

## TRANSFORMACIONES SOBRE LAS GRAMÁTICAS LIBRES DEL CONTEXTO

Basado en “Languages and Machines (An Introduction to the Theory of Computer Science).”

Chapter 5 - Thomas Sudkamp

**Recordar:** estas son notas de clase y, por lo tanto, se debe completar el estudio de los temas utilizando la bibliografía sugerida.

### 1. Formas Normales

Una **forma normal** es definida imponiendo restricciones sobre la forma de las reglas permitidas en una gramática.

Las transformaciones consisten en una serie de técnicas que agregan y eliminan producciones, considerando que cada uno de estos esquemas de reemplazo preservan el lenguaje generado por la gramática.

Las restricciones impuestas sobre las producciones por una forma normal, a menudo aseguran que las derivaciones de las gramáticas tengan ciertas propiedades deseables.

Se impone al símbolo de comienzo, la restricción de no ser recursivo.

Es decir, dicho símbolo está limitado a iniciar las secuencias de derivaciones. Es por ello que, si el símbolo distinguido es recursivo, se agrega al conjunto de producciones una nueva producción, tal cual se puede observar en el siguiente ejemplo:

Sean las siguientes reglas de la gramática  $G$

$$S \rightarrow aS/AB$$

$$A \rightarrow a$$

$$B \rightarrow c/bB$$

Dado que  $G$  es recursivo, entonces se agrega la producción  $S' \rightarrow S$  y el nuevo símbolo distinguido es  $S'$ , entonces se tiene la siguiente gramática  $G'$ :

$$S' \rightarrow S$$

$$S \rightarrow aS/AB$$

$$A \rightarrow a$$

$$B \rightarrow c/bB$$

## 1.1. Eliminación de Producciones Nulas

Un *no terminal* que eventualmente, puede derivar en la cadena nula es llamado un **no terminal anulable**.

Una producción de la forma  $A \rightarrow \lambda$  se llama **producción nula**.

### Algoritmo 1. Construcción de una GLC libre de no terminales anulables - Algoritmo 5.1.1

Dada una GLC  $G = \langle N, \Sigma, P, S \rangle$

Entrada: Una GLC  $G' = \langle N', \Sigma, P', S' \rangle$  donde

$G' = G$  si  $S$  no aparece en el lado derecho de ninguna producción

Caso contrario, se designa un nuevo símbolo inicial  $S'$

Se incorpora una nueva producción  $S' \rightarrow S$  en  $P'$  y

$N' = N \cup \{S'\}$

Salida: Una GLC  $G_L = \langle N_L, \Sigma, P_L, S_L \rangle$  que satisface

- a.  $L(G_L) = L(G)$
  - b.  $S_L$  no aparece en el lado derecho de ninguna producción
  - c.  $A \rightarrow \lambda \in P_L$  si y sólo si  $\lambda \in L(G)$  y  $A = S_L$
1. Se construye el conjunto NULL de no terminales anulables (ver Algoritmo 2)
  2.  $N_L$  es igual a  $N'$
  3.  $P_L$  se define como:
    - 3.1. Si  $\lambda \in L(G')$  entonces  $S_L \rightarrow \lambda \in P_L$
    - 3.2. Si  $A \rightarrow w \in P'$ , y  $w$  puede ser escrito como:
 

$w_1 A_1 w_2 A_2 \dots w_k A_k w_{k+1}$  donde  $A_1, A_2, \dots, A_k$  son un subconjunto de las ocurrencias de no terminales anulables en  $w$ , entonces  $A \rightarrow w_1 w_2 \dots w_k w_{k+1}$  es una producción de  $P_L$

Si  $w$  contiene  $n$  no terminales anulables, se obtendrán  $2^n$  producciones

Por ejemplo, si  $A, B, C \in N$ ,  $A, B \in \text{NULL}$ ,  $a \in \Sigma$  y  $A \rightarrow CABa \in P'$ .

Luego en  $P_L$  se suman las siguientes producciones:

$A \rightarrow CABa$   
 $A \rightarrow CABa$   
 $A \rightarrow CBa$   
 $A \rightarrow CAa$   
 $A \rightarrow Ca$
    - 3.3. Si  $A \rightarrow w \in P'$  y no cumple con el punto 3.2, luego  $A \rightarrow w \in P_L$
    - 3.4.  $A \rightarrow \lambda \in P_L$  sólo si  $\lambda \in L(G)$  y  $A = S_L$  (elimina toda producción nula)

Una gramática que no contiene producciones nulas, con la excepción  $S \rightarrow \lambda$  y  $S$  no está en el lado derecho de ninguna producción, se llama **monótona no decreciente**, (noncontracting).

Veamos ahora como obtener el conjunto de no terminales anulables.

**Algoritmo 2. Construcción del conjunto de no terminales anulables****Algoritmo 5.1.2**

Entrada: una GLC  $G = \langle N, \Sigma, P, S \rangle$

Salida: El conjunto de no terminales anulables de  $G$ , llamado NULL

1. NULL :=  $\{A/A \rightarrow \lambda \in P\}$
  2. repetir
    - 2.1 PREV := NULL
    - 2.2 Para cada no terminal  $A \in N$ 

Si hay una producción  $A \rightarrow w$  y  $w \in \text{PREV}^*$  entonces

NULL := NULL  $\cup$   $\{A\}$
- hasta NULL = PREV

**1.1.1. Ejemplo**

A continuación se plantea un ejemplo en el que se eliminan las producciones  $\lambda$ , aplicando los **Algoritmos 1 y 2**:

$$\begin{aligned}
 G &= \langle \{S, A, B, C\}, \{a, b, c\}, P, S \rangle \\
 P &= \{S \rightarrow ACA \\
 &\quad A \rightarrow aAa \mid B \mid C \\
 &\quad B \rightarrow bB \mid b \\
 &\quad C \rightarrow cC \mid \lambda\}
 \end{aligned}$$

1. En primer lugar, según **Algoritmo 1**, hay que observar si el símbolo distinguido es recursivo. En este caso no lo es. Por lo tanto **no es necesario** agregar un nuevo símbolo distinguido, ni la producción  $S' \rightarrow S$ .
2. Construir el conjunto NULL, para ello aplicar el **Algoritmo 2**:

Iteración	NULL	PREV
0	$\{C\}$	
1	$\{A, C\}$	$\{C\}$
2	$\{S, A, C\}$	$\{A, C\}$
3	$\{S, A, C\}$	$\{S, A, C\}$

Dicho conjunto NULL =  $\{S, A, C\}$ , contiene los símbolos no terminales nulos o anulables.

3. Continuando con la aplicación del **Algoritmo 1**, se obtiene  $N_L = \{S, A, B, C\}$

4. Ahora es necesario obtener el conjunto de producciones  $P_L$ :

$$\lambda \in L(G') \Rightarrow S_L \rightarrow \lambda \in P_L$$

$$P_L = \{S \rightarrow \lambda \mid ACA \mid CA \mid AA \mid AC \mid A \mid C$$

$$A \rightarrow aAa \mid B \mid C \mid aa$$

$$B \rightarrow bB \mid b$$

$$C \rightarrow cC \mid c\}$$

## 1.2. Eliminación de Producciones Unitarias

Si  $A, B \in N$ , luego una producción de la forma  $A \rightarrow B$  es llamada una **producción unitaria** y el no terminal  $A$  es un **no terminal unitario**. A continuación se describe un algoritmo que permite eliminar tales producciones.

Al eliminar las producciones unitarias se van a eliminar también las producciones encadenadas, lo cual es deseable, en particular en el contexto de análisis sintáctico, por ejemplo  $A \rightarrow B$ ,  $B \rightarrow C$  y  $C \rightarrow A$ .

---

**Algoritmo 3. Construcción de una GLC libre de no terminales encadenados**  
**Teorema 5.2.3**

Entrada: Una GLC  $G = \langle N, \Sigma, P, S \rangle$  noncontracting

Salida: Una GLC  $G_C = \langle N_C, \Sigma, P_C, S_C \rangle$  que satisface:

- a.  $L(G_C) = L(G)$
  - b.  $G_C$  no contiene producciones encadenadas
1. Para todo  $A \in N$  construir el conjunto CHAIN(A) (ver Algoritmo 4)
  2. La producción  $A \rightarrow w \in P_C$  si hay un no terminal  $B$  y una cadena  $w$  que satisface:
    - 2.1  $B \in \text{CHAIN}(A)$
    - 2.2  $B \rightarrow w \in P$
    - 2.3  $w \notin N$

---

El siguiente algoritmo permite obtener el conjunto de no terminales encadenados a cada uno de los no terminales de la gramática.

**Algoritmo 4. Construcción del conjunto CHAIN(A) - Algoritmo 5.2.1**

Entrada: Una GLC noncontracting  $G = \langle N, \Sigma, P, S \rangle$

Salida: El conjunto de no terminales encadenados para el no terminal A,

CHAIN(A)

1. CHAIN(A) := {A}
  2. PREV :=  $\emptyset$
  3. repetir
    - 3.1 NEW := CHAIN(A) - PREV
    - 3.2 PREV := CHAIN(A)
    - 3.2 Para cada no terminal  $B \in \text{NEW}$  hacer
      - Para cada producción  $B \rightarrow C$  hacer
 
$$\text{CHAIN(A)} := \text{CHAIN(A)} \cup \{C\}$$
- hasta CHAIN(A) = PREV

**1.2.1. Ejemplo**

A continuación se plantea un ejemplo en el que se eliminan las producciones encadenadas, aplicando los **Algoritmo 3 y 4**, utilizando la gramática resultado del ejemplo anterior:

$$\begin{aligned}
 G &= \langle \{S, A, B, C\}, \{a, b, c\}, P, S \rangle \\
 P_L &= \{S \rightarrow \lambda \mid ACA \mid CA \mid AA \mid AC \mid A \mid C \\
 &\quad A \rightarrow aAa \mid B \mid C \mid aa \\
 &\quad B \rightarrow bB \mid b \\
 &\quad C \rightarrow cC \mid c\}
 \end{aligned}$$

1. En primer lugar es necesario analizar, según lo que el **Algoritmo 3** pide como entrada, que la gramática sea monótona no decreciente. En este caso vemos que la gramática dada, es libre de producciones  $\lambda$ , solamente contiene la producción  $S \rightarrow \lambda$ .
2. En primer lugar hay que obtener los conjuntos CHAIN, para cada no terminal de la gramática, según lo que indica el **Algoritmo 4**:

El CHAIN(S) se obtiene de la siguiente manera:

Iteración	PREV	NEW	CHAIN
0	$\emptyset$	{S}	{S}
1	{S}	{A, C}	{S, A, C}
2	{S, A, C}	{B}	{S, A, C, B}
3	{S, A, C, B}		

Como PREV es igual a CHAIN, el algoritmo para, luego  $\text{CHAIN(S)} = \{S, A, C, B\}$ .

De igual manera se obtiene CHAIN(A):

Iteración	PREV	NEW	CHAIN
0	$\emptyset$	$\{A\}$	$\{A\}$
1	$\{A\}$	$\{A, B, C\}$	$\{A, B, C\}$
2	$\{A, B, C\}$		$\{A, B, C\}$

Como PREV es igual a CHAIN, el algoritmo para, luego  $\text{CHAIN}(A) = \{A, B, C\}$ .  
 $\text{CHAIN}(B) = \{B\}$  y  $\text{CHAIN}(C) = \{C\}$ , se deja como ejercicio corroborarlo.

3. Para obtener las reglas de la gramática resultado, se aplica:

La producción  $A \rightarrow w \in P_C$  si hay un no terminal  $B$  y una cadena  $w$  que satisfice:

2.1  $B \in \text{CHAIN}(A)$

2.2  $B \rightarrow w \in P$

2.3  $w \notin N$

Lo que dice el algoritmo es que se reemplaza cada una de las producciones de la forma  $A \rightarrow B$ , con  $A, B \in P_C$ , y  $B \rightarrow w \in P$  por las reglas  $A \rightarrow w$ . Entonces el conjunto de reglas de la gramática resultante será:

$$P_L = \{S \rightarrow \lambda \mid ACA \mid CA \mid AA \mid AC \mid aAa \mid aa \mid bB \mid b \mid cC \mid c \\ A \rightarrow aAa \mid aa \mid bB \mid b \mid cC \mid c \\ B \rightarrow bB \mid b \\ C \rightarrow cC \mid c\}$$

### 1.3. Eliminación de Símbolos Inútiles

Si  $G = \langle N, \Sigma, P, S \rangle$  es una GLC y  $x \in (N \cup \Sigma)$ , luego  $x$  es un **símbolo útil** si existe una derivación  $S \xRightarrow{*} u x v \xRightarrow{*} w$ , donde  $u, v \in (N \cup \Sigma)^*$  y  $w \in \Sigma^*$ . Un símbolo que no es útil, es llamado inútil.

De lo anterior, se puede decir que un **símbolo terminal es útil** si aparece en una cadena del lenguaje generado por  $G$ ; un **símbolo no terminal es útil** si es **alcanzable**, es decir, si se encuentra al menos una ocurrencia de él en una secuencia de derivaciones que comienza desde el símbolo inicial y finaliza en una cadena de terminales.

Este algoritmo permite obtener el conjunto de todos los símbolos no terminales que son alcanzables.

**Algoritmo 5. Construcción de una GLC libre de Símbolos Inútiles****Teorema 5.3.6**

Entrada: Una GLC  $G = \langle N, \Sigma, P, S \rangle$

Salida: Una GLC  $G_U = \langle N_U, \Sigma_U, P_U, S \rangle$  que satisface:

- a.  $L(G_U) = L(G)$
- b.  $G_U$  no contiene símbolos inútiles
1. Contruir la GLC  $G_T$  donde todos los no terminales derivan cadenas de terminales, a través del Algoritmo 6
2. Aplicar a  $G_T$  el Algoritmo 8 para construir el conjunto de no terminales alcanzables
3.  $N_U = \text{REACH}$
4.  $P_U = \{ A \rightarrow w \mid A \rightarrow w \in P_T, A \in \text{REACH}, \text{ y } w \in (\text{REACH} \cup \Sigma)^* \}$
5.  $\Sigma_U = \{ a \in \Sigma \mid a \text{ ocurre en el lado derecho de alguna producción } P_U \}$

**Algoritmo 6. Construcción de una GLC donde todo no terminal deriva cadenas de terminales - Teorema 5.3.3**

Entrada: Una GLC  $G = \langle N, \Sigma, P, S \rangle$

Salida: Una GLC  $G_T = \langle N_T, \Sigma_T, P_T, S \rangle$  que satisface:

- a.  $L(G_T) = L(G)$
- b. Todo no terminal en  $G_T$  deriva una cadena de terminales en  $G_T$
1. Construir el conjunto TERM de no terminales que derivan en cadenas de terminales (ver Algoritmo 7)
2.  $N_T = \text{TERM}$
3.  $P_T = \{ A \rightarrow w \mid A \rightarrow w \in P, A \in \text{TERM}, \text{ y } w \in (\text{TERM} \cup \Sigma)^* \}$
4.  $\Sigma_T = \{ a \in \Sigma \mid a \text{ ocurre en el lado derecho de alguna producción en } P_T \}$

**Algoritmo 7. Construcción del conjunto de no terminales que derivan en cadenas de terminales - Algoritmo 5.3.2**

Entrada: Una GLC  $G = \langle N, \Sigma, P, S \rangle$

Salida: El conjunto de no terminales que derivan en cadenas de terminales, TERM

1. TERM :=  $\{A \mid \text{hay una producción } A \rightarrow w \in P \text{ con } w \in \Sigma^* \}$
2. repetir
  - 2.1 PREV := TERM
  - 2.2 Para cada no terminal  $A \in N$  hacer
 

Si hay una producción  $A \rightarrow w$  y  $w \in (\text{PREV} \cup \Sigma)^*$  entonces

TERM := TERM  $\cup$   $\{A\}$

hasta PREV = TERM

**Algoritmo 8. Conjunto de no terminales alcanzables - Algoritmo 5.3.4**

Entrada: Una GLC  $G = \langle N, \Sigma, P, S \rangle$

Salida: El conjunto de no terminales alcanzables, REACH

1. REACH := {S}
  2. PREV :=  $\emptyset$
  3. repetir
    - 3.1 NEW := REACH - PREV
    - 3.2 PREV := REACH
    - 3.3 Para cada no terminal  $A \in$  NEW hacer
      - Para cada producción  $A \rightarrow w$  hacer
      - agregar todos los no terminales en  $w$  a REACH
- hasta REACH = PREV

**1.3.1. Ejemplo**

A partir de la siguiente GLC, obtener una GLC equivalente pero que no contenga símbolos inútiles, aplicando los **Algoritmo 5, 6, 7 y 8**:

$$G = \langle \{S, A, B, C, D, E, F\}, \{a, b, c\}, P, S \rangle$$

$$P_L = \{S \rightarrow AC \mid BS \mid B$$

$$A \rightarrow aA \mid aF$$

$$B \rightarrow CF \mid b$$

$$C \rightarrow cC \mid D\}$$

$$D \rightarrow aD \mid BD \mid C\}$$

$$E \rightarrow aA \mid BSA\}$$

$$F \rightarrow bB \mid b\}$$

1. Construir la gramática  $G_T$  a través del **Algoritmo 6 y 7**:

Iteración	TERM	PREV
0	{B, F}	
1	{B, F, A, S}	{B, F}
2	{B, F, A, S, E}	{B, F, A, S}
3	{B, F, A, S, E}	{B, F, A, S, E}

2. A partir del conjunto TERM obtenido en el punto anterior, se obtiene la gramática  $G_T$ , según lo que dice el **Algoritmo 6**:

$$G_T = \langle \{S, A, B, E, F\}, \{a, b\}, P_T, S \rangle$$

$$P_T = \{S \rightarrow BS \mid B$$

$$A \rightarrow aA \mid aF$$

$$B \rightarrow b$$

$$E \rightarrow aA \mid BSA\}$$

$$F \rightarrow bB \mid b\}$$

3. El siguiente paso es aplicar el **Algoritmo 8**, para obtener los no terminales que son alcanzables desde el símbolo distinguido, para obtener la GLC  $G_U$  a partir de  $G_T$ :

Iteración	REACH	PREV	NEW
0	$\{S\}$	$\emptyset$	
1	$\{S, B\}$	$\{S\}$	$\{S\}$
2	$\{S, B\}$	$\{S, B\}$	$\{B\}$

4. A partir de aquí se aplican los últimos pasos del **Algoritmo 5**:

3.  $N_U = \text{REACH}$

4.  $P_U = \{ A \rightarrow w \mid A \rightarrow w \in P_T, A \in \text{REACH}, \text{ y } w \in (\text{REACH} \cup \Sigma)^* \}$

5.  $\Sigma_U = \{ a \in \Sigma \mid a \text{ ocurre en el lado derecho de alguna producción } P_U \}$

$$G_U = \langle \{S, B\}, \{b\}, P_U, S \rangle$$

$$P_U = \{ S \rightarrow BS \mid B \rightarrow b \}$$

#### 1.4. Forma Normal de Chomsky (FNC)

**Definición:** Una GLC  $G = \langle N, \Sigma, P, S \rangle$  está en Forma Normal de Chomsky si las producciones son de la forma:

1.  $A \rightarrow BC$

2.  $A \rightarrow a$

3.  $S \rightarrow \lambda$ ,

donde  $B, C \in N - \{S\}$

**Algoritmo 9. Construcción de una GLC en FNC - Teorema 5.4.2**

Entrada: Una GLC  $G = \langle N, \Sigma, P, S \rangle$  que cumple con:

- El símbolo de inicio de  $G$  es no recursivo
- $G$  no contiene producciones  $\lambda$ , excepto  $S \rightarrow \lambda$
- $G$  está libre de producciones encadenadas
- $G$  está libre de símbolos inútiles

Salida: Una GLC  $G' = \langle N', \Sigma, P', S \rangle$  que cumple con la definición de la FNC.

- Sea  $A \rightarrow w$  una producción en  $P$  con  $|w| > 1$ , entonces remover los terminales del lado derecho de tales producciones, de la siguiente manera:

$$A \rightarrow bDcF$$

puede ser reemplazada por las siguientes producciones:

$$A \rightarrow B'DC'F$$

$$B' \rightarrow b$$

$$C' \rightarrow c$$

- Se reemplaza la producción  $A$  por las siguientes producciones:

$$A \rightarrow B'T_1$$

$$T_1 \rightarrow DT_2$$

$$T_2 \rightarrow C'F$$

- Las restantes producciones de  $P$  que cumplen con la definición de la Forma Normal de Chomsky quedan en  $P'$

**1.4.1. Ejemplo**

A partir de la siguiente GLC, obtener una GLC equivalente en Forma Normal Chomsky, aplicando el **Algoritmo 9**:

$$G = \langle \{S, A, B, C\}, \{a, b, c\}, P, S \rangle$$

$$P_L = \{S \rightarrow aABC \mid a$$

$$A \rightarrow aA \mid a$$

$$B \rightarrow bcB \mid bc$$

$$C \rightarrow cC \mid c\}$$

- En primer lugar es necesario corroborar que la gramática dada no tenga el símbolo distinguido recursivo, que no posea producciones  $\lambda$  salvo  $S \rightarrow \lambda$ , ni producciones encadenadas, ni tampoco símbolos inútiles. En este caso la gramática  $G$  cumple con los requisitos de entrada.
- Ahora se aplica el paso 1. del **Algoritmo 9**, para reemplazar los terminales de las partes derechas de cada regla, cuando la longitud de dichas reglas es mayor que 1, el proceso nos va a dar como resultado, lo siguiente:

$$\begin{aligned}
 S &\rightarrow A'ABC \mid a \\
 A' &\rightarrow a \\
 A &\rightarrow A'A \mid a \\
 B &\rightarrow B'C'B \mid B'C' \\
 B' &\rightarrow b \\
 C &\rightarrow C'C \mid c \\
 C' &\rightarrow c
 \end{aligned}$$

3. Por último se aplica el paso 2. del **Algoritmo 9**, con el fin de terminar de construir las reglas de la gramática, de manera tal que las partes derechas de las producciones tengan longitud igual a 2 (dos no terminales) o longitud igual a 1 (un terminal):

$$\begin{aligned}
 S &\rightarrow A'T_1 \mid a \\
 T_1 &\rightarrow AT_2 \\
 T_2 &\rightarrow BC \\
 A' &\rightarrow a \\
 A &\rightarrow A'A \mid a \\
 B &\rightarrow B'T_3 \mid B'C' \\
 T_3 &\rightarrow C'B \\
 B' &\rightarrow b \\
 C &\rightarrow C'C \mid c \\
 C' &\rightarrow c
 \end{aligned}$$

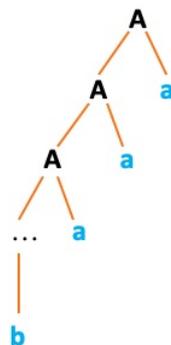
**Ejercicio:** corroborar usando árboles de derivación qué cadenas genera la gramática obtendia. ¿Qué característica tienen los árboles construidos?

## 1.5. Recursión Directa a Izquierda

La recursión directa a izquierda se presenta en producciones de la forma:

$$A \rightarrow Aa \mid b$$

donde la aplicación reiterada de la producción recursiva a izquierda  $A \rightarrow Aa$  produce una cadena de la forma  $Aa^i, i \geq 0$ , y termina la derivación con la producción no recursiva  $A \rightarrow b$ , generando cadenas de la forma  $ba^*$ .



Esta recursión directa a izquierda puede causar algunos problemas, especialmente en el análisis sintáctico, por lo que existe un método que permite eliminar tal tipo de recursión, el cual se puede observar en el siguiente algoritmo.

---

**Algoritmo 10. Eliminación de la recursión directa a izquierda**

Para remover la recursión directa a izquierda, las producciones se dividen en dos categorías:

A. Las producciones directamente recursivas

$$A \rightarrow Au_1 \mid Au_2 \mid \dots \mid Au_j$$

B. Las producciones:

$$A \rightarrow v_1 \mid v_2 \mid \dots \mid v_k \text{ en las cuales el primer símbolo de } v_i \text{ no es } A$$

Entonces para eliminar la recursión se agregan las siguientes producciones eliminando las producciones de las categorías A y B:

$$\begin{aligned} A &\rightarrow v_1 \mid v_2 \mid \dots \mid v_k \mid v_1Z \mid v_2Z \mid \dots \mid v_kZ \\ Z &\rightarrow u_1Z \mid u_2Z \mid \dots \mid u_jZ \mid u_1 \mid u_2 \mid \dots \mid u_j \end{aligned}$$


---

### 1.5.1. Ejemplo

Aplicar el **Algoritmo 10** a las siguientes producciones con recursión directa a izquierda:

1.  $A \rightarrow Aa \mid b$
2.  $A \rightarrow Aa \mid Ab \mid b \mid c$
3.  $A \rightarrow AB \mid BA \mid a$   
 $B \rightarrow b \mid c$

El resultado es:

1.  $A \rightarrow bZ \mid b$   
 $Z \rightarrow aZ \mid a$
2.  $A \rightarrow bZ \mid cZ \mid b \mid c$   
 $Z \rightarrow aZ \mid bZ \mid a \mid b$
3.  $A \rightarrow BAZ \mid aZ \mid BA \mid a$   
 $Z \rightarrow BZ \mid B$   
 $B \rightarrow b \mid c$

## 1.6. Recursión a derecha

Al eliminar la recursión a izquierda, se genera para algunas producciones recursión a derecha. También es deseable, en el contexto de análisis sintáctico eliminar dicha recursión. Para ello la idea es expresar la gramática en forma iterativa, para lo cual se utiliza la notación *BNFE* (*BNFExtendida*).

Es posible también, utilizar la factorización (como en matemáticas), como se observa en el siguiente ejemplo.

$$Z \rightarrow aZ \mid bZ \mid a \mid b$$

La misma puede ser escrita usando BNFE de la siguiente manera:

$$\langle Z \rangle ::= (a|b)\{a|b\}^*$$