6. Gramáticas Libres de Contexto.

Las **gramáticas libres de contexto** generan lenguajes **libres de contexto (o independientes de contexto),** una clase más amplia de lenguajes.

Definición.

Es una 4-tupla $G =< N, \Sigma, S, P >$, donde:
N es un conjunto finito de no terminales (variables).
Σ es un conjunto finito de terminales (alfabeto).
S es un símbolo distinguido en N, llamado símbolo inicial.
P es un conjunto de producciones de la forma $\boldsymbol{A \rightarrow \beta}$, donde $\boldsymbol{A \in N \wedge \beta \in (N \cup \Sigma)^*}$

Derivaciones.

Derivación (en general).

Derivación $\underset{G}{\Rightarrow}$:
Si $\alpha A \gamma$ es una cadena en $(N \cup \Sigma)^*$ y $A \rightarrow \delta$ es una producción, entonces $\alpha A \gamma \Rightarrow^G \alpha \delta \gamma$.

Derivaciones más a la Izquierda.

En cada paso de derivación se reemplaza la variable situada ***más a la izquierda***.

Derivaciones más a la Derecha.

En cada paso de derivación se reemplaza la variable situada ***más a la derecha***.

Ejemplo:
$G =< \{E, I\}, \{0,1, +, *, (,)\}, E, P >$, donde:

$$P = \{E \rightarrow I | E + E \mid E * E | (E)$$
$$I \rightarrow I0 | I1 | 0 | 1\}$$

Es una gramática que genera expresiones que podrían ser de un lenguaje de programación, como por ejemplo: $(\mathbf{11} + \mathbf{0}) * \mathbf{1}$

- Una derivación ***más a la izquierda*** podría ser:

$E \Rightarrow E * E \Rightarrow (E) * E \Rightarrow (E + E) * E \Rightarrow (I + E) * E \Rightarrow (I1 + E) * E \Rightarrow (11 + E) * E$
$\Rightarrow (11 + I) * E \Rightarrow (11 + 0) * E \Rightarrow (11 + 0) * I \Rightarrow (\mathbf{11} + \mathbf{0}) * \mathbf{1}$

- Una derivación ***más a la derecha*** podría ser:

$E \Rightarrow E * E \Rightarrow E * I \Rightarrow E * 1 \Rightarrow (E) * 1 \Rightarrow (E + E) * 1 \Rightarrow (E + I) * 1 \Rightarrow (E + 0) * 1$
$\Rightarrow (I + 0) * 1 \Rightarrow (I1 + 0) * 1 \Rightarrow (\mathbf{11} + \mathbf{0}) * \mathbf{1}$

Árboles de derivación.

Las derivaciones se pueden representar en forma de árbol.
Dicho árbol, nos muestra cómo los símbolos de una cadena terminal se agrupan en subcadenas, cada una de las cuales pertenece al lenguaje de alguna variable de la gramática.

El **árbol de derivación** se utiliza en los compiladores para representar la estructura sintáctica del programa fuente.
Esta representación facilita la traducción a código ejecutable, permitiendo que el proceso de traducción sea desencadenado por funciones recursivas.

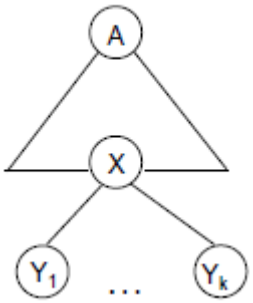
Definición.

Sea $G =< N, \Sigma, S, P >$
Árbol de derivación $\tau(A)$es un grafo tal que:

1. Si $A \in N$, $\tau(A)$ es un árbol de derivación y su representación es un nodo.

2. Si $\tau(A)$ es un árbol de derivación y X es una hoja del árbol tal que $X \in N \wedge X \rightarrow Y_1 Y_2 \ldots Y_k$, entonces también es un árbol de derivación el que resulta de conectar X con aristas a nuevos nodos $Y_1, Y_2, \ldots, Y_k$.

Gráficamente:



Resultado de un árbol de derivación

Es la cadena que se obtiene concatenando las hojas desde la izquierda.
Los árboles de derivación en los que:

1. Su resultado es una cadena terminal (las hojas están etiquetadas con terminales con con λ)
2. La raíz está etiquetada con el símbolo inicial.

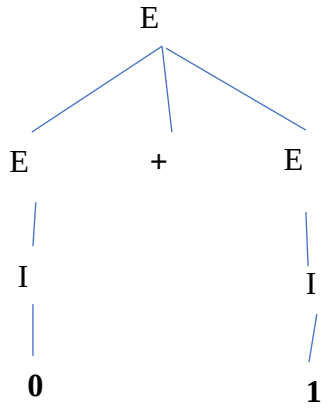
Tienen como resultado cadenas del lenguaje de la gramática subyacente.

Ejemplo:

$G =< \{E, I\}, \{0,1, +, *, (,)\}, E, P >$, donde:

$$P = \{E \rightarrow I | E + E \mid E * E | (E)$$
$$I \rightarrow I0 | I1 | 0 | 1\}$$

El siguiente árbol de derivación tiene como resultado $0 + 1 \in L(G)$



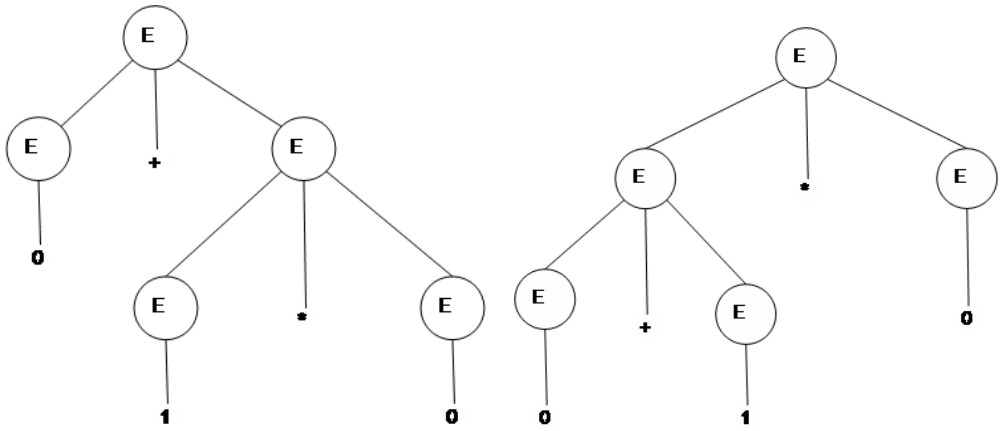
Gramáticas ambiguas.

Una GLC $G =< N, \Sigma, P, S >$ **es ambigua** si existe alguna cadena que es resultado de dos árboles de derivación distintos.
Es decir:

G ambigua $\Leftrightarrow \exists \alpha \in L(G) | \exists \tau(S), \tau'(S) \wedge \tau(S) \neq \tau'(S)$ y α es base de ambos.

Ejemplo:

$G =< \{E\}, \{0,1, +, *\}, E, P >$, donde: $P = \{E \rightarrow E + E | E * E | 0 | 1\}$
Presenta dos árboles de derivación distintos para la cadena **0+1*0**



Eliminación de la ambigüedad.

No existe un algoritmo que nos diga si una GLC es ambigua.

Sin embargo, en algunas construcciones como en la gramática del ejemplo anterior, se puede encontrar gramáticas equivalentes no ambiguas.

Ejemplo:
Existe una gramática no ambigua equivalente:
$G' =< \{E,T,I\},\{0,1,+,*\},E,P >$, donde:
$P = \{E \to E + T|T$
$T \to T * I|I$
$I \to 0|1\}$
Esta gramática consigue eliminar ambigüedades forzando ***precedencia*** y ***asociatividad*** introduciendo variables nuevas, cada una de las cuales representa aquellas expresiones que comparten el mismo nivel de "fuerza de acoplamiento".

Derivaciones más a la izquierda y la ambigüedad.

Una GLC es ambigua si y sólo si para una cadena ω existen dos derivaciones más a la izquierda distintas desde S.

Ambigüedad inherente.

Un lenguaje es ***inherentemente ambiguo*** (o intrínsecamente ambiguo) si todas las gramáticas que lo pueden generar son ambiguas.

- El lenguaje de las expresiones no es inherentemente ambiguo, porque encontramos una gramática **NO ambigua** que lo genera.
  $G' =< \{E,T,I\},\{0,1,+,*\},E,P >$, donde:
  $P = \{E \to E + T|T$
  $T \to T * I|I$
  $I \to 0|1\}$
- El lenguaje $L = \{a^n b^n c^m d^m | n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n | n \geq 1, m \geq 1\}$ es inherentemente ambiguo.
  Una gramática podría ser: $G =< \{S,A,B,C,D\},\{a,b,c,d\},S,P >$, donde:
  $P = \{S \to AB|C$
  $A \to aAb|ab$
  $B \to cBd|cd$
  $C \to aCd|aDd$
  $D \to bDc \mid bc\}$

Formas Normales para las Gramáticas Libres de Contexto.

Forma Normal de Chomsky:

Todo Lenguaje Libre de Contexto (LLC) sin λ es generado por una GLC en la que todas las producciones son de la forma $A \to BC$ o $A \to a$, con $A \in N, B \in N, C \in N, a \in \Sigma$

Para llegar a la ***Forma Normal de Chomsky***, las simplificaciones preliminares son:

1. Eliminación de símbolos inútiles.
2. Eliminación de producciones λ.
3. Eliminación de producciones unitarias.

Eliminación de símbolos inútiles.

Un símbolo $\boldsymbol{X} \in (\boldsymbol{N} \cup \boldsymbol{\Sigma})^*$ ***es útil***, para una gramática $G =< N, \Sigma, P, S >$ si existe alguna derivación de la forma $S \Rightarrow^* \alpha X \beta \Rightarrow^* \omega$, donde $\omega \in \Sigma^*$
Por lo tanto, al eliminar los símbolos inútiles no cambiará el lenguaje generado.

Obtención del conjunto de símbolos generadores o productivos:

Sea $G =< N, \Sigma, P, S >$
BASE: Todo $a \in \Sigma$ es productivo.
PASO INDUCTIVO: Si existe una producción $A \to \alpha$ tal que todo símbolo de α es productivo, entonces A es productivo.

Obtención del conjunto de símbolos alcanzables:

Sea $G =< N, \Sigma, P, S >$
BASE: S es alcanzable.
PASO INDUCTIVO: Si A es alcanzable, entonces para todas las producciones con A en la parte izquierda, todos los símbolos de las partes derechas de dichas producciones son alcanzables.

Eliminación de producciones λ.

Obtención del conjunto de símbolos anulables:

Sea $G =< N, \Sigma, P, S >$
BASE: Si $A \to \lambda$.es una producción de G, entonces A es anulable.
PASO INDUCTIVO: Si existe una producción $B \to C_1 C_2 \dots C_k$ donde cada $C_i$es anulable, entonces B es anulable.

Algoritmo para eliminar los símbolos anulables:

ENTRADA:

$$G =< N, \Sigma, P, S >$$
$$Anulables = \{A \in N | A \Rightarrow * \lambda\}$$

SALIDA:

$G' =< N, \Sigma, P, S >$ sin símbolos anulables.

PASOS:

Mientras $Anulables \neq \emptyset$

- Tomar un $X \in Anulables$. Por cada producción donde X aparece en la parte derecha, agregar una producción sin X en la parte derecha.
- Eliminar toda producción en donde X aparece en la parte izquierda y λ en la derecha.
- Eliminar X de $Anulables$.

Eliminación de producciones unitarias.

Obtención del conjunto de pares unitarios:

Sea $G =< N, \Sigma, P, S >$
BASE: (A, A) es un par unitario para cualquier variable $A \in N$.
PASO INDUCTIVO: Si (A, B) es un par unitario, y $B \to C$ es una producción, donde C es una variable, entonces (A, C) es un par unitario.

Algoritmo para eliminar las producciones unitarias:

ENTRADA:

$$G =< N, \Sigma, P, S >$$
$$Pares\ Unitarios$$

SALIDA:

$G' =< N, \Sigma, P, S >$ sin producciones unitarias.

PASOS:

Por cada par $(A, B) \in Pares\ Unitarios$, añadir a P todas las producciones $A \to \alpha$ donde α es la parte derecha de B (es decir, $B \to \alpha$ es una producción no unitaria)

Forma Normal de Chomsky

Para terminar de lograr la Forma Normal de Chomsky quedan dos tareas:

a) Conseguir que todos los cuerpos de longitud 2 o superior estén formados por variables.
b) Descomponer los cuerpos de longitud 3 o superior en una cascada de producciones, teniendo cada una de ellas un cuerpo formado sólo por dos variables.

Para eso los pasos son:

a) Para todo símbolo a que aparezca en un cuerpo de longitud 2 o superior, crear una nueva variable (por ejemplo X) y una producción $X \to a$.
Reemplazar a por X en toda otra producción de longitud 2 o superior.
b) Descomponer cada producción $A \to B_1 B_2 \dots B_k$ para $k \geq 3$ en un grupo de producciones con dos variables en cada parte derecha. Introducimos $k-2$ nuevas variables, $C_1 C_2 \dots . C_{k-2}$ y la producción original se reemplaza por las $k-1$ producciones:

$$A \to B_1 C_1$$
$$C_1 \to B_2 C_2$$
$$C_2 \to B_3 C_3 \dots$$
$$C_{k-2} \to B_{k-1} B_k$$

Cualquier árbol de derivación para una gramática en FNC resulta ser un árbol binario.

Equivalencia entre Autómatas de Pila y Gramáticas Libres de Contexto.

De las Gramáticas a los Autómatas de Pila.

Dada una GLC G, construimos un Autómata de Pila que simule las derivaciones más a la izquierda de G.
Eso se logra con la siguiente
Construcción.

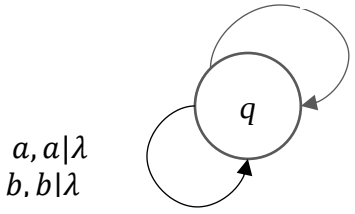
Sea $G =< N, \Sigma, S, P_g >$ una Gramática Libre de Contexto.
Construimos $P =< \{q\}, \Sigma, N \cup \Sigma, \delta, q, S >$
donde la función de transición δ se define de la forma siguiente:

1. $\forall A \to \beta \in \mathrm{P_g}: \delta(q, \lambda, A)$ contiene a (q, β)
2. $\forall a \in \Sigma: \ \delta(q, a, a) = \{(q, \lambda)\}$

Ejemplo:

Sea $G =< \{S, A\}, \{\mathrm{a}, \mathrm{b}\}, \ S, P_g >$
Tal que:
$P_g = \{S \to aAa \mid bSb \mid \lambda$
$A \to aAa \mid b\}$

$$\lambda, S|aAa$$
$$\lambda, S|bSb$$
$$\lambda, S|\lambda$$
$$\lambda, A|aAa$$
$$\lambda, A|b$$



De los Autómatas de Pila a las Gramáticas Libres de Contexto.

Dado un Autómata de Pila, existe una gramática libre de contexto tal que $L(G) = L_{pv}(P)$
Construcción.

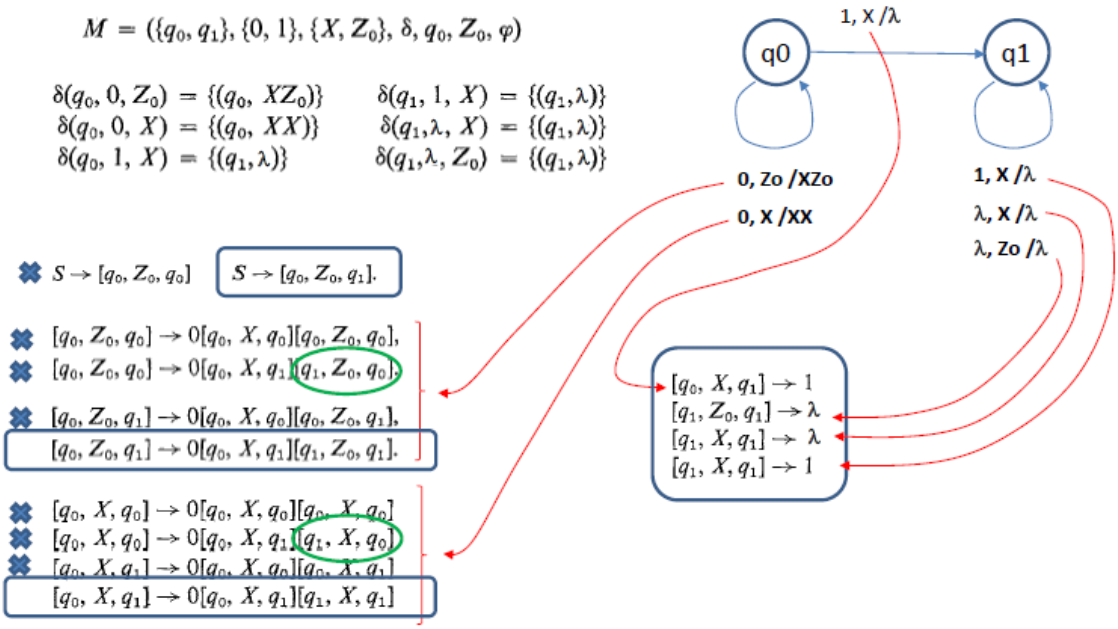
Sea $P =< Q, \Sigma, \Gamma, \delta, q_0, z_0 >$ un autómata de pila que reconoce por pila vacía
Construimos $G =< N, \Sigma, R, S >$ donde:
$N = \{S\} \cup \{[t] | t \in (Q \times \Gamma \times Q)\}$
Es decir, $[t] = [pXq]$ con $p \in Q \wedge q \in Q \wedge X \in \Gamma$

Y las producciones de G son las siguientes:

1. $\forall q \in Q: S \rightarrow [q_0\, z_0 q] \in R$
2. $\forall q \in Q, \forall a \in (\Sigma \cup \{\lambda\}), \forall A, B_1, B_2 \ldots \in \Gamma$:
   Si $(q_1, B_1\, B_2 \ldots B_m) \in \delta(q, a, A)$agregar a R las producciones:
   $[qAp] \rightarrow a[q_1 B_1 q_2][q_2 B_2 q_3] \ldots [q_m B_m q_{m+1}]$ donde $q_{m+1} = p$
   Si $(p, \lambda) \in \delta(q, a, A)$ agregar a R la producción:
   $[qAp] \rightarrow a$

Ejemplo:

Sea $M =< \{q_0, q_1\}, \{0,1\}, \{X, Z_0\}, \delta, q_0, Z_0 >$
Entonces en la nueva gramática $G =< N, \Sigma, R, S >$ los posibles símbolos no terminales son:
$N = \{S,$
$[q_0 Z_0 q_0], [q_1 Z_0 q_0]$
$[q_0 Z_0 q_1], [q_1 Z_0 q_1]$
$[q_0 X q_0], [q_1 X q_0],$
$[q_0 X q_1], [q_1 X q_1]$
$\}$
Luego se analizan las producciones, y se descartan aquellas que sean improductivas e inalcanzables.



Finalmente, se renombran los símbolos no terminales definitivos:
$A = [q_0 Z_0 q_1]$
$B = [q_0 X q_1]$
$C = [q_1 Z_0 q_1]$
$D = [q_1 X q_1]$

$$S \rightarrow A$$
$$A \rightarrow 0BC$$
$$B \rightarrow 0BD$$
$$B \rightarrow 1$$
$$C \rightarrow \lambda$$
$$D \rightarrow \lambda$$
$$D \rightarrow 1$$

Entonces en la nueva gramática queda $G_{final} =< \{S, A, B, C, D\}, \{0,1\}, R, S >$
Con

$$R = \{$$
$$S \rightarrow A$$
$$A \rightarrow 0BC$$

$B \to 0BD \mid 1$
$C \to \lambda$
$D \to 1|\lambda$
$\}$

Luego se puede convertir a alguna Forma Normal o hacerle alguna simplificación.

Gramáticas Ambiguas y Autómatas de Pila Deterministas.

- Todos los lenguajes que son aceptados por un APD tienen gramáticas **no ambiguas**.
- Hay lenguajes generados por gramáticas no ambiguas que no pueden ser reconocidos por ningún APD.

Ejemplo:

$$G =< \{S\}, \{0,1\}, R, S >$$

Con

$$R = \{S \to 0S0 \mid 1S1|\lambda\}$$

Es una gramática no ambigua, y $L(G) = \{\omega\omega^r|\omega \in \{0,1\}^*\}$ pero ya vimos que no era posible encontrar un APD que reconociera el mismo lenguaje.

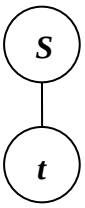
Se demuestra para un Autómata de Pila Determinista que reconoce por Pila vacía, y luego se puede extender a APD que reconozcan por estado final.

Lema de Bombeo para Lenguajes Independientes de Contexto.

Tamaño de los árboles de derivación.

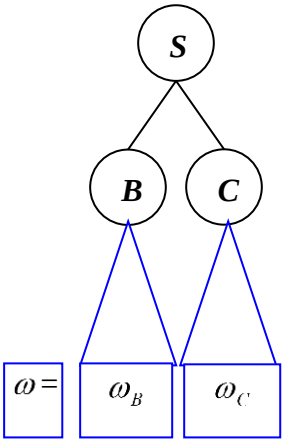
Teorema. Dada una gramática $G = \langle \Sigma, V, S, P \rangle$ en FNC y un árbol de derivación $\tau(S)$ en dicha gramática cuyo resultado es la palabra ω, si la longitud del camino más largo (altura) es n, entonces $|\omega| \le 2^{n-1}$

Demostración. Por inducción en n.



Base: (Probar que para $n = 1$ es cierto.)

Un árbol con una longitud de camino máxima de 1 consta únicamente de una raíz (S) y de una hoja etiquetada con un símbolo terminal. La cadena ω es por lo tanto ese terminal. Entonces $|\omega| = 1 = 2^0 = 2^{1-1} \Rightarrow 2^{n-1}$ con $n = 1$ se cumple.



Paso inductivo: (Probar que si para $n < h$ es cierto, entonces para $n = h$ también lo es.)

El camino más largo en el árbol $\tau(S)$ tiene longitud $n = h$, mayor que 1 por lo que no se puede empezar usando una producción del tipo ***A → t***, y siendo que la gramática está en FNC entonces debe comenzarse con una producción de tipo ***S → BC***. Ningún camino en los subárboles $\tau(B)$ y $\tau(C)$ tiene longitud $n = h$ sino, a lo sumo $n = h - 1$. Por lo que entonces en esos casos se cumple la hipótesis inductiva.

Así, en $\tau(B)$ se obtiene una palabra ω_B, y por hipótesis inductiva $|\omega_B| \le 2^{h-2}$, y en $\tau(C)$ se obtiene una palabra ω_C, y por hipótesis inductiva $|\omega_C| \le 2^{h-2}$.

Por lo tanto, $\tau(S)$ permite obtener una palabra ω, tal que
$|\omega| = |\omega_B \omega_C| \leq 2^{h-2} + 2^{h-2} = 2^{h-2}.2 = 2^{h-1} = 2^{n-1}$ con $n = h$

Lema de bombeo.

Enunciado:

Sea L un lenguaje libre de contexto. Entonces existe una constante p tal que si α es una cadena de longitud mayor o igual que p, podemos escribir $\alpha = rxyzs$ con las siguientes condiciones:

1. $|xyz| \leq p$. Es decir, la parte central no es demasiado larga.
2. $xz \neq \lambda$. Es decir, al menos una de las cadenas a bombear no es vacía.
3. Para todo $i \geq 0$, $rx^i yz^i s$ está en L.

Simbólicamente:

$$\forall L, \text{LIC} : \exists \text{p} > 0 / \left(\forall \alpha \in \text{L} : \left(|\alpha| \geq p \Rightarrow \exists r, x, y, z, s / \alpha = rxyzs \wedge |xyz| \leq p \wedge \left(x \neq \lambda \vee z \neq \lambda \right) \wedge \left(\forall i \geq 0 : rx^i yz^i s \in L \right) \right) \right)$$

Demostración:

En primer lugar se encuentra una gramática $G = \langle \Sigma, V, S, P \rangle$ en forma normal de Chomsky, tal que genere $L(G) = L - \{\lambda\}$.

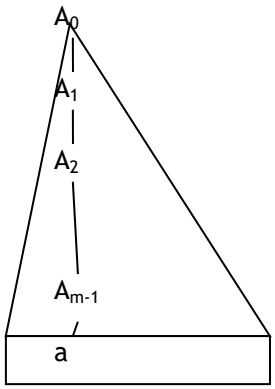
Observación 1: La FNC no permite encontrar una gramática si L es $\varnothing$. Pero si $L = \varnothing$ no se viola el enunciado del lema de bombeo, ya que el lema exige la existencia de una cadena α en L.

Observación 2: La gramática en FNC G generará $L(G) = L - \{\lambda\}$, pero eso no tiene importancia porque el lema exige que la palabra α que se elija sea $|\alpha| \geq p > 0$, es decir que $\alpha \neq \lambda$

Suponiendo que, en $G, \#V = m$ (hay m variables), se elige $p = 2^m$.

La palabra α debe tener $|\alpha| \geq p > 0$.

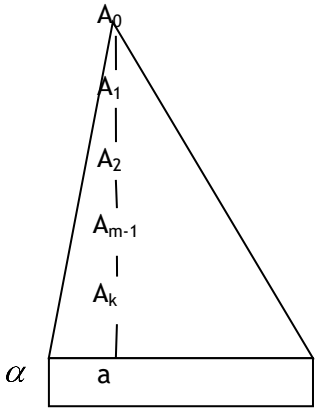
Si se tiene en cuenta el teorema previo , todo árbol de derivación cuyo camino más largo tiene una longitud m o menor debe generar una palabra de longitud 2^{m-1}.



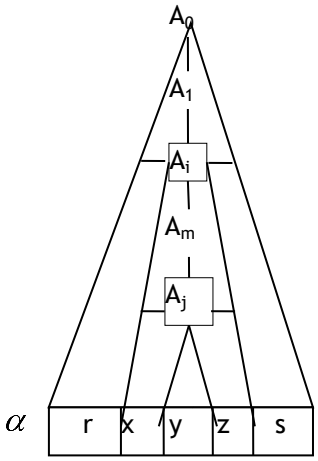
Entonces no podría generar la palabra α, ya que $2^{m-1} = \frac{2^m}{2} = \frac{p}{2}$.

El árbol de derivación que pueda generar α tiene que tener longitud de al menos $m+1$.

Por lo tanto, deberá usar al menos $m+1$ variables. Como la gramática tiene m variables, alguna variable se usa dos veces.



Suponiendo que $A_i = A_j$, para $k - m \le i < j \le k$.



Entonces se puede dividir el árbol como se muestra en la figura:

El subárbol de raíz A_j genera la cadena y.

Las cadenas x y z son las que están a la izquierda y la derecha, respectivamente, de y en la cadena generada por el subárbol más grande, cuya raíz es A_i.

Observación: Como no existen producciones unitarias, x y z no pueden ser ambas iguales a λ, aunque una de ellas sí podría serlo.

Finalmente, r y s son los trozos de α situados a la izquierda y a la derecha, respectivamente, del subárbol con la raíz A_i.

Por lo tanto:

$$I) A_0 \overset{*}{\Rightarrow} rA_i s \quad II) A_i \overset{*}{\Rightarrow} xA_j z \quad III) A_j \overset{*}{\Rightarrow} y$$

Como $A_i = A_j = A$:

$$I) A_0 \overset{*}{\Rightarrow} rAs \quad II) A \overset{*}{\Rightarrow} xAz \quad III) A \overset{*}{\Rightarrow} y$$

Por inducción, se prueba que $\forall i \ge 0 : rx^i yz^i s \in L$:

Base: (Probar que para $i = 0$ es cierto.)

Por (I) y (III) $A_0 \overset{*}{\Rightarrow} rAs \overset{*}{\Rightarrow} rys = rx^0 yz^0 s$. Por lo tanto, $rx^0 yz^0 s \in L$

Paso inductivo: (Para $i = h$ es cierto, entonces es cierto para $i = h+1$)

Como por hipótesis inductiva $rx^h yz^h s \in L$, para lograrlo se tuvo que usar (I), (II) h veces y finalmente (III): $A_0 \overset{*}{\Rightarrow} rAs \overset{*}{\Rightarrow} rx^h Az^h s \overset{*}{\Rightarrow} rx^h yz^h s$.

Pero después de usar (I), podría haberse usado (II) h+1 veces y finalmente (III):
$A_0 \overset{*}{\Rightarrow} rAs \overset{*}{\Rightarrow} rx^{h+1}Az^{h+1}s \overset{*}{\Rightarrow} rx^{h+1}yz^{h+1}s$.

Observación: el lema dice que $|xyz| \leq p$. Pero si A_i se elige cerca de la base del árbol, es decir con $k-i \leq m$, luego el camino más largo del subárbol con la raíz A_i no es mayor que $m+1$, y por lo tanto $|xyz| \leq 2^m = p$

Propiedades de los Lenguajes Libres de Contexto.

Unión de LLC.
Si L_1 es LLC y L_2 es LLC, entonces $L_1 \cup L_2$ es LLC.

Concatenación de LLC.
Si L_1 es LLC y L_2 es LLC, entonces L_1L_2 es LLC.

Clausura + y Clausura * de LLC.
Si L es LLC, entonces L^+es LLC y L^* es LLC.

Reverso de LLC.
Si L es LLC, entonces L^Res LLC.

Intersección de LLC.
La intersección de LLC no es cerrada.
Ejemplo:

$$L_1 = \{0^n 1^n 2^i, n \geq 1, i \geq 1\}$$
$$L_2 = \{0^i 1^n 2^n, n \geq 1, i \geq 1\}$$

Son ambos libres de contexto.
Sin embargo, la intersección no lo es:

$$L = \{0^n 1^n 2^n, n \geq 1\}$$

Intersección de LLC con Lenguaje Regular.
La intersección de un lenguaje regular con uno LLC es un LLC.
Si L_1 es LR y L_2 es LLC, entonces $L_1 \cap L_2$ es LLC.