

Capítulo 5

Lenguajes y gramáticas independientes del contexto

Como se ha visto, los autómatas son dispositivos que leen cadenas de entrada, símbolo a símbolo. Hemos estudiado los modelos básicos AFD, AFN y AFN- λ , y los autómatas con pila AFPD y AFPN. En capítulos posteriores consideraremos modelos de autómatas con mayor poder computacional. En el presente capítulo estudiaremos una noción completamente diferente, aunque relacionada, la de gramática generativa, que es un mecanismo para generar cadenas a partir de un símbolo inicial.

- ☞ Los autómatas *leen* cadenas
- ☞ Las gramáticas *generan* cadenas

5.1. Gramáticas generativas

Las gramáticas generativas fueron introducidas por Noam Chomsky en 1956 como un modelo para la descripción de los lenguajes naturales (español, inglés, etc). Chomsky clasificó las gramáticas en cuatro tipos: 0, 1, 2 y 3. Las gramáticas de tipo 2, también llamadas gramáticas independientes del contexto, se comenzaron a usar en la década de los sesenta del siglo XX para presentar la sintaxis de lenguajes de programación y para el diseño de analizadores sintácticos en compiladores.

Una *gramática generativa* es una cuádrupla, $G = (V, \Sigma, S, P)$ formada por dos alfabetos disyuntos V (alfabeto de *variables* o *no-terminales*) y Σ (alfabeto de *terminales*), una variable especial $S \in V$ (llamada *símbolo inicial*) y un conjunto finito P de *producciones* o *reglas de re-escritura*; $P \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$. Una producción $(\alpha, \beta) \in P$ se denota por $\alpha \rightarrow \beta$ y se lee “ α produce β ”; α se denomina la *cabeza* y β el *cuerpo* de la producción.



— — a — a — — — — ¹²⁷ — a

Quiz # 3

5.2 - 5.5

5.11 5.12

Jueve 27

Feb

1 hr

Introducción a la Teoría de la Computación.

Capítulo 5

128

11

El significado de la producción $\alpha \rightarrow \beta$ es: la cadena α se puede reemplazar (sobre-escribir) por la cadena β . Comenzando con el símbolo inicial S y aplicando las producciones de la gramática, en uno o más pasos, se obtienen cadenas que pueden tener tanto terminales como no-terminales. Aquellas cadenas que sólo tengan terminales conforman lo que se denomina el *lenguaje generado por G* .

Las gramáticas se clasifican de acuerdo con el tipo de sus producciones:

Parcial #3 Maquinas de

Turing Jueves

6 de marzo

Gramáticas de tipo 0. Sus producciones no tienen restricciones. También se llaman *gramáticas no-restringidas* o *irrestringidas*, o *gramáticas con estructura de frase* en razón de su origen lingüístico.

Gramáticas de tipo 1. Las producciones son de la forma $\alpha A \beta \rightarrow \alpha \gamma \beta$, donde A es una variable, $\alpha, \beta, \gamma \in (V \cup \Sigma)^*$ y $\gamma \neq \lambda$. También se llaman *gramáticas sensibles al contexto* o *gramáticas contextuales*.

Gramáticas de tipo 2. Las producciones son de la forma $A \rightarrow \beta$ donde A es una variable y $\beta \in (V \cup \Sigma)^*$. También se llaman *gramáticas independientes del contexto* o *gramáticas no-contextuales*.

Gramáticas de tipo 3. Las producciones son de la forma $A \rightarrow aB$ o de la forma $A \rightarrow \lambda$, donde A y B son variables y a es un símbolo terminal. También se llaman *gramáticas regulares*.

Para cada $i \in \{0, 1, 2, 3\}$, se dice que un lenguaje es de tipo i si es generado por una gramática de tipo i ; esta clasificación de lenguajes se conoce como la *jerarquía de Chomsky*. Cada familia de lenguajes en esta jerarquía se puede caracterizar exactamente como la colección de lenguajes aceptados por un tipo particular de autómata finito, tal como se exhibe en la siguiente tabla:

Lenguajes	Máquinas aceptadoras	Gramáticas
Regulares	Autómatas Finitos (AFD, AFN, AFN- λ)	Regulares, $G = (V, \Sigma, S, P)$. Producciones: $A \rightarrow aB$ o $A \rightarrow \lambda$, donde $A, B \in V, a \in \Sigma$.
Independientes del Contexto (LIC)	Autómatas con pila no deterministas (AFPN)	GIC, $G = (V, \Sigma, S, P)$. Producciones: $A \rightarrow \beta$ donde $A \in V, \beta \in (V \cup \Sigma)^*$.
Sensibles al Contexto (LSC)	Autómatas linealmente acotados (ALA)	GSC, $G = (V, \Sigma, S, P)$. Producciones: $\alpha A \beta \rightarrow \alpha \gamma \beta$, donde $A \in V, \alpha, \beta, \gamma \in (V \cup \Sigma)^*$ y $\gamma \neq \lambda$.
Turing-aceptables (LTA) o recursivamente enumerables	Máquinas de Turing (MT)	Irrestringidas, $G = (V, \Sigma, S, P)$. Producciones: $\alpha \rightarrow \beta$, donde $\alpha, \beta \in (V \cup \Sigma)^*$.

En el presente capítulo se estudiarán las gramáticas de tipo 2 (independientes del contexto) y las de tipo 3 (regulares).

5.2. Gramáticas independientes del contexto

Una *gramática independiente del contexto* (GIC), también llamada *gramática no-contextual* o *gramática de tipo 2*, es una cuádrupla, $G = (V, \Sigma, S, P)$ formada por:

1. Un alfabeto V cuyos elementos se llaman *variables* o *símbolos no-terminales*.
2. Un alfabeto Σ cuyos elementos se llaman *símbolos terminales*. Se exige que los alfabetos Σ y V sean disyuntos.
3. Una variable especial $S \in V$, llamada *variable inicial* o *símbolo inicial* de la gramática.
4. Un conjunto finito $P \subseteq V \times (V \cup \Sigma)^*$ de *producciones* o *reglas de re-escritura*. Una producción $(A, \beta) \in P$ de G se denota por $A \rightarrow \beta$ y se lee “ A produce β ”. A es una variable y se denomina la *cabeza* de la producción; β es una cadena en $(V \cup \Sigma)^*$, y se denomina el *cuerpo* de la producción (formado por concatenaciones de terminales con no-terminales). El significado de una producción $A \rightarrow \beta$ es el siguiente: la variable A se puede reemplazar (o sobre-escribir) por la cadena β .

Notación y definiciones. En ejemplos concretos, las variables se denotan con letras mayúsculas A, B, C, \dots , mientras que los elementos de Σ o símbolos terminales se denotan con primeras letras minúsculas a, b, c, \dots . Las cadenas de terminales se denotan con las letras minúsculas u, v, w, x, y, z , y las cadenas de $(V \cup \Sigma)^*$, que pueden tener tanto terminales como no terminales, se denotan con letras griegas minúsculas $\alpha, \beta, \gamma, \dots$.

Si $A \rightarrow \beta$ es una producción, entonces en una cadena como $\gamma A \gamma'$, donde $\gamma, \gamma' \in (V \cup \Sigma)^*$, la variable A se puede reemplazar por β para obtener $\gamma \beta \gamma'$; esto se denota como

$$\gamma A \gamma' \Longrightarrow \gamma \beta \gamma'.$$

Se dice que $\gamma \beta \gamma'$ se *deriva* o se *genera directamente* (o en un paso) de $\gamma A \gamma'$. Si se quiere hacer referencia a la gramática G , se escribe

$$\gamma A \gamma' \xRightarrow{G} \gamma \beta \gamma' \quad \text{ó} \quad \gamma A \gamma' \Longrightarrow_G \gamma \beta \gamma'.$$

Si $\alpha_1, \alpha_2, \dots, \alpha_n$ son cadenas en $(V \cup \Sigma)^*$ y hay una sucesión de derivaciones directas

$$\alpha_1 \xRightarrow{G} \alpha_2, \quad \alpha_2 \xRightarrow{G} \alpha_3, \quad \dots, \quad \alpha_{n-1} \xRightarrow{G} \alpha_n$$

se dice que α_n se deriva de α_1 y se escribe $\alpha_1 \xRightarrow{*} \alpha_n$. La anterior sucesión de derivaciones directas se representa como

$$\alpha_1 \Longrightarrow \alpha_2 \Longrightarrow \alpha_3 \Longrightarrow \dots \Longrightarrow \alpha_{n-1} \Longrightarrow \alpha_n$$

y se dice que es una *derivación* o una *generación* de α_n a partir de α_1 . Para toda cadena α se asume que $\alpha \xRightarrow{*} \alpha$; por lo tanto, $\alpha \xRightarrow{*} \beta$ significa que β se obtiene de α en un

número finito de pasos utilizando producciones de la gramática (cero, uno o más pasos). Análogamente, $\alpha \xRightarrow{+} \beta$ significa que β se obtiene de α en un número finito de pasos (uno o más pasos). Nótese que se utilizan flechas simples, \rightarrow , para producciones y flechas dobles, \Rightarrow , al aplicar las producciones en derivaciones concretas.

El lenguaje generado por una gramática G se denota por $L(G)$ y se define como

$$L(G) := \{w \in \Sigma^* : S \xRightarrow{+} w\}.$$

Es decir, el lenguaje generado por G está formado por las cadenas de terminales que se pueden derivar (o generar) en varios pasos a partir de la variable inicial S , aplicando en cada paso una producción. La igualdad $L(G) = L$ es estricta y requiere que se satisfagan las dos contencencias $L(G) \subseteq L$ y $L \subseteq L(G)$; es decir, toda cadena generada por G debe estar en L , y toda cadena de L debe ser generada por G .

Un lenguaje L sobre un alfabeto Σ se dice que es un *lenguaje independiente del contexto* (LIC) o *lenguaje no-contextual* si existe una GIC G tal que $L(G) = L$. Dos GIC G_1 y G_2 son *equivalentes* si $L(G_1) = L(G_2)$.

La denominación “independiente del contexto” proviene del hecho de que en una derivación cada producción o regla de re-escritura $A \rightarrow \beta$ se aplica a la variable A independientemente de los caracteres que la rodean, es decir, independientemente del contexto en el que aparece A . En inglés, estas gramáticas se denominan *context-free grammars*, traducido en ocasiones, de manera no muy apropiada, como “gramáticas de contexto libre”.

Ejemplo Sea $G = (V, \Sigma, S, P)$ una gramática dada por:

$$\begin{aligned} V &= \{S, A\} \\ \Sigma &= \{a, b\} \\ P &= \{S \rightarrow aS, S \rightarrow bA, S \rightarrow \lambda, A \rightarrow bA, A \rightarrow b, A \rightarrow \lambda\}. \end{aligned}$$

La manera más conveniente de presentar una gramática es hacer una lista de sus producciones, separando con una barra vertical $|$ las producciones de una misma variable. Se supone siempre que las letras mayúsculas representan variables y las letras minúsculas representan símbolos terminales. Así la gramática G del presente ejemplo se puede presentar simplemente como:

$$G : \begin{cases} S \rightarrow aS \mid bA \mid \lambda \\ A \rightarrow bA \mid b \mid \lambda \end{cases}$$

Se tiene $S \Rightarrow \lambda$. Todas las demás derivaciones en G comienzan ya sea con la producción $S \rightarrow aS$ o con $S \rightarrow bA$. Por lo tanto, tenemos

$$\begin{aligned} S &\Rightarrow aS \xRightarrow{*} a \cdots aS \Rightarrow a \cdots a. \\ S &\Rightarrow bA \xRightarrow{*} b \cdots bA \Rightarrow b \cdots b. \\ S &\Rightarrow aS \xRightarrow{*} a \cdots aS \Rightarrow a \cdots abA \xRightarrow{*} a \cdots ab \cdots bA \Rightarrow a \cdots ab \cdots b. \end{aligned}$$

Por consiguiente $L(G) = a^*b^*$.

Las siguientes cuatro gramáticas también generan el lenguaje a^*b^* y son, por lo tanto, equivalentes a G :

$$\begin{array}{ll} G_1 : \begin{cases} S \rightarrow aS \mid bA \mid \lambda \\ A \rightarrow bA \mid \lambda \end{cases} & G_2 : \begin{cases} S \rightarrow aS \mid A \\ A \rightarrow bA \mid \lambda \end{cases} \\ G_3 : \begin{cases} S \rightarrow AB \\ A \rightarrow aA \mid \lambda \\ B \rightarrow bB \mid \lambda \end{cases} & G_4 : \begin{cases} S \rightarrow AB \mid \lambda \\ A \rightarrow aA \mid a \mid \lambda \\ B \rightarrow bB \mid b \mid \lambda \end{cases} \end{array}$$

Para generar la cadena vacía λ con la gramática G_3 se requieren tres pasos:

$$S \Rightarrow AB \Rightarrow B \Rightarrow \lambda.$$

En realidad, es posible generar el lenguaje a^*b^* utilizando una sola variable, por medio de la siguiente gramática G_5 :

$$G_5 : \begin{cases} S \rightarrow aS \mid Sb \mid \lambda \end{cases}$$

En esta gramática las *aes* se generan de izquierda a derecha y las *bes* de derecha a izquierda. Nótese que G_5 es diferente de la siguiente gramática G_6 :

$$G_6 : \begin{cases} S \rightarrow aS \mid bS \mid \lambda \end{cases}$$

la cual genera todas las cadenas, es decir, $L(G_6) = (a \cup b)^*$.

Ejemplo La gramática

$$G : \begin{cases} S \rightarrow aS \mid aA \\ A \rightarrow bA \mid b \end{cases}$$

genera el lenguaje a^+b^+ . Otra gramática equivalente es:

$$G' : \begin{cases} S \rightarrow AB \\ A \rightarrow aA \mid a \\ B \rightarrow bB \mid b \end{cases}$$

Este lenguaje se puede generar con una sola variable usando la siguiente gramática:

$$G'' : \begin{cases} S \rightarrow aS \mid Sb \mid ab \end{cases}$$

Ejemplo La gramática

$$\begin{cases} S \rightarrow 1A \mid 0 \\ A \rightarrow 0A \mid 1A \mid \lambda \end{cases}$$

genera el lenguaje de los números naturales en numeración binaria. Nótese que la única cadena que comienza con 0, generable con esta gramática, es la cadena 0.

Ejemplo Encontrar una GIC que genere el lenguaje $L = 0^*10^*10^*$ sobre $\Sigma = \{0, 1\}$, es decir, el lenguaje de todas las cadenas con exactamente dos unos.

Solución.

$$G : \begin{cases} S \rightarrow A1A1A \\ A \rightarrow 0A \mid \lambda \end{cases}$$

Una gramática equivalente es

$$G' : \begin{cases} S \rightarrow 0S \mid 1A \\ A \rightarrow 0A \mid 1B \\ B \rightarrow 0B \mid \lambda \end{cases}$$

Esta última gramática G' se puede obtener a partir de un autómata que acepte el lenguaje L , tal como se explicará en la sección 5.3.

Ejemplo Encontrar una GIC que genere el lenguaje $L = \{a^n b^n : n \geq 0\}$ sobre $\Sigma = \{a, b\}$, el cual no es un lenguaje regular.

Solución.

$$S \rightarrow aSb \mid \lambda.$$

Ejemplo Encontrar una GIC que genere el lenguaje de todos los palíndromes sobre $\Sigma = \{a, b\}$, el cual no es lenguaje regular.

Solución.

$$S \rightarrow aSa \mid bSb \mid a \mid b \mid \lambda.$$

Ejemplo Encontrar una GIC que genere el lenguaje L de todas las cadenas sobre $\Sigma = \{a, b\}$ que tienen un número par de símbolos.

Solución. Las cadenas de longitud par (aparte de la cadena vacía λ) se obtienen concatenando los cuatro bloques aa , ab , ba y bb . Por lo tanto, para generar el lenguaje L basta una sola variable que permita concatenar los cuatro bloques de todas las formas posibles. Las siguientes tres gramáticas generan el lenguaje L :

$$\begin{aligned} G_1 : & \begin{cases} S \rightarrow aaS \mid abS \mid baS \mid bbS \mid \lambda \end{cases} \\ G_2 : & \begin{cases} S \rightarrow Saa \mid Sab \mid Sba \mid Sbb \mid \lambda \end{cases} \\ G_3 : & \begin{cases} S \rightarrow aSa \mid aSb \mid bSa \mid bSb \mid \lambda \end{cases} \end{aligned}$$

G_1 genera las cadenas de L de izquierda a derecha, G_2 las genera de derecha a izquierda y G_3 por simetría izquierda-derecha. Así por ejemplo, la cadena $baabbb$ se genera de la

siguiente manera en las tres gramáticas:

$$\text{En } G_1 : S \Rightarrow baS \Rightarrow baabS \Rightarrow baabbbS \Rightarrow baabbb.$$

$$\text{En } G_2 : S \Rightarrow Sbb \Rightarrow Sabbb \Rightarrow Sbaabbb \Rightarrow baabbb.$$

$$\text{En } G_3 : S \Rightarrow bSb \Rightarrow baSbb \Rightarrow baaSbbb \Rightarrow baabbb.$$

Si se combinan las producciones de G_1 con las de G_2 no necesariamente se genera el lenguaje L . Por ejemplo, la gramática G_4 ,

$$G_4 : \left\{ S \rightarrow aaS \mid Sab \mid baS \mid Sbb \mid \lambda \right.$$

genera ciertamente cadenas de longitud par, pero no las genera todas ya que es imposible generar cadenas como $abbabb$ y $aaabba$ (y muchas otras). Se cumple que $L(G_4) \subseteq L$ pero $L \not\subseteq L(G_4)$ y por ende no se tiene la igualdad $L = L(G_4)$.

Otra gramática que genera el lenguaje L es:

$$G_5 : \left\{ \begin{array}{l} S \rightarrow AAS \mid \lambda \\ A \rightarrow a \mid b \end{array} \right.$$

Ejemplo Encontrar una GIC que genere el lenguaje

$$L = \{a^k b^m c^n : m = k + n, k, m, n \geq 0\}$$

sobre el alfabeto $\Sigma = \{a, b, c\}$.

Solución. Las cadenas de L se pueden escribir como $a^k b^m c^n = a^k b^{k+n} c^n = a^k b^k b^n c^n$. Utilizamos la variable A para generar $a^k b^k$, con $k \geq 0$, y la variable B para generar $b^n c^n$, con $n \geq 0$:

$$G : \left\{ \begin{array}{l} S \rightarrow AB \\ A \rightarrow aAb \mid \lambda \\ B \rightarrow bBc \mid \lambda \end{array} \right.$$

Ejemplo Encontrar una GIC que genere el lenguaje

$$L = \{a^k b^m c^n : m > k + n, k, n \geq 0, m \geq 1\}$$

sobre el alfabeto $\Sigma = \{a, b, c\}$.

Solución. Para resolver este problema, utilizamos como punto de partida la gramática G del ejemplo anterior. La condición $m > k + n$ significa que hay estrictamente más *bes* que el total $k + n$; el exceso deseado de *bes* lo podemos obtener por medio de la producción $A \rightarrow Ab$. Obsérvese que el lenguaje L incluye cadenas que solamente tienen *bes* (cuando $k = n = 0$).

$$G_1 : \left\{ \begin{array}{l} S \rightarrow AB \\ A \rightarrow aAb \mid Ab \mid b \\ B \rightarrow bBc \mid \lambda \end{array} \right.$$

Las *bes* adicionales también se pueden generar por medio de la variable B , como se hace en la siguiente gramática:

$$G_2 : \begin{cases} S \rightarrow AB \\ A \rightarrow aAb \mid \lambda \\ B \rightarrow bBc \mid bB \mid b \end{cases}$$

Otra manera de generar este lenguaje es utilizar una nueva variable C , colocada entre A y B , que genere el exceso de *bes*. Obtenemos así la gramática G_3 ,

$$G_3 : \begin{cases} S \rightarrow ACB \\ A \rightarrow aAb \mid \lambda \\ B \rightarrow bBc \mid \lambda \\ C \rightarrow bC \mid b \end{cases}$$

Ejercicios de la sección 5.2

① Encontrar GIC que generen los siguientes lenguajes sobre $\Sigma = \{a, b\}$:

- (i) $a^*b \cup a$.
- (ii) $a^*b \cup b^*a$.
- (iii) $(a \cup b)^*$.
- (iv) $(ab \cup ba)^*$.
- (v) $a^*(ab \cup b)^+$.

② Encontrar GIC que generen los siguientes lenguajes sobre $\Sigma = \{a, b\}$:

- (i) $\{a^{n+1}b^{2n+1} : n \geq 0\}$.
- (ii) $\{a^n b^{n+1} a : n \geq 1\}$.
- (iii) $\{a^n b^m a^{n+1} : n \geq 0, m \geq 1\}$.
- (iv) $\{a^{n+1} b^m a^{2n} : n, m \geq 0\}$.
- (v) $\{a^m b^n : m > n \geq 0\}$.
- (vi) $\{a^m b^n : m, n \geq 0, m \neq n\}$.
- (vii) $\{a^m b^n : m, n \geq 0, n > 2m\}$.
- (viii) $\{a^m b^n : 0 \leq m \leq n \leq 2m\}$.

③ Encontrar GIC que generen los siguientes lenguajes sobre $\Sigma = \{a, b, c\}$:

- (i) $\{a^k b^m c^n : k, m, n \geq 0, n = k + 2m\}$
- (ii) $\{a^k b^m c^n : k, m \geq 0, n \geq 1, n > k + 2m\}$
- (iii) $\{a^k b^m c^n : k, m, n \geq 0, k = m + 2n\}$

$$(iv) \{a^k b^m c^n : m, n \geq 0, k \geq 1, k > m + 2n\}$$

$$(v) \{a^k b^m c^n : k, m, n \geq 0, m = 2k + n\}$$

$$(vi) \{a^k b^m c^n : k, n \geq 0, m \geq 1, m > 2k + n\}$$

④ Encontrar GIC que generen los siguientes lenguajes sobre $\Sigma = \{a, b, c, d\}$:

$$(i) \{a^m b^{2n} c^n d^{2m} : m, n \geq 1\}.$$

$$(ii) \{a^{2n} b^n c^m d^{2m} : m, n \geq 1\}.$$

⑤ Sea $\Sigma = \{0, 1\}$. Encontrar una GIC que genere el lenguaje de las cadenas que tienen igual número de ceros que de unos.

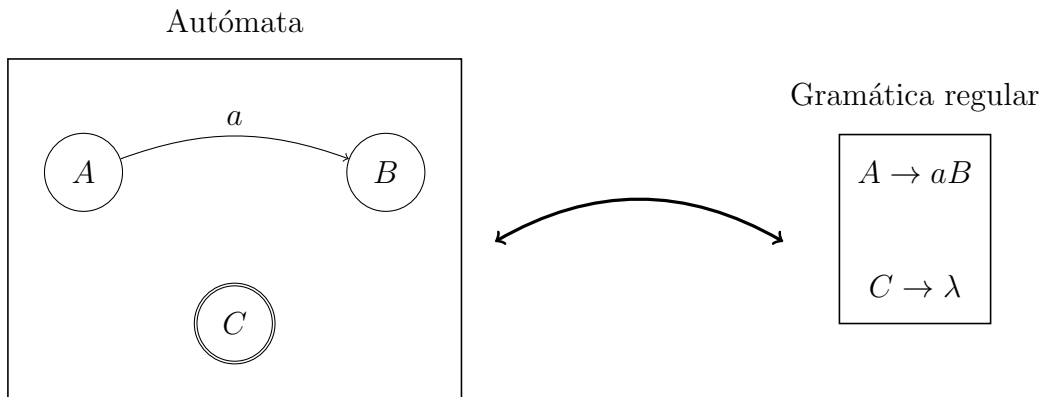
5.3. Gramáticas regulares

5.3.1 Definición. Una GIC $G = (V, \Sigma, S, P)$ se llama *regular* si sus producciones son de la forma

$$\begin{cases} A \rightarrow aB, & a \in \Sigma, A, B \in V. \\ A \rightarrow \lambda & A \in V. \end{cases}$$

En la producción $A \rightarrow aB$, las variables A y B no necesariamente son distintas.

A partir de un autómata AFD o AFN M se puede obtener una gramática regular G , tal que $L(M) = L(G)$, siguiendo el procedimiento esquematizado en el diagrama siguiente. Los estados de M se convierten en las variables de G ; el estado inicial de M pasa a ser la variable inicial de G . Un arco del estado A al estado B con etiqueta a da lugar a la producción $A \rightarrow aB$, y un estado de aceptación C da lugar a la producción $C \rightarrow \lambda$.



El procedimiento es completamente reversible y establece una correspondencia directa entre autómatas y gramáticas regulares. En el Teorema 5.3.2 se demuestra que si M es un AFD, la gramática regular G obtenida satisface $L(M) = L(G)$, y en el Teorema 5.3.3 se extiende este resultado a autómatas no-deterministas.

5.3.2 Teorema. Dado un AFD $M = (\Sigma, Q, q_0, F, \delta)$, existe una GIC regular $G = (V, \Sigma, S, P)$ tal que $L(M) = L(G)$.

Demostración. Sea $V = Q$ y $S = q_0$. Las producciones de G están dadas por

$$\begin{cases} q \rightarrow ap & \text{si y sólo si } \delta(q, a) = p. \\ q \rightarrow \lambda & \text{si y sólo si } q \in F. \end{cases}$$

Demostraremos primero que para toda $w \in \Sigma^*$, $w \neq \lambda$ y para todo $p, q \in Q$ se tiene

(1) Si $\delta(q, w) = p$ entonces $q \xRightarrow{*} wp$.

La demostración de (1) se hace por inducción sobre w . Si $w = a$ y $\delta(q, a) = p$, entonces $q \rightarrow ap$ es una producción de G y obviamente se concluye $q \Rightarrow ap$. Para el paso inductivo, sea $\delta(q, wa) = p'$. Entonces

$$p' = \delta(q, wa) = \delta(\delta(q, w), a) = \delta(p, a)$$

donde $\delta(q, w) = p$. Por hipótesis de inducción $q \xRightarrow{*} wp$ y como $\delta(p, a) = p'$, entonces $p \Rightarrow ap'$. Por lo tanto,

$$q \xRightarrow{*} wp \Rightarrow wap'$$

que era lo que se quería demostrar.

A continuación demostraremos el recíproco de (1): para toda $w \in \Sigma^*$, $w \neq \lambda$ y para todo $p, q \in Q$ se tiene

(2) Si $q \xRightarrow{*} wp$ entonces $\delta(q, w) = p$.

La demostración de (2) se hace por inducción sobre la longitud de la derivación $q \xRightarrow{*} wp$, es decir, por el número de pasos o derivaciones directas que hay en $q \xRightarrow{*} wp$. Si la derivación tiene longitud 1, necesariamente $q \Rightarrow ap$ lo cual significa que $\delta(q, a) = p$. Para el paso inductivo, supóngase que $q \xRightarrow{*} wp$ tiene longitud $n + 1$, $w = w'a$ y en el último paso se aplica la producción $p' \rightarrow ap$. Entonces

$$q \xRightarrow{*} w'p' \Rightarrow w'ap = wp.$$

Por hipótesis de inducción, $\delta(q, w') = p'$ y por consiguiente

$$\delta(q, w) = \delta(q, w'a) = \delta(\delta(q, w'), a) = \delta(p', a) = p,$$

que era lo que se quería demostrar.

Como consecuencia de (1) y (2) se puede ahora demostrar que

(3) Para toda cadena $w \in \Sigma^*$, $\delta(q_0, w) \in F$ si y sólo si $S \xRightarrow{*}_G w$,

lo cual afirma que $L(M) = L(G)$. En efecto, si $w = \lambda$, $\delta(q_0, w) \in F$ si y sólo si $q_0 \in F$. Por lo tanto, $q_0 \rightarrow \lambda$ es una producción de G . Así que $S \Rightarrow \lambda$. Recíprocamente, si $S \xRightarrow{*}_G \lambda$, necesariamente $S \Rightarrow \lambda$, $q_0 \in F$ y $\delta(q_0, \lambda) \in F$.

Sea ahora $w \neq \lambda$. Si $\delta(q_0, w) = p \in F$, por (1) se tiene $q_0 \xRightarrow{*} w$, o sea, $S \xRightarrow{*}_G w$. Recíprocamente, si $S \xRightarrow{*}_G w$, entonces $q_0 \xRightarrow{*}_G wp \Rightarrow w$ donde $p \rightarrow \lambda$. Utilizando (2), se tiene $\delta(q_0, w) = p \in F$. \square

5.3.3 Teorema. Dada una GIC regular $G = (V, \Sigma, S, P)$, existe un AFN $M = (Q, \Sigma, q_0, F, \Delta)$ tal que $L(M) = L(G)$.

Demostración. Se construye $M = (Q, \Sigma, q_0, F, \Delta)$ haciendo $Q = V$, $q_0 = S$ y

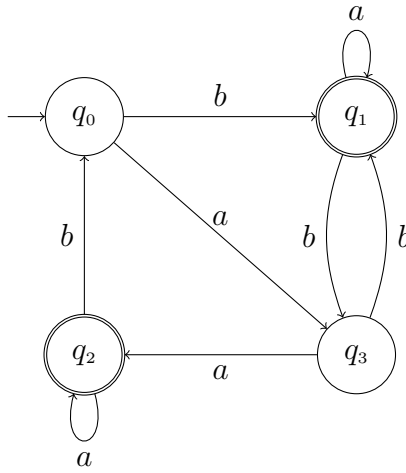
$$\begin{cases} B \in \Delta(A, a) & \text{para cada producción } A \rightarrow aB. \\ A \in F & \text{si } A \rightarrow \lambda. \end{cases}$$

Usando razonamientos similares a los del Teorema 5.3.2, se puede demostrar que

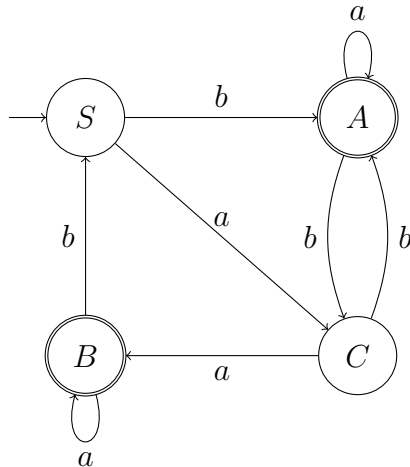
$$A \xRightarrow{*}_G wB \quad \text{si y sólo si} \quad B \in \Delta(A, w), \quad \text{para todo } w \in \Sigma^*, w \neq \lambda,$$

de donde $L(M) = L(G)$. □

Ejemplo Dado el siguiente AFD M , encontrar una gramática regular G tal que $L(M) = L(G)$.



Solución. Según la construcción mencionada arriba, los estados del autómata M son las variables de la gramática G . Renombramos los estados de M con las letras mayúsculas S , A , B y C . Toda transición de M da lugar a una producción en G ; los estados de aceptación A y B inducen las producciones $A \rightarrow \lambda$ y $B \rightarrow \lambda$, respectivamente.



$$G : \begin{cases} S \rightarrow bA \mid aC \\ A \rightarrow aA \mid bC \mid \lambda \\ B \rightarrow bS \mid aB \mid \lambda \\ C \rightarrow bA \mid aB \end{cases}$$

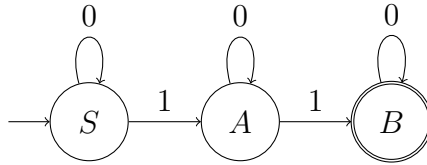
Puesto que el autómata M es determinista, para cada cadena aceptada u existe una única trayectoria etiquetada por los símbolos de u , desde el estado inicial hasta un estado de aceptación; tal trayectoria corresponde a una única derivación $S \xRightarrow{*} u$ en la gramática regular G . Así por ejemplo, la cadena $baabaa$ tiene una única trayectoria de aceptación, a saber, $S \xrightarrow{b} A \xrightarrow{a} A \xrightarrow{a} A \xrightarrow{b} C \xrightarrow{a} B \xrightarrow{a} B$, la cual corresponde a una única derivación en G :

$$S \Rightarrow bA \Rightarrow baA \Rightarrow baaA \Rightarrow baabC \Rightarrow baabaB \Rightarrow baabaaB \Rightarrow baabaa.$$

Ejemplo Para el lenguaje regular $0^*10^*10^*$, sobre $\Sigma = \{0, 1\}$ (el lenguaje de todas las cadenas con exactamente dos unos), vimos en la sección 5.2 una gramática G que lo genera:

$$G : \begin{cases} S \rightarrow A1A1A \\ A \rightarrow 0A \mid \lambda \end{cases}$$

Esta gramática no es regular, pero por medio del AFD



y de la construcción del Teorema 5.3.2 se puede obtener una GIC regular G' que genere $0^*10^*10^*$:

$$G' : \begin{cases} S \rightarrow 0S \mid 1A \\ A \rightarrow 0A \mid 1B \\ B \rightarrow 0B \mid \lambda \end{cases}$$

Los teoremas anteriores permiten concluir que la familia de los lenguajes regulares está estrictamente contenida en la familia de los Lenguajes Independientes del Contexto, tal como se enuncia en el siguiente corolario.

5.3.4 Corolario.

1. Un lenguaje es regular si y solamente si es generado por una gramática regular.
2. Todo lenguaje regular es un LIC (pero no viceversa).

Demostración.

1. Se sigue del Teorema 5.3.2, el Teorema 5.3.3 y del Teorema de Kleene.
2. Se sigue de la parte 1. Por otro lado, tenemos muchos ejemplos de lenguajes LIC que no son regulares, como $\{a^n b^n : n \geq 0\}$ y el lenguaje de los palíndromes sobre el alfabeto $\{a, b\}$. \square

Ejercicios de la sección 5.3

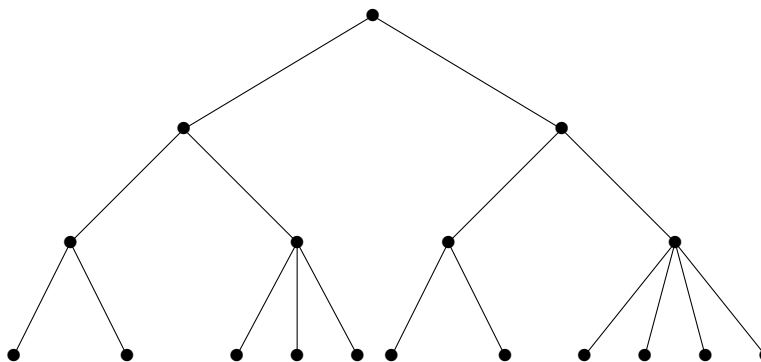
Encontrar gramáticas regulares que generen los siguientes lenguajes sobre el alfabeto $\Sigma = \{a, b\}$:

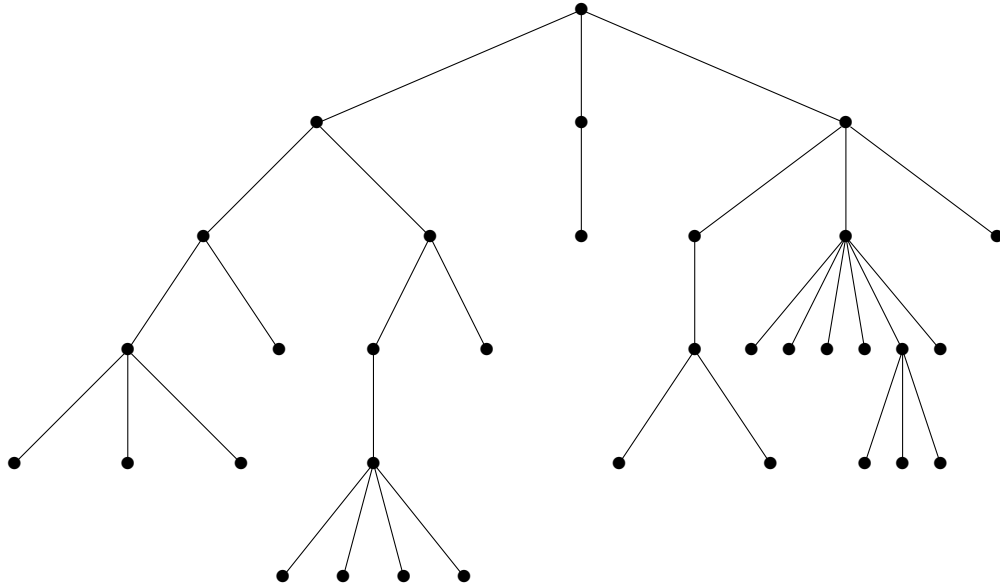
- ① $ba^*b \cup b^+$.
- ② a^+b^*a .
- ③ $a^*b \cup b^*a$.
- ④ El lenguaje de todas las cadenas que tienen un número impar de a 'es y un número par de b 'es.

5.4. Árboles sintácticos

Un *árbol con raíz* es un tipo muy particular de grafo no-dirigido; tiene un nodo especial, llamado la *raíz* del árbol, la cual se ramifica en nodos, llamados *descendientes inmediatos*, cada uno de los cuales puede tener, a su vez, descendientes inmediatos, y así sucesivamente. Un nodo puede tener 0, 1, o más descendientes inmediatos pero tiene un único antecesor inmediato. El único nodo que no tiene antecesores es la raíz. Los nodos que tienen descendientes, excepto la raíz, se denominan *nodos interiores*. En la terminología usualmente utilizada, los descendientes inmediatos de un nodo también se denominan *hijos*, y los nodos que no tienen descendientes se denominan *hojas*. Un árbol queda caracterizado por la siguiente propiedad: hay una única trayectoria entre la raíz y cualquier otro nodo. Los nodos que aparecen en la única trayectoria entre la raíz y un nodo determinado N se denominan los *ancestros* de N .

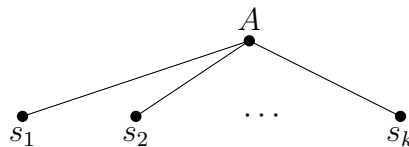
Ejemplo Dos árboles con raíz:





5.4.1 Definición. Dada una GIC $G = (V, \Sigma, S, P)$, el *árbol de una derivación* $S \xRightarrow{+} w$, con $w \in \Sigma^*$, es un árbol con raíz y con nodos etiquetados, definido recursivamente de la siguiente forma:

1. La raíz está etiquetada con el símbolo inicial S .
2. Si en un paso la derivación se aplica la producción $A \rightarrow s_1 s_2 \cdots s_k$, donde cada $s_i \in V \cup \Sigma$, el nodo A tiene k descendientes inmediatos etiquetados con s_1, s_2, \dots, s_k , escritos de izquierda a derecha:



Si en un paso de la derivación se aplica la producción $A \rightarrow \lambda$, el nodo A tiene un único descendiente etiquetado con λ :



De esta manera, los nodos interiores están etiquetados con símbolos no terminales, y las hojas del árbol están etiquetadas con símbolos terminales o con λ . Si se leen de izquierda a derecha las hojas del árbol de una derivación de $S \xRightarrow{+} w$, se obtiene precisamente la cadena w , con algunos λ intercalados.

Los árboles de derivaciones se suelen llamar *árboles sintácticos*.

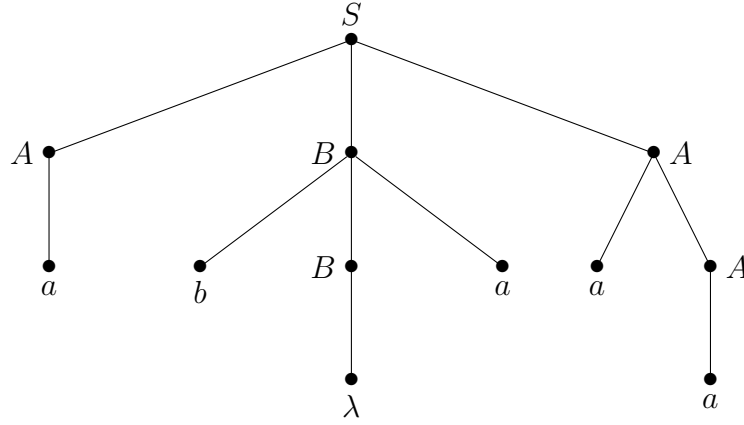
Ejemplo Sea G la gramática:

$$G : \begin{cases} S \rightarrow ABA \mid AaB \\ A \rightarrow aA \mid a \\ B \rightarrow bBa \mid \lambda \end{cases}$$

Consideremos la siguiente derivación de la cadena $abaaa$:

$$(1) \quad S \Rightarrow \underline{A}BA \Rightarrow Ab\underline{B}aA \Rightarrow Ab\underline{B}aaA \Rightarrow Abaa\underline{A} \Rightarrow Abaaa \Rightarrow abaaa.$$

En cada paso de la derivación, se ha subrayado la variable para la cual se ha utilizado una producción de la gramática G . El árbol de la derivación (1) es:



Las producciones utilizadas en la derivación (1) se pueden aplicar en diferente orden; obtenemos, por ejemplo, las siguientes derivaciones:

$$(2) \quad S \Rightarrow \underline{A}BA \Rightarrow aB\underline{A} \Rightarrow aBa\underline{A} \Rightarrow aBaa \Rightarrow abBaaa \Rightarrow abaaa.$$

$$(3) \quad S \Rightarrow \underline{A}BA \Rightarrow a\underline{B}A \Rightarrow ab\underline{B}aA \Rightarrow abaA \Rightarrow abaaA \Rightarrow abaaa.$$

Las derivaciones (1), (2) y (3) tienen todas seis pasos y ellas se aplican exactamente las mismas producciones pero en diferente orden. Las tres derivaciones tienen todas el mismo árbol, exhibido arriba. Los árboles sintácticos muestran únicamente las producciones utilizadas, no el orden en que se aplican.

Entre las posibles derivaciones de cadenas se distinguen ciertas derivaciones “estándares”, las llamadas derivaciones a izquierda.

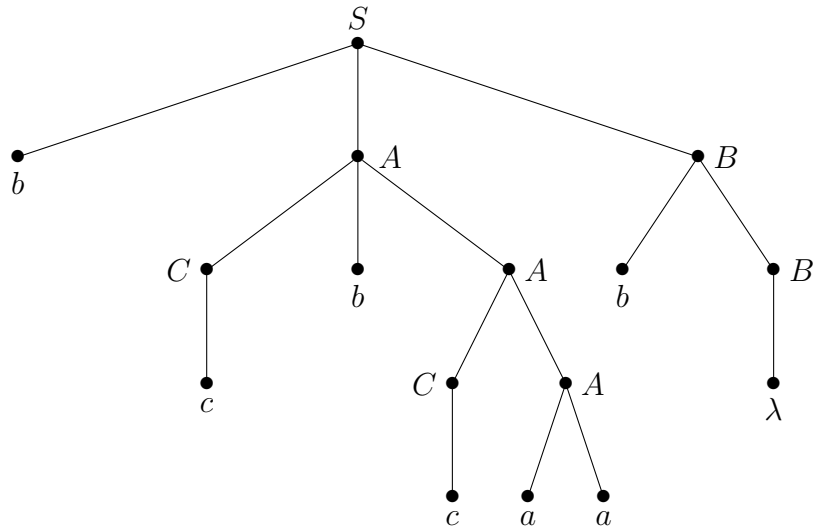
5.4.2 Definición. Una derivación se llama *derivación a izquierda* (o *derivación más a la izquierda*) si en cada paso se aplica una producción a la variable que está más a la izquierda.

En el ejemplo anterior, la derivación (3) es una derivación a izquierda. En general, una derivación cualquiera se puede transformar siempre en una única derivación a izquierda

cambiando el orden en que se aplican las producciones, de tal forma que en cada paso se aplica una producción a la variable que esté más a la izquierda. Además, existe una correspondencia biyectiva entre derivaciones a izquierda y árboles sintácticos, tal como se enuncia en la siguiente proposición.

5.4.3 Proposición. Toda derivación a izquierda determina un único árbol sintáctico. Recíprocamente, cualquier árbol sintáctico en una gramática G determina una única derivación a izquierda en G .

Ejemplo Encontrar la única derivación a izquierda determinada por el siguiente árbol sintáctico T proveniente de cierta gramática G con alfabeto de variables $V = \{S, A, B, C\}$ y alfabeto de terminales $\Sigma = \{a, b, c\}$.



Solución.

$$\begin{aligned} S &\Rightarrow bAB \Rightarrow bCbAB \Rightarrow bcbAB \Rightarrow bcbCAB \Rightarrow bcbcAB \Rightarrow bcbcaaB \\ &\Rightarrow bcbcaabB \Rightarrow bcbcaab\lambda. \end{aligned}$$

Las hojas del árbol sintáctico forman la cadena generada $bcbcaab\lambda = bcbcaab$.

Ejercicios de la sección 5.4

① Sea G siguiente gramática:

$$G : \begin{cases} S \longrightarrow aS \mid AaB \\ A \longrightarrow aA \mid a \\ B \longrightarrow bBbB \mid b \end{cases}$$

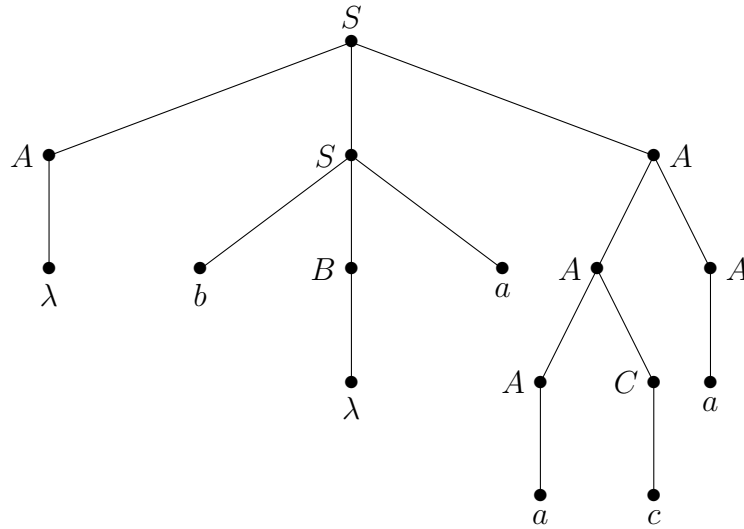
Encontrar una derivación de la cadena $aaaabbbb$ y hallar el árbol de tal derivación.

② Sea G la siguiente gramática:

$$G : \begin{cases} S \rightarrow ABC \mid BaC \mid aB \\ A \rightarrow Aa \mid a \\ B \rightarrow BAB \mid bab \\ C \rightarrow cC \mid \lambda \end{cases}$$

Encontrar derivaciones a izquierda de las cadenas $w_1 = abab$, $w_2 = babacc$, $w_3 = ababababc$ y hallar los árboles de tales derivaciones.

③ Encontrar la única derivación a izquierda determinada por el siguiente árbol sintáctico proveniente de cierta gramática G con alfabeto de variables $V = \{S, A, B, C\}$ y alfabeto de terminales $\Sigma = \{a, b, c\}$.



5.5. Gramáticas ambiguas

5.5.1 Definición. Una GIC $G = (V, \Sigma, S, P)$ es *ambigua* si existe (por lo menos) una cadena $w \in L(G)$ para la cual hay dos derivaciones a izquierda diferentes. Según la Proposición 5.4.3, se puede decir de manera equivalente que una GIC $G = (V, \Sigma, S, P)$ es ambigua si existe una cadena $w \in L(G)$ con dos árboles sintácticos diferentes. En tal caso, se dice también que la cadena w es ambigua en $L(G)$.

Como consecuencia de esta definición, se concluye que una GIC $G = (V, \Sigma, S, P)$ *no es ambigua* cuando toda cadena $w \in L(G)$ tiene una única derivación a izquierda. Equivalentemente, G no es ambigua cuando toda cadena $w \in L(G)$ tiene un único árbol sintáctico, o sea, cuando no hay cadenas ambiguas en $L(G)$.

Como se mostrará más adelante, las GIC se utilizan para formalizar la sintaxis de los lenguajes de programación, y resulta imperativo que las gramáticas involucradas no sean ambiguas ya que un programa computacional debe tener una única interpretación.

Ejemplo Considérese el alfabeto de terminales $\Sigma = \{x, y, z, 0, 1, +, *\}$ y la gramática G_1 cuya única variable es S y cuyas producciones son:

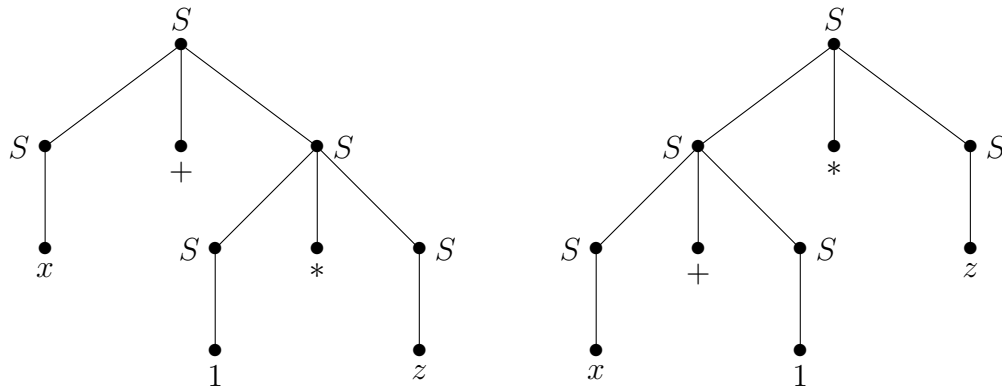
$$G_1 : \{S \rightarrow S + S \mid S * S \mid x \mid y \mid z \mid 0 \mid 1\}$$

Si se interpreta el símbolo $+$ como ‘suma’ y el símbolo $*$ como ‘producto’, G_1 genera expresiones algebraicas sencillas que involucran los terminales $x, y, z, 0$ y 1 . La ambigüedad surge porque algunas expresiones se pueden interpretar ya sea como sumas, o ya sea como productos. Por ejemplo, la cadena $x + 1 * z$ es ambigua en $L(G_1)$ ya que se puede generar como una suma o como un producto por medio de dos derivaciones a izquierda diferentes:

$$S \Rightarrow S + S \Rightarrow x + S \Rightarrow x + S * S \Rightarrow x + 1 * S \Rightarrow x + 1 * z.$$

$$S \Rightarrow S * S \Rightarrow S + S * S \Rightarrow x + S * S \Rightarrow x + 1 * S \Rightarrow x + 1 * z.$$

Los árboles de derivación correspondientes a las dos anteriores derivaciones son:



La ambigüedad en la gramática G_1 se puede eliminar introduciendo paréntesis, como se hace en la siguiente gramática G_2 :

$$G_2 : \{S \rightarrow (S + S) \mid (S * S) \mid x \mid y \mid z \mid 0 \mid 1\}$$

En G_2 el alfabeto de terminales sería $\Sigma = \{x, y, z, 0, 1, +, *, (,)\}$. En esta gramática se pueden generar las expresiones $((x + 1) * z)$ y $(x + (1 * z))$ que eliminan la ambigüedad de la expresión $x + 1 * z$.

Precedencia de los operadores. Aunque la introducción de paréntesis elimina la ambigüedad, las expresiones algebraicas generadas en la gramática G_2 del ejemplo anterior tienen un excesivo número de paréntesis ya que cada aplicación de los operadores $+$ y $*$ produce un par de ellos. Esto dificulta el análisis sintáctico (en un compilador, por ejemplo). Lo más corriente en estas situaciones es utilizar gramáticas que establezcan un orden de precedencia para los operadores. Lo usual es establecer que $*$ tenga precedencia sobre $+$, es decir, por convención el producto $*$ actúa antes que la suma $+$. Por ejemplo, la expresión $x + 1 * z$ se interpreta como $x + (1 * z)$ sin necesidad de usar paréntesis. En la siguiente

gramática G_3 se implementa la precedencia del operador $*$ sobre el operador $+$ para generar expresiones algebraicas sobre el alfabeto de terminales $\Sigma = \{x, y, z, 0, 1, +, *, (,)\}$. El alfabeto de variables de G_3 es $V = \{S, T, F\}$.

$$G_3 : \begin{cases} S \rightarrow T + S \mid T \\ T \rightarrow F * T \mid F \\ F \rightarrow x \mid y \mid z \mid 0 \mid 1 \mid (S) \end{cases}$$

En G_3 toda expresión algebraica se genera como una suma de “términos”:

$$S \Rightarrow T + S \Rightarrow T + T + S \xRightarrow{*} T + T + \dots + S \Rightarrow T + T + \dots + T,$$

y todo término generado con la variable T es un producto de “factores”:

$$T \Rightarrow F * T \Rightarrow F * F * T \xRightarrow{*} F * F * \dots * T \Rightarrow F * F * \dots * F.$$

Uno de los posibles factores generados con la variable F es (S) , que se utiliza para generar factores encerrados entre paréntesis. Vamos a ilustrar cómo se puede generar en G_3 la expresión algebraica $x^2 + y(x + z^3 + 1)$. Puesto que G_3 únicamente posee los operadores aritméticos $+$ y $*$, dicha expresión se debe generar como $x * x + y * (x + z * z * z + 1)$.

$$\begin{aligned} S &\Rightarrow T + S \Rightarrow T + T \xRightarrow{2} F * T + F * T \xRightarrow{2} F * F + F * F \xRightarrow{4} x * x + y * (S) \\ &\xRightarrow{3} x * x + y * (T + T + T) \xRightarrow{4} x * x + y * (x + T + 1) \\ &\xRightarrow{3} x * x + y * (x + F * F * F + 1) \xRightarrow{3} x * x + y * (x + z * z * z + 1). \end{aligned}$$

Eliminación de la ambigüedad. La ambigüedad es un asunto delicado porque no existe un algoritmo o procedimiento general que permita decidir si una GIC dada G es o no ambigua. En otras palabras, el problema de decisión, “Dada una GIC G , ¿es G ambigua?” es matemáticamente insoluble. La inexistencia de un algoritmo para este problema de decisión es una imposibilidad matemática cuya demostración se puede consultar en las referencias bibliográficas. Por consiguiente, en cada caso concreto hay que proceder por inspección o por ensayo y error, recurriendo a las definiciones, para poder concluir si una GIC dada es o no ambigua.

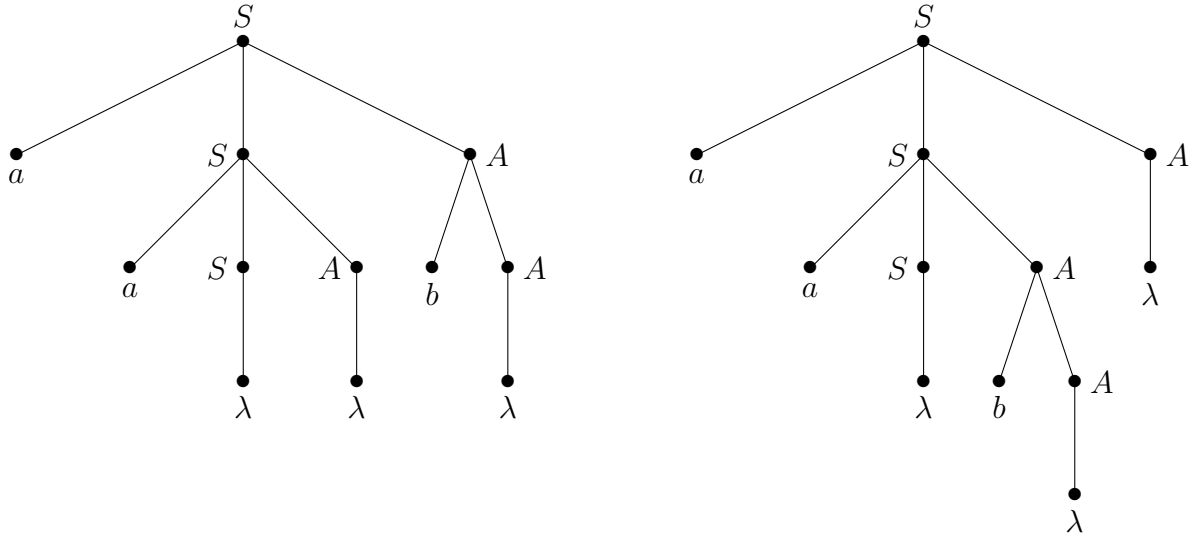
Ejemplo Demostrar que la siguiente gramática G es ambigua y encontrar una gramática G' no ambigua equivalente a G , es decir, tal que $L(G) = L(G')$.

$$G : \begin{cases} S \rightarrow aSA \mid \lambda \\ A \rightarrow bA \mid \lambda \end{cases}$$

Solución. G es ambigua porque hay cadenas con dos derivaciones a izquierda diferentes, es decir, cadenas ambiguas en $L(G)$. Por ejemplo, para la cadena aab tenemos:

$$\begin{aligned} S &\Rightarrow aSA \Rightarrow aaSAA \Rightarrow aaAA \Rightarrow aaA \Rightarrow aabA \Rightarrow aab. \\ S &\Rightarrow aSA \Rightarrow aaSAA \Rightarrow aaAA \Rightarrow aabAA \Rightarrow aabA \Rightarrow aab. \end{aligned}$$

Los árboles sintácticos de estas dos derivaciones se exhiben a continuación.

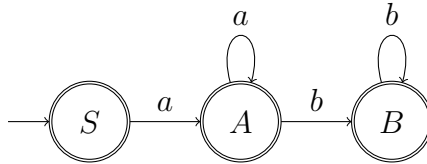


Por simple inspección, observamos que el lenguaje generado por esta gramática es $a^+b^*\cup\lambda$. Se puede construir una gramática no-ambigua que genere el mismo lenguaje:

$$G' : \begin{cases} S \rightarrow AB \mid \lambda \\ A \rightarrow aA \mid a \\ B \rightarrow bB \mid \lambda \end{cases}$$

Para ver que la gramática G' no es ambigua se puede razonar de la siguiente manera: la cadena λ se puede generar de manera única con la derivación $S \Rightarrow \lambda$. Una derivación de una cadena no vacía debe comenzar aplicando la producción $S \rightarrow AB$; la variable A genera cadenas de a 'es de manera única (iterando la producción $A \rightarrow aA$), y B genera cadenas de b 'es también de manera única (iterando la producción $B \rightarrow bB$). Por consiguiente, toda cadena tiene una única derivación a izquierda.

Como el lenguaje $L = a^+b^* \cup \lambda$ es regular, podemos encontrar otra gramática G'' no ambigua equivalente a G a partir de un AFD M tal que $L(M) = L$.



La gramática regular inducida por M , según el procedimiento presentado en la sección 5.3 es G'' :

$$G'' : \begin{cases} S \rightarrow aA \mid \lambda \\ A \rightarrow aA \mid bB \mid \lambda \\ B \rightarrow bB \mid \lambda \end{cases}$$

Puesto que M es un autómata determinista, para cada cadena de L existe una única trayectoria de aceptación en M , y en consecuencia, cada cadena en $L(G'') = L$ se puede generar de manera única. Esto muestra que G'' no es ambigua.

Lenguajes inherentemente ambiguos. En muchos casos es posible encontrar una gramática no ambigua equivalente a una gramática dada, tal como se ha ilustrado en el ejemplo anterior, pero no siempre es posible hacerlo. Se dice que un lenguaje independiente del contexto es *inherentemente ambiguo* si toda GIC que lo genere es ambigua. Es importante resaltar que ser “inherentemente ambiguo” es una propiedad de los lenguajes mientras que ser “ambiguo” es una propiedad de las gramáticas.

El siguiente lenguaje

$$L = \{a^k b^m c^n : m = k \vee m = n, \text{ con } k, m, n \geq 0\}$$

es inherentemente ambiguo. El lenguaje L también se puede definir en la forma

$$L = \{a^k b^k c^n : k, n \geq 0\} \cup \{a^k b^m c^m : k, m \geq 0\}.$$

Intuitivamente, en una gramática que genere a L debe haber una manera de generar cadenas de la forma $a^k b^k c^n$ y otra manera de generar cadenas de la forma $a^k b^m c^m$, lo cual implica que las cadenas de la forma $a^k b^k c^k$ son ambiguas porque se pueden generar de dos maneras diferentes. La demostración rigurosa de este hecho no es sencilla y depende del llamado Lema de Ogden (véase la Bibliografía).

Ejercicios de la sección 5.5

① Demostrar que las siguientes gramáticas son ambiguas:

- (i) $G : \{S \longrightarrow aSb \mid aaSb \mid \lambda.$
- (ii) $G : \{S \longrightarrow aSb \mid aSbb \mid \lambda.$
- (iii) $G : \{S \longrightarrow aSb \mid abS \mid \lambda.$
- (iv) $G : \{S \longrightarrow aaS \mid aaaS \mid \lambda$

② Para cada una de las siguientes gramáticas G , demostrar que G es ambigua y hallar explícitamente el lenguaje $L(G)$. Encontrar luego una gramática no-ambigua que genere el mismo lenguaje $L(G)$.

- | | |
|---|---|
| (i) $G : \begin{cases} S \longrightarrow ASB \mid AB \\ A \longrightarrow aA \mid a \\ B \longrightarrow bB \mid \lambda \end{cases}$ | (iv) $G : \begin{cases} S \longrightarrow AaSbB \mid \lambda \\ A \longrightarrow aA \mid a \\ B \longrightarrow bB \mid \lambda \end{cases}$ |
| (ii) $G : \begin{cases} S \longrightarrow aaSB \mid b \mid \lambda \\ B \longrightarrow Bb \mid \lambda \end{cases}$ | (v) $G : \begin{cases} S \longrightarrow aSb \mid aBb \\ A \longrightarrow aAb \mid b \\ B \longrightarrow Bb \mid A \end{cases}$ |
| (iii) $G : \begin{cases} S \longrightarrow ABA \mid b \mid \lambda \\ A \longrightarrow aA \mid \lambda \\ B \longrightarrow bB \mid \lambda \end{cases}$ | (vi) $G : \begin{cases} S \longrightarrow aSb \mid aAb \\ A \longrightarrow aA \mid B \\ B \longrightarrow S \mid a \end{cases}$ |

5.6. Gramáticas para lenguajes naturales

Para los lenguajes naturales, como el español, se pueden presentar GIC que generen las frases u oraciones permitidas en la comunicación hablada o escrita. Una GIC para generar una porción de las oraciones del idioma español se presenta a continuación; las variables aparecen encerradas entre paréntesis $\langle \rangle$ y $\langle \text{Oración} \rangle$ es la variable inicial de la gramática. Las variables $\langle \text{Art.} \rangle$, $\langle \text{Sustant.} \rangle$, $\langle \text{Prepos.} \rangle$ y $\langle \text{Complemento Circunst.} \rangle$ representan las categorías gramaticales *Artículo*, *Sustantivo*, *Preposición* y *Complemento Circunstancial*, respectivamente. Los terminales son las palabras propias del idioma.

$\langle \text{Oración} \rangle$	$\rightarrow \langle \text{Sujeto} \rangle \langle \text{Verbo} \rangle \langle \text{Complemento Directo} \rangle \mid$ $\langle \text{Sujeto} \rangle \langle \text{Verbo} \rangle \langle \text{Complemento Directo} \rangle \langle \text{Complemento Circunst.} \rangle \mid$ $\langle \text{Sujeto} \rangle \langle \text{Verbo} \rangle \langle \text{Complemento Indirecto} \rangle \langle \text{Complemento Circunst.} \rangle$
$\langle \text{Sujeto} \rangle$	$\rightarrow \langle \text{Sustant.} \rangle \mid \text{Juan} \mid \text{Pedro} \mid \text{María} \mid \dots$
$\langle \text{Complemento Directo} \rangle$	$\rightarrow \langle \text{Prepos.} \rangle \langle \text{Artículo} \rangle \langle \text{Sustant.} \rangle \mid$ $\langle \text{Prepos.} \rangle \langle \text{Artículo} \rangle \langle \text{Sustant.} \rangle \langle \text{Prepos.} \rangle \langle \text{Artículo} \rangle \langle \text{Sustant.} \rangle \mid$ $\langle \text{Prepos.} \rangle \langle \text{Artículo} \rangle \langle \text{Sustant.} \rangle \langle \text{Prepos.} \rangle \langle \text{Sustant.} \rangle \langle \text{Adjetivo} \rangle$
$\langle \text{Complemento Indirecto} \rangle$	$\rightarrow \langle \text{Prepos.} \rangle \langle \text{Artículo} \rangle \langle \text{Sustant.} \rangle \mid$ $\langle \text{Prepos.} \rangle \langle \text{Artículo} \rangle \langle \text{Sustant.} \rangle \langle \text{Adjetivo} \rangle \mid$ $\langle \text{Prepos.} \rangle \langle \text{Sustant.} \rangle \langle \text{Prepos.} \rangle \langle \text{Sustant.} \rangle$
$\langle \text{Complemento Circunst.} \rangle$	$\rightarrow \langle \text{Prepos.} \rangle \langle \text{Artículo} \rangle \langle \text{Sustant.} \rangle \mid$ $\langle \text{Prepos.} \rangle \langle \text{Artículo} \rangle \langle \text{Sustant.} \rangle \langle \text{Adjetivo} \rangle \mid \langle \text{Adverbio} \rangle \mid$ $\langle \text{Prepos.} \rangle \langle \text{Art.} \rangle \langle \text{Sustant.} \rangle \langle \text{Prepos.} \rangle \langle \text{Artículo} \rangle \langle \text{Sustant.} \rangle$
$\langle \text{Sustant.} \rangle$	$\rightarrow \text{casa} \mid \text{perro} \mid \text{libro} \mid \text{lápiz} \mid \text{mesa} \mid \lambda \mid \dots$
$\langle \text{Adjetivo} \rangle$	$\rightarrow \text{rojo} \mid \text{azul} \mid \text{inteligente} \mid \text{malvado} \mid \text{útil} \mid \lambda \mid \dots$
$\langle \text{Prepos.} \rangle$	$\rightarrow \text{a} \mid \text{ante} \mid \text{bajo} \mid \text{con} \mid \text{contra} \mid \text{de} \mid \text{desde} \mid \text{en} \mid \text{entre} \mid \text{hacia} \mid$ $\text{hasta} \mid \text{para} \mid \text{por} \mid \text{según} \mid \text{sin} \mid \text{so} \mid \text{sobre} \mid \text{tras} \mid \lambda \mid \dots$
$\langle \text{Artículo} \rangle$	$\rightarrow \text{el} \mid \text{la} \mid \text{lo} \mid \text{las} \mid \text{los} \mid \text{un} \mid \text{uno} \mid \text{una} \mid \text{unas} \mid \text{unos} \mid \lambda$
$\langle \text{Adverbio} \rangle$	$\rightarrow \text{muy} \mid \text{bastante} \mid \text{poco} \mid \text{demasiado} \mid \text{lento} \mid \text{lentamente} \mid$ $\text{rápido} \mid \text{rápidamente} \mid \lambda \mid \dots$
$\langle \text{Verbo} \rangle$	$\rightarrow \text{escribir} \mid \text{escribo} \mid \text{escribe} \mid \text{escribes} \mid \text{escriben} \mid \text{escribí} \mid$ $\text{escribiste} \mid \text{escribieron} \mid \dots$

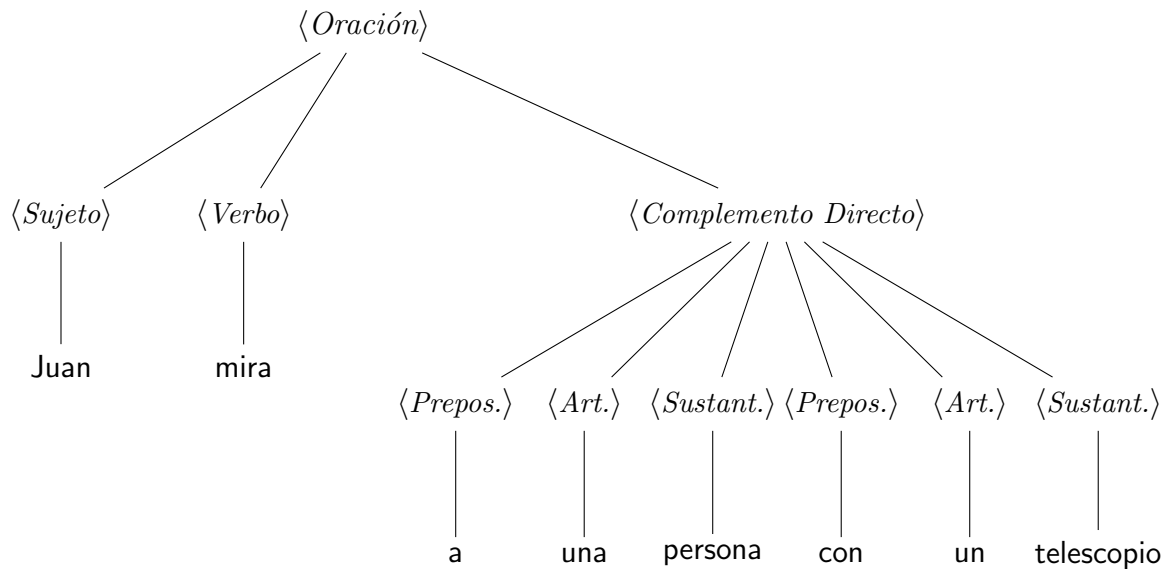
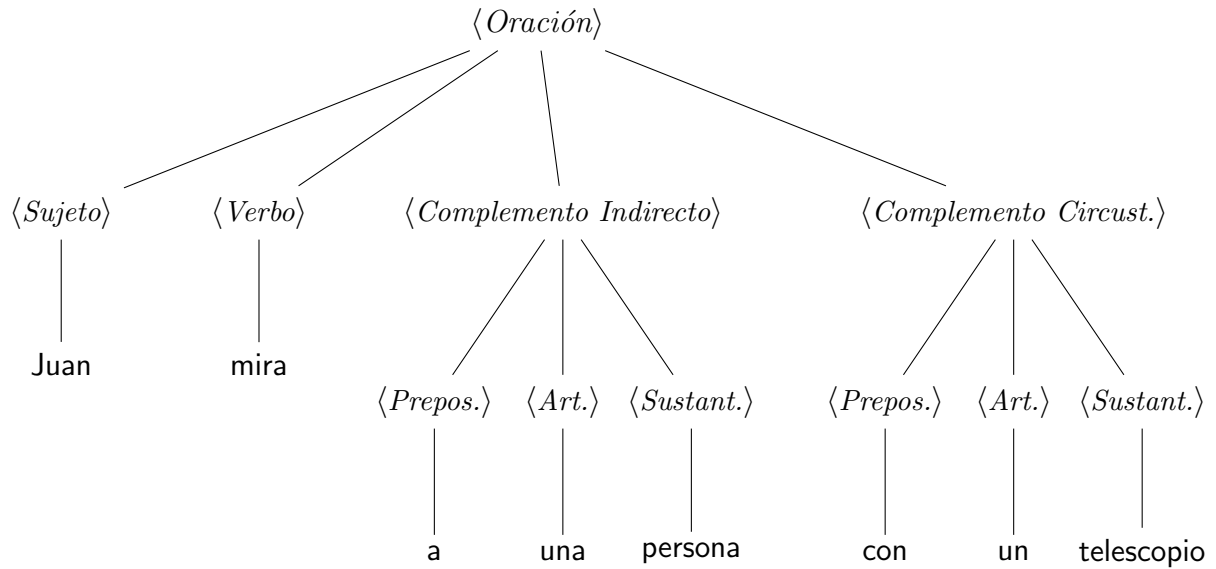
Las gramáticas de los lenguajes naturales son ambiguas porque existen muchas reglas de producción, lo que da lugar a múltiples árboles de derivación para ciertas oraciones.

Ejemplo

La oración

Juan mira a una persona con un telescopio

es ambigua porque las producciones permiten dos árboles de derivación diferentes:



Ejercicios de la sección 5.6

Usando la gramática exhibida en la presente sección, mostrar que las siguientes oraciones del idioma español son ambiguas. En cada caso hacer los árboles de derivación.

- ① María escucha la conversación tras la puerta.
- ② Pedro ve la televisión en la mesa.
- ③ Manuel observa a la chica con vestido rojo.
- ④ Ana soñó con un gato en pijama.

5.7. Gramáticas para lenguajes de programación

Para presentar rigurosamente la sintaxis de los lenguajes de programación se utilizan con frecuencia gramáticas independientes del contexto en una notación denominada *Forma Backus-Naur* (FBN). En la FBN las variables de la gramática se representan mediante palabras descriptivas encerradas entre paréntesis angulares $\langle \rangle$, en la forma $\langle \text{Variable} \rangle$. Además, se utiliza el símbolo $::=$ para las producciones, en vez de la flecha \rightarrow .

Ejemplo A continuación presentamos la gramática G_3 de la sección 5.5, primero en la notación simple y luego en la FBN. Puesto que G_3 genera expresiones algebraicas, la variable inicial S se representa en la FBN como $\langle \text{Expresión} \rangle$. Por otro lado, las variables T y F se representan mediante $\langle \text{Término} \rangle$ y $\langle \text{Factor} \rangle$, respectivamente. En ambos casos, el alfabeto de terminales es $\Sigma = \{x, y, z, 0, 1, +, *, (,)\}$.

$$\text{Notación simple.} \quad G_3 : \begin{cases} S \rightarrow S + T \mid T \\ T \rightarrow T * F \mid F \\ F \rightarrow x \mid y \mid z \mid 0 \mid 1 \mid (S) \end{cases}$$

$$\text{Forma Backus-Naur.} \quad G_3 : \begin{cases} \langle \text{Expresión} \rangle ::= \langle \text{Expresión} \rangle + \langle \text{Término} \rangle \mid \langle \text{Término} \rangle \\ \langle \text{Término} \rangle ::= \langle \text{Término} \rangle * \langle \text{Factor} \rangle \mid \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle ::= x \mid y \mid z \mid 0 \mid 1 \mid (\langle \text{Expresión} \rangle) \end{cases}$$

Ejemplo En el recuadro de la siguiente página se presenta la gramática G_C que genera comandos para un lenguaje de programación genérico. G_C está escrita en la Forma Backus-Naur y su variable inicial es $\langle \text{Comando} \rangle$, por medio de la cual se pueden generar cuatro tipos de comandos (condicionales, asignaciones, comandos *while* y comandos *begin*). La sintaxis de los comandos condicionales utiliza las expresiones en negrilla **if**, **then** y **else**. Tales expresiones hacen parte del alfabeto de terminales de la gramática. También son terminales las expresiones en negrilla **while**, **do**, **begin** y **end**, así como los símbolos que no aparecen explícitamente encerrados entre paréntesis angulares $\langle \rangle$.

El símbolo de la igualdad tiene tres usos claramente diferenciados; $::=$ se utiliza para presentar las producciones, $:=$ se utiliza en los comandos de asignación y $=$ es uno de los operadores de comparación generados por la variable $\langle \text{Op. Comparación} \rangle$. Nótese también que la variable $\langle \text{Lista-Comandos} \rangle$ genera listas de comandos, separados por el símbolo ‘punto y coma’ (dicho símbolo es un terminal).

Gramática G_C

$\langle \text{Comando} \rangle ::= \langle \text{Condicional} \rangle \mid \langle \text{Comando While} \rangle \mid \langle \text{Asignación} \rangle \mid \langle \text{Comando Begin} \rangle$
 $\langle \text{Condicional} \rangle ::= \text{if } \langle \text{Expresión Booleana} \rangle \text{ then } \langle \text{Comando} \rangle \text{ else } \langle \text{Comando} \rangle$
 $\langle \text{Comando While} \rangle ::= \text{while } \langle \text{Expresión Booleana} \rangle \text{ do } \langle \text{Comando} \rangle$
 $\langle \text{Asignación} \rangle ::= \langle \text{Variable} \rangle := \langle \text{Expres. Aritmética} \rangle$
 $\langle \text{Comando Begin} \rangle ::= \text{begin } \langle \text{Lista-Comandos} \rangle \text{ end}$
 $\langle \text{Lista-Comandos} \rangle ::= \langle \text{Comando} \rangle \mid \langle \text{Comando} \rangle ; \langle \text{Lista-Comandos} \rangle$
 $\langle \text{Expresión Booleana} \rangle ::= \langle \text{Expres. Aritmética} \rangle \langle \text{Op. Comparación} \rangle \langle \text{Expres. Aritmética} \rangle$
 $\langle \text{Op. Comparación} \rangle ::= < \mid > \mid \leq \mid \geq \mid = \mid \neq$
 $\langle \text{Expres. Aritmética} \rangle ::= \langle \text{Variable} \rangle \mid \langle \text{Numeral} \rangle \mid$
 $\quad (\langle \text{Expres. Aritmética} \rangle \langle \text{Op. Aritmético} \rangle \langle \text{Expres. Aritmética} \rangle)$
 $\langle \text{Numeral} \rangle ::= \langle \text{Numeral} \rangle \langle \text{Numeral} \rangle \mid 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
 $\langle \text{Op. Aritmético} \rangle ::= + \mid - \mid * \mid /$
 $\langle \text{Variable} \rangle ::= a \mid b \mid c \mid d \mid e \mid f \mid g \mid i \mid j \mid m \mid n \mid p \mid q \mid r \mid s \mid t \mid u \mid v \mid w \mid x \mid y \mid z$

Como ilustración, generamos a continuación el comando: **while** $x > 1$ **do** $y := (x + 2)$.

$\langle \text{Comando} \rangle \implies \langle \text{Comando While} \rangle \implies \text{while } \langle \text{Expresión Booleana} \rangle \text{ do } \langle \text{Comando} \rangle$
 $\xRightarrow{2} \text{while } \langle \text{Expres. Aritmética} \rangle \langle \text{Op. Comparación} \rangle \langle \text{Expres. Aritmética} \rangle \text{ do } \langle \text{Asignación} \rangle$
 $\xRightarrow{4} \text{while } \langle \text{Variable} \rangle > \langle \text{Numeral} \rangle \text{ do } \langle \text{Variable} \rangle := \langle \text{Expres. Aritmética} \rangle$
 $\xRightarrow{4} \text{while } x > 1 \text{ do } y := (\langle \text{Expres. Aritmética} \rangle \langle \text{Op. Aritmético} \rangle \langle \text{Expres. Aritmética} \rangle)$
 $\xRightarrow{3} \text{while } x > 1 \text{ do } y := (\langle \text{Variable} \rangle + \langle \text{Numeral} \rangle) \xRightarrow{2} \text{while } x > 1 \text{ do } y := (x + 2).$

Ejercicios de la sección 5.7

① Generar los siguientes comandos con la gramática G_C :

- (i) **if** $x > (y + 1)$ **then** $z := (x + 1)$ **else** $z := (x - 1)$.
- (ii) **begin if** $z = (y + 5)$ **then** $x := z$ **else** $z := (x + 1)$ **end**.
- (iii) **while** $x \leq y$ **do begin** $x := (x + 1); y := (y - 1)$ **end**.
- (iv) **if** $x > 1$ **then while** $y < 0$ **do** $z := (x + y)$ **else begin** $z := (x + 2); y := (x - 2)$ **end**.
- (v) **begin if** $y \neq 1$ **then while** $x \geq 0$ **do** $z := (x + y)$ **else begin** $u := x; v := y$ **end; z := (u/(v + y)) end**.

② La siguiente gramática G' es una modificación de la gramática G_C presentada en la presente sección, con dos tipos de comandos condicionales.

Gramática G'

$\langle \text{Comando} \rangle ::= \langle \text{Comando Begin} \rangle \mid \langle \text{Condicional} \rangle \mid \langle \text{Condicional Alternativo} \rangle \mid \langle \text{Asignación} \rangle$
 $\langle \text{Comando Begin} \rangle ::= \mathbf{begin} \langle \text{Lista-Comandos} \rangle \mathbf{end}$
 $\langle \text{Lista-Comandos} \rangle ::= \langle \text{Comando} \rangle \mid \langle \text{Comando} \rangle ; \langle \text{Lista-Comandos} \rangle$
 $\langle \text{Condicional} \rangle ::= \mathbf{if} \mathbf{T} \mathbf{then} \langle \text{Comando} \rangle$
 $\langle \text{Condicional Alternativo} \rangle ::= \mathbf{if} \mathbf{T} \mathbf{then} \langle \text{Comando} \rangle \mathbf{else} \langle \text{Comando} \rangle$
 $\langle \text{Asignación} \rangle ::= x := \langle \text{Numeral} \rangle$
 $\langle \text{Numeral} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

En esta gramática el alfabeto Σ de terminales contiene todos los símbolos que no aparecen encerrados entre paréntesis angulares $\langle \rangle$; en particular, el símbolo $:=$ y el símbolo ‘punto y coma’. Explícitamente,

$$\Sigma = \{\mathbf{begin}, \mathbf{end}, \mathbf{if} \mathbf{T} \mathbf{then}, \mathbf{else}, x, :=, ;, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}.$$

- (i) Demostrar que la gramática G' es ambigua.
- (ii) Encontrar una gramática no-ambigua que genere el mismo lenguaje generado por G' .

5.8. Transformación de gramáticas

Es posible modificar o eliminar producciones en una gramática dada G de tal manera que G se transforme en una gramática equivalente G' , esto es, sin alterar el lenguaje generado: $L(G) = L(G')$. En esta sección veremos que producciones de la forma $A \rightarrow \lambda$ o de la forma $A \rightarrow B$ no son en realidad necesarias y se pueden eliminar transformando adecuadamente la gramática original.

5.8.1. Eliminación de las producciones λ

5.8.1. Definiciones. Sea $G = (V, \Sigma, S, P)$ una GIC.

- (i) Una producción de G que tenga la forma $A \rightarrow \lambda$ se llama *producción λ* .
- (ii) Una variable A de G se dice que es *anulable* si $A \xRightarrow{*} \lambda$.

Es decir, A es anulable si se puede transformar en la cadena vacía λ utilizando las producciones de G . El conjunto de todas las variables anulables de G se denota como **ANUL**, o sea,

$$\mathbf{ANUL} = \{A \in V : A \xRightarrow{*} \lambda\}.$$

Hay un algoritmo muy sencillo para hallar **ANUL**, presentado a continuación.

Algoritmo para encontrar las variables anulables de una gramática G **INICIALIZAR:****ANUL** := $\{A \in V : A \rightarrow \lambda \text{ es una producción de } G\}$.**REPETIR:**

Examinar una por una las producciones de G buscando producciones de la forma $B \rightarrow A_1 A_2 \cdots A_k$, donde $A_1, A_2, \dots, A_k \in \mathbf{ANUL}$. Para cada producción de esa forma añadir B a **ANUL**.

HASTA:No se añaden nuevas variables a **ANUL**.

El siguiente teorema establece que las producciones λ se pueden eliminar de cualquier gramática G transformando adecuadamente G en una gramática G' equivalente. La única producción λ que no se puede eliminar es $S \rightarrow \lambda$, en el caso en el que $\lambda \in L(G)$.

5.8.2 Teorema. Dada una GIC G , se puede construir una GIC G' equivalente a G sin producciones λ , excepto (posiblemente) $S \rightarrow \lambda$.

Demostración. Una vez que se haya encontrado el conjunto **ANUL** de variables anulables, por medio del algoritmo anterior, las producciones de λ se pueden eliminar (excepto $S \rightarrow \lambda$) añadiendo nuevas producciones que simulen el efecto de las producciones λ eliminadas. Más concretamente, por cada producción $A \rightarrow u$ de G se añaden *todas* las producciones de la forma $A \rightarrow v$ obtenidas suprimiendo de la cadena u una, dos o más variables anulables presentes, de todas las formas posibles. La gramática G' así obtenida es equivalente a la gramática original G , es decir, $L(G) = L(G')$. \square

Ejemplo Eliminar las producciones λ de la siguiente gramática G . Más precisamente, encontrar una gramática G' sin producciones λ , excepto (posiblemente) $S \rightarrow \lambda$, de tal manera que $L(G) = L(G')$.

$$G : \begin{cases} S \rightarrow AB \mid ACA \mid ab \\ A \rightarrow aAa \mid B \mid CD \\ B \rightarrow bB \mid bA \\ C \rightarrow cC \mid \lambda \\ D \rightarrow aDc \mid CC \mid ABb \end{cases}$$

Solución. Primero encontramos el conjunto **ANUL** de todas las variables anulables de G por medio del mencionado algoritmo. Inicialmente **ANUL** = $\{C\}$ ya que C es la única variable con una producción λ , a saber, $C \rightarrow \lambda$. En las subsiguientes iteraciones se

actualiza el conjunto **ANUL** como se indica a continuación:

ANUL = $\{C, D\}$, por la producción $D \rightarrow CC$.

ANUL = $\{C, D, A\}$, por la producción $A \rightarrow CD$.

ANUL = $\{C, D, A, S\}$, por la producción $S \rightarrow ACA$.

Así obtenemos que **ANUL** = $\{C, D, A, S\}$. Al eliminar de G las producciones λ (la única es $C \rightarrow \lambda$) se añaden nuevas producciones simuladoras, como se precisó en la demostración del Teorema 5.8.2, y se obtiene finalmente la siguiente gramática G' equivalente a G :

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

5.8.2. Eliminación de las producciones unitarias

Sea $G = (V, \Sigma, S, P)$ una GIC. Una producción de la forma $A \rightarrow B$ donde A y B son variables, se llama *producción unitaria*. El siguiente teorema establece que las producciones unitarias se pueden eliminar de cualquier gramática G transformando adecuadamente G en una gramática G' equivalente. La idea básica es que una producción de la forma $A \rightarrow B$ solo sirve de “puente” entre A y B , y se puede eliminar añadiendo a las producciones de A todas las de B . Sin embargo, hay que tener en cuenta que la gramática original puede tener muchas producciones unitarias secuencialmente encadenadas.

En la demostración del teorema se utiliza el llamado *grafo de dependencia* de G , en el cual los vértices son las variables y hay un arco entre los vértices C y D cuando $C \rightarrow D$ es una producción de G .

5.8.3 Teorema. Dada una GIC G , se puede construir una GIC G' sin producciones unitarias equivalente a G .

Demostración. Es claro que una producción unitaria de la forma $A \rightarrow A$ se puede eliminar sin alterar el lenguaje generado. Para todo par A, B de variables diferentes se trata de determinar si $A \xRightarrow{*} B$ por medio de producciones unitarias. La manera más sencilla de conseguirlo es por medio del grafo de dependencia de G ; se deduce que $A \xRightarrow{*} B$, por medio de una derivación que utilice únicamente producciones unitarias, si y solamente si en el grafo hay una trayectoria desde A hasta B .

Para obtener G' primero se mantienen las producciones no-unitarias de G . A continuación, para cada par de variables A, B tales que $A \xRightarrow{*} B$, se añaden las producciones de B a las de A , es decir, se añaden las producciones

$$A \rightarrow v_1 \mid v_2 \mid \cdots \mid v_k,$$

siempre que $B \rightarrow v_1 \mid v_2 \mid \cdots \mid v_k$ sean las producciones (no-unitarias) de B en G . De esta manera, las producciones unitarias de G desaparecen y $L(G) = L(G')$. \square

Ejemplo Eliminar las producciones unitarias de la siguiente gramática.

$$G : \begin{cases} S \rightarrow AS \mid AA \mid BA \mid \lambda \\ A \rightarrow aA \mid a \\ B \rightarrow bB \mid cC \mid C \\ C \rightarrow aA \mid bA \mid B \mid ab \end{cases}$$

Solución. El grafo de dependencia de G , considerando únicamente producciones unitarias es:



Claramente se tiene que $B \xRightarrow{*} C$ y $C \xRightarrow{*} B$.

En la nueva gramática G' mantenemos inicialmente las producciones no-unitarias de G :

$$G' : \begin{cases} S \rightarrow AS \mid AA \mid BA \mid \lambda \\ A \rightarrow aA \mid a \\ B \rightarrow bB \mid cC \\ C \rightarrow aA \mid bA \mid ab \end{cases}$$

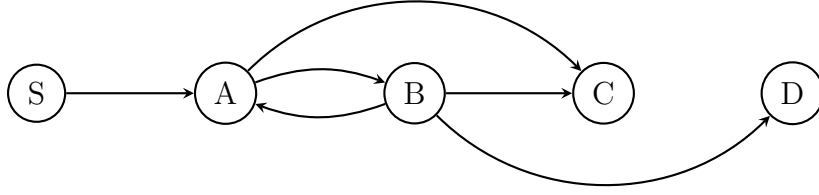
A las producciones de B añadimos las producciones de C , y a las producciones de C añadimos las de B , obteniendo así una gramática G' equivalente:

$$G' : \begin{cases} S \rightarrow AS \mid AA \mid BA \mid \lambda \\ A \rightarrow aA \mid a \\ B \rightarrow bB \mid cC \mid aA \mid bA \mid ab \\ C \rightarrow aA \mid bA \mid ab \mid bB \mid cC \end{cases}$$

Ejemplo Eliminar las producciones unitarias de la siguiente gramática.

$$G : \begin{cases} S \rightarrow ACA \mid CA \mid ADA \mid A \mid \lambda \\ A \rightarrow aAa \mid aa \mid B \mid C \\ B \rightarrow cC \mid D \mid C \\ C \rightarrow bC \\ D \rightarrow aA \mid \lambda \end{cases}$$

Solución. Se obtiene el grafo de dependencia de G , considerando únicamente producciones unitarias:



A partir del grafo se deduce que, utilizando únicamente producciones unitarias,

$$(5.8.1) \quad \begin{aligned} S &\xRightarrow{*} A, S \xRightarrow{*} B, S \xRightarrow{*} C, S \xRightarrow{*} D, \\ A &\xRightarrow{*} B, A \xRightarrow{*} C, A \xRightarrow{*} D, \\ B &\xRightarrow{*} A, B \xRightarrow{*} C, B \xRightarrow{*} D \end{aligned}$$

En la nueva gramática G' mantenemos inicialmente las producciones no-unitarias de G :

$$G' : \begin{cases} S \rightarrow ACA \mid CA \mid ADA \mid \lambda \\ A \rightarrow aAa \mid aa \\ B \rightarrow cC \\ C \rightarrow bC \\ D \rightarrow aA \mid \lambda \end{cases}$$

Teniendo en cuenta (5.8.1), a las producciones de S añadimos las producciones de las variables A, B, C y D :

$$S \rightarrow ACA \mid CA \mid ADA \mid \lambda \mid aAa \mid aa \mid cC \mid bC \mid aA.$$

A las producciones de A añadimos las de las variables B, C y D :

$$A \rightarrow aAa \mid aa \mid cC \mid bC \mid aA.$$

Finalmente, a las producciones de B añadimos las de las variables A, C y D :

$$B \rightarrow cC \mid bC \mid aA.$$

Se obtiene así la gramática G' sin producciones unitarias, equivalente a G :

$$G' : \begin{cases} S \rightarrow ACA \mid CA \mid ADA \mid \lambda \mid aAa \mid aa \mid bC \mid cC \mid aA \\ A \rightarrow aAa \mid aa \mid cC \mid bC \mid aA \mid \lambda \\ B \rightarrow cC \mid bC \mid aA \mid \lambda \\ C \rightarrow bC \\ D \rightarrow aA \mid \lambda \end{cases}$$

5.9. Forma Normal de Chomsky (FNC)

Una GIC $G = (V, \Sigma, S, P)$ está en *Forma Normal de Chomsky* (FNC) si todas sus producciones son de la forma: $A \rightarrow BC$ (donde B y C son variables, no necesariamente distintas) ó $A \rightarrow a$ (con $a \in \Sigma$). Las producciones de la forma $A \rightarrow BC$ se denominan *producciones binarias*, y las de la forma $A \rightarrow a$ se llaman *producciones simples*. La única producción λ permitida en la FNC es $S \rightarrow \lambda$, para el caso específico en el que $\lambda \in L(G)$.

5.9.1 Teorema (Procedimiento de conversión a FNC). Toda GIC G es equivalente a una gramática en Forma Normal de Chomsky.

Demostración. Podemos transformar G en una gramática en FNC, equivalente a G , mediante el siguiente procedimiento:

1. Eliminar las producciones λ (excepto, posiblemente, $S \rightarrow \lambda$).
2. Eliminar las producciones unitarias.
3. Las producciones resultantes (diferentes de $S \rightarrow \lambda$) son de la forma: $A \rightarrow a$ ó $A \rightarrow w$, donde $|w| \geq 2$. Estas últimas se pueden simular con producciones de la forma $A \rightarrow BC$ o $A \rightarrow a$. Se introduce primero, para cada $a \in \Sigma$, una variable nueva T_a cuya única producción es $T_a \rightarrow a$. A continuación, se introducen nuevas variables, con producciones binarias, para simular las producciones deseadas. \square

Los pasos 1 y 2 del procedimiento no se pueden invertir ya que al eliminar las producciones λ pueden aparecer nuevas producciones unitarias. La parte 3 del procedimiento anterior se ilustra en los dos siguientes ejemplos.

Ejemplo Simular la producción $A \rightarrow abBaC$ mediante producciones binarias y simples.

Solución. Introducimos las variables T_a y T_b , y las producciones $T_a \rightarrow a$ y $T_b \rightarrow b$. Entonces $A \rightarrow abBaC$ se simula con:

$$\begin{cases} A \rightarrow T_a T_b B T_a C \\ T_a \rightarrow a \\ T_b \rightarrow b \end{cases}$$

Ahora introducimos nuevas variables T_1, T_2, T_3 y las producciones binarias necesarias. Las únicas producciones de estas nuevas variables son las mostradas:

$$\begin{cases} A \rightarrow T_a T_1 \\ T_1 \rightarrow T_b T_2 \\ T_2 \rightarrow B T_3 \\ T_3 \rightarrow T_a C \\ T_a \rightarrow a \\ T_b \rightarrow b \end{cases}$$

Ejemplo Simular la producción $A \rightarrow BAaCbb$ mediante producciones binarias y simples.

Solución. Introducimos las variables T_a y T_b , y las producciones $T_a \rightarrow a$ y $T_b \rightarrow b$. Entonces $A \rightarrow BAaCbb$ se simula con:

$$\begin{cases} A \rightarrow BAT_aCT_bT_b \\ T_a \rightarrow a \\ T_b \rightarrow b \end{cases}$$

Ahora introducimos nuevas variables T_1, T_2, T_3, T_4 y las producciones binarias necesarias. Las únicas producciones de estas nuevas variables son las mostradas:

$$\begin{cases} A \rightarrow BT_1 \\ T_1 \rightarrow AT_2 \\ T_2 \rightarrow T_aT_3 \\ T_3 \rightarrow CT_4 \\ T_4 \rightarrow T_bT_b \\ T_a \rightarrow a \\ T_b \rightarrow b \end{cases}$$

En el siguiente ejemplo se ilustra el procedimiento completo para convertir una gramática dada a la Forma Normal de Chomsky (FNC).

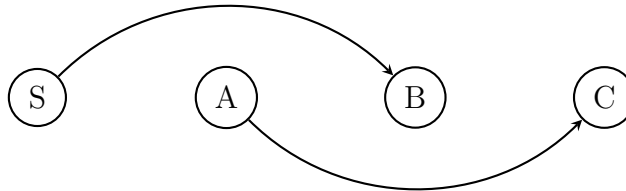
Ejemplo Encontrar una GIC en FNC equivalente a la siguiente a la gramática G .

$$G : \begin{cases} S \rightarrow AB \mid aBC \mid SBS \\ A \rightarrow aA \mid C \\ B \rightarrow bbB \mid b \\ C \rightarrow cC \mid \lambda \end{cases}$$

Solución. El conjunto de variables anulables es **ANUL** = $\{C, A\}$. Al eliminar las producciones λ de G (la única es $C \rightarrow \lambda$) se obtiene la gramática equivalente G_1 :

$$G_1 : \begin{cases} S \rightarrow AB \mid aBC \mid SBS \mid B \mid aB \\ A \rightarrow aA \mid C \mid a \\ B \rightarrow bbB \mid b \\ C \rightarrow cC \mid c \end{cases}$$

El grafo de dependencia, correspondiente a las producciones unitarias es:



Es decir, con producciones unitarias, $S \xRightarrow{*} B$ y $A \xRightarrow{*} C$. Al eliminar las producciones unitarias obtenemos la gramática equivalente G_2 :

$$G_2 : \begin{cases} S \rightarrow AB \mid aBC \mid SBS \mid aB \mid bbB \mid b \\ A \rightarrow aA \mid a \mid cC \mid c \\ B \rightarrow bbB \mid b \\ C \rightarrow cC \mid c \end{cases}$$

Luego introducimos las variables nuevas T_a , T_b y T_c , y las producciones $T_a \rightarrow a$, $T_b \rightarrow b$ y $T_c \rightarrow c$ con el propósito de que todas las producciones sean unitarias o de la forma $A \rightarrow w$, donde $|w| \geq 2$.

$$G_3 : \begin{cases} S \rightarrow AB \mid T_aBC \mid SBS \mid T_aB \mid T_bT_bB \mid b \\ A \rightarrow T_aA \mid a \mid T_cC \mid c \\ B \rightarrow T_bT_bB \mid b \\ C \rightarrow T_cC \mid c \\ T_a \rightarrow a \\ T_b \rightarrow b \\ T_c \rightarrow c \end{cases}$$

Finalmente, se introducen nuevas variables, con producciones binarias, para simular las producciones de la forma $A \rightarrow w$, donde $|w| \geq 2$:

$$G_4 : \begin{cases} S \rightarrow AB \mid T_aT_1 \mid ST_2 \mid T_aB \mid T_bT_3 \mid b \\ A \rightarrow T_aA \mid T_cC \mid a \mid c \\ B \rightarrow T_bT_3 \mid b \\ C \rightarrow T_cC \mid c \\ T_1 \rightarrow BC \\ T_2 \rightarrow BS \\ T_3 \rightarrow T_bB \\ T_a \rightarrow a \\ T_b \rightarrow b \\ T_c \rightarrow c \end{cases}$$

5.9.2 Definición. Una variable se llama recursiva si tiene una producción de la forma:

$$A \rightarrow \alpha A \beta, \quad \alpha, \beta \in (V \cup \Sigma)^*.$$

En otras palabras, A es recursiva si A aparece en el cuerpo de alguna producción de A .

Para caracterizar los lenguajes generados, es necesario exigir en algunas situaciones que la variable inicial S no sea recursiva. El siguiente teorema es un resultado muy sencillo; establece que cualquier GIC se puede transformar en una GIC equivalente en la cual la variable inicial no es recursiva.

5.9.3 Teorema. Dada una GIC $G = (V, \Sigma, S, P)$ se puede construir una GIC $G' = (V', \Sigma, S', P')$ equivalente a G de tal manera que el símbolo inicial S' de G' no aparezca en lado derecho de las producciones de G' .

Demostración. La nueva gramática G' tiene una variable más que G , la variable S' , que actúa como la nueva variable inicial. Es decir, $V' = V \cup \{S'\}$. El conjunto de producciones P' está dado por $P' = P \cup \{S' \rightarrow S\}$. Es claro que $L(G) = L(G')$ y el símbolo inicial S' no aparece en el cuerpo de las producciones. \square

Ejemplo Encontrar una GIC G' equivalente a la siguiente gramática G de tal manera que la variable inicial de G' no sea recursiva.

$$G : \begin{cases} S \rightarrow ASB \mid BA \\ A \rightarrow aA \mid a \\ B \rightarrow bBS \mid \lambda \end{cases}$$

Solución. Según se indicó en la demostración del Teorema 5.9.3, la gramática pedida G' es

$$G' : \begin{cases} S' \rightarrow S \\ S \rightarrow ASB \mid BA \\ A \rightarrow aA \mid a \\ B \rightarrow bBS \mid \lambda \end{cases}$$

Nótese que S sigue siendo recursiva pero ya no es la variable inicial de la gramática.

Ejercicios de la sección 5.9

- ① Eliminar las producciones λ de la siguiente gramática G :

$$G : \begin{cases} S \rightarrow BCB \\ A \rightarrow aA \mid ab \\ B \rightarrow bBa \mid A \mid DC \\ C \rightarrow aCb \mid D \mid b \\ D \rightarrow aB \mid \lambda \end{cases}$$

- ② Eliminar las producciones λ de la siguiente gramática G :

$$G : \begin{cases} S \rightarrow EA \mid SaBb \mid aEb \\ A \rightarrow DaD \mid bD \mid BEB \\ B \rightarrow bB \mid Ab \mid \lambda \\ D \rightarrow aEb \mid ab \\ E \rightarrow aA \mid bB \mid \lambda \end{cases}$$

- ③ Eliminar las producciones unitarias de la siguiente gramática G :

$$G : \begin{cases} S \rightarrow Ba \mid A \mid \lambda \\ A \rightarrow Aa \mid a \\ B \rightarrow bB \mid S \end{cases}$$

- ④ Eliminar las producciones unitarias de la siguiente gramática G :

$$G : \begin{cases} S \rightarrow BBa \mid A \mid B \mid ab \mid \lambda \\ A \rightarrow Aa \mid B \mid D \mid aC \\ B \rightarrow bB \mid aA \mid b \\ C \rightarrow ABb \mid A \mid aB \\ D \rightarrow cC \mid c \end{cases}$$

- ⑤ Eliminar las producciones unitarias de la siguiente gramática G :

$$G : \begin{cases} S \rightarrow ACA \mid ab \mid B \mid CA \mid A \mid C \mid \lambda \\ A \rightarrow aAa \mid B \mid CD \mid aa \mid D \\ B \rightarrow bB \mid bA \mid b \\ C \rightarrow cC \mid c \\ D \rightarrow ABb \mid ac \mid C \mid Bb \end{cases}$$

- ⑥ Encontrar una gramática en FNC equivalente a la siguiente gramática G :

$$G : \begin{cases} S \rightarrow ABC \mid BaC \mid aB \\ A \rightarrow Aa \mid a \\ B \rightarrow BAB \mid bab \\ C \rightarrow cC \mid c \end{cases}$$

- ⑦ Encontrar una gramática en FNC equivalente a la siguiente gramática G :

$$G : \begin{cases} S \rightarrow aASb \mid BAb \\ A \rightarrow Aa \mid a \mid \lambda \\ B \rightarrow BAB \mid bAb \\ C \rightarrow cCS \mid \lambda \end{cases}$$


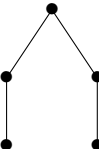
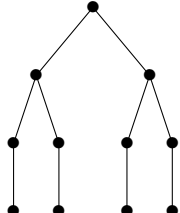
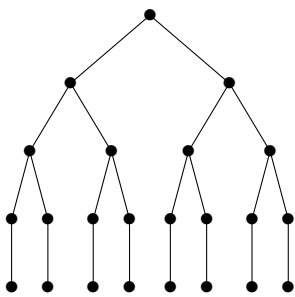
- ⑧ Para la gramática del ejercicio ⑦ encontrar una GIC equivalente en FNC, de tal manera que su variable inicial no sea recursiva.

5.10. Lema de bombeo para los lenguajes LIC

El Lema de bombeo para los lenguajes LIC, análogo al Lema de bombeo para lenguajes regulares, es la herramienta más importante para demostrar que ciertos lenguajes no son LIC. La demostración más simple del lema se obtiene con gramáticas en Forma Normal de Chomsky (FNC), cuyos árboles sintácticos son binarios.

5.10.1 Teorema. Sea $G = (V, \Sigma, S, P)$ una gramática en FNC y $z \in \Sigma^*$. Si la trayectoria más larga (desde la raíz) en el árbol de una derivación $S \xRightarrow{*} z$ tiene k (o menos) nodos, con $k \geq 2$, entonces $|z| \leq 2^{k-2}$.

Demostración. La siguiente tabla muestra las relaciones obtenidas entre $k =$ número de nodos de la trayectoria más larga de $S \xRightarrow{*} z$ y la longitud de z , en los casos $k = 2$, $k = 3$, $k = 4$ y $k = 5$. En la tabla se muestran los casos extremos, es decir, los árboles con el mayor número posible de nodos. Se observa que $|z| \leq 2^{k-2}$. Una demostración rigurosa del caso general se hace por inducción sobre k . \square

$k =$ número de nodos de la trayectoria más larga	Árbol de derivación	Longitud de z
$k = 2$		$ z = 1 = 2^0 = 2^{k-2}$
$k = 3$		$ z \leq 2 = 2^1 = 2^{k-2}$
$k = 4$		$ z \leq 4 = 2^2 = 2^{k-2}$
$k = 5$		$ z \leq 8 = 2^3 = 2^{k-2}$

Se puede observar que el resultado de este teorema también se aplica a los árboles de las derivaciones de la forma $A \xRightarrow{*} z$, donde A es un nodo interior.

5.10.2 Corolario. Sea $G = (V, \Sigma, S, P)$ una gramática en FNC y $z \in \Sigma^*$.

- (1) Si la trayectoria más larga en el árbol de una derivación $S \xRightarrow{*} z$ tiene $k+2$ (o menos) nodos, entonces $|z| \leq 2^k$. Aquí $k \geq 0$.
- (2) Si $|z| > 2^k$ (con $k \geq 0$) entonces la trayectoria más larga en un árbol de derivación de $S \xRightarrow{*} z$ tiene más de $k+2$ nodos.

Demostración.

(1) Se sigue inmediatamente del Teorema 5.10.1.

(2) Es la afirmación contra-recíproca de la parte (1). □

5.10.3. Lema de bombeo para LIC. Dado un LIC L , existe una constante $n \geq 2$ (llamada constante de bombeo para L) tal que toda $z \in L$ con $|z| > n$ se puede descomponer en la forma $z = uvwxy$ donde:

- (1) $uv^iwx^iy \in L$ para todo $i \geq 0$.
- (2) $v \neq \lambda$ ó $x \neq \lambda$.
- (3) $|vwx| \leq n$.

Demostración. Sea $G = (V, \Sigma, S, P)$ una gramática en FNC, con variable inicial no recursiva, tal que $L(G) = L$. Tal gramática existe por los Teoremas 5.9.1 y 5.9.3. Sea $k = |V|$ = número de variables de G y $n = 2^k$. Sea $z \in L$ con $|z| > n = 2^k$. Por la parte (2) del Corolario 5.10.2, la trayectoria más larga en el árbol de una derivación $S \xRightarrow{*} z$ tiene más de $k+2$ nodos. El último nodo de tal trayectoria es un terminal y los restantes nodos son variables. Como hay sólo k variables en la gramática, entonces hay por lo menos una variable repetida en la trayectoria. Sea A la última variable que se repite (desde la raíz). Por lo tanto, existen cadenas $u, v, w, x, y \in \Sigma^*$ tales que

$$S \xRightarrow{*} uAy, \quad A \xRightarrow{*} vAx, \quad A \xRightarrow{*} w.$$

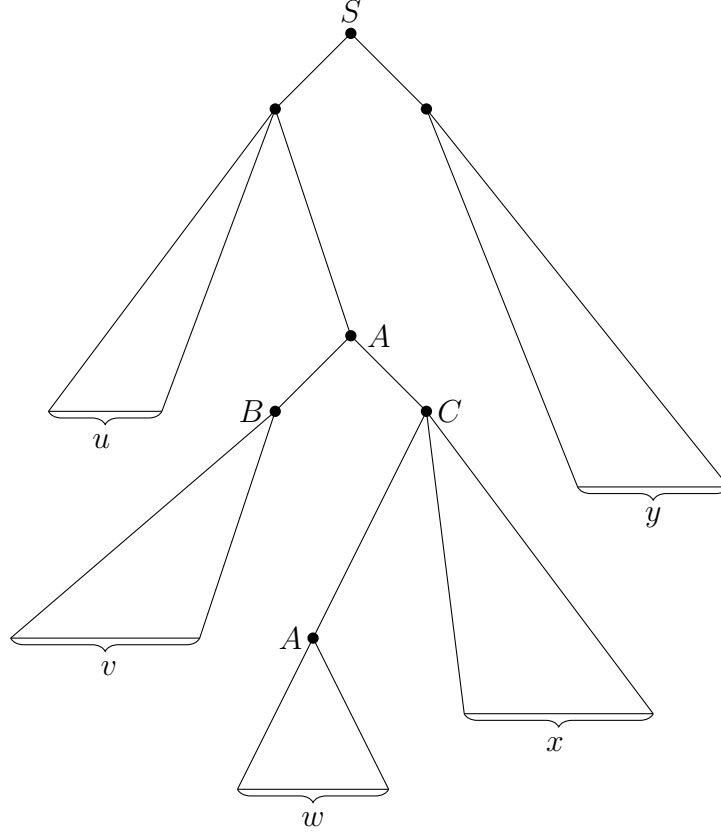
Así que

$$S \xRightarrow{*} uAy \xRightarrow{*} uvAxy \xRightarrow{*} uvwxy = z.$$

La gráfica exhibida en la siguiente página ilustra la situación. Se tiene que

$$S \xRightarrow{*} uAy \xRightarrow{*} uvAxy \xRightarrow{*} uv^iAx^iy \xRightarrow{*} uv^iwx^iy, \quad \text{para todo } i \geq 1,$$

lo cual demuestra la propiedad (1). Obsérvese que el caso $i = 0$ corresponde a la derivación $S \xRightarrow{*} uAy \xRightarrow{*} uwy$.



Consideremos el árbol cuya raíz es A , correspondiente a la derivación

$$A \xRightarrow{*} vAx \xRightarrow{*} vwx.$$

La trayectoria más larga en este árbol tiene $k + 2$ nodos o menos, lo cual implica, por la parte (1) del Corolario 5.10.2, que $|vwx| \leq 2^k = n$. Esto demuestra la propiedad (2).

Para demostrar la propiedad (3), la derivación $A \xRightarrow{*} vAx$ se puede escribir como

$$A \Rightarrow BC \xRightarrow{*} vAx$$

utilizando una producción de la forma $A \rightarrow BC$ como primer paso (véase la gráfica). Se deduce que v y x no pueden ser ambas λ porque se tendría $BC \xRightarrow{*} A$, lo cual es imposible en una gramática en FNC (ya que la única producción λ en la gramática es, posiblemente, $S \rightarrow \lambda$; pero S no aparece en el cuerpo de ninguna producción de G dado que S no es recursiva). Se deduce entonces que $v \neq \lambda$ ó $x \neq \lambda$. \square

Ejemplo

Demostrar que el lenguaje $L = \{a^i b^i c^i : i \geq 0\}$ sobre $\Sigma = \{a, b, c\}$ no es un LIC.

Solución. Razonamiento por contradicción. Si L fuera LIC, por el lema de bombeo, existiría una constante de bombeo n . Sea $z = a^n b^n c^n$; se tiene que $z \in L$ y $|z| > n$. Por lo tanto, z se puede descomponer como $z = a^n b^n c^n = uvwxy$ con las propiedades (1), (2) y (3) del lema de bombeo. Puesto que $|vwx| \leq n$, en la cadena vwx no pueden aparecer los tres terminales a , b y c simultáneamente (para que aparezcan los tres terminales simultáneamente, una subcadena de $a^n b^n c^n$ debe tener longitud $\geq n + 2$). Como $v \neq \lambda$ ó $x \neq \lambda$, se distinguen dos casos:

- Caso 1. Alguna de las cadenas v ó x contiene dos tipos de terminales. Entonces en uv^2wx^2y aparecen algunas *bes* seguidas de *aes* o algunas *ces* seguidas de *bes*. En cualquier caso, $uv^2wx^2y \notin L$.
- Caso 2. Las cadenas v y x contienen un sólo tipo de terminal cada una (o sólo *aes* o sólo *bes* o sólo *ces*). Como en vwx no aparecen los tres terminales a , b y c simultáneamente, en la cadena bombeada uv^2wx^2y se altera el número de dos de los terminales a, b, c , a lo sumo, pero no de los tres. Por lo tanto, $uv^2wx^2y \notin L$.

Pero el lema de bombeo afirma que $uv^2wx^2y \in L$. Esta contradicción muestra que L no es un LIC.

Ejemplo Demostrar que el lenguaje $L = \{a^i : i \text{ es primo}\}$ sobre $\Sigma = \{a\}$ no es un LIC.

Solución. Razonamiento por contradicción. Si L fuera LIC, por el lema de bombeo, existiría una constante de bombeo n . Sea $z = a^p$ donde p es un número primo $p > n$ (p existe porque el conjunto de los números primos es infinito). Entonces $z \in L$ y $|z| > n$. Por lo tanto, z se puede descomponer como $z = a^p = uvwxy$ con las propiedades (1), (2) y (3) del lema de bombeo.

Por la propiedad (1) del lema de bombeo, $uv^iwx^iy \in L$ para todo $i \geq 0$; es decir, $|uv^iwx^iy|$ es primo para todo $i \geq 0$. Pero cuando $i = p + 1$ se tiene

$$|uv^{p+1}wx^{p+1}y| = |uvwxy| + |v^p x^p| = |a^p| + p|v| + p|x| = p + p|vx| = p(1 + |vx|).$$

Por la propiedad (3), $|vx| \geq 1$ y como $p \geq 2$ se tiene que $p(1 + |vx|)$ es el producto de dos números naturales ≥ 2 y, por consiguiente, no es primo. Esta contradicción muestra que L no es un LIC.

Ejercicios de la sección 5.10

Utilizar el lema de bombeo para demostrar que los siguientes lenguajes no son LIC:

- ① $L = \{a^i b^j c^j : j > i\}$, sobre $\Sigma = \{a, b, c\}$.
- ② $L = \{a^i b^j c^k : 1 < i < j < k\}$, sobre $\Sigma = \{a, b, c\}$.
- ③ $L = \{0^i 1^{2i} 0^i : i \geq 1\}$, sobre $\Sigma = \{0, 1\}$.
- ④ $L = \{a^i b^i c^i d^i : i \geq 0\}$, sobre $\Sigma = \{a, b, c, d\}$.

- ⑤ $L = \{a^i b^j c^i d^j : i, j \geq 0\}$, sobre $\Sigma = \{a, b, c, d\}$.
- ⑥ $L = \{ww : w \in \{a, b\}^*\}$.
- ⑦ $L = \{ww^R w : w \in \{a, b\}^*\}$.
- ⑧ $L = \{a^i : i \text{ es un cuadrado perfecto}\}$, sobre $\Sigma = \{a\}$.
- ⑨ $L = \{a^i : i \text{ es una potencia de 2}\}$, sobre $\Sigma = \{a\}$.
- ⑩ $L = \{a^i : i \text{ es una potencia de 3}\}$, sobre $\Sigma = \{a\}$.

5.11. Propiedades de clausura de los LIC

En el Capítulo 2 se vio que los lenguajes regulares son cerrados bajo la unión, la concatenación, la estrella de Kleene y todas las operaciones booleanas (intersección, diferencia y complemento). Los LIC poseen propiedades de clausura mucho más restringidas: son cerrados para las operaciones regulares (Teorema 5.11.1) pero, en general, no son cerrados para intersección, complementos ni diferencias (Teorema 5.11.2).

5.11.1 Teorema. La colección de los lenguajes independientes del contexto es cerrada para las operaciones regulares (unión, concatenación y estrella de Kleene). Es decir, dadas GIC $G_1 = (V_1, \Sigma, S_1, P_1)$ y $G_2 = (V_2, \Sigma, S_2, P_2)$ tales que $L(G_1) = L_1$ y $L(G_2) = L_2$, se pueden construir GIC que generen los lenguajes $L_1 \cup L_2$, $L_1 L_2$ y L_1^* , respectivamente.

Demostración. Sin pérdida de generalidad, podemos suponer que G_1 y G_2 no tienen variables en común (en caso contrario, simplemente cambiamos los nombres de las variables). Para construir una GIC G que genere $L_1 \cup L_2$ introducimos una variable *nueva* S , la variable inicial de G , junto con las producciones $S \rightarrow S_1$ y $S \rightarrow S_2$. Las producciones de G_1 y G_2 se mantienen. Concretamente,

$$G = (V_1 \cup V_2 \cup \{S\}, \Sigma, S, P_1 \cup P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}).$$

Esquemáticamente, G tiene el siguiente aspecto:

$$\begin{array}{lcl} S & \rightarrow & S_1 \mid S_2 \\ \left. \begin{array}{l} S_1 \rightarrow \cdots \\ \vdots \quad \quad \vdots \end{array} \right\} & & \text{producciones de } G_1 \\ \left. \begin{array}{l} S_2 \rightarrow \cdots \\ \vdots \quad \quad \vdots \end{array} \right\} & & \text{producciones de } G_2 \end{array}$$

Claramente, $L(G) = L_1 \cup L_2$.

Una GIC G que genere $L_1 L_2$ se construye similarmente, añadiendo la producción $S \rightarrow S_1 S_2$. Es decir,

$$G = (V_1 \cup V_2 \cup \{S\}, \Sigma, S, P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\}).$$

Esquemáticamente, G es la gramática:

$$\left. \begin{array}{l} S \rightarrow S_1 S_2 \\ S_1 \rightarrow \cdots \\ \vdots \quad \quad \vdots \end{array} \right\} \text{ producciones de } G_1$$

$$\left. \begin{array}{l} S_2 \rightarrow \cdots \\ \vdots \quad \quad \vdots \end{array} \right\} \text{ producciones de } G_2$$

Claramente, $L(G) = L_1 L_2$.

Para generar L_1^* se define G como

$$G = (V_1 \cup \{S\}, \Sigma, S, P_1 \cup \{S \rightarrow S_1 S, S \rightarrow \lambda\}).$$

Esquemáticamente, G es la gramática:

$$\left. \begin{array}{l} S \rightarrow S_1 S \mid \lambda \\ S_1 \rightarrow \cdots \\ \vdots \quad \quad \vdots \end{array} \right\} \text{ producciones de } G_1$$

Ejemplo Utilizar las construcciones del Teorema 5.11.1 para encontrar una GIC que genere el lenguaje $L_1 L_2^*$ donde $L_1 = ba^+$ y $L_2 = \{a^m b^n a^m : m \geq 0, n \geq 1\}$.

Solución. El lenguaje L_1 se puede generar con la gramática

$$\left\{ \begin{array}{l} S_1 \rightarrow bA \\ A \rightarrow aA \mid a \end{array} \right.$$

y L_2 con

$$\left\{ \begin{array}{l} S_2 \rightarrow aS_2 a \mid bB \\ B \rightarrow bB \mid \lambda \end{array} \right.$$

La siguiente gramática genera L_2^* :

$$\left\{ \begin{array}{l} S_3 \rightarrow S_2 S_3 \mid \lambda \\ S_2 \rightarrow aS_2 a \mid bB \\ B \rightarrow bB \mid \lambda \end{array} \right.$$

Finalmente, el lenguaje $L_1 L_2^*$ se puede generar con

$$\left\{ \begin{array}{l} S \rightarrow S_1 S_3 \\ S_1 \rightarrow bA \\ A \rightarrow aA \mid a \\ S_3 \rightarrow S_2 S_3 \mid \lambda \\ S_2 \rightarrow aS_2 a \mid bB \\ B \rightarrow bB \mid \lambda. \end{array} \right.$$

5.11.2 Teorema. Sean L , L_1 y L_2 lenguajes independientes del contexto con el mismo alfabeto de terminales Σ . Entonces

- (1) La intersección $L_1 \cap L_2$ no necesariamente es un LIC.
- (2) El complemento \bar{L} no necesariamente es un LIC.
- (3) La diferencia $L_1 - L_2$ no necesariamente es un LIC.

Demostración.

- (1) La intersección de dos LIC puede ser un lenguaje que no es LIC. Considérense, como ejemplo, los lenguajes

$$\begin{aligned} L_1 &= \{a^i b^i c^j : i, j \geq 0\}, \\ L_2 &= \{a^i b^j c^j : i, j \geq 0\}. \end{aligned}$$

Tanto L_1 como L_2 son LIC porque son generados por las gramáticas G_1 y G_2 , respectivamente:

$$G_1 : \begin{cases} S \rightarrow AB \\ A \rightarrow aAb \mid \lambda \\ B \rightarrow cB \mid \lambda \end{cases} \quad G_2 : \begin{cases} S \rightarrow AB \\ A \rightarrow aA \mid \lambda \\ B \rightarrow bBc \mid \lambda \end{cases}$$

Pero $L_1 \cap L_2 = \{a^i b^i c^i : i \geq 0\}$ no es un LIC, según se demostró en la sección 5.10.

- (2) En el Ejercicio ⑥ de la sección 5.10 se pide demostrar, utilizando el Lema de Bombeo, que el lenguaje $L = \{ww : w \in \{a, b\}^*\}$ no es LIC. Aunque parezca extraño a primera vista, el complemento de L , es decir,

$$\bar{L} = \{u \in \Sigma^* : u \text{ no es una cadena de la forma } ww, \text{ con } w \in \{a, b\}^*\}.$$

sí es un lenguaje LIC. Se puede demostrar que la siguiente gramática G genera el lenguaje \bar{L} :

$$G : \begin{cases} S \rightarrow AB \mid BA \mid A \mid B \\ A \rightarrow CAC \mid a \\ B \rightarrow CBC \mid b \\ C \rightarrow a \mid b. \end{cases}$$

- (3) Como ejemplo podemos considerar $L_1 = \Sigma^*$ y $L_2 = \bar{L}$, donde \bar{L} es el lenguaje presentado en el numeral (2). Tanto L_1 como L_2 son LIC pero $L_1 - L_2 = \Sigma^* - \bar{L} = L$, que no es un LIC. □

Ejercicios de la sección 5.11

- ① Utilizar las construcciones del Teorema 5.11.1 para encontrar GIC que generen los siguientes lenguajes:
- (i) $a^+(a \cup bab)^*(b^* \cup a^*b)$.
 - (ii) $(L_1 \cup L_2)L_3^*$, donde $L_1 = ab^*a$, $L_2 = a \cup b^+$ y $L_3 = \{a^ib^ja^i : i \geq 0\}$.
 - (iii) $L_1 \cup L_2^*L_3$, donde $L_1 = ab^*a$, $L_2 = \{a^ib^jc^jd^i : i, j \geq 1\}$ y $L_3 = b^+$.
- ② Sea $G = (V, \Sigma, S, P)$ una gramática que genera al lenguaje L . ¿La gramática $G = (V, \Sigma, S, P \cup \{S \rightarrow SS, S \rightarrow \lambda\})$ genera a L^* ?
- ③ Demostrar que los LIC son cerrados para la operación de reflexión. Concretamente, demostrar que si L es un LIC, también lo es el lenguaje $L^R = \{w^R : w \in L\}$.

5.12. Autómatas con pila y Lenguajes Independientes del Contexto

Los lenguajes aceptados por los AFPN son exactamente los lenguajes generados por las gramáticas GIC, o sea, los lenguajes independientes del contexto. En primer lugar, a partir de una GIC G es fácil construir un AFPN M tal que $L(G) = L(M)$. El autómata M solamente utiliza dos estados, como se establece en el Teorema 5.12.1.

5.12.1 Teorema. Dada una GIC G , existe un AFPN M tal que $L(G) = L(M)$.

Bosquejo de la demostración. Para una gramática $G = (\Sigma, V, S, P)$ dada, se construye un AFPN que utiliza la pila para simular la derivación de cadenas realizada por G . M requiere solamente dos estados, independientemente del número de variables y producciones de G . Concretamente, el autómata M se define como $M = (Q, q_0, F, \Sigma, \Gamma, \Delta)$, donde $Q = \{q_0, q_1\}$, $F = \{q_1\}$ y $\Gamma = \Sigma \cup V$. La función de transición Δ se define de la siguiente manera:

1. $\Delta(q_0, \lambda, \lambda) = \{(q_1, S)\}$. Transición λ mediante la cual M coloca el símbolo inicial de la gramática, S , en el fondo de la pila al iniciar el procesamiento de una cadena de entrada.
2. Para cada variable $A \in V$,

$$\Delta(q_1, \lambda, A) = \{(q_1, u) : A \rightarrow u \text{ es una producción de la gramática } G\}.$$

De esta manera, todas y cada una de las producciones de G se convierten en transiciones λ de M . Mediante estas transiciones, M utiliza la pila para simular las derivaciones: si el tope de la pila es A y en la derivación se usa la producción $A \rightarrow u$, el tope de la pila A es substituido por u . Recuérdese que este tipo de transiciones está permitido en un AFPN según la Proposición 4.2.1.

3. Para cada símbolo terminal $a \in \Sigma$, $\Delta(q_1, a, a) = \{(q_1, \lambda)\}$. Mediante estas transiciones, M borra los terminales del tope de la pila al consumirlos sobre la cinta de entrada.

El autómata M está diseñado de tal forma que si $S \xRightarrow{*} w$ es una derivación a izquierda en la gramática G , entonces existe un procesamiento

$$[q_0, w, \lambda] \vdash [q_1, w, S] \vdash^* [q_1, \lambda, \lambda]$$

que simula la derivación.

Recíprocamente, puede demostrarse que si $[q_0, w, \lambda] \vdash^* [q_1, \lambda, \lambda]$ entonces $S \xRightarrow{*} w$ en la gramática G . \square

Ejemplo Sea G la gramática:

$$G : \begin{cases} S \rightarrow aAbS \mid bBa \mid \lambda \\ A \rightarrow aA \mid a \\ B \rightarrow bB \mid b \end{cases}$$

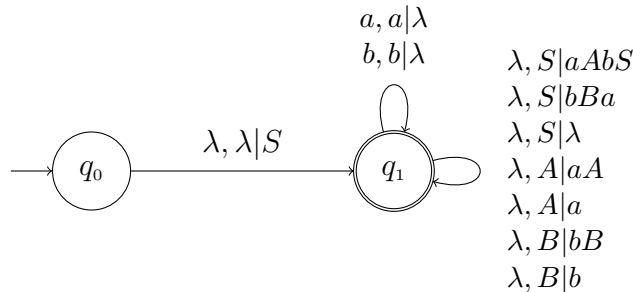
Según la construcción del Teorema 5.12.1, el autómata M está dado por $M = (Q, q_0, F, \Sigma, \Gamma, \Delta)$ donde

$$\begin{aligned} Q &= \{q_0, q_1\}, \\ F &= \{q_1\}, \\ \Gamma &= \{a, b, S, A, B\}, \end{aligned}$$

y la función de transición Δ es:

$$\begin{aligned} \Delta[q_0, \lambda, \lambda] &= \{(q_1, S)\}, \\ \Delta[q_1, \lambda, S] &= \{(q_1, aAbS), (q_1, bBa), (q_1, \lambda)\}, \\ \Delta[q_1, \lambda, A] &= \{(q_1, aA), (q_1, a)\}, \\ \Delta[q_1, \lambda, B] &= \{(q_1, bB), (q_1, b)\}, \\ \Delta[q_1, a, a] &= \{(q_1, \lambda)\}, \\ \Delta[q_1, b, b] &= \{(q_1, \lambda)\}. \end{aligned}$$

El autómata puede presentarse también por medio de un grafo:



Podemos ilustrar la correspondencia entre derivaciones en G y procesamientos en M con la cadena $aabbba$, la cual tiene la siguiente derivación a izquierda:

$$S \Rightarrow aAbS \Rightarrow aabS \Rightarrow aabbBa \Rightarrow aabbba.$$

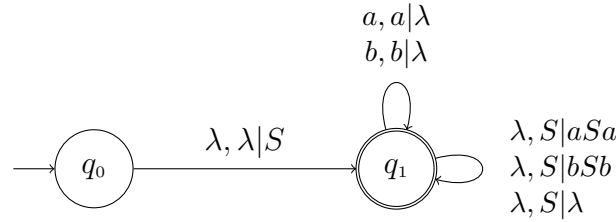
El autómata M simula esta derivación de la cadena $aabbba$ así:

$$\begin{aligned} [q_0, aabbba, \lambda] &\vdash [q_1, aabbba, S] \vdash [q_1, aabbba, aAbS] \\ &\vdash [q_1, abbba, AbS] \vdash [q_1, abbba, abS] \\ &\vdash [q_1, bbba, bS] \vdash [q_1, bba, S] \vdash [q_1, bba, bBa] \\ &\vdash [q_1, ba, Ba] \vdash [q_1, ba, ba] \vdash [q_1, a, a] \vdash [q_1, \lambda, \lambda]. \end{aligned}$$

Ejemplo La siguiente gramática genera los palíndromos de longitud par, sobre el alfabeto $\Sigma = \{a, b\}$, es decir, el lenguaje $L = \{ww^R : w \in \Sigma^*\}$:

$$S \rightarrow aSa \mid bSb \mid \lambda.$$

Siguiendo el procedimiento del Teorema 5.12.1 podemos construir un AFPN que acepta a L ; su grafo es:



Puede observarse que este autómata es diferente al exhibido en la sección 4.3 para aceptar el mismo lenguaje.

Ejercicios de la sección 5.12

① Sea G la gramática

$$G : \begin{cases} S \longrightarrow aA \mid aAA \\ A \longrightarrow aAB \mid aA \mid a \\ B \longrightarrow aBAB \mid b. \end{cases}$$

- (i) Siguiendo el procedimiento del Teorema 5.12.1, construir un AFPN M que acepte el lenguaje generado por G .
- (ii) Encontrar una derivación a izquierda en G de la cadena $u = aaaabab$ y hacer el árbol de esa derivación.
- (iii) Utilizando la notación de configuración (o descripción) instantánea, procesar la cadena u con el autómata M , paso a paso, simulando la derivación encontrada en (ii).