Capítulo 4

Lenguajes y gramáticas independientes del contexto

Como se ha visto, los autómatas son dispositivos que procesan cadenas de entrada. En capítulos posteriores consideraremos modelos de autómatas con mayor poder computacional que el de los modelos AFD-λ, AFN y AFD. En el presente capítulo iniciaremos el estudio de las **gramáticas generativas**, que son mecanismos para generar cadenas a partir de un símbolo inicial (el estudio de gramáticas continuará en el Capítulo 8).

☞ Los autómatas *procesan* cadenas
☞ Las gramáticas *generan* cadenas

Las gramáticas generativas y, en particular, las gramáticas independientes del contexto (GIC), fueron introducidas por Noam Chomsky en 1956 como un modelo para la descripción de los lenguajes naturales (como el español, el inglés, etc). En la década de los sesenta se comenzaron a usar GIC para presentar la sintaxis de lenguajes de programación y para el diseño de analizadores sintácticos en compiladores.

4.1. Gramáticas independientes del contexto

Una **gramática independiente del contexto** (GIC), también llamada **gramática contextual**, 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 **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 $(A, w) \in P$ de G se denota por $A \to w$ y se lee "A produce w"; su significado es: la variable A se puede reemplazar (sobre-escribir) por la cadena w. En la producción $A \to w$, A se denomina la **cabeza** y w el **cuerpo** de la producción.

Notación y definiciones. Las variables se denotan con las letras mayúsculas A, B, C, ... Los elementos de Σ o símbolos terminales se denotan con letras minúsculas $a, b, c, \ldots$. Si $u, v \in (V \cup \Sigma)^*$ y $A \to w$ es una producción, se dice que uwv se **deriva directamente** de uAv, lo cual se denota por

$$uAv \Rightarrow uwv.$$

Si se quiere hacer referencia a la gramática G, se escribe

$$uAv \overset{G}{\Longrightarrow} uwv \quad \text{ó} \quad uAv \Longrightarrow_G uwv.$$

Si $u_1, u_2, \ldots, u_n$ son cadenas en $(V \cup \Sigma)^*$ y hay una sucesión de derivaciones directas

$$u_1 \overset{G}{\Longrightarrow} u_2, \quad u_2 \overset{G}{\Longrightarrow} u_3, \ \ldots \ , u_{n-1} \overset{G}{\Longrightarrow} u_n$$

se dice que u_n se deriva de u_1 y se escribe $u_1 \overset{*}{\Longrightarrow} u_n$. La anterior sucesión de derivaciones directas se representa como

$$u_1 \Rightarrow u_2 \Rightarrow u_3 \Rightarrow \cdots \Rightarrow u_{n-1} \Rightarrow u_n$$

y se dice que es una **derivación** o una **generación** de u_n a partir de u_1. Para toda cadena w se asume que $w \overset{*}{\Longrightarrow} w$; por lo tanto, $u \overset{*}{\Longrightarrow} v$ significa que v se obtiene de u utilizando cero, una o más producciones de la gramática. Análogamente, $u \overset{+}{\Longrightarrow} v$ significa que v se obtiene de u utilizando una o más producciones.

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

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

Un lenguaje L sobre un alfabeto Σ se dice que es un **lenguaje independiente del contexto** (LIC) 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 cada producción o regla de re-escritura $A \to w$ se aplica a la variable A independientemente de los caracteres que la rodean, es decir, independientemente del contexto en el que aparece A.

Ejemplo Sea G una gramática (V, Σ, S, P) dada por:

$$\begin{aligned} V &= \{S\} \\ \Sigma &= \{a\} \\ P &= \{S \to aS,\ S \to \lambda\} \end{aligned}$$

Se tiene $S \Rightarrow \lambda$ y

$$S \Rightarrow aS \overset{*}{\Longrightarrow} a \cdots aS \Rightarrow a \cdots a.$$

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

De manera más simple, podemos presentar una gramática GIC listando sus producciones y separando con el símbolo | 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 del ejemplo anterior se presenta simplemente como:

$$S \to aS \mid \lambda.$$

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

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

se puede presentar como

$$\begin{cases} S \to aS \mid bA \mid \lambda \\ A \to bA \mid b \mid \lambda \end{cases}$$

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

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

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

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

$$\begin{cases} S \to AB \\ A \to aA \mid \lambda \\ B \to bB \mid \lambda \end{cases} \qquad \begin{cases} S \to AB \mid \lambda \\ A \to aA \mid a \mid \lambda \\ B \to bB \mid b \mid \lambda \end{cases} \qquad \begin{cases} S \to aS \mid A \\ A \to bA \mid \lambda \end{cases}$$

Ejemplo La gramática

$$\begin{cases} S \to aS \mid aA \\ A \to bA \mid b \end{cases}$$

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

$$\begin{cases} S \to AB \\ A \to aA \mid a \\ B \to bB \mid b \end{cases}$$

Ejemplo La gramática

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

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 $0^*10^*10^*$ sobre $\Sigma = \{0, 1\}$, es decir, el lenguaje de todas las cadenas con exactamente dos unos.

Solución:

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

Una gramática equivalente es

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

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

Solución:

$$S \to aSb \mid \lambda.$$

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

Solución:

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

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

Solución:

$$S \to aSa \mid bSb \mid aSb \mid bSa \mid \lambda$$

Dos gramáticas equivalentes son:

$$\begin{cases} S \to aaS \mid bbS \mid abS \mid baS \mid \lambda \end{cases} \qquad \begin{cases} S \to AAS \mid \lambda \\ A \to a \mid b \end{cases}$$

Ejemplo Encontrar una GIC que genere el lenguaje $(ab \cup ba)^*$ sobre $\Sigma = \{a, b\}$.

Solución:

$$S \to abS \mid baS \mid \lambda.$$

Ejemplo Demostrar que la gramática G dada por:

$$S \to (S)S \mid \lambda$$

genera el lenguaje de todas las cadenas de paréntesis anidados y equilibrados; es decir, cadenas como `(())`, `()()()`, `(())((()))`.

Solución: Es fácil ver que G genera cadenas de paréntesis anidados y equilibrados ya que cada aplicación de la producción $S \to (S)S$ da lugar a un par de paréntesis equilibrados.

Para establecer la dirección recíproca demostraremos por inducción sobre n la siguiente afirmación: "Toda cadena con $2n$ paréntesis anidados y equilibrados se puede generar en G".

$n = 1$: `()` se genera con $S \Rightarrow (S)S \overset{2}{\Longrightarrow} ()$.

$n = 2$: `()()` se genera con $S \Rightarrow (S)S \Rightarrow (S)(S)S \overset{3}{\Longrightarrow} ()()$.

`(())` se genera con $S \Rightarrow (S)S \Rightarrow ((S)S)S \overset{3}{\Longrightarrow} (())$.

Paso inductivo: una cadena con $2n$ paréntesis anidados y equilibrados es de la forma (u) ó $(u)v$ donde u y v tienen estrictamente menos de $2n$ paréntesis anidados y equilibrados. Por hipótesis de inducción $S \overset{+}{\Longrightarrow} u$ y $S \overset{+}{\Longrightarrow} v$. Por lo tanto,

$$S \Rightarrow (S)S \overset{+}{\Longrightarrow} (u)S \Rightarrow (u).$$
$$S \Rightarrow (S)S \overset{+}{\Longrightarrow} (u)S \overset{+}{\Longrightarrow} (u)v.$$

Ejercicios de la sección 4.1

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

   (i) El lenguaje de las cadenas que tienen un número par de $b$es.

   (ii) El lenguaje de las cadenas que comienzan con b y terminan con ba.

   (iii) $a^*b \cup a$.

(iv) $a^*b \cup b^*a$.

(v) $(ab^* \cup b^*a)^*$.

(vi) $\{a^i b^{2i} : i \geq 0\}$.

(vii) $\{ab^i ab^i : i \geq 1\}$.

2. Encontrar GIC que generen los siguientes lenguajes sobre el alfabeto $\{a, b, c, d\}$:

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

   (ii) $\{a^i b^i c^j d^j : i, j \geq 1\}$.

   (iii) $\{a^i b^j c^j d^i : i, j \geq 1\} \cup \{a^i b^i c^j d^j : i, j \geq 1\}$.

3. Encontrar una GIC que genere el lenguaje $\{a^i b^j c^{i+j} : i \geq 0, j \geq 1\}$, sobre $\{a, b, c\}$.

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

5. Demostrar que la gramática $S \rightarrow SS \mid (S) \mid \lambda$ también genera el lenguaje de todas las cadenas de paréntesis anidados y equilibrados.

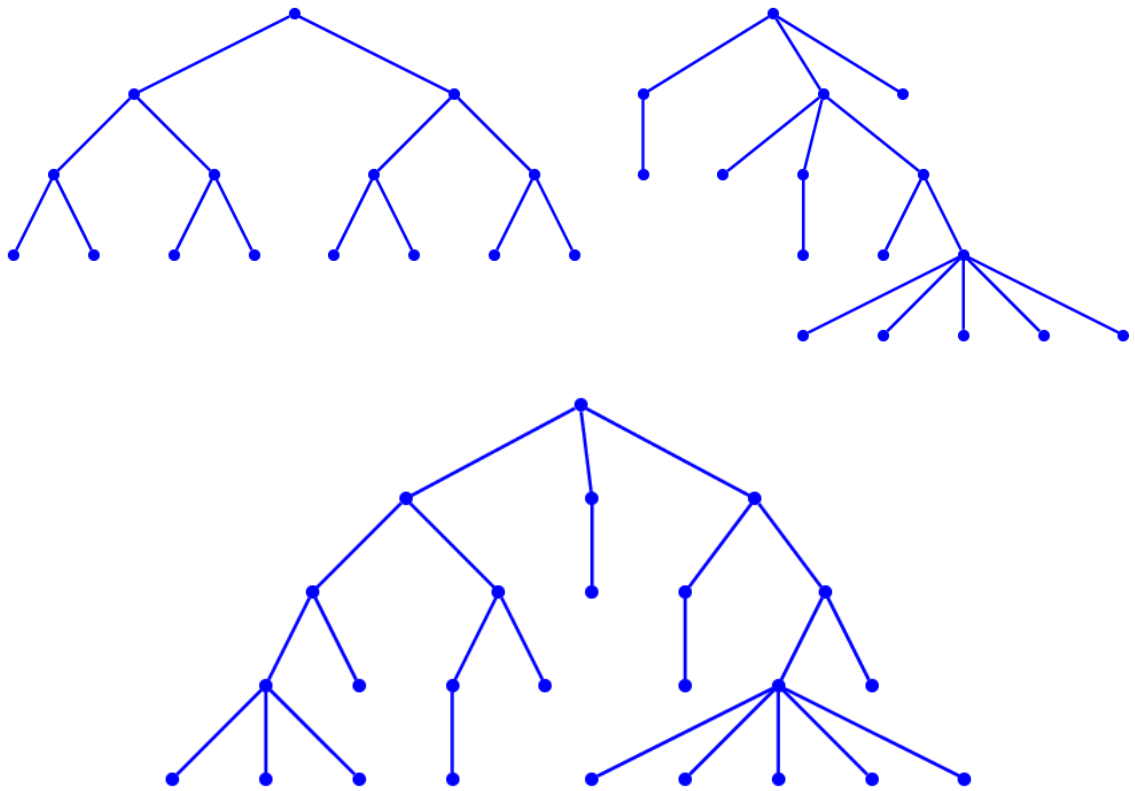
6. Encontrar una gramática que genere el lenguaje de todas las cadenas de paréntesis circulares y/o angulares, anidados y equilibrados. Es decir cadenas como `([()])`, `([])[()]`, `[()([[()()]])]`, etc. Cadenas como `([)]` ó `[([)]]` tienen paréntesis equilibrados pero no anidados y, por lo tanto, no pertenecen a este lenguaje.

4.2. Árbol de una derivación

Un **árbol con raíz** es un tipo muy particular de grafo no-dirigido; está formado por un conjunto de vértices o nodos conectados entre sí por arcos o aristas, con la siguiente propiedad: existe un nodo especial, llamado la **raíz** del árbol, tal que hay una única trayectoria entre cualquier nodo y la raíz. De esta manera, la raíz se ramifica en nodos, llamados **descendientes inmediatos**, cada uno de los cuales puede tener, a su vez, descendientes inmediatos, y así sucesivamente.

La propiedad que caracteriza a un árbol garantiza que un nodo puede tener 0, 1, o más descendientes inmediatos pero 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**. Los nodos que no tienen descendientes se llaman **hojas** del árbol. En la terminología usualmente utilizada, los descendientes de un nodo también se denominan **hijos** y los antecesores **padres** o **ancestros**. El número de vértices (y por lo tanto, el número de arcos) de un árbol puede ser infinito.

Ejemplo Algunos árboles con raíz:



El **árbol de una derivación** $S \overset{*}{\Longrightarrow} w$, $w \in \Sigma^*$, en una GIC es un árbol con raíz que se forma de la siguiente manera:

1. La raíz está etiquetada con el símbolo inicial S.
2. Cada nodo interior está etiquetado con un no terminal.
3. Cada hoja está etiquetada con terminal o con λ.
4. Si en la derivación se utiliza la producción $A \to s_1 s_2 \cdots s_k$, donde $s_i \in (V \cup \Sigma)^*$, el nodo A tiene k descendientes inmediatos: s_1, s_2,..., s_k, escritos de izquierda a derecha.

De esta forma, las hojas del árbol de derivación de $S \overset{*}{\Longrightarrow} w$ son precisamente los símbolos de la cadena w, leídos de izquierda a derecha y, posiblemente, algunos λ.

La búsqueda de un árbol de derivación para una cadena $w \in \Sigma^*$ se denomina **análisis sintáctico** de w. Los árboles de derivación también se suelen llamar árboles sintácticos o árboles de análisis sintáctico.

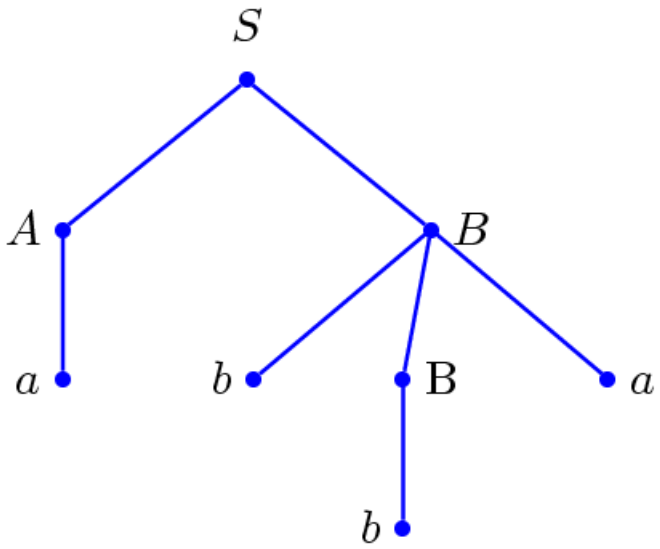
Ejemplo Sea G la gramática:

$$\begin{cases} S \to AB \mid AaB \\ A \to aA \mid a \\ B \to bBa \mid b \end{cases}$$

El árbol de la derivación

$$S \Rightarrow AB \Rightarrow AbBa \Rightarrow abBa \Rightarrow abba$$

es



El anterior es también el árbol de la derivación

$$S \Rightarrow AB \Rightarrow aB \Rightarrow abBa \Rightarrow abba.$$

Esto muestra que dos derivaciones diferentes pueden tener el mismo árbol de derivación ya que en el árbol aparecen las producciones utilizadas pero no el orden en que han sido aplicadas.

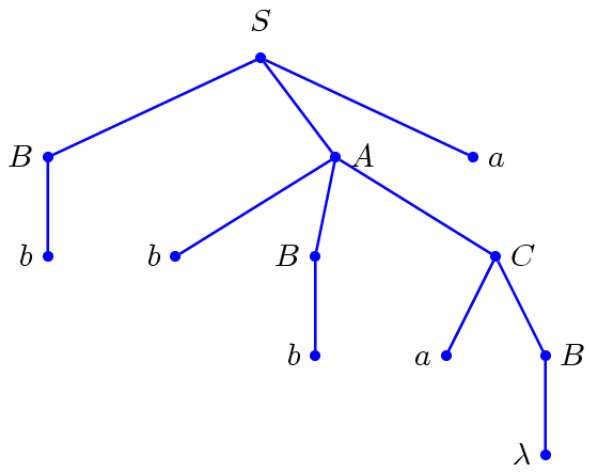
Ejemplo Sea G la gramática:

$$\begin{cases} S \to BAa \\ A \to bBC \mid a \\ B \to bB \mid b \mid \lambda \\ C \to aB \mid aa \end{cases}$$

El árbol de la derivación

$$S \Rightarrow BAa \Rightarrow bAa \Rightarrow bbBCa \Rightarrow bbbCa \Rightarrow bbbaBa \Rightarrow bbbaa$$

es



Ejercicios de la sección 4.2

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:

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

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

4.3. Gramáticas ambiguas

La noción de ambigüedad se puede presentar en términos de árboles sintácticos o en términos de ciertas derivaciones "estándares": las llamadas derivaciones a izquierda.

4.3.1 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.

Una derivación cualquiera se puede transformar siempre en una única derivación a izquierda aplicando, en cada paso, producciones a la variable que esté más a la izquierda.

4.3.2 Definición. Una GIC G es **ambigua** si existe una cadena $w \in \Sigma^*$ para la cual hay dos derivaciones a izquierda diferentes. Equivalentemente, una GIC G es **ambigua** si existe una cadena $w \in \Sigma^*$ con dos árboles de derivación diferentes.

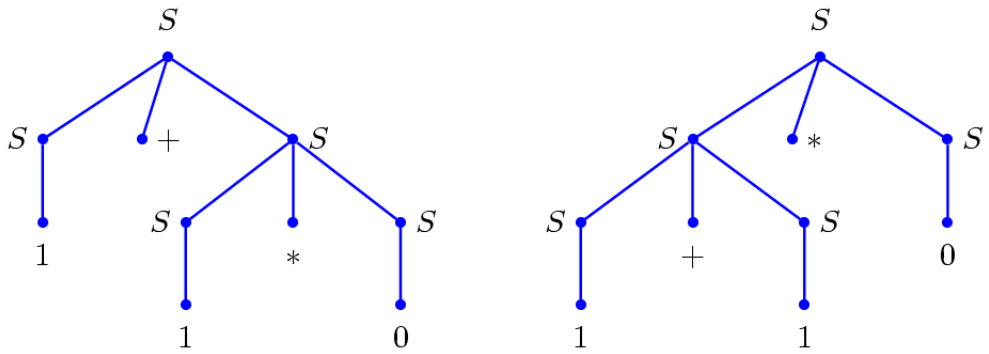
Ejemplo Considérese el alfabeto $\Sigma = \{0, 1, +, *, (,)\}$. La siguiente gramática G_{sp} para generar números naturales, sumas y productos, en numeración binaria, es ambigua:

$$S \to S + S \mid S * S \mid (S) \mid 0S \mid 1S \mid 0 \mid 1$$

La cadena $1 + 1 * 0$ tiene dos derivaciones a izquierda diferentes:

$$S \Rightarrow S + S \Rightarrow 1 + S \Rightarrow 1 + S * S \Rightarrow 1 + 1 * S \Rightarrow 1 + 1 * 0.$$
$$S \Rightarrow S * S \Rightarrow S + S * S \Rightarrow 1 + S * S \Rightarrow 1 + 1 * S \Rightarrow 1 + 1 * 0.$$

Los árboles de derivación correspondientes a las anteriores derivaciones son:



En la gramática G_{sp} la ambigüedad se puede eliminar introduciendo paréntesis:

$$S \to (S + S) \mid (S * S) \mid (S) \mid 0S \mid 1S \mid 0 \mid 1$$

Aunque la introducción de paréntesis elimina la ambigüedad, las expresiones generadas tienen un excesivo número de paréntesis lo que dificulta el análisis sintáctico (en un compilador, por ejemplo). Lo más corriente en estos casos es utilizar gramáticas ambiguas como G_{sp} siempre y cuando se establezca un orden de precedencia para los operadores. Lo usual es establecer que $*$ tenga una mayor orden de precedencia que $+$, es decir, por convención $*$ actúa antes que $+$. Por ejemplo, la expresión $1 * 1 + 0$ se interpreta como $(1 * 1) + 0$ y la expresión $10 + 11 * 110 + 1$ se interpreta como $10 + (11 * 110) + 1$.

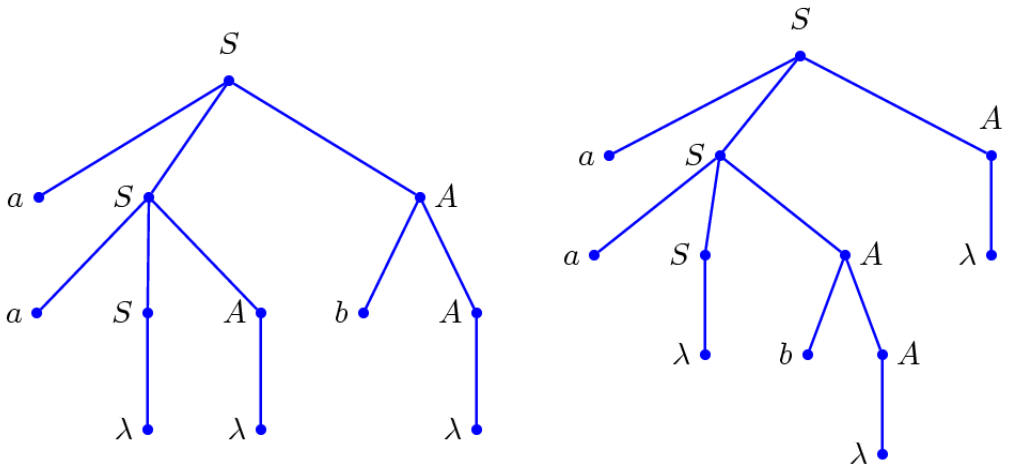
Ejemplo La siguiente gramática es ambigua:

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

G es ambigua porque para la cadena aab hay dos derivaciones a izquierda diferentes.

$$S \Rightarrow aSA \Rightarrow aaSAA \Rightarrow aaAA \Rightarrow aaA \Rightarrow aabA \Rightarrow aab.$$
$$S \Rightarrow aSA \Rightarrow aaSAA \Rightarrow aaAA \Rightarrow aabAA \Rightarrow aabA \Rightarrow aab.$$

Los árboles de derivación para estas dos derivaciones son:



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

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

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 $S \to AB$; la variable A genera cadenas de $a$es de manera única y B genera cadenas de $b$es también de manera única. Por consiguiente, toda cadena tiene una única derivación a izquierda.

✎ Existen lenguajes independientes del contexto para los cuales toda gramática es ambigua. Tales lenguajes se llaman **inherentemente ambiguos**. Un ejemplo es el lenguaje

$$L = \{a^i b^i c^j d^j : i, j \geq 1\} \cup \{a^i b^j c^j d^i : i, j \geq 1\}.$$

sobre el alfabeto $\{a, b, c, d\}$. En el Ejercicio 4.1.2. (iii) se pidió mostrar que L es un LIC. La demostración de que L es inherentemente ambiguo es bastante intrincada y puede encontrarse en la referencia [HU].

✎ No existe ningún algoritmo que permita determinar si una GIC dada es o no ambigua. Con la terminología de la sección 3.6, esto quiere decir que el problema de la ambigüedad para GIC es indecidible. Éste no es un resultado trivial; para su demostración remitimos al lector a la referencia [HMU].

Ejercicios de la sección 4.3

1. Mostrar que las siguientes gramáticas son ambiguas:

   (i) $S \to aSbS \mid bSaS \mid \lambda$.

   (ii) $S \to abS \mid abScS \mid \lambda$.

2. Mostrar que las gramáticas G_1 y G_2 siguientes son ambiguas. En cada caso, encontrar el lenguaje generado por la gramática y construir luego una gramática no-ambigua que genere el mismo lenguaje.

$$G_1 : \begin{cases} S \to AaSbB \mid \lambda \\ A \to aA \mid a \\ B \to bB \mid \lambda \end{cases}$$

$$G_2 : \begin{cases} S \to ASB \mid AB \\ A \to aA \mid a \\ B \to bB \mid \lambda \end{cases}$$

3. Encontrar una GIC no-ambigua que genere el lenguaje $a^*b(a \cup b)^*$.

4.4. Gramáticas para lenguajes de programación

La sintaxis de los lenguajes de programación, o al menos una gran porción de ésta, se presenta usualmente por medio de gramáticas GIC. En tales casos se dice que el lenguaje está en la **forma Backus-Naur** o, simplemente, en la **forma BNF**. Los lenguajes que están en **BNF** ofrecen ventajas significativas para el diseño de analizadores sintácticos en compiladores.

Ejemplo A continuación se exhibe una gramática para generar los números reales sin signo, similar a la utilizada en muchos lenguajes de programación. Las variables aparecen encerradas entre paréntesis $\langle\ \rangle$ y $\langle real\rangle$ es la variable inicial de la gramática. El alfabeto de terminales es $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \texttt{.}, \texttt{+}, \texttt{-}, \texttt{E}\}$. En el contexto de los lenguajes de programación los terminales se denominan también **componentes léxicos**, **lexemas** o **tokens**.

$$
\begin{array}{lcl}
\langle real\rangle & \rightarrow & \langle dígitos\rangle\langle decimal\rangle\langle exp\rangle \\
\langle dígitos\rangle & \rightarrow & \langle dígitos\rangle\langle dígitos\rangle \mid 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
\langle decimal\rangle & \rightarrow & \texttt{.}\langle dígitos\rangle \mid \lambda \\
\langle exp\rangle & \rightarrow & \texttt{E}\langle dígitos\rangle \mid \texttt{E+}\langle dígitos\rangle \mid \texttt{E-}\langle dígitos\rangle \mid \lambda
\end{array}
$$

Esta gramática genera expresiones como 47.236, 321.25E+35, 0.8E9 y 0.8E+9. Las partes decimal y exponencial son "opcionales" debido a las producciones $\langle decimal\rangle \rightarrow \lambda$ y $\langle exp\rangle \rightarrow \lambda$, pero no se generan expresiones como .325, E125, 42.5E ni 0.1E+.

Ejemplo Gramática para generar los **identificadores** en lenguajes de programación, es decir, cadenas cuyo primer símbolo es una letra que va seguida de letras y/o dígitos. Las variables aparecen encerradas entre paréntesis $\langle\ \rangle$ e $\langle identificador\rangle$ es la variable inicial de la gramática. La variable $\langle lsds\rangle$ representa "letras o dígitos". Los terminales en esta gramática son las letras, minúsculas o mayúsculas, y los dígitos.

$$
\begin{array}{lcl}
\langle identificador\rangle & \rightarrow & \langle letra\rangle\langle lsds\rangle \\
\langle lsds\rangle & \rightarrow & \langle letra\rangle\langle lsds\rangle \mid \langle dígito\rangle\langle lsds\rangle \mid \lambda \\
\langle letra\rangle & \rightarrow & \text{a} \mid \text{b} \mid \text{c} \mid \cdots \mid \text{x} \mid \text{y} \mid \text{z} \mid \text{A} \mid \text{B} \mid \text{C} \mid \cdots \mid \text{Y} \mid \text{Z} \\
\langle dígito\rangle & \rightarrow & 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{array}
$$

Ejercicios de la sección 4.4

1. Con la gramática del primer ejemplo de esta sección hacer derivaciones y árboles de derivación para las cadenas 235.101E+25 y 0.01E-12.

2. Encontrar una GIC, con una sola variable, para generar los identificadores, es decir, el lenguaje del segundo ejemplo de esta sección.

3. Una gramática similar a la siguiente se usa en muchos lenguajes de programación para generar expresiones aritméticas formadas por sumas y productos de números en binario:

$$\begin{array}{lcl} \langle \textit{expresión} \rangle & \rightarrow & \langle \textit{término} \rangle \mid \langle \textit{expresión} \rangle \texttt{+} \langle \textit{término} \rangle \\ \langle \textit{término} \rangle & \rightarrow & \langle \textit{factor} \rangle \mid \langle \textit{término} \rangle \texttt{*} \langle \textit{factor} \rangle \\ \langle \textit{factor} \rangle & \rightarrow & \langle \textit{número} \rangle \mid \texttt{(} \langle \textit{expresión} \rangle \texttt{)} \\ \langle \textit{número} \rangle & \rightarrow & \texttt{1} \langle \textit{número} \rangle \mid \texttt{0} \langle \textit{número} \rangle \mid \texttt{0} \mid \texttt{1} \end{array}$$

El alfabeto de terminales de esta gramática es $\{\texttt{0}, \texttt{1}, \texttt{+}, \texttt{*}, \texttt{(}, \texttt{)}\}$.

(i) Hacer derivaciones y árboles de derivación para las siguientes cadenas:

```
10+101*10
(101+10*10)
(10+111)*(100+10*101+1111)
```

(ii) Demostrar que esta gramática no es ambigua.

4.5. 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 $\langle \textit{Oración} \rangle$ es la variable inicial de la gramática. Los terminales son las palabras propias del idioma.

$$\begin{array}{lcl} \langle \textit{Oración} \rangle & \rightarrow & \langle \textit{Sujeto} \rangle \langle \textit{Verbo} \rangle \langle \textit{Compl. Directo} \rangle \mid \\ & & \langle \textit{Sujeto} \rangle \langle \textit{Verbo} \rangle \langle \textit{Compl. Directo} \rangle \langle \textit{Compl. Circunst.} \rangle \mid \\ & & \langle \textit{Sujeto} \rangle \langle \textit{Verbo} \rangle \langle \textit{Compl. Indirecto} \rangle \langle \textit{Compl. Circunst.} \rangle \\ \langle \textit{Sujeto} \rangle & \rightarrow & \langle \textit{Sustant.} \rangle \mid \text{Juan} \mid \text{Pedro} \mid \text{María} \mid \cdots \\ \langle \textit{Compl. Directo} \rangle & \rightarrow & \langle \textit{Prepos.} \rangle \langle \textit{Artículo} \rangle \langle \textit{Sustant.} \rangle \mid \\ & & \langle \textit{Prepos.} \rangle \langle \textit{Artículo} \rangle \langle \textit{Sustant.} \rangle \langle \textit{Prepos.} \rangle \langle \textit{Artículo} \rangle \langle \textit{Sustant.} \rangle \mid \\ & & \langle \textit{Prepos.} \rangle \langle \textit{Artículo} \rangle \langle \textit{Sustant.} \rangle \langle \textit{Prepos.} \rangle \langle \textit{Sustant.} \rangle \langle \textit{Adjetivo} \rangle \\ \langle \textit{Compl. Indirecto} \rangle & \rightarrow & \langle \textit{Prepos.} \rangle \langle \textit{Artículo} \rangle \langle \textit{Sustant.} \rangle \mid \\ & & \langle \textit{Prepos.} \rangle \langle \textit{Artículo} \rangle \langle \textit{Sustant.} \rangle \langle \textit{Adjetivo} \rangle \mid \\ & & \langle \textit{Prepos.} \rangle \langle \textit{Sustant.} \rangle \langle \textit{Prepos.} \rangle \langle \textit{Sustant.} \rangle \end{array}$$

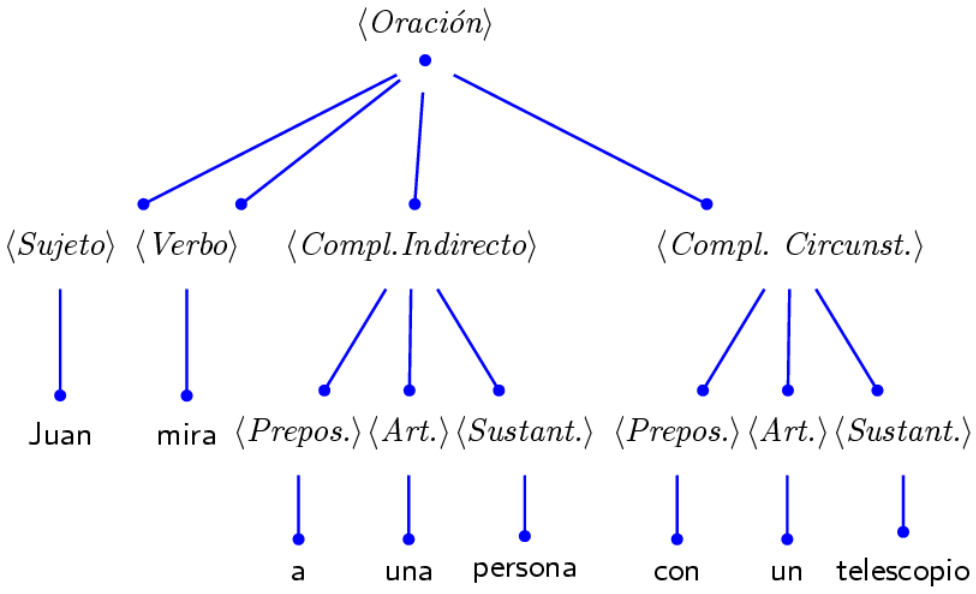
$$
\begin{aligned}
\langle \textit{Compl. Circunst.}\rangle &\to \langle \textit{Prepos.}\rangle\langle \textit{Artículo}\rangle\langle \textit{Sustant.}\rangle \mid \\
&\quad \langle \textit{Prepos.}\rangle\langle \textit{Artículo}\rangle\langle \textit{Sustant.}\rangle\langle \textit{Adjetivo}\rangle \mid \langle \textit{Adverbio}\rangle \mid \\
&\quad \langle \textit{Prepos.}\rangle\langle \textit{Artículo}\rangle\langle \textit{Sustant.}\rangle\langle \textit{Prepos.}\rangle\langle \textit{Artículo}\rangle\langle \textit{Sustant.}\rangle \\
\langle \textit{Sustant.}\rangle &\to \text{casa} \mid \text{perro} \mid \text{libro} \mid \text{lápiz} \mid \text{mesa} \mid \lambda \mid \cdots \\
\langle \textit{Adjetivo}\rangle &\to \text{rojo} \mid \text{azul} \mid \text{inteligente} \mid \text{malvado} \mid \text{útil} \mid \lambda \mid \cdots \\
\langle \textit{Prepos.}\rangle &\to \text{a} \mid \text{ante} \mid \text{bajo} \mid \text{con} \mid \text{contra} \mid \text{de} \mid \text{desde} \mid \text{en} \mid \text{entre} \mid \text{hacia} \mid \\
&\quad \text{hasta} \mid \text{para} \mid \text{por} \mid \text{según} \mid \text{sin} \mid \text{so} \mid \text{sobre} \mid \text{tras} \mid \lambda \mid \cdots \\
\langle \textit{Artículo}\rangle &\to \text{el} \mid \text{la} \mid \text{lo} \mid \text{las} \mid \text{los} \mid \text{un} \mid \text{uno} \mid \text{una} \mid \text{unas} \mid \text{unos} \mid \lambda \\
\langle \textit{Adverbio}\rangle &\to \text{muy} \mid \text{bastante} \mid \text{poco} \mid \text{demasiado} \mid \text{lento} \mid \text{lentamente} \mid \\
&\quad \text{rápido} \mid \text{rápidamente} \mid \lambda \mid \cdots \\
\langle \textit{Verbo}\rangle &\to \text{escribir} \mid \text{escribo} \mid \text{escribe} \mid \text{escribes} \mid \text{escriben} \mid \text{escribí} \mid \\
&\quad \text{escribiste} \mid \text{escribieron} \mid \cdots
\end{aligned}
$$

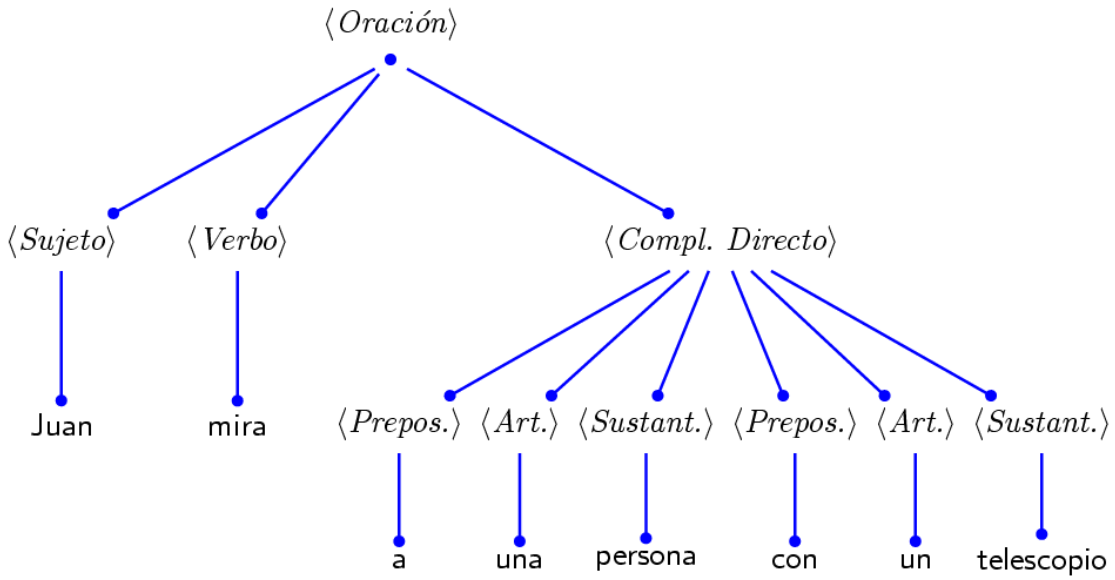
Los lenguajes naturales son casi siempre ambiguos porque existen muchas reglas de producción, lo que da lugar a múltiples árboles de derivación para ciertas oraciones.

Ejemplo La oración

Juan mira a una persona con un telescopio

es ambigua. La ambigüedad surge porque las producciones permiten dos árboles de derivación:





Ejercicios de la sección 4.5

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.

4.6. Gramáticas regulares

4.6.1 Definición. Una GIC se llama **regular** si sus producciones son de la forma

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

Los siguientes teoremas establecen la conexión entre los lenguajes regulares y las gramáticas regulares.

4.6.2 Teorema. *Dado un AFD* $M = (Q, \Sigma, q_0, F, \delta)$, *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

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

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 $q \overset{*}{\Longrightarrow} wp$.

La demostración de (1) se hace por inducción sobre w. Si $w = a$ y $\delta(q,a) = p$, entonces $q \to ap$ 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 $q \overset{*}{\Longrightarrow} wp$ y como $\delta(p,a) = p'$, entonces $p \Rightarrow ap'$. Por lo tanto,

$$q \overset{*}{\Longrightarrow} wp \Rightarrow wap'$$

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 $q \overset{*}{\Longrightarrow} wp$ entonces $\delta(q,w) = p$.

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

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

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.

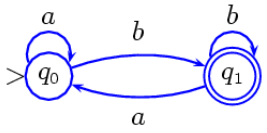
Como consecuencia de (1) y (2) se puede ahora demostrar que

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

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, $q_0 \to \lambda$ es una producción de G. Así que $S \Rightarrow \lambda$. Recíprocamente, si $S \overset{*}{\Longrightarrow} \lambda$, 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 $q_0 \overset{*}{\Longrightarrow} w$, o sea, $S \overset{*}{\Longrightarrow} w$. Recíprocamente, si $S \overset{*}{\Longrightarrow}_G w$, entonces $q_0 \overset{*}{\Longrightarrow}_G wp \Rightarrow w$ donde $p \to \lambda$. Utilizando (2), se tiene $\delta(q_0, w) = p \in F$. □

Ejemplo El siguiente AFD M, presentado en el último ejemplo de la sección 2.3, acepta las cadenas que terminan en b:



M induce la gramática regular

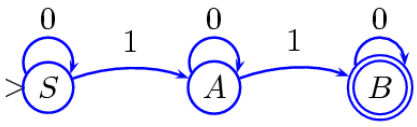
$$G: \quad \begin{cases} q_0 \to aq_0 \mid bq_1 \\ q_1 \to bq_1 \mid aq_0 \mid \lambda \end{cases}$$

que cumple $L(M) = L(G)$. Las variables de G son q_0 y q_1, siendo q_0 la variable inicial.

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 4.1 una gramática que lo genera:

$$\begin{cases} S \to A1A1A \\ A \to 0A \mid \lambda \end{cases}$$

Esta gramática no es regular, pero por medio del AFD



y el Teorema 4.6.2 se puede obtener una GIC regular que genere $0^*10^*10^*$:

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

4.6.3 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

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

Usando razonamientos similares a los del Teorema 4.6.2, se puede demostrar que

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

de donde $L(M) = L(G)$. Los detalles se dejan como ejercicio. □

4.6.4 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 4.6.2, el Teorema 4.6.3 y del Teorema de Kleene.
2. Se sigue de la parte 1. Por otro lado, $\{a^i b^i : i \geq 0\}$ es LIC pero no es regular. □

4.6.5 Definición. Una GIC se llama **regular por la derecha** si sus producciones son de la forma

$$\begin{cases} A \to wB, & w \in \Sigma^*, \ B \in V, \\ A \to \lambda \end{cases}$$

4.6.6 Teorema. *Las gramáticas regulares y las gramáticas regulares por la derecha generan los mismos lenguajes, es decir, los lenguajes regulares. Dicho de otra manera, la definición de gramática regular es equivalente a la definición de gramática regular por la derecha.*

Demostración: Una gramática regular es obviamente regular por la derecha. Recíprocamente, en una gramática regular por la derecha $G = (V, \Sigma, S, P)$, una producción de la forma

$$A \to a_1 a_2 \cdots a_n B$$

donde los $a_i \in \Sigma$, $n \geq 2$, $B \in V$, se puede simular con producciones de la forma $A \to aB$ y $A \to \lambda$. En efecto, se introducen $n-1$ variables *nuevas* $A_1, \ldots, A_{n-1}$ cuyas *únicas* producciones son:

$$\begin{array}{lcl} A & \to & a_1 A_1 \\ A_1 & \to & a_2 A_2 \\ & \vdots & \\ A_{n-1} & \to & a_n B \end{array}$$

De esta manera se puede construir una gramática regular equivalente a G. □

Ejercicios de la sección 4.6

1. Encontrar GIC regulares que generen los siguientes lenguajes:

   (i) ab^*a.

   (ii) $(ab \cup ba)^*$.

   (iii) $a^+b \cup b^+a^*b$.

   (iv) $0^*(10^* \cup 01^*)$.

2. Una GIC se llama **regular por la izquierda** si sus producciones son de la forma:

$$\begin{cases} A \to Bw, & w \in \Sigma^*,\ B \in V \\ A \to \lambda \end{cases}$$

   Demostrar que las gramáticas regulares y las gramáticas regulares por la izquierda generan los mismos lenguajes.

3. Completar los detalles de la demostración del Teorema 4.6.3.

4.7. Eliminación de las variables inútiles

En una GIC puede haber dos tipos de variables inútiles: aquéllas que nunca aparecen en el curso de una derivación y aquéllas que no se pueden convertir en cadenas de terminales. A continuación se precisan estos conceptos.

4.7.1. Definiciones.

(i) Una variable A es **alcanzable** o **accesible** si existen $u, v \in (V \cup \Sigma)^*$ tales que $S \overset{*}{\Longrightarrow} uAv$. La variable inicial S es alcanzable por definición.

(ii) Una variable A es **terminable** si existe $w \in \Sigma^*$ tal que $A \overset{*}{\Longrightarrow} w$. En particular, si $A \to \lambda$ es una producción entonces A es terminable.

(iii) A es una variable **inútil** si no es alcanzable o no es terminable.

Existen algoritmos para detectar todas las variables inútiles de una GIC que permiten construir una gramática G' equivalente a una gramática G dada, de tal manera que G' no tenga variables inútiles.

4.7.2. Algoritmo para encontrar las variables terminables.

El siguiente algoritmo sirve para encontrar todas las variables terminables de una GIC.

$\mathbf{TERM}_1 := \{A \in V : \text{Existe una producción de la forma } A \to w, w \in \Sigma^*\}$.

$\mathbf{TERM}_{i+1} := \mathbf{TERM}_i \cup \{A \in V : \exists \text{ producción } A \to w,\ w \in (\Sigma \cup \mathbf{TERM}_i)^*\}$.

Obtenemos una sucesión creciente de conjuntos de variables:

$$\mathbf{TERM}_1 \subseteq \mathbf{TERM}_2 \subseteq \mathbf{TERM}_3 \subseteq \cdots$$

Como el conjunto de variables es finito, existe k tal que

$$\mathbf{TERM}_k = \mathbf{TERM}_{k+1} = \mathbf{TERM}_{k+2} = \cdots$$

El conjunto **TERM** de variables terminables es entonces

$$\mathbf{TERM} := \bigcup_{i \geq 1} \mathbf{TERM}_i$$

El anterior algoritmo se puede presentar de la siguiente forma:

INICIALIZAR:

TERM $:= \{A \in V : \exists \text{ producción } A \to w, w \in \Sigma^*\}$

REPETIR:

TERM $:=$ **TERM** $\cup \{A \in V : \exists \text{ producción } A \to w, w \in (\Sigma \cup \mathbf{TERM})^*\}$

HASTA:

No se añaden nuevas variables a **TERM**

Ejemplo Encontrar las variables terminables de la siguiente gramática.

$$G: \begin{cases} S \to ACD \mid bBd \mid ab \\ A \to aB \mid aA \mid C \\ B \to aDS \mid aB \\ C \to aCS \mid CB \mid CC \\ D \to bD \mid ba \\ E \to AB \mid aDb \end{cases}$$

Solución:

$\mathbf{TERM}_1 = \{S, D\}$.
$\mathbf{TERM}_2 = \{S, D\} \cup \{B, E\} = \{S, D, B, E\}$.
$\mathbf{TERM}_3 = \{S, D, B, E\} \cup \{A\} = \{S, D, B, E, A\}$.
$\mathbf{TERM}_4 = \{S, D, B, E, A\} \cup \{\ \} = \{S, D, B, E, A\}$.
$\mathbf{TERM} = \{S, A, B, D, E\}$.

Conjunto de variables no terminables: $\{C\}$.

4.7.3. Algoritmo para encontrar las variables alcanzables.

El siguiente algoritmo sirve para encontrar todas las variables alcanzables de una GIC.
$\mathbf{ALC}_1 := \{S\}$.
$\mathbf{ALC}_{i+1} := \mathbf{ALC}_i \cup \{A \in V : \exists \text{ produc. } B \to uAv, B \in \mathbf{ALC}_i \; u, v \in (V \cup \Sigma)^*\}$.
Obtenemos una sucesión creciente de conjuntos de variables:

$$\mathbf{ALC}_1 \subseteq \mathbf{ALC}_2 \subseteq \mathbf{ALC}_3 \subseteq \cdots$$

Como el conjunto de variables es finito, existe k tal que

$$\mathbf{ALC}_k = \mathbf{ALC}_{k+1} = \mathbf{ALC}_{k+2} = \cdots$$

El conjunto **ALC** de variables alcanzables es entonces

$$\mathbf{ALC} = \bigcup_{i \geq 1} \mathbf{ALC}_i$$

El anterior algoritmo se puede presentar de la siguiente forma:

INICIALIZAR:
    ALC := $\{S\}$
REPETIR:
    ALC := **ALC** $\cup \{A \in V : \exists$ produc. $B \to uAv$, $B \in$ **ALC** y $u, v \in (V \cup \Sigma)^*\}$
HASTA:
    No se añaden nuevas variables a **ALC**

Ejemplo Encontrar las variables alcanzables de la siguiente gramática.

$$G: \begin{cases} S \to aS \mid AaB \mid ACS \\ A \to aS \mid AaB \mid AC \\ B \to bB \mid DB \mid BB \\ C \to aDa \mid ABD \mid ab \\ D \to aD \mid DD \mid ab \\ E \to FF \mid aa \\ F \to aE \mid EF \end{cases}$$

Solución:

$\mathbf{ALC}_1 = \{S\}$.
$\mathbf{ALC}_2 = \{S\} \cup \{A, B, C\} = \{S, A, B, C\}$.
$\mathbf{ALC}_3 = \{S, A, B, C\} \cup \{D\} = \{S, A, B, C, D\}$.
$\mathbf{ALC}_4 = \{S, A, B, C, D\} \cup \{\ \} = \{S, A, B, C, D\}$
$\mathbf{ALC} = \{S, A, B, C, D\}$.

Conjunto de variables no alcanzables: $\{E, F\}$.

Dada una GIC G, los dos algoritmos anteriores (algoritmo 4.7.2 y algoritmo algoritmo 4.7.3) se pueden llevar a cabo en dos órdenes diferentes:

(I) $G \xrightarrow{Algoritmo}$ Eliminar variables no-terminables $G_1 \xrightarrow{Algoritmo}$ Eliminar variables no-alcanzables G_2

(II) $G \xrightarrow{Algoritmo}$ Eliminar variables no-alcanzables $G_3 \xrightarrow{Algoritmo}$ Eliminar variables no-terminables G_4

Esto da lugar a las siguientes preguntas:

(1) ¿Es $G_2 = G_4$?

(2) ¿G_2 tiene variables inútiles?

(3) ¿G_4 tiene variables inútiles?

El siguiente ejemplo muestra que la respuesta a la pregunta (1) es *no* y que al realizar los algoritmos en el orden (II) pueden permanecer en G_4 algunas variables inútiles.

Ejemplo Considérese la siguiente gramática G.

$$G: \quad \begin{cases} S \to a \mid AB \\ A \to aA \mid \lambda \end{cases}$$

Aplicación de los algoritmos en el orden (I):
Variables terminables de G: **TERM**$= \{S, A\}$.

$$G_1: \quad \begin{cases} S \to a \\ A \to aA \mid \lambda \end{cases}$$

Variables alcanzables de G_1: **ALC**$= \{S\}$.

$$G_2: \quad \{S \to a$$

Se tiene que $L(G_2) = \{a\}$.

Aplicación de los algoritmos en el orden (II):
Variables alcanzables de G: **ALC**$=\{S, A, B\}$.

$$G_3: \quad \begin{cases} S \to a \mid AB \\ A \to aA \mid \lambda \end{cases}$$

Variables terminables de G_3: **TERM**$=\{S, A\}$.

$$G_4: \quad \begin{cases} S \to a \\ A \to aA \mid \lambda \end{cases}$$

Se observa que en G_4, A no es alcanzable. Además, $G_2 \neq G_4$.

No obstante, si los algoritmos se llevan a cabo en el orden (I) la gramática G_2 ya no tiene variables inútiles. Nos podemos convencer de eso observando que las variables alcanzables obtenidas al finalizar el procedimiento siguen siendo terminables. Más precisamente, si una variable A permanece al finalizar el procedimiento completo, será es alcanzable, y si la derivación $A \stackrel{*}{\Longrightarrow} w$, con $w \in \Sigma^*$, se podía hacer antes de eliminar las variables no alcanzables, también se podrá realizar en la gramática final ya que todas las variables que aparezcan en esa derivación serán alcanzables.

Ejemplo Eliminar las variables inútiles de la siguiente gramática G.

$$G: \quad \begin{cases} S \to SBS \mid BC \mid Bb \\ A \to AA \mid aA \\ B \to aBCa \mid b \\ C \to aC \mid ACC \mid abb \\ D \to aAB \mid ab \\ E \to aS \mid bAA \\ F \to aDb \mid aF \end{cases}$$

Solución: Ejecutamos los algoritmos en el orden (I):

$$\begin{aligned} \mathbf{TERM}_1 &= \{B, C, D\}. \\ \mathbf{TERM}_2 &= \{B, C, D\} \cup \{S, F\}. \\ \mathbf{TERM}_3 &= \{B, C, D, S, F\} \cup \{E\} = \{B, C, D, S, F, E\}. \\ \mathbf{TERM}_4 &= \{B, C, D, S, F, E\} \cup \{\ \}. \end{aligned}$$

La única variable no-terminable de G es A. Por lo tanto, G es equivalente a la siguiente gramática G_1:

$$G_1: \quad \begin{cases} S \to SBS \mid BC \mid Bb \\ B \to aBCa \mid b \\ C \to aC \mid abb \\ D \to ab \\ E \to aS \\ F \to aDb \mid aF \end{cases}$$

Variables alcanzables de G_1:

$$\begin{aligned} \mathbf{ALC}_1 &= \{S\}. \\ \mathbf{ALC}_2 &= \{S\} \cup \{B, C\}. \\ \mathbf{ALC}_3 &= \{S, B, C\} \cup \{\ \}. \end{aligned}$$

Las variables D, E, F son no alcanzables. Por lo tanto, G es equivalente a la siguiente gramática G_2, que no tiene variables inútiles.

$$G_2: \begin{cases} S \to SBS \mid BC \mid Bb \\ B \to aBCa \mid b \\ C \to aC \mid abb \end{cases}$$

Ejercicios de la sección 4.7

1. Eliminar las variables inútiles de la siguiente gramática:

$$G: \begin{cases} S \to SS \mid SBB \mid CCE \\ A \to aE \mid bE \\ B \to bB \mid Db \\ C \to aC \mid bB \\ D \to aDb \mid ab \mid \lambda \\ E \to aA \mid bB \end{cases}$$

2. Eliminar las variables inútiles de la siguiente gramática:

$$G: \begin{cases} S \to EA \mid SaBb \mid aEb \\ A \to DaD \mid bD \\ B \to bB \mid Ab \mid \lambda \\ C \to aC \mid bBC \\ D \to aEb \mid ab \\ E \to aA \mid bB \mid \lambda \\ F \to Fb \mid Fa \mid a \end{cases}$$

4.8. Eliminación de las producciones λ

4.8.1. Definiciones.

(i) Una producción de la forma $A \to \lambda$ se llama **producción** λ.

(ii) Una variable A se llama **anulable** si $A \overset{*}{\Longrightarrow} \lambda$.

4.8.2. Algoritmo para encontrar las variables anulables.

$\mathbf{ANUL}_1 := \{A \in V : A \to \lambda \text{ es una producción}\}$.
$\mathbf{ANUL}_{i+1} := \mathbf{ANUL}_i \cup \{A \in V : \exists \text{ producción } A \to w, w \in (\mathbf{ANUL}_i)^*\}$.
Obtenemos una sucesión creciente de conjuntos de variables:

$$\mathbf{ANUL}_1 \subseteq \mathbf{ANUL}_2 \subseteq \mathbf{ANUL}_3 \subseteq \cdots$$

Como el conjunto de variables es finito, existe $k \in \mathbb{N}$ tal que

$$\mathbf{ANUL}_k = \mathbf{ANUL}_{k+1} = \mathbf{ANUL}_{k+2} = \cdots$$

El conjunto **ANUL** de variables anulables es entonces

$$\mathbf{ANUL} := \bigcup_{i \geq 1} \mathbf{ANUL}_i$$

El anterior algoritmo se puede presentar de la siguiente forma:

INICIALIZAR:
    ANUL := $\{A \in V : A \to \lambda$ es una producción$\}$
REPETIR:
    ANUL := **ANUL** $\cup \{A \in V : \exists$ prod. $A \to w,\ w \in (\mathbf{ANUL})^*\}$
HASTA:
    No se añaden nuevas variables a **ANUL**

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

Demostración: Una vez que haya sido determinado el conjunto **ANUL** de variables anulables, por medio del algoritmo 4.8.2, las producciones de λ se pueden eliminar (excepto $S \to \lambda$) añadiendo nuevas producciones que simulen el efecto de las producciones λ eliminadas. Más concretamente, por cada producción $A \to u$ de G se añaden las producciones de la forma $A \to v$ 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. □

Ejemplo Eliminar las producciones λ de la siguiente gramática G.

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

Solución: Primero encontramos las variables anulables de G por medio del algoritmo 4.8.2:

$$\begin{aligned}
\mathbf{ANUL}_1 &= \{C\}.\\
\mathbf{ANUL}_2 &= \{C\} \cup \{D\} = \{C, D\}.\\
\mathbf{ANUL}_3 &= \{C, D\} \cup \{A\} = \{C, D, A\}.\\
\mathbf{ANUL}_4 &= \{C, D, A\} \cup \{S\} = \{C, D, A, S\}.\\
\mathbf{ANUL}_5 &= \{C, D, A, S\} \cup \{\ \} = \{C, D, A, S\}.
\end{aligned}$$

Al eliminar de G la producciones λ (la única es $C \to \lambda$) se obtiene la siguiente gramática equivalente a G:

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

Ejercicios de la sección 4.8

1. Eliminar las producciones λ de la siguiente gramática:

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

2. Eliminar las producciones λ de la siguiente gramática:

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

4.9. Eliminación de las producciones unitarias

4.9.1. Definiciones.

(i) Una producción de la forma $A \to B$ donde A y B son variables, se llama **producción unitaria**.

(ii) El **conjunto unitario** de una variable A (también llamado **conjunto cadena** de A) se define de la siguiente manera:

$$\mathbf{UNIT}(A) := \{X \in V : \exists \text{ una derivación } A \overset{*}{\Longrightarrow} X \text{ que usa únicamente producciones unitarias}\}.$$

Por definición, $A \in \mathbf{UNIT}(A)$.

4.9.2. Algoritmo para encontrar las producciones unitarias.

El siguiente algoritmo sirve para encontrar el conjunto unitario $\mathbf{UNIT}(A)$ de una variable A.

$\mathbf{UNIT}_1(A) := \{A\}$.
$\mathbf{UNIT}_{i+1}(A) := \mathbf{UNIT}_i(A) \cup \{X \in V : \exists \text{ producción } Y \to X,\ Y \in \mathbf{UNIT}_i(A)\}$.

Para el conjunto de producciones unitarias se tiene que:

$$\mathbf{UNIT}_1(A) \subseteq \mathbf{UNIT}_2(A) \subseteq \mathbf{UNIT}_3(A) \subseteq \cdots$$

Puesto que el conjunto de variables es finito, la anterior es una sucesión finita y se tiene

$$\mathbf{UNIT}(A) = \bigcup_{i \geq 1} \mathbf{UNIT}_i(A)$$

El anterior algoritmo se puede representar de la siguiente forma:

INICIALIZAR:
    UNIT(A):={A}
REPETIR:
    UNIT(A):= $\mathbf{UNIT}(A) \cup \{X \in V : \exists \text{ una produc. } Y \to X \text{ con } Y \in \mathbf{UNIT}(A)\}$
HASTA:
    No se añaden nuevas variables $\mathbf{UNIT}(A)$

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

Demostración: Las producciones unitarias de G se pueden eliminar añadiendo para cada variable A de G las producciones (no unitarias) de las variables contenidas en el conjunto unitario $\mathbf{UNIT}(A)$. La gramática G' así obtenida es equivalente a la gramática original G. □

Ejemplo Eliminar las producciones unitarias de la siguiente gramática.

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

Solución: Aplicando el algoritmo para cada una de las variables de G, se tiene que:

$$\begin{array}{l} \mathbf{UNIT}_1(S) = \{S\}. \\ \mathbf{UNIT}_2(S) = \{S\} \cup \{\ \} = \{S\}. \\ \mathbf{UNIT}_1(A) = \{A\}. \\ \mathbf{UNIT}_2(A) = \{A\} \cup \{\ \} = \{A\}. \\ \mathbf{UNIT}_1(B) = \{B\}. \\ \mathbf{UNIT}_2(B) = \{B\} \cup \{C\} = \{B, C\}. \\ \mathbf{UNIT}_3(B) = \{B, C\} \cup \{B\} = \{B, C\}. \\ \mathbf{UNIT}_1(C) = \{C\}. \\ \mathbf{UNIT}_2(C) = \{C\} \cup \{B\} = \{C, B\}. \\ \mathbf{UNIT}_3(C) = \{C, B\} \cup \{C\} = \{C, B\}. \end{array}$$

Eliminando las producciones unitarias se obtiene una gramática G' equivalente:

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

Ejemplo Eliminar las producciones unitarias de la siguiente gramática.

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

Solución: Realizando el algoritmo para cada una de las variables de G se obtiene:

$$\begin{aligned}
&\mathbf{UNIT}(S) = \{S, A, C, B, D\}.\\
&\mathbf{UNIT}(A) = \{A, B, C, D\}.\\
&\mathbf{UNIT}(B) = \{B, C, D\}.\\
&\mathbf{UNIT}(C) = \{C\}.\\
&\mathbf{UNIT}(D) = \{D\}.
\end{aligned}$$

Eliminando las producciones unitarias se obtiene una gramática G' equivalente:

$$G' \quad \begin{cases} S \to ACA \mid CA \mid AA \mid \lambda \mid aAa \mid aa \mid bC \mid cC \mid aA \\ A \to aAa \mid aa \mid cC \mid bC \mid aA \mid \lambda \\ B \to cC \mid bC \mid aA \mid \lambda \\ C \to bC \\ D \to aA \mid \lambda \end{cases}$$

Ejercicios de la sección 4.9

1. Eliminar las producciones unitarias de la siguiente gramática:

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

2. Eliminar las producciones unitarias de la siguiente gramática:

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

3. Eliminar las producciones unitarias de la siguiente gramática:

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

4.10. Forma Normal de Chomsky (FNC)

Una GIC G está en **Forma Normal de Chomsky** (FNC) si satisface:

1. G no tiene variables inútiles.
2. G no tiene producciones λ (excepto posiblemente $S \to \lambda$).
3. Todas las producciones son de la forma: $A \to a$ (producciones simples) ó $A \to BC$ (producciones binarias).

En particular, una gramática en FNC no tiene producciones unitarias.

4.10.1 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, ejecutando los algoritmos de las secciones anteriores en el siguiente orden:

1. Eliminar las variables no terminales.
2. Eliminar las variables no alcanzables.
3. Eliminar las producciones λ (excepto, posiblemente, $S \to \lambda$).
4. Eliminar las producciones unitarias.
5. Las producciones resultantes (diferentes de $S \to \lambda$) son de la forma: $A \to a$ ó $A \to w$, donde $|w| \geq 2$. Estas últimas se pueden simular con producciones de la forma $A \to BC$ o $A \to a$. Se introduce primero, para cada $a \in \Sigma$, una variable nueva T_a cuya única producción es $T_a \to a$. A continuación, se introducen nuevas variables, con producciones binarias, para simular las producciones deseadas. □

La parte 5 del procedimiento anterior se ilustra en los dos siguientes ejemplos.

Ejemplo Simular la producción $A \to abBaC$ con producciones simples y binarias.

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

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

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:

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

Ejemplo Simular la producción $A \to BAaCbb$ con producciones simples y binarias.

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

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

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:

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

En los siguientes ejemplos 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:

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

Solución: El conjunto de variables terminables es

$$\mathbf{TERM} = \{B, C, S, A\},$$

y el conjunto de variables alcanzables es

$$\mathbf{ALC} = \{S, A, B, C\}.$$

Es decir, la gramática no tiene variables inútiles. El conjunto de variables anulables es

$$\mathbf{ANUL} = \{C, A\}.$$

Al eliminar las producciones λ de G (la única es $C \to \lambda$) se obtiene la gramática equivalente G_1:

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

A continuación encontramos los conjuntos unitarios de todas las variables:

$$\begin{array}{l} \mathbf{UNIT}(S) = \{S, B\}. \\ \mathbf{UNIT}(A) = \{A, C\}. \\ \mathbf{UNIT}(B) = \{B\}. \\ \mathbf{UNIT}(C) = \{C\}. \end{array}$$

Al eliminar las producciones unitarias obtenemos la gramática equivalente G_2:

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

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

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

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

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

Ejemplo Encontrar una GIC en FNC equivalente a la siguiente a la gramática:

$$G: \begin{cases} S \to aS \mid aA \mid D \\ A \to aAa \mid aAD \mid \lambda \\ B \to aB \mid BC \\ C \to aBb \mid CC \mid \lambda \\ D \to aB \mid bA \mid aa \mid A \end{cases}$$

Solución: $\mathbf{TERM} = \{A, C, D, S\}$. Eliminando la variable no-terminable B obtenemos:

$$G_1: \begin{cases} S \to aS \mid aA \mid D \\ A \to aAa \mid aAD \mid \lambda \\ C \to CC \mid \lambda \\ D \to bA \mid aa \mid A \end{cases}$$

El conjunto de las variables alcanzables de G_1 es $\mathbf{ALC} = \{S, A, D\}$. Eliminando la variable no-alcanzable C obtenemos:

$$G_2: \begin{cases} S \to aS \mid aA \mid D \\ A \to aAa \mid aAD \mid \lambda \\ D \to bA \mid aa \mid A \end{cases}$$

El conjunto de variables anulables de G_2 es $\mathbf{ANUL} = \{A, D, S\}$. Eliminando las producciones λ obtenemos:

$$G_3: \begin{cases} S \to aS \mid aA \mid D \mid a \mid \lambda \\ A \to aAa \mid aAD \mid aa \mid aA \mid aD \mid a \\ D \to bA \mid aa \mid A \mid b \end{cases}$$

A continuación encontramos los conjuntos unitarios de todas las variables:

$$\begin{array}{l}\mathbf{UNIT}(S) = \{S, D, A\}.\\ \mathbf{UNIT}(A) = \{A\}.\\ \mathbf{UNIT}(D) = \{D, A\}.\end{array}$$

Al eliminar las producciones unitarias obtenemos la gramática equivalente G_4:

$$G_4: \quad \begin{cases} S \to aS \mid aA \mid a \mid \lambda \mid aAa \mid aAD \mid aa \mid aD \mid bA \mid b \\ A \to aAa \mid aAD \mid aa \mid aA \mid aD \mid a \\ D \to bA \mid aa \mid b \mid aAa \mid aAD \mid aa \mid aA \mid aD \mid a \end{cases}$$

Finalmente, simulamos las producciones de G_4 con producciones unitarias y binarias:

$$G_5: \quad \begin{cases} S \to T_aS \mid T_aA \mid T_aT_1 \mid T_aT_2 \mid T_aT_a \mid T_aD \mid T_bA \mid a \mid b \mid \lambda \\ A \to T_aT_1 \mid T_aT_2 \mid T_aT_a \mid T_aA \mid T_aD \mid a \\ D \to T_bA \mid T_aT_a \mid b \mid T_aT_1 \mid T_aT_2 \mid T_aT_a \mid T_aA \mid T_aD \mid a \\ T_1 \to AT_a \\ T_2 \to AD \\ T_a \to a \\ T_b \to b \end{cases}$$

En algunas aplicaciones de la FNC es necesario exigir que la variable inicial S no aparezca en el cuerpo de ninguna producción. Si S aparece en el lado derecho de alguna producción se dice que S es **recursiva** ya que esto da lugar a derivaciones de la forma $S \overset{+}{\Longrightarrow} uSv$, con $u, v \in (V \cup \Sigma)^*$. 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.

4.10.2 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 $P' = P \cup \{S' \to S\}$. Es claro que $L(G) = L(G')$ y el símbolo inicial S' no aparece en el cuerpo de las producciones. □

Según este resultado, el papel de la variable inicial de la nueva gramática G' es únicamente iniciar las derivaciones.

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

$$G: \quad \begin{cases} S \to ASB \mid BB \\ A \to aA \mid a \\ B \to bBS \mid \lambda \end{cases}$$

Solución: Según se indicó en la demostración del Teorema 4.10.2, la gramática pedida G' es

$$G': \quad \begin{cases} S' \to S \\ S \to ASB \mid BB \\ A \to aA \mid a \\ B \to bBS \mid \lambda \end{cases}$$

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

Ejemplo Encontrar una GIC en FNC equivalente a la gramática G del ejemplo anterior, de tal manera que su variable inicial no sea recursiva.

Solución: Comenzamos transformando G en G', como se hizo en el ejemplo anterior. En G' todas las variable son útiles y $\mathbf{ANUL} = \{B, S, S'\}$. Eliminando las producciones λ obtenemos:

$$G_1: \quad \begin{cases} S' \to S \mid \lambda \\ S \to ASB \mid BB \mid AB \mid AS \mid A \\ A \to aA \mid a \\ B \to bBS \mid bS \mid bB \mid b \end{cases}$$

Los conjuntos unitarios son:
$\mathbf{UNIT}(S') = \{S', S\}$, $\mathbf{UNIT}(S) = \{S, A\}$, $\mathbf{UNIT}(A) = \{A\}$, $\mathbf{UNIT}(B) = \{B\}$.
Eliminando las producciones unitarias se obtiene la gramática:

$$G_2: \quad \begin{cases} S' \to ASB \mid BB \mid AB \mid AS \mid aA \mid a \mid \lambda \\ S \to ASB \mid BB \mid AB \mid AS \mid aA \mid a \\ A \to aA \mid a \\ B \to bBS \mid bS \mid bB \mid b \end{cases}$$

Simulando las producciones de G_2 con producciones unitarias y binarias se obtiene:

$$G_3: \begin{cases} S' \to AT_1 \mid BB \mid AB \mid AS \mid T_aA \mid a \mid \lambda \\ S \to AT_1 \mid BB \mid AB \mid AS \mid T_aA \mid a \\ A \to T_aA \mid a \\ B \to T_bT_2 \mid T_bS \mid T_bB \mid b \\ T_a \to a \\ T_b \to b \\ T_1 \to SB \\ T_2 \to BS \end{cases}$$

Ejercicios de la sección 4.10

1. Encontrar una gramática en FNC equivalente a la siguiente GIC:

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

2. Encontrar una gramática en FNC equivalente a la siguiente GIC:

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

3. Para la gramática del ejercicio 2 anterior encontrar una GIC equivalente en FNC, de tal manera que su variable inicial no sea recursiva.

4.11. Forma Normal de Greibach (FNG)

Una GIC está en **Forma Normal de Greibach** (FNG) si

1. La variable inicial no es recursiva.
2. G no tiene variables inútiles.
3. G no tiene producciones λ (excepto posiblemente $S \to \lambda$).

4. Todas las producciones son de la forma: $A \to a$ (producciones simples) ó $A \to aB_1B_2 \dots B_k$, donde las B_i son variables.

Las derivaciones en una gramática que esté en FNG tienen dos características notables: en cada paso aparece un único terminal y, en segundo lugar, la derivación de una cadena de longitud n ($n \geq 1$) tiene exactamente n pasos.

Existe un procedimiento algorítmico para transformar una GIC dada en una gramática equivalente en FNG. Para presentar el procedimiento necesitamos algunos resultados preliminares.

4.11.1 Definición. Una variable se llama **recursiva a la izquierda** si tiene una producción de la forma:

$$A \to Aw, \quad w \in (V \cup \Sigma)^*.$$

La recursividad a izquierda se puede eliminar, como se muestra en el siguiente teorema.

4.11.2 Teorema (Eliminación de la recursividad a la izquierda). *Las producciones de una variable A cualquiera se pueden dividir en dos clases:*

$$\begin{cases} A \to A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_n \\ A \to \beta_1 \mid \beta_2 \mid \cdots \mid \beta_m \end{cases}$$

donde $\alpha_i, \beta_i \in (V \cup \Sigma)^*$ y el primer símbolo de β_i es diferente de A. Sin alterar el lenguaje generado, las anteriores producciones se pueden simular, reemplazándolas por las siguientes:

$$\begin{cases} A \to \beta_1 \mid \beta_2 \mid \cdots \mid \beta_m \mid \beta_1 Z \mid \beta_2 Z \mid \cdots \mid \beta_m Z \\ Z \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n \mid \alpha_1 Z \mid \alpha_2 Z \mid \cdots \mid \alpha_n Z \end{cases}$$

donde Z es una variable completamente nueva.

<u>*Demostración*</u>: Se puede observar que, tanto con las producciones originales como las nuevas, A genera el lenguaje

$$\{\beta_1, \beta_2, \dots, \beta_m\} \cdot \{\alpha_1, \alpha_2, \dots, \alpha_n\}^* \qquad \square$$

4.11.3 Lema. *En una GIC cualquiera, una producción $A \to uBv$ se puede reemplazar (simular) por*

$$A \to uw_1v \mid uw_2v \mid \cdots \mid uw_nv$$

siendo $B \to w_1 \mid w_2 \mid \cdots \mid w_n$ todas las producciones de B.

Demostración: Inmediato. □

4.11.4 Teorema (Procedimiento de conversión a FNG). *Toda GIC G es equivalente a una gramática en Forma Normal de Greibach.*

Demostración: Suponemos que la gramática dada está en FNC. Esto simplifica el procedimiento, aunque éste es válido (con modificaciones menores) para una gramática arbitraria si se eliminan primero las variables inútiles, las producciones λ y las producciones unitarias. La conversión a FNG se realiza ejecutando los siguientes pasos:

1. Enumerar las variables en un orden arbitrario pero fijo durante el procedimiento. S debe ser la variable con orden 1.
2. Para cada variable A de la gramática original, siguiendo el orden elegido, modificar sus producciones de tal manera que el primer símbolo del cuerpo de cada producción (primer símbolo a la derecha de la flecha) sea un terminal o una variable con orden mayor que el de A. Para lograrlo se usa el teorema de eliminación de la recursividad a la izquierda, Teorema 4.11.2, y el Lema 4.11.3, todas las veces que sea necesario.
3. Utilizar el Lema 4.11.3 para modificar las producciones de las variables *originales* de tal manera que el primer símbolo del cuerpo de cada producción sea un terminal. Esto se hace siguiendo el orden inverso de enumeración de las variables: última, penúltima, etc.
4. Utilizar de nuevo el Lema 4.11.3, para modificar las producciones de las variables *nuevas* de tal manera que el primer símbolo del cuerpo de cada producción sea un terminal. □

Ejemplo Encontrar una gramática en FNG equivalente a la siguiente gramática (que está en FNC):

$$G: \quad \begin{cases} S \to AA \mid a \\ A \to AA \mid b \end{cases}$$

Paso 1: Aquí solamente hay un orden posible para variables: S, A.

Paso 2: En este paso sólo hay que eliminar la recursividad a izquierda de la variable A. Al hacerlo se obtiene la gramática:

$$\begin{cases} S \to AA \mid a \\ A \to b \mid bZ \\ Z \to A \mid AZ \end{cases}$$

Paso 3:

$$\begin{cases} S \to bA \mid bZA \mid a \\ A \to b \mid bZ \\ Z \to A \mid AZ \end{cases}$$

Paso 4:

$$\begin{cases} S \to bA \mid bZA \mid a \\ A \to b \mid bZ \\ Z \to b \mid bZ \mid bZZ \end{cases}$$

Ejemplo Encontrar una gramática en FNG equivalente a la siguiente gramática (que está en FNC):

$$\begin{cases} S \to AB \mid BC \\ A \to AB \mid a \\ B \to AA \mid CB \mid a \\ C \to a \mid b \end{cases}$$

Paso 1: Orden de las variables: S, B, A, C. Este orden es muy adecuado porque el cuerpo de las producciones de B comienza con A o C, que son variables de orden mayor.

$$\begin{cases} S \to AB \mid BC \\ B \to AA \mid CB \mid a \\ A \to AB \mid a \\ C \to a \mid b \end{cases}$$

Paso 2:

$$\begin{cases} S \to AB \mid BC \\ B \to AA \mid CB \mid a \\ A \to a \mid aZ \\ C \to a \mid b \\ Z \to B \mid BZ \end{cases}$$

Paso 3:

$$\begin{cases} S \to aB \mid aZB \mid aAC \mid aZAC \mid aBC \mid bBC \mid aC \\ B \to aA \mid aZA \mid aB \mid bB \mid a \\ A \to a \mid aZ \\ C \to a \mid b \\ Z \to B \mid BZ \end{cases}$$

Paso 4:

$$\begin{cases} S \to aB \mid aZB \mid aAC \mid aZAC \mid aBC \mid bBC \mid aC \\ B \to aA \mid aZA \mid aB \mid bB \mid a \\ A \to a \mid aZ \\ C \to a \mid b \\ Z \to aA \mid aZA \mid aB \mid bB \mid a \mid aAZ \mid aZAZ \mid aBZ \mid bBZ \mid aZ \end{cases}$$

El siguiente ejemplo ilustra que el procedimiento de conversión a la forma FNG puede dar lugar a docenas de producciones, incluso a partir de una gramática relativamente sencilla.

Ejemplo Encontrar una gramática en FNG equivalente a la siguiente gramática:

$$\begin{cases} S \to AB \\ A \to AB \mid CB \mid a \\ B \to AB \mid b \\ C \to AC \mid c \end{cases}$$

Paso 1: Orden de las variables: S, A, B, C.

Paso 2:

$$\begin{cases} S \to AB \\ A \to CB \mid a \mid CBZ_1 \mid aZ_1 \\ B \to CBB \mid aB \mid CBZ_1B \mid aZ_1B \mid b \\ C \to CBC \mid aC \mid CBZ_1C \mid aZ_1C \mid c \\ Z_1 \to B \mid BZ_1 \end{cases}$$

Prosiguiendo con el paso 2, se elimina la recursividad a izquierda de la variable C:

$$\begin{cases} S \to AB \\ A \to CB \mid a \mid CBZ_1 \mid aZ_1 \\ B \to CBB \mid aB \mid CBZ_1B \mid aZ_1B \mid b \\ C \to aC \mid aZ_1C \mid c \mid aCZ_2 \mid aZ_1CZ_2 \mid cZ_2 \\ Z_1 \to B \mid BZ_1 \\ Z_2 \to BC \mid BZ_1C \mid BCZ_2 \mid BZ_1CZ_2 \end{cases}$$

Paso 3:

$$\begin{cases} S \to 14 \text{ producciones} \\ A \to a \mid aZ_1 \mid 6 \text{ producciones} \mid 6 \text{ producciones} \\ B \to aB \mid aZ_1B \mid b \mid 6 \text{ producciones} \mid 6 \text{ producciones} \\ C \to aC \mid aZ_1C \mid c \mid aCZ_2 \mid aZ_1CZ_2 \mid cZ_2 \\ Z_1 \to B \mid BZ_1 \\ Z_2 \to BC \mid BZ_1C \mid BCZ_2 \mid BZ_1CZ_2 \end{cases}$$

Paso 4: El número de producciones de la nueva gramática se incrementa drásticamente:

$$\begin{cases} S \to 14 \text{ producciones} \\ A \to a \mid aZ_1 \mid 6 \text{ producciones} \mid 6 \text{ producciones} \\ B \to aB \mid aZ_1B \mid b \mid 6 \text{ producciones} \mid 6 \text{ producciones} \\ C \to aC \mid aZ_1C \mid c \mid aCZ_2 \mid aZ_1CZ_2 \mid cZ_2 \\ Z_1 \to 15 \text{ producciones} \mid 15 \text{ producciones} \\ Z_2 \to 15 \text{ producciones} \mid 15 \text{ producciones} \mid 15 \text{ producciones} \mid \\ \qquad 15 \text{ producciones} \end{cases}$$

La gramática original tenía 8 producciones; la nueva gramática en FNG tiene un total de 139 producciones.

Ejercicios de la sección 4.11

1. Encontrar una gramática en FNG equivalente a la siguiente GIC:

$$\begin{cases} S \to CA \mid AC \mid a \\ A \to BA \mid AB \mid b \\ B \to AA \mid a \mid b \\ C \to AC \mid CC \mid a \end{cases}$$

2. Encontrar una gramática en FNG equivalente a la siguiente GIC:

$$\begin{cases} S \to BB \mid BC \mid b \\ A \to AC \mid CA \mid a \\ B \to BB \mid a \\ C \to BC \mid CA \mid a \end{cases}$$

3. Encontrar una gramática en FNG equivalente a la siguiente GIC:

$$\begin{cases} S \to SC \mid AA \mid a \\ A \to CA \mid AB \mid a \\ B \to AC \mid b \\ C \to CA \mid AS \mid b \end{cases}$$

Nota: hay que eliminar primero la recursividad de la variable S.

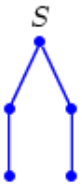
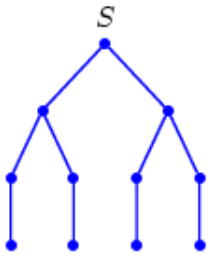
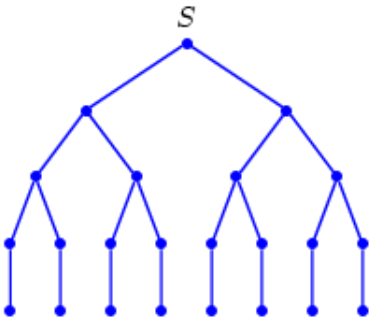
4.12. Lema de bombeo para LIC

Una de las consecuencias más importantes de la Forma Normal de Chomsky es el lema de bombeo para lenguajes independientes del contexto, el cual es útil, entre muchas aplicaciones, para demostrar que ciertos lenguajes no son LIC.

Nos referiremos a gramáticas en FNC con variable inicial no recursiva. Puesto que las producciones son unitarias ($A \to a$) o binarias ($A \to BC$), en cada nodo el árbol de una derivación se ramifica en dos nodos, a lo sumo. Tales árboles se denominan **binarios**. Si la producción $S \to \lambda$ está presente, su único propósito es generar la cadena λ.

4.12.1 Teorema. *Sea $G = (V, \Sigma, S, P)$ una gramática en FNC y $w \in \Sigma^*$. Si la longitud de la trayectoria más larga en un árbol de derivación de $S \overset{*}{\Longrightarrow} w$ tiene k (o menos) nodos, entonces $|w| \leq 2^{k-2}$. Aquí $k \geq 2$.*

Demostración: La siguiente tabla muestra las relaciones obtenidas entre k = número de nodos de la trayectoria más larga de $S \overset{*}{\Longrightarrow} w$ y la longitud de w, 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 $|w| \leq 2^{k-2}$. Una demostración rigurosa del caso general se hace por inducción sobre k. □

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

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

(1) Si la longitud de la trayectoria más larga en un árbol de derivación de $S \overset{*}{\Longrightarrow} w$ tiene $k+2$ (o menos) nodos, entonces $|w| \leq 2^k$. Aquí $k \geq 0$.

(2) Si $|w| > 2^k$ (con $k \geq 0$) entonces la longitud de la trayectoria más larga en un árbol de derivación de $S \overset{*}{\Longrightarrow} w$ tiene más de $k+2$ nodos.

Demostración:

(1) Se sigue inmediatamente del Teorema 4.12.1.

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

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

(1) $|vwx| \leq n$.

(2) $uv^iwx^iy \in L$ *para todo* $i \geq 0$.

(3) $v \neq \lambda$ *ó* $x \neq \lambda$.

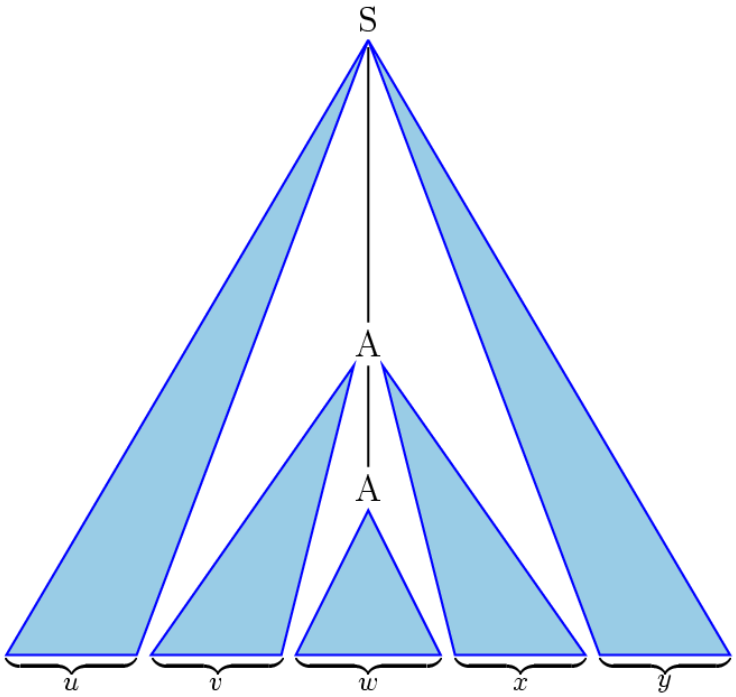
Demostración: Sea $G = (V, \Sigma, S, P)$ una gramática en FNC, con variable inicial no recursiva, tal que $L(G) = L$. Tal gramática existe por el Teorema 4.10.1 y el Teorema 4.10.2. Sea $k = |V|$ = número de variables de G y $n = 2^k$. Sea $z \in L$ con $|z| > n = 2^k$. Por la parte (2) del Corolario 4.12.2, la trayectoria más larga en el árbol de una derivación $S \overset{*}{\Longrightarrow} z$ tiene más de $k + 2$ nodos. Consideremos los últimos $k + 2$ nodos de tal trayectoria (siguiendo el orden que va desde la raíz S hasta las hojas del árbol). El último nodo de esa trayectoria es un terminal $a \in \Sigma$ y los restantes $k + 1$ nodos son variables. Como hay sólo k variables en la gramática, entonces hay por lo menos una variable $A \neq S$ repetida en la trayectoria. Por lo tanto, existen cadenas $u, v, w, x, y \in \Sigma^*$ tales que

$$S \overset{*}{\Longrightarrow} uAy, \qquad A \overset{*}{\Longrightarrow} vAx, \qquad A \overset{*}{\Longrightarrow} w.$$

Así que

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

La siguiente gráfica ilustra la situación:



Puesto que la trayectoria más larga en el árbol de derivación de $A \stackrel{*}{\Longrightarrow} vAx \stackrel{*}{\Longrightarrow} vwx$ tiene $k+2$ nodos o menos, por la parte (1) del Corolario Corolario 4.12.2, podemos concluir que $|vwx| \leq 2^k = n$. Además:

$$S \stackrel{*}{\Longrightarrow} uAy \stackrel{*}{\Longrightarrow} uvAxy \stackrel{*}{\Longrightarrow} uv^iAx^iy \stackrel{*}{\Longrightarrow} uv^iwx^iy, \quad \text{para todo } i \geq 0.$$

Obsérvese que el caso $i = 0$ corresponde a la derivación $S \stackrel{*}{\Longrightarrow} uAy \stackrel{*}{\Longrightarrow} uwy$.

Finalmente, la derivación $A \stackrel{*}{\Longrightarrow} vAx$ se puede escribir como

$$A \Rightarrow BC \stackrel{*}{\Longrightarrow} vAx$$

utilizando una producción de la forma $A \to BC$ como primer paso. Se deduce que u y x no pueden ser ambas λ porque se tendría $BC \stackrel{*}{\Longrightarrow} A$, lo cual es imposible en una gramática en FNC (recuérdese que la única producción λ en la gramática es, posiblemente, $S \to \lambda$; pero S no aparece en el cuerpo de ninguna producción de G ya que S no es recursiva). Se deduce entonces que $v \neq \lambda$ ó $x \neq \lambda$. Esto demuestra las propiedades (1), (2) y (3) del lema de bombeo. □

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

Solución: Argumento por contradicción. Si L fuera LIC, por el lema de bombeo, existiría una constante de bombeo n. Sea $z = a^nb^nc^n$; se tiene que $z \in L$ y $|z| > n$. Por lo tanto, z se puede descomponer como $z = a^nb^nc^n = uvwxy$ 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 $a^nb^nc^n$ 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 $b$es seguidas de $a$es o algunas $c$es seguidas de $b$es. 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 $a$es o sólo $b$es o sólo $c$es). 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: Argumento por contradicción. Si L fuera LIC, por el lema de bombeo, existiría una constante de bombeo n. Sea $z = a^m$ con m primo $m > n$ y $m > 2$ (m 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 $z = a^m = uvwxy$ con las propiedades (1), (2) y (3) del lema de bombeo.

Sea $|u| + |w| + |y| = k$; entonces $|v| + |x| = m - k \geq 1$. Por el 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

$$|uv^iwx^iy| = k + |v^i| + |x^i| = k + i|v| + i|x| = k + i(|v| + |x|) = k + i(m - k).$$

(i) Si $k = 0$, tomando $i = m$ se obtiene que $k + i(m-k) = 0 + i(m-0) = im = mm$ que no es primo, pues $m > 2$.

(ii) Si $k = 1$, tomando $i = 0$ se obtiene que $k+i(m-k) = 1+i(m-1) = 1+0(m-1) = 1$ que no es un número primo.

(iii) Si $k > 1$, tomando $i = k$ se obtiene que

$$|uv^kwx^ky| = k + k(m - k) = k(1 + m - k),$$

el cual no es un número primo pues $k > 1$ y como $m-k \geq 1$, entonces $1+m-k \geq 2$.

Por (i), (ii) y (iii) se puede escoger i de tal manera que $|uv^iwx^iy|$ no sea un número primo, lo cual contradice que $uv^iwx^iy \in L$ para todo $i \geq 0$. Esta contradicción muestra que L no es un LIC.

Ejercicios de la sección 4.12

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

1. $L = \{a^ib^ic^j : j \geq i\}$, sobre $\Sigma = \{a, b, c\}$.
2. $L = \{a^ib^jc^k : 1 \leq i \leq j \leq k\}$, sobre $\Sigma = \{a, b, c\}$.
3. $L = \{a^ib^{2i}a^i : i \geq 1\}$, sobre $\Sigma = \{a, b\}$.
4. $L = \{a^ib^ic^id^i : i \geq 0\}$, sobre $\Sigma = \{a, b, c, d\}$.
5. $L = \{a^ib^jc^id^j : i, j \geq 0\}$, sobre $\Sigma = \{a, b, c, d\}$.
6. $L = \{ww : w \in \{a, b\}^*\}$.
7. $L = \{a^i : i \text{ es un cuadrado perfecto}\}$, sobre $\Sigma = \{a\}$.

4.13. Propiedades de clausura de los LIC

En la sección 3.2 se vio que los lenguajes regulares son cerrados bajo la concatenación, la estrella de Kleene y todas las operaciones booleanas. Los LIC poseen propiedades de clausura mucho más restringidas: son cerrados para las operaciones de unión, concatenación y estrella de Kleene Teorema 4.13.1 pero, en general, no son cerrados para intersección, complementos ni diferencias (Teorema 4.13.2).

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

Demostración: 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 $S \to S_1$ y $S \to S_2$. Las producciones de G_1 y G_2 se mantienen. Concretamente,

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

Esquemáticamente, G tiene el siguiente aspecto:

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

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

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

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

Esquemáticamente, G es la gramática:

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

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

Para generar L_1^* basta definir G como

$$G = (V_1, \Sigma, S_1, P_1 \cup \{S_1 \to S_1 S_1,\ S_1 \to \lambda\})$$

Esquemáticamente, G es la gramática:

$$\left.\begin{array}{ccc} S_1 & \to & S_1 S_1 \mid \lambda \mid \cdots \\ \vdots & & \vdots \end{array}\right\}$$

donde los puntos suspensivos representan las producciones originales de G_1. De esta forma, $L(G) = L_1^*$. □

Ejemplo Utilizar las construcciones del Teorema 4.13.1 para encontrar una GIC que genere el lenguaje $L_1 L_2^*$ donde $L_1 = (ab \cup ba)^*$ y $L_2 = \{a^i b^i : i \geq 0\}$.

Solución: El lenguaje L_1 se puede generar con la gramática

$$S_1 \to S_1 S_1 \mid ab \mid ba \mid \lambda$$

y el lenguaje L_2^* se puede generar con

$$S_2 \to S_2 S_2 \mid a S_2 b \mid \lambda.$$

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

$$\begin{cases} S_3 \to S_1 S_2 \\ S_1 \to S_1 S_1 \mid ab \mid ba \mid \lambda \\ S_2 \to S_2 S_2 \mid a S_2 b \mid \lambda. \end{cases}$$

4.13.2 Teorema. *La colección de los lenguajes independientes del contexto no es cerrada (en general) para las siguientes operaciones:*

(1) Intersección.

(2) Complemento.

(3) Diferencia.

Demostración:

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

$$L_1 = \{a^i b^i c^j : i, j \geq 1\},$$
$$L_2 = \{a^i b^j c^j : i, j \geq 1\}.$$

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

$$G_1: \begin{cases} S \to AB \\ A \to aAb \mid ab \\ B \to cC \mid c \end{cases} \qquad G_2: \begin{cases} S \to AB \\ A \to aA \mid a \\ B \to bBc \mid bc \end{cases}$$

Pero $L_1 \cap L_2 = \{a^i b^i c^i : i \geq 1\}$ no es un LIC, según se mostró, usando el lema de bombeo, en la sección 4.12.

(2) Razonamos por contradicción: si el complemento de todo LIC fuera un LIC se podría concluir que la intersección de dos LIC L_1 y L_2 sería un LIC ya que $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$. Esto estaría en contradicción con la parte (1) del presente teorema.

(3) Razonamos por contradicción: si la diferencia de dos LIC cualesquiera fuera un LIC se podría concluir que el complemento de un LIC L sería también un LIC ya que $\overline{L} = \Sigma^* - L$. Esto estaría en contradicción con la parte (2) del presente teorema. □

El siguiente teorema afirma que los LIC también son cerrados bajo homomorfismos.

4.13.3 Teorema. *Sea $h : \Sigma^* \to \Gamma^*$ un homomorfismo. Si L es un LIC sobre Σ, entonces $h(L)$ es un LIC sobre Γ.*

Demostración: La demostración consiste en transformar una gramática G que genere el lenguaje L en una gramática G' que genere $h(L)$. Para ello basta mantener las mismas variables de G y definir las producciones de G', a partir de las de G, cambiando cada terminal a por $h(a)$. Es fácil ver que una derivación $S \overset{+}{\Longrightarrow} w$ en G, con $w \in \Sigma^*$, se puede transformar en una derivación $S \overset{+}{\Longrightarrow} h(w)$ en G'; esto muestra $h(L) \subseteq L(G')$.

Para establecer la otra contenencia, es decir, $L(G') \subseteq h(L)$, hay que demostrar que si $S \overset{+}{\Longrightarrow}_{G'} z$, con $z \in \Gamma^*$, entonces z es de la forma $z = h(w)$ para algún $w \in L$. Esto puede hacerse considerando el árbol de la derivación $S \overset{+}{\Longrightarrow}_{G'} z$. Dicho árbol se puede transformar en un árbol de una derivación en G, modificando adecuadamente las hojas: las hojas del nuevo árbol forman una cadena $w \in \Sigma^*$ y las hojas del árbol inicial forman la cadena $h(w)$. □

Ejemplo Utilizar homomorfismos para concluir que el lenguaje $L = \{0^i 1^i 2^i 3^i : i \geq 0\}$, sobre el alfabeto $\{0, 1, 2, 3\}$ no es LIC.

Solución: La idea es "convertir" L en el lenguaje $\{a^i b^i c^i : i \geq 0\}$, que no es LIC, según se mostró en el primer ejemplo de la sección 4.12. Razonamos de la siguiente manera: si L fuera un LIC, lo sería también $h(L)$, donde h es el homomorfismo $h : \{0, 1, 2, 3\}^* \to \{a, b, c\}^*$ definido por $h(0) = a$, $h(1) = b$, $h(2) = c$ y $h(3) = \lambda$. Pero

$$h(L) = \{h(0)^i h(1)^i h(2)^i h(3)^i : i \geq 0\} = \{a^i b^i c^i : i \geq 0\}.$$

Por consiguiente, L no es un LIC.

Ejercicios de la sección 4.13

1. Utilizar las construcciones del Teorema 4.13.1 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 $L_3 = aa \cup bb \cup aba$.

   (iii) $L_1 \cup L_2L_3^*$, donde $L_1 = ab^*a$, $L_2 = b^+$ y $L_3 = \{a^iba^i : i \geq 0\}$.

2. (i) Mostrar que los dos lenguajes siguientes, sobre $\Sigma = \{a, b, c, d\}$, son LIC:

$$L_1 = \{a^ib^ic^jd^j : i, j \geq 1\},$$
$$L_2 = \{a^ib^jc^jd^k : i, j, k \geq 1\}.$$

   (ii) Demostrar que $L_1 \cap L_2$ no es un LIC.

3. Utilizar homomorfismos para concluir que los siguientes lenguajes sobre el alfabeto $\{0, 1\}$ no son LIC:

   (i) $L = \{u : |u| \text{ es un número primo}\}$.

   (ii) $L = \{u : |u| \text{ es un cuadrado perfecto}\}$.

4. Demostrar que los LIC son cerrados para la operación de reflexión. Concretamente, demostrar que si L es un LIC, también lo es el lenguaje $L^R = \{w^R : w \in L\}$.

4.14. Algoritmos de decisión para GIC

En esta sección consideraremos problemas de decisión para GIC, similares a los problemas para autómatas presentados en la sección 3.6. Dada una propiedad $\mathcal{P}$, referente a gramáticas independientes del contexto, un problema de decisión para $\mathcal{P}$ consiste en buscar un algoritmo, aplicable a una GIC arbitraria G, que responda SI o NO a la pregunta: ¿satisface G la propiedad $\mathcal{P}$? Los algoritmos vistos en el presente capítulo (para encontrar las variables terminables, las alcanzables, las anulables, etc) son frecuentemente útiles en el diseño de algoritmos de decisión más complejos.

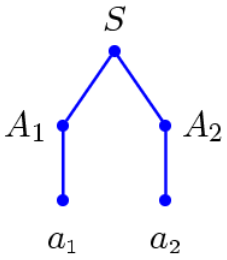
Problema 1 (Problema de la vacuidad). *Dada una gramática $G = (V, \Sigma, S, P)$, ¿es $L(G) \neq \varnothing$?*

Algoritmo de decisión: ejecutar el algoritmo para determinar el conjunto **TERM** de variables terminables. $L(G) \neq \varnothing$ si y sólo si $S \in$ **TERM**.

Problema 2 (Problema de la pertenencia). *Dada una gramática* $G = (V, \Sigma, S, P)$ *y una cadena* $w \in \Sigma^*$*, ¿se tiene* $w \in L(G)$*?*

Para resolver este problema primero convertimos G a la forma FNC, con variable inicial no recursiva, siguiendo el procedimiento de la sección 4.10.

A partir de una GIC G en FNC podemos diseñar un algoritmo bastante ineficiente para decidir si $w \in L(G)$: se encuentran *todas* las posibles derivaciones a izquierda (o los árboles de derivación) que generen cadenas de longitud $n = |w|$. Más específicamente, las cadenas de longitud 1 se pueden derivar únicamente con producciones de la forma $S \to a$. Las cadenas de longitud 2 sólo tienen árboles de derivación de la forma:



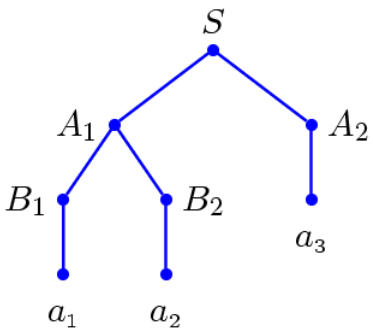
en los que aparecen exactamente 3 variables. Para derivar cadenas de longitud 3 sólo se puede proceder de dos formas:

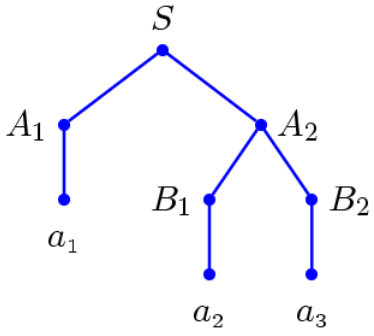
$$S \Rightarrow A_1A_2 \Rightarrow B_1B_1A_2 \overset{3}{\Longrightarrow} a_1a_2a_3,$$

ó

$$S \Rightarrow A_1A_2 \Rightarrow a_1A_2 \Rightarrow a_1B_1B_2 \overset{2}{\Longrightarrow} a_1a_2a_3.$$

Los árboles de estas derivaciones son:





Cada uno de estos árboles tiene exactamente 5 nodos etiquetados con variables. La situación general es la siguiente: un árbol de derivación de una cadena de longitud n tiene exactamente $2n-1$ nodos etiquetados con variables. Puesto que la raíz del árbol es S y S no es recursiva, en un árbol de derivación de una cadena de longitud n hay

exactamente $2n-2$ nodos interiores etiquetados con variables. La demostración general puede hacerse por inducción sobre n y la dejamos como ejercicio para el lector.

Por consiguiente, para determinar si una cadena dada de longitud n es o no generada por G, consideramos todos los posibles árboles de derivación con $2n-2$ variables interiores. Este algoritmo es ineficiente porque si G tiene k variables, hay que chequear no menos de k^{2n-2} árboles (esto sin contar las posibilidades para las hojas). Por consiguiente, el procedimiento tiene complejidad exponencial con respecto al tamaño de la entrada.

Para resolver el problema de la pertenencia hay un algoritmo muy eficiente (su complejidad es polinomial) en el que se usa la llamada *programación dinámica* o *tabulación dinámica*, técnica para llenar tablas progresivamente, re-utilizando información previamente obtenida. El algoritmo que presentaremos se denomina *algoritmo CYK* (nombre que corresponde a las iniciales de los investigadores Cocke, Younger y Kasami). El algoritmo (exhibido en la página siguiente) tiene como entrada una GIC G en FNC y una cadena de n terminales $w = a_1 a_2 \cdots a_n$; se aplica llenando una tabla de n filas (una por cada terminal de la entrada w) y n columnas. X_{ij} es el conjunto de variables de las que se puede derivar la subcadena de w cuyo primer símbolo está en la posición i y cuya longitud es j. O sea,

$$X_{ij} = \text{conjunto de variables } A \text{ tales que } A \overset{+}{\Longrightarrow} a_i a_{i+1} \cdots a_{i+j-1}.$$

Al determinar los conjuntos X_{ij} se obtienen las posibles maneras de derivar subcadenas de w que permitan construir una derivación de la cadena completa w. La tabla se llena por columnas, de arriba hacia abajo; la primera columna ($j = 1$) corresponde a las subcadenas de longitud 1, la segunda columna ($j = 2$) corresponde a las subcadenas de longitud 2, y así sucesivamente. La última columna ($j = n$) corresponde a la única subcadena de longitud n que tiene w, que es la propia cadena w. Se tendrá que $w \in L(G)$ si y sólo si $S \in X_{1n}$.

Ejemplo Vamos a aplicar el algoritmo CYK a la gramática:

$$G: \quad \begin{cases} S \to BA \mid AC \\ A \to CC \mid b \\ B \to AB \mid a \\ C \to BA \mid a \end{cases}$$

y a la cadena $w = bbab$. Se trata de determinar si $w \in L(G)$ o no. La tabla obtenida al hallar los X_{ij}, $1 \le i, j \le 4$, es la siguiente:

Algoritmo CYK

ENTRADA:

Gramática G en FNC y cadena de n terminales $w = a_1 a_2 \cdots a_n$.

INICIALIZAR:

$j = 1$. Para cada i, $1 \leq i \leq n$,

$X_{ij} = X_{i1} :=$ conjunto de variables A tales que $A \to a_i$ es una producción de G.

REPETIR:

$j := j + 1$. Para cada i, $1 \leq i \leq n - j + 1$,

$X_{ij} :=$ conjunto de variables A tales que $A \to BC$ es una producción de G, con $B \in X_{ik}$ y $C \in X_{i+k,j-k}$, considerando todos los k tales que $1 \leq k < j - 1$.

HASTA: $j = n$.

SALIDA: $w \in L(G)$ si y sólo si $S \in X_{1n}$.

		$j=1$	$j=2$	$j=3$	$j=4$
b	$i=1$	$\{A\}$	—	$\{B\}$	$\{S,C\}$
b	$i=2$	$\{A\}$	$\{B,S\}$	$\{S,C\}$	
a	$i=3$	$\{B,C\}$	$\{S,C\}$		
b	$i=1$	$\{A\}$			

A continuación se indica de manera detallada cómo se obtuvo la tabla anterior, columna por columna:

$j = 1$. Se obtiene directamente de las producciones de G.

$j = 2$. Para X_{12} se buscan cuerpos de producciones en $X_{11}X_{21} = \{A\}\{A\} = \{A\}$. Así que $X_{12} = \{\ \}$.
Para X_{22} se buscan cuerpos de producciones en $X_{21}X_{31} = \{A\}\{B,C\} = \{AB, AC\}$. Así que $X_{22} = \{B, S\}$.
Para X_{23} se buscan cuerpos de producciones en $X_{31}X_{41} = \{B,C\}\{A\} = \{BA, CA\}$. Así que $X_{23} = \{S, C\}$.

$j = 3$. Para X_{13} se buscan cuerpos de producciones en $X_{11}X_{22} \cup X_{12}X_{31} = \{A\}\{B, S\} \cup \{\ \} = \{AB, AS\}$. Así que $X_{13} = \{B\}$.
Para X_{23} se buscan cuerpos de producciones en $X_{21}X_{32} \cup X_{22}X_{41} = \{A\}\{S, C\} \cup \{B, S\}\{A\} = \{AS, AC\} \cup \{BA, SA\}$. Así que $X_{23} = \{S, C\}$.

$j = 4$. Para X_{14} se buscan cuerpos de producciones en $X_{11}X_{23} \cup X_{12}X_{32} \cup X_{13}X_{41} = \{A\}\{S, C\} \cup \{\ \} \cup \{B\}\{A\} = \{AS, AC\} \cup \{BA\}$. Así que $X_{14} = \{S, C\}$.

Puesto que la variable S pertenece al conjunto X_{14}, se concluye que la cadena $w = bbab$ es generada por G.

Consideremos ahora la entrada $w = baaba$, de longitud 5. Al hallar los X_{ij}, $1 \leq i, j \leq 5$, se obtiene la tabla siguiente. Como $S \in X_{15}$, se concluye que $w \in L(G)$.

		$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
b	$i=1$	$\{A\}$	$\{B, S\}$	—	—	$\{S, B, C\}$
a	$i=2$	$\{B, C\}$	$\{A\}$	$\{A\}$	$\{S, B, C\}$	
a	$i=3$	$\{B, C\}$	$\{S, C\}$	$\{A\}$		
b	$i=4$	$\{A\}$	$\{B, S\}$			
a	$i=5$	$\{B, C\}$				

Al procesar la entrada $w = aaba$ se obtiene la tabla siguiente. Como S no pertenece al conjunto X_{14}, se deduce que w no es generada por G.

		$j=1$	$j=2$	$j=3$	$j=4$
a	$i=1$	$\{B, C\}$	$\{A\}$	$\{B\}$	—
a	$i=2$	$\{B, C\}$	—	—	
b	$i=3$	$\{B\}$	—		
a	$i=4$	$\{B, C\}$			

Problema 3 (Problema de la infinitud). *Dada una gramática $G = (V, \Sigma, S, P)$, ¿es $L(G)$ infinito?*

El lema de bombeo sirve para establecer un criterio que permite resolver este problema (de manera análoga a lo que sucede en el caso de los lenguajes regulares). El criterio aparece en el siguiente teorema.

4.14.1 Teorema. *Sea $G = (V, \Sigma, S, P)$ una gramática en FNC, con variable inicial no recursiva, tal que $L(G) = L$, y sea $k = |V|$ = número de variables de G. El lenguaje L es infinito si y solo si contiene una cadena z tal que $2^k < |z| \leq 2^{k+1}$.*

Demostración: Si $z \in L$ y $2^k < |z| \leq 2^{k+1}$, entonces por la demostración del lema de bombeo, z se puede descomponer como $z = uvwxy$, donde $|vwx| \leq 2^k$, $v \neq \lambda$. L posee infinitas cadenas: uv^iwx^iy para todo $i \geq 0$.

Recíprocamente, si L es infinito, existe $z \in L$ con $|z| \geq 2^k$. Por la demostración del lema de bombeo, $w = uvwxy$ donde $|vwx| \leq 2^k$; además, $v \neq \lambda$ ó $x \neq \lambda$. Si $|z| \leq 2^{k+1}$, la demostración termina. Si $|z| > 2^{k+1} = 2^k + 2^k$, puesto que $|z| = |uy| + |vwx|$, se tendrá

$$|uwy| \geq |uy| = |z| - |vwx| \geq |z| - 2^k > 2^k.$$

De nuevo, si $|uwy| < 2^{k+1}$, la demostración termina; en caso contrario, se prosigue de esta forma hasta encontrar una cadena en L cuya longitud ℓ satisfaga $2^k < \ell \leq 2^{k+1}$. □

Utilizando el Teorema 4.14.1 podemos ahora presentar un algoritmo de decisión para el problema de la infinitud:

1. Convertir la gramática G dada a una gramática equivalente G' en Forma Normal de Chomsky.

2. Aplicar el algoritmo CYK a G', con cada una de las cadenas $|z|$ cuya longitud ℓ satisfaga $2^k < \ell \leq 2^{k+1}$, siendo k el número de variables de G'. L es infinito si y sólo si alguna de las cadenas examinadas está en $L(G')$.

Obsérvese que este algoritmo tiene de complejidad exponencial ya que el número de cadenas z tales que $2^k < |z| \leq 2^{k+1}$ es m^{2^k}, donde m es el número de terminales en la gramática dada.

✎ Hay muchos problemas referentes a gramáticas que son *indecidibles*; para estos problemas no existen algoritmos de decisión. Entre ellos mencionamos:

1. Dada una gramática G, ¿es G ambigua?

2. Dada una gramática G, ¿genera G *todas* las cadenas de terminales?, es decir, ¿$L(G) = \Sigma^*$?

3. Dadas dos gramáticas G_1 y G_2, ¿generan G_1 y G_2 el mismo lenguaje?, es decir, ¿$L(G_1) = L(G_2)$?

Ejercicios de la sección 4.14

1. Sea $G = (V, \Sigma, S, P)$ una gramática dada. Encontrar algoritmos para los siguientes problemas de decisión:

   (i) ¿Hay en $L(G)$ alguna cadena de longitud 1250?

   (ii) ¿Hay en $L(G)$ alguna cadena de longitud mayor que 1250?

   (iii) ¿Hay en $L(G)$ por lo menos 1250 cadenas?

2. Encontrar un algoritmo para el siguiente problema de decisión: dada una gramática $G = (V, \Sigma, S, P)$ y una variable $A \in V$, ¿es A recursiva?, es decir, ¿existe una derivación de la forma $A \overset{+}{\Longrightarrow} uAv$, con $u, v \in (V \cup \Sigma)^*$?

3. Encontrar un algoritmo para el siguiente problema de decisión: dado un lenguaje finito L y una GIC G, ¿se tiene $L \subseteq L(G)$?

4. Sea G la gramática

$$G: \quad \begin{cases} S \to BA \mid AB \\ A \to CA \mid a \\ B \to BB \mid b \\ C \to BA \mid c \end{cases}$$

   Ejecutar el algoritmo CYK para determinar si las siguientes cadenas w son o no generadas por G:

   (i) $w = bca$. (iii) $w = cabb$.

   (ii) $w = acbc$. (iv) $w = bbbaa$.