

Autómatas y Lenguajes Formales 2017-1
Facultad de Ciencias UNAM
Nota de Clase 10, Gramáticas libres de contexto

Favio E. Miranda Perea A. Liliana Reyes Cabello Lourdes González Huesca

17 de mayo de 2019

La normalización de gramáticas consiste en transformar todas las producciones de una gramática de manera que tengan cierta forma sintáctica en particular. Así la normalización de gramáticas libres de contexto es útil para homogeneizar la forma de las producciones además para optimizar los procesos de derivación de cadenas. Con las **formas normales** se facilita una solución al problema de la pertenencia, es decir decidir si una cadena pertenece o no a un lenguaje:

Dada una gramática G y una palabra w , ¿se cumple que $w \in L(G)$?
Es decir, pertenece w al lenguaje generado por G .

- Si la palabra w es generada por G , el proceso para la construcción de un árbol de derivación terminará eventualmente.
- En caso contrario **no** podemos saber cuándo parar en la construcción de un árbol de derivación.

1. Normalización de Gramáticas

Para obtener gramáticas normales, son necesarias varias transformaciones que simplificarán y reorganizarán las producciones.

1.1. Eliminación de variables inútiles

Definición 1 Decimos que una variable A es **accesible** o **alcanzable** si existen $u, v \in (V \cup T)^*$ tales que $S \xrightarrow{*} uAv$. Obsérvese que según esta definición S siempre es alcanzable.

Definición 2 Una variable A es **productiva** o **terminable** si existe $w \in T^*$ tal que $A \xrightarrow{*} w$. En particular si $A \xrightarrow{\epsilon}$ es una producción entonces A es productiva.

Definición 3 Una variable A es **inútil** si no es alcanzable o no es productiva.

A continuación damos un algoritmo para hallar variables productivas, mediante la transformación de una gramática:

Iniciar el conjunto $Prod$ con todas las producciones que contienen cadenas de terminales a la derecha de \rightarrow :

$$Prod := \{A \in V \mid A \rightarrow w \in P, w \in T^*\}$$

Repetir la incorporación de variables cuyas producciones contienen variables productivas y símbolos terminales a la derecha de \rightarrow :

$$Prod := Prod \cup \{A \in V \mid A \rightarrow w, w \in (T \cup Prod)^*\}$$

Hasta que no se añaden nuevas variables a $Prod$

Ejemplo: Calculemos las variables productivas de la gramática:

$$\begin{aligned} S &\rightarrow ACD \mid bBd \mid ab \\ A &\rightarrow aB \mid aA \mid C \\ B &\rightarrow aDS \mid aB \\ C &\rightarrow aCS \mid CB \mid CC \\ D &\rightarrow bD \mid ba \\ E &\rightarrow AB \mid aDb \end{aligned}$$

Iniciamos con $Prod = \{S, D\}$. Las iteraciones nos llevan a que C es la única variable improductiva, se elimina esta variable junto con todas las reglas donde figure:

$$\begin{aligned} S &\rightarrow bBd \mid ab \\ A &\rightarrow aB \mid aA \\ B &\rightarrow aDS \mid aB \\ D &\rightarrow bD \mid ba \\ E &\rightarrow AB \mid aDb \end{aligned}$$

El siguiente algoritmo permite hallar variables accesibles:

Iniciar

$$Acc := \{S\}$$

Repetir la incorporación de variables que aparecen a la derecha de \rightarrow en producciones de variables accesibles:

$$Acc := Acc \cup \{A \in V \mid \exists B \rightarrow uAv \in P, B \in Acc, u, v \in (V \cup T)^*\}$$

Hasta que no se añaden nuevas variables a Acc

Ejemplo: Calculemos las variables accesibles de la gramática:

$$\begin{array}{ll} S \rightarrow aS \mid AaB \mid ACS & D \rightarrow aD \mid DD \mid ab \\ A \rightarrow aS \mid AaB \mid AC & E \rightarrow FF \mid aa \\ B \rightarrow bB \mid DB \mid BB & F \rightarrow aE \mid EF \\ C \rightarrow aDa \mid ABD \mid ab & \end{array}$$

Iniciamos con $Acc = \{S\}$. El resultado es: $Acc = \{S, A, B, C, D\}$ ya que E y F son variables inaccesibles, se eliminan junto con todas las reglas donde figuren:

$$\begin{array}{l} S \rightarrow aS \mid AaB \mid ACS \\ A \rightarrow aS \mid AaB \mid AC \\ B \rightarrow bB \mid DB \mid BB \\ C \rightarrow aDa \mid ABD \mid ab \\ D \rightarrow aD \mid DD \mid ab \end{array}$$

Para eliminar variables inútiles se aplican los dos algoritmos anteriores **en el siguiente orden:**

1. Eliminar variables no productivas.
2. Eliminar variables no accesibles.

La importancia del orden de los algoritmos radica en que si se aplican los algoritmos en orden inverso el resultado puede ser una gramática que aún contenga variable inútiles. Veamos un ejemplo:

Ejemplo: Considere la siguiente gramática

$$S \rightarrow a \mid AB \quad A \rightarrow aA \mid \epsilon$$

Al eliminar primero las variables no accesibles se obtiene la misma gramática, al ser todas las variables accesibles.

Posteriormente al eliminar variables improductivas resulta

$$S \rightarrow a \quad A \rightarrow aA \mid \epsilon$$

y claramente A es inútil por ser inaccesible.

1.2. Eliminación de ϵ -producciones

Las gramáticas libres de contexto permiten el uso de producciones de la forma $A \rightarrow \epsilon$. La eliminación de estas producciones llamadas ϵ -producciones genera una transformación.

Definición 4 Una variable A se llama **anulable** si $A \rightarrow^* \epsilon$, es decir si una derivación que empieza en A genera la cadena vacía.

Veamos un algoritmo para hallar variables anulables:

Iniciar el conjunto $Anul$ con las variables que tienen ϵ como producción

$$Anul := \{A \in V \mid A \rightarrow \epsilon \in P\}$$

Repetir la incorporación de variables que tienen producciones cadenas de variables anulables

$$Anul := Anul \cup \{A \in V \mid \exists A \rightarrow w \in P, w \in Anul^*\}$$

Hasta que no se añaden nuevas variables a $Anul$

Una vez que se han identificado las variables anulables, la siguiente transformación de una gramática libre de contexto elimina exactamente las ϵ -producciones:

Para cada producción en la gramática que tenga la forma $A \rightarrow w_1 \dots w_n$ se deben agregar las producciones $A \rightarrow v_1 \dots v_n$ que son resultantes de los cambios de símbolos donde:

- $v_i = w_i$ si $w_i \notin Anul$, se respetan las variables no anulables
- $v_i = w_i$ ó $v_i = \epsilon$ si $w_i \in Anul$, las variables anulables pueden dejarse o eliminarse

Verificando que no se anulen todos los v_i al mismo tiempo.

Es decir, se van a respetar las producciones existentes y si alguna contiene las variables anulables se agregarán las producciones que resulten de eliminar las variables anulables en esa producción. Las ϵ -producciones desaparecerán.

Ejemplo: Eliminación de ϵ -producciones de la gramática

$$\begin{aligned} S &\rightarrow AB \mid ACA \mid ab \\ A &\rightarrow aAa \mid B \mid CD \\ B &\rightarrow bB \mid bA \\ C &\rightarrow cC \mid \epsilon \\ D &\rightarrow aDc \mid CC \mid ABb \end{aligned}$$

Primero obtenemos las variables anulables, iniciando con $Anul = \{C\}$:

$$Anul = \{C, D, A, S\}$$

El proceso de anulación de variables hace que se elimine la producción $C \rightarrow \epsilon$, se dejan las producciones existentes y se agregan las producciones que eliminan los elementos de $Anul$. Se obtiene la siguiente gramática:

$$\begin{aligned} S &\rightarrow AB \mid ACA \mid ab \mid \mathbf{B} \mid \mathbf{CA} \mid \mathbf{AA} \mid \mathbf{AC} \mid \mathbf{A} \mid \mathbf{C} \\ A &\rightarrow aAa \mid B \mid CD \mid \mathbf{aa} \mid \mathbf{C} \mid \mathbf{D} \\ B &\rightarrow bB \mid bA \mid \mathbf{b} \\ C &\rightarrow cC \mid \mathbf{c} \\ D &\rightarrow aDc \mid CC \mid ABb \mid \mathbf{ac} \mid \mathbf{C} \mid \mathbf{Bb} \end{aligned}$$

Acerca de la palabra vacía Si originalmente se tenía $\epsilon \in L(G)$ la eliminación de ϵ -producciones genera una gramática que **no** genera a ϵ . Es posible saber si se pierde la palabra vacía al eliminar ϵ -producciones verificando si $S \in Anul$.

Si se quiere recuperar a ϵ debe agregarse un nuevo símbolo inicial S' así como las producciones $S' \rightarrow S$ y $S' \rightarrow \epsilon$. De esta forma, $S' \rightarrow \epsilon$ es la única ϵ -producción permitida.

1.3. Eliminación de producciones unitarias

Definición 5 Una producción de la forma $A \rightarrow B$ donde A y B son ambas variables se llama **producción unitaria**. El **conjunto unitario** de A se define como sigue:

$$Unit(A) = \{B \in V \mid A \rightarrow^* B \text{ usando sólo producciones unitarias}\}$$

Obsérvese que por definición se tiene $A \in Unit(A)$ si existe $A \rightarrow B$.

Ahora veamos un algoritmo para hallar el conjunto $Unit(A)$ para cualquier variable o símbolo no-terminal:

Iniciar

$$Unit(A) := \{A\}$$

Repetir la incorporación de variables que tengan producciones unitarias

$$Unit(A) := Unit(A) \cup \{B \in V \mid \exists C \rightarrow B, C \in Unit(A)\}$$

Hasta que no se añaden nuevas variables a $Unit(A)$.

Para la eliminación de producciones unitarias en una gramática se debe realizar la siguiente transformación, para cada variable se debe calcular su conjunto unitario y realizar las siguientes acciones:

- Para cada $B \in Unit(A)$ y cada producción $A \rightarrow w$ agregar la producción

$$B \rightarrow w$$

- Eliminar todas las producciones unitarias

Ejemplo: Eliminación de producciones unitarias de la gramática:

$$\begin{aligned} S &\rightarrow AS \mid AA \mid BA \mid \varepsilon \\ A &\rightarrow aA \mid a \\ B &\rightarrow bB \mid bC \mid C \\ C &\rightarrow aA \mid bA \mid B \mid ab \end{aligned}$$

Los conjuntos unitarios para cada variable son:

$$\begin{aligned} Unit(S) &= \{S\} & Unit(A) &= \{A\} \\ Unit(B) &= \{B, C\} & Unit(C) &= \{C, B\} \end{aligned}$$

Así la gramática obtenida al eliminar producciones unitarias es:

$$\begin{aligned} S &\rightarrow AS \mid AA \mid BA \mid \varepsilon \\ A &\rightarrow aA \mid a \\ B &\rightarrow bB \mid bC \mid aA \mid bA \mid ab \\ C &\rightarrow aA \mid bA \mid ab \mid bB \mid bC \end{aligned}$$

1.4. Remplazo simple de producciones

Esta transformación eliminará reglas de la forma $A \rightarrow uBv$ donde $B \rightarrow w_1 | w_2 | \dots | w_n$. Esto se logra al redefinir el conjunto de reglas P en la gramática G con

$$P' = (P - \{A \rightarrow uBv\}) \cup \{A \rightarrow uw_1v | uw_2v | \dots | uw_nv\}$$

Ejemplo: Reemplazar la regla $D \rightarrow ABF$ de la gramática siguiente:

$$\begin{aligned} S &\rightarrow DA | aF \\ D &\rightarrow ABF | a \\ A &\rightarrow a \\ B &\rightarrow b | ba \\ F &\rightarrow AF | A \end{aligned}$$

Se obtiene:

$$\begin{aligned} S &\rightarrow DA | aF \\ D &\rightarrow AbF | AbaF | a \\ A &\rightarrow a \\ B &\rightarrow b | ba \\ F &\rightarrow AF | A \end{aligned}$$

1.5. Remoción de producciones recursivas por la izquierda

Una producción recursiva por la izquierda es de la forma $A \rightarrow Aw$ con $w \in (V \cup T)^*$. Ellas serán reemplazadas por producciones cuya recursión es por la derecha de la siguiente forma, para cada variable o símbolo no-terminal A crear dos categorías de reglas:

1. $A_{izq} = \{A \rightarrow Au_i \in P \mid u_i \in (V \cup T)^*\}$
2. $A_{der} = \{A \rightarrow v_j \in P \mid v_j \in (V \cup T)^*\}$

A continuación se obtendrán las reglas para la variable A como sigue:

$$A \rightarrow v_1 | \dots | v_m | v_1Z | \dots | v_nZ$$

Además se incorporarán las siguientes reglas a P :

$$Z \rightarrow u_1Z | \dots | u_nZ | u_1 | \dots | u_n$$

Ejemplo: Considere las siguientes producciones:

$$A \rightarrow Aa | Aab | bb | b$$

Se eliminarán las producciones recursivas por la izquierda:

$$A_{izq} = \{A \rightarrow Aa, A \rightarrow Aab\} \quad A_{der} = \{A \rightarrow bb, A \rightarrow b\}$$

generando la siguiente gramática:

$$\begin{aligned} A &\rightarrow bb | b | bbZ | bZ \\ Z &\rightarrow aZ | abZ | a | ab \end{aligned}$$

1.6. Problema de la pertenencia

Dada una gramática G sin ε -producciones ni producciones unitarias y una palabra w de longitud n , ¿se cumple que $w \in L(G)$? Analicemos esta situación:

- Si la palabra w es generada por G la construcción de un árbol de derivación terminará eventualmente.
- En caso contrario basta con construir un árbol hasta el nivel **2n – 1** para concluir que $w \notin L(G)$, es decir un árbol que pueda tener como hojas a la cadena w .
- En cada paso de la construcción de un árbol se obtiene un nuevo terminal (a lo más n pasos) o se aumenta la longitud de la palabra en 1 (a lo más $n - 1$ pasos)

2. Formas Normales

Después de haber introducido varias transformaciones de gramáticas libres de contexto, veamos ahora dos formas normales de gran utilidad.

2.1. Forma Normal de Chomsky

Definición 6 Una gramática libre de contexto G está en **forma normal de Chomsky (FNC)** si:

- G no contiene variables inútiles.
- G no contiene producciones unitarias ni ε -producciones (salvo $S \rightarrow \varepsilon$)
- Todas las producciones son binarias con variables o terminales, es decir de la forma:

$$A \rightarrow BC \quad \text{o} \quad A \rightarrow a \quad \text{donde } B, C \in V, a \in T$$

Cualquier gramática libre de contexto es equivalente a una gramática en **FNC**, lo cual se logra parcialmente como sigue:

1. Eliminar las variables inútiles.
2. Eliminar las ε -producciones, salvo cuando la cadena ε pertenece al lenguaje original $L(G)$ y en ese caso se agrega un nuevo símbolo inicial S' y $S' \rightarrow \varepsilon \mid S$.
3. Eliminar producciones unitarias.
4. Las producciones restantes son todas de la forma $A \rightarrow a$ con $a \in T$ ó $A \rightarrow w$ con $|w| \geq 2$

El proceso restante es la eliminación de producciones $A \rightarrow w$ donde $|w| \geq 2$. A este proceso le llamaremos **simulación de producciones**, para ello basta hacer lo siguiente para cada producción P de la forma $A \rightarrow \alpha_1\alpha_2\dots\alpha_n$ con $\alpha_i \in V \cup T$ y $n \geq 2$ (obsérvese que si $n = 2$, al menos uno de α_1, α_2 debe ser terminal, pues si no la producción ya es válida para FNC)

- Si $\alpha_i \in T$, digamos $\alpha_i = a$, entonces
 1. Agregar la producción $T_a \rightarrow a$, donde T_a es una nueva variable.

2. Cambiar α_i por T_a en la producción P
- Para cada producción P de la forma $A \rightarrow B_1B_2 \dots B_m$ con $B_i \in V$, $m \geq 3$
 1. Agregar $(m - 2)$ nuevas variables D_1, \dots, D_{m-2} y reemplazar a P con las siguientes producciones:

$$A \rightarrow B_1D_1 \quad D_1 \rightarrow B_2D_2 \quad \dots \quad D_{m-2} \rightarrow B_{m-1}B_m$$

Ejemplo: Simulación de producciones $A \rightarrow w_1w_2 \dots w_n$, $n \geq 2$.

La producción $A \rightarrow abBaC$ se simula con producciones simples y binarias como sigue:

- Agregamos las nuevas variables T_a, T_b y las producciones

$$A \rightarrow T_aT_bBT_aC \quad T_a \rightarrow a \quad T_b \rightarrow b$$

- Para simular la producción $A \rightarrow T_aT_bBT_aC$ agregamos nuevas variables D_1, D_2, D_3 y las producciones binarias necesarias obteniendo finalmente la gramática:

$$\begin{array}{llll} A \rightarrow T_aD_1 & D_1 \rightarrow T_bD_2 & D_2 \rightarrow BD_3 & D_3 \rightarrow T_aC \\ & & T_a \rightarrow a & T_b \rightarrow b \end{array}$$

Ejemplo: Transformar la siguiente gramática a FNC:

$$\begin{array}{ll} S \rightarrow AB \mid aBC \mid SBS & A \rightarrow aA \mid C \\ B \rightarrow bbB \mid b & C \rightarrow cC \mid \epsilon \end{array}$$

La gramática equivalente es:

$$\begin{array}{llll} S \rightarrow AB \mid T_aD_1 \mid SD_2 \mid T_aB \mid T_bD_3 \mid b & D_1 \rightarrow BC & T_a \rightarrow a \\ A \rightarrow T_aA \mid T_cC \mid a \mid c & D_2 \rightarrow BS & T_b \rightarrow b \\ B \rightarrow T_bD_3 \mid b & D_3 \rightarrow T_bB & T_c \rightarrow c \\ C \rightarrow T_cC \mid c & & \end{array}$$

2.2. Forma Normal de Greibach

Otra forma normal es la llamada de Greibach, esta forma normal es útil en algoritmos de parsing ya que se da prioridad a los prefijos terminales de las cadenas generadas por las reglas de producción de la gramática. Esto asegura que no existe recursión por la izquierda de ninguna forma.

Definición 7 Una gramática libre de contexto G está en **forma normal de Greibach (FNG)** si:

- La variable inicial S no es recursiva, es decir, no figura en el lado derecho de las producciones.
- G no tiene variables inútiles ni ϵ -producciones salvo $S \rightarrow \epsilon$.
- Todas las producciones son de la forma $A \rightarrow aa$ con $a \in T$ y $\alpha \in V^*$.

Veamos cómo transformar una gramática en una equivalente en forma normal de Greibach, obsérvese que cualquier gramática libre de contexto es equivalente a una gramática en **FNG**.

El proceso requiere primero que la gramática esté en Forma Normal de Chomsky, de esta forma se eliminan las producciones inútiles y las ϵ -producciones.

Después, se aplica el siguiente método para transformar a **FNG**:

1. Verificar que el símbolo inicial no sea recursivo, si lo es entonces agregar un nuevo símbolo inicial con las producciones del anterior.
2. Eliminar las producciones recursivas por izquierda.
3. Enumerar u ordenar las variables o símbolos no-terminales de la gramática: iniciar con el símbolo inicial y el resto de las variables siguen cualquier orden.
4. Reemplazar las producciones que contengan variables de orden mayor en el lado derecho de \rightarrow , es decir todas las reglas deben tener la forma:
 - $A \rightarrow Bw$ donde la variable A debe estar por debajo de la variable B en el orden propuesto.
 - $A \rightarrow aw$ con $a \in T$

Finalmente la gramática resultante está en forma normal de Greibach, esta transformación permite simplificar el problema de la pertenencia: dada una gramática G **en forma normal de Greibach** y una palabra w de longitud n , ¿Se cumple $w \in L(G)$? Es decir, pertenece w al lenguaje generado por G .

- Si w es generada por G la construcción de un árbol de derivación entonces debe terminar eventualmente.
- En caso contrario basta con construir el árbol hasta el nivel n para concluir que $w \notin L(G)$.
- En cada paso se obtiene un nuevo terminal (a lo más n pasos) y por lo general hay una menor ramificación.

Ejemplo: Transformar la siguiente gramática a **FNG**.

$$S \rightarrow SaB \mid aB \quad B \rightarrow bB \mid \epsilon$$

1. Eliminación de variables inútiles

$$Prod = \{B\} \quad Acc = \{S, B\}$$

2. Eliminación de ϵ -producciones

$$\begin{aligned} Anul &= \{B\} & S &\rightarrow SaB \mid aB \mid Sa \mid a \\ B &\rightarrow bB \mid b \end{aligned}$$

3. Eliminación de producciones unitarias

$$Unit(S) = \{S\} \quad Unit(B) = \{B\}$$

4. Simulación de producciones

Símbolos terminales:

$$\begin{array}{ll} S \rightarrow ST_aB \mid T_aB \mid ST_a \mid a & T_a \rightarrow a \\ B \rightarrow T_bB \mid b & T_b \rightarrow b \end{array}$$

Producciones binarias:

$$\begin{array}{ll} S \rightarrow SD_1 \mid T_aB \mid ST_a \mid a & T_a \rightarrow a \\ B \rightarrow T_bB \mid b & T_b \rightarrow b \\ D_1 \rightarrow T_aB & \end{array}$$

5. Símbolo inicial no-recursivo

$$\begin{array}{ll} S' \rightarrow SD_1 \mid T_aB \mid ST_a \mid a & T_a \rightarrow a \\ S \rightarrow SD_1 \mid T_aB \mid ST_a \mid a & T_b \rightarrow b \\ B \rightarrow T_bB \mid b & \\ D_1 \rightarrow T_aB & \end{array}$$

6. Eliminación de producciones recursivas por la izquierda

$$\begin{array}{ll} S'_{izq} = \emptyset & S_{izq} = \{S \rightarrow SD_1, S \rightarrow ST_a\} \\ B_{izq} = \emptyset & D_{1izq} = \emptyset \\ T_{bizq} = \emptyset & T_{bizq} = \emptyset \end{array}$$

$$\begin{array}{lll} S' \rightarrow SD_1 \mid T_aB \mid ST_a \mid a & & T_a \rightarrow a \\ S \rightarrow SD_1 \mid T_aB \mid ST_a \mid a \mid T_aBZ & & T_b \rightarrow b \\ B \rightarrow T_bB \mid b & & \\ D_1 \rightarrow T_aB & & \\ Z \rightarrow D_1Z \mid T_aZ \mid D_1 \mid T_a & & \end{array}$$

7. Reemplazo simple de producciones

<i>orden</i>		<i>reemplazar</i>
0	$S' \rightarrow SD_1 \mid T_aB \mid ST_a \mid a$	T_a ya que se debe empezar con un terminal
1	$S \rightarrow SD_1 \mid T_aB \mid ST_a \mid a \mid T_aBZ$	T_a ya que se debe empezar con un terminal
2	$B \rightarrow T_bB \mid b$	T_b ya que se debe empezar con un terminal
3	$D_1 \rightarrow T_aB$	T_a ya que se debe empezar con un terminal
4	$Z \rightarrow D_1Z \mid T_aZ \mid D_1 \mid T_a$	D_1 ya que el orden de D_1 es menor que el de Z
5	$T_a \rightarrow a$	T_a ya que se debe empezar con un terminal
6	$T_b \rightarrow b$	--

$$\begin{array}{l} S' \rightarrow aBD_1 \mid aD_1 \mid aBZD_1 \mid aZD_1 \mid aB \mid aBT_a \mid aT_a \mid aBZT_a \mid aZT_a \mid a \\ S \rightarrow aB \mid a \mid aBZ \mid aZ \\ B \rightarrow bB \mid b \\ D_1 \rightarrow aB \\ Z \rightarrow aBZ \mid aZ \mid aB \mid a \\ T_a \rightarrow a \\ T_b \rightarrow b \end{array}$$