

Tema 4: Gramáticas Independientes del Contexto

Serafín Moral

Universidad de Granada

Noviembre, 2020

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

Gramática Independiente del Contexto

Una gramática $G = (V, T, P, S)$ se dice que es independiente del contexto o de tipo 2 si y solo si todas las producciones tienen la forma:

$$A \rightarrow \alpha$$

donde $A \in V$ y $\alpha \in (V \cup T)^*$.

Los lenguajes generados por estas gramáticas se llaman también independiente del contexto o de tipo 2.

$$S \rightarrow a|b|c$$

$$S \rightarrow (S + S)$$

$$S \rightarrow (S * S)$$

Esta es una gramática independiente del contexto: En la parte izquierda de las producciones solo aparece una variable.

Al language generado por esta gramática pertenece la palabra $((a + b) * c)$. Solo hay que aplicar la siguiente cadena de producciones

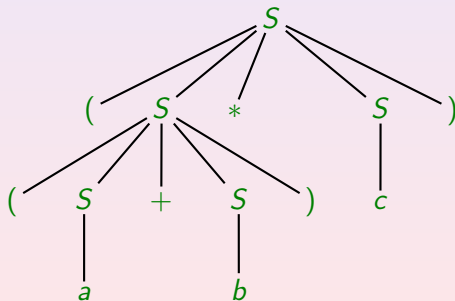
$$\begin{aligned} S &\Rightarrow (S * S) \Rightarrow ((S + S) * S) \Rightarrow ((a + S) * S) \Rightarrow \\ &((a + b) * S) \Rightarrow ((a + b) * c) \end{aligned}$$

Arbol de Derivación

Derivación:

$$S \Rightarrow (S * S) \Rightarrow ((S + S) * S) \Rightarrow ((a + S) * S) \Rightarrow ((a + b) * S) \Rightarrow ((a + b) * c)$$

Árbol:



Cada nodo del árbol va a contener un símbolo.

En el nodo raíz se pone el símbolo inicial S .

Se efectúa una ramificación del árbol por cada producción que se aplique: Si a la variable de un nodo, A , se le aplica una determinada regla $A \rightarrow \alpha$, entonces para cada símbolo que aparezca en α se añade un hijo con el símbolo correspondiente, situados en el orden de izquierda a derecha.

Este proceso se repite para todo paso de la derivación.

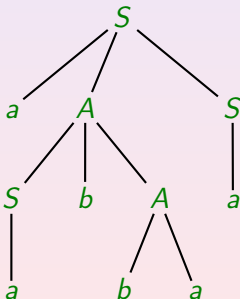
Si la parte derecha es una cadena vacía, entonces se añade un solo hijo, etiquetado con ϵ .

En cada momento, leyendo los nodos de izquierda a derecha se lee la palabra generada.

Ejemplo

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

La palabra *aabbaa* tiene asociado el árbol:

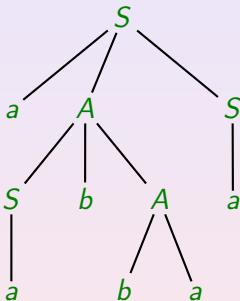


Un árbol de derivación puede proceder de dos cadenas de derivación distintas.

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

Ejemplo

Dado el árbol de la palabra *aabbbaa*



Derivación por la izquierda:

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

Derivación por la derecha:

$S \Rightarrow aAS \Rightarrow aAa \Rightarrow aSbAa \Rightarrow aSbbbaa \Rightarrow aabbbaa$

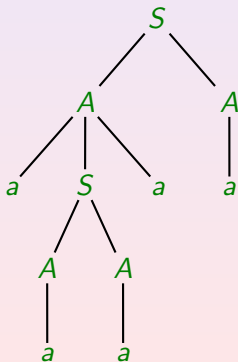
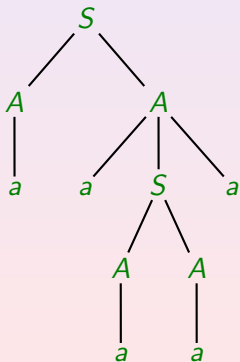


Gramática Ambigua

Definición

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

Ejemplo: La gramática: $S \rightarrow AA$, $A \rightarrow aSa$, $A \rightarrow a$ es ambigua, ya que la palabra a^5 tiene los dos árboles siguientes:



Lenguaje inherentemente ambiguo

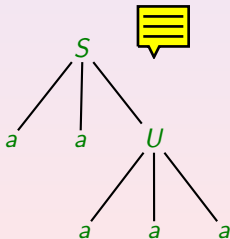
Definición

Un lenguaje de tipo 2 es **inherentemente ambiguo** si toda gramática que lo genera es ambigua.

Ejemplo: El lenguaje generado por la gramática anterior c , no es inherentemente ambiguo.

Este lenguaje es $\{a^{2+3i} : i \geq 0\}$ y puede ser generado por la gramática:
 $S \rightarrow aa$, $S \rightarrow aaU$, $U \rightarrow aaaU$, $U \rightarrow aaa$, que no es ambigua.

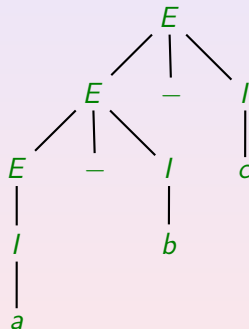
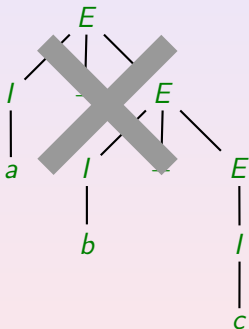
Único árbol para a^5 :



Ejemplo

$E \rightarrow I$, $E \rightarrow I - E$, $E \rightarrow E - I$, $I \rightarrow a|b|c|d$

es ambigua, ya que la palabra $a - b - c$ admite dos árboles de derivación distintos:



Eliminando la producción $E \rightarrow I - E$ la gramática deja de ser ambigua.

Lenguaje inherentemente ambiguo

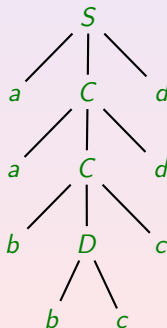
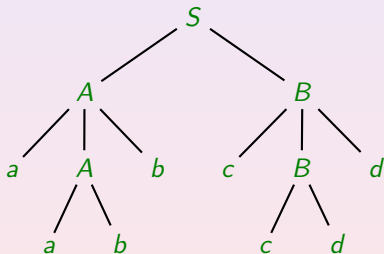
Lenguaje:

$$L = \{a^n b^n c^m d^m : n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n : n \geq 1, m \geq 1\}$$

Gramática:

$$\begin{aligned} S &\rightarrow AB, & A &\rightarrow ab, & A &\rightarrow aAb, & B &\rightarrow cd, & B &\rightarrow cBd, \\ S &\rightarrow aCd, & C &\rightarrow aCd, & C &\rightarrow bDc, & C &\rightarrow bc, & D &\rightarrow bDc, & D &\rightarrow bc \end{aligned}$$

La palabra **aabbccdd** tiene dos árboles de derivación distintos:



Lenguaje inherentemente ambiguo

Lenguaje:

$$L = \{a^n b^n c^m d^m : n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n : n \geq 1, m \geq 1\}$$

Lo visto anteriormente no demuestra que el lenguaje es ambiguo.

Sólo se prueba que una gramática que lo genera es ambigua.

Habría que probar que toda gramática que genera este lenguaje es ambigua.

No lo vamos a probar aquí, pero es cierto que la ambigüedad no se puede evitar en cualquier gramática que genere este lenguaje.

Símbolos y Producciones Inútiles

Definición

Un símbolo $X \in (V \cup T)$ se dice **útil** si y solo si existe una cadena de derivaciones en G tal que

$$S \xRightarrow{*} \alpha X \beta \xRightarrow{*} w \in T^*$$



Definición

Una producción se dice **útil** si y solo si todos sus símbolos son útiles. Esto es equivalente a que pueda usarse en la derivación de alguna palabra del lenguaje asociado a la gramática.

Propiedad

Eliminando todos los símbolos y producciones inútiles (no útiles) el lenguaje generado por la gramática no cambia.

El algoritmo para eliminar los símbolos y producciones inútiles consta de dos pasos:

- 1 Eliminar las variables desde las que no se puede llegar a una palabra de T^* y las producciones en las que aparezcan.
- 2 Eliminar aquellos símbolos que no sean alcanzables desde el símbolo inicial, S , y las producciones en las que estos aparezcan.

Orden de los algoritmos

Es importante aplicar los algoritmos anteriores en el orden especificado.

$$S \rightarrow AB, \quad S \rightarrow a, \quad A \rightarrow a$$

En el primer paso se elimina B y la producción $S \rightarrow AB$.

Entonces en el segundo se elimina la variable A y la producción $A \rightarrow a$.

Si aplicamos primero el segundo algoritmo, entonces no se elimina nada.

Al aplicar despues el primero de los algoritmos se elimina B y la producción $S \rightarrow AB$. En definitiva, nos queda la gramática

$$S \rightarrow a, \quad A \rightarrow a$$

donde todavía nos queda la variable inútil A .



Es posible que una variable X no haya sido detectada como inútil en la primera parte aunque haya usado en la derivación de una palabra formada sólo por símbolos terminales una variable Y que no es accesible desde la variable inicial S :

$$X \xRightarrow{*} \alpha Y \beta \xRightarrow{*} u \in T^*$$

En ese caso, como Y no es accesible desde S , tenemos que X tampoco lo será (ya que Y es accesible desde X) y X será eliminada también por la segunda parte del algoritmo.

Primera Parte

Se diseña un algoritmo calculando V_t , conjunto de variables que se pueden substituir por símbolos terminales.

- **Condición Básica:** Si $A \rightarrow u$, $u \in T^*$, entonces $A \in V_t$
- **Condición Recursiva:** Si $A \rightarrow \beta_1 \dots \beta_k$ y cada β_j está en $T \cup V_t$, entonces $A \in V_t$.

1. $V_t = \emptyset$
2. Para cada producción de la forma $A \rightarrow w$, A se introduce en V_t .
3. Mientras V_t cambie
 4. Para cada producción $B \rightarrow \alpha$
 5. Si todas las variables de α pertenecen a V_t ,
 B se introduce en V_t
6. Eliminar las variables que esten en V y no en V_t
7. Eliminar todas las producciones donde aparezca una variable de las eliminadas en el paso anterior

Ejemplo

$S \rightarrow gAe, \quad S \rightarrow aYB, \quad S \rightarrow cY, \quad A \rightarrow bBY,$
 $A \rightarrow ooC, \quad B \rightarrow dd, \quad B \rightarrow D, \quad C \rightarrow jVB,$
 $C \rightarrow gi, \quad D \rightarrow n, \quad U \rightarrow kW, \quad V \rightarrow baXXX,$
 $V \rightarrow oV, \quad W \rightarrow c, \quad X \rightarrow fV, \quad Y \rightarrow Yhm$

$V_t = \{B, C, D, W, A, U, S\}, \quad V \setminus V_t = \{V, X, Y\}$

$S \rightarrow gAe,$
 $A \rightarrow ooC, \quad B \rightarrow dd, \quad B \rightarrow D,$
 $C \rightarrow gi, \quad D \rightarrow n, \quad U \rightarrow kW,$
 $W \rightarrow c,$

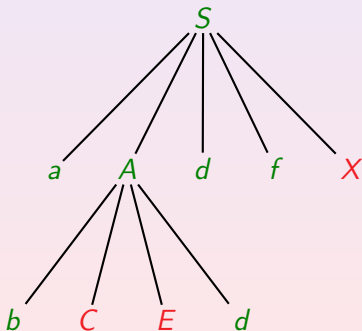


Segunda Parte

Realizamos una búsqueda recursiva a partir del símbolo inicial S de todos los símbolos que se pueden alcanzar a partir de él.

$$S \rightarrow aAd, \quad S \rightarrow fX, \quad A \rightarrow b, \quad A \rightarrow CE d, \dots$$

V_S variables obtenidas
 T_S símbolos terminales obtenidos
 J variables por analizar



Segunda Parte

- V_S y J son conjuntos de variables.
- T_S es un conjunto de símbolos terminales

1. $J = \{S\}$, $V_S = \{S\}$, $T_S = \emptyset$
2. Mientras $J \neq \emptyset$
 3. Extraer un elemento de J : A , ($J = J - \{A\}$).
 4. Para cada produccion de la forma $A \rightarrow \alpha$
 5. Para cada variable B en α
 6. Si B no está en V_S añadir B a J y a V_S
 7. Poner todos los simbolos terminales de α en T_S
 8. Eliminar todas las variables que no estén en V_S
y todos los simbolos terminales que no esten en T_S .
 9. Eliminar todas las producciones donde aparezca
un simbolo o variable de los eliminados en el paso anterior.

Ejemplo: Segunda parte

$$\begin{aligned} S &\rightarrow gAe, & A &\rightarrow ooC, \\ B &\rightarrow dd, & B &\rightarrow D, & C &\rightarrow gi, \\ D &\rightarrow n, & U &\rightarrow kW, & W &\rightarrow c \end{aligned}$$

$$\begin{aligned} V_S &= \{S, A, C\} \\ J &= \{SAC\} \\ T_S &= \{g, e, o, i\} \end{aligned}$$

Única derivación posible: $S \Rightarrow gAe \Rightarrow gooCe \Rightarrow googie$
Lenguaje generado: $\{googie\}$.

Si el lenguaje generado por una gramática es vacío, esto se detecta en que la variable **S resulta inútil** en el primer algoritmo. En ese caso se pueden eliminar directamente todas las producciones, pero no el símbolo S .

Ejemplo:

$$S \rightarrow aSb, \quad S \rightarrow bcD, \quad S \rightarrow cSE,$$

$$E \rightarrow aDb, \quad F \rightarrow abc, \quad E \rightarrow abF$$



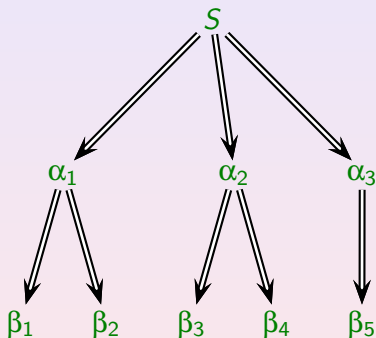
$$V_t = \{F, E\}, \quad L(G) = \emptyset$$

Definen unas características que deben de verificar todas las producciones de una gramática.

- Producciones nulas
- Producciones unitarias
- Forma normal de Chomsky
- Forma normal de Greibach

Motivación: el problema de la pertenencia

Problema: Dada una gramática independiente del contexto G y una palabra u , ¿pertenece u a $L(G)$?



Si la palabra es generada, nos sale en esta búsqueda.

Si la palabra **no es generada**, ¿hasta **qué profundidad** tenemos que generar para convencernos de que no se puede?

Producciones Nulas

Son las que tienen la forma:

$$A \rightarrow \epsilon.$$

Algoritmo

Es posible construir un algoritmo que dada una gramática G , construye una gramática G_n sin producciones nulas y tal que $L(G_n) = L(G) \setminus \{\epsilon\}$.

Si tenemos, $A \rightarrow \epsilon$, $D \rightarrow aABc$



$$D \Rightarrow aABc \Rightarrow aBc$$

Añadimos $D \rightarrow aBc$

Si también $B \rightarrow \epsilon$, entonces habría que añadir

$$D \rightarrow aBc, \quad D \rightarrow aAc, \quad D \rightarrow ac$$

Variables Anulables

Si tenemos $C \rightarrow AB$, $A \rightarrow \epsilon$, $B \rightarrow \epsilon$, habría que añadir, al quitar las producciones nulas

$$C \rightarrow A, \quad C \rightarrow B, \quad C \rightarrow \epsilon$$

Y después habría que eliminar $C \rightarrow \epsilon$

Nosotros vamos a calcular desde el principio todas las variables, E , tales que en algún momento aparece $E \rightarrow \epsilon$. Estas variables se dicen **anulables**. Son variables tales que $E \xRightarrow{*} \epsilon$

Después vamos a **eliminar** todas las producciones nulas y a **añadir** las producciones que compensen esta eliminación.

H es el conjunto de las variables anulables

Cálculo de Variables Anulables:

1. $H = \emptyset$
2. Para cada produccion $A \rightarrow \epsilon$, se hace $H = H \cup \{A\}$
3. Mientras H cambie
4. Para cada produccion $B \rightarrow A_1 A_2 \dots A_n$,
donde $A_i \in H$ para todo $i = 1, \dots, n$, se hace $H = H \cup \{B\}$

Eliminar y Añadir:

1. Se eliminan todas las producciones nulas de la gramática
2. Para cada produccion de la gramática de la forma
 $A \rightarrow \alpha_1 \dots \alpha_n$, donde $\alpha_i \in V \cup T$.
3. Se añaden todas las producciones de la forma
 $A \rightarrow \beta_1 \dots \beta_n$ donde $\beta_i = \alpha_i$ si $\alpha_i \notin H$ y
 $(\beta_i = \alpha_i) \vee (\beta_i = \epsilon)$ si $\alpha_i \in H$
y no todos los β_i puedan ser nulos al mismo tiempo

- Si G generaba inicialmente la palabra nula, entonces la nueva gramática no la genera.
- Se puede saber si se pierde la palabra vacía comprobando si $S \in H$.
- Si tenemos una gramática G , podemos construir una gramática G_v con una sola producción nula y que genera el mismo lenguaje que G más la palabra vacía. Para ello se añade una nueva variable, S_v , que pasa a ser el símbolo inicial de la nueva gramática, G_v . También se añaden dos producciones:

$$S_v \rightarrow S, \quad S_v \rightarrow \epsilon$$

Ejemplo

Eliminar las producciones nulas de la siguiente gramática:

$$\begin{array}{lll} S \rightarrow ABb, & S \rightarrow ABC, & C \rightarrow abC, \\ B \rightarrow bB, & B \not\rightarrow \varepsilon, & A \rightarrow aA, \\ A \not\rightarrow \varepsilon, & C \rightarrow AB & \end{array}$$

Cálculo de $H = \{A, B, C, S\}$.

Al ser S anulable la palabra vacía puede generarse mediante esta gramática.

Ejemplo: Segunda Parte

$S \rightarrow ABb,$	$S \rightarrow ABC,$	$C \rightarrow abC,$	$B \rightarrow bB,$
$A \rightarrow aA,$	$C \rightarrow AB,$	$S \rightarrow Ab,$	$S \rightarrow Bb,$
$S \rightarrow b,$	$S \rightarrow AB,$	$S \rightarrow AC,$	$S \rightarrow BC,$
$S \rightarrow A,$	$S \rightarrow B,$	$S \rightarrow C,$	$C \rightarrow ab,$
$B \rightarrow b,$	$A \rightarrow a,$	$C \rightarrow B,$	$C \rightarrow A,$

$$H = \{A, B, C, S\}$$

Producciones Unitarias

Son las que tienen la forma: $A \rightarrow B$, $A, B \in V$

Algoritmo

Dada una gramática sin producciones nulas G , es posible construir una gramática sin producciones nulas ni unitarias G' que genera el mismo lenguaje.

Si queremos eliminar $A \rightarrow B$, perdemos la posibilidad de

$$A \Rightarrow B \Rightarrow \alpha$$

Para eliminar $A \rightarrow B$, añadimos todas las producciones $A \rightarrow \alpha$ donde $B \rightarrow \alpha$ es una producción.

Si tenemos $B \rightarrow C$ introduciríamos una unitaria que habría que eliminar después.

Para poder eliminar todas de una vez calculamos, H : conjunto de parejas (A, B) tales que B derivable a partir de A .

Algoritmo

Se supone que no hay transiciones nulas

H conjunto de parejas (A, B) tales que B derivable a partir de A .

1. $H = \emptyset$
2. Para toda produccion de la forma $A \rightarrow B$,
la pareja (A, B) se introduce en H .
3. Mientras H cambie
 4. Para cada dos parejas $(A, B), (B, C) \in H$
 5. Si la pareja (A, C) no esta en H
 (A, C) se introduce en H
6. Se eliminan las producciones unitarias
7. Para cada pareja $(B, A) \in H$
 8. Para cada produccion $A \rightarrow \alpha$
 9. Se añade una produccion $B \rightarrow \alpha$

Ejemplo (I)

$S \rightarrow aBc$ $S \not\rightarrow A$ $A \rightarrow aAb$
 $A \not\rightarrow B$ $A \rightarrow cd$ $B \rightarrow ccBS$
 $B \rightarrow dc$

Cálculo de $H = \{(S, A), (A, B), (S, B)\}$

Ejemplo (II)

$$\begin{array}{lll} S \rightarrow aBc & A \rightarrow aAb & A \rightarrow cd \\ B \rightarrow ccBS & B \rightarrow dc & S \rightarrow aAb \\ S \rightarrow cd & A \rightarrow ccBS & A \rightarrow dc \\ S \rightarrow ccBS & S \rightarrow dc & \end{array}$$

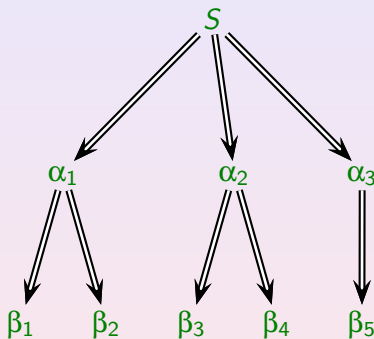
$$H = \{(S, A), (A, B), (S, B)\}$$

$$H = \{(S, A), (A, B), (S, B)\}$$

Esta es la gramática resultante.

El problema de la pertenencia(II)

Problema: Dada G sin producciones nulas ni unitarias y u de longitud n , ¿pertenece u a $L(G)$?



Si la palabra **no es generada**, ¿Hasta **qué profundidad** tenemos que generar para convencernos de que no se puede? $2n - 1$ (en cada paso o sacamos al menos un nuevo símbolo terminal (n) o aumenta al menos en 1 la longitud ($n - 1$))

Forma Normal de Chomsky

Una gramática está en **forma normal de Chomsky** si todas las producciones tienen la forma

$$A \rightarrow BC, \quad A \rightarrow a,$$

donde $A, B, C \in V$, $a \in T$.

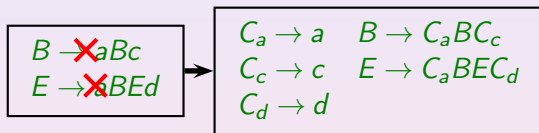
Hay un algoritmo que transforma cualquier gramática sin producciones nulas ni unitarias en otra gramática en forma normal de Chomsky que genera el mismo lenguaje.

Hay dos operaciones básicas:

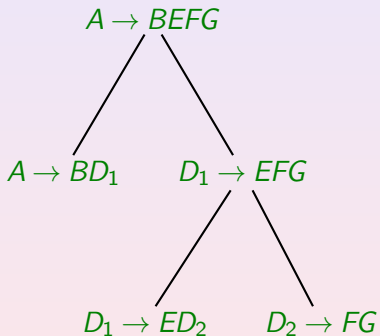
Eliminar terminales en producciones que no sean $A \rightarrow a$:

Transformar producciones con más de dos variables en la parte derecha en un conjunto de producciones equivalentes de la forma $A \rightarrow BC$:

Eliminar terminales en producciones que no sean $A \rightarrow a$



Eliminar producciones con una longitud en la parte derecha mayor de 2:



1. Para cada producción $A \rightarrow \alpha_1 \dots \alpha_n$, $\alpha_i \in (V \cup T)$, $n \geq 2$
2. Para cada α_i , si α_i es terminal: $\alpha_i = a \in T$
 3. Se añade la producción $C_a \rightarrow a$
 4. Se cambia α_i por C_a en $A \rightarrow \alpha_1 \dots \alpha_n$
5. Para cada producción de la forma $A \rightarrow B_1, \dots, B_m$, $m \geq 3$
 6. Se añaden $(m-2)$ variables D_1, D_2, \dots, D_{m-2}
(distintas para cada producción)
 7. La producción $A \rightarrow B_1 \dots B_m$ se reemplaza por
 $A \rightarrow B_1 D_1, \quad D_1 \rightarrow B_2 D_2, \quad \dots, \quad D_{m-2} \rightarrow B_{m-1} B_m$

Ejemplo

$S \rightarrow bA$ $S \rightarrow aB$ $A \rightarrow bAA$ $A \rightarrow AS$
 $A \rightarrow a$ $B \rightarrow aBB$ $B \rightarrow bS$ $B \rightarrow b$



$S \rightarrow C_bA$ $S \rightarrow C_aB$ $A \rightarrow C_bAA$ $A \rightarrow AS$
 $A \rightarrow a$ $B \rightarrow C_aBB$ $B \rightarrow C_bS$ $B \rightarrow b$
 $C_a \rightarrow a$ $C_b \rightarrow b$

Ejemplo (II)

$$\begin{array}{llll} S \rightarrow C_b A & S \rightarrow C_a B & A \rightarrow C_b AA & A \rightarrow AS \\ A \rightarrow a & B \rightarrow C_a BB & B \rightarrow C_b S & B \rightarrow b \\ C_a \rightarrow a & C_b \rightarrow b & A \rightarrow C_b D_1 & D_1 \rightarrow AA \end{array}$$

Ejemplo (II)

$$\begin{array}{llll} S \rightarrow C_b A & S \rightarrow C_a B & & A \rightarrow AS \\ A \rightarrow a & B \rightarrow C_a BB & B \rightarrow C_b S & B \rightarrow b \\ C_a \rightarrow a & C_b \rightarrow b & A \rightarrow C_b D_1 & D_1 \rightarrow AA \\ B \rightarrow C_a E_1 & E_1 \rightarrow BB & & \end{array}$$

Resultado:

$$\begin{array}{llll} S \rightarrow C_b A & S \rightarrow C_a B & & A \rightarrow AS \\ A \rightarrow a & & B \rightarrow C_b S & B \rightarrow b \\ C_a \rightarrow a & C_b \rightarrow b & A \rightarrow C_b D_1 & D_1 \rightarrow AA \\ B \rightarrow C_a E_1 & E_1 \rightarrow BB & & \end{array}$$

Forma Normal de Greibach

Todas las producciones tienen la forma

$$A \rightarrow a\alpha, \quad a \in T, \alpha \in V^*$$

Existe un algoritmo para transformar una gramática en otra equivalente en forma normal de Greibach.

Condiciones para aplicar el algoritmo

Todas las producciones tienen la forma:


- $A \rightarrow a\alpha, \quad a \in T, \alpha \in V^*.$
- $A \rightarrow \alpha, \quad \alpha \in V^*, \quad |\alpha| \geq 2.$

Se cumplen si la gramática está en forma normal de Chomsky.

Dos operaciones básicas:

$$A \rightarrow B\alpha \quad B \rightarrow \beta_1 \quad B \rightarrow \beta_2 \quad B \rightarrow \beta_3$$

Concentramos las dos reglas en una:

$$A \Rightarrow B\alpha \Rightarrow \beta_2\alpha$$


Pasamos a:

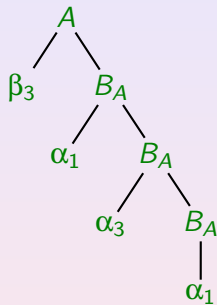
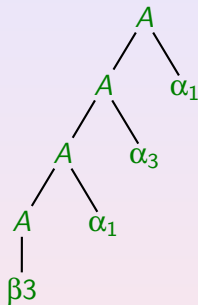
$$A \rightarrow \beta_1\alpha \quad A \rightarrow \beta_2\alpha \quad A \rightarrow \beta_3\alpha$$
$$B \rightarrow \beta_1 \quad B \rightarrow \beta_2 \quad B \rightarrow \beta_3$$

ELIMINA₁($A \rightarrow B\alpha$)

1. Eliminar $A \rightarrow B\alpha$
2. Para cada produccion $B \rightarrow \beta$
3. Añadir $A \rightarrow \beta\alpha$

Segunda Operación

$$\begin{array}{lll} A \rightarrow A\alpha_1 & A \rightarrow A\alpha_2 & A \rightarrow A\alpha_3 \\ A \rightarrow \beta_1 & A \rightarrow \beta_2 & A \rightarrow \beta_3 \end{array}$$



Queda:

$$\begin{array}{lll} A \rightarrow \beta_1 & A \rightarrow \beta_2 & A \rightarrow \beta_3 \\ A \rightarrow \beta_1 B_A & A \rightarrow \beta_2 B_A & A \rightarrow \beta_3 B_A \\ B_A \rightarrow \alpha_1 & B_A \rightarrow \alpha_2 & B_A \rightarrow \alpha_3 \\ B_A \rightarrow \alpha_1 B_A & B_A \rightarrow \alpha_2 B_A & B_A \rightarrow \alpha_3 B_A \end{array}$$

ELIMINA₂(A)

1. Añadir una nueva variable B_A
2. Para cada produccion $A \rightarrow A\alpha$
 3. Añadir $B_A \rightarrow \alpha$ y $B_A \rightarrow \alpha B_A$
 4. Eliminar $A \rightarrow A\alpha$
5. Para cada produccion $A \rightarrow \beta$, β no empieza por A
 6. Añadir $A \rightarrow \beta B_A$.

Primera parte

Objetivo: todas las producciones son de la forma:

- $A \rightarrow a\alpha, \quad a \in T, \alpha \in V^*.$
- $A_i \rightarrow A_j\alpha, \quad j > i, \alpha \in V^*.$
- $B_j \rightarrow A_i\alpha, \quad \alpha \in V^*$

donde (B_k variable que se añade al eliminar A_k con $\text{ELIMINA}_2(A_k)$).

1. Para cada $k = 1, \dots, m$
2. Para cada $j = 1, \dots, k - 1$
3. Para cada produccion $A_k \rightarrow A_j\alpha$
4. $\text{ELIMINA}_1(A_k \rightarrow A_j\alpha)$
5. Si existe alguna produccion de la forma $A_k \rightarrow A_k\alpha$
6. $\text{ELIMINA}_2(A_k)$

Segunda parte

1. Para cada $i = m-1, \dots, 1$
 2. Para cada produccion de la forma $A_i \rightarrow A_j \alpha$, $j > i$
 3. $\text{ELIMINA}_1(A_i \rightarrow A_j \alpha)$
4. Para cada $i = 1, 2, \dots, m$
 5. Para cada produccion de la forma $B_j \rightarrow A_i \alpha$.
 6. $\text{ELIMINA}_1(B_j \rightarrow A_i \alpha)$

Ejemplo

$$\underline{A_1 \rightarrow A_2 A_3}, \quad A_2 \rightarrow A_3 A_1, \quad A_2 \rightarrow b,$$

$$\boxed{A_3 \rightarrow A_1 A_2}, \quad A_3 \rightarrow a$$

Aplicamos ELIMINA_1 a $A_3 \rightarrow A_1 A_2$

Se elimina esta producción y se añade: $A_3 \rightarrow A_2 A_3 A_2$

Queda:

$$A_1 \rightarrow A_2 A_3, \quad A_2 \rightarrow A_3 A_1, \quad A_2 \rightarrow b,$$

$$A_3 \rightarrow a, \quad A_3 \rightarrow A_2 A_3 A_2$$

Ejemplo

$$A_1 \rightarrow A_2 A_3, \quad \underline{A_2 \rightarrow A_3 A_1}, \quad \underline{A_2 \rightarrow b},$$

$$A_3 \rightarrow a, \quad \boxed{A_3 \rightarrow A_2 A_3 A_2}$$

Aplicamos ELIMINA_1 a $A_3 \rightarrow A_2 A_3 A_2$

Se elimina esta producción y se añaden:

$$A_3 \rightarrow A_3 A_1 A_3 A_2, \quad A_3 \rightarrow b A_3 A_2$$

Queda:

$$A_1 \rightarrow A_2 A_3, \quad A_2 \rightarrow A_3 A_1, \quad A_2 \rightarrow b,$$

$$A_3 \rightarrow a, \quad A_3 \rightarrow A_3 A_1 A_3 A_2, \quad A_3 \rightarrow b A_3 A_2$$

Ejemplo

$$\begin{array}{l} A_1 \rightarrow A_2 A_3, \quad A_2 \rightarrow A_3 A_1, \quad A_2 \rightarrow b, \quad \underline{A_3 \rightarrow a}, \\ \boxed{A_3 \rightarrow A_3 A_1 A_3 A_2}, \quad \underline{A_3 \rightarrow b A_3 A_2} \end{array}$$

Aplicamos ELIMINA₂ a A_3

Se añade B_3 y las producciones $B_3 \rightarrow A_1 A_3 A_2$, $B_3 \rightarrow A_1 A_3 A_2 B_3$

Se elimina $A_3 \rightarrow A_3 A_1 A_3 A_2$.

Se añaden las producciones: $A_3 \rightarrow a B_3$, $A_3 \rightarrow b A_3 A_2 B_3$

Queda:

$$\begin{array}{llll} A_1 \rightarrow A_2 A_3, & A_2 \rightarrow A_3 A_1, & A_2 \rightarrow b, & A_3 \rightarrow a, \\ A_3 \rightarrow b A_3 A_2 & B_3 \rightarrow A_1 A_3 A_2, & B_3 \rightarrow A_1 A_3 A_2 B_3 & A_3 \rightarrow a B_3, \\ A_3 \rightarrow b A_3 A_2 B_3 & & & \end{array}$$

Ejemplo

$$\begin{array}{l} A_1 \rightarrow A_2 A_3, \quad \boxed{A_2 \rightarrow A_3 A_1}, \quad A_2 \rightarrow b, \quad \underline{A_3 \rightarrow a}, \\ \underline{A_3 \rightarrow b A_3 A_2} \quad B_3 \rightarrow A_1 A_3 A_2, \quad B_3 \rightarrow A_1 A_3 A_2 B_3 \quad \underline{A_3 \rightarrow a B_3}, \\ \underline{A_3 \rightarrow b A_3 A_2 B_3} \end{array}$$

Se aplica ELIMINA_1 a $A_2 \rightarrow A_3 A_1$

Se elimina esta producción y se añaden:

$$A_2 \rightarrow a A_1, \quad A_2 \rightarrow a B_3 A_1, \quad A_2 \rightarrow b A_3 A_2 B_3 A_1, \quad A_2 \rightarrow b A_3 A_2 A_1$$

Ejemplo

$$\begin{array}{l}
 \boxed{A_1 \rightarrow A_2 A_3}, \quad \underline{A_2 \rightarrow b}, \quad \underline{A_2 \rightarrow a A_1}, \quad \underline{A_2 \rightarrow a B_3 A_1}, \\
 \underline{A_2 \rightarrow b A_3 A_2 B_3 A_1}, \quad \underline{A_2 \rightarrow b A_3 A_2 A_1}, \quad A_3 \rightarrow a, \quad A_3 \rightarrow b A_3 A_2, \\
 \underline{B_3 \rightarrow A_1 A_3 A_2}, \quad \underline{B_3 \rightarrow A_1 A_3 A_2 B_3}, \quad A_3 \rightarrow a B_3, \quad A_3 \rightarrow b A_3 A_2 B_3
 \end{array}$$

Se aplica ELIMINA_1 a $A_1 \rightarrow A_2 A_3$:

Se elimina esta producción y se añaden:

$$A_1 \rightarrow b A_3, \quad A_1 \rightarrow a A_1 A_3, \quad A_1 \rightarrow a B_3 A_1 A_3,$$

$$A_1 \rightarrow b A_3 A_2 B_3 A_1 A_3, \quad A_1 \rightarrow b A_3 A_2 A_1 A_3$$

Ejemplo

$$\begin{array}{lll} A_2 \rightarrow b, & A_2 \rightarrow aA_1 & A_2 \rightarrow aB_3A_1, \\ A_2 \rightarrow bA_3A_2B_3A_1, & A_2 \rightarrow bA_3A_2A_1, & A_3 \rightarrow a, \\ A_3 \rightarrow bA_3A_2, & \boxed{B_3 \rightarrow A_1A_3A_2}, & B_3 \rightarrow A_1A_3A_2B_3, \\ A_3 \rightarrow aB_3, & A_3 \rightarrow bA_3A_2B_3, & \underline{A_1 \rightarrow bA_3}, \\ \underline{A_1 \rightarrow aA_1A_3}, & \underline{A_1 \rightarrow aB_3A_1A_3}, & \underline{A_1 \rightarrow bA_3A_2B_3A_1A_3}, \\ \underline{A_1 \rightarrow bA_3A_2A_1A_3} & & \end{array}$$

Se aplica ELIMINA₁ a $B_3 \rightarrow A_1A_3A_2$ Se elimina esta producción y se añaden:

$$\begin{array}{lll} B_3 \rightarrow bA_3A_3A_2, & B_3 \rightarrow aA_1A_3A_3A_2, & B_3 \rightarrow aB_3A_1A_3A_3A_2, \\ B_3 \rightarrow bA_3A_2B_3A_1A_3A_3A_2, & B_3 \rightarrow aB_3A_1A_3A_3A_2, & \end{array}$$

Ejemplo

$$\begin{aligned}
 A_2 &\rightarrow b, \\
 A_2 &\rightarrow bA_3A_2B_3A_1, \\
 A_3 &\rightarrow bA_3A_2, \\
 A_3 &\rightarrow bA_3A_2B_3, \\
 \underline{A_1 &\rightarrow aB_3A_1A_3}, \\
 B_3 &\rightarrow bA_3A_3A_2, \\
 B_3 &\rightarrow bA_3A_2B_3A_1A_3A_3A_2,
 \end{aligned}$$

$$\begin{aligned}
 A_2 &\rightarrow aA_1 \\
 A_2 &\rightarrow bA_3A_2A_1 \\
 \boxed{B_3 &\rightarrow A_1A_3A_2B_3}, \\
 \underline{A_1 &\rightarrow bA_3}, \\
 \underline{A_1 &\rightarrow bA_3A_2B_3A_1A_3}, \\
 B_3 &\rightarrow aA_1A_3A_3A_2, \\
 B_3 &\rightarrow aB_3A_1A_3A_3A_2,
 \end{aligned}$$

$$\begin{aligned}
 A_2 &\rightarrow aB_3A_1, \\
 A_3 &\rightarrow a, \\
 A_3 &\rightarrow aB_3, \\
 \underline{A_1 &\rightarrow aA_1A_3}, \\
 \underline{A_1 &\rightarrow bA_3A_2A_1A_3} \\
 B_3 &\rightarrow aB_3A_1A_3A_3A_2,
 \end{aligned}$$

Se aplica ELIMINA₁ a $B_3 \rightarrow A_1A_3A_2B_3$. Se elimina esta producción y se añaden:

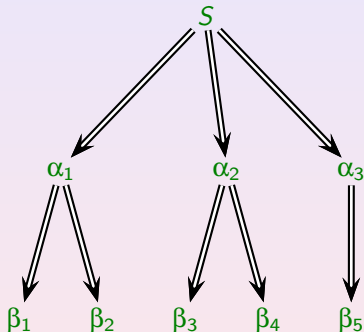
$$\begin{aligned}
 B_3 &\rightarrow bA_3A_3A_2B_3, & B_3 &\rightarrow aA_1A_3A_3A_2B_3, & B_3 &\rightarrow aB_3A_1A_3A_3A_2B_3, \\
 B_3 &\rightarrow bA_3A_2B_3A_1A_3A_3A_2B_3, & B_3 &\rightarrow aB_3A_1A_3A_3A_2B_3,
 \end{aligned}$$

Ejemplo: Resultado

$A_2 \rightarrow b,$	$A_2 \rightarrow aA_1,$	$A_2 \rightarrow aB_3A_1,$
$A_2 \rightarrow bA_3A_2B_3A_1,$	$A_2 \rightarrow bA_3A_2A_1$	$A_3 \rightarrow a,$
$A_3 \rightarrow bA_3A_2,$	$A_3 \rightarrow aB_3,$	$A_3 \rightarrow bA_3A_2B_3,$
$A_1 \rightarrow bA_3,$	$A_1 \rightarrow aA_1A_3,$	$A_1 \rightarrow aB_3A_1A_3,$
$A_1 \rightarrow bA_3A_2B_3A_1A_3,$	$A_1 \rightarrow bA_3A_2A_1A_3,$	$B_3 \rightarrow bA_3A_3A_2,$
$B_3 \rightarrow aA_1A_3A_3A_2,$	$B_3 \rightarrow aB_3A_1A_3A_3A_2,$	$B_3 \rightarrow bA_3A_2B_3A_1A_3A_3A_2,$
$B_3 \rightarrow aB_3A_1A_3A_3A_2,$	$B_3 \rightarrow bA_3A_3A_2B_3,$	$B_3 \rightarrow aA_1A_3A_3A_2B_3,$
$B_3 \rightarrow aB_3A_1A_3A_3A_2B_3,$	$B_3 \rightarrow bA_3A_2B_3A_1A_3A_3A_2B_3,$	$B_3 \rightarrow aB_3A_1A_3A_3A_2B_3$

El problema de la pertenencia(III)

Problema: Dada G en forma normal de Greibach y u de longitud n , ¿pertenece u a $L(G)$?



Si la palabra **no es generada**, ¿Hasta **qué profundidad** tenemos que generar para convencernos de que no se puede? n (en cada paso o sacamos al menos un nuevo símbolo terminal (n)). Además suele haber menos ramificación.

- Construcción de compiladores: *yacc Yet Another Compiler Compiler*.

Visitar: <http://dinosaur.compilertools.net/>

- Para definir textos con instrucciones para formato, como HTML o Latex.
- Para definir datos estructurados como en XML (*Extensible Markup Language*). Los documentos en XML constituyen un lenguaje independiente del contexto.

Visitar:

<http://www.w3schools.com/xml/default.asp>

Además se pueden validar frente a definiciones en DTD (*Document Type Definition*). Una definición en DTD es una gramática independiente del contexto cómo ha de ser un dato válido.