Autómatas y Lenguajes Formales
Nota 07. Gramáticas libres de contexto*

Noé Salomón Hernández S.

1. Gramáticas libres de contexto

Las gramáticas son una notación para la definición recursiva de lenguajes (lenguajes naturales o de programación).

Los lenguajes libres de contexto son importantes en la definición de lenguajes de programación, en la simplificación de la traducción de lenguajes de programación y en otras aplicaciones del procesamiento de cadenas. Las gramáticas libres del contexto son útiles para describir expresiones aritméticas que tengan una anidación arbitraria de paréntesis balanceados y estructuras de bloque en los lenguajes de programación.

La motivación original para el desarrollo de las gramáticas libres de contexto fue la necesidad de describir los lenguajes naturales. Podemos escribir reglas como las siguientes:

$$\begin{aligned}
\langle \text{oración} \rangle &\longrightarrow \langle \text{frase sustantiva} \rangle\ \langle \text{frase verbal} \rangle \\
\langle \text{frase sustantiva} \rangle &\longrightarrow \langle \text{frase sustantiva} \rangle\ \langle \text{adjetivo} \rangle \\
\langle \text{frase sustantiva} \rangle &\longrightarrow \langle \text{artículo} \rangle\ \langle \text{sustantivo} \rangle \\
\langle \text{artículo} \rangle &\longrightarrow \textit{la} \\
\langle \text{sustantivo} \rangle &\longrightarrow \textit{casa} \\
\langle \text{adjetivo} \rangle &\longrightarrow \textit{grande}
\end{aligned}$$

en donde las categorías sintácticas[1] se escriben entre paréntesis angulares y las terminales se denotan con palabras en itálicas sin paréntesis, como *casa* y *grande*.

Las gramáticas libres de contexto no se consideran como adecuadas para la descripción de los lenguajes naturales. Sin embargo, éstas juegan un papel importante en lingüística computacional.

Mientras los lingüistas estudiaban las gramáticas libres de contexto, los científicos de la computación comenzaron a describir lenguajes de programación mediante una notación conocida como *forma de Backus-Naur* (BNF).

Ejemplo 1.1 [**Palíndromos**] Las cadenas palíndromos son aquellas que se leen igual al derecho y al revés. Así el lenguaje de estas palabras es $L_{PAL} = \{w \in \{0,1\}^* \mid w^{\mathcal{R}} = w\}$

*Esta nota se basa en los libros: M. Sipser *Introduction to the Theory of Computation* PWS Publishing Company, 1997, y en J. E. Hopcroft, R. Motwani, J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. También en las notas del prof. Rajeev Motwani, aquí las encuentran.

[1] *Caterogías sintácticas* es un sinónimo de *variables*. Se prefiere el primero cuando se habla de lenguajes naturales.

La gramática que lo define tiene por reglas:

$$\begin{aligned} P &\longrightarrow \varepsilon \\ P &\longrightarrow 0 \\ P &\longrightarrow 1 \end{aligned}$$

$$\begin{aligned} P &\longrightarrow 0P0 \\ P &\longrightarrow 1P1 \end{aligned}$$

Las primeras tres reglas son la base, ε, 0 y 1 son palíndromos. Las últimas dos reglas son inductivas, si P es un palíndromo, entonces los son también $0P0$ y $1P1$.

De manera implícita todos los palíndromos pueden ser obtenidos al aplicar las reglas anteriores recursivamente, y únicamente los palíndromos pueden ser obtenidos a través de dicha aplicación.

Ejemplo 1.2 La gramática para el lenguaje $L_{Eq} = \{w \in \{0,1\}^* \mid w$ tiene mismo número de $0s$ y de $1s\}$ es

$$\begin{aligned} S &\longrightarrow \varepsilon \\ S &\longrightarrow 0A \\ S &\longrightarrow 1B \end{aligned}$$

$$\begin{aligned} A &\longrightarrow 1S \\ A &\longrightarrow 0AA \end{aligned}$$

$$\begin{aligned} B &\longrightarrow 0S \\ B &\longrightarrow 1BB \end{aligned}$$

Observe A genera las cadenas con un 1 más que $0s$, y B genera todas las cadenas con un 0 más que $1s$, así S genera las cadenas con mismo número de $0s$ y de $1s$, el cual es el lenguaje buscado. También note que la gramática anterior está haciendo uso de una recursión entrelazada, en el sentido de que para definir el lenguaje de S se emplean los lenguajes de A y B. La definición de los lenguajes de A y B, a su vez emplean el lenguaje de S.

Definición 1.3 Formalmente, una gramática G es una tupla $G = (V, T, P, S)$ donde

- T es un conjunto finito de símbolos terminales. En esencia $T = \Sigma$.
- V es un conjunto finito de símbolos no terminales o variables las cuales representan conjuntos de cadenas que están siendo definidos recursivamente, es decir, cada variable genera un lenguaje.
- S es el símbolo inicial, $S \in V$. Es una variable especial que genera el lenguaje deseado.
- P es el conjunto finito de reglas de producción, o reglas gramaticales, que contiene a las definiciones recursivas.

Por ejemplo, para la gramática del lenguaje L_{Eq} tenemos que $G_{Eq} = (V, T, P, S)$ donde

- $T = \{0, 1\}$,
- $V = \{S, A, B\}$,
- $P = \left\{ \begin{array}{lcl} S & \longrightarrow & \varepsilon \,|\, 0A \,|\, 1B \\ A & \longrightarrow & 1S \,|\, 0AA \\ B & \longrightarrow & 0S \,|\, 1BB \end{array} \right\}$

Arriba introducimos la notación que abrevia el conjunto de reglas $A \longrightarrow \alpha_1$, $A \longrightarrow \alpha_2$, ..., $A \longrightarrow \alpha_k$ como $A \longrightarrow \alpha_1 \,|\, \alpha_2 \,|\, \ldots \,|\, \alpha_k$.

Definición 1.4 Las **gramáticas libres de contexto (GLC)** permiten solamente las reglas de producción de la forma

$$\begin{array}{lcl} A & \longrightarrow & \varepsilon \\ A & \longrightarrow & \alpha_1\alpha_2 \ldots \alpha_k, \quad k \geq 1 \end{array}$$

donde

- $A \in V$,
- $\alpha_i \in V \cup T$, para toda i, $1 \leq i \leq k$.

En una gramática que no es libre del contexto las reglas pueden especificar el contexto bajo el cual se puede realizar la sustitución que indica la regla en cuestión. Por ejemplo, $0A1 \longrightarrow 0\alpha_1\alpha_2 \ldots \alpha_n 1$. Por lo que tal regla no puede aplicarse en otro contexto en el que la variable A no tenga a la izquierda a 0 y a la derecha a 1.

La descripción de lenguajes de programación en la *forma de Backus-Naur* se basa en las gramáticas libres de contexto con algunos cambios en la notación, escribiendo `::=` en lugar de $\longrightarrow$ en las reglas de producción y, en algunas ocasiones, los nombres de variables figuran entre paréntesis angulares, $\langle$ y $\rangle$.

Definición 1.5 (**Derivación**) Sea G una GLC. Para toda $\alpha, \beta, \gamma \in (V \cup T)^*$, y para toda $A \in V$ decimos que $\alpha A \beta$ se **deriva** en un paso, o se **deriva** directamente, en $\alpha\gamma\beta$ si la regla $A \longrightarrow \gamma$ está en G, y escribimos

$$\alpha A \beta \Rightarrow \alpha\gamma\beta.$$

Definición 1.6 (**Secuencia de derivación**) Supongamos que $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \alpha_3 \Rightarrow \ldots \Rightarrow \alpha_n$. De manera concisa decimos que $\alpha_1 \Rightarrow^* \alpha_n$, lo que significa que α_1 deriva en α_n en 0 o más pasos de derivación. Observamos que $\Rightarrow^*$ es la cerradura reflexiva y transitiva de $\Rightarrow$, es decir, si $\alpha \Rightarrow \alpha'$, entonces $\alpha \Rightarrow^* \alpha'$; además $\Rightarrow^*$ cumple

Reflexividad $\alpha \Rightarrow^* \alpha$,

Transitividad $\alpha \Rightarrow^* \beta$, $\beta \Rightarrow^* \gamma$, implica $\alpha \Rightarrow^* \gamma$.

Ejemplo 1.7

- Tomando la gramática del lenguaje L_{PAL}, se tiene la siguiente derivación para la cadena 00100,

$$P \Rightarrow 0P0 \Rightarrow 00P00 \Rightarrow 00100.$$

- Tomando la gramática G_{Eq}, la derivación para 001110 es

$$\begin{aligned} S &\Rightarrow 0A \Rightarrow 00AA \Rightarrow 001SA \Rightarrow 001S1S \\ &\Rightarrow 0011S \Rightarrow 00111B \Rightarrow 001110S \Rightarrow 001110. \end{aligned}$$

  Observemos que como $001SA \Rightarrow 001S1S \Rightarrow 0011S$, podemos escribir $001SA \Rightarrow^* 0011S$. También podemos escribir $S \Rightarrow^* 001110$.

Definición 1.8 El lenguaje de una GLC $G = (V, T, P, S)$ se define como $\mathcal{L}(G) = \{w \in T^* \mid S \Rightarrow^* w\}$. Un **lenguaje libre de contexto (LLC)** L es aquel para el cual existe una GLC G tal que $L = \mathcal{L}(G)$.

Definición 1.9 Definimos las gramáticas G_1 y G_2 como **equivalentes** si $\mathcal{L}(G_1) = \mathcal{L}(G_2)$.

2. Diseñando gramáticas libres de contexto

Como en el diseño de autómatas finitos, el diseño de gramáticas libres de contexto requiere creatividad. Las gramáticas libres de contexto son un poco más complejas que los autómatas finitos porque estamos más acostumbrados a programar una máquina para una tarea específica que describir lenguajes con gramáticas. Las siguientes técnicas son útiles, de manera individual o en conjunto, cuando tenemos la tarea de construir una GLC.

En primer lugar, muchas GLC se forman como la unión de GLC más sencillas. Si debemos construir una GLC para un LLC que puede ser dividido en piezas más sencillas, construyamos gramáticas individuales para cada pieza y después unámoslas. Estas gramáticas pueden ser mezcladas fácilmente para formar la que nos interesa al combinar sus reglas y al agregar la regla nueva $S \longrightarrow S_1 \mid S_2 \mid \ldots \mid S_k$, donde las variables S_i son las variables iniciales para las gramáticas individuales. Resolver muchos problemas sencillos es a menudo más fácil que resolver el problema complicado. Por ejemplo, al obtener la gramática para el lenguaje $\{0^n1^n \mid n \geq 0\} \cup \{1^n0^n \mid n \geq 0\}$, primero construimos la gramática

$$S_1 \longrightarrow 0S_11 \mid \varepsilon$$

para el lenguaje $\{0^n1^n \mid n \geq 0\}$, y la gramática

$$S_2 \longrightarrow 1S_20 \mid \varepsilon$$

para el lenguaje $\{1^n0^n \mid n \geq 0\}$. Entonces agregamos la regla $S \longrightarrow S_1 \mid S_2$ para llegar a la gramática

$$\begin{aligned} S &\longrightarrow S_1 \mid S_2 \\ S_1 &\longrightarrow 0S_11 \mid \varepsilon \\ S_2 &\longrightarrow 1S_20 \mid \varepsilon \end{aligned}$$

En segundo lugar, algunos lenguajes libres del contexto contienen cadenas compuestas por dos subcadenas que pueden ser ligadas como lo haría un autómata para dicho lenguaje que necesitara recordar una cantidad ilimitada de información acerca de una de esas subcadenas para emparejarla con la otra subcadena. Esta situación ocurre con el lenguaje $\{0^n1^n \mid n \geq 0\}$ porque un autómata necesitaría recordar el número de $0s$ para verificar que es igual al número de $1s$. Podemos construir

una GLC para esta situación al usar una regla de la forma $R \longrightarrow uRv$, la cual genera cadenas donde la porción que contenga a la u corresponde con la porción que contiene a la v.

Finalmente, en lenguajes más complejos, las cadenas pueden contener ciertas estructuras que pueden aparecer recursivamente como parte de la misma u otras estructuras. Esto sucede en la gramática G_{Eq}, ya que en la regla $A \longrightarrow 1S$ la estructura S que genera cadenas con mismo número de $0s$ y $1s$ aparece del lado derecho. Así A produce cadenas con un 1 más que $0s$. Para lograr este efecto, tenemos que poner la variable generando la estructura deseada en el lugar de las reglas que corresponda a donde esa estructura se necesita de manera recursiva. En nuestro ejemplo la S se necesita para definir las estructuras correspondientes a A y a B.

3. Terminología, derivaciones canónicas y árboles de derivación

Definición 3.1 Una **oración** es cualquier $w \in T^*$ tal que $S \Rightarrow^* w$, i.e., $w \in L$.

Definición 3.2 Una **forma oracional** es cualquier $\alpha \in (V \cup T)^*$ tal que $S \Rightarrow^* \alpha$.

En general usaremos la siguiente notación:

- Terminales: $a, b, c, \ldots$
- Variables: $A, B, C, \ldots$
- Cadenas de terminales: $\ldots, u, v, w, x, y, z$
- Elementos en $V \cup T$: $\ldots, X, Y, Z$
- Forma oracional: $\alpha, \beta, \gamma, \ldots$

Una de las características de las GLC es el no-determinismo al decidir qué regla aplicar durante una derivación. Digamos que hemos alcanzado la forma oracional $\alpha A \beta$ y se tienen las reglas $A \longrightarrow \gamma_1$,$A \longrightarrow \gamma_2$,...,$A \longrightarrow \gamma_k$, así que al considerar A en el siguiente paso de la derivación se tienen k posibles opciones. ¿Cuál de ellas aplicar? Incluso, en $\alpha A \beta$ pueden haber muchas variables que sustituir, ¿qué variable tomamos? Este no-determinismo le brinda a las gramáticas libres de contexto expresividad y flexibilidad pero también nos sirve para confundirnos, y confundir a los *parsers*[2].

Definición 3.3 Una **derivación más a la izquierda** siempre sustituye, en una forma oracional obtenida durante una derivación, la variable más a la izquierda por una regla de producción. Se denota como $\underset{lm}{\Rightarrow}$. Similarmente podemos definir una derivación más a la derecha, $\underset{rm}{\Rightarrow}$. Ahora podemos hablar de secuencias de derivaciones canónicas $\underset{lm}{\Rightarrow}^*$ o $\underset{rm}{\Rightarrow}^*$.

Definición 3.4 Los árboles de derivación, también conocidos como árboles de análisis sintáctico en compiladores, representan gráficamente una derivación. En general, para representar $A \Rightarrow^* \alpha$ realizamos lo siguiente:

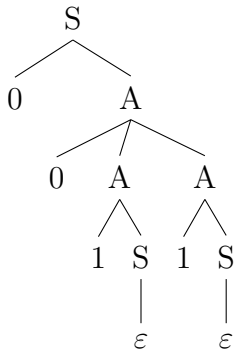
[2]Un *parser* es un analizador sintáctico.

- La raíz se etiqueta con A.
- El *producto* del árbol de derivación, α, es la secuencia de etiquetas de hojas leídas de izquierda a derecha.
- Los nodos internos son etiquetados con variables y sus hijos indican la regla de producción que fue aplicada sobre dicha variable.

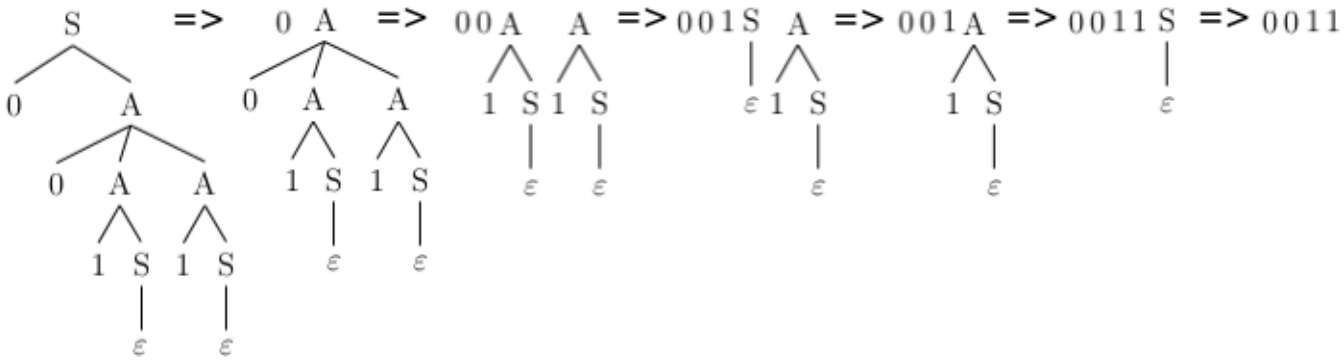
Definición 3.5 Un **árbol-A** es un árbol de derivación cuya raíz es la variable A. Si la raíz es S simplemente lo llamamos árbol de derivación.

Teorema 3.6 *α es el producto de un árbol-A si y solo si $A \Rightarrow^* \alpha$.*

Una derivación más a la izquierda se obtiene al visitar por profundidad el árbol de análisis sintáctico, tomando siempre los subárboles izquierdos antes que los derechos. Por ejemplo, considere la gramática G_{Eq} y el siguiente árbol de análisis sintáctico resultante para la cadena 0011,



del árbol anterior obtenemos la derivación más a la izquierda como se muestra a continuación:



Similarmente, una derivación más a la derecha se logra con una visita a profundidad tomando los subárboles derechos primero.

Teorema 3.7 *Los siguientes son enunciados equivalentes dada la GLC $G = (V, T, P, S)$ y la cadena $w \in T^*$:*

- $w \in \mathcal{L}(G)$

- $S \underset{lm}{\Rightarrow}^* w$
- $S \underset{rm}{\Rightarrow}^* w$
- *existe un árbol de derivación (con raíz S) que tiene como producto a w.*

Ejemplo 3.8 Sea G' la GLC

$$\begin{array}{rcl} S & \longrightarrow & bSa \,|\, aY \,|\, Yb \\ Y & \longrightarrow & bY \,|\, aY \,|\, \varepsilon \end{array}$$

Dé una descripción sencilla de $L(G')$ y muestre la derivación $S \Rightarrow bbbaa$.

Respuesta: el lenguaje de la gramática G' es el conjunto de cadenas de la forma $b^n x a^n$, $|n| \geq 0$, tales que $x \in \{a, b\}^+$ y no es el caso que x comience con b y termine con a.

La derivación buscada se muestra enseguida:

$$S \Rightarrow bSa \Rightarrow bbSaa \Rightarrow bbYbaa \Rightarrow bbbaa.$$

Ejemplo 3.9 Exhiba una gramática libre de contexto que genere el lenguaje descrito a continuación.

$$\{w \in \{0,1\}^* \,|\, w \text{ comienza y termina con el mismo símbolo}\}$$

Respuesta:

$$\begin{array}{rcl} S & \longrightarrow & 0T0 \,|\, 1T1 \,|\, 0 \,|\, 1 \\ T & \longrightarrow & 0T \,|\, 1T \,|\, \varepsilon \end{array}$$

Ejemplo 3.10 El lenguaje de paréntesis balanceados puede ser generado por la GLC:

$$S \longrightarrow SS \,|\, (S) \,|\, \varepsilon.$$

Muestre una derivación más a la izquierda para la cadena ()(())().

Respuesta:

$$S \underset{lm}{\Rightarrow} SS \underset{lm}{\Rightarrow} (S)S \underset{lm}{\Rightarrow} ()S \underset{lm}{\Rightarrow} ()SS \underset{lm}{\Rightarrow} ()(S)S \underset{lm}{\Rightarrow} ()((S))S \underset{lm}{\Rightarrow} ()(())S \underset{lm}{\Rightarrow} ()(())(S) \underset{lm}{\Rightarrow} ()(())().$$

Ejercicios

1. Conteste los siguientes incisos respecto a la gramática libre de contexto G que se muestra a continuación:

$$\begin{array}{rcl} R & \longrightarrow & XRX \,|\, S \\ S & \longrightarrow & aTb \,|\, bTa \\ T & \longrightarrow & XTX \,|\, X \,|\, \varepsilon \\ X & \longrightarrow & a \,|\, b \end{array}$$

a. ¿Cuántas variables tiene G?

b. ¿Cuántos terminales tiene G?

c. ¿Cuál es el símbolo inicial de G?

d. Dé tres cadenas en $L(G)$.

e. Dé tres cadenas *no* en $L(G)$.

f. V ó F: $T \Rightarrow aba$.

g. V ó F: $T \Rightarrow^* aba$.

h. V ó F: $T \Rightarrow T$.

i. V ó F: $T \Rightarrow^* T$.

j. V ó F: $XXX \Rightarrow^* aba$.

k. V ó F: $X \Rightarrow^* aba$.

l. V ó F: $T \Rightarrow^* XX$.

m. V ó F: $T \Rightarrow^* XXX$.

n. V ó F: $S \Rightarrow^* \varepsilon$.

ñ. Describa en español el lenguaje $L(G)$.

2. Dé gramáticas libres de contexto que generen los siguientes lenguajes. En todos los casos el alfabeto es $\{0, 1\}$.

   a. $\{w \mid w$ contiene al menos tres 1s$\}$

   b. $\{w \mid w$ tiene longitud par$\}$

3. Sea $G_1 = (V, T, P, S)$ donde $V = \{S, T, U\}$, $T = \{0, \#\}$, y P el conjunto de producciones:

$$\begin{array}{lcl} S & \longrightarrow & TT \mid U \\ T & \longrightarrow & 0T \mid T0 \mid \# \\ U & \longrightarrow & 0U00 \mid \# \end{array}$$

   Describa en español el lenguaje $L(G_1)$

4. En cada caso diga el lenguaje (subconjunto de $\{a, b\}$) que es generado por la gramática libre de contexto con las producciones indicadas:

   a. $S \to aS \mid bS \mid \varepsilon$

   b. $S \to SS \mid bS \mid a$

   c. $S \to SaS \mid b$

   d. $S \to SaS \mid b \mid \varepsilon$

   e. $S \to TT \qquad T \to aT \mid Ta \mid b$

   f. $S \to aSa \mid bSb \mid aAb \mid bAa$
   $A \to aAa \mid bAb \mid a \mid b \mid \varepsilon$

   g. $S \to aT \mid bT \mid \varepsilon \qquad T \to aS \mid bS$

   h. $S \to aT \mid bT \qquad T \to aS \mid bS \mid \varepsilon$

5. En cada inciso, encuentre un GLC que genere el lenguaje dado.

   a. El conjunto de cadenas en $\{a, b\}^*$ de longitud impar con símbolo medio a.

   b. El conjunto de cadenas en $\{a, b\}^*$ de longitud par con los dos símbolos medios iguales.

   c. El conjunto de cadenas en $\{a, b\}^*$ de longitud impar con los símbolos inicial, medio y final iguales.

4. Ambigüedad

Podemos usar siempre una derivación más a la izquierda para especificar un modo canónico de derivación de cualquier $w \in \mathcal{L}(G)$, o dado un árbol sintáctico lo podemos convertir a una única derivación, simplificando así la tarea de los analizadores sintácticos.

Pero, ¿qué sucede si para alguna $w \in \mathcal{L}(G)$ se tienen dos árboles sintácticos distintos y, por lo tanto, dos derivaciones más a la izquierda distintas? Esto representa no sólo un problema sintáctico, esto también indica dos interpretaciones semánticas distintas.

Definición 4.1 Una GLC G es **ambigua** si para alguna $w \in \mathcal{L}(G)$ existen dos árboles sintácticos distintos.

En compiladores los árboles sintácticos determinan la interpretación, por lo que no podemos permitir ambigüedad. Así que debemos diseñar gramáticas que no sean ambiguas. Una estrategia para evitar la ambigüedad es que la gramática se formule estableciendo la precedencia de los operadores. Aunque no siempre es posible obtener gramáticas no ambiguas para lenguajes libres de contexto, más aun, no existe algoritmo para eliminar la ambigüedad dada una gramática. Por lo que se tiene la siguiente definición.

Definición 4.2 Un lenguaje libre de contexto es **inherentemente ambiguo** si toda gramática que lo genera es ambigua.

Ejemplo 4.3 El lenguaje $L = \{a^n b^n c^m d^m \,|\, n, m \geq 1\} \cup \{a^n b^m c^m d^n \,|\, n, m \geq 1\}$ es inherentemente ambiguo ya que para las cadenas de la forma $a^k b^k c^k d^k$ no es posible decir si vienen del primer o del segundo tipo de cadenas en L, y cualquier GLC que genere a L debe permitir ambas posibilidades.

Ejemplo 4.4 En el lenguaje de programación C, un enunciado `if` puede ser definido por las siguientes reglas de producción:

$$\begin{array}{rcl} S & \longrightarrow & \texttt{if}\ (E)\ S\ \mid \\ & & \texttt{if}\ (E)\ S\ \texttt{else}\ S\ \mid \\ & & OS. \end{array}$$

En nuestra notación, E representa un expresión, S un enunciado, y OS otros enunciados. Un enunciado en C que ilustra la ambigüedad de estas reglas es:

$$w = \texttt{if}\ (e_1)\ \texttt{if}\ (e_2)\ \texttt{f}();\ \texttt{else}\ \texttt{g}();$$

El problema es que aunque en C el `else` debe asociarse con el segundo `if`, como en

$$\texttt{if}\ (e_1)\ \{\ \texttt{if}\ (e_2)\ \texttt{f}();\ \texttt{else}\ \texttt{g}();\ \}$$

no hay nada en estas reglas que excluya la otra interpretación:

$$\texttt{if}\ (e_1)\ \{\ \texttt{if}\ (e_2)\ \texttt{f}();\ \}\ \texttt{else}\ \texttt{g}();$$

Se pueden obtener dos árboles de derivación, uno para cada interpretación, del enunciado w.

Hay gramáticas equivalentes que generan las mismas cadenas que la original y que no son ambiguas. Una posibilidad es:

$$\begin{array}{rcl} S & \longrightarrow & S_1 \,|\, S_2 \\ S_1 & \longrightarrow & \texttt{if}\ (E)\ S_1\ \texttt{else}\ S_1\ \mid OS \\ S_2 & \longrightarrow & \texttt{if}\ (E)\ S\ \mid \\ & & \texttt{if}\ (E)\ S_1\ \texttt{else}\ S_2. \end{array}$$

La variable S_1 representa un enunciado en donde cada `if` se asocia con su correspondiente `else`, y cada enunciado derivado de S_2 contiene al menos un `if` sin `else` asociado. La única variable apareciendo antes que un `else` en estas reglas es S_1; de modo que este `else` no puede asociarse con ninguno de los `if`s en el enunciado derivado de S_1, así se asocia con el `if` que surgió al mismo tiempo que él.

Ejemplo 4.5 Considere la gramática para expresiones aritméticas siguiente, donde a representa identificadores y contantes numéricas:

$$S \longrightarrow a \,|\, S + S \,|\, S * S \,|\, (S).$$

La cadena $a+a*a$ tiene dos derivaciones más a la izquierda, una correspondiendo a la interpretación $(a + a) * a$ y la otra a $a + (a * a)$, esta última interpretación es la correcta pues la multiplicación tiene precedencia sobre la suma. La gramática anterior no define la precedencia de los operadores ya que asigna a ambos la precedencia más alta, además no permite que las operaciones con la misma precedencia se lleven a cabo de izquierda a derecha. Una gramática equivalente que toma en cuenta estos aspectos es:

$$\begin{array}{lcl} S & \longrightarrow & S + T \,|\, T \\ T & \longrightarrow & T * F \,|\, F \\ F & \longrightarrow & a \,|\, (S). \end{array}$$

Ejemplo 4.6 Como se vio en el Ejemplo 3.10, el lenguaje de paréntesis balanceados es generado por la gramática:

$$S \longrightarrow SS \,|\, (S) \,|\, \varepsilon$$

la cual es ambigua. Una gramática no ambigua para el mismo lenguaje es:

$$S \longrightarrow (S)S \,|\, \varepsilon$$