Lenguajes regulares

Como se señaló, éstos son los lenguajes más restringidos en la jerarquía de Chomsky pero ampliamente utilizados en informática. Son necesarios durante la etapa de análisis léxico de los compiladores (para estudiar la morfología del lenguaje), que tiene como tarea agrupar símbolos que tienen en conjunto un significado colectivo para el lenguaje, separando al programa fuente en *componentes léxicos o tokens*: ***1959*** es una secuencia de cuatro dígitos que tiene el significado de *número entero*, ***importe*** es el nombre de una posible variable (*identificador*) válida en la mayoría de los lenguajes de programación. Todos los números de punto flotante constituyen un lenguaje regular, lo mismo que todas las palabras clave de un lenguaje de programación y todos los símbolos de puntuación del mismo.

Una palabra o conjunto de palabras con un patrón común (por ejemplo, nombres de archivo que terminen con ***.temp***) que suelen ser buscadas en un archivo de texto por utilitarios de línea de comando como *grep*, *find*, *dir*, *ls*, *vi*, etcétera, conforman un lenguaje regular. Son infinitos los ejemplos.

Lenguaje regular

Los lenguajes regulares admiten la siguiente definición recursiva:

a) Cualquier lenguaje ***finito*** (con un número natural de cadenas) $\mathbf{L_1}$ definido sobre algún alfabeto Σ, es regular.

b) Si $\mathbf{L_1}$ y $\mathbf{L_2}$ son lenguajes regulares, entonces también lo son *su unión* $\mathbf{L_1 \cup L_2}$ y *su concatenación* $\mathbf{L_1 \circ L_2}$.

c) Si $\mathbf{L_1}$ es un lenguaje regular, entonces *su estrella de Kleene* $\mathbf{L_1^*}$, también es un lenguaje regular.

d) Solo son lenguajes regulares, los construidos con **a)**, **b)** y **c)**.

Ésta es una definición útil y constructiva. Dado un alfabeto, nos permite a partir de bloques constructivos sencillos (símbolos y palabras sobre el alfabeto), construir palabras más complejas que formarán parte del lenguaje.

Además, se han formulado otras diversas representaciones para los lenguajes regulares con el objetivo de utilizarlas en distintos momentos y con distintos objetivos: definición recursiva, definiciones usuales de conjuntos, gramáticas regulares, de tipo 3 o lineales (ya sea por la derecha o por la izquierda), máquinas abstractas (autómatas finitos deterministas, no deterministas y bidireccionales) y expresiones regulares. Todas estas formas de expresar lenguajes regulares son equivalentes y existen teoremas y procedimientos que permiten transformar una en otra para un mismo lenguaje descripto. Se discutirán oportunamente.

Expresiones regulares

Solo se quiere avanzar aquí brevemente, sobre un formalismo denominado *expresiones regulares* que es una forma aún más compacta de especificar lenguajes regulares que las gramáticas tipo 3 de Chomsky ya citadas. Constituyen una notación elegante, concisa y cómoda para denotar lenguajes regulares.

Se definen las ***expresiones regulares*** recursivamente como sigue:

<u>*Base de recursión*</u>: Sea Σ un alfabeto; entonces:

a) $\varnothing$ es una expresión regular que denota al lenguaje vacío (sin palabras): $\mathbf{L(\varnothing) = \varnothing}$.

b) λ es una expresión regular que denota al lenguaje cuyo único elemento es la cadena vacía: $\mathbf{L(\lambda) = \{\lambda\}}$.

c) Cualquier símbolo **a** del alfabeto Σ es una expresión regular que denota al lenguaje cuya única palabra es la de largo unitario formada por ese símbolo: **L(a) = {a}**.

<u>*Paso recursivo*</u>: Si **E** y **F** son expresiones regulares, entonces

d) **E+F** es una expresión regular que denota al lenguaje *unión* de los lenguajes denotados por **E** y por **F**: **L(E+F) = L(E) ∪ L(F)**.

e) **E.F** (o sencillamente **EF**) es una expresión regular que denota al lenguaje *concatenación* de los lenguajes denotados por **E** y por **F**: **L(EF) = L(E) ∘ L(F)**.

f) **E*** es una expresión regular que denota al lenguaje formado por la *estrella de Kleene* del lenguaje denotado por **E**: **L(E*) = [L(E)]***.

g) **(E)** es una expresión regular que denota al mismo lenguaje denotado por **E**: **L((E))=L(E)**.

h) Solo son expresiones regulares las construidas con los pasos **a)** al **g)**.

Esta notación puede ser expandida de varias formas. Comandos de búsqueda de patrones en archivos como *grep*, *fgrep*, *egrep* del sistema operativo UNIX tienen su propia notación extendida para expresiones regulares; el procesador de línea de comandos de los sistemas Windows (*command*) y LINUX (*shell*), expanden sus entradas entendiendo ciertas expresiones regulares.

Ejemplo 2.20

Sea $\Sigma_2 = \{0, 1\}$ el alfabeto binario. El lenguaje que denota la expresión regular **(01+1)***, puede determinarse aplicando la definición paso por paso:

$L((01+1)^*) = [\ L((01+1))\ ]^*$; por f

$= [\ L(01+1)\ ]^*$; por g

$= [\ L(01) \cup L(1)\ ]^*$; por d

$= [\ (L(0) \cdot L(1)) \cup L(1)\ ]^*$; por e

$= [\ (\{0\} \cdot \{1\}) \cup \{1\}\ ]^*$; por c

$= [\ \{01\} \cup \{1\}\ ]^*$; concatenación

$= [\ \{01,1\}\ ]^*$; unión

$= \{01,1\}^0 \cup \{01,1\}^1 \cup \{01,1\}^2 \cup \ldots$; * de Kleene

$= \{\lambda, 01, 1, 0101, 011, 101, 11, \ldots\}$; unión

Reordenando: $= \{\lambda, 1, 01, 11, 011, 101, 0101, \ldots\}$

Nótese en el anterior ejemplo, que con **(01+1)***, en una sola línea de texto se expresa un lenguaje de infinitas cadenas. La conveniencia de esta notación ha hecho que se utilice en la mayoría de las herramientas de software de construcción de compiladores (como *lex, yacc, flex, bison, jflex, cup* y otros) para describir los componentes léxicos de los lenguajes de programación.

Ejemplo 2.21

Sea nuevamente $\Sigma_2 = \{0, 1\}$ el alfabeto binario. Se puede describir el lenguaje de todas las cadenas binarias de longitud par de las siguientes formas:

$$(00 + 01 + 10 + 11)^* \quad o \quad ((0 + 1) (0 + 1))^*$$

Tomando la primera de ellas:

$L((00+01+10+11)^*) =$

$= [L((00+01+10+11))]^*$; por f

$= [L(00+01+10+11)]^*$; por g

$= [L(00) \cup L(01) \cup L(10) \cup L(11)]^*$; por d

$= [(L(0)L(0)) \cup (L(0)L(1)) \cup (L(1)L(0)) \cup (L(1)L(1))]^*$; por e

$= [\{0\}.\{0\} \cup \{0\}.\{1\} \cup \{1\}.\{0\} \cup \{1\}.\{1\}]^*$; por c

$= [\{00\} \cup \{01\} \cup \{10\} \cup \{11\}]^*$; concatenac.

$= \{00, 01, 10, 11\}^*$; por unión

$= \{00, 01, 10, 11\}^0 \cup \{00, 01, 10, 11\}^1 \cup \{00, 01, 10, 11\}^2 \cup \ldots$

; * de Kleene

$= \{\lambda, 00, 01, 10, 11, 0000, 0001, 0010, 0011, 0100, 0101,$

$0110, 0111, 1100, 1101, 1110, 1111, \ldots \}$

Se deja como ejercicio mostrar que con la segunda expresión se generan las mismas cadenas del lenguaje indicado.

Lenguajes Independientes del Contexto (LIC)

Como ya se señaló, las gramáticas independientes del contexto son particularmente importantes para las ciencias informáticas porque describen la sintaxis de los lenguajes de programación. Por esto mismo, desde la década del sesenta, se ha invertido mucho esfuerzo en el estudio y la investigación de las características de estas gramáticas y en el desarrollo de procedimientos para el tratamiento algorítmico de los lenguajes que generan. Con estos fines, se han especificado y definido gran cantidad de conceptos que se aplican especialmente a los lenguajes independientes del contexto (LIC) y a sus gramáticas (GIC).

Se introducirán en lo que sigue, algunos de ellos; durante el desarrollo, siempre se estará hablando de una gramática independiente del contexto **$G = (\Sigma_T, \Sigma_N, S, P)$**.

Gramática limpia

Cuando una gramática es diseñada por un especialista, es de esperar que no tenga reglas o símbolos inútiles; si se incluyeron en la gramática por algo habrá sido, eran necesarios para la descripción del lenguaje que se estaba definiendo. Sin embargo, en muchos casos las producciones de la gramática de un lenguaje han sido recuperadas automáticamente por un algoritmo desde algún otro dispositivo formal o deducidas desde un conjunto de cadenas dado, por lo que pueden aparecer impurezas y desviaciones. Es entonces menester limpiar las gramáticas de esas impurezas que solo llevan a confusiones y ocupan lugar en la memoria y tiempo de procesamiento, durante su uso y análisis.

Por supuesto, en cada depuración efectuada debe asegurarse que la gramática resultante sea equivalente a la original, esto es, que genere exactamente las mismas cadenas, ni una más, ni una menos.

Se llama ***regla innecesaria*** a una regla del tipo **A:=A** en la que únicamente aparece a ambos lados del símbolo de producción un mismo no terminal. Claramente, si una producción como esta aparece en una gramática, la misma no aporta ningún conocimiento adicional sobre el lenguaje que se está describiendo, ni sobre ninguna de sus cadenas. Aplicada durante una derivación se obtendría nuevamente la misma forma sentencial a la que ya se había arribado, desperdiciándose esfuerzo en este paso.

Por lo anterior, esta regla ***puede eliminarse del conjunto*** **P** de producciones de cualquier gramática, sin ningún cambio adicional en la misma, obteniendo con ello una gramática totalmente

equivalente. Debe notarse que, siendo el conjunto **P** de producciones finito, una regla innecesaria siempre puede detectarse por simple inspección secuencial de los elementos del conjunto **P**.

Ejemplo 2.22

La siguiente gramática genera un lenguaje con cadenas de la forma $\mathbf{a^n cb^n}$ para cualquier entero positivo **n**:

$$G_{(22.1)} = (\ \{a, b, c\}, \{S, A\}, S, \{S := aAb,\ A := A \mid aAb \mid c\}\ )$$

Nótese que la regla **A:=A** no aporta absolutamente nada al proceso de generación de estas cadenas; al eliminarla se obtiene:

$$G_{(22.2)} = (\ \{a, b, c\}, \{S, A\}, S, \{S := aAb,\ A := aAb \mid c\}\ )$$

que es equivalente a la anterior ya que $\mathbf{L(G_{(22.1)}) = L(G_{(22.2)})}$.

Por otra parte, puede darse el caso de que alguno de los símbolos terminales o no terminales de la gramática, no pueda ser alcanzado desde el axioma por ninguna derivación válida; esto significa que usando las producciones de la gramática y partiendo desde el axioma, no hay manera de derivar una forma sentencial que lo contenga. Diremos en este caso que el símbolo $\mathbf{x \in (\Sigma_T \cup \Sigma_N)}$ es ***inaccesible***. En símbolos:

$$\mathbf{x}\ \textit{es inaccesible} \Leftrightarrow \nexists\ (S \rightarrow^* \alpha x \beta) \quad \text{con} \quad \alpha, \beta \in (\Sigma_T \cup \Sigma_N)^*$$

Si esto ocurre, el símbolo **x** no intervendrá en la derivación de ninguna cadena del lenguaje generado por la gramática, por lo que si se lo quita, y junto con él todas las producciones que lo contengan, no se alterará el lenguaje descripto.

Ejemplo 2.23

Si la gramática $\mathbf{G_{(22.2)}}$ del ejemplo anterior hubiera tenido más terminales y/o no terminales en sus alfabetos, como por ejemplo:

$$G_{(23.1)} = (\ \{a, b, c, d\}, \{S, A, X\}, S, \{S := aAb,\ A := aAb \mid c\}\ )$$

pero con las mismas producciones, claramente seguiría generando el mismo lenguaje con cadenas de la forma $\mathbf{a^n cb^n}$ con $\mathbf{n>0}$.

Debe notarse que el símbolo no terminal **X** y el terminal **d** son, en este caso, símbolos inútiles porque nunca intervienen en ninguna derivación, ya que no están en las producciones: ***son inaccesibles*** desde el axioma. Entonces, se los puede quitar sin más trámite de los alfabetos, para volver a obtener una gramática equivalente:

$$G_{(23.2)} = (\ \{a, b, c\ \}, \{S, A\}, S, \{S := aAb,\ A := aAb \mid c\}\ )$$

La Tabla 2.3 aplicada en este ejemplo, muestra un procedimiento posible para detectar símbolos inaccesibles:

	S	A	X	a	b	c	d
S := aAb	•	•		•	•		
A := aAb		•		•	•		
A := c						•	

Tabla 2.3: Determinación de símbolos inaccesibles del Ejemplo 2.23.

1. Se construye una tabla con tantas filas como producciones tenga la gramática (etiquetando cada fila con una producción, iniciando con las reglas del axioma) y tantas columnas como símbolos de $\Sigma_N \cup \Sigma_T$, etiquetando cada columna con un símbolo no terminal o terminal.
2. Se marca con un punto el axioma en la primera casilla de la tabla.
3. Para cada producción del axioma **S**, se marca con un punto la celda correspondiente a los símbolos terminales y no terminales que componen el lado derecho de la misma.
4. Para cada no terminal marcado en el paso anterior (salvo el axioma), se inspeccionan todas sus producciones, marcando en la tabla los símbolos que conforman el lado derecho de las mismas.
5. Se repite el paso 4, hasta que no quede ningún símbolo no terminal marcado por analizar. Al finalizar, las columnas sin ninguna marca indican cuáles son los símbolos inaccesibles.

Por otro lado, se sabe que el lenguaje formal generado por una gramática es un conjunto de cadenas de símbolos terminales derivables desde el axioma; si existiera algún símbolo no terminal en la gramática desde el cual nunca se pudiese llegar a una cadena de terminales utilizando producciones válidas, ese símbolo no terminal no solo sería inútil, sino nocivo para cualquier proceso de derivación que lo utilizare ya que generaría caminos sin salida. Se denomina ***símbolo superfluo*** a un símbolo no terminal que no permite generar desde él al menos una cadena vacía o de solo símbolos terminales:

$$\mathbf{X\ es\ superfluo} \Leftrightarrow \nexists\ (X \rightarrow^* \alpha) \text{ con } \alpha \in \Sigma_T^*$$

Para eliminar estos símbolos inútiles, tanto inaccesibles como superfluos, debe quitárselos del alfabeto respectivo y eliminar todas las producciones del conjunto **P** que los contengan, obteniendo una nueva gramática equivalente a la anterior.

Considere una gramática, en la cual para el símbolo no terminal **X** solo se dispone de producciones del tipo $\mathbf{X:=\alpha X \beta}$ (veremos luego que este tipo de producciones recibe el nombre de *recursivas*. Inclusive podría pensarse en recursión en más de un paso) con alfa y beta cadenas de terminales y no terminales de cualquier largo. Entonces, si se llegase desde el axioma a alguna forma sentencial que lo contenga, $\mathbf{S \rightarrow^* \gamma X \delta}$, esta derivación no podría llegar nunca a una sentencia, ya que todas las producciones de **X** lo contienen, restituyéndolo en la forma sentencial.

Este símbolo superfluo, siempre que no sea el axioma, junto a todas las producciones que lo contengan, pueden entonces eliminarse de la gramática sin cambiar las cadenas de terminales generadas, esto es, el lenguaje generado.

Ejemplo 2.24

Analícese la siguiente gramática:

$$G_{(24.1)} = (\ \{a, b\}, \{S, A, B\}, S, \{S := aAb,\ A := aAb \mid ab \mid aB,\ B := aBb\}\ )$$

El no terminal **B** resulta superfluo, ya que desde **B** es imposible derivar una cadena de terminales. Puede entonces eliminárselo del conjunto de no terminales y quitar las dos producciones que lo contienen, obteniendo la gramática equivalente:

$$G_{(24.2)} = (\ \{a, b\}, \{S, A\}, S, \{S:=aAb,\ A:= aAb \mid ab\}\ )$$

La Tabla 2.4 muestra cómo identificar los símbolos superfluos y sus producciones:

	1	**2**
S := aAb		•
A := aAb		•
A := ab	•	
A := aB		
B := aBb		

Tabla 2.4: Determinación de símbolos superfluos del Ejemplo 2.24.

1. Se construye una tabla con tantas filas como producciones tenga la gramática (etiquetando cada fila con una producción, iniciando por las del axioma) y tantas columnas como iteracio-

nes hagan falta (etiquetadas con el número de iteración), lo que se verá durante el procedimiento.

2. En la primera iteración, se marcan en la primera columna, aquellas producciones donde un no terminal produce solo símbolos terminales, cadenas de terminales o la cadena vacía λ.
3. En la siguiente iteración, se marcan en la siguiente columna, aquellas producciones que tengan en su lado derecho, **terminales y/o** solo no terminales **que estén** del lado izquierdo de las producciones marcadas en la**s** columna**s** anterior**es**.
4. Repitiendo el paso 3 hasta que no puedan marcarse nuevas producciones, quedan finalmente sin marcar las producciones que deben eliminarse y los símbolos superfluos que son los no terminales que figuran en el lado izquierdo de las producciones que nunca fueron marcadas en iteraciones anteriores.

Se dice que una gramática independiente del contexto **está limpia** si y solo si no tiene reglas innecesarias, ni símbolos inaccesibles, ni símbolos superfluos.

Ejemplo 2.25

Se trabajará con la siguiente gramática, analizando los distintos aspectos que determinan si la misma está limpia o no:

$$G_{(25.1)} = (\{a, b, c, d\}, \{A, B, C, D, E\}, A, P_{(25.1)}), \text{ con:}$$

$$P_{(25.1)} = \{A:=Da \mid Eba \mid \lambda,\ B:=bCd \mid d,\ \underline{C:=C},\ D:=bA,\ E:=aE \mid cE\}$$

Reglas innecesarias: claramente la regla **C:=C** es innecesaria, por ello debe quitarse directamente del conjunto de producciones $P_{(25.1)}$, obteniendo la gramática equivalente:

$$G_{(25.2)} = (\{a, b, c, d\}, \{A, \underline{B}, \underline{C}, D, E\}, A, P_{(25.2)}), \text{ donde:}$$

$$P_{(25.2)} = \{A := Da \mid Eba \mid \lambda,\ \underline{B := bCd \mid d},\ D := bA,\ E := aE \mid cE\}$$

Símbolos inaccesibles: primero nótese que desde el axioma **A** de la gramática, siguiendo la primera producción podemos llegar a **D** y siguiendo la segunda a **E**; estos dos no terminales se reescriben con cadenas que involucran nuevamente **A** y **E**, pero no hay forma de utilizar el no terminal **B** o el **C**. Resultan ser entonces **B** y **C** no terminales inaccesibles y deben, por lo tanto, quitarse del conjunto de no terminales y quitar sus producciones que nunca serán utilizadas en derivaciones desde el axioma:

$$G_{(25.3)} = (\{a, b, c, \underline{d}\}, \{A, D, E\}, A, P_{(25.3)}), \text{ donde:}$$

$$P_{(25.3)} = \{A := Da \mid Eba \mid \lambda,\ D := bA,\ E := aE \mid cE\}$$

	A	**B**	**C**	**D**	**E**	**a**	**b**	**c**	**d**
A := Da	●			●		●			
A := Eba					●	●	●		
A := λ									
B := bCd									
B := d									
D := bA	●						●		
E := aE					●	●			
E := cE					●			●	

Tabla 2.5: Determinación de símbolos inaccesibles del Ejemplo 2.25.

En esta nueva gramática, el símbolo terminal **d** ya no puede encontrarse en ninguna producción, por lo cual nunca va a ser utilizado en una derivación desde el axioma; se concluye que es inaccesible y entonces puede ser eliminado del conjunto Σ_T, sin que sea necesario alterar el conjunto de producciones:

$$G_{(25.4)} = (\{a, b, c\}, \{A, D, E\}, A, P_{(25.4)}), \text{ donde:}$$

$$P_{(25.4)} = \{A := Da \mid Eba \mid \lambda,\ D := bA,\ E := aE \mid cE\} = P_{(25.3)}$$

Ahora los símbolos **a, b, D, E** son accesibles desde el axioma con solo una derivación directa aplicando la primera o segunda producción; **c** es accesible con una derivación en dos pasos usando la segunda y luego la sexta producción.

Símbolos superfluos: mírense las producciones del símbolo no terminal **E**; desde **E** solo pueden obtenerse formas sentenciales que volverán a tener como integrante al símbolo **E**, por lo cual no podremos llegar nunca a una cadena de terminales:

$$A \rightarrow Eba \rightarrow aEba \rightarrow acEba \rightarrow \ldots \rightarrow \alpha E \beta$$

Resulta ser **E** un *símbolo superfluo* y debemos eliminarlo junto con todas las producciones que lo contengan (ver Tabla 2.6):

$$G_{(25.5)} = (\{a, b, c\}, \{A, D\}, A, P_{(25.5)}), \text{ donde:}$$

$$P_{(25.5)} = \{A := Da \mid \lambda,\ D := bA\}$$

En esta nueva gramática equivalente, al eliminar las producciones del no terminal **E**, se ha vuelto *símbolo inaccesible* el símbolo terminal **c**, por lo cual debe ser eliminado del correspondiente alfabeto:

$$G_{(25.6)} = (\{a, b\}, \{A, D\}, A, \{A := Da \mid \lambda,\ D := bA\})$$

	1	**2**	**3**
A := Da			•
A := Eba			
A := λ	•		
D := bA		•	
E := aE			
E := cE			

Tabla 2.6: Determinación de símbolos superfluos del Ejemplo 2.25.

Hemos llegado así a una gramática equivalente a la primera, que no tiene reglas innecesarias, ni símbolos inaccesibles, ni superfluos, por lo que **$G_{(25.6)}$ *está limpia***.

Gramática bien formada

Según la jerarquía de Chomsky de los lenguajes formales, el formato de las reglas de una gramática independiente del contexto indica que no pueden existir reglas de reescritura compresoras, esto es, producciones con lado derecho de menor longitud que el lado izquierdo.

En particular, una regla del tipo **A:=λ**, no siendo **A** el axioma de la gramática, es una regla compresora denominada ***regla no generativa***. Si es el axioma el que produce la cadena vacía, **S:=λ**, la regla se permite como excepción y ya sabemos que se denomina *regla lambda*.

Si el lenguaje generado por la gramática contiene como elemento la cadena vacía, el lenguaje debe poder derivar desde el axioma esta cadena, por lo cual una regla lambda no puede quitarse de una gramática sin modificar el lenguaje generado. Sin embargo, sí resulta factible, y en ocasiones deseable, eliminar las reglas no generativas de una gramática.

Pero una regla no generativa no puede eliminarse sin más trámite; se analizará el siguiente caso:

$$G = (\{a, b\}, \{S, A\}, S, \{S := aAb,\ A := aAb \mid \lambda\})$$

Las cadenas generadas por estas producciones son: ab, aabb, aaabbb, …, $a^n b^n$, …, un conjunto infinito de palabras. Si se eliminara la producción no generativa **A:=λ**, con las dos producciones que quedan no se podría derivar ninguna cadena, ya que el símbolo **A** pasaría a ser superfluo y al eliminarlo, el conjunto de producciones quedaría vacío, es decir que el lenguaje generado

pasa de **L(G) = {$a^n b^n$ / n > 0}** teniendo en cuenta **A:=λ**, a **L(G)=∅** en caso de no tenerla en cuenta.

Para poder quitar una regla no generativa **A:=λ,** debe procederse de la siguiente forma:

a) Para cada producción **X:=αAβ** que contenga el no terminal **A** en el lado derecho, agregar la regla de reescritura **X:=αβ** que se obtiene de reemplazar **A** por la cadena vacía.

b) Luego eliminar del conjunto de producciones **A:=λ**, ya que todos los efectos que produciría la misma, han sido incluidos explícitamente como producciones en el paso anterior.

Ejemplo 2.26

Se eliminará la regla no generativa **A:=λ** de la gramática:

$$G_{(26.1)} = (\ \{a, b\}, \{S, A\}, S, \{S := aAb, A := aAb \mid \lambda\}\ )$$

Para ello, notamos que **A** existe en el lado derecho de dos reglas de reescritura, por lo cual debemos analizarlas:

a) **S:=aAb**, esto hace que agreguemos **S:=ab** al conjunto de producciones, donde se ha reemplazado **A** por la cadena vacía.

b) **A:=aAb**, para explicitar la posibilidad de **A:=λ** agregamos la producción **A:=ab**.

Luego de estos cambios puede eliminarse la regla no generativa y se obtiene la gramática equivalente:

$$G_{(26.2)} = (\ \{a, b\}, \{S, A\}, S, \{S := aAb \mid ab, A := aAb \mid ab\}\ )$$

Pasemos a otro posible problema. En los lenguajes naturales como el español, existe lo que se llama *sinonimia* y *polisemia*. Decimos de dos palabras distintas que significan lo mismo (o casi lo mismo, significados semejantes) que son *sinónimos*, palabras que, en general, pueden intercambiarse en un discurso sin cambiar el sentido del mismo; algunos ejemplos podrían ser *alegre/contento*, *marido/esposo*, *regreso/retorno*, *cerdo/puerco/chancho* y muchos otros. Los sinónimos suelen utilizarse para seguir tratando un mismo tema y no ser reiterativo.

Por otro lado, también en los lenguajes naturales existen palabras que pueden tener más de un significado y en este caso decimos que se produce *polisemia*; por dar un caso, *sierra* puede referirse tanto a un tipo de formación montañosa (en geografía), como a la herramienta que sirve para cortar madera o caños, e inclusive a nombres propios de pueblos y personas. El significado específico con el que se usa, suele determinarse según el contexto en el que aparece.

Tanto la sinonimia como la polisemia, le dan al lenguaje natural gran flexibilidad y la capacidad de expresar las mismas ideas de maneras muy distintas, lo que en general es considerado como una característica positiva del lenguaje. Sin embargo, si se mezclan ambas características (una palabra sinónimo de otra que tiene varios significados) podrían presentarse problemas para determinar claramente el significado luego de alguna secuencia de reemplazos.

La situación descripta podría reproducirse en un lenguaje formal generado por una gramática, si tenemos producciones del tipo **A:=B** donde **A** y **B** son símbolos no terminales; esta producción, llamada ***regla de redenominación*** dice que el no terminal **A** puede ser reescrito como **B** en cualquier contexto donde se encuentre (algo así como sinónimos en el lenguaje natural). Sin embargo, **A** puede tener otras reglas que indiquen reescritura (y así tener distintas definiciones del mismo símbolo) y lo mismo puede ocurrir con **B**, pudiéndose generar sinonimia y polisemia simultáneamente. Además, el par de producciones **A:=B** y **B:=A** podrían confundir a no pocas rutinas de análisis sintáctico que las analicen, generando posibles lazos infinitos.

En un lenguaje formal, sobre todo en lenguajes que determinan secuencias específicas de acciones para explicitar procedimientos, como nuestros lenguajes de programación de computadoras, es absolutamente necesario poder determinar sin lugar a dudas:

a) Si una cadena está correctamente escrita (***sintaxis***), esto es que pueda derivarse desde el axioma de la gramática del lenguaje usando las producciones definidas, y

b) El significado de la cadena (***semántica***), qué acciones implica y en qué orden deben realizarse.

Por lo anterior, en general se considera necesario eliminar estas reglas de redenominación de las gramáticas. Como en el caso de las reglas no generativas, para poder quitarlas debemos primero efectuar otros cambios en la gramática, haciendo que el resultado obtenido sea una gramática equivalente. Si se desea eliminar la regla **A:=B** de una gramática, deberemos:

a) Por cada regla **B:=**α existente en la gramática, agregar una regla **A:=**α, lo cual hace explícita como producción la posible derivación en dos pasos **A** $\rightarrow$ **B** $\rightarrow$ α.

b) Luego puede eliminarse **A:=B** del conjunto de producciones y la gramática obtenida será equivalente a la original.

Ejemplo 2.27

Consideremos la gramática:

$$G_{(27.1)} = (\{0, 1\}, \{S, T\}, S, \{S := 0S \mid S1 \mid T, T := 01 \mid 0T\})$$

Esta gramática tiene la regla de redenominación **S:=T**. Por ello, debemos agregar al conjunto de producciones de la gramática **S:=01** y **S:=0T**, para recién poder quitar la producción no deseada. Luego:

$$G_{(27.2)} = (\{0, 1\}, \{S, T\}, S, \{S := 0S \mid S1 \mid 01 \mid 0T, T := 01 \mid 0T\})$$

es equivalente a $G_{(27.1)}$, pero sin reglas de redenominación.

Se dice que una gramática independiente del contexto está **bien formada**, si y solo si, está limpia y no tiene reglas no generativas ni de redenominación.

Ejemplo 2.28

Se desea obtener una gramática bien formada, que describa el mismo lenguaje que genera la siguiente:

$$G_{(28.1)} = (\{0, 1, 2\}, \{S, A, B, C\}, S, P_{(28.1)}), \text{ donde}$$

$$P_{(28.1)} = \{S := AB \mid A, A := 0AS \mid A0 \mid C0 \mid \lambda, B := B1 \mid 1, C:=C0\}$$

Se analiza primero, si está limpia:

Reglas innecesarias: no tiene producciones del tipo **X:=X**.

Símbolos inaccesibles: ninguna regla hace referencia al terminal **2**, por lo que el mismo es innecesario y debe ser quitado del alfabeto de símbolos terminales:

$$G_{(28.2)} = (\{0, 1\}, \{S, A, B, C\}, S, P_{(28.2)}), \text{ con } P_{(28.2)} = P_{(28.1)}$$

Símbolos superfluos: el símbolo no terminal **C** tiene una única regla recursiva que no permite obtener cadenas de terminales desde él, por lo cual es superfluo y debe eliminarse junto a todas las producciones que lo tengan en la parte derecha o izquierda.

$$G_{(28.3)} = (\{0, 1\}, \{S, A, B\}, S, P_{(28.3)}), \text{ donde}$$

$$P_{(28.3)} = \{S := AB \mid A, A := 0AS \mid A0 \mid \lambda, B := B1 \mid 1\}$$

Esta gramática está ahora ***limpia.***

Reglas no generativas: la regla no generativa **A:=**λ debe eliminase, pero antes debemos agregar S:=B, S:=λ, A:=0S y A:=0 con lo cual se hicieron explícitos los efectos de la misma. Luego:

$$G_{(28.4)} = (\{0, 1\}, \{S, A, B\}, S, P_{(28.4)}), \text{ siendo}$$

$$P_{(28.4)} = \{S := AB \mid A \mid B \mid \lambda, A := 0AS \mid A0 \mid 0S \mid 0, B := B1 \mid 1\}$$

S**:=**λ está permitida porque **S** es el axioma (regla lambda). Esta nueva gramática está limpia y sin reglas no generativas pero tiene ahora dos reglas de redenominación: **S:=A** y **S:=B**.

Reglas de redenominación: para eliminar **S:=A**, es necesario hacer que **S** produzca todo lo que produce **A**, agregando **S:=0AS|A0|0S|0**. Para eliminar **S:=B**, se deben agregar **S:=B1|1** haciendo que **S** produzca todo lo que produce **B**.

Luego, la ***gramática bien formada*** resultante es:

$$G_{(28.5)} = (\{0, 1\}, \{S, A, B\}, S, P_{(28.5)}), \text{ con}$$

$$P_{(28.5)} = \{S:=AB|0AS|A0|0S|B1|1|0|\lambda,\ A:=0AS|A0|0S|0,\ B:=B1|1\}$$

Análisis sintáctico

¿ $\alpha \in$ L(G) ?

Las gramáticas formales **G = (Σ_T, Σ_N, S, P)** son herramientas que sirven para describir lenguajes formales; constituyen una notación que permite crear una especificación de un lenguaje determinado, pero de ninguna manera conforman en sí mismas un procedimiento. Es el proceso de derivación el que permite obtener las cadenas del lenguaje **L(G)** descripto por la gramática y, en ese sentido, decimos que la gramática genera el lenguaje.

Surge entonces como un problema central en la teoría de los lenguajes formales, el de determinar si una cadena α dada de símbolos terminales, puede o no puede ser generada por una gramática. En otras palabras, podemos preguntarnos si la cadena en cuestión está escrita de acuerdo con las reglas de la gramática (*¿está bien escrita?*), es decir:

$$\textbf{¿ } \alpha \in \textbf{L(G) ?}$$

La forma que conocemos hasta ahora de lidiar con este problema y responder la pregunta anterior es:

SI) Encontrar una derivación $S \rightarrow^* \alpha$ que, aplicando una cantidad finita de producciones de la gramática, logre transformar el axioma de la misma en la cadena de terminales bajo análisis, o

NO) Demostrar que tal derivación no existe.

Se llama **análisis sintáctico** de la cadena α, al proceso de búsqueda de esta derivación.

El análisis sintáctico de una cadena puede hacerse manualmente probando distintas derivaciones desde el axioma, aplicando sucesiva y alternativamente una u otra regla de reescritura, para determinar si alguna secuencia llega a generar la cadena en cuestión; si no se llega, se reinicia el proceso con otra secuencia de producciones y se sigue probando hasta agotar todas las posibilidades. Según el tamaño de la gramática (cantidad de símbolos terminales, símbolos no terminales y producciones) y el largo de la cadena bajo análisis, el proceso de análisis sintáctico llevado a cabo de esta forma puede ser corto e inclusive obvio, o extremadamente largo y tedioso, por lo que un abordaje por prueba y error suele ser inadecuado, salvo en ejemplos muy sencillos.

Árbol de derivación

Se necesitan entonces procedimientos claros y repetibles que, de manera sistemática, permitan responder si una cadena pertenece o no, a un lenguaje independiente del contexto. En ese camino, se define una forma pictórica de representar una derivación: ***el árbol de derivación*** o ***árbol de análisis sintáctico*** de la cadena.

Al derivar una cadena de un lenguaje generado por una gramática, siempre debe iniciarse el proceso partiendo del axioma, aplicando al mismo una producción que lo tenga como lado izquierdo.

Éste puede entonces ser reescrito, como una cadena solo de terminales, solo de no terminales, o de símbolos terminales y no terminales según sea el lado derecho de la producción: **S := $a_1a_2\ldots a_n$**.

Podemos graficar esto como un árbol con nodo raíz **S** y **n** nodos hijos ordenados de izquierda a derecha a_1, a_2, …, a_i, …, a_n (Fig. 2.2). Si el símbolo a_i fuera un no terminal, existirá alguna pro-

ducción en la gramática $a_i := b_1 b_2 \ldots b_m$ que lo tenga como lado izquierdo, con lo que al aplicarla y representar esto en el árbol, se generarían **m** hijos para este nodo a_i (Fig. 2.3).

Por otro lado, si a_i era un símbolo terminal, entonces no podrá tener hijos y será una hoja del árbol de análisis sintáctico. Cuando todos los nodos hoja correspondan a símbolos terminales, entonces la cadena generada por la derivación podrá leerse en las hojas del árbol de izquierda a derecha.

En resumen, el árbol de análisis sintáctico tendrá:

a) El axioma **S** de la gramática como raíz.

b) Símbolos no terminales de Σ_N como nodos internos.

c) Para el nodo interno del no terminal **A**, si $A:=a_1 a_2 \ldots a_k$ es la producción usada, se tendrán **k** nodos hijos etiquetados con los símbolos del lado derecho en el orden en el que aparecen.

d) Símbolos terminales de Σ_T como hojas.

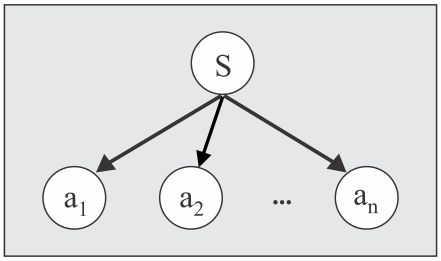


Figura 2.2: Árbol de derivación, paso 1.

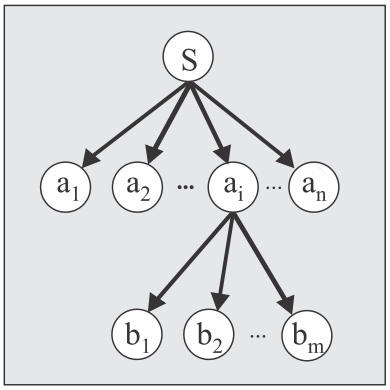


Figura 2.3: Árbol de derivación, paso 2.

Para cada derivación de una cadena en la gramática, se puede generar su correspondiente árbol de análisis sintáctico, por lo que el problema de análisis sintáctico se puede reformular diciendo que: *una cadena α de símbolos terminales pertenece al lenguaje* **L(G)** *generado por la gramática* **G,** *si y solo si, es posible construir su árbol de análisis sintáctico, con el axioma* **S** *como raíz y la cadena α leída en las hojas de izquierda a derecha.*

Ejemplo 2.29

Con la siguiente gramática como especificación:

$$G_{(29)} = (\{0, 1\}, \{S, P, Q\}, S, \{S := PQ \mid 0S1, P := 0Q \mid 1, Q := 1P \mid 0\})$$

se generan algunas cadenas por derivaciones desde el axioma (coloque usted el número de producción utilizado en cada caso sobre el símbolo de derivación), para luego mostrar sus respectivos árboles de derivación.

Derivación por la derecha: $S \rightarrow 0S1 \rightarrow 0PQ1 \rightarrow 0P01 \rightarrow 0101$

Derivación por la izquierda: $S \rightarrow 0S1 \rightarrow 0PQ1 \rightarrow 01Q1 \rightarrow 0101$

Notemos que tanto la derivación realizada por izquierda como la construida por derecha de la cadena **0101**, tienen el mismo árbol de derivación; por la derecha, primero se sustituye **P** por uno y luego **Q** por cero; por la izquierda primero se cambia **Q** y luego **P**. Estas diferencias no pueden verse en el árbol de análisis sintáctico.

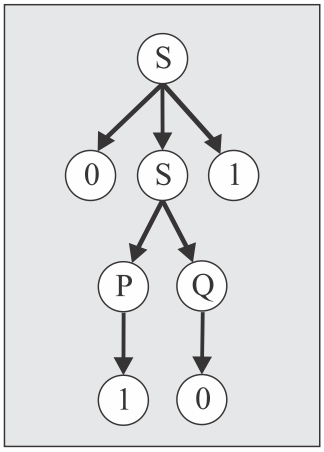


Figura 2.4: Árbol de la cadena 0101.

Construyamos ahora otra derivación, correspondiente a una subpalabra de 0101:

$$S \rightarrow PQ \rightarrow 1Q \rightarrow 10$$

Si observamos su árbol de derivación vemos que es un subárbol del anterior. Esto ocurre porque la cadena que representa en este caso es una subpalabra de la cadena anterior, pero no necesariamente ocurre esto siempre.

Hemos hecho una derivación por izquierda de la cadena **10** pero si hubiéramos derivado $S \rightarrow PQ \rightarrow P0 \rightarrow 10$, por la derecha, hubiésemos obtenido el mismo resultado que antes, tanto en la cadena como en el árbol resultante.

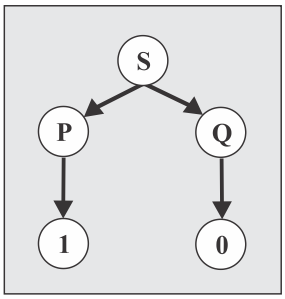


Figura 2.5: Árbol de la cadena 10.

Ejemplo 2.30

Suponga que se tiene un lenguaje de programación donde deben declararse las variables mediante un tipo (*entero* o *real*), un guión bajo y un nombre (*identificador* compuesto por una letra **a**, **b** o **c,** o una de esas letras y un dígito del **0** al **9**), terminando con un punto y coma. Una gramática simple para especificar esto podría ser:

$$G_{(30)} = (\Sigma_{T30}, \Sigma_{N30}, \text{<declaración>}, P_{30})$$

donde:

Σ_{T30} = { entero, real, a, b, c, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, ;, _ }

Σ_{N30} = { <declaración>, <tipo>, <identificador>, <letra>, <dígito> }

P_{30} = { <declaración> := <tipo> _ <identificador> ; ,

<tipo> := real | entero ,

<identificador> := <letra> | <letra> <dígito> ,

<letra> := a | b | c ,

<dígito> := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 }

Con esta gramática podemos derivar, por ejemplo, la siguiente forma correcta de definir una variable en este lenguaje:

<declaración> $\rightarrow$ <tipo> _ <identificador>; $\rightarrow$ entero_ <identificador>; $\rightarrow$

$\rightarrow$ entero_ <letra> <dígito>; $\rightarrow$ entero_a <dígito>; $\rightarrow$

$\rightarrow$ entero_a0;

Así, podemos escribir **<declaración> $\rightarrow$* entero_a0;** y ver que esta derivación de la cadena **entero_a0;** se ha efectuado por la izquierda, ya que cada vez que se tuvo alternativa de reemplazo de un no terminal, se eligió el de más a la izquierda en la forma sentencial. Por ejemplo, en el segundo paso de derivación, se puede optar por reescribir el no terminal ***<tipo>*** aplicando una de sus producciones, o se puede reescribir ***<identificador>*** según sus reglas de reescritura. En la derivación efectuada, se ha elegido el no terminal de más a la izquierda, es decir, ***<tipo>*** para reemplazarlo por el terminal ***entero***. El árbol de análisis sintáctico correspondiente a esta derivación, se muestra en la Figura 2.6. Nótese que la cadena de cinco símbolos terminales se puede leer en las hojas del árbol de izquierda a derecha.

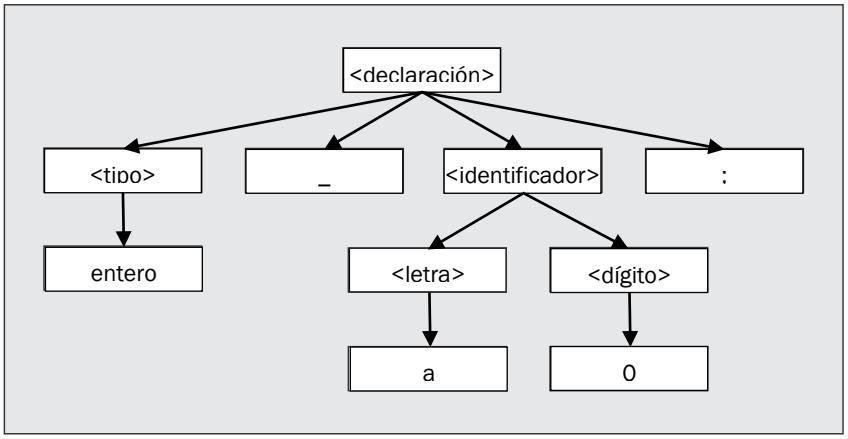


Figura 2.6: Árbol de derivación de la declaración ***entero_a0;***.

También se podría haber generado esta cadena operando con una derivación por la derecha:

<declaración> $\rightarrow$ <tipo>_<identificador>;$\rightarrow$ <tipo>_<letra><dígito>; $\rightarrow$

$\rightarrow$ <tipo>_<letra>0; $\rightarrow$ <tipo>_a0; $\rightarrow$ entero_a0;

Nuevamente, esta derivación, distinta de la anterior, tiene el mismo árbol de análisis sintáctico mostrado anteriormente.

Se han ideado interesantes algoritmos para construir estos árboles de derivación, por lo que la reformulación del problema de análisis sintáctico desde "*encontrar una derivación*" a *"construir el árbol*" ha resultado sumamente útil y práctica; además, una vez construido el árbol, las distintas asignaciones de significados y funciones que pueden hacerse a sus nodos y las distintas formas

de recorrerlos, ofrecen alternativas para la interpretación semántica de la cadena codificada por el árbol.

Pero un mismo árbol de análisis sintáctico podría representar más de una derivación. Existe cierta pérdida de información al generar el árbol, ya que no queda unívocamente identificado el orden de aplicación de las producciones. Un árbol no cambia si en un mismo nivel, primero se crean los hijos de $\mathbf{a_i}$ y luego los de $\mathbf{a_j}$, o si primero se crean los de $\mathbf{a_j}$ y luego los de $\mathbf{a_i}$. El que dos derivaciones distintas sean representadas por el mismo árbol, no crea en principio problema alguno.

Ambigüedad

Un caso enteramente distinto ocurre cuando la misma cadena α de símbolos terminales, puede ser generada por distintas derivaciones que además generan *distintos árboles de derivación*. Hablamos aquí de árboles *visiblemente distintos*, abstrayendo el contenido de sus nodos interiores (*de distinto dibujo*). Hay características de las cadenas de un lenguaje, y entre ellas destaca su *significado*, que dependen fuertemente del árbol de derivación de la misma. Véase el siguiente ejemplo antes de continuar:

Ejemplo 2.31

Sea por caso la siguiente pequeña gramática para las expresiones aritméticas simples:

$$G_{(31)} = (\ \{num, +, *, (,)\}, \{E\}, E, \{E := E + E \mid E * E \mid (E) \mid num\}\ )$$

Esta gramática muestra una simple definición recursiva de expresión algebraica, a saber: una expresión es un número o un par de expresiones separadas por los símbolos de operación **+** o *****, o una expresión entre paréntesis.

Para una expresión válida como **num+num*num**, sus derivaciones correctas con esta gramática podrían ser:

Derivación por izquierda:

$E \rightarrow E+E \rightarrow num+E \rightarrow num+E*E \rightarrow num+num*E \rightarrow num+num*num$

Derivación por derecha:

$E \rightarrow E+E \rightarrow E+E*E \rightarrow E+E*num \rightarrow E+num*num \rightarrow num+num*num$

Ambas derivaciones tendrán finalmente el mismo árbol de análisis sintáctico, aunque la secuencia en la que este árbol se construye puede verse que es bien distinta en ambos casos (Fig. 2.7).

Si se está derivando por *izquierda*, primero se debe expandir el nodo de expresión marcado con **1** y luego recién el marcado con **2**. Si se lo hace por la derecha, primero se debe expandir el nodo de expresión marcado con **2** y luego el nodo marcado con **1** (Fig. 2.7).

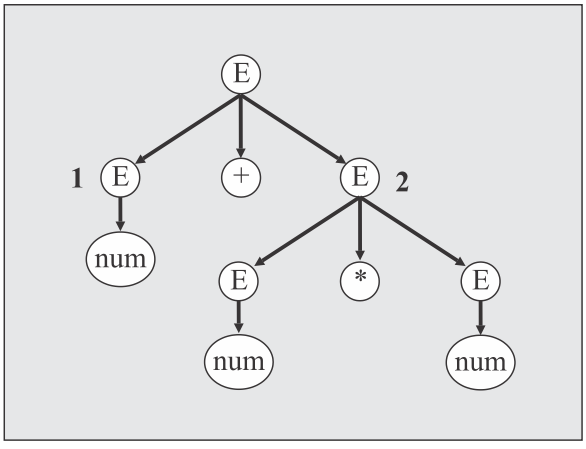


Figura 2.7: Árbol de la derivación por izquierda.

Sin embargo, considere esta otra derivación posible por izquierda y su correspondiente árbol de análisis sintáctico:

$$E \rightarrow E*E \rightarrow E+E*E \rightarrow num+E*E \rightarrow num+num*E \rightarrow num+num*num$$

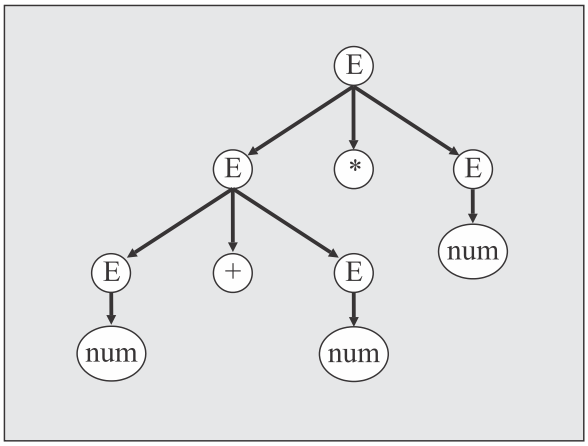


Figura 2.8: Árbol de la derivación alternativa.

Nótese que la cadena que se quería derivar, también se lee en sus hojas de izquierda a derecha, por lo que ha sido derivada correctamente pero el árbol es claramente distinto al anterior, a pesar de ser ambas derivaciones hechas en forma correcta por la izquierda (Fig. 2.8).

Las tres derivaciones aseguran que la cadena **num+num*num** pertenece al lenguaje generado por la gramática de expresiones $\mathbf{G_{(31)}}$. El problema no es sintáctico *sino semántico*, ya que cuando se quiera asignar significado a la expresión del primer árbol, posiblemente se hará de tal forma que signifique **num+(num*num)**, y al usar la del segundo **(num+num)*num**, ofreciendo resultados finales distintos de las expresiones cuando éstas deban ser evaluadas.

En este caso, y bajo el supuesto de tener una cadena más de un árbol de derivación, se produciría el problema de más de un significado para una misma frase, problema que no puede admitirse al trabajar en programación de computadoras, ya que se necesita que siempre las instrucciones dadas sean interpretadas unívocamente y no según qué árbol de derivación se haya creado durante la compilación.

Se dice que *una **cadena es ambigua** si puede ser generada por derivaciones que admiten distintos árboles de análisis sintáctico.*

Si una gramática permite esto, como lo hace la pequeña gramática de expresiones, también se dice que ella misma es ***ambigua*** y que *genera su lenguaje en forma ambigua*. Sin embargo, en muchos casos, la gramática puede ser modificada o reescrita completamente para obtener una gramática equivalente (que genera el mismo lenguaje) pero que no sea ambigua.

Existen algunos lenguajes independientes del contexto que solo pueden ser generados por gramáticas ambiguas. Se los denomina lenguajes ***inherentemente ambiguos***. Claramente, no son lenguajes que resulten útiles para la programación de computadoras (ver Ejercicio 40).

Pero para buscar una gramática equivalente no ambigua, primero hay que detectar que la gramática es ambigua y esto puede ser un trabajo largo, tedioso y hasta imposible; piénsese que una gramática puede generar infinitas cadenas y cada una de ellas obtenida con múltiples derivaciones, por lo que probar todas las alternativas no sería posible. En realidad, la cuestión es más de fondo; puede demostrarse que el problema es ***indecidible***, esto significa que *no es posible diseñar un algoritmo aplicable a cualquier gramática, que determine si es o no es ambigua*. En otras palabras, dada una gramática medianamente compleja capaz de generar infinitas cadenas, si en la búsqueda de ambigüedad el azar estuvo de nuestro lado y pudimos detectar una cadena ambigua, podremos afirmar con certeza que la gramática es ambigua; pero si nuestra búsqueda resultó infructuosa, no podremos afirmar con certeza que la gramática no es ambigua, solo que *parece no ambigua*.

Recursión

Como se anticipó, una producción de una gramática independiente del contexto **$G=(\Sigma_T, \Sigma_N, S, P)$** se dice que es ***recursiva*** si el no terminal de su lado izquierdo se encuentra también en el lado derecho:

$$A := \alpha A \beta$$

donde $\mathbf{A \in \Sigma_N}$ es un símbolo no terminal de la gramática y $\alpha, \beta \in (\Sigma_T \cup \Sigma_N)^*$ son cadenas de terminales y no terminales de cualquier largo.

Una gramática que tiene una regla de reescritura recursiva, se dice que tiene ***recursión en un paso***. En el caso en el que un no terminal del lado izquierdo de una producción pueda derivarse en una cadena que lo contenga, en varios pasos:

$$A \rightarrow \delta_1 \rightarrow \delta_2 \rightarrow \ldots \rightarrow \alpha A \beta$$

siendo $\mathbf{A \in \Sigma_N}$ el símbolo no terminal y $\alpha, \beta, \delta_i \in (\Sigma_T \cup \Sigma_N)^*$ cadenas de terminales y no terminales de cualquier largo, se dice que existe en la gramática ***recursión en más de un paso***.

La recursividad es una herramienta fundamental en las gramáticas porque permite trasladar la potencia de las definiciones recursivas a las mismas. Es la recursividad lo que posibilita describir un lenguaje de infinitas cadenas, con solo un número finito de producciones. Si una gramática no tiene recursión (en un paso o en más de un paso), solo podrá generar un número finito de cadenas.

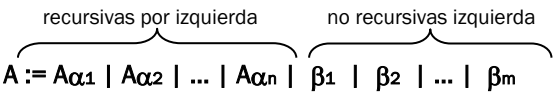
Si en la regla recursiva $\mathbf{A:=\alpha A \beta}$ la cadena α es vacía, esto es $\mathbf{A:=A\beta}$, se dice que la regla es ***recursiva por la izquierda*** y que la gramática tiene recursión izquierda. Si en cambio es β la cadena vacía, esto es $\mathbf{A:=\alpha A}$, se dice que la regla es ***recursiva por la derecha*** y que la gramática tiene recursión derecha.

Para algunos algoritmos de análisis sintáctico (análisis por descenso recursivo, LL(k) y otros), la recursión por izquierda suele ser fatal, pudiendo producir en el código de implementación recursiones funcionales infinitas o lazos iterativos sin fin, que los llevan a errores de ejecución y cancelación prematura del proceso de análisis. En estos casos, es menester eliminar la recursión izquierda modificando la gramática pero haciendo que siga generando el mismo lenguaje, esto es, encontrando una gramática equivalente a la dada que no tenga recursión izquierda.

Esto siempre puede hacerse apelando a los siguientes teoremas, que no demostraremos aquí.

Eliminación de recursión izquierda en un paso

Sea $\mathbf{G=(\Sigma_T, \Sigma_N, S, P)}$ una gramática independiente del contexto y $\mathbf{A \in \Sigma_N}$ un símbolo no terminal para el cual existen reglas de reescritura recursivas por la izquierda y algunas producciones no recursivas por izquierda:



con $\alpha_i, \beta_j \in (\Sigma_T \cup \Sigma_N)^+$ cadenas de terminales y no terminales de largo arbitrario. Entonces, siempre *se puede obtener una gramática equivalente sin recursión izquierda* haciendo lo siguiente:

a) Crear un nuevo símbolo no terminal **X** y agregarlo al alfabeto de símbolos no terminales: $\mathbf{\Sigma_N' = \Sigma_N \cup \{ X \}}$

b) Eliminar todas las producciones en **P** para el no terminal **A**.

c) Agregar al conjunto **P** las producciones:

$\mathbf{A := \beta_1 X \mid \beta_2 X \mid \ldots \mid \beta_m X \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m}$

$\mathbf{X := \alpha_1 X \mid \alpha_2 X \mid \ldots \mid \alpha_n X \mid \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n}$

La nueva gramática $\mathbf{G' = (\Sigma_T, \Sigma_N', S, P')}$ obtenida, es equivalente a la gramática **G** original y sus reglas ahora no son recursivas por izquierda (lo son solo por derecha en el símbolo **X**).

Ejemplo 2.32

La gramática para las expresiones aritméticas sencillas discutida anteriormente, tiene dos reglas recursivas por izquierda para el símbolo no terminal **E**:

$$G_{(31)} = (\ \{num, +, *, (,)\}, \{E\}, E, \{E := E + E \mid E * E \mid (E) \mid num\}\ )$$

Aplicando el anterior teorema, podemos determinar una gramática equivalente pero sin recursión izquierda. Para ello, definimos un nuevo no terminal **X** y haciendo α_1**=+E**, α_2**=*E**, β_1**=num** y β_2**=(E)**, obtenemos:

$$G_{(32)} = (\ \{num, +, *, (,)\}, \{E, X\}, E, P_{(32)}\ ), \text{ donde:}$$

$$P_{(32)} = \{E := numX \mid (E)X \mid num \mid (E),\ X := +EX \mid *EX \mid +E \mid *E\}\ )$$

Como muestra, analicemos ahora la misma cadena del Ejemplo 2.31 **num+num*num** utilizando $\mathbf{G_{(32)}}$, con una derivación por izquierda:

$$E \rightarrow numX \rightarrow num+EX \rightarrow num+numX \rightarrow num+num*E \rightarrow num+num*num$$

Eliminación de recursión izquierda en más de un paso

Dada la gramática $\mathbf{G = (\Sigma_T, \Sigma_N, S, P)}$ independiente del contexto con recursión izquierda en más de un paso, ***se obtendrá una gramática equivalente sin recursión izquierda*** haciendo lo siguiente:

a) Asignar un ***orden*** cualquiera a los símbolos no terminales, digamos $\mathbf{A_1, A_2, \ldots, A_k}$.

b) Para cada **i=1, 2, …, k**, hacer:

   a. Para cada **j=1, 2, …, k**, hacer:

      Si $\mathbf{i \neq j}$, reemplazar cada $\mathbf{A_i := A_j\delta}$ en **P** (eliminarla y agregar) por $\mathbf{A_i := \gamma_1\delta \mid \gamma_2\delta \mid \ldots \mid \gamma_h\delta}$ donde los γ_m son los lados derechos de todas las producciones de $\mathbf{A_j}$.

   b. Eliminar recursión izquierda en un paso de $\mathbf{A_i}$, si la hubiere, aplicando el anterior teorema.

Al terminar el proceso, las recursiones izquierdas habrán sido eliminadas y la gramática obtenida será equivalente a la original.

Factorización por izquierda

Otra situación que trae problemas a algunos de los analizadores sintácticos descendentes, que se introducirán más adelante, es el caso de dos o más producciones que inicien con una parte común:

$$A := \alpha\beta \quad y \quad A := \alpha\gamma$$

donde α, β y γ son cadenas cualesquiera de terminales y no terminales.

En este caso, el proceso de análisis sintáctico no tiene en claro cuál de las dos producciones utilizar durante una derivación, aun sabiendo que tiene que emparejar una parte de la cadena bajo análisis con α. Sin embargo, si se crea un nuevo no terminal **X** y se reemplazan las anteriores producciones por:

$$A := \alpha X \quad y \quad X := \beta \mid \gamma$$

el procedimiento no tiene ahora dudas de lo que debe utilizar: la primera producción primero, y recién al terminar de emparejar α con la cadena bajo análisis, deberá decidir con cuál de las producciones de **X** continúa el análisis de la cadena bajo estudio.

Ejemplo 2.33

En la mayoría de los lenguajes de programación, se dispone de alguna instrucción de bifurcación del tipo ***if … then … else***. Considere las siguientes producciones:

<condicional> := **if** <condición> **then** <sentencia>

<condicional> := **if** <condición> **then** <sentencia> **else** <sentencia>

como parte de la gramática de algún lenguaje, donde ***condicional, condición*** y ***sentencia*** son símbolos no terminales e ***if, then*** y ***else*** son símbolos terminales. Estas dos producciones pueden ser factorizadas por izquierda sin agregar ni quitar ninguna de las cadenas que generan, haciendo:

<condicional> := **if** <condición> **then** <sentencia> <elseOpcional>

<elseOpcional> := λ | **else** <sentencia>

Nótese que se creó un nuevo no terminal ***elseOpcional*** y que aparece una regla compresora al efectuar la factorización por izquierda.

Formas normales de gramáticas independientes del contexto

Las gramáticas independientes del contexto pueden ser expresadas en formas tales que, los lados derechos de sus producciones estén restringidos a formatos normalizados. Es decir que el lado izquierdo debe seguir siendo solo un no terminal, pero sus lados derechos estarán normados para obtener algunas características deseables.

Forma Normal de Chomsky (FNC)

Una gramática se dice que está en ***Forma Normal de Chomsky*** si y solo si, todas sus producciones tienen en el lado derecho dos símbolos no terminales, o un solo símbolo terminal o la cadena vacía (este último caso, solo si el axioma se encuentra del lado izquierdo):

$$A := BC \quad \text{ó} \quad A := a \quad \text{ó} \quad S := \lambda$$

donde $\mathbf{A, B, C, S \in \Sigma_N}$ son símbolos no terminales, **S** es el símbolo inicial de la gramática, y $\mathbf{a \in \Sigma_T}$ representa un símbolo terminal. Nótese que una gramática en Forma Normal de Chomsky siempre tendrá árboles de derivación binarios.

Cualquier gramática independiente del contexto $\mathbf{G=(\Sigma_T, \Sigma_N, S, P)}$, podrá ser transformada en una gramática equivalente en *Forma Normal de Chomsky,* mediante el siguiente procedimiento:

a) Transformar **G** en una gramática bien formada, esto es, limpia y sin reglas no generativas ni de redenominación.

b) Para cada símbolo terminal $\mathbf{a \in \Sigma_T}$, crear un nuevo símbolo no terminal $\mathbf{X_a}$ y una nueva producción $\mathbf{X_a := a}$. En símbolos:

$$\Sigma_N' = \Sigma_N \cup \{ X_a \} \qquad P' = P \cup \{ X_a := a \}$$

c) Para cada producción de la gramática que contenga en su lado derecho tanto símbolos no terminales como símbolos terminales, reemplazarla por una nueva que tenga en lugar del terminal **a** su correspondiente nuevo no terminal $\mathbf{X_a}$. Esto es:

$$A := \alpha a \beta \quad \text{es reemplazada por} \quad A := \alpha X_a \beta$$

cualesquiera sean α y β. Al terminar con estos pasos se tendrá una gramática equivalente con producciones que solo contienen en su lado derecho un solo símbolo terminal (es decir que está en Forma Normal de Chomsky) o una cadena de dos o más símbolos no terminales. Si tiene solo dos no terminales, entonces ya está en Forma Normal de Chomsky.

En caso contrario:

d) Para cada producción con más de dos símbolos no terminales en su lado derecho, digamos $\mathbf{A:=B\eta}$ donde η contiene dos o más no terminales, crear un nuevo símbolo no terminal **X** y reemplazar la producción por el par $\mathbf{A:=BX}$ y $\mathbf{X:=\eta}$.

Procediendo reiteradamente de esta forma y siendo el conjunto **P** finito, se llegará a tener todas las producciones en Forma Normal de Chomsky.

El procedimiento es sencillo y automatizable, por lo cual puede hacerse algorítmicamente.

Ejemplo 2.34

Determinar una gramática equivalente a la que sigue, pero que la misma se encuentre en Forma Normal de Chomsky:

$$G = (\{a, b, c\}, \{A,B,C\}, A, \{A := CBc \mid bB \mid \lambda,\ B := BC \mid b,\ C := c\})$$

Para ello, primero se la revisa y se determina que no tiene reglas innecesarias, el terminal **a** es un símbolo inaccesible, no tiene símbolos superfluos, ni reglas de redenominación, ni reglas no generativas (al quitar **a** del alfabeto de terminales queda ***bien formada***), por lo que:

a) Se elimina el símbolo terminal **a** del alfabeto de terminales.

b) Se crean los no terminales $\mathbf{X_b}$ y $\mathbf{X_c}$ y las producciones:

$$X_b := b \qquad\qquad X_c := c$$

c) Al estudiar ahora una por una las reglas de la gramática original, se determina que:

   a. **A:=CBc** no está en FNC. Primero, se la reemplaza por $\mathbf{A:=CBX_c}$ con lo cual se logra que solo tenga tres no terminales en el lado derecho. Sigue sin estar en FNC, por lo que se crea el nuevo no terminal **X** y se reemplaza la producción por el par **A:=CX** y $\mathbf{X:=BX_c}$, quedando ambas producciones en FNC.

   b. **A:=bB** no está en FNC, por lo cual debe ser reemplazada por $\mathbf{A:= X_bB}$ que ahora sí está en FNC.

   c. **A :=** λ es una regla lambda válida por ser A el axioma de la gramática; está así en FNC por lo cual queda sin cambios.

   d. Las últimas tres producciones **B:=BC, B:=b** y **C:=c** están en FNC por lo que no necesitan ningún proceso.

d) Finalmente, la nueva gramática G' equivalente a la dada será:

$$G' = (\{b, c\}, \{A,B,C,X, \mathbf{X_b}, \mathbf{X_c}\}, A, \{A := CX \mid \mathbf{X_b}B \mid \lambda,$$
$$B := BC \mid b,\ C := c,\ X := B\mathbf{X_c},\ \mathbf{X_b} := b,\ \mathbf{X_c} := c\})$$

Nótese que la producción $\mathbf{X_c:=c}$ es redundante en este caso porque ya existe un no terminal **C** que lo único que produce es **c**; por esto si se quiere puede reemplazarse el $\mathbf{X_c}$ por **C** en todas las producciones y entonces eliminar el símbolo $\mathbf{X_c}$ quedando:

$$G'' = (\{b,c\}, \{A,B,C,X, \mathbf{X_b}\}, A, \{A := CX \mid \mathbf{X_b}B \mid \lambda,$$
$$B := BC \mid b,\ C := c,\ X := BC,\ \mathbf{X_b} := b\})$$

El caso de $\mathbf{X_b:=b}$ es enteramente distinto, porque si bien el no terminal **B** produce **b**, también produce **BC** por lo cual no puede ser utilizado en $\mathbf{A:=X_bB}$ para reemplazar a $\mathbf{X_b}$, porque cambiaría el lenguaje generado.

Forma Normal de Greibach (FNG)

Una gramática independiente del contexto está en ***Forma Normal de Greibach***, si y solo si, todas sus producciones inician su lado derecho con un símbolo terminal al que le sigue, opcionalmente, una cadena de símbolos no terminales de cualquier largo. En símbolos, las producciones tienen una de las siguientes formas:

$$A := a\eta \qquad \text{ó} \qquad S:=\lambda$$

con $\mathbf{A, S \in \Sigma_N}$ símbolos no terminales, **S** símbolo inicial de la gramática, $\eta \in \Sigma_N^*$ cadena de símbolos no terminales de cualquier largo y $\mathbf{a \in \Sigma_T}$ un símbolo terminal. Nótese que η puede ser vacía.

Las gramáticas en Forma Normal de Greibach, son especialmente utilizadas con ciertos algoritmos de análisis sintáctico (ver Capítulo 5).

Toda gramática **G = (Σ_T, Σ_N, S, P)** independiente del contexto puede ser reescrita para obtener una gramática equivalente en FNG. Se sigue para ello el siguiente procedimiento:

a) Transformar **G** en una gramática bien formada, esto es, limpia y sin reglas no generativas ni de redenominación.

b) Quitar la recursividad izquierda de la gramática.

c) Asignar un orden cualquiera a los símbolos no terminales de la gramática, digamos **A_1, A_2, …, A_k**.

d) Separar las producciones del conjunto **P** en tres grupos:

   a. *Grupo 1*: todas las producciones que comiencen con un terminal (**$A:=a\alpha$** siendo $\alpha \in (\Sigma_T \cup \Sigma_N)^*$ una cadena de terminales y no terminales de cualquier largo) y, si existiere en la gramática **G**, la regla lambda **$S:=\lambda$**.

   b. *Grupo 2*: producciones **$A_i := A_j\alpha$** con $\alpha \in (\Sigma_T \cup \Sigma_N)^+$ y con el símbolo **A_i** anterior a **A_j** en el ordenamiento dado (**$i < j$**).

   c. *Grupo 3*: producciones **$A_i := A_j\alpha$** con $\alpha \in (\Sigma_T \cup \Sigma_N)^+$ y con el símbolo **A_i** posterior a **A_j** en el ordenamiento dado (**$i > j$**).

   El caso **$i = j$** no puede producirse porque se ha eliminado la recursión por izquierda anteriormente.

e) Para cada producción del tercer grupo **$A_i := A_j\alpha$**, iniciando por aquellas con el subíndice **i** más pequeño, reemplazarlas (eliminar y agregar) por **$A_i := \delta_1\alpha \mid \delta_2\alpha \mid \ldots \mid \delta_h\alpha$** donde los δ_l son los lados derechos de todas las producciones de **A_j**. Al terminar, este proceso, todas las producciones pertenecerán al grupo 1 o 2.

f) Repetir el proceso anterior para las producciones del segundo grupo. Al terminar, todas las producciones serán del grupo 1, por lo cual todas iniciarán con un símbolo terminal.

g) Para cada símbolo terminal **$a \in \Sigma_T$** que esté en el lado derecho de las producciones resultantes, pero no al inicio de las mismas, crear un nuevo símbolo no terminal **X_a** y una nueva producción **X_a:=a**. En símbolos:

$$\Sigma_N' = \Sigma_N \cup \{ X_a \} \qquad P' = P \cup \{ X_a := a \}$$

h) Para cada producción de la gramática que contenga en su lado derecho, luego del primer símbolo terminal, tanto símbolos no terminales como símbolos terminales, reemplazarla por una nueva que tenga en lugar del terminal no inicial **a** su correspondiente nuevo no terminal **X_a**. Esto es

$$A := x\alpha a\beta \quad \text{es reemplazada por} \quad A := x\alpha X_a\beta$$

siendo **x** el primer símbolo terminal del lado derecho, **a** otro terminal de la producción y cualesquiera α y β.

Al terminar el procedimiento, todas las producciones estarán en Forma Normal de Greibach.

Nuevamente, el procedimiento es sencillo y automatizable, por lo cual puede hacerse algorítmicamente.

Ejemplo 2.35

Utilizando la misma gramática de contexto libre convertida anteriormente a Forma Normal de Chomsky:

$$G = (\{b, c\}, \{A, B, C\}, A, \{A := CBc \mid bB \mid \lambda,\ B := BC \mid b,\ C := c\})$$

se determinará una equivalente pero en Forma Normal de Greibach.

Se verifica primero que no tiene reglas innecesarias, ni símbolos inaccesibles (se ha quitado **a** del anterior ejemplo), no tiene símbolos superfluos, ni reglas de redenominación, ni reglas no generativas (está bien formada), por lo que:

a) Nos preocupamos por la recursividad izquierda. El símbolo no terminal **B** tiene regla recursiva izquierda (**B:=BC**) y no recursiva (**B:=b**), por lo cual, utilizando el teorema de eliminación de recursividad izquierda en un paso, se crea el nuevo no terminal **X** y se la reemplaza por **B:=bX | b** y **X:=CX | C**. En este caso, queda una regla de redenominación **X:=C** que debe reemplazarse por **X:=c** para volver a dejar la gramática bien formada (nótese que **C** solo produce **c**).

La nueva gramática equivalente a la dada es ahora:

G'= ({b, c}, {A, B, C, X}, A,{A:=CBc|bB|λ, B:=bX|b, C:=c, X:=CX|c})

b) Tomando el orden lexicográfico para los no terminales, esto es A, B, C, X, se pueden separar las producciones en:

Grupo 1: A := bB | λ, B := bX | b, C := c, X := c

Grupo 2: A := CBc, ya que **A** antecede a **C** en el orden.

Grupo 3: X := CX, ya que **X** es posterior a **C** en el orden.

c) Se opera sobre la producción del grupo 3:

X := CX es reemplazada por X := cX.

Luego se trabaja sobre la producción del grupo 2:

A := CBc es reemplazada por A := cBc.

Quedando ahora todas las producciones en el grupo 1:

A := cBc | bB | λ, B := bX | b, C := c, X := cX | c

d) Como solo la primera producción del anterior grupo no está en Forma Normal de Greibach, solo se deberá crear el no terminal $\mathbf{X_c}$ y agregar la producción $\mathbf{X_c := c}$; como se comentó en el anterior ejemplo, ya que el no terminal **C** lo único que produce es el terminal **c**, no hace falta crear este nuevo no terminal, por lo que la primera producción es reemplazada por **A := cBC**.

e) Finalmente, la nueva gramática G' equivalente en **FNG** es:

G'' = ({b, c}, {A, B, C, X}, A, { A := cBC | bB | λ, B := bX | b, C := c,

X := cX | c})

Actividades prácticas

Ejercicios propuestos de gramática limpia

Para cada una de las siguientes gramáticas, obtener una gramática limpia equivalente (su definición formal) indicando: reglas innecesarias, símbolos inaccesibles terminales y no terminales, y símbolos superfluos, si los hubiera.

Ejercicio 12

$G_1 = (\{ 0, 1, 2, 3 \}, \{ S, A, B, C, D, E \}, S, P_1)$

$P_1 = \{ S:= 0A \mid 1B \mid 01, A:= A \mid 1B \mid 0, B:= 0C \mid 0E \mid 10, C:= 1, E:= 0E,$

$D:= 0A \mid 1B \mid 0 \}$

Ejercicio 13

$G_2 = (\{ 0, 1, 2 \}, \{ S, A, B, C \}, S, P_2)$

$P_2 = \{ S:= 0A \mid 1 \mid S, A:= 1B0 \mid 01, C:= 0 \mid 1B \mid 1 \mid C, B:= 1A \mid A0 \mid 1B \}$

Ejercicio 14

$G_3 = (\{ a, b, c \}, \{ S, A, B, C, D, E \}, S, P_3)$

$P_3 = \{ S:= aBb \mid \lambda, A:= bB \mid Ca, B:= bA \mid b \mid a \mid bE \mid B, C:= a \mid bB \mid aD,$

$D:= a, E:= aE \mid E \}$

Ejercicio 15

$G_4 = (\{ 0, 1, 2 \}, \{ Q, R, S, T \}, Q, P_4)$

$P_4 = \{ Q:= 1R0 \mid \lambda, R:= 0S1 \mid 0T \mid 1, T:= 0R \mid RT1, S:= 0 \}$

Ejercicios propuestos de gramática bien formada

Para cada una de las siguientes gramáticas, generar la gramática bien formada equivalente, eliminando si las hay, *reglas no generativas* y *reglas de redenominación* (recuerde que primero deben estar limpias).

Ejercicio 16

$G_1 = (\{ 0, 1 \}, \{ S, A, B \}, S, P_1)$

$P_1 = \{ S:= A\ B1 \mid \lambda, A:= BA \mid \lambda, B:= 0A \mid \lambda \}$

Ejercicio 17

$G_2 = (\{ 0,1, 2, 3 \}, \{ S, A, B, C, D \}, S, P_2)$

$P_2 = \{ S:= C0 \mid \lambda \mid D10, A:= 1C3, B:= B, C:= 1 \mid \lambda \mid 0, D:= 1D \}$

Ejercicio 18

$G_3 = (\{ a, b, c, d \}, \{ A, B, C, D \}, A, P_3)$

$P_3 = \{ A:= bBa, B:=bDa \mid aC \mid b \mid \lambda, C:= BB \mid A, D:= \lambda \mid a \mid b \}$

Ejercicios propuestos de eliminación de recursividad izquierda

Ejercicio 19

$G_1 = (\{a, b\}, \{P, Q, R\}, P, P_1)$

$P_1 = \{ P:= abP \mid aQ, Q:= a \mid bR, R:= Ra \mid b \}$

Ejercicio 20

$G_2 = (\{a, b, c\}, \{S, A, B, C, D\}, S, P_2)$

$P_2 = \{ S:= AB \mid c , A:= aC , B:= aD , C:= Ca \mid Cab \mid b , D:= b \}$

Ejercicio 21

$G_3 = (\{a, b, c\}, \{S, A, B\}, S, P_3)$

$P_3 = \{ S:= aAb, A:= aB \mid a \mid Ac, B:= c \}$

Ejercicio 22

$G_4 = (\{a, b\}, \{M, N, P\}, M, P_4)$

$P_4 = \{ M:= Ma \mid aP \mid b , N:= aP \mid a , P:= b \mid aN \mid Pb \}$

Ejercicio 23

$G_5 = (\{a, b\}, \{M, P\}, M, P_5)$

$P_5 = \{ M:= Pa \mid b, P:= Mb \mid b \}$

Ejercicios propuestos de formas normales

Ejercicio 24

A las cuatro gramáticas obtenidas de los ejercicios de gramática limpia, llevarlas a gramáticas bien formadas, expresarlas en FNC y derivar dos palabras con la gramática antes y después de haber aplicado la FNC.

Expresar las siguientes gramáticas en Forma Normal de Greibach:

Ejercicio 25

$G_1 = (\{0, 1\}, \{S, A, B, C\}, S, P_1)$

$P_1 = \{ S:= A0 \mid 1 \mid C1, A:= 0B \mid 1, B:= A0 \mid 0, C:= 1 \}$

Ejercicio 26

$G_2 = (\{0, 1\}, \{S, A, B, C, D\}, S, P_2)$

$P_2 = \{ S:= DA \mid 1, A:= B0 \mid 1, B:= B1 \mid 0, C:= 0, D:= C0 \mid 01 \}$

Ejercicio 27

$G_3 = (\{a, b, c\}, \{S, A, B, C, D, E, F\}, S, P_3)$

$P_3 = \{ S:= Aa \mid bB \mid bC, A:= Da \mid D, B:= Ba \mid b \mid a \mid B, C:= a \mid Db \mid bE ,$

$D:= a, E:= bF, F:= aF \}$

Ejercicios propuestos de expresiones regulares

Ejercicio 28

Construya ocho cadenas correspondientes a cada uno de los lenguajes regulares representados por:

a) $L_1 ((11+0)^*)$

b) $L_2 ((a+bb)^*+ab)$

Ejercicio 29

Determine una expresión regular para cada uno de los siguientes conjuntos de palabras:

a) Cadenas de bits que empiezan con **1** y terminan con **0** (números binarios pares).

b) Cadenas de bits que se expresen como la unidad seguida de ceros (potencias de dos escritas en sistema binario).

Ejercicios resueltos de gramática limpia

Determinar y definir formalmente, para cada una de las siguientes gramáticas, una gramática limpia equivalente, indicando reglas innecesarias, símbolos inaccesibles terminales y no terminales, y símbolos superfluos si los hubiera.

Ejercicio 42

$G_1 = (\{ 0, 1, 2, 3 \} , \{ A, B \} , A , P_1)$

$P_1 = \{ A:= 0B \mid 2, B:= 0A \mid 1 \mid B \}$

Solución:

Reglas innecesarias: **B := B**

Símbolos inaccesibles no terminales: no hay

Símbolos inaccesibles terminales: **3**

Símbolos superfluos: no hay

Gramática limpia: **$G_1 = (\{ 0, 1, 2 \}, \{ A, B \}, A, P_1)$**

$P_1 = \{ A:= 0B \mid 2, B:= 0A \mid 1 \}$

Ejercicio 43

$G_2 = (\{ 0, 1 \} , \{ S, A, B, C \} , S , P_2)$

$P_2 = \{ S:= 0A1 \mid 0, A:= 0A1 \mid 0B \mid 0 \mid A, B:= 0B \mid 0, C:= 1C \}$

Solución:

Reglas innecesarias: **A:= A**

Símbolos inaccesibles no terminal: **C**

Símbolos inaccesibles terminales: no hay

Símbolos superfluos: no hay (ya que **C** es eliminado por inaccesible)

Gramática limpia: **$G_2 = (\{0, 1\}, \{S, A, B\}, S, P_2)$**

$P_2 = \{ S:= 0A1 \mid 0, A:= 0A1 \mid 0B \mid 0, B:= 0B \mid 0 \}$

Ejercicio 44

$G_3 = (\{ 0, 1 \} , \{ S, A, B, C \} , S , P_3)$

$P_3 = \{ S:= 0A \mid 1B \mid S \mid 0C, A:= 0A \mid 1S \mid 1, B:= 1BB \mid 0S \mid 0 \mid 1C, C:= 0C \}$

Solución:

Reglas innecesarias: **S := S**

Símbolos inaccesibles terminal: no hay

Símbolos inaccesibles no terminales: no hay

Símbolos superfluos: **C**

Gramática limpia: **$G_3 = (\{ 0, 1 \}, \{S, A, B\}, S, P_3)$**

$P_3 = \{ S:= 0A \mid 1B, A:= 0A \mid 1S \mid 1, B:= 1BB \mid 0S \mid 0 \}$

Ejercicio 45

$G_4 = (\{ a, b, c \} , \{ S, A, B, C \} , S , P_4)$

$P_4 = \{ S:= aBc \mid aAc , A: = a \mid Cc \mid A, B: = b \mid a \mid B, C:= Cc \}$

Solución:

Reglas innecesarias: **A := A** , **B := B**

Símbolos inaccesibles terminal: no hay

Símbolos inaccesibles no terminales: no hay

Símbolos superfluos: **C**

Gramática limpia: **G_4 = ({ a, b, c }, {S, A, B}, S, P_4)**

P_4 = { S:= aBc | aAc, A:= a, B:= b | a }

Ejercicio 46

G_5 = ({ 0, 1 } , { S, B, C } , S , P_5)

P_5 = { S:= CB | BC | 0C1 , B:= 0B1 | 0 | 1 | B, C:= 0C1 | 0 | C }

Solución:

Reglas innecesarias: **B:= B** , **C:= C**

Símbolos inaccesibles terminales: no hay

Símbolos inaccesibles no terminales: no hay

Símbolos superfluos: no hay

Gramática limpia: **G_5 = ({ 0, 1 }, {S, B, C}, S, P_5)**

P_5 = { S:= CB| BC| 0C1, B:= 0B1| 0| 1, C:= 0C1| 0 }

Ejercicios resueltos de gramática bien formada

Para cada una de las siguientes gramáticas, generar la gramática bien formada equivalente, indicando si las hubiera, *reglas no generativas* y *reglas de redenominación.*

Ejercicio 47

G_1 = ({a, b, z}, {S, M, N, P}, S, P_1)

P_1 = { S:= zMNz, M:= λ | aMa, N:= λ | bNb | z, P:= AM | zNP | P }

Solución:

Primero, debemos limpiar la gramática:

Reglas innecesarias: **P:= P**

Símbolos inaccesibles terminales: no hay

Símbolos inaccesibles no terminales: **P**

Símbolos superfluos: no hay

Gramática limpia: **G_{1L} = ({a, b, z}, {S, M, N}, S, P_{1L})**

P_{1L} = { S:= zMNz, M:= λ | aMa, N:= λ | bNb | z }

Reglas no generativas:

1. Eliminación de la regla **M:= λ**

   P_{1A} = { S:= zMNz | zNz, M:=aMa | aa, N:= λ | bNb | z }

2. Eliminación de la regla **N:= λ**

   P_{1B} = {S:= zMNz | zNz | zMz | zz, M:=aMa | aa, N:= bNb | bb | z}

Reglas de redenominación: no hay

Gramática bien formada: **G_{1B} = ({a, b, z}, {S, M, N}, S, P_{1B})**

P_{1B} = { S:= zMNz | zNz | zMz | zz, M:=aMa | aa, N:= bNb | bb | z }

Ejercicio 48

$G_2 = (\{x, y, z\}, \{S, A, B\}, S, P_2)$

$P_2 = \{ S:= xAx \mid \lambda, A:=xAx \mid yB, B:=yB \mid y \mid AB \mid \lambda \mid B \}$

Solución:

Primero, debemos limpiar la gramática:

Reglas innecesarias: **B:= B**

Símbolos inaccesibles terminales: **z**

Símbolos inaccesibles no terminales: no hay

Símbolos superfluos: no hay

Gramática limpia: $\mathbf{G_{2L} = (\{x, y\}, \{S, A, B\}, S, P_{2L})}$

$$\mathbf{P_{2L} = \{ S:= xAx \mid \lambda, A:=xAx \mid yB, B:=yB \mid y \mid AB \mid \lambda \}}$$

Reglas no generativas:

Eliminación de la regla **B:= λ**

$P_{2A} = \{ S:= xAx \mid \lambda, A:= xAx \mid yB \mid y, B:=yB \mid y \mid AB \mid A \}$

Reglas de redenominación:

Eliminación de la regla **B:=A**

$P_{2B} = \{ S:= xAx \mid \lambda, A:= xAx \mid yB \mid y, B:= \underline{yB} \mid y \mid AB \mid xAx \mid \underline{yB} \}$

Gramática bien formada: $\mathbf{G_{2B} = (\{x, y\}, \{S, A, B\}, S, P_{2B})}$

$\mathbf{P_{2B} = \{ S:= xAx \mid \lambda, A:=xAx \mid yB \mid y, B:= yB \mid y \mid AB \mid xAx \}}$

Ejercicio 49

$G_3 = (\{a, b, c, d\}, \{S, B, C, D\}, S, P_3)$

$P_3 = \{ S:= aB, B:= \lambda \mid aBB \mid b \mid B, C:=aC, D:= a \}$

Solución:

Primero debemos limpiar la gramática:

Reglas innecesarias: **B:= B**

Símbolos inaccesibles terminales: **c, d**

Símbolos inaccesibles no terminales: **C, D**

Símbolos superfluos: no hay

Gramática limpia: $\mathbf{G_{3L} = (\{a, b\}, \{S, B\}, S, P_{3L})}$

$$\mathbf{P_{3L} = \{ S := aB, B := \lambda \mid aBB \mid b \}}$$

Reglas no generativas:

Eliminación de la regla **B:= λ**

$P_{3B} = \{ S := aB \mid a, B := aBB \mid aB \mid b \mid a \}$

Reglas de redenominación: no hay

Gramática bien formada: $\mathbf{G_{3B} = (\{a, b\}, \{S, B\}, S, P_{3B})}$

$$\mathbf{P_{3B} = \{ S:= aB \mid a, B:=aBB \mid aB \mid b \mid a \}}$$

Ejercicios resueltos de eliminación de recursividad por izquierda

Para cada una de las siguientes gramáticas, generar una gramática equivalente con reglas de producción que no presenten recursividad por izquierda.

Ejercicio 50

G_1= ({a, b}, {S, A, B}, S, P_1)

P_1= { S:= Aa | bB, A:= aA | Ab | b, B:= Ba | b }

Solución:

Existe recursividad por izquierda en los símbolos **A** y B. Entonces, se crean los nuevos no terminales **X** y **Y**:

Reglas A:= Ab | aA | b se deben reemplazan por:

X:= b | bX

A:= aA | aAX | b | bX

Reglas B:= Ba | b se deben reemplazar por:

Y:= a | aY

B:= b | bY

G_1' = ({a, b}, (S, A, B, X, Y}, S, P_1')

P_1' = { S:= Aa | **bB, A:= aA** | **aAX** | **b** | **bX, B:= b** | **bY, X:= b** | **bX,**

Y:= a | **aY }**

Ejercicio 51

G_2= ({0, 1, 2}, {A, B, C, D}, A, P_2)

P_2= { A:= 1B | 0D, B:= B1 | 1C, C:= 0 | 1 | C0, D:= 0 }

Solución:

Existe recursividad por izquierda en los símbolos **B** y **C**. Entonces, se crean los nuevos no terminales **X** y **Y**:

Reglas B:= B1 | 1C se deben reemplazar por:

X:= 1 | 1X

B:= 1C | 1CX

Reglas C:= 0 | 1 | C0 se deben reemplazar por:

Y:= 0 | 0Y

C:= 1 | 1Y | 0 | 0Y

G_2' = ({0, 1}, (A, B, C, D, X, Y}, A, P_2')

P_2' = { A:= 1B | **0D, B:= 1C** | **1CX, C:= 0** | **0Y** | **1** | **1Y, D:= 0,**

X:= 1 | **1X, Y:= 0** | **0Y }**

Consideremos las cadenas 00, 111 y 1101. Obtengamos derivaciones de ellas según las gramáticas G_2 y G_2':

- Con **G_2:**

A $\rightarrow$ 0D $\rightarrow$ 00

A $\rightarrow$ 1B $\rightarrow$ 11C $\rightarrow$ 111

A $\rightarrow$ 1B $\rightarrow$ 1B1 $\rightarrow$ 11C1 $\rightarrow$ 1101

- Con **G_2':**

A $\rightarrow$ 0D $\rightarrow$ 00

A $\rightarrow$ 1B $\rightarrow$ 11C $\rightarrow$ 111

A $\rightarrow$ 1B $\rightarrow$ 11CX $\rightarrow$ 110X $\rightarrow$ 1101

Si bien ésta no es una demostración suficiente de equivalencia entre las gramáticas G_2 y G_2', resulta una condición necesaria. (Para que sea suficiente, habría que hacer la comprobación sobre todas las cadenas posibles del lenguaje).

Ejercicios resueltos de formas normales

A las cuatro gramáticas obtenidas de los ejercicios de gramática limpia, llevarlas a la Forma Normal de Chomsky y derivar dos palabras con la gramática antes y después de haber convertido a FNC.

Ejercicio 52

Solución:

La gramática se encuentra *bien formada,* no tiene reglas de redenominación ni reglas no generativas.

Se crea el nuevo no terminal **X** y se agrega la producción **X:= 0**.

Luego, se reemplaza **A:= 0B** por **A:= XB** y **B:= 0A** por **B:= XA**.

FNC: **G_1 = ({ 0, 1, 2 }, { A, B, X }, A, P_1)**

P_1 = { A:= XB | 2, B:= XA | 1, X:= 0 }

Derivaciones:

Antes: $A \rightarrow 0B \rightarrow 00A \rightarrow$ **002**

$A \rightarrow$ **2**

Después: $A \rightarrow XB \rightarrow 0B \rightarrow 0XA \rightarrow 00A \rightarrow$ **002**

$A \rightarrow$ **2**

Ejercicio 53

Solución:

La gramática se encuentra *bien formada,* no tiene reglas de redenominación ni reglas no generativas.

Se crean nuevos no terminales **X**, **Y** y **Z**, y se agregan las producciones **X:=0**, **Z:= 1** y **Y:=A1**. Luego, se efectúan los siguientes reemplazos:

Y:= A1 por Y:= AZ

S:= 0A1 por S:= XY

A:= 0A1 por A:= XY

A:= 0B por A:= XB

B:= 0B por B:= XB

FNC: **G_2 = ({ 0, 1 }, {S, A, B, X, Y, Z}, S, P_2)**

P_2 = {S:=XY|0, Y:=AZ, Z:=1, A:=XY|XB|0, B:=XB|0, X:=0}

Derivaciones:

Antes: $S \rightarrow 0A1 \rightarrow 00B1 \rightarrow$ **0001**

$S \rightarrow 0A1 \rightarrow$ **001**

Después: $S \rightarrow XY \rightarrow 0Y \rightarrow 0AZ \rightarrow 0XBZ \rightarrow 00BZ \rightarrow 000Z \rightarrow$ **0001**

$S \rightarrow XY \rightarrow 0Y \rightarrow 0AZ \rightarrow 00Z \rightarrow$ **001**

Ejercicio 54

Solución:

La gramática se encuentra *bien formada,* no tiene reglas de redenominación ni reglas no generativas.

Se crean nuevos no terminales **X**, **Y** y **Z**, y se agregan las producciones **X:=0** e **Y:=1**. Luego, se efectúan los siguientes reemplazos:

S:= 0A por S:= XA

S:= 1B por S:= YB

A:= 0A por A:= XA

A:= 1S por A:= YS

B:= 1BB por B:= ZB y Z:= 1B

Z:= 1B por Z:= YB

B:= 0S por B:= XS

FNC: **G_3 = ({ 0, 1 }, { S, A, B, X, Y, Z, }, S, P_3)**

P_3 = { S:=XA|YB, A:=XA|YS|1, B:=ZB|XS|0, X:=0, Y:=1, Z:=YB }

Derivaciones:

Antes: $S \rightarrow 0A \rightarrow 00A \rightarrow$ **001**

$S \rightarrow 1B \rightarrow 11BB \rightarrow 110B \rightarrow$ **1100**

Después: $S \rightarrow XA \rightarrow 0A \rightarrow 0XA \rightarrow 00A \rightarrow$ **001**

$S \rightarrow YB \rightarrow 1B \rightarrow 1ZB \rightarrow 1YBB \rightarrow 11BB \rightarrow 110B \rightarrow$ **1100**

Ejercicio 55

Solución:

La gramática se encuentra *bien formada,* no tiene reglas de redenominación ni reglas no generativas.

Se crean los nuevos símbolos no terminales **Y**, **Z** y **W**, y se agrega la producción **Z:=c**. Luego, se efectúan los siguientes reemplazos:

S:= aBc por S:= AY y Y:= Bc

Y:= Bc por Y:= BZ

S:= aAc por S:= AW y W:= Ac

W:= Ac por W:= AZ

FNC: **G_4 = ({a, b, c}, {A, B, S, Y, W, Z}, S, P_4)**

P_4 = { S:= AY|AW, Z:=c, A:= a, B:= b|a, Y:= BZ, W:= AZ }

Derivaciones:

Antes: $S \rightarrow aBc \rightarrow$ **abc**

$S \rightarrow aAc \rightarrow$ **aac**

Después: $S \rightarrow AY \rightarrow aY \rightarrow aBZ \rightarrow abZ \rightarrow$ **abc**

$S \rightarrow AW \rightarrow aW \rightarrow aAZ \rightarrow aaZ \rightarrow$ **aac**

Ejercicio 56

Solución:

La gramática se encuentra *bien formada,* no tiene reglas de redenominación ni reglas no generativas.

Se crean los nuevos símbolos no terminales **X**, **Y**, **Z**, **W** y se agregan las producciones **X:=0** y **Z:=1**. Luego, se efectúan los siguientes reemplazos:

S:= 0C1 por S:= XY y Y:=C1

Y:= C1 por Y:= CZ

B:= 0B1 por B:= XW y W:=B1

W:= B1 por W:= BZ

C:= 0C1 por C:= XY

FNC: $\mathbf{G_5 = (\{0, 1\}, \{B, C, S, X, Y, W, Z\}, S, P_5)}$

$\mathbf{P_5 = \{ S:= XY \mid BC \mid CB, B:= XW \mid 0 \mid 1, C:= XY \mid 0,}$

Y:= CZ, W:= BZ, Z:= 1, X:= 0 }

Convertir las siguientes gramáticas independientes del contexto a la Forma Normal de Greibach (FNG):

Ejercicio 57

$G_1 = (\{0, 1, 2\}, \{A, B, C\}, A, P_1)$

$P_1 = \{ A := CB \mid 2, B := A1 \mid 1, C := 0 \mid C1 \}$

Solución:

a) La gramática está bien formada ya que está sin reglas innecesarias, símbolos inaccesibles terminales y no terminales, símbolos superfluos, reglas de redenominación y reglas no generativas.

b) Existe recursión por izquierda en el símbolo **C**, por lo cual se crea un nuevo no terminal **X** y se reemplazan las producciones de **C** por: **X:=1|1X** y **C:=0|0X**. La gramática resultante es:

$G'_1 = (\{0, 1, 2 \}, \{A, B, C, X\}, A, P'_1)$

$P'_1 = \{ A := CB \mid 2,\ B :=A1 \mid 1, C :=0 \mid 0X , X := 1 \mid 1X \}$

c) Se establece el orden lexicográfico A, B, C, X para los no terminales y se separan en grupos las producciones:

Grupo 3: **B := A1**

Se reemplaza por **B:= CB1** (ahora del grupo 2) y por **B:= 21** (ahora del grupo 1) que a su vez se reemplaza por **B := 2Y** (ahora en FNG), donde **Y** es un nuevo no terminal que produce únicamente uno: **Y := 1**.

Grupo 2: **B := CB1** y **A := CB**

La primera se reemplaza por **B := 0B1** y **B := 0XB1** (ahora del grupo 1) y, haciendo uso del no terminal **Y** creado en el paso anterior, éstas a su vez se transforman en **B := 0BY** y **B := 0XBY** (ahora en FNG). La segunda se reemplaza por **A := 0B** y **A := 0XB**, quedando ambas en FNG.

d) Como todas las otras producciones del grupo 1 ya estaban en FNG, la gramática resultante es:

$\mathbf{G''_1 = (\{0, 1, 2 \}, (A, B, C, X, Y\}, A, P''_1)}$

$\mathbf{P''_1 = \{ A := 0B \mid 0XB \mid 2, B := 0BY \mid 0XBY \mid 1, C := 0 \mid 0X,}$

X := 1 | 1X, Y := 1 }

Ejercicio 58

G_2 = ({0, 1 }, {S, A, B}, S, P_2)

P_2 = { S := 0A1 | 1, A := 0A1 | B0 | 0, B := B0 | 0 | 1 }

Solución:

a) La gramática está bien formada ya que está sin reglas innecesarias, símbolos inaccesibles terminales y no terminales, símbolos superfluos, reglas de redenominación y reglas no generativas.

b) Existe recursión por izquierda en el símbolo **B**, por lo cual se crea un nuevo no terminal **X** y se reemplazan las producciones de **B** por: **X:=0|0X** y **B:=0|0X|1|1X**. La gramática resultante es:

G'_2 = ({0, 1}, {S, A, B, X}, S, P'_2)

P'_2 = { S := 0A1 | 1, A := 0A1 | B0 | 0, B := 0 | 0X | 1 | 1X,

X := 0 | 0X }

c) Se establece el orden S, A, B, X para los no terminales y se separan en grupos las producciones, haciendo entonces los reemplazos adecuados:

Grupo 3: **no hay**

Grupo 2: **A:= B0** $\Rightarrow$ A:= 0X0 $\Rightarrow$ A:= 0XY con Y:= 0

y A:= 00 $\Rightarrow$ A:= 0Y

y A:= 10 $\Rightarrow$ A:= 1Y

y A:= 1X0 $\Rightarrow$ A:= 1XY

Grupo 1: **S:= 0A1** $\Rightarrow$ S:= 0AZ con Z:= 1

A:= 0A1 $\Rightarrow$ A:= 0AZ

d) Como todas las otras producciones del grupo 1 ya estaban en FNG, la gramática resultante es:

G''_2 = ({0, 1}, {S, A, B, X, Y, Z}, S, P''_2)

P''_2 = { S := 0AZ | 1, A := 0AZ | 0XY | 0Y | 1Y | 1XY | 0,

B := 0 | 0X | 1 | 1X, X := 0 | 0X, Y := 0, Z := 1 }

Ejercicio 59

G_3 = ({0, 1}, {S, A, B}, S, P_3)

P_3 = { S := A0 | 1B, A := 0A | 1, B := 1B0 | A0 | 1 }

Solución:

a) La gramática está bien formada ya que está sin reglas innecesarias, símbolos inaccesibles terminales y no terminales, símbolos superfluos, reglas de redenominación y reglas no generativas.

b) No existe recursión por izquierda.

c) Se establece el orden S, A, B para los no terminales y se separan en grupos las producciones, haciendo entonces los reemplazos adecuados:

Grupo 3: **B:= A0** $\Rightarrow$ B:= 0A0 $\Rightarrow$ B:= 0AY con Y:= 0

y B:= 10 $\Rightarrow$ B:= 1Y

Grupo 2: **S:= A0** $\Rightarrow$ S:= 0A0 $\Rightarrow$ S:= 0AY

y S:= 10 $\Rightarrow$ S:= 1Y

Grupo 1: **B:= 1B0** $\Rightarrow$ B:= 1BY

d) Como todas las otras producciones del grupo 1 ya estaban en FNG, la gramática resultante es:

G'_3 = ({0, 1}, {S, A, B, Y}, S, P'_3)

P'_3 = { S := 0AY | 1Y | 1B, A := 0A | 1,

B :=1BY | 0AY | 1Y | 1, Y:= 0 }

Ejercicio 60

G_4 = ({0, 1}, {S, B, C}, S, P_4)

P_4 = { S := CB | BC | 0C1, B := 0B | 0 | 1, C := 0C | 0 }

Solución:

a) La gramática está bien formada ya que está sin reglas innecesarias, símbolos inaccesibles terminales y no terminales, símbolos superfluos, reglas de redenominación y reglas no generativas.

b) No existe recursión por izquierda.

c) Se establece el orden **S**, **B**, **C** para los no terminales y se separan en grupos las producciones, haciendo entonces los reemplazos adecuados:

Grupo 3: **no hay**

Grupo 2: **S:= BC** $\Rightarrow$ S:= 0BC

y S:= 0C

y S:= 1C

S:= CB $\Rightarrow$ S:= 0CB

y S:= 0B

S:= 0C1 $\Rightarrow$ S:= 0CX con X:= 1

d) Como todas las otras producciones del grupo 1 ya estaban en FNG, la gramática resultante es:

$G'_4 = (\{0, 1\}, \{S, B, C, X\}, S, P'_4)$

$P'_4 = \{ S := 0BC \mid 0C \mid 1C \mid 0CB \mid 0B \mid 0CX, B := 0B \mid 0 \mid 1,$

$C := 0C \mid 0, X := 1 \}$

Ejercicios resueltos de expresiones regulares

Ejercicio 61

Construya ocho cadenas correspondientes a cada uno de los lenguajes regulares representados por:

a) $(1+0)1^*$

b) $ab^*(ab)^*$

Solución:

Se procederá aplicando la definición de *expresión regular*, transformándola en una expresión con operaciones de conjuntos.

a) $L((1+0)1^*) = L((1+0)) \circ L(1^*)$

$= L(1+0) \circ L(1^*)$

$= (L(1) \cup L(0)) \circ (L(1))^*$

$= (\{1\} \cup \{0\}) \circ \{1\}^*$

$= \{0, 1\} \circ \{\lambda, 1, 11, 111, \ldots\}$

$= \{0, 1, 01, 11, 011, 111, 0111, 1111, ...\}$

b) $L(ab^*(ab)^*) = L(ab^*) \circ L((ab)^*)$

$= (L(a) \circ L(b^*)) \circ (L((ab)))^*$

$= (\{a\} \circ (L(b))^*) \circ (L(ab))^*$

$= (\{a\} \circ \{b\}^*) \circ (L(a) \circ L(b))^*$

$= (\{a\} \circ \{\lambda, b, bb, bbb, \ldots\}) \circ (\{a\} \circ \{b\})^*$

$= \{a, ab, abb, abbb, \ldots\} \circ \{ab\}^*$

$= \{a, ab, abb, abbb, \ldots\} \circ \{\lambda, ab, abab, ababab, \ldots\}$

$= \{a, aab, aabab, aababab, \ldots,$

ab, abab, ababab, abababab, …,

abb, abbab, abbabab, abbababab, ..

abbb, abbbab, abbbabab, abbbababab, …}

Ejercicio 62

Determine una expresión regular para cada uno de los siguientes conjuntos de palabras:

a) Cadenas de bits que empiezan con **1** y terminan con **1** (números binarios impares).

b) Cadenas de bits que empiezan con **1** y terminan en doble **0** (números binarios múltiplos de cuatro).

Solución:

a) El conjunto de todas las cadenas posibles de bits **0** y **1**, inclusive la cadena vacía, se representa por la expresión regular **(0+1)***. Para que inicien y terminen con uno, debemos concatenar antes y después de estas cadenas un **1**, por lo que la expresión regular resultante será: **1(0+1)*****1**

b) De igual forma, para lograr el segundo requerimiento, deben anteponerse un **1** y posponerse un **00** a la expresión de todas las cadenas de bits, quedando como resultado: **1(0+1)*****00**