sigma_N \cup \lambda)\Sigma_T{}^*$ **P** = {(S::=λ) ó (A::= **B**a) ó (A::=a)|A,B∈Σ_N, a∈Σ_T+}	La **longitud de la izquierda debe ser igual a 1 y no terminal**, la parte derecha puede ser una secuencia de solo terminales, o de terminales con un no terminal como sufijo o como prefijo. **Formato estándar**: $N_1\rightarrow tN_2$, N→**t** , S→λ **Ejemplo: G_I: $G_0 = (\Sigma_T, \Sigma_N, S, P) = (\{0,1\}, \{A, B\}, A, P)$** Producciones: P : **A** ::= 1**B** **B**::= 1 |0**C**| 1**C** **C** ::= 1 Derivacion: A→1B→**11**, A→1B→10C→**101**, A→1B→11C→**111** Genera el lenguaje: $L_I = \{ w^n \mid w = w^{-1} \} = \{111, 101, 11, \ldots\}$,

GRAMATICA tipo G_0 : IRRESTRICTA G_I

Definición	Componentes
Gramatica estructurada por frases, que no tiene ninguna restricción. Es la cuatrupla: $G_0 = G_I = (\Sigma_T, \Sigma_N, S, P)$	• Σ_T: Alfabeto de **símbolos terminales**. • Σ_N: Alfabeto de **símbolos no terminales**, donde $\Sigma=\Sigma_T \cup \Sigma_N$ y $\Sigma_T \cap \Sigma_N=0$ • **S**: Símbolo Inicial o **Axioma** gramátical, es símbolo no terminal: $S \in \Sigma_N$ • **Producciones:** $P = \{(u ::= v) \mid u = xAy,\ u \in \Sigma^+, v,x,y \in \Sigma^*, A \in \Sigma_N\}$ • La **izquierda tienen al menos un símbolo no terminal (Σ_N)**; • La **derecha no plantea restricciones**. Su formato es del tipo: $\alpha \rightarrow \beta$, donde: ▪ α: Cadena no vacía de no terminales y/o terminales. ▪ β: Cadena de terminales y/o no terminales: Las producciones de estas gramáticas, la parte**:** ○ **Izquierda** no puede ser la palabra vacía **λ,** así **no** hay producciones: $\lambda \rightarrow \beta$**.** ○ **Derecha** puede ser cadena vacía **λ,** así hay producciones de la forma $\beta \rightarrow \lambda$**.**

Ejemplo:

La G_I: $G_0 = (\Sigma_T, \Sigma_N, S, P) = (\{ 0,1 \}, \{ A, B, S \}, S, P)$ con:

Producciones		Derivaciones		
Pares	Simplificado	S→A0→1B1→**111**	S→A0→1B1→**101**	S→A0→1B1→1λ1→**11**
P={(S ::=A0), (A0::=1B1), (1A::=AB0), (B::= λ), (B::=1), (B::=0)}	P : S::=A0 A0::=1B1 1A::=AB0 B::=λ| 1 | 0			

Genera el **lenguaje irrestricto** o **recursivamente enumerable**: $L_I = \{ w^n \mid w = w^{-1} \} = = \{ 111,\ 101,\ 11, , . \}$

Ejemplo:

La G_I: $G_0 = (\Sigma_T, \Sigma_N, S, P) = (\{a, i, o, u, c, f, l, n, p, r, t, w \}, \{ A, B, S \}, S, P)$ con:

Producciones	Derivaciones
S ::= cBo | wAi **B**o ::= aAi | pio | lo | tnB| rtB | λ **A**i ::= rBo | iBo | c**B** ::= uBo | fBo	• **S** → w**Ai** → wi**Bo** → wilo • **S** → c**Bo** →ca**Ai** → car**Bo** → carpio • **S** → c**Bo** → u**Bo**o → utn**Bo** → utnλ → utn • **S** → c**Bo** → f**Bo**o → frt**Bo** → frtλ → frt

Lenguaje generado: L_{IC} = { **wilo, carpio, utn, frt** , . . }

GRAMATICA tipo G_1 : DEPENDIENTE del CONTEXTO G_{DC}

Definición	Componentes
Es gramatica estructurada por frases, llamada también sensible al contexto. Definida como la cuatrupla:	• Σ_T: Alfabeto de **símbolos terminales**.
	• Σ_N: Alfabeto de **símbolos no terminales**, donde $\Sigma=\Sigma_T\cup\Sigma_N$ y $\Sigma_T\cap\Sigma_N=0$
	• **S**: Símbolo Inicial o **Axioma** gramátical, es un símbolo no terminal: $S\in\Sigma_N$
$G_1=G_{DC}=(\Sigma_T,\Sigma_N,S,P)$	• **Producciones:** $P=\{(S::=\lambda)$ **ó** $(xAy::=xvy)\mid x,y\in\Sigma^*, A\in\Sigma_N, v\in\Sigma^+\}$ Su formato es: $\alpha\ A\ \beta \rightarrow \alpha\ \gamma\ \beta$ donde: ▪ **A** Símbolo no terminal ▪ α, β y γ: Cadenas de terminales y no terminales donde: $\alpha,\beta\in(\Sigma_N\cup\Sigma)^*$ ▪ La **longitud de la derecha no puede ser menor que la izquierda**; $\lvert\alpha\rvert \le \lvert\beta\rvert$ ▪ α y β Pueden ser vacíos, pero $\gamma \neq \lambda$, la cadena vacía λ no puede ser generada en un L_{DC}. ▪ Considera el símbolo ubicado antes y despues del símbolo a sustituir, así en **xAy** los símbolos **x**, **y** son el **contexto**. ▪ Admite la **regla compresora** en el axioma: **S ::= λ**; así, la longitud de la derecha puede ser igual o mayor que de la izquierda.

Ejemplo: La G_{DC}: $G_1 = (\Sigma_T, \Sigma_N, S, P) = (\{0,1\}, \{B, S\}, S, P)$ con las:

Producciones		Derivaciones		
Pares	Simplificado	S → 11	S→1B1 → 101	S→1B1→ 111
P={ (S::= 1B1), (S::= 11), (1B1::=101), (1B1::=111)}	P : S ::= 1B1 | 11 1B1::=101 |111	S / \\ 1 1	S / | \\ 1 B 1 | \\ 0 1	S / | \\ 1 B 1 | 1

Lenguaje generado: $L_{DC} = \{ w^n \mid w = w^{-1} \} = \{11, 101, 111, \ldots\}$

Ejemplo: La G_{DC}: $G_1 = (\Sigma_T, \Sigma_N, S, P) = (\{a,b,c\}, \{B, D, T, S\}, S, P)$, con:

Producciones		Derivaciones
Pares	Simplificado	
P={ (S::= T), (D::= c), (BD::=BD), (T::=aTBD) (T::=abD) (bB::=BB) }	P : S ::= T D ::= c BD ::= DB T ::= aTBD|abD bB ::= BB	S → T → abD → **abc** S → T → aTBD → aabDBD → aabBDD → aabbDD → aabbcD → **aabbcc** S → T → aTBD → aaTBDBD → aaabDBDBD → aaabBDDBD → aaabBDDBD → aaabbDDBD → aaabbDBDD → aaabbBDDD → aaabbbDDD → **aaabbbccc**

Lenguaje generado: $L_{DC} = \{ a^n b^n c^n \mid n \ge 1\} = \{ a^1b^1c^1, a^2b^2c^2, a^3b^3c^3, \ldots, a^n b^n c^n \}$

Ejemplo: La G_{DC}: $G_1 = (\Sigma_T, \Sigma_N, S, P) = (\{a, i, o, u, c, f, l, n, p, r, t, w\}, \{A, B, S\}, S, P)$ con:

Producciones		Derivaciones
Pares	Simplificado	
P={(S::=cBo), (Bo::= wAi),(Bo::= pio) (Bo::= lo),(Bo::=tnB) (Ai::=rBo), (Ai::=iBo) (cB::=uBo), (cB::=fBo) (cBo::=uBo), (cBo::=frt)}	S ::=cBo| wAi Bo::=aAi|pio|lo|tnB Ai::=rBo|iBo| cB::=uBo|fBo cBo::=uBo|frt	S → c**B**o →ca**A**i → car**B**o → **carpio** S → w**A**i →wi**B**o → **wilo** S → c**B**o → u**Bo**o → utn**Bo** → utnλ→ **utn** S → c**B**o → **frt**

Lenguaje generado: L_{IC} = { **carpio**, **wilo**, **utn**, **frt** , . . }

GRAMATICA tipo G_2 : INDEPENDIENTE del CONTEXTO G_{IC}

Describe en notación BNF la sintaxis de lenguajes de programación. Simplifica el diseño de algoritmos eficientes de análisis sintáctico

Definición: Gramatica estructurada por frases, **libre del contexto**.

Cuatrupla: $G_2=G_{IC} = (\Sigma_T, \Sigma_N, S, P)$ donde:

- Σ_T: Alfabeto de **símbolos terminales**.
- Σ_N: Alfabeto de **símbolos no terminales**, donde $\Sigma = \Sigma_T \cup \Sigma_N$ y $\Sigma_T \cap \Sigma_N = 0$
- **S**: Símbolo Inicial o **Axioma** de la gramática, es un símbolo no terminal: $S \in \Sigma_N$
- **P**: **Producciones**: $P = \{ (S ::= \lambda)$ ó $(A ::= v) \mid A \in \Sigma_N, v \in \Sigma_T + \}$
  - La **longitud de la izquierda debe ser 1** y **puede tener un solo símbolo no terminal**.
  - A la derecha admite pares del producto Σ_N x $\Sigma^* (\Sigma_N \cup \varepsilon)$, así puede tener cualquier numero de símbolos terminales o no terminales.
  - La regla de producción es de forma: $A \rightarrow v$, donde:
    - **A** : Símbolo no terminal
    - **v** : Cadena de terminales y/o no terminales: Es **libre de contexto** por que **A** puede sustituirse por **v** sin considera el contexto

Un lenguaje formal es **lenguaje libre de contexto** L_{IC} si hay una gramatica estructurada por frases G_{IC} que lo genera.

Ejemplo: La G_{IC}: $G_2 = (\Sigma_T, \Sigma_N, S, P) = (\{0,1\}, \{ A, B \}, A, P)$ con:

Producciones		Derivaciones		
Pares	Simplificado	A → 11	A →1B1 → 101	A →1B1→ 111
P={(A::=1B1), (A::= 11), (B::= 1), (B ::= 0) }	A ::= 1B1 | 11 B ::= 1 | 0	A / \\ 1 1	A / | \\ 1 B 1 | 0	A / | \\ 1 B 1 | 1

Lenguaje generado: $L_{IC} = \{ w^n \mid w = w^{-1} \}$ = { 11, 101, 111 , . . }

DISEÑO de la G_{IC}, por:

1) DEFINICIÓN

Ejemplo: La G_{IC}: $G_2 = (\Sigma_T, \Sigma_N, S, P) = (\{a, i, o, u, c, f, l, n, p, r, t, w \}, \{ A, B, S \}, S, P)$ con:

Producciones		Derivaciones
Pares	Simplificado	
P={(S::=cBo), (S::= wAo) (S::= utn), (S ::= frt) (A::=rBi), (A::=il) (B::=aAo), (B::=p) }	S ::= cBo | wAo | utn | frt A ::= rBi | il B ::= aAo | p |	S → cBo →caAo → carBio → **carpio** S → wAo → **wilo** S → **utn** S → **frt**

Lenguaje generado: L_{IC} = { **carpio**, **wilo**, **utn**, **frt** , . . }

Ejemplo: La G_{IC}: $G_2 = (\Sigma_T, \Sigma_N, S, P) = (\{a, b \}, \{ S \}, S, P)$ con:

Producciones		Derivaciones
Pares	Simplificado	
P={(S::=aSb), (S::= ε) }	S ::= a S b | ε	S → aSb → ab = $a^1 b^1$ S → aSb → aabb = $a^2 b^2$ S → aSb → a aSb b → a aabb b = $a^3 b^3$ S → aSb → a aSb b → a a aSb b b → a a aabb b b = $a^4 b^4$

Genera el: $L_{IC} = \{a^n b^n \mid n \geq 0 \}$ = { $a^1 b^1$, $a^2 b^2$, $a^3 b^3$, $a^4 b^4$, . . }

2) ADAPTACIÓN

Ejemplo: Para adaptar la $G_{IC}= (\Sigma_T, \Sigma_N, S, P) = (\{a,b\},\{S\}, S, P)$ del ejemplo anterior, que generó: $L_{IC}= \{a^n b^n \mid n\geq 0 \}$,

Ejemplo: La G_{IC}: $G_2 = (\Sigma_T, \Sigma_N, S, P) = (\{a, b \}, \{ S \}, S, P)$ con:		
Producciones		Derivaciones
Pares	Simplificado	
P={(S::=aSb), (S::= ε) }	S ::= a S b | ε	S → aSb → ab = $a^1 b^1$ S → aSb → aabb = $a^2 b^2$ S → aSb → a aSb b → a aabb b = $a^3 b^3$ S → aSb → a aSb b → a a aSb b b → a a aabb b b = $a^4 b^4$

Genera el: $L_{IC} = \{a^n b^n \mid n\geq 0 \} = \{ a^1 b^1, a^2 b^2, a^3 b^3, a^4 b^4, \ldots \}$

Ejemplo: A la G_{IC}: $G_2 = (\Sigma_T, \Sigma_N, S, P) = (\{a, b \}, \{ S \}, S, P)$ agregamos las reglas necesarias:		
Adaptación de las Producciones		Derivaciones
Pares	Simplificado	
P={(S::=aSb), (S::=R), (S::=aS),(R::=ε)}	S::=aSb|aS|R R::= ε	S → aSb →aaSb→ aaRb = $a^2 b$ S →aSb→aaSbb→aaaSbb→aaaRbb→aaabb=$a^3 b^2$ S →aSb→aaSbb→aaaSbbb→ aaaaSbbb→ aaaaRbbb = $a^4 b^3$

Genera el: $L_{IC} = \{ a^n b^m \mid n, m \geq 0, n>m \} = \{ a^2 b^1, a^3 b^2, a^4 b^3, \ldots \}$

3) MEZCLA

Ejemplo: Sea $G_{IC1} = (\Sigma_T, \Sigma_N, S, P) = (\{ a,b\},\{S_1\}, S, P)$:		
Producciones		Derivaciones
Pares	Simplificado	
P={(S_1::=a S_1b), (S_1::= ε) }	S_1 ::= a S_1 b | ε	S_1→ a S_1b → aabb = $a^2 b^2$ S_1→ a S_1b → aaS_1bb → a aabb b = $a^3 b^3$ S_1→ aS_1b→aaS_1bb→aaa S_1bbb→aaaabbbb = $a^4 b^4$

Lenguaje generado: $L_{IC} = \{ a^n b^n \mid n \geq 0 \} = \{ a^1 b^1, a^2 b^2, a^3 b^3, a^4 b^4, \ldots \}$

Luego, sea $G_{IC2} = (\Sigma_T, \Sigma_N, S, P) = (\{ a,b \},\{S_2\}, S, P)$:		
Producciones		**Derivaciones**
Pares	Simplificado	
P={(S_2::=aSbb), (S_2::= ε) }	S_2::= a S_2 bb | ε	S_2 → a S_2bb→aab = $a^1 b^2$ S_2 → a S_2bb→aa S_2bbbb→aabbb b = $a^2 b^4$ S_2 →aS_2bb→aaS_2bbbb→aaaS_2bbbbbb→aaabbbbbb = $a^3 b^6$

Lenguaje generado: $L_{IC} = \{ a^n b^{2n} \mid n \geq 0 \} = L_{IC} = \{ a^1 b^2, a^2 b^4, a^3 b^6, a^4 b^8, \ldots \}$

Al mezclar G_{IC1} con G_{IC2} :		
Producciones		Derivaciones
Pares	Simplificado	
P={(S::=aSbb), (S::= aSb|ε) }	S ::= a S bb| aSb | ε	S → aSbb→ aab = $a^1 b^2$ S → aSbb→ aaSbbb→aabbb = $a^2 b^3$ S → aSbb→ aaSbbbb→aaaSbbbbb→aaabbbbb = $a^3 b^5$

Se genera el $L_{IC} = \{ a^n b^m \mid n,m\geq 0 ; m\geq n \}$: $L_{IC} = \{ a^1 b^2, a^2 b^3, a^3 b^5, \ldots \}$

4) OPERACIONES

a) UNION

Ejemplo: Sea $G_{IC1} = (\Sigma_T, \Sigma_N, S, P) = (\{a,b\},\{S_1\}, S, P)$:

Producciones		Derivaciones
Pares	Simplificado	$S_1 \rightarrow a\,S_1 b \rightarrow aabb = a^2 b^2$
$P=\{(S_1::=a\,S_1 b), (S_1::=\varepsilon)\}$	$S_1 ::= a\,S_1\,b \mid \varepsilon$	$S_1 \rightarrow a\,S_1 b \rightarrow aaS_1bb \rightarrow aaabbb = a^3 b^3$ $S_1 \rightarrow aS_1b \rightarrow aaS_1bb \rightarrow aaaS_1bbb \rightarrow aaaabbbb = a^4 b^4$

Lenguaje generado: $L_{IC} = \{ a^n b^n \mid n \geq 0 \} = \{ a^1 b^1,\ a^2 b^2,\ a^3 b^3, a^4 b^4, \ldots \}$

Luego, sea $G_{IC2} = (\Sigma_T, \Sigma_N, S, P) = (\{a,b\},\{S_2\}, S, P)$:

Producciones		Derivaciones
Pares	Simplificado	$S_2 \rightarrow a\,S_2 bb \rightarrow aab = a^1 b^2$
$P=\{(S_2::=aSbb), (S_2::=\varepsilon)\}$	$S_2::= a\,S_2\,bb \mid \varepsilon$	$S_2 \rightarrow a\,S_2 bb \rightarrow aa\,S_2 bbbb \rightarrow aabbb = a^2 b^4$ $S_2 \rightarrow aS_2bb \rightarrow aaS_2bbbb \rightarrow aaaS_2bbbbbb \rightarrow aaabbbbbb = a^3 b^6$

Lenguaje generado: $L_{IC} = \{ a^n b^{2n} \mid n \geq 0 \} = L_{IC} = \{ a^1 b^2,\ a^2 b^4,\ a^3 b^6, a^4 b^8, \ldots \}$

La unión de G_{IC1} con G_{IC2} :

Producciones	Derivaciones
$S_3::= S_1 \mid S_2$	$S_3 \rightarrow S_1 \rightarrow a\,S_1 b \rightarrow ab = a^1 b^1$
$S_1 ::= a\,S_1\,b \mid \varepsilon$	$S_3 \rightarrow S_1 \rightarrow aS_1b \rightarrow a\,aS_1b\,b \rightarrow aabb = a^2 b^2$
$S_2::= a\,S_2\,bb \mid \varepsilon$	$S_3 \rightarrow S_2 \rightarrow aS_2bb \rightarrow abb = a^1 b^2$

Genera: $G_{IC3} = G_{IC1} \cup G_{IC2} \rightarrow L_{IC} = \{ a^n b^m \mid n,m \geq 0 ;\ 2n \geq m \geq n \} = \{ a^1 b^2,\ a^2 b^3,\ a^3 b^5, \ldots \}$

b) CONCATENACION

Ejemplo: Sea $G_{IC1} = (\Sigma_T, \Sigma_N, S, P) = (\{a,b\},\{S_1\}, S, P)$:

Producciones		Derivaciones
Pares	Simplificado	$S_1 \rightarrow a\,S_1 b \rightarrow aabb = a^2 b^2$
$P=\{(S_1::=a\,S_1 b), (S_1::=\varepsilon)\}$	$S_1 ::= a\,S_1\,b \mid \varepsilon$	$S_1 \rightarrow a\,S_1 b \rightarrow aaS_1bb \rightarrow aaabbb = a^3 b^3$ $S_1 \rightarrow aS_1b \rightarrow aaS_1bb \rightarrow aaaS_1bbb \rightarrow aaaabbbb = a^4 b^4$

Lenguaje generado: $L_{IC} = \{ a^n b^n \mid n \geq 0 \} = \{ a^1 b^1,\ a^2 b^2,\ a^3 b^3, a^4 b^4, \ldots \}$

Luego, sea $G_{IC2} = (\Sigma_T, \Sigma_N, S, P) = (\{a,b\},\{S_2\}, S, P)$:

Producciones		Derivaciones
Pares	Simplificado	$S_2 \rightarrow a\,S_2 bb \rightarrow aab = a^1 b^2$
$P=\{(S_2::=aSbb), (S_2::=\varepsilon)\}$	$S_2::= a\,S_2\,bb \mid \varepsilon$	$S_2 \rightarrow a\,S_2 bb \rightarrow aa\,S_2 bbbb \rightarrow aabbb = a^2 b^4$ $S_2 \rightarrow aS_2bb \rightarrow aaS_2bbbb \rightarrow aaaS_2bbbbbb \rightarrow aaabbbbbb = a^3 b^6$

Lenguaje generado: $L_{IC} = \{ a^n b^{2n} \mid n \geq 0 \} = L_{IC} = \{ a^1 b^2,\ a^2 b^4,\ a^3 b^6, a^4 b^8, \ldots \}$

La concatenación de G_{IC1} con G_{IC2} :

Producciones	Derivaciones
$S_3::= S_1\,S_2\,S_1$	$S_3 \rightarrow S_1S_2S_1 \rightarrow S_1S_1 \rightarrow aS_1bS_1 \rightarrow abS_1 \rightarrow ab = a^1 b^2$
$S_1 ::= aS_1b \mid \varepsilon$	$S_3 \rightarrow S_1S_2S_1 \rightarrow S_2S_1 \rightarrow aS_2bbS_1 \rightarrow aaS_2bbbbS_1 \rightarrow aaS_2bbbb \rightarrow aabbbb = a^2 b^4$
$S_2::= aS_2bb \mid \varepsilon$	$S_3 \rightarrow S_1S_2S_1 \rightarrow S_2S_1 \rightarrow aS_2bbS_1 \rightarrow aaS_2bbbbS_1 \rightarrow aaS_2bbbb \rightarrow aaaS_2bbbbbb \rightarrow aaabbbbbb = a^3 b^6$

Genera: $G_{IC4} = G_{IC1}.G_{IC2} \rightarrow L_{IC} = \{ a^n b^m \mid n,m \geq 0 ;\ m = 2n \} = \{ a^1 b^2,\ a^2 b^4,\ a^3 b^6, \ldots \}$

c) ESTRELLA DE KLEENE

Ejemplo: Sea $G_{IC} = (\Sigma_T, \Sigma_N, S, P) = (\{a,b\},\{S_1\}, S, P)$:	
Producciones	Derivaciones
$S_1 ::= a\ S_1 b \mid a\ b \mid \lambda$	$S \rightarrow ab = \mathbf{a^1 b^1}$ $S \rightarrow aSb \rightarrow aabb = \mathbf{a^2 b^2}$ $S \rightarrow aSb \rightarrow a\ aSb\ b \rightarrow a\ aabb\ b = \mathbf{a^3 b^3}$ $S \rightarrow aSb \rightarrow a\ aSb\ b \rightarrow a\ a\ aSb\ b\ b \rightarrow a\ a\ aabb\ b\ b = \mathbf{a^4 b^4}$

Lenguaje generado: $L_{IC} = \{ a^n b^n \mid n \geq 0 \} = \{ a^1 b^1,\ a^2 b^2,\ a^3 b^3, a^4 b^4, \ldots \}$

La ESTRELLA de KLEENE de: $G_{IC} = (\Sigma_T, \Sigma_N, S, P) = (\{a,b\},\{S_3\}, S_3, P)$:	
Producciones	Derivaciones
$S_3 ::= S_1 S_3 \mid \lambda$ $S_1 ::= a\ S_1 b \mid a\ b \mid \lambda$	$S_3 \rightarrow S_1 S_3 \rightarrow a\ S_1 b S_3 \rightarrow ab = \mathbf{a^1 b^1}$ $S_3 \rightarrow S_1 S_3 \rightarrow a\ S_1 b S_3 \rightarrow aa S_1 bb S_3 \rightarrow aabb S_3 \rightarrow aabb = \mathbf{a^2 b^2}$ $S_3 \rightarrow S_1 S_3 \rightarrow S_1 S_1 S_3 \rightarrow a S_1 b S_1 S_3 \rightarrow a S_1 ba S_1 b S_3 \rightarrow abab = \mathbf{abab}$ $S_3 \rightarrow S_1 S_3 \rightarrow S_1 S_1 S_3 \rightarrow S_1 S_1 S_1 S_3 \rightarrow a S_1 ba S_1 ba S_1 b S_3 \rightarrow ababab = \mathbf{ababab}$

Lenguaje generado: L_{IC} = { $\mathbf{a^1 b^1}$, $\mathbf{a^2 b^2}$, **abab**, ababab . . }

EJEMPLOS SUELTOS: La $G_{IC} = (\Sigma_T, \Sigma_N, S, P) = (\{x,y\},\{S,A,B\}, S, P)$:

Producciones	Derivaciones
S::= x A y| x B y y| x y| x y y y| λ A::= x A y | x y B::= x B y y | x y y	$S \rightarrow xy = \mathbf{x^1 y^1}$ $S \rightarrow xyy = \mathbf{x^1 y^2}$ $S \rightarrow xAy \rightarrow xxyy = \mathbf{x^2 y^2}$ $S \rightarrow xAy \rightarrow xxAyy \rightarrow xxxyyy = \mathbf{x^3 y^3}$ $S \rightarrow xByy \rightarrow xxyyyy = \mathbf{x^2 y^4}$ $S \rightarrow xByy \rightarrow x\ xByy\ yy \rightarrow xx\ xyy\ yyyy = \mathbf{x^3 y^6}$

Genera: $L_{IC} = \{ a^n b^m \mid n,m \geq 0 ;\ m=n, \text{ o } m=2n \} = \{ x^1 y^1, x^1 y^2, x^2 y^2, x^3 y^3, x^2 y^4, x^3 y^6 \ldots$

Ejemplo: La G_{DC}: $G_1 = (\Sigma_T, \Sigma_N, S, P) = (\{ a,b,c \}, \{B, C, S\}, S, P)$, con:

Producciones	Derivaciones
S → aSBC|aBC CB→BC aB→ab bB→bb bC→bc cC→cc	• $S \rightarrow aBC \rightarrow \underline{ab}C \rightarrow \underline{abc} = a^1b^1c^1$ • $S \rightarrow aSBC \rightarrow a\underline{aBC}BC \rightarrow a\underline{ab}CBC \rightarrow aab\underline{BC}C \rightarrow aa\underline{bb}CC \rightarrow aab\underline{bc}C \rightarrow aabb\underline{cc} = a^2b^2c^2$ • $S \rightarrow aSBC \rightarrow aaSBCBC \rightarrow aaaBCBCBC \rightarrow aaaBBCCBC \rightarrow aaaBBCBCC \rightarrow aaaBBBCCC \rightarrow$ aaabBBCCC→aaabbBCCC→aaabbbCCC→aaabbbcCC→aaabbbc**cC**→aaabbbc**cc** = $a^3b^3c^3$

Lenguaje generado: $L_{DC} = \{ a^1b^1c^1,\ a^2b^2c^2,\ a^3b^3c^3, \ldots, a^n b^n c^n \}$

Ejemplo: La G_{IC} para expresiones enteras algebraicas sintácticamente correctas sobre las variables **x, y** y **z**: **S → x | y | z | S + S | S - S | S *S | S/S | (S)** Genera la cadena: **(x + y) *x - z *y / (x + x)**

Ejemplo: La G_{IC} para un lenguaje de cadenas formadas un número diferente de las letras **a** y **b**:

S → U | V; U → TaU | TaT — **T** genera todas las cadenas con la misma cantidad de letras **a** que **b**,
V → TbV | TbT — **U** genera todas las cadenas con más letras **a**
T → aTbT | bTaT | ε — **V** todas las cadenas con más letras **b**.

G_{IC}: PROPIEDADES:

- La unión y concatenación de dos L_{IC} es también un L_{IC}, la intersección no siempre.
- El conjunto de los L_{IC} esta cerrado para la unión, la concatenación y el cierre de Kleene
- El conjunto de los L_{IC} no esta cerrado para la intersección ni complementación.

  Asi: $L_1 = \{ a^n b^n c^n \mid n \geq 0\}$, $L_2 = \{ a^n b^n c^m \mid n,m \geq 0\}$, $L_3 = \{ a^n b^m c^m \mid n,m \geq 0\}$ donde L_2 y L_3 son L_{IC}, pero $L_1 = L_2 \cap L_3$, no es L_{IC}.
- Si **L** es un L_{IC} y **R** es un L_{RE}, luego $L \cap R$ es un L_{IC}.
- El conjunto de los L_{IC} esta cerrado para las sustituciones.
- Un lenguaje es L_{IC} si puede ser aceptado por un autómata no determinista.
- Un lenguaje puede ser también modelado como un conjunto de todas las secuencias de terminales aceptadas por la gramática.
- El inverso de un L_{IC} es también L_{IC}, pero el complemento no tiene por que serlo.
- El lenguaje regular es L_{IC} por que pueden ser descritos mediante una gramática regular.
- Existen G_{IC} que no son libres de contexto.
- Para demostrar que un lenguaje dado no es L_{IC}, se puede emplear el **Lema del bombeo** para L_{IC}.
- El problema de determinar si una G_{IC} describe un L_{IC} es indecidible.

G_{IC}: PROBLEMAS INDECIDIBLES:

Las propiedades de las G_{IC} y los L_{IC} son de naturaleza **decidible, porque existen algoritmos de decisión para resolverlos**.

En los L_{RE}, existen problemas de naturaleza **indecidible**, porque carecen de tales algoritmos.
Este lenguaje es una reducción del problema de parada de una máquina de Turing con una entrada particular, o sea es un problema indecidible.

Es indecidible la comparación entre dos G_{IC} para comprobar si el lenguaje generado coincide. Por el contrario, el problema de determinar si dada una cadena es aceptada por una G_{IC}, sí que es decidible, y así podrá escribirse el correspondiente algoritmo para decidirlo.

G_{IC}: RECURSIVIDAD EN GRAMATICAS:

Las producciones de la **gramatica recursiva** tiene al menos una producción recursiva, donde la recursividad implica la **reutilización de un mismo concepto en su propia definición** y abarca la:

PRODUCCION RECURSIVA:
Ocurre cuando existe un (**A** ::= x**A**y) ∈ P, (x y ∈ Σ*), aparece en ambos lados de la producción el mismo símbolo no terminal.

Ejemplo: **A**::= 0**A**0, **B**::= **B**10, **C**::= 111**C**

La recursividad puede ser por:

- **DERECHA**:
  Si existe A::= x**A**, (A ∈ Σ_N, x ∈ Σ*), aparece último por la parte derecha de la producción el símbolo no terminal. **Ejemplo**: **B**::= 111**B**

- **IZQUIERDA**:
  Si existe A::=**A**y, (A ∈ Σ_N, y ∈ Σ^*), aparece primero por la parte izquierda de la producción el símbolo no terminal. **Ejemplo**: **B**::= **B**10

- **IZQUIERDA EN MAS DE UN PASO**:
  Cuando **A::=Bx, A ≠ B, (A, B ∈ Σ_N, x ∈ Σ*)** , existe más de una regla no recursiva por la izquierda pero desde **B** hay una derivación del tipo **B →*Ay, (y∈Σ*)** En este caso la recursividad está respecto del símbolo no terminal **A**.

Ejemplo: E::= T + E | T * E | variable | número T::= E | (E) pues E→T→E→E→E

En el lado izquierdo de una producción aparece una de las cadenas del lado derecho:

Ejemplo: En la notación BNF:
- <v_o> ::= a<w>
- <***w***> ::= bb<***w***> | c

Como en el lado izquierdo de una producción aparecen también en una de las cadenas del lado derecho, pues, <***w***> aparece a la izquierda, y aparece en la cadena bb<***w***>, la producción ***w*** →bb ***w*** es **recursiva**.

Si la producción recursiva tiene a ***w*** como lado izquierdo, la **producción es normal** si ***w*** **aparece sólo una vez en el lado derecho y es el símbolo del extremo derecho**.

Al lado derecho pueden tener otros símbolos no terminales. La producción recursiva ***w*** →bb ***w*** es normal.

Ejemplo.
La sintaxis de números decimales es una minigramática, cuyo lenguaje consta de:

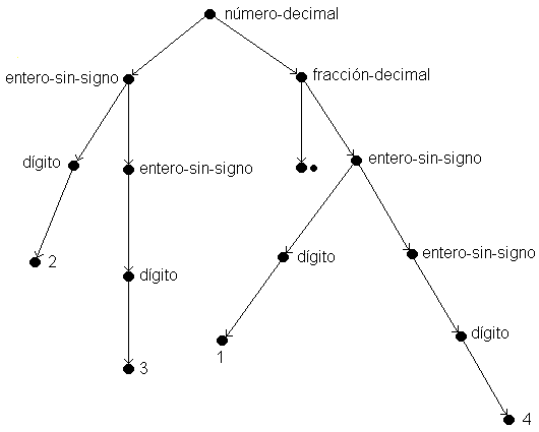
- **S = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, .}. ∈ (Σ_T)**
- **N ={nro-decimal, fracción-decimal, entero-sin-signo, dígito}∈(Σ_N)**
- **V= S ∪ N**
- **G = (Σ_T, Σ_N, A, P) con producciones P en forma BNF:**

<número-decimal> ::= <entero-sin-signo> | <fracción-decimal> |
<fracción-decimal> ::= <entero-sin-signo> <fracción-decimal>
<entero-sin-signo> ::= <dígito> |<dígito><entero-sin-signo>
<dígito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Arbol de deducción: muestra la gramática, del número decimal **23.14**. Ver que el enunciado BNF número 3 es recursivo en la segunda parte de su lado derecho.
Así, la producción "**entero-sin-signo →dígito entero-sin-signo**" es recursiva y también es normal.
Reemplazando la línea anterior número 3 con la línea: 3'.

<entero-sin-signo> ::= <dígito> | <entero-sin-signo> <dígito>

Se tendrá otra gramática que produce el mismo lenguaje, con los números decimales formados de manera correcta.
Esta gramática tiene una producción recursiva que no es normal.



G_{IC}: GRAMATICA DE ESTRUCTURA DE FRASE:

Se logra eliminando en la G_{IC} la restricción respecto del lado derecho de las producciones, lograremos que la misma pueda ubicarse en cualquier cadena no vacía sobre $\Sigma_N \cup \Sigma_T$.
Así los lados izquierdos de las reglas de producción P estarán formados por cualquier cadena no vacía de $\Sigma_N \cup \Sigma_T$, que contienen algún no terminal y satisface:

$$P \subseteq (\Sigma_N \cup \Sigma_T)^* \ \Sigma_N \ (\Sigma_N \cup \Sigma)^* \times (\Sigma_N \cup \Sigma_T)^*$$

La G_{IC}, posee estructura de frase, cuyas producciones se limitan a $\alpha \rightarrow \beta$, tal que $|\alpha| \geq |\beta|$, cuya forma normal, especificará que sus producciones permiten que el no terminal **A** será reemplazado por la cadena $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$, para $\beta \neq \lambda$ pero solo si **A** aparece en el contexto α_1 y α_2.

G_{IC}: LEMA DEL BOMBEO: (Enunciado por **Y. Bar-Hillel, M. Perles, E. Shamir**)

Propone que en un lenguaje, toda cadena de caracteres de al menos una cierta **longitud de bombeo**, contiene una sección que puede ser eliminada o repetida cualquier número de veces, generando otra cadena resultante perteneciente a ese lenguaje. La prueba de este lema requiere argumentos de conteo como los del **principio del palomar**.

Ejemplos son el lema de bombeo para L_{RE} y el para la G_{IC}, que permiten ver si un lenguaje **no está** en una clase de lenguajes, pero no pueden ser usados para determinar si un lenguaje está en una clase, puesto que satisfacer el lema del bombeo es una condición necesaria, pero no una suficiente, para ser miembro de una clase. El lema de Ogden es un segundo lema de bombeo, para L_{IC}.

Sea L un L_{IC}. Entonces existe una constante $n \in N$ tal que para toda palabra $x \in L$, con $|x| \geq n$,

Existe una descomposición **x= y z u v w** tal que:
|zuv| ≤ n
|zv|>0
∀i ≥ 0, x= y z^i u u^i w∈ L

Para **|x|** es la longitud de la palabra y x^i la repetición de la palabra **x**, **i** veces, el teorema implica que en todo L_{IC}, para toda palabra suficientemente larga **|x|≥ n**, se pueden encontrar 1 o 2 subcadenas izquierda **z** y derecha **v**, cuya longitud conjunta es a lo sumo **n** **|zuv| ≤ n**, que pueden eliminarse, o repetirse simultáneamente las veces que se desee y $z^i u u^i w$, obteniendo de dicha forma palabras que también pertenecen al lenguaje.

El contrarrecíproco de este teorema se puede aplicar para demostrar que un lenguaje no es L_{IC}:

1. Suponemos **n** la constante de bombeo.
2. Escogemos cuidadosamente una determinada palabra del lenguaje tal que **|x| ≥ n**
3. Si hallamos una descomposición: **|zuv| ≤n**, **|zv|>0** tal que alguna de las palabras: $yz^iuu^iw \geq 0$ no pertenece al lenguaje, entonces dicho lenguaje no L_{IC}.

G_{IC}: ARBOLES DE DERIVACION:

Representan las producciones utilizadas para construir una palabra de la $G_{IC} = (\Sigma_T, \Sigma_N, S , P)$, para ello:

- Los nodos internos representan los símbolos no terminales.
- Las hojas representan los símbolos terminales.
- Una producción **A**::= $X_1 \ldots X_n$, representa un subárbol donde **A** es el nodo padre y $X_1 \ldots X_n$ son los nodos hijos.
- La **derivación** es por:
  - **Izquierda** cuando la producción aplicada esta a la izquierda del símbolo no terminal.
  - **Derecha** cuando la producción aplicada esta a la derecha del símbolo no terminal.

Sea la $G_{IC} = (\Sigma_T, \Sigma_N, S , P)$ = (**{iden, cte, (,),+,-,*,/}, {<expre>, <opera>, <expre>,S, P}**)

Producciones	Derivaciones	

P= {
<ex>::= <ex> <op> <ex>|(<ex>)|id|ct
<op>::= +|-|*|/
}
	<ex>
	/ | \\
	<ex> <op> <ex>
	/ | \\ | |
	<ex><op><ex> | |
	| | | | |
	id + ct * id
		<ex>
		/ | \\
		<ex> <op> <ex>
		| | / | \\
		| | <ex><op><ex>
		| | | | |
		id + ct * id

Estos árboles de derivación son diferentes pero generan la misma palabra. La gramatica es ambigua.

G_{IC}: AMBIGUEDAD:

Sus postulados son:

- Si para al menos una cadena perteneciente a L(G) existe más de un árbol de derivación, G es una gramática ambigua.
- Si para todas las cadenas pertenecientes a L(G) existe solo un árbol de derivación, G es una gramática no ambigua.
- El lenguaje L(G) será un lenguaje ambiguo o un lenguaje no ambiguo según el análisis de ambigüedad para G.
- Si todas las gramáticas generadoras del lenguaje L son ambiguas, L es un lenguaje inherentemente ambiguo.

La ambigüedad en la G_{IC} puede ser de:

Nivel	SENTENCIA	GRAMATICA	LENGUAJE
	Posee más de una derivación o árbol de derivación.	Posee al menos una sentencia ambigua	Generada por una gramática ambigua
	Ejemplo: En la gramática **G=({1},{A,B},A,{(A::=1B|11),(B::=1)})** La sentencia **11** se genera por las derivaciones **A→1B→11** y **A→11**.	Ejemplo: La gramática anterior es ambigua, por que **11** es generada por sentencia ambigua.	Ejemplo: Como la gramática anterior es ambigua, el leguaje **L = { 11 }** es ambiguo.

Ejemplo:

$G_{IC} = (\Sigma_T,\Sigma_N,S,P) = (\{a,b,p,q,if,then,else\}, \{A,B\}, A, P)$

Producciones	Derivaciones	
	if **a** then if **b** then **p** else **q**	if **a** then if **b** then **p** else **q**
P={(**A**::= if **B** then **A** else **A**), (**A**::= if **B** then **A**),(**A**::= **p**),(**A**::= **q**) (**B** ::= **a**), (**B** ::= **b**) }	**A** / / / \ \ \ if **B** then **A** else **A** | //\\\ | | if **B** then **A** | | | | | **a b p q**	**A** / / \ \ if **B** then **A** | ///\\\ | if **B** then **A** else **A** | | | | **a b p q**

Esta gramatica es ambigua porque una misma derivación: **a b p q**, tiene los dos siguientes arboles de derivación

Para eliminar la ambigüedad de una G_{IC} no existe un algoritmo único, cada caso es particular, en el ejemplo de la $G_{IC}=(\Sigma_T,\Sigma_N,S,P)=$({a,b,p,q,if,then,else},{A,B},A,P), insertamos nuevos no terminales A_1 y A_2 así, queda la: $G_{IC}=(\Sigma_T,\Sigma_N,S,P)=$({a,b,p,q,if,then,else},{A, A_1, A_2, B},A,P)

Producciones	Derivaciones
Pares	if **a** then if **b** then **p** else **q**
P={ (**A**::= A_1), (**A**::= A_2) (A_1::= if **B** then **A**) (A_1::= if **B** then A_2 else A_1), (A_2::= if **B** then A_2 else A_1), (A_2::= p),(A_2::= q) (**B**::=a), (**B**::=b), }	**A** | A_1 / / \ \ if **B** then A_2 | ///\\\ | if **B** then A_2 else A_2 | | | | **a** **b** **p** **q**

Lenguaje generado: L_{IC} = {a b p q, }

G_{IC}: SIMPLIFICACION:

Para transformar una G_{IC} en otra equivalente, cuyas producciones permitan construir reconocedores, conviene **corregir 3 tipos de defectos**:

a) **Prefijos comunes**: Tienen dos o mas producciones que teniendo la misma parte izquierda, tienen símbolos coincidentes al principio de la derecha.
b) **Recursividad por izquierda**: Tienen la forma **X**::= **aXb** y cuando **X**::= **Xb** son recursivas por izquierda (Complican construir reconocedores) y si su forma es **X**::= **aX** son recursivas por derecha
c) **Ambigüedad**: Aunque no existe un algoritmo genérico para eliminar la ambigüedad, puede resolverse el problema analizando sus causas.

Ejemplo: Simplificar: $G_{IC} = (\Sigma_T, \Sigma_N, S, P) = (\{a,b,c\}, \{A,B,C,D,E,F,G,H,S\}, S, P)$

Con **P**: **S→D|abGc|C; A→bAc|aHbcA|B; B→A| λ |abHba|aGEa;**
C→aCb|ab|cZc; D→aDc|B; E→cEb| λ;
F→E|aFa; G→aGb|bGa; H→cHa|aHc

Para simplificar la G_{IC} se debe:

1- Eliminar Reglas Innecesarias:
O producciones inocuas que tienen no terminales inútiles, o **N** que no cumplen con:
- $S \rightarrow^* \alpha N \beta$ y $N \rightarrow^+ w$, donde: $w \in \Sigma^*_T$ y $\alpha, \beta \in \Sigma^*$
- La forma: $N \rightarrow N$

S→D | C
A→bAc | B
B→A | λ
C→aCb | ab
D→aDc | B

2- Generación de λ:
Si $S \rightarrow^*$, entonces:
Si **S** aparece a la derecha de alguna producción:
- Introducir un nuevo síbolo inicial: S_i
- Agregar la regla: $S_i \rightarrow S \mid \lambda$

S→D | C | λ
A→bAc | B
B→A| λ
C→aCb | ab
D→aDc | B

3- Eliminar reglas de borrado: $N \rightarrow \lambda$:
Identificar los no terminales anulables: $N \rightarrow \lambda$
Reemplazar: $N \rightarrow \lambda$, por reglas que surgen de reemplazar **N** por λ en todas las reglas donde figure **N**, excepto $S \rightarrow \lambda$.

S→D | C | λ
A→bAc | B | bc
B→A
C→aCb | ab
D→aDc | B | ac

4- Eliminar reglas de renombrado: $N_1 \rightarrow N_2$:
Se reemplaza: $N_1 \rightarrow N_2$, por reglas que surgen de reemplazar N_2 por las partes derechas de sus producciones

La $G_{IC} = (\{a,b,c\},\{A,B,C,D,E,F,G,H,S\},S,P)$ simplificada es: $G_{IC} = (\{a,b,c\},\{A,B,C,D,S\},S,P)$

FORMAS NORMALES

Una **forma normal es un intento de estandarizar las producciones y conseguir que todas tengan una apariencia similar**. Con este fin, partiendo de una G_{IC} simplificada, se desarrollan la:

A. Forma Normal de Chomsky: FNC

Una gramática formal está en FNC si todas sus reglas de producción tienen el formato:

$A \rightarrow B\ C$ o
$A \rightarrow \alpha$

Donde:

- **A, B** y **C**: Símbolos no terminales
- **α** : Símbolo terminal.

Todo L_{IC} que no posee a la cadena vacía:

- Es expresable mediante una gramática en **FNC** y recíprocamente.
- Dada una G_{IC}, es posible algorítmicamente producir una G_{FNC} equivalente, es decir, que genera el mismo lenguaje.

Asi como, cualquier G_{FNC} produce un L_{IC}; para cualquier L_{IC} existe una G_{FNC} que lo define.
Esta definición se diferencia en permitir una regla **ε** de la forma:

$A \rightarrow BC$ o
$A \rightarrow \alpha$ o
$S \rightarrow \varepsilon$

- **S** símbolo inicial de la gramática,
- **A** símbolo no terminal,
- **B** y **C** símbolos no terminales pero distintos de **S**.
- **α** símbolo terminal, y
- **ε** cadena vacía.

Ejemplo: Gramática en FNC: $A \rightarrow AZ$ $A \rightarrow w$ $Z \rightarrow c$

Transformación de la G_{IC} a la FNC

Descripción

1. Eliminar reglas unitarias.

Verificar si hay reglas unitarias que obstruyan el desarrollo de FNC.

Ejemplo Regla unitaria: $A \rightarrow X \rightarrow z$
El No terminal **A** deriva en otro No terminal **X**, que a su vez deriva en un Terminal **z**, esto es redundante y por tanto se procede a eliminar el No terminal **X** y pasando el Terminal **z** al No terminal **A**

2. Eliminar reglas no productivas

La regla no productiva es un No Terminal no accesible desde el No Terminal principal y sus respectivas derivaciones, del mismo o de las que provoquen sus No Terminal que se encuentren en su propia derivación.

Ejemplo Regla no productiva: $A \rightarrow AZ$ $W \rightarrow X$ $Z \rightarrow c$
El No Terminal **W** es una regla no productiva porque no es accesible desde el No Terminal **A**, ni de su derivación **AZ**.

3. Dar formato de FNC

REGLAS BÁSICAS Y ÚNICAS PARA TENER EL FORMATO FNC:

a) Un No Terminal solo puede derivarse en otros dos No Terminales.

b) Un No Terminal solo puede derivar en un Terminal.

Ejemplo:

Gramatica	Proceso
A → c B + **B → q**	No tiene ni reglas unitarias ni reglas no productivas, luego procedemos con el paso 3º
A → c B +	El No Terminal **A** deriva en: **c B +**, cuyo 1er elemento es un Terminal, procedemos a crear un nuevo No Terminal con este elemento y agregamos al No Terminal **A**, respetando el orden de la gramática.
A → Z B + **Z → c**	En FNC, un NoTerminal solo puede derivar en 2 NoTerminales, el NoTerminal **A** que deriva en **ZB+**, que contiene un elemento terminal más, creamos un nuevo NoTerminal para respetar tal propiedad.
A→ Z Y **Z → c** **Y→B+**	Los No Terminal **A** y **Z** cumplen con las propiedades de la FNC, excepto el No Terminal **Y** que deriva en un No Terminal y un Terminal, por lo que procedemos a crear el último No Terminal que cumpla la FNC. **A→ Z Y Z → c Y→B X X → +** Se obtiene el resultado: **A→ Z Y Z → c Y→B X X → + B → q**

B. Forma Normal de Greibach: FNG

Una G_{IC} está en FNG cuando sus reglas de producción tienen un consecuente que empieza por un símbolo terminal, tienen la estructura:

$$A \rightarrow aw$$

Donde:

- **A**: Antecedente de la regla, que en el caso de las G_{IC} debe ser un solo símbolo auxiliar.
- **a**: Comienzo del consecuente, por tanto, un símbolo terminal.
- **w**: Concatenación genérica de elementos gramaticales, ó una sucesión exclusiva de auxiliares, inclusive, puede ser la palabra vacía; en tal caso, se tendría una regla llamada "terminal":

$$A \rightarrow a$$

Un teorema prueba que cualquier G_{IC}, cuyo lenguaje no contiene a la palabra vacía, si no lo está ya, se puede transformar en otra equivalente que sí esté en **FNG**.

G_{IC}: FORMA NORMAL DE GREINBACH: FNG

Una G_{IC} está en FNG si sus:

- Reglas de producción tienen un símbolo terminal.
- Reglas tienen estructura: **A→aw**
- Producciones son: $P=\{(A::= aX) \text{ó } (S::=\lambda) | A\in\Sigma_N, X\in\Sigma_N^*, a\in\Sigma_T\}$ donde:
  - **A**: Antecedente de la regla, debe ser un solo símbolo auxiliar o no terminal de la G_{IC}.
  - **a**: Comienzo del consecuente, símbolo terminal.
  - **w**: Concatenación de elementos gramaticales, o sucesisión exclusiva de auxiliares, incluso la palabra vacía; así se tendría una regla "terminal": **A→a**

Proceso para convertir G_{IC} a su equivalente	Ejemplo
ELIMINAR RECURSIVIDAD POR IZQUIERDA	La gramática **G = ({ 0,1, 2 }, { A, B, C }, A, P)** es limpia La regla **B::=BC** es **recursiva por izquierda**: Borrar las reglas de **B** se añaden: **B::=1B', B'::=CB', B'::=λ** Para evitar reglas no generativas: **B::=1B', B::=1, B'::= CB', B'::=C** Eliminar regla de redenominación, **B' ::= C**. Añadir la regla **B'::= 2**.
SUSTITUIR REGLAS: Cuyas partes derechas comiencen con un no terminal, fijando un orden en los símbolos no terminales A_1, A_2, A_3, …, A_n. ▪ Se dividen las reglas en tres grupos y se trata de conseguir el **objetivo que todas las reglas sean del Grupo 1**: ❑ **Grupo 1**: $A_i ::= a\,x$, $(a\in\Sigma_T, x\in\Sigma^*)$ ❑ **Grupo 2**: $A_i ::= A_j\,x$ $(A_i, A_j\in\Sigma_N, i\ j, x\in\Sigma^*)$ ❑ **Grupo 3**: $A_i ::= A_j\,x$ $(A_i, A_j\in\Sigma_N, i\ j, x\in\Sigma^*)$ ▪ Seleccionar 1ro reglas del grupo 3 de mínimo orden **i**. ▪ Sustituir estas por las obtenidas al sustituir **Aj** por las partes derechas de sus producciones. Iterar proceso en orden creciente de **i** hasta lograr el objetivo. ▪ Luego se aplica este proceso al Grupo 2.	Luego de eliminar recursividad por izquierda y limpiar **G=({0,1,2}, {A,B,C}, A, P)**, fijar orden de no terminales **A, B, B', C**; así no habrá regla del grupo 3, quedando solo reglas del grupo 2. Cambiar las apariciones como 1er símbolo de la derecha de **C** en las producciones de **B, B'** y **A**, las apariciones de **B'** en **B** y **A** y las de **B** en **A**. Así la regla **A::= CB2** genera **A::=2B2** y borra **A::=CB2**. de modo que, en esta fase las producciones serán: **P' = { (A::= 2B2), (A::= 1B), (A::= λ), (B::= 1B'), (B::= 1) , (B'::= 2B') , (B'::= 2) , (C::= 2) }**
❑ Excepto **A::=λ**, todas las partes derechas de las reglas empiezan por un símbolo terminal. ❑ Si todas las reglas pertenecen alguno de los tres tipos de FNG, no se hace nada. ❑ Por cada regla tipo $A::=xA, A\in\Sigma_N, X\in\Sigma^*, a\in\Sigma_T$ con símbolos terminales **b** en **X**, y por cada símbolo **b**, **se sustituye su aparición** en esa producción por un no terminal **N**, que solo tiene una producción: **N::= b**.	❑ En la gramática **G=({0,1,2},{A,B,C},A,P)** donde se obtuvo las reglas del ej anterior, solo A::=2B**2** es regla que no esta en FNC porque tiene un terminal derecho **2**, que puede sustituirse con el terminal **C** que tiene una producción de terminal **2**. ❑ Si **C** tuviese otros símbolos, se debería crear un nuevo no terminal **N,** crear una producción **N::=2** y cambiar en la regla **N** por **2**, quedando **A::=2BN**, La gramática expresada como FNG, será **G' = ({ 0,1, 2 }, { A, B, B', C }, A, P)** Y sus producciones serán **P' = { (A::= 2B2),(A::=1B), (A::=λ),(B::=1B'),(B::=1) , (B'::= 2B'),(B'::= 2),(C::=2)}**

Fuente: 2006, Dr. Arno Formella, Universidad de Vigo, Departamento de Informática

Fuente: 2006, Dr. Arno Formella, Universidad de Vigo, Departamento de Informática

G_{IC}: FORMA NORMAL DE CHOMSKY FNC

Si $G = (\Sigma_N, \Sigma_T, P, \$)$ es gramática con $P \subset \Sigma_N \times (\Sigma_T \cup \Sigma_N)^*$; donde $\$$ es el símbolo inicial y $X \in \Sigma_N$ un símbolo no terminal, que puede ser:

- **Acesible**: Si existe derivación desde $\$$ que contiene X: $\$ \rightarrow^* \alpha X \beta$ donde $\alpha, \beta \in \Sigma^*$
- **Generativa**: Si existe derivación desde la variable w: $X \rightarrow^* w$, donde $w \in \Sigma_N^*$.
- **Util**: Si existe derivación desde $\$$ usando X que produce una sentencia w: $\$ \rightarrow^* \alpha X \beta \rightarrow^* w$ donde $\alpha, \beta \in \Sigma^*$ y $w \in w \in \Sigma_T^*$.

Una gramática está en FNC si contiene:

1- Σ_N solo variables útiles y si las producciones de G son $X,Y,Z \in \Sigma_N$, asi todas las variables son necesarias para derivar al menos una sentencia

2- La forma $X \rightarrow Y,Z$ con $X \in \Sigma_N$ y $\sigma \in \Sigma_T$, asi garantiza que un árbol de derivación es un árbol binario

3- La forma $X \rightarrow \sigma$ con $X \in \Sigma_N$ y $\sigma \in \Sigma_T$ y si $\$$ no está a la derecha de ninguna producción. Además:
Se permite que $\$ \rightarrow \lambda \in P$, para poder derivar λ.

Si $\$$ aparece a la derecha, primero habrá que sustituir las producciones implicadas

Una gramática en **FNC** es una G_{IC} que se verifica analizando la forma de producciones permitidas. También para cualquier L_{IC} existe una gramática en **FNC**, que genera el mismo lenguaje.

Ejemplo: Sea L_G un L_{IC} generado por la gramática $G = (\Sigma_N, \Sigma_T, T, \$)$, asumir que $\lambda \notin L$, generar una nueva gramática en FNC mediante los siguientes pasos:

- **Eliminar las variables inútiles**
- **Modificar reglas sin mezcla de variables y constantes a la derechas de las producciones y que todas las reglas con constantes tengan la forma $X \rightarrow \sigma$**
- **Sustituir las reglas cuya longitud de su parte derecha es > 2**
- **Sustituir las reglas de tipo: $X \rightarrow \varepsilon$ o $X \rightarrow \lambda$**
- **Sustituir las reglas de tipo $X \rightarrow Y$, las reglas unitarias.**

Las gramáticas después de cada paso respectivamente son:
$G = G_0, G_1, G_2, \ldots G_5 = G_{FNC}$
La gramática inicial:
$G_0 = (\{ \$, A, B, C, D, E, F \}, \{a, b, c\}, P_0, \$)$, donde P_0 contiene→

$$
\begin{aligned}
\$ &\longrightarrow bDD \mid Ca \mid bc \\
A &\longrightarrow B \mid aCC \mid baD \\
B &\longrightarrow cBD \mid \epsilon \mid AC \\
C &\longrightarrow bD \mid aBA \\
D &\longrightarrow CD \mid a \mid EF \\
E &\longrightarrow Eb \\
F &\longrightarrow a
\end{aligned}
$$

Cuando el no terminal es no-generativa o inaccesible:

Eliminar:

- Variables no-generativas **N** (y todas las reglas con ellas) llamando a la gramática resultante G'_1
- Variables inaccesibles **I** (y todas las reglas con ellas). Recorrer en forma estructurada las variables y reglas:

Para calcular **N** empezar con las variables que producen directamente sentencias (incluyir la palabra vacía $\varepsilon \in L$) y seguir el uso de reglas con dichas variables para producir así sucesivamente sentencias.
Empezar con $N = \Sigma_N$ y borrar del conjunto todas aquellas variables que o bien directamente deriven una sentencia o bien lo hacen indirectamente.
Solo **E** es un símbolo no-generativo, es decir, $N = \{E\}$, P'_1:

$$
\begin{aligned}
\$ &\longrightarrow bDD \mid Ca \mid bc \\
A &\longrightarrow B \mid aCC \mid baD \\
B &\longrightarrow cBD \mid \epsilon \mid AC \\
C &\longrightarrow bD \mid aBA \\
D &\longrightarrow CD \mid a \\
F &\longrightarrow a
\end{aligned}
$$

Para calcular **I** empezar con el símbolo inicial, ver a que variables se puede llegar directamente y seguir el uso de reglas con dichas variables para llegar así sucesivamente a nuevas variables.

$$
\begin{aligned}
\$ &\longrightarrow bDD \mid Ca \mid bc \\
A &\longrightarrow B \mid aCC \mid baD \\
B &\longrightarrow cBD \mid \epsilon \mid AC \\
C &\longrightarrow bD \mid aBA \\
D &\longrightarrow CD \mid a
\end{aligned}
$$

Observar que solo **F** es un símbolo inaccesible: **I = {E}**, **P_1** entonces:
G_1 no contiene símbolos inútiles.

Añadir a cada símbolo terminal **σ** una regla **Wσ** y sustituir **σ** en todas las reglas de **P_1**, **P_2** es:

$$
\begin{aligned}
\$ &\longrightarrow W_bDD \mid CW_a \mid W_bW_c \\
A &\longrightarrow B \mid W_aCC \mid W_bW_aD \\
B &\longrightarrow W_cBD \mid \epsilon \mid AC \\
C &\longrightarrow W_bD \mid W_aBA \\
D &\longrightarrow CD \mid W_a \\
W_a &\longrightarrow a \\
W_b &\longrightarrow b \\
W_c &\longrightarrow c
\end{aligned}
$$

P_2 solo tiene reglas con partes derechas siendo **ε**, un símbolo terminal, o una palabra de variables.

Sustituir cada regla del tipo X → **Y_1 Y_1**… **Y_k** con k > 2 por las reglas:

$$
\begin{aligned}
X &\longrightarrow Y_1X_1 \\
X_1 &\longrightarrow Y_2X_2 \\
&\vdots \longrightarrow \vdots \\
X_{k-3} &\longrightarrow Y_{k-2}X_{k-2} \\
X_{k-2} &\longrightarrow Y_{k-1}Y_k
\end{aligned}
$$

donde las **X_i** son nuevas variables, **P_3** entonces es:

$$
\begin{aligned}
\$ &\longrightarrow W_b\$_1 \mid CW_a \mid W_bW_c \\
\$_1 &\longrightarrow DD \\
A &\longrightarrow B \mid W_aA_1 \mid W_bA_2 \\
A_1 &\longrightarrow CC \\
A_2 &\longrightarrow W_aD \\
B &\longrightarrow W_cB_1 \mid \epsilon \mid AC \\
B_1 &\longrightarrow BD \\
C &\longrightarrow W_bD \mid W_aC_1 \\
C_1 &\longrightarrow BA \\
D &\longrightarrow CD \mid W_a \\
W_a &\longrightarrow a \\
W_b &\longrightarrow b \\
W_c &\longrightarrow c
\end{aligned}
$$

P_3 solo tiene reglas con partes derechas siendo **ε**, símbolo terminal, o una palabra de una o dos variables.

Eliminar las reglas que producen **ε** , distinguiendo entre variables que solo producen **ε** y aquellas que también producen **ε**. Realizar:
1-Calcular los conjuntos de variables
E = {**V|V→*ε** } (las variables que posiblemente producen **ε**) y
E_C ={**V |V→*ε** y no existe **V→*w** con **w≠ ε**} **E_C** (variables que solo producen **ε**).

2-Calcular conjuntos con el mismo algoritmo usado en el 1er paso para detectar variables no-generativas.
Añadir para cada regla del tipo **X →YZ**
X →Y si **Y∉ E_C** y **Z∈ E**
X →Z si **Y∈ E** y **Z∉ E_C** .
3- Eliminar todas reglas de tipo:
X →ε ,y de tipo **X →Y** con **Y ∈ E_C** y
todas las reglas de tipo **X →YZ** con **Y, Z ∈ E_C** .

En el ejemplo tenemos:
E = {A, B, C_1 }, E_C = ϕ , y por eso **P_4** es:

$$
\begin{aligned}
S &\longrightarrow W_bS \mid CW_a \mid W_bW_c \\
S_1 &\longrightarrow DD \\
A &\longrightarrow B \mid W_aA_1 \mid W_bA_2 \\
A_1 &\longrightarrow CC \\
A_2 &\longrightarrow W_aD \\
B &\longrightarrow W_cB_1 \mid \epsilon \mid AC \\
B_1 &\longrightarrow BD \\
C &\longrightarrow W_bD \mid W_aC_1 \\
C_1 &\longrightarrow BA \\
D &\longrightarrow CD \mid W_a \\
W_a &\longrightarrow a \\
W_b &\longrightarrow b \\
W_c &\longrightarrow c
\end{aligned}
$$

Para eliminar las reglas unitarias de tipo **X →Y**:
Calcular el conjunto de las reglas unitarias **U** = { **(X, Y)** | **X →*Y**} (no basta con **(X,Y) ∈U** si **X →Y**, hay que calcular la clausura transitiva).

- Partir del conjunto de reglas unitarias del sistema de producciones: **U_1** = { **(X, Y)** | **X →Y ∈ P**}
- Construir **U_2** insertando para cada par de parejas **(X, Y), (Y, Z) ∈ U_1** la pareja **(X, Y)** , o en general, Construir **U_i** insertando para cada par de parejas **(X, Y), (Y, Z) ∈ $U_{j<i}$ U_j** la pareja **(Y, Z)** .

Seguir con el procedimiento hasta que encontramos U_i vacía para cierto **i**, es decir, no se ha añadido nada más. (El índice **i** de U_i indica con cuantos 'saltos' son necesarios.)

Como el número de producciones es finito, el algoritmo termina y logrando: $U = \bigcup_{i=1}^{\infty} U_i$

Para cada **(X,Y)∈U** y para cada regla **Y→α** que no es regla unitaria, añadir una regla **X→α**

Eliminar todas las reglas unitarias.

$$
\begin{aligned}
\$ &\longrightarrow W_b\$_1 \mid CW_a \mid W_bW_c\\
\$_1 &\longrightarrow DD\\
A &\longrightarrow B \mid W_aA_1 \mid W_bA_2\\
A_1 &\longrightarrow CC\\
A_2 &\longrightarrow W_aD\\
B &\longrightarrow W_cB_1 \mid \epsilon \mid AC\\
B_1 &\longrightarrow BD\\
C &\longrightarrow W_bD \mid W_aC_1\\
C_1 &\longrightarrow BA\\
D &\longrightarrow CD \mid W_a\\
W_a &\longrightarrow a\\
W_b &\longrightarrow b\\
W_c &\longrightarrow c
\end{aligned}
$$

En el ejemplo tenemos:

$$
\begin{aligned}
U_1 &= \{(A,B),(B,C),(B_1,D),(C,W_a),(C_1,A),(C_1,B),(D,W_a)\}\\
U_2 &= \{(A,C),(B,W_a),(B_1,W_a),(C_1,C)\}\\
U_3 &= \{(A,W_a),(C_1,W_a)\}\\
U_4 &= \emptyset
\end{aligned}
$$

P_5, el sistema de producciones final:

$$
\begin{aligned}
\$ &\longrightarrow W_b\$_1 \mid CW_a \mid W_bW_c\\
\$_1 &\longrightarrow DD\\
A &\longrightarrow B \mid W_aA_1 \mid W_bA_2\\
A_1 &\longrightarrow CC\\
A_2 &\longrightarrow W_aD\\
B &\longrightarrow W_cB_1 \mid \epsilon \mid AC\\
B_1 &\longrightarrow BD\\
C &\longrightarrow W_bD \mid W_aC_1\\
C_1 &\longrightarrow BA\\
D &\longrightarrow CD \mid W_a\\
W_a &\longrightarrow a\\
W_b &\longrightarrow b\\
W_c &\longrightarrow c
\end{aligned}
$$

No se añadieron variables inútiles, se borraron reglas, así, todas las variables siguen siendo útiles y después de cada paso la gramática resultante genera el mismo lenguaje: **$L(G_0)$ = $L(G_1)$ = . . = $L(G_5)$**, donde la gramática **G_5** es en forma normal de Chomsky.
Si el lenguaje de partida **L** contiene la palabra vacía (**ε∈L**), se detecta en el pasa 4 que el símbolo inicial pertenece a **E** (o incluso a **E_C**), en este caso eliminar con un nuevo símbolo, por ejemplo **$'**, la aparencia de **$** en los lados derechos y añadir la regla **$→λ**. Tal gramática sigue estando en FNC y genera **L**.
El cálculo de los conjuntos: **N, I, E, E_C**, y **U** que se necesitan para sucesivamente modificar los sistemas de producciones se realiza con un recorrido estructurado sobre las variables y producciones.

Como se eliminó producciones, puede ser que también en el alfabeto de los símbolos terminales **$Σ_T$** hay símbolos superfluos, es decir, que no se pueden producir con las producciones restantes. Dichos símbolos se pueden borrar de **$Σ_T$** sin que se cambie el lenguaje generado.
Al eliminar reglas unitarias se eliminan las reglas innecesarias tipo X→X que se podría borrar de antemano.

Obtener una FNC de una: G_{IC}= ($Σ_T$,$Σ_N$,S,P) = ({a,b},{S},S,P) con S::=aSb |a b | λ genera el **L_{IC}={a^n b^n|n≥0}**:

Pasos	Cambios
Simplificar la G_{IC}: Eliminar las reglas innecesarias	S::=aSb | a b | λ ya está simplificada
Llevar G_{IC} a FNC tipo 0: Cambiar terminales de la derecha por nuevos no terminales y agregar las reglas asociadas	S::=aSb | AB | λ; A::=a; B::=b
Reducir la longitud: De reglas que no cumplen con FNC tipo 2, agregando los no terminales necesarios.	S::=AC | AB | λ; A::=a; B::=b; C::=SB

OBTENER una FNC de una G_{IC} Usando un **algoritmo de conversion**:

Producciones	Reglas	Donde
P={(A::=BC);(S::= λ);(A::=α) | A,B,C∈Σ_N,α∈Σ_T }	A→BC A→α	**A**, **B** y **C**: Símbolos no terminales **.α**: Símbolo terminal. Los **árboles de derivación binarios**, excepto en las derivaciones.

Algoritmo

Función FNC (P, G): P'

- **P**: Entrada a la función; producción de la forma **A ::= x**
- **G**: Entrada a función, gramática definida por (Σ_T, Σ_N, S, P)
- **P'**: Salida de función, producciones en que P está en FNC

P'::= P

Si x∈Σ_T **ó** x = BC (B, C∈ Σ_N)
Entonces devolver P'

Sino Si x = ay (a ∈ Σ_T, y ∈ Σ^+)
Entonces Si ∃ (C ::= a) ∈ P y Д(C ::= x) ∈ P, x ≠ a
Entonces N:= C
Sino N:= nuevo- Σ_N;
P':= P' U { (N::= a)}
Si y ∈ Σ_N,
Entonces devolver P':= P' U {(A::= Ny)} – { P }
Sino devolver FNC-auxiliar (A, N, y, G, P')
Sino (* x=By (B ∈ Σ_N, y ∈ Σ^+)*)
Devolver FNC-auxiliar (A, B, y, G, P')

Función FNC-Auxiliar (A,N, y, G, P): P'

- **A**: Símbolo NoTerminal a izquierda de la producción inicial
- **N**: Símbolo NoTerminal 1º a derecha de la nueva producción.
- **.y**: Resto de los símbolos a la derecha de nueva producción.
- **P**: Conjunto de producciones
- **G**: Gramática definida por (Σ_T, Σ_N, S, P)
- **P'**: Nuevo conjunto de producciones.

Si y∈ Σ_T
Entonces Si ∃ (B::=y) ∈P y Д(B::=z) ∈P,z ≠ y
Entonces devolver P':=P'U{(A::=NB)}–{P}
Sino N':= nuevo- Σ_N;
Devolver P':= PU{(N'::= y),(A::=NN')}–P
Sino N':= nuevo- Σ_N;
P':= (N'::= y)
P':= P U { P', (A::= NN')} – P
Devolver FNC (P', G) donde P_c es P'

Ejemplo

Llevar la gramática G=({0,1,2}, {A,B,C}, A, P) a FNC

P = { (A::=CB2), (A::=1B)(A::= λ),
(B::=BC), (B::=1),
(C::=2) }

- Como A::=CB2 no está en FNC, y C∈Σ_N, se llama a la función auxiliar FNC-auxiliar con los argumentos **(A, C, B2, G, P)**.
- Como B2∉Σ_T se crea un nuevo no terminal **D**, se crea nueva regla **D::=B2**, se elimina la regla **A::=CB2** y se crea una nueva regla **A::=CD**.
- **A::=CD** ya está en FNC, pero **(D::=B2)** hay que volverla a tratarla.
- Activar recursivamente a FNC, activará a FNC auxiliar con argumentos **(D,B,2,G,P)**.
  En este caso 2∈Σ_T y se da cuenta que existe un no terminal **C** que lleva a **2** y a ninguna otra parte derecha, por tanto borra **D::=B2** y añade la regla **D::=BC**.

- Como A::=1B no está en FNC, 1∈Σ_T y no hay un no terminal que solo vaya a 1, se crea uno nuevo **E**, una nueva producción **E::=1** y dado que **B**∈ Σ_N elimina la regla **A::=1B** y añade la regla **A::=EB**
- A::= λ Ya está en FNC.
- B::= BC Ya está en FNC.
- B::= 1Ya está en FNC.
- C::= 2 Ya está en FNC.

Por tanto la gramática quedaría:

G' = ({ 0,1, 2 }, { A, B, C, D, E }, A, P)

**P = { (A::=CD), (A::=EB), (A::= λ),
(B::=BC), (B::=1),
(C::=2), (D::=BC), (E::=1) }**

Una G_{IC} tiene FNG si:

- Σ_N de G_{IC} solo contiene no terminales útiles
- Las producciones de G_{IC} son de forma $X \rightarrow \sigma \Upsilon$ donde $\sigma \in \Sigma_T$ y $\Upsilon \in \Sigma^*_N$ y las reglas tienen como 1er símbolo en sus derechas un símbolo terminal, seguido por una palabra de no terminales.
- Para cualquier L_{IC} existe una gramática en FNG, que genera el lenguaje.

Ejemplo: Sea L un L_{IC} y $G = (\Sigma_N, \Sigma_T, P, \$)$ una gramática que genere L es decir $L=(L_G)$ Se asume: $\varepsilon \notin L$:
Para construir una gramática equivalente en FNG:

- Sustituir las reglas recursivas a la izquierda: $X \rightarrow XY$
- Establecer un orden de variables: $\Sigma_N=\{X_1, X_2, .. X_n\}$, así las reglas serán tipo $X_i \rightarrow X_j Y$ con $i<j$ $Y \in \Sigma_N$.
- Sustituir las reglas sin símbolo terminal como 1er símbolo en su parte derecha.

Las gramáticas después de cada paso serán: $G = G_0, G_1, G_2, .. G_5 = G_{FNC}$ respectivamente.

Proceso	Operaciones
Gramática inicial: $G_5 = (\{\$, A, B, C, D, E, F\}, \{a, b, c\}, P_0, \$)$, donde P_0: $\$ \longrightarrow bDD \mid Ca \mid bc$ $A \longrightarrow B \mid aCC \mid baD$ $B \longrightarrow cBD \mid \epsilon \mid AC$ $C \longrightarrow bD \mid aBA$ $D \longrightarrow CD \mid a \mid EF$ $E \longrightarrow Eb$ $F \longrightarrow a$ Transformación a FNC. Entonces P_1: →	$\$ \longrightarrow CW_a \mid W_b\$_1 \mid W_bW_c$ $\$_1 \longrightarrow DD$ $A \longrightarrow AC \mid W_cB_1 \mid W_aA_1 \mid W_bA_2 \mid W_bD \mid W_aC_1 \mid W_a$ $A_1 \longrightarrow CC$ $A_2 \longrightarrow W_aD$ $B \longrightarrow AC \mid W_cB_1 \mid W_bD \mid W_aC_1 \mid W_a$ $B_1 \longrightarrow BD \mid CD \mid W_a$ $C \longrightarrow W_bD \mid W_aC_1 \mid W_a$ $C_1 \longrightarrow AC \mid BA \mid W_aA_1 \mid W_bA_2 \mid W_cB_1 \mid W_bD \mid W_aC_1 \mid W_a$ $D \longrightarrow CD \mid W_a$ $W_a \longrightarrow a$ $W_b \longrightarrow b$ $W_c \longrightarrow c$
Reordenar para que aparezcan las partes derechas con variables al principio del comienzo de las listas. En cada producción recursiva a la izquierda de tipo $X \rightarrow \alpha X$ con $X \in \Sigma_N$ y $\alpha \in \Sigma$ se realiza: 1. Sustituir $X \rightarrow X\alpha$ por $X \rightarrow \alpha Y$ donde Y nueva variable 2. Agregar reglas $Y \rightarrow \alpha Y \mid \alpha$ a cada regla $X \rightarrow \beta$ se añade $X \rightarrow \beta Y$ si β no comienza con X 3. En P_1 hay una regla recursiva a la izquierda: $A \rightarrow AC$, sustituir por $A \rightarrow CA_3$. Añadir $A_3 \rightarrow CA_3 \mid C$ y agregar las reglas para A. Resulta el conjunto P_2:	$\$ \longrightarrow CW_a \mid W_b\$_1 \mid W_bW_c$ $\$_1 \longrightarrow DD$ $A \longrightarrow AC \mid W_cB_1 \mid W_aA_1 \mid W_bA_2 \mid W_bD \mid W_aC_1 \mid W_a$ $A_1 \longrightarrow CC$ $A_2 \longrightarrow W_aD$ $B \longrightarrow AC \mid W_cB_1 \mid W_bD \mid W_aC_1 \mid W_a$ $B_1 \longrightarrow BD \mid CD \mid W_a$ $C \longrightarrow W_bD \mid W_aC_1 \mid W_a$ $C_1 \longrightarrow AC \mid BA \mid W_aA_1 \mid W_bA_2 \mid W_cB_1 \mid W_bD \mid W_aC_1 \mid W_a$ $D \longrightarrow CD \mid W_a$ $W_a \longrightarrow a$ $W_b \longrightarrow b$ $W_c \longrightarrow c$

Las reglas en P_2 tienen diferentes longitudes en sus derechas. Dado que la gramática en FNG se genera con cada producción exactamente un símbolo terminal, cada palabra derivable con tal gramática tiene una derivación igual a la longitud de la palabra.

GRAMÁTICA SUCIA: El árbol de derivación posee reglas de derivación o símbolos no deseables, tales como:

- **REGLAS INNECESARIAS**: Reglas de la forma A → A.
- **NO TERMINALES SIN DERIVACIÓN**: Con producciones que no derivan en ninguna cadena de terminales.
- **SÍMBOLOS INACCESIBLES**: Símbolos a los cuales nunca se puede llegar produciendo sobre S.

GRAMÁTICA BIEN FORMADA: No posee reglas:

- **No generativas** de formato A→λ, con A≠S. El no terminal que no es símbolo inicial da lugar a una regla no generativa, es un no terminal anulable.
- **De redenominación**: O producciones unitarias, de formato A → B, con A ≠ B y ambos pertenecientes a N.

G_{IC}: LIMPIEZA Y NORMALIZACIÓN

Como una $\mathbf{G_{IC}}$, puede presentar símbolos y producciones no deseables o inútiles, que permitien la existencia de árboles de derivación innecesariamente complejos o inútilmente sencillos, se hace necesario lograr una forma normal, como las siguientes técnicas de simplificación y limpieza de gramaticas, que plantean: "**Si un lenguaje se genera a partir de una gramática, también podrá generarse por otra más eficiente, de menos símbolos, sentencias, producciones y pasos**".

I. **ELIMINACIÓN DE SÍMBOLOS NO TERMINALES** (SNT: símbolo no terminal)**:** Suprime símbolos que no producen derivaciones.

Ejemplo: **S → B e | D**
B → e
D → "no posee producción"

Transformar la $\mathbf{G_{IC}}=(\Sigma_T,\Sigma_N,S,P)$ = (N,∑,S,P) a otra equivalente más eficiente G'=(N',∑,S,P'), con **SNT** que deriven cadenas de símbolos terminales, para evitar el problema del **árbol incompleto**, Así: **L(G) = L(G') | ∀ A ∈ N', A ⇒* w, w ∈ ∑*** , aplicando el siguiente algoritmo:

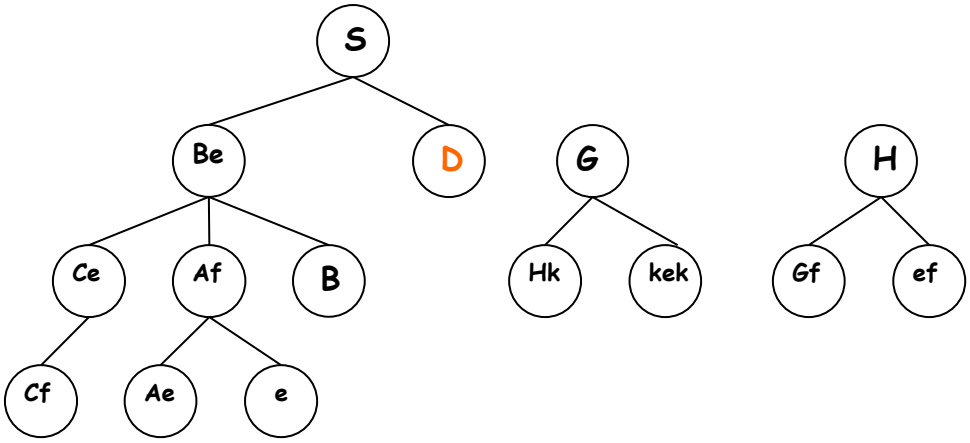
- Eliminar el SNT para el que no es posible construir una derivación que lo convierta en una cadena de terminales: **SNT = { A ∈ N : ¬ ∃ A →*w con w ∈ ∑* }**

- G = (∑,N,P,S) → G' = (∑,N',P',S) con L(G) = L(G') y sin SNT

  - Inicializar N' con los SNT A para los cuales A → w es una producción de G con w ∈ ∑*.
  - Inicializar P' con todas las producciones A → w para las cuales A ∈ N' y w ∈ ∑*.
  - Repetir hasta que no se puedan añadir más SNT a N':
    Añadir a N' los A para los cuales A → w sea una producción de P tal que w ∈ (N' ∪ ∑)*.
    Añadir esa producción a A → w a P'.

Ejemplo: Limpiar la $\mathbf{G_{IC}} = (\sum, N, P, S) = (\{e, f, k\}, \{A,B,C,D,G,H\}, P, S)$ para las producciones:

P :
S ::= B e | **D**
A ::= A e | e
B ::= C e | A f | B
C ::= C f
G ::= H k | k e k
H ::= G f | e f



Eliminar del árbol el SNT sin producciones **D**, aplicando el algoritmo de eliminación de SNT.

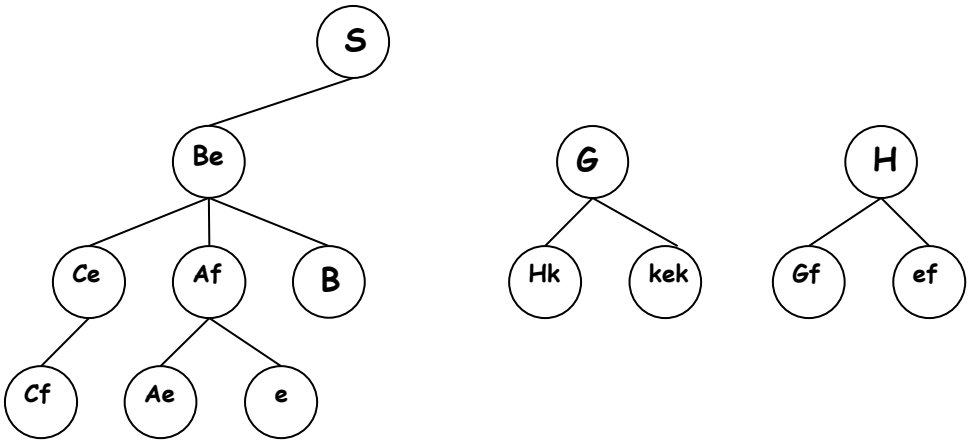
1) Inicializar la colección de SNT N' con todos los SNT A para los que A→w, es una producción de G, con w ∈ Σ*, de manera que: N'= {A}
2) Inicializar P' con las producciones A→w para las cuales A ∈ N' y w ∈ Σ*.
3) Repetir hasta que no se puedan añadir más SNT a N'.
   Añadir a N' los SNT A para A→w, y w→(N'∪Σ)* que sea producción de P y añadirla a P'

- S ::= B e
- A ::= A e | e
- B ::= C e | A f | B
- C ::= C f
- G ::= H k | k e k
- H ::= G f | e f

N' = {S, A, B, C, G, H}: Falta el no terminal **D** que no poseía producción alguna.

Luego de aplicar el algoritmo queda G'= (∑, N', P', S) = ({e,f,k}, {A,B,C, G, H, S}, P, S):

P :
S ::= B e
A ::= A e | e
B ::= C e | A f | B
C ::= C f
G ::= H k | k e k
H ::= G f | e f



II. ELIMINACIÓN DE SIMBOLOS INACCESIBLES O NO ALCANZABLES: SNA

Vemos que con P = (S::= B e, B::=e, **C** ::= **a**), de la GIC: G = (N, ∑, S, P) = ({ S, B, C },{a, e}, S, P), el terminal **a** y el no terminal **C**, no serán usados. Para eliminar este defecto, requiere que todo símbolo terminal o no terminal de una nueva gramática equivalente G' = (N', ∑', S, P'), se afecte a partir de sucesivas derivaciones sobre el axioma S, para que todo símbolo sea útil, de manera que:

- L(G) = L(G').
- ∀ X ∈ (N' ∪ ∑'), S ⇒+ α X β,
  para subcadenas α y β de la forma (N' ∪ ∑')*

El algoritmo requiere:
Inicializar N' con SNT S, luego P' y ∑' sin ningún elemento.
N'={S}
P' = ∅ (sin elemento)
∑' = ∅

Agregamos las producciones de elementos de N' a las producciones P' y los elementos de las producciones a N' y ∑' hasta no poder agregar ninguna producción, quedando:

P' = {S→B e, B→e }
N' = {S,B}
∑' = {e}

En general el algoritmo será:

- Eliminar todo SNA, inacesible desde el axioma: SNA = {A∈N: ¬ ∃S →*wAv con w, v∈(∑ ∪ N)* }
- G = (∑, N, P, S) → G'= (∑', N', P', S) con L(G) = L(G') y sin SNA
  I. Inicializar N' de modo que contenga el símbolo inicial S, e inicializar P' y ∑' a ∅.
  II. Repetir hasta que no se pueda añadir nuevas producciones
  Para A ∈ N', si A→w es una producción de P, entonces:
  - Introducir A→w en P'
  - Para todo no terminal B de w, introducir B en N'.
  - Para todo terminal σ de w, introducir ∑'.

Ejemplo: Aplicar el algoritmo para eliminar símbolos no alcanzables (SNA), al problema anterior:

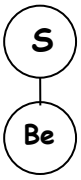
I. Inicializar N' de forma que contenga el símbolo inicial S, e inicializar P' y ∑' a ∅.
N' ={S} (Colección de no terminales inicializados solo con S)
P' = ∅ (Colección de producciones inicializadas sin elementos)
∑' = ∅ (Colección de terminales inicializados sin elementos)

II. Repetir hasta que no se pueda añadir nuevas producciones
Para A ∈ N', si A::=w es una producción de P, entonces:
a) Introducir A::=w en P'
b) Para todo no terminal B de w, introducir B en N'.
c) Para todo terminal σ de w introducir ∑'.

Paso 1:
S ::= B e
N' = {S, B} (No terminales: se agrego "S" y "B")
∑' = {e} (Terminales: se agrego "e")

Ingresando la producción generada por "S", ingresamos S y B al conjunto de los no terminales (N') y como "B" contiene en su producción al terminal "e" este se agrega a el alfabeto ∑'.

El árbol será:

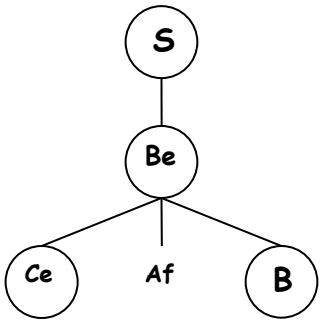


Paso 2:

B ::= C e | A f | B
N' = {S, A, B, C} (No terminales: se agregó "A" y" C")
∑' = {e, f} (Terminales: se agregó "f")

Se sumó la producción generada por B la cual ya esta incluida en N'. La producción de "B" contiene los no terminales "C" y "A" los cuales serán agregados a N' Además el terminal "f" se encuentra en la producción por lo cual es agregado al alfabeto ∑'.

El árbol será:

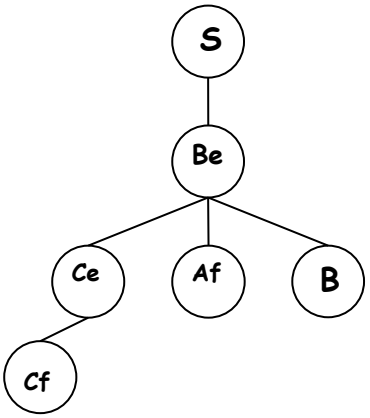


Paso 3:

C ::= C f
N' = {S, A, B, C} (No terminales: no varia del paso 2)
∑' = {e, f} (Terminales: no varia del paso 2)

Se agrego la producción del no terminal **C** que no generará cambios a la gramática, pués sus producciones que esta produce posee como símbolos no terminales y terminales a símbolos existentes dentro de la gramática (C y f).
A ::= A e | e
N' = {S, A, B, C} (No terminales: no varía de pasos anteriores)
∑' = {e, f} (Terminales: no varía de pasos anteriores)
No modifica el estado anterior

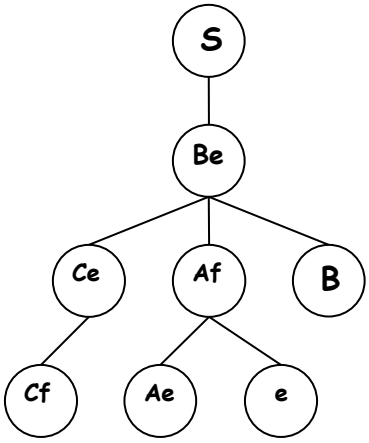
El árbol será:



Paso 4:

Finalmente:
Al no poder agregar más producciones el algoritmo finaliza y la gramática final será: G' = (∑', N' ,P', S) = ({e,f},{A,B,C,S}, P', S)
P': S::= B e
A ::= A e | e
B ::= C e | A f | B
C ::= C f

En G', ya no están las producciones que generaban los no terminales **G** y **H**, ya no están en la colección de no terminales, al igual que el terminal **k**, que al no ser alcanzable **G**, tampoco podrá ser alcanzado



III. ELIMINACIÓN DE NO TERMINALES ANULABLES: NTA

Es anulable A, si desde ella se deriva la cadena vacía: Anulables = { A ∈ N : A ::=*λ}, tal que el SNA permite A::=* λ, deseable solo si λ ∈ L(G); y como la única producción necesaria es S::= λ., deben eliminarse las demás producciones de tal tipo. A tal fin:

Se identifica el conjunto Z de NTA de una gramática:

- Inicializar Z con los no terminales A para los cuales existe la producción de la forma A::=λ.
- Repetir hasta que no puedan añadirse más no terminales a Z:
  Si B::= w tal que w ∈ (N ∪ ∑)* y todos los símbolos de w están en Z, agregar B en Z.

Identificados los NTA, para las producciones $B::=X_1X_2...X_n \in P$, se agregan en P' las combinaciones posibles de $B ::= Y_1Y_2...Y_n$ tal que:

- Con X_i no anulable, solo se considere $Y_i = X_i$.
- Con X_i anulable, se consideren las posibilidades $Y_i = X_i$ y además $Y_i = \lambda$. Es decir, un anulable en P da lugar a dos producciones en P'.
- Y_i no es λ para todo i (si así lo fuera, se estaría introduciendo una producción λ).

Notar que se sustituyen las producciones $B::=X_1 \dots X_n$ por las formas posibles, obtenidas al considerar los sucesivos X_i que sean anulables. (si hay m X_i anulables, 2^m - 1 formas ; B::= ε no se considera).

IV. ELIMINACIÓN DE PRODUCCIONES UNITARIAS o NO GENERATIVAS (PNG)

La PNG, tiene la forma A::=B y no generan nada dentro del lenguaje y para eliminarla, el algoritmo simplifica producciones de formato A1 ::= A2 ::=... An, transformándolas a una producción A::= An., evitando que el árbol de derivación sea innecesariamente delgado y profundo. Su proceso es:

- Define el **Conjunto Unitario** de cada no terminal **A**, como conjunto de SNT B, tal que según P, exista la producción A ::=* B utilizando solo producciones unitarias.

  Para esto, por definición, el primer elemento del conjunto Unitario(A) es el elemento {A}.

  Por lo tanto: Para A ∈ N se define: Unitario(A) = {B N | A ::=*B usando solo producciones unitarias}

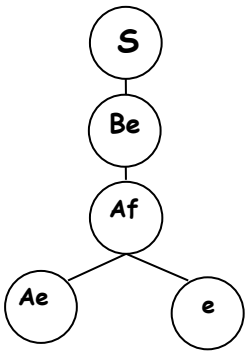
- Luego de identificar los conjuntos unitarios de los no terminales de G = (N, ∑, S, P), se obtiene una gramática equivalente G' = (N, ∑, S, P') de modo que P' carece de producciones unitarias, osea que:

  G = (∑, N, P, S) → G'=(∑, N, P', S) con L(G) = L(G') y sin PNG

  - Inicializar P' de forma que contenga todos los elementos de P, o sea inicializar P' = P
  - Para cada A, obtener el conjunto Unitario (A).
  - Para cada A ∈ N para el cual Unitario(A) ≠ {A}
  - Para cada B ∈ Unitario(A) y para cada producción no unitaria B::= w de P, añadir en P', bajo la forma A::= w.
  - Eliminar todas las producciones unitarias que haya en P'.

Ejemplo: Eliminar PNG del problema anterior:

Inicializar P' de forma que contenga todos los elementos de P del problema anterior.

Para cada: A, obtener el conjunto Unitario (A) A para el cual Unitario (A) ≠ {A} B∈Unitario y para cada producción no unitaria B→w de P añadir A→w a P'	**P': S::= B e; A::= A e | e B::= C e | A f | B C::= C f**
Eliminar todas las producciones unitarias que haya en P'	**P': S::= B e; A::= A e | e B::= A f**
La nueva gramática será: G' = (∑',N',P',S)=({e,f},{A, B, S},P',S) Y la colección final de producciones:	**P': S::= B e; A::= A e | e B::= A f**
Se eliminó la producción no alcanzable de C, no generativa, eliminada cuando está en B	

Ejemplo: Limpiar, con la técnica del algoritmo de producciones no generativas la gramática

GIC G = (∑, N, P, S) = ({a, b},{S, A, B, C, D},P, S), para las producciones

P : S::= A | A a; A::= B; B::= C | b; C::= D | a b; D::= b

	Proceso	Ejemplo
I	Detectar los unitarios de cada uno de los símbolos no terminales:	S::= A (producción unitaria o no generativa del no terminal S) A::= B (producción unitaria o no generativa del no terminal A) B::= C(producción unitaria o no generativa del no terminal B) C::= D (producción unitaria o no generativa del no terminal C)
II	Reemplazar en la producción unitaria, generada por el no terminal "C" por la producción C::= b (producción generada por el no terminal D)	Queda: C::= b | ab (Nueva producción del no terminal C)
III	Iterar el mismo proceso para la producción generada por el no terminal "B" y luego reemplazar por **B::= b | ab**,	Queda: B::= b | ab | b , pero como es redundante la existencia de dos "b" queda: B::= ab | b (Nueva producción del no terminal C) Para la producción no generativa de "A" reemplazar por A→b | ab, queda: A::= b | ab (Nueva producción del no terminal C)
IV	Realizar el mismo proceso para la producción no generativa generada por el símbolo inicial "S" donde reemplazar por **S::= b | ab** queda: **S::= b | ab | Aa** (Nueva producción del no terminal C)	Terminado este proceso las producciones no generativas se eliminaron y las producciones serán: **S::= b | a b | A a** **A::= b | ab** **B::= a b | b** **C::= b | a b** **D::= b**
	Luego la gramática queda definida como: **G'=(∑,N,P',S))=({a,b},{S,A,B,C,D},P,S)**, con las siguientes producciones resultantes luego de aplicar el algoritmo:	**P' : S::= b | a b | A a** **A::= b | ab** **B::= a b | b** **C::= b | a b** **D::= b**

Ejemplo: La $\mathbf{G_{IC} = (\Sigma_T, \Sigma_N, S, P)}$ **= (**{ x, y },{ **S, A, B** }, S, P**)**
genera el **LIC**: $\mathbf{L_2 = \{ x^n y^m \mid m, n \geq 0 \text{ y } m=n \text{ ó } m=2n \} = \{ x^1 y^1, x^1 y^2, x^2 y^2, x^3 y^3, \ldots \}}$:

Ejemplo: La $\mathbf{G_{IC}}$ para generar el $\mathbf{G_{IC}= \{ a^n b^m c^{m+n} \mid n \geq 0,\ m \geq 0 \}}$ tiene: **S → aSc | B; B → bBc | E**

m = n
S::= x A y | λ
A::= x A y | λ

m = 2n
S::= x B y y | λ
B::= x B y y | λ

.m = n y m = 2n:
S::= x A y | x B y y | λ
A::= x A y | λ
B::= x B y y | λ

Limpiando estas producciones:
S::= x A y | x B y y | x y | x y y | λ
A::= x A y | x y
B::= x B y y | x y y

Con las **derivaciones**:
- **S** → xy = $\mathbf{x^1 y^1}$
- **S** → xyy = $\mathbf{x^1 y^2}$
- **S** → xAy → xxyy = $\mathbf{x^2 y^2}$
- **S** → xAy→ xxAyy → xxxyyy = $\mathbf{x^3 y^3}$
- **S** → xByy→ xxyyyy = $\mathbf{x^2 y^4}$
- **S** → xByy→ x xByy yy → xx xyy yyyy = $\mathbf{x^3 y^6}$

PRINCIPIO DEL PALOMAR

La inspiración para el nombre del principio: aves en un palomar. Aquí n = 7 y m = 9

El principio del palomar, o **principio de Dirichlet** (1834 con el nombre de Schubfachprinzip "principio de los cajones"), establece que si **n** palomas se distribuyen en **m** palomares, y si **n** > **m**, entonces al menos habrá un palomar con más de una paloma.

Sean **m**, **n** y **p** tres números naturales. Si se desean colocar **np + m** palomas en **n** cajas, alguna caja debe contener al menos **p + 1** objetos.
Demostración: Si cada caja contiene como mucho **p** objetos, el número total de objetos que podemos colocar es: **np < np + 1 ≤ np + m**.

Este principio dice que no puede existir una aplicación inyectiva entre un conjunto de **m** elementos y otro de **n** elementos, si **m** > **n**. Equivalentemente, si se desean colocar m objetos en **n** cajas, con **m** > **n**, al menos una caja debe contener al menos 2 objetos.

FORMULACIÓN MATEMÁTICA:
Si A y B son conjuntos finitos con **|A|>|B|** entonces no existe ninguna función inyectiva de A en B.

Demostración por inducción
Paso base:
Si **|B|=0**, es decir, **B=0**. No existe ninguna función f: **A→B**, en particular no existe ninguna función inyectiva.

Hipótesis inductiva:
f: **A→B** no es inyectiva para todo conjunto finito A y para todo conjunto finito B, que cumplan **|A|>|B|** y **|B| <= n**, con **n > = 0**.

Tesis inductiva:
Para **|A|>|B|= n+1** , no existe una función f: **A→B** inyectiva.

Demostración del paso inductivo:
Como A no es vacío, elijamos un $a \in A$. Pueden ocurrir dos cosas.

- Existe otro elemento distinto a a en A, llamémosle a' que cumpla f(a) = f(a').
- No existe tal elemento. Si el caso es que existe, la función f no es inyectiva y termina la demostración.

Tomemos el caso que no existe, entonces f(a) tiene solo una preimagen que es a.
Consideramos la función $g: A - \{a\} \to B - \{f(a)\}$
que coincide con f en todos los elementos de A − {a}.

Ahora aplicamos la hipótesis inductiva pues:
B − {f(a)} tiene n elementos y $|A - \{a\}| = |A| - 1 > |B| - 1 = |B - \{f(a)\}| = n$,
por tanto g no es inyectiva. Como g no es inyectiva, f no es inyectiva, que es lo que queríamos demostrar.

GRAMATICA tipo G_3 : REGULAR G_{RE}

O Gramatica lineal estructurada por frases, definida como la cuatrupla: $G_3=G_{RE}=(\Sigma_T, \Sigma_N, S, P)$

Componentes
• Σ_T: Alfabeto de **símbolos terminales**.
• Σ_N: Alfabeto de **símbolos no terminales**, donde $\Sigma=\Sigma_T \cup \Sigma_N$ y $\Sigma_T \cap \Sigma_N=0$
• **S**: Símbolo Inicial o **Axioma** de la gramática, es un símbolo no terminal: $S \in \Sigma_N$
• **P: Producciones: La longitud de su parte izquierda debe ser igual a uno y no terminal**, mientras que la parte derecha puede ser una secuencia de solo terminales, o de terminales **con un no terminal como sufijo o como prefijo** Pero el **FORMATO ESTÁNDAR** es: $N_1 \rightarrow tN_2$, $N \rightarrow t$, $S \rightarrow \lambda$, y puede ser: ▪ **LINEAL por DERECHA** G_{RED}: $P \subseteq \Sigma_N \times \Sigma_T^* (\Sigma_N \cup \lambda)$ $P = \{ (S ::= \lambda)$ ó $(A ::= aB)$ ó $(A ::= a) \mid A, B \in \Sigma_N , a \in \Sigma_T + \}$ ▪ **LINEAL por IZQUIERDA** G_{REI}: $P \subseteq \Sigma_N \times (\Sigma_N \cup \lambda) \Sigma_T^*$ $P = \{ (S ::= \lambda)$ ó $(A ::= Ba)$ ó $(A ::= a) \mid A, B \in \Sigma_N , a \in \Sigma_T + \}$

Ejemplo: La G_{RE}: $G_3=(\Sigma_T, \Sigma_N, S, P) = (\{ 0,1 \}, \{ A, B\}, S, P)$

Producciones		Derivaciones		
Pares	Simplificado	A→1B→11	A→1B→10C→101	A→1B→11C→111
P = {(A::=1**B**), (B::=1), (B::= 0**C**), (B::=1**C**), (C::= 1)}	P : **A** ::= 1**B** **B**::= 1 |0**C**| 1**C** **C** ::= 1	**A** / \ 1 **B** | 1	**A** / | 1 **B** / \ 0 **C** | 1	**A** / | 1 **B** / \ 1 **C** | 1
Genera en **formato estandar** el **lenguaje regular**: $L_{RE} = \{ w^n \mid w = w^{-1} \} = \{11, 101, 111, . . \}$				

Ejemplo: G_{RE}: $G_3 = (\Sigma_T, \Sigma_N, S, P) = (\{a, i, o, u, c, f, l, n, p, r, t, w \}, \{ A, B, S \}, S, P)$, con

Producciones		Derivaciones
Pares	Simplificado	
P={(**S**::= ca **B**), (**S**::= wi **A**) (**S**::= utn), (**S** ::= frt) (**A**::= pi **B**), (**A**::=lo) (**B**::= r **A**), (**B**::=o) }	**S** ::= ca **B** | wi **A** | utn | frt **A** ::= pi **B** | lo **B** ::= r **A** | o	**S** → ca**B** → car**A** → carpi**B** → carpio **S** → wi**A** → wilo **S** → utn

Lenguaje generado es regular sin formato estandar: L_{RE} = { **carpio**, **wilo**, **utn**, **frt** , . . }

CONCLUSION: Notemos que todas las gramáticas pueden generar el mismo lenguaje, por lo tanto son equivalentes. O sea que: $G_3 \subseteq G_2 \subseteq G_1 \subseteq G_0$

FE: FORMATO ESTANDAR

Recordemos que:

Gramáticas de tipo 0

Llamadas gramáticas sin restricciones o gramáticas recursivamente enumerables cuyas producciones tienen la forma:

$$xAy ::= v \quad \text{donde} \quad A \in \Sigma_N,\ x,y,v \in \Sigma^*$$

Cualquier lenguaje de tipo 0 puede ser también generado por las gramáticas con estructura de frase, pues ambas poseen el mismo poder generativo. Las producciones de las gramáticas con estructura de frase tienen la forma:

$$xAy ::= xvy \text{ donde } A \in \Sigma_N,\ x,y,v \in \Sigma^*$$

Los formatos gramáticales pueden simplificarse transformando sus reglas para obtener una gramática equivalente, reemplazando los terminales que no poseen **FE** llevándolos a **Formatos Estandares**

TIPO	DESCRIPCION	Ejemplo
FET0: 0 o Irrestrictas	Incluyen reglas "nuevo no terminal deriva a terminal correspondiente". Así, si **N** y **M** son no terminales y **t** es terminal: • $N_1 N_2 \ldots N_i \rightarrow M_1 M_2 \ldots M_i \mid \lambda$ • $N \rightarrow t$	Para: $G_i = (\Sigma_T, \Sigma_N, S, P)$ $= (\{0,1\}, \{ J,K,L,M,X,Y,Z,S \}, S, P)$ las producciones P son: • $S \rightarrow LMX$ $\quad KM \rightarrow MK$ • $ZX \rightarrow MKX$ $\quad YX \rightarrow MJX$ • $YK \rightarrow KY$ $\quad YJ \rightarrow JY$ • $ZK \rightarrow KZ$ $\quad ZJ \rightarrow JZ$ • $X \rightarrow \lambda$ $\quad JM \rightarrow MJ$ • $LM \rightarrow KLZ \mid JLY \mid \lambda$ $\quad J \rightarrow 0$ • $K \rightarrow 1$

Gramáticas de tipo 1

Llamadas gramáticas dependientes del contexto y sus producciones poseen la forma:

$$xAy ::= xvy \text{ donde } A \in \Sigma N, v \in \Sigma^+ \ x,y \in \Sigma^*$$

Estas gramáticas son también gramáticas con estructura de frase, con la restricción que la longitud de la parte derecha de las producciones es siempre mayor o igual que la de la parte izquierda, luego, no hay producciones compresoras.

Sus producciones pueden interpretarse como que **A** se convierte en **v**, siempre que se encuentre en un contexto, precedida por **x**, seguida por **y**.

TIPO	DESCRIPCION	Ejemplo
FET1: 1 o Dependientes de Contexto	Incluyen además regla: Compresora del axioma **S → λ**, Donde **N** puede reemplase por **β** en caso que **N** esté en el contexto de **α** • **α_1 N α_2 → α_1 N α_2** • **S → λ** Para transformar las reglas a FE: De las reglas a transformar, lograr un **FET0**. Agregar a los viejos No Terminales: **N_1 N_2. . . N_i → M_1 M_2. . . M_i,** nuevos No Terminales **Y_1 Y_2. . . Y_i.** Reemplazar por las producciones: **X_1X_2... X_L → Y_1X_2. . . X_L,** **$Y_1X_2X_3$... X_L → $Y_1Y_2X_3$. . . X_L,** . . . . . **Y_1Y_2...Y_{L-1}...X_L→Y_1Y_2...$Y_{L-1}Y_LZ_{L+1}$ ...Z_K,** **Y_1Y_2..$Y_{L-1}Y_LZ_{L+1}$..Z_K→Z_1Y_2..Y_LZ_{L+1}..Z_K,** . . . . . **Z_1Z_2...$Z_{L-1}Y_LZ_{L+1}$... Z_K→Z_1Z_2...Z_K,**	En: **G_{DC} = (Σ_T, Σ_N, S, P) =** **({a, b, c},{B,D,T,S},S,P)** que genera las cadenas del lenguaje L = { **a^n b^n c^n| n≥1**} Con las reglas **P**: **S→T; DB→BD; D→c;** **T→aTBD | abD; bB→bb** ○ Como la regla: **DB → BD** no cumple con el **FE** y está en el **FET0**, le agregamos los nuevos no terminales **EF** y reemplazamos por las reglas: **DB→ED, ED→EF, EF→DF, DF→BD** ○ Luego la gramática equivalente es: **G_i = (Σ_T, Σ_N, S, P) =** **({a, b, c},{B,D, E,F,T,S},S,P)** con las reglas P: **S →T; DB→ED, ED→EF,** **EF→DF, DF→BD; D→c;** **T→aTBD|abD; bB→bb**

Para agregar a una **G_{DC}** la generación de la **palabra vacia λ**, planteamos la alternativa de:

- **Agregar la regla de borrado S → λ :**
  Ejemplo:
  La: **G_{DC}=(Σ_T, Σ_N, S, P)=({a, b, c},{B,D,T,S},S,P)**
  con las reglas: **P: S→T; DB→BD; D→c; T→aTBD \| abD; bB→bb**
  Genera las cadenas del lenguaje **L = { a^n b^n c^n\| n≥0}**

  La gramatica resultante sería: **G_{DC} = (Σ_T, Σ_N, S, P) = ({a, b, c},{B,D,T,S},S,P)**
  con las reglas
  P: S→T\| λ; DB→BD; D→c; T→aTBD \| abD; bB→bb

- **Agregar un axioma ficticio S_i:**
  Ejemplo:
  Para agregar la generación de la palabra vacia **λ** y el axioma **S** aparece en la derecha de una regla:

  G_i = (Σ_T, Σ_N, S, P) = ({a,b},{A,S},S,P),

  Con las reglas: **P: S→bSbb\|SAS; Sb→aaA; A→bb; SA→aa**

  Con el artificio de introducir un nuevo símbolo **S_i**, transformamos la regla en **S_i →S \| λ**

  La gramatica equivalente es: **G_i = (Σ_T, Σ_N, S, P) = ({a,b},{A,S, S_i }, S_i,P),**

  Con las reglas: **P: S_i →S \| λ; S→bSbb\|SAS; Sb→aaA; A→bb; SA→aa**

FE: FORMATO ESTANDAR

Llevar **FET3** la: $G_{RED} = (\Sigma_T, \Sigma_N, S, P) = (\{a, b\},\{A,B,C,D,E,S\},S,P)$ que genera: $L = \{ a^n b^n c^n | n \geq 0\}$
S→abaB|bb|A|λ; A→baS|aabC|a; B→aaA|B| λ; C→baC|aC; D→bbB|aaa|abaD; E→aaE|abC

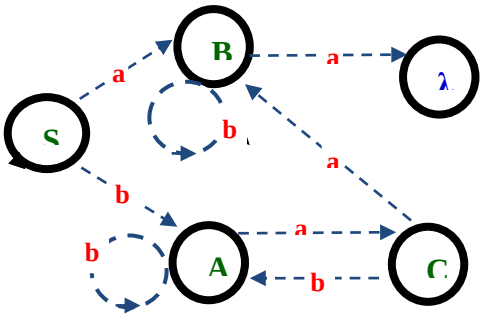
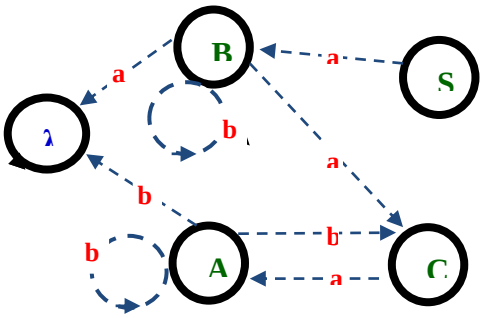
TIPO	DESCRIPCION	Ejemplo
FET3: 3 o Regular	**Eliminar:** **Reglas Innecesarias**: O producciones inocuas que tienen: -Noterminales inútiles, o **N** que no cumplen con: **S→*αNβ** y **N→+w**, donde: **w∈Σ*$_T$** y **α,β ∈ Σ*** -La forma: **N→N**	**S→abaB|bb|A|λ** **A→baS| |a** **B→aaA| | λ**
	El axioma S de la derecha, si: **λ ∈ L_{GR}**: Efectuando el artificio de agregar: • Un nuevo síbolo inicial: S_i • La regla: S_i →S | λ	**S_1→S| λ** **S→abaB|bb|A|λ** **A→baS|a** **B→aaA| λ**
	Reglas de renombrado: **N_1 → N_2**: Se reemplaza: **N_1 → N_2**, por reglas que surgen de reemplazar **N_2** por las partes derechas de sus producciones	**S_1→abaB|bb|baS|a|λ** **S→ abaB|bb|baS|a|λ** **A→baS|a** **B→aaA| λ**
	Reglas de borrado: **N→ λ**: Se reemplaza: **N→ λ**, por reglas que surgen de reemplazar **N** por **λ** en todas las reglas donde figure **N**, excepto **S→ λ**.	**S_1→abaB|bb|baS|a|ba|aba|λ** **S→ abaB|bb|baS|a|ba|aba** **A→baS|a|ba** **B→aaA**
	Reducir a 1 la longitud de secuencias de terminales de las reglas, reemplazando: Las reglas: **N_1→t_1t_2...$t_K N_2$** por Las reglas: **N_1→t_1M_1**, **M_1→t_2M_2**,..**M_{K-1}→$t_K N_2$**, donde **M_i** son nuevos no terminales.	**S_1→aF|bH|bI|bJ|aK|a|λ** **B→bG** **G→aB** **H→b** **I→aS** **J→a** **K→bJ** **S→aF|bH|bI|bJ|aK|a** **A→bI|bJ|a** **B→aL** **L→aA**

G_{RED} = (Σ_T, Σ_N, S, P) = ({a, b},{A,B,F,G,H,I,J,K,L,S,S_1},S,P).

EQUIVALENCIA ENTRE GRAMATICAS REGULARES

Estando en **FE**, puede generarse una gramatica G_{RED} equivalente de G_{REI} y viceversa.

Ejemplo Sea: $G_{REI} = (\Sigma_T, \Sigma_N, S, P)$, para P: **S→BaB|ab; A→Sa|Ab; B→Bb|a**

DESCRIPCION	Ejemplo
1.- Transformar la G_{RE}: Cuando el axioma está en la derecha de la regla Crear nuevo NoTerminal **L** Si **S** es axioma, **t ∈ Σ_T** y **N ∈ Σ_N** entonces: • Para cada regla: **S→Nt** crear una regla: **L→Nt** • Para cada regla: **S→St** cambiar por regla: **N→Nt**	Agregamos **C** nuevo terminal: **P:** **S→BaB|ab** **A→ Ca|Ab; B→Bb|a; C →Ba|Ab**
2.- Crear GRAFO DIRIGIDO: Crear un **Nodo** por NoTerminal N y por **λ** **Arco** rotulado **t**, desde el nodo **N_1** al **N_2** para cada regla de la forma: **N_1→N_2t** **Arco** rotulado **t**, desde el nodo **N** al **λ** para cada regla de la forma: **N→t** **Arco** no rotulado **t**, desde el nodo **S** al **λ**, cuando haya regla de la forma: **N→ λ**	
3.- Transformar el Grafo: • Intercambiar rotulos de los nodos **N** y **λ** • Invertir sentidos de los arcos.	
4.- Crear REGLAS: Desde la transformación del grafo, para cada arco • Rotulado **t**, desde el nodo **N_1** al **N_2** se crea la regla: **N_1→N_2t**. • Rotulado **t**, desde el nodo **N** al **λ** se crea la regla: **N→t** • No rotulado, desde el nodo **S** al **λ**, crear la regla: **S→ λ**	**S→aB** **A→bC|bA|A** **B→aC|bB|a** **C→aA**

LENGUAJES REGULARES L_{RE}

Recordemos que una G_{RE} que genera al L_{RE}, se carateriza por:

1. **DEFINICIÓN:** $G_{RE} = G_3 = (\Sigma_T, \Sigma_N, S, P)$

2. **PRODUCCIONES:**
   La **longitud de su parte izquierda debe ser uno y no terminal**, la parte derecha puede ser una secuencia de solo terminales, o de terminales **con un no terminal como sufijo o como prefijo**

   El **FORMATO ESTÁNDAR** es: $N_1 \rightarrow t\,N_2$, $N \rightarrow t$, $S \rightarrow \lambda$, y puede ser:

   - **LINEAL por DERECHA** G_{RED}: $P \subseteq \Sigma_N \times \Sigma_T^* (\Sigma_N \cup \lambda)$
     $P = \{ (S ::= \lambda)$ ó $(A ::= aB)$ ó $(A ::= a) \mid A, B \in \Sigma_N, a \in \Sigma_T + \}$

   - **LINEAL por IZQUIERDA** G_{REI}: $P \subseteq \Sigma_N \times (\Sigma_N \cup \lambda)\, \Sigma_T^*$
     $P = \{ (S ::= \lambda)$ ó $(A ::= Ba)$ ó $(A ::= a) \mid A, B \in \Sigma_N, a \in \Sigma_T + \}$

EJEMPLO: La G_{RE}: $G_3 = (\Sigma_T, \Sigma_N, S, P) = (\{ 0, 1 \}, \{ A, B\}, S, P)$

Producciones		Derivaciones		
Pares	Simplificado	A→1B→11	A→1B→10C→101	A→1B→11C→111
P = {(A::=1B), (B::=1), (B::= 0C), (B::=1C), (C::= 1)}	P : A ::= 1B B::= 1 |0C| 1C C ::= 1	A / \ 1 B | 1	A / | 1 B / \ 0 C | 1	A / | 1 B / \ 1 C | 1
Genera en **formato estandar** el **lenguaje regular**: $L_{RE} = \{ w^n \mid w = w^{-1} \} = \{11, 101, 111, \ldots\}$				

Un L_{RE} se genera por una G_{RE} si cumple con las condiciones:

1. Todo lenguaje finito es un L_{RE}
2. Si L_1 y L_2 son L_{RE}, luego $L_1 \cup L_2$ y $L_1 \cap L_2$ son también L_{RE}
3. Si L es un L_{RE} luego L^* es también L_{RE}
4. Todo L_{RE} puede definirse por las condiciones 1, 2 y 3.

Para el alfabeto $\Sigma = \{a, b\}$ Se afirma que:

Condición de **recursividad**	El lenguaje	Es regular porque cumple con la condición:
1. $\emptyset$ Es un L_{RE}.	$\emptyset$ y $\{\varepsilon\}$	1 y 2 de recursividad
2. $\{ \varepsilon \}$ Es un L_{RE}.	{a} y {b}	3 de recursividad
3. Para todo $a \in \Sigma$, {a} Es un L_{RE}	{a, b}	4, unión de los lenguajes {a} y {b}
4. Si A y B son L_{RE}s, entonces $A \cup B$, $A \cap B$ y A^* son L_{RE}s	{a . b}	Las dos razones anteriores
5. Ningún otro lenguaje Σ es regular.	{a, ab, b}	Las razones anteriores
	$\{ a^i \mid i \geq 0 \}$	4, a^i es potencia iésima de {a} que es concatenación sucesiva.
	$\{a^i b^j \mid i \geq 0, j \geq 0 \}$	Las razones anteriores
	$\{ (ab)^i \mid i \geq 0 \}$	Las razones anteriores

EXPRESION REGULAR: E_{RE}

E_{RE} es una **FORMA**	Ejemplo	
	E_{RE}	L_{RE}
• **SIMPLIFICADA** de representación del L_{RE}	Φ	{}
	$.\lambda$	$\{\lambda\}$
	$.x$	$\{x\}$
	E_1+E_2	$L_1 \cup L_2$
	$E_1.E_2$	$L_1 \cap L_2$
	E^*	L^*

• **ALGEBRAICA** definida sobre el alfabeto Σ y otro especial
$\Sigma'=\{+, *, ., \phi, \lambda, (,)\}$ si cumple las condiciones:

1. $\phi, \lambda, x \in \Sigma$ son E_{RE}
2. Si E_1 y E_2 son E_{RE}s luego:
   E_1+E_2 y $E_1.E_2$ son también E_{RE}s
3. Si E es un E_{RE} luego E^* es también E_{RE}
4. Toda E_{RE} se define por las condiciones 1, 2 y 3.

Si **r**, α, **s** y **t** son E_{RE}

$\emptyset.\alpha = \alpha . \emptyset = \emptyset$	$\alpha * .\alpha^* = \alpha^*$
$\alpha . \alpha^* = \alpha^* . \alpha$	$\emptyset^* = \varepsilon$
$\alpha^* = \lambda + \alpha .\alpha^*$	$r(s\ r)^* = (r\ s)^*\ r$
$(r^*\ s)^* = \varepsilon \cup (r \cup s)^*\ s$	
$(rs^*)^* = \varepsilon \cup r\ (r \cup s)^*\ s$	
$s(r \cup \varepsilon)^*(r \cup \varepsilon) \cup s = sr^*$	
$(a^*\ b)^* = \{a\} \cup \{b\}.\{c\}^*)^*$	
$(a^*\ b)^* = \varepsilon \cup (a \cup b\ )^* b$	

• **RECURSIVAMENTE** definida sobre el alfabeto Σ:

1. $\emptyset$ y ε Son E_{RE}s
2. $.a$ Es una E_{RE} para toda: $a \in \Sigma$
3. Si **r** y **s** son E_{RE}s, luego **r+s**, **r.s** y **r*** serán E_{RE}s
4. Ninguna otra secuencia de caracteres de Σ es E_{RE}.

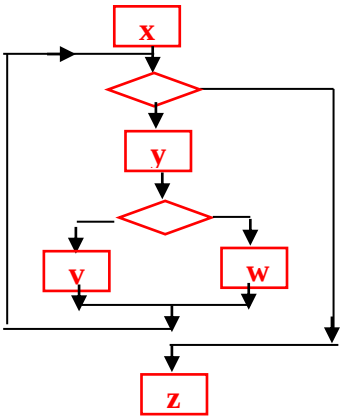
Si **r**, **s** y **t** son E_{RE}

$r . s = s . r$	$r . \emptyset = r = \emptyset . r$
$r . r = r$	$(r.s) .t = r . (s.t)$
$r\ \varepsilon = \varepsilon\ r = r$	$r\ \emptyset = \emptyset\ r = r$
$(r\ s)\ t = r\ (s\ t)$	
$r\ r^* = r^*\ r = r.r^* = r^+$	
$\lambda^* = \lambda$	$\emptyset^* = \lambda$

La E_{RE} simplifica la representación y análisis, por ejemplo, de:

- La estructura algorítmica del diagrama: **x.(y.(v+w))*.z**
- Un léxico de un lenguaje de programación: **PalabraReservada=do+while+switch+case+…** .
- Un léxico de un lenguaje numérico: **Digito=0+1+2+…**
- Un lenguaje **01*** formado por palabras iniciadas en **0**, seguido uno, o más o ningún **1**
- Cadenas **(0+1)*** de ninguno, uno o más números **0** ó **1**

x . (y . (v + w))* . z



La E_{RE} posee las siguientes

Caracteristicas	Ejemplo
• **Prioridad** de operadores	**Primero**: * (Potencia); **luego**: . (Producto) y **finalmente**: + (Unión)
• **Abreviatura**:	**{ a } = a**

• **Correspondencia**: A toda expresión regular le corresponde un lenguaje regular y viceversa.

E_{RE}	L_{RE}	E_{RE}	L_{RE}
Φ	**{}**	**a.b**	**{ a, b }**
λ	**{ λ }**	**a+b**	**{a, b }={a}∪{b}**
x	**{ x }**	**a ***	**{a*}**
E_1+E_2	$L_1 \cup L_2$	**Ø***	**{ε}**
$E_1.E_2$	$L_1 \cap L_2$	a^+	$\{a\}^+$
E*	**L***	**c*(a+b.c*)***	**{c}* ({a}∪{b}∩{c}*)***

	E_{RE}	
• **Propiedades**	**Φ*= λ**	**E*= λ +E.E***
	E.E*=E*.E	$(E_1^*+E_2^*)^* = (E_1+E_2)^* = (E_1^*+E_2)^*. E_1^*$
		$E^* = (\lambda+E^1+E^2+\ldots+E^{n-1}).(E^n)^*$

Toda E_{RE} α define recursivamente a un L_{RE}:

Si α es E_{RE}	Define recursicamente al lenguaje L_{RE}
$\alpha = \emptyset$	$L(\alpha) = \emptyset$
$\alpha = \lambda$	$L(\alpha) = \{\lambda\}$
$\alpha = a,\ a \in \Sigma$	$L(\alpha) = \{a\}$
α y β son E_{RE}	$L(\alpha + \beta) = L(\alpha) \cup L(\beta)$
α y β son E_{RE}	$L(\alpha . \beta) = L(\alpha) . L(\beta)$
α es E_{RE}	$L(\alpha^*) = L(\alpha)^*$
α es E_{RE}	$L((\alpha)) = L(\alpha)$
La E_{RE}: (01)*	$L(01^*) = L(0)\,L(1^*) = \{0\}L(1)^* = \{0\}\bigcup_{i=0}^{\infty}\{1\}^i$ $= \{0\}\{\lambda,1,11,111,..\} = \{0,01,011,0111,..\}$
La E_{RE}: (0+1)*	$L(0+1)^* = [L(0+1)]^* = [L(0)+L(1)]^* = [\{0,1\}]^*$ $= \bigcup_{i=0}^{\infty}\{0,1\}^i = \{\lambda,0,1,00,01,10,11,\}$

Si r, s y t son E_{RE} de un mismo alfabeto Σ. Se cumple que:

1. $r \cup s = s \cup r$	15 $r(s \cup t) = rs \cup rt$ y $(r \cup s)\,t = r\,t \cup s\,t$
2. $r \cup \emptyset = r = \emptyset \cup r$	16 $r^* = r^{**} = r^*r^* = (\varepsilon \cup r)^* = r^*(r \cup \varepsilon) = (r \cup \varepsilon)r^* = \varepsilon \cup rr^*$
3. $r \cup r = r$	17 $(r \cup s)^* = (r^* \cup s^*)^* = (r^*s^*)^* = (r^*s)^*r^* = r^*(s\,r^*)^*$
4. $(r \cup s) \cup t = r \cup (s \cup t)$	18 $r(s\,r)^* = (r\,s)^*\,r$
5. $r\,\varepsilon = \varepsilon\,r = r$	19 $(r^*\,s)^* = \varepsilon \cup (r \cup s)^*\,s$
6. $r\,\emptyset = \emptyset\,r = r$	20 $(rs^*)^* = \varepsilon \cup r\,(r \cup s)^*\,s$
7. $(r\,s)\,t = r\,(s\,t)$	21 $s(r \cup \varepsilon)^*(r \cup \varepsilon) \cup s = sr^*$
8. $r\ r^* = r^*\,r = r.r^* = r^+$	22 $(a^*\,b)^* = \{a\} \cup \{b\}.\{c\}^*\ )^*$
9. $\lambda^* = \lambda$	23 $.ab \cup \varepsilon \cup (a \cup b)^*\,b = ab \cup (a^*\,b)^*$
10. $\emptyset^* = \lambda$	24 $(\alpha^* . \beta^*) = (\alpha + \beta)^* = (\alpha^* . \beta)^*$
11. $\emptyset . \alpha = \alpha . \emptyset = \emptyset$	25 $(a^*\,b)^* = \varepsilon \cup (a \cup b)^*b$
12. $\alpha * \alpha^* = \alpha^*$	26 Si $\alpha = \beta^* . \gamma$ entonces: $\alpha = \beta . \alpha + \gamma$
13. $\alpha . \alpha^* = \alpha^* . \alpha$	27 $\alpha^* = \lambda + \alpha . \alpha^*$
14. $\emptyset^* = \varepsilon$	

ECUACIÓN CARACTERÍSTICA de la G_{RE}

La G_{RE} posee definición regular de formato **X=R+TX**, si $X \in \Sigma_N$ y (**R,T**) son E_{RE} sobre **Σ** efectuando:

1. Agrupar las reglas de igual parte izquierda
2. Con el conectivo "+", igualar cada noTerminal **X** con la unión de partes derechas respectivas.
3. Con el conectivo "**.**", sustituir la yuxtaposición de símbolos con su concatenación.

Ejemplo: En la $G_{RE} = (\Sigma_T, \Sigma_N, S, P) = (\{a,b,c\},\{A,B,C,S\},S,P)$

Con:

Reglas **P**	→	**Ecuaciones características**
S→aA|bB|Cc	→	**S=a.A+b.B+c.C**
A→bA|aB	→	**A=b.A+a.B**
B→cB| a	→	**B=c.B+a**
C→cC|a	→	**C=c.C+a**

E_{RE} del lenguaje generado por una G_{RE}:

Concepto	Ejemplo
Efectuar: 1. Planteo del sistema de ecuaciones catacterísticas. 2. Solución al sistema por sustitución y aplicación de solución de la ecuación genérica.	En $G_{RE} = (\Sigma_T, \Sigma_N, S, P) = (\{a,b,c\},\{A,B,C,S\},S,P)$, Con Reglas **P** → **Ecuacion característica** **S→aA|bB|cC → S=a.A+b.B+c.C** **A→bA|aB → A=b.A+a.B** **B→cB| a → B=c.B+a** **C→cC|a → C=c.C+a**
3. La E_{RE} del lenguaje generado por la G_{RE}, es solución al axioma de la gramatica.	Obtención de la E_{RE} **C = c*.c** **B = c*a** **A = b.a+a.c*.a A=b*.a.c*.a** **S = a.b*.a.c*.a+b.c*.a+c.c*.c**

ECUACIONES GENÉRICA con E_{RE}

Para determinar la E_{RE} de **X** de la ecuación genérica **X =R+T.X**, partimos de la E_{RE} de **T**

- **T*= λ +T.T***, concatenando con **R** resulta:
- **T*.R = (λ +T.T*).R**, que resolviendo:
- **T*.R = R +T.T*.R** y comparando con: **X= R +T.X**, resulta **X= T*.R**.

DERIVADA de la E_{RE} **E** respecto de $x \in \Sigma$ se define como: $D_x(E) = \{ w \mid x.w \in E \}$

Concepto	Ejemplo
PROPIEDADES:	1. $D_x(\Phi) = \Phi$ 2. $D_x(\lambda) = \Phi$ 3. $D_x(x) = c$ 4. $D_x(y) = \Phi$ 5. $D_x(E_1 + E_2) = D_x(E_1) + D_x(E_2)$ 6. $D_x(E_1 . E_2) = D_x(E_1).E_2 + f(E_1).D_x(E_2)$ → $f(E_1)=\lambda$, si $\lambda \in E_1$, ó $f(c)=\Phi$, si λno$\in E_1$ 7. $D_x(E^*) = D_x(E).E^*$
De las palabras **w** representadas por **E**, se eligen las que comienzan con **x** respecto al símbolo que se deriva. La derivada es el conjunto de restos de tales palabras sin el prefijo **x**.	Sea la E_{RE}: $E=a^*.b.(a+b)^*.b$ $D_a(E)= D_a(a^*).b.(a+b)^*.b + f(a^*).D_a(b.(a+b)^*.b)$ $=D_a(a).a^*.b.(a+b)^*.b + \lambda.(D_a(b).(a+b)^*.b + f(b).D_a((a+b)^*.b))$ $= \lambda.a^*.b.(a+b)^*.b + \Phi.(a+b)^*.b + \Phi.D_a((a+b)^*.b)$ $= a^*.b.(a+b)^*.b + \Phi + \Phi = a^*.b.(a+b)^*.b = E$ $D_b(E)=D_b(a^*).b.(a+b)^*.b + f(a^*).D_b(b.(a+b)^*.b) =$ $= \Phi.b.(a+b)^*.b + \lambda.(D_b(b).(a+b)^*.b + f(b).D_b((a+b)^*b))$ $= \Phi + \lambda.(a+b)^*.b + \Phi D_b((a+b)^*b) = (a+b)^*.b$
COMPOSICION Las derivadas se componen como: $D_{xy}(E) = D_y(D_x(E))$	$D_{ab}(E) = D_b(D_a(E)) = D_b(E) = (a+b)^*.b$ $D_{ba}(E) = D_a(D_b(E)) = D_a((a+b)^*.b)$ $= D_a((a+b)^*.b) + f((a+b)^*).D_a(b)$ $= D_a((a+b).(a+b)^*.b + \lambda.\Phi = (a+b)^*.b$

G_{RE} del lenguaje generado por una E_{RE}:

Concepto	Ejemplo
Si E_0 es E_{RE} y **D** es conjunto de distintas E_{RE} E_i obtenidas por derivación compuesta respecto a los símbolos **x** de Σ, el alfabeto de base E_0; los componentes de la gramatica **G** generado por el lenguaje definido por E_0 son $\Sigma_N=\{ E_0\} \cup D$, $\Sigma_T=\Sigma$, $S= E_0$, Donde **P**: • Si: $D_X(E_1)= E_2$, $E_2 \neq \lambda$, $E_2 \neq \Phi$ Crear la regla: $E_1 \rightarrow x\,E_2$ • Si: $\lambda \in D_X(E_1)= E_2$ Crear la regla: $E_1 \rightarrow x$ • Si: $\lambda \in E_0$ Crear la regla: $E_1 \rightarrow \lambda$ • Si: $D_X(E_1)= \Phi$ No crear regla.	Sea la E_{RE}: $E=a^*.b.(a+b)^*.b$ $D_a(E_0) = E_0$ $D_b(E_0) = (a+b)^*.b = E_1$ $D_a(E_1) = E_1$ $D_b(E_1) = D_b((a+b)^*.b) = D_b((a+b)^*.b + f((a+b)^*).D_b(b)$ $= D_b(a+b).(a+b)^*.b + \lambda.\lambda = (a+b).b.\lambda = E_2$ $D_a(E_2) = D_a(E_1) + D_a(\lambda) = E_1 + \Phi = E_1$ $D_b(E_2) = D_b(E_1) + D_b(\lambda) = E_2 + \Phi = E_2$ Luego: $G_{RE} = (\Sigma_T, \Sigma_N, S, P) = (\{a,b\},\{ E_0, E_1, E_2\},S,P)$ Con las Reglas **P** $S \rightarrow E_0$ $E_0 \rightarrow aE_0 \mid bE_1$ $E_1 \rightarrow aE_1 \mid bE_2 \mid b$ Donde: $E_0 \rightarrow aE_0 \mid bE_1$, $E_1 \rightarrow aE_1 \mid bE_1 \mid b$, $\Sigma_N=\{E_0,E_1\}$ $E_2 \rightarrow aE_1 \mid bE_2 \mid b$

- **LEMA DEL BOMBEO en L_{RE}:**

  Describe una propiedad esencial de todo lenguaje regular. Informalmente, dice que cualquier palabra suficientemente larga en un lenguaje regular puede ser bombeada, o sea repetir una sección en la mitad de la palabra un número arbitrario de veces, para producir una nueva palabra que también pertenece al mismo lenguaje. Demuestra que un lenguaje específico no es regular.

  ENUNCIADO: Sea **L** un L_{RE}. Entonces existe un entero: **P ≥ 1** ("longitud de bombeo" y que dependerá exclusivamente de **L**) tal que cualquier cadena **w** perteneciente a **L**, de longitud mayor o igual que **p**, puede ser escrita como **w = xyz** (Ej. dividiendo w en tres subcadenas), de forma que se satisfacen las siguientes condiciones:

  - **.y** : Subcadena que puede ser bombeada (borrada o repetida un número ***i*** de veces como se indica en $\forall \mid i \geq 0, xy^iz \in L$, y la cadena resultante seguirá perteneciendo a ***L***)
  - **|y| ≥ 1** Cadena *y* que se bombea debe tener como mínimo longitud uno.
  - **|xy| ≤ p** luego **y** esta dentro de los **p** primeros caracteres. No hay restricciones acerca de **x** o **z**.

  El lema del bombeo puede probar que un lenguaje particular no es L_{RE}: Una demostración por reducción al absurdo (de que un lenguaje no es L_{RE}) puede consistir en encontrar una palabra (de una longitud requerida) en el lenguaje, que carece de la propiedad descrita en el lema del bombeo.

  Ejemplo: Demostrar que, sobre el alfabeto **σ = *a,b***, el lenguaje **$L=\{a^nb^n\}$: n ≥ 0** no es regular. Supongamos que L es regular. Sean ***w, x, y, z, p,* e *i.***

  Sea **w ∈ L** dado por $w = a^pb^p$.

  Por el lema del bombeo, debe haber una descomposición w = xyz con **|xy| ≤ p** e **|y| ≥ 1** tales que: $xy^iz \in L \ \forall i \mid i \geq 0$. Si hacemos que | xy | = p y que | z | = p, entonces xy será la primera mitad de w, (las p aes). Como **|y| ≥ 1**, y tiene una cantidad no nula de aes, y por tanto cualquier cadena bombeada como p. ej. xy2z tendrá un número mayor de aes que de bes y no pertenecerá al lenguaje L, lo que contradice el tercer punto del lema. La suposición de que L es regular debe ser incorrecta. Por tanto L no es regular.

- **LEMA de BOMBEO en la E_{RE}:**

  Si **L** es un L_{RE} aceptado un A_{FD} con **n** estados:
  Si **L** contiene una cadena de largo ≥ **n**, **L** es necesariamente infinito.

  Para toda $w \in L$, $|w| \geq n$,
  existen $x,y,z \in \Sigma^*$,
  tales que **w=xyz**, **|y|>0**, **|xy|≤ n**
  y para todo **i≥0**, $xy^iz \in L$.

  Se dice que la subsecuencia **y** se puede **borrar o repetir(Bombear) i** veces
  y **w** sigue **∈ L**.

  Ejemplo:
  $L=\{ a^i \mid i= k^2 \ , k>0 \}$ es un L_{RE}?
  Si **n**: cantidad de estados de A_{FD}, que acepta al L_{RE}, **L**:
  Notar que a^i con $i= n^2 \in L$, por el lema de bombeo existe la secuencia **xyz**, con $|xyz|= n^2$, tal que $1 \leq |y| \leq n$ y las secuencias $xy^iz \in L$ para todo **i≥0**, entonces se tiene que:

  $$n^2 = |xyz| < |xy^2z| \leq n^2+n < (n+1)^2,$$

  La longitud de xy^2z está estrictamente, entre dos cuadrados perfectos, por tanto no es un cuadrado perfecto.
  Así, la secuencia no∈**L**, luego **L** no es regular.

Bibliográfica

- **Teoría de autómatas y lenguajes formales**.
  Autómatas y complejidad. Kelly Dean Editorial Prentice Hall

- **Introducción a la teoría de autómatas, lenguajes y computación**
  John E. Hopcroft; Jeffrey D. Ullman Editorial Cecsa

- **Teoría de la computación**
  J. Gleuu Brokshear Editorial Addison Wesley Iberoamericana

CONTENIDOS. 2013

- **Unidad 1)** Lingüística Matemática: Alfabetos, palabras y lenguajes. Operaciones con cadenas y con lenguajes. Niveles de un lenguaje. Gramáticas para estructuras de frases. Diagramas de Sintaxis y Formato BNF. Formalismos para el análisis semántico: Sistemas Canónicos de Donovan y Esquemas de Traducción. Proceso de Compilación.
- **Unidad 2)** Gramáticas y Modelos: Jerarquía de Chomsky y formatos estándares para tipos 0 y 1. Los aceptores de lenguajes formales: Máquina de Turing (MT) y Autómata Linealmente Limitado (ALL). Lenguajes Recursivos y recursívamente enumerables. Resolubilidad y complejidad computacional.
- **Unidad 3)** Lenguajes Regulares (LR) y Autómatas Finitos (AF): Gramáticas Regulares (GR), Expresiones Regulares (ER). Máquinas Secuenciales: Modelos de Mealy y de Moore. Autómata Finito Determinista (AFD). Minimización. Propiedades de los LR. Autómatas Finitos No Deterministas (AFND). Operaciones con AF. Conversión AFND/AFD. Obtención del AF a partir de una ER. Analizador Lexico (Scanner).
- **Unidad 4)** Lenguajes Independientes del Contexto (LIC) y Autómatas de Pila: Gramáticas Independientes del Contexto (GIC). Árboles de derivación. Ambigüedad. Árbol Total del Lenguaje. Simplificaciones de una GIC. Formas Normales. LIC y Autómatas de Pila (AP). Criterios de aceptación por estado final y por pila vacía. Diseño de un AP a partir de una GIC. Obtención de la GIC a partir de un AP. Propiedades de los LIC. Analizador Sintáctico (Parser) Descendente (LL) y Ascendente (LR).