Introducción a los lenguajes

Para iniciar el estudio de la Teoría de la Computación, se debe comenzar por los lenguajes ya que son éstos los que permiten describir y formular problemas. Más aún, tanto más precisa será la formulación de problemas cuanto más sistemática y rigurosa sea la definición de los lenguajes con que se los describe.

El concepto de lenguaje es muy amplio, abarcando mucho más de lo que intuitivamente se reconoce como tal y trascendiendo la comunicación oral y escrita de las personas. Todas las comunidades de seres vivos adoptan alguna forma de lenguaje para intercomunicarse, aunque el interés en este capítulo estará centrado, como se verá más adelante, en un tratamiento muy particular de este tema. Eso sí, no debe confundirse al propio lenguaje con los códigos y medios usados en su representación. Un mismo lenguaje, por ejemplo el español, admite una representación oral, escrita, de código Morse y otras. Sin embargo, debe reconocerse que algunos lenguajes son relativamente independientes de los medios empleados y otros, por el contrario, están fuertemente ligados a ellos (lenguaje de las manos, lenguaje naval de banderas, etcétera).

Para comenzar, puede definirse el lenguaje como el conjunto de signos (o palabras) y de reglas que los organizan. El signo es, a su vez, un concepto abstracto que permite hacer referencia a objetos. Como ejemplos: clavel es un signo de tipo de flor, el semáforo verde es un signo de paso libre, etc. El proceso por el cual se toma un objeto como signo de algo se denomina interpretación. La interpretación determina que el signo actúe como tal, ya sea porque permite relacionar al signo con su objeto o bien porque el signo se asocia a una idea o pensamiento. Fundamentalmente, interesa destacar que sin interpretación no hay signo. Si bien pueden ser tomados como sinónimos, un ***símbolo*** es un signo creado convencionalmente.

Ampliando estos conceptos, se llama ***denotado*** al objeto al que se refiere el signo y ***designado*** a las características o propiedades a las que se remite el signo. Por ejemplo: el signo *vaso* designa a un objeto con forma de recipiente destinado a contener líquidos y denota a todos los objetos a los que es aplicable el signo, es decir a todos los vasos. El designado de un signo se determina por medio de la definición. Por otra parte, el hecho de que un signo tenga más de un designado se denomina ambigüedad.

Se definió al lenguaje como el conjunto de signos y de reglas que organizan esos signos. Es decir que no basta con reunir algunos signos para obtener un lenguaje, ya que hace falta además establecer las relaciones entre ellos. Estas reglas destinadas a establecer las relaciones entre los signos de un lenguaje son de tres tipos: sintácticas, semánticas y pragmáticas. Las reglas sintácticas establecen el orden y relación entre los signos. Las reglas semánticas por su parte relacionan los signos y sus significados. Por último, las reglas pragmáticas vinculan los signos con sus usuarios. La existencia y respeto por estas reglas es la que hace posible la comunicación.

Al establecer el orden y la relación entre los signos, las reglas sintácticas permiten combinar las palabras para formar frases correctas. De la forma que toman las palabras se ocupa la morfología y de su expresión oral se ocupa la fonética. La morfología, la sintaxis y la fonética componen la gramática de los lenguajes.

El lenguaje, medio esencial de comunicación entre miembros de una misma especie, adquiere una nueva proyección al posibilitar el diálogo hombre–máquina a partir del desarrollo del ordenador. Es así que la Teoría de los Lenguajes y las Gramáticas se ve revolucionada cuando, en la década del cincuenta, el lingüista norteamericano Avram Noam Chomsky presenta su “Teoría de las Gramáticas Transformacionales”, que estableció las bases de la Lingüística Matemática y que proporcionó una herramienta que no solo era aplicable a los lenguajes tradicionales (naturales), sino que además iba a ser inmediatamente aprovechada en la formalización de los entonces incipientes lenguajes de computación.

Como se señaló anteriormente, los lenguajes pueden dividirse en *naturales* y *formales*. Los lenguajes de programación de computadores pertenecen a la segunda clase, son desarrollados a partir de gramáticas propuestas y no admiten forma alguna de excepción o desviación en su uso. Por ello, los lenguajes formales son los apropiados para la interpretación inequívoca de mensajes, que es requisito primordial en el desarrollo de compiladores y, en general, de todas las aplicaciones informáticas que incluyen alguna forma de comunicación.

Inclusive las técnicas y herramientas desarrolladas para el estudio y manipulación de los lenguajes formales constituyen el abordaje primario para el tratamiento computarizado de los lenguajes naturales, tanto en la *traducción automática*, en la *interpretación de significados*, y en el *diálogo con sistemas expertos*, entre muchas otras aplicaciones. Como ya se indicó, hoy se reconoce como necesario *entender* el significado de una frase antes de poder traducirla correctamente o transformarla en una orden, por lo cual su tratamiento debe ser tanto sintáctico como semántico.

El objetivo de este capítulo es el tratamiento de los lenguajes y gramáticas formales desde la óptica de la propuesta de Chomsky, con el fin de sentar bases que serán más adelante utilizadas en la Teoría de Autómatas. Es necesario señalar que la Teoría de los Lenguajes Formales es muy amplia y es una disciplina independiente. Por lo tanto, la presentación que se hace aquí debe reconocerse como introductoria ya que solo avanzará en su formalismo, en la medida de lo necesario para los fines enunciados.

Lingüística matemática

Con *lingüística matemática*, suele hacerse referencia al estudio de los lenguajes utilizando conceptos, técnicas y herramientas cercanos a la Matemática Discreta y al Álgebra.

Se asumirá que los conceptos matemáticos básicos tales como lógica, conjuntos, relaciones, funciones, inducción matemática, definiciones recursivas y otros, son ya conocidos, por lo que aquí no nos ocuparemos de su estudio, sino de su aplicación a nuestro campo.

Símbolos, alfabetos y palabras

Las computadoras actuales y todos los dispositivos digitales solo almacenan dos señales diferentes: cero-uno, norte-sur, monte-valle, prendido-apagado, si-no.

Con solo dos señales y una adecuada codificación (UNICODE, ASCII, EBCDIC son algunos códigos usuales hoy en computación), puede representarse cualquier conjunto de símbolos, tales como:

- letras de algún alfabeto humano (o aun de varios de ellos como sucede con UNICODE),
- caracteres especiales de control de dispositivos electrónicos (salto de página, escape, nueva línea, alarma, limpieza de pantalla, apagado),
- dígitos y símbolos especiales (0, 1, 2, …, 9, arroba, punto y coma, pesos, euro, más, menos, por, barra),

o cualquier otro (piezas de ajedrez, instrucciones de un lenguaje de programación, acciones posibles de un robot, operadores y variables de un álgebra dada, símbolos viales, etcétera) utilizado para modelar un problema bajo estudio o un sistema y su comportamiento.

Sin importar lo que estén denotando los símbolos que se utilicen, al menos se debería tener uno y, en caso de ser más de uno, deberían ser finitos en número, ya que no podríamos distinguir (¡y ni que hablar recordar!) infinitos símbolos.

Alfabeto

Se denomina ***alfabeto*** *a cualquier conjunto finito y no vacío de símbolos*. Se usarán en adelante para denotarlos, las letras griegas mayúsculas Σ, Γ, Λ, Ω, …, con o sin subíndices.

Ejemplo 2.1

Algunos ejemplos de alfabetos son:

$\Sigma_1 = \{a\}$ alfabeto unitario compuesto de un único símbolo **a**.

$\Sigma_2 = \{0, 1\}$ alfabeto binario con dos símbolos distintos **0** (cero) y **1** (uno).

$\Sigma_3 = \{a, b, c, \ldots, z\}$ alfabeto de letras minúsculas españolas.

$\Gamma = \{\text{torre, caballo, alfil, rey, reina, peón}\}$
alfabeto de símbolos para el juego de ajedrez.

Ω = { int, char, float, double, struct, union, long, short, unsigned, auto, extern, register, typedef, static, goto, return, if, else, sizeof, break, continue, for, do, while, switch, case, default, entry } alfabeto de palabras clave del lenguaje C.

En este contexto, los símbolos de un alfabeto suelen llamarse también letras o caracteres del alfabeto, pero no debemos confundirnos con el sentido corriente que se tiene de estos términos en los lenguajes naturales. Por ejemplo, "torre" es una secuencia de cinco símbolos yuxtapuestos (escritos uno a continuación del otro) vistos desde la perspectiva del alfabeto Σ_3 del Ejemplo 2.1, pero es un único símbolo pensado como parte del alfabeto Γ, representando a la pieza del mismo nombre del juego de ajedrez.

Lo mismo ocurre con Ω del Ejemplo 2.1. El manual de consulta presentado por Kernighan y Ritchie en el apéndice del libro *El Lenguaje de Programación C*, indica los elementos de Ω como las palabras clave del lenguaje, esto es, palabras que tienen un significado especial para un compilador C y que no pueden ser utilizadas como identificadores (reservadas). En la fase de análisis léxico (véase el Apéndice A referido a Compiladores e Intérpretes), el compilador detectará estas palabras como cadenas construidas con símbolos del alfabeto inglés {a, b, …, y, z}, pero durante el análisis sintáctico serán tomadas como símbolos, por lo que en esta fase el compilador usará algún alfabeto que incluya al alfabeto Ω.

Como se trata de conjuntos:

- *Dos alfabetos son **iguales** si y solo si tienen exactamente los mismos símbolos*, sin considerar el orden de presentación de los mismos o las repeticiones que pudieran mostrarse.
- Si todos los símbolos de un alfabeto Ω_1 están también en otro alfabeto Ω_2, se dice que *el primero está **incluido** en el segundo* y se lo denota $\Omega_1 \subseteq \Omega_2$ (*inclusión amplia*); también se dice que Ω_1 es un *subconjunto* de Ω_2. Si además, el segundo alfabeto posee al menos un símbolo que el primero no tiene, se dice que *el primero está **estrictamente incluido** en el segundo*, denotándose en este caso $\Omega_1 \subset \Omega_2$ (*inclusión estricta*); se expresa en este caso que Ω_1 es un *subconjunto propio* de Ω_2.
- El ***cardinal*** de un alfabeto dado Ω, denotado $|\Omega|$, es *la cantidad de símbolos que el mismo posee* y siempre será, por definición, un número entero positivo.

Es particularmente útil e importante la idea de combinar símbolos de un alfabeto, escribiendo uno luego del otro *concatenándolos* (o *yuxtaponiéndolos*), para formar nuevos objetos.

Palabra, cadena, tira o string

Una ***palabra definida sobre un alfabeto Σ*** *es cualquier secuencia finita de símbolos de* Σ**,** escritos uno a continuación del otro. Se usarán letras griegas minúsculas α, β, γ, δ, ω, …, con o sin subíndices para nombrarlas.

Las palabras (*cadenas, tiras* o *strings*) formadas con símbolos de algún alfabeto, pueden a su vez concatenarse entre ellas para formar nuevas palabras. Si α**=casa** y β**=blanca** son palabras definidas sobre el alfabeto Σ_3 de las letras minúsculas españolas (Ejemplo 2.1), entonces ***casablanca*** es otra palabra sobre el mismo alfabeto obtenida al concatenar α y β en ese orden (la concatenación de palabras claramente no es conmutativa pero sí es asociativa). Se indica el resultado de esta operación entre palabras con $\alpha \circ \beta$ o sencillamente $\alpha\beta$, donde se omite el símbolo de operación y quedan yuxtapuestas las palabras en el orden indicado.

Ejemplo 2.2

Sobre el alfabeto Σ_3 = {a, b, c, …, z}, se construyen las siguientes palabras y se ejemplifican algunas concatenaciones:

α = casa, β = blanca → $\alpha \circ \beta$ = casablanca

→ $\beta \circ \alpha$ = blancacasa

→ $\alpha \circ \beta \neq \beta \circ \alpha$ (no conmutativa)

α = torre, β = rey	→	$\alpha\beta$ = torrerey
	→	$\alpha\alpha = \alpha^2$ = torretorre
α = abra, β = cadabra	→	$(\alpha \circ \beta) \circ \alpha$ = abracadabra $\circ \alpha$
		= abracadabraabra
	→	$\alpha \circ (\beta \circ \alpha) = \alpha \circ$ cadabraabra
		= abracadabraabra
	→	$\mathbf{(\alpha \circ \beta) \circ \alpha = \alpha \circ (\beta \circ \alpha)}$ (asociativa)
$\alpha = a = a^1$	→	$\alpha \circ \alpha = aa = a^2$
	→	$(\alpha \circ \alpha) \circ \alpha = aa \circ \alpha = aaa = a^3$
	→	$((\alpha \circ \alpha) \circ \alpha) \circ \alpha = aaa \circ \alpha =$ aaaa $= a^4$
		(potenciación de símbolos)
α = a, β = b	→	$(\alpha \circ \alpha) \circ \beta = aa \circ \beta = aab = a^2b^1$
	→	$((\alpha \circ \alpha) \circ \beta) \circ \beta = aab \circ \beta = aabb = a^2b^2$

El ejemplo anterior, muestra una notación alterna para escribir una palabra que se forma como la concatenación de un símbolo consigo mismo; se llama a esto ***potenciación de símbolos de un alfabeto*** y se lo define recursivamente de la siguiente forma:

$$a^n = \begin{cases} \lambda & , \sin = 0 \\ a \circ a^{n-1} & , \sin > 0 \end{cases}$$

donde **a** es un símbolo del alfabeto, $\mathbf{n} \in \mathbb{N}$ es un número natural y λ denota a la cadena sin símbolos (ver luego, cadena vacía).

Longitud de una palabra

La cantidad de símbolos que conforman una palabra es llamada su ***longitud*** o ***largo*** y se denota con el nombre de la cadena entre barras verticales.

Ejemplo 2.3

Sobre el alfabeto Σ_3 = {a, b, c, …, z}, se pueden ejemplificar:

α = casa	→	$\lvert\alpha\rvert = 4$
β = blanca	→	$\lvert\beta\rvert = 6$
$\alpha \circ \beta$ = casablanca	→	$\lvert\alpha \circ \beta\rvert = \lvert\alpha\rvert + \lvert\beta\rvert = 10$
χ = torre	→	$\lvert\chi\rvert = 5$
δ = rey	→	$\lvert\chi\rvert = 3$
$\chi \circ \delta$ = torrerey	→	$\lvert\chi \circ \delta\rvert = 8$
a^3b^4 = aaabbbb	→	$\lvert a^3b^4 \rvert = \lvert$ aaabbbb $\rvert = 7$

Sobre el alfabeto Γ = {torre, caballo, alfil, rey, reina, peón}:

χ = torre	→	$\lvert\chi\rvert = 1$
δ = rey	→	$\lvert\delta\rvert = 1$
$\chi \circ \delta$ = torrerey	→	$\lvert\chi \circ \delta\rvert = 2$

Debe notarse que *torre* y *rey* representan en este último caso a un solo símbolo de Γ.

Cadena vacía

Como se anticipó, se denota con λ (lambda) a la ***palabra vacía***, *palabra que no tiene símbolos* y cuyo largo es cero: $|\lambda| = 0$.

¿Le parece extraño el concepto de palabra vacía?

Piense que cada vez que se define un concepto, por regla general, resulta interesante pensar en los límites del mismo. Si hablamos de conjuntos, llamamos vacío, denotado $\varnothing=\{\}$, al conjunto que no tiene elementos. Si hablamos de números naturales, llamamos cero (**0**) a aquel número que no tiene unidades; en el mismo sentido, llamamos palabra vacía (λ) a aquella que no tiene símbolos. Seguramente, usted conoce cómo funciona la instrucción iterativa **for** de C++ o Java, y debe entender lo que hace el fragmento de código:

```
for (int k=0; k < 10; k++);
```

Bien, piense ¿cuál es el cuerpo de esta instrucción **for**?

La palabra vacía resulta ser así, *elemento neutro* respecto de la operación de concatenación entre palabras, cualquiera sea el alfabeto sobre el cual las palabras estén definidas. En símbolos:

$$\forall\Sigma\text{: }\forall\alpha \text{ definida sobre } \Sigma\text{: } \alpha \circ \lambda = \lambda \circ \alpha = \alpha$$

Puede demostrarse además, que λ es la *única* palabra con estas propiedades (matemáticamente, el conjunto de todas las cadenas de símbolos de Σ junto a la operación de concatenación, conforman un semigrupo con elemento neutro λ).

La concatenación de una palabra consigo misma (ver Ejemplo 2.2) permite definir, en la misma forma que se ha hecho para símbolos, la ***potencia enésima de una palabra*** como *la concatenación **n** veces de la palabra consigo misma*; recursivamente se puede decir que si α es una palabra definida sobre algún alfabeto Σ y $n\in\mathbb{N}$ es un número natural, entonces su potencia enésima es:

$$\alpha^n = \begin{cases} \lambda & , \text{sin} = 0 \\ \alpha \circ \alpha^{n-1} & , \text{sin} > 0 \end{cases}$$

Si bien la potenciación de palabras está definida para números naturales, se acepta la notación α^{-1} para denotar a la ***palabra refleja de una cadena α***, obtenida escribiendo los símbolos de α en orden inverso.

Ejemplo 2.4

Sobre el alfabeto Σ_3={a, b, c, …, z}, consideremos la palabra α**=casa**. Entonces, podemos formar:

$\alpha^0 = \lambda$	α^2 = casa**casa**
α^1 = casa	α^3 = casa**casa**casa
α^{-1} = asac	α^4 = casa**casa**casa**casa**
$\alpha^1\alpha^{-1}$ = casaasac	α^5 = casa**casa**casa**casa**casa

Debe notarse que α^{-1} no es una potencia de α, sino solo una notación para la palabra refleja de α.

¿Qué es una palabra en programación?

Como lo indica su definición, una palabra, sobre algún alfabeto, es una secuencia finita de símbolos tomados del mismo. Pero ¿cómo sabemos, y en especial, cómo saben una computadora y nuestros programas, dónde empieza y termina una palabra?

Escrita sobre una hoja de papel el problema resulta casi trivial: la palabra **superfragilísticoespiralidoso** inicia con una letra **s**, termina con una **o** y está compuesta de 29 letras; nos damos cuenta de esto porque hay un espacio en blanco antes y después de la palabra delimitándola.

En algunos lenguajes de programación de computadoras, existe un tipo de datos ***string*** para definir palabras; en otros, las palabras se representan como arreglos de símbolos. Pero para su definición, almacenamiento y tratamiento existen básicamente dos formas de definir una palabra.

a) Se conoce su posición inicial (la dirección en la memoria de su primera letra, normalmente asociada al nombre de la variable que la contiene) y su longitud:

α = (2F8, 29) y en la dirección 2F8 estará **superfragilísticoespiralidoso**

b) Se conoce su posición inicial y existe un símbolo especial (*fin de cadena*, no perteneciente al alfabeto de símbolos sobre el cual se define la palabra) que marca su fin:

α = (2F8) y en la dirección 2F8 estará **superfragilísticoespiralidosoFDC**

Aquí FDC es un carácter especial que marca el final de la cadena (el lenguaje de programación C por ejemplo, utiliza el byte hexadecimal 00 para esto, denotado **\0**).

El diseñador de un lenguaje de programación de computadoras (y el constructor de su compilador), deberá tomar una decisión y optar por una de las formas de representar las palabras, para que los programas escritos con el mismo puedan utilizarlas.

Aquellas palabras que son iguales a sus reflejas se denominan *palíndromos*, como por ejemplo *neuquen*, *1234321*, *ala, orejero* y *rotor*; estas palabras se leen igual de izquierda a derecha y de derecha a izquierda. Si los símbolos son dígitos, suele también llamarse *capicúas* a los números que se leen igual en ambos sentidos.

Partes de una palabra

Por supuesto, si dos palabras pueden "juntarse" por concatenación, una palabra cualquiera puede dividirse en sus símbolos constituyentes y hasta en subpalabras. Supongamos que una cadena puede escribirse como concatenación de otras tres, digamos $\omega=\alpha\chi\beta$; entonces decimos que α es un ***prefijo*** de ω (primera parte de omega), que β es un ***sufijo*** de ω (última parte de omega) y que χ es una ***subpalabra*** de ω (una parte de omega). Aquí, cualquiera de las tres cadenas α, χ o β pueden ser vacías. De un sufijo o prefijo de una palabra se dice que es ***propio*** si no es la misma palabra en cuestión o la palabra vacía.

Ejemplo 2.5

Sea Σ_3 = {a, b, c, …, z} el alfabeto de las letras minúsculas españolas y considérese la palabra **ω=prueba** definida sobre él. Sus prefijos y sufijos son mostrados en la Tabla 2.1. Además, tanto los prefijos como los sufijos indicados son subpalabras de ω y también lo son cadenas intermedias como **ru**, **rue**, **rueb**, **r**, **u**, **e**, **b**, **eb**, **ueb**, **ue**.

Prefijos de ω		Sufijos de ω	
	λ	λ	
Propios	p	a	Propios
	pr	ba	
	pru	eba	
	prue	ueba	
	prueb	rueba	
	prueba	prueba	

Tabla 2.1: Prefijos y sufijos.

Nótese que para una palabra cualquiera de largo **n**, siempre se tienen (**n+1**) sufijos y (**n+1**) prefijos debido a que la palabra vacía siempre puede considerarse concatenada a izquierda o derecha por ser λ elemento neutro de esta operación.

Operaciones con alfabetos

Retomando el tema de alfabetos, como conjuntos que son, los mismos pueden unirse $\Sigma_1 \cup \Sigma_2$={todos los símbolos comunes y no comunes de ambos alfabetos}, intersecarse $\Sigma_1 \cap \Sigma_2$={todos los símbolos comunes a ambos alfabetos}, restarse $\Sigma_1 - \Sigma_2$={todos los símbolos del primer alfabeto que no estén en el segundo} (también se dice que este conjunto es el complemento relativo del Σ_2 respecto de Σ_1) y complementarse $\overline{\Sigma_1}$ (en este caso, deberemos definir el universal contra el cual complementar).

También pueden operarse con el producto cartesiano para obtener pares ordenados de símbolos, operación que, en general, no tiene mayor interés; pero pensando en obtener *palabras* en lugar de *tuplas* ordenadas, se quiere proponer algo similar al producto cartesiano, una nueva ope-

ración: *dados dos alfabetos* Σ_1 *y* Σ_2*, la* ***concatenación de*** Σ_1 ***y*** Σ_2*, denotada* $\Sigma_1 \circ \Sigma_2$ *o simplemente* $\Sigma_1\Sigma_2$*, es el conjunto de palabras formadas por la concatenación de cada símbolo de* Σ_1 *con cada uno de* Σ_2*, en ese orden*.

Debe notarse que lo obtenido al concatenar dos alfabetos, no es generalmente un nuevo alfabeto, sino un conjunto de palabras; el alfabeto mismo puede ser pensado como un conjunto de palabras de longitud uno, con lo cual podemos extender esta operación para concatenar más de dos alfabetos e inclusive cualquier par de conjuntos de palabras, con solo cambiar "*símbolo de*" por "*palabra sobre*", en la anterior definición.

Ejemplo 2.6

Sean los alfabetos $\Sigma_2 = \{0, 1\}$ y $\Sigma_5 = \{a, b, c\}$. Entonces:

$$\Sigma_2 \circ \Sigma_5 = \{0a, 0b, 0c, 1a, 1b, 1c\}$$

$$\Sigma_5 \circ \Sigma_2 = \{a0, b0, c0, a1, b1, c1\}$$

Extendiendo esta idea, ahora se podrían concatenar dos o más alfabetos haciendo:

$$\Sigma_5 \circ \Sigma_2 \circ \Sigma_5 = \{a0a, b0a, c0a, a1a, b1a, c1a,$$
$$a0b, b0b, c0b, a1b, b1b, c1b,$$
$$a0c, b0c, c0c, a1c, b1c, c1c\}$$

donde a cada palabra de $(\Sigma_5 \circ \Sigma_2)$ se le agrega al final por concatenación un símbolo de Σ_5 obteniéndose un nuevo conjunto de palabras.

Ejemplo 2.7

Es interesante observar lo que ocurre al concatenar un alfabeto consigo mismo. Tomemos el alfabeto binario $\Sigma_2 = \{0, 1\}$; al concatenarlo consigo mismo se obtiene:

$$\Sigma_2 \circ \Sigma_2 = \{0, 1\} \circ \{0, 1\} = \{00, 01, 10, 11\}$$

que constituye el conjunto de todas las palabras de largo dos (**2**) que pueden formarse con los símbolos **0** y **1** del alfabeto; denotemos Σ_2^2 a este resultado, correspondiendo el exponente simultáneamente a la cantidad de alfabetos concatenados y al largo de las palabras obtenidas. Claramente, sin símbolos solo podrá formarse la palabra vacía y con solo un símbolo (los del alfabeto sin combinar), se podrán formar tantas palabras de largo uno como símbolos tenga el alfabeto. Así, puede escribirse:

$\Sigma_2^0 = \{\lambda\}$ $\quad$ $\Sigma_2^1 = \{0, 1\}$ $\quad$ $\Sigma_2^2 = \Sigma_2^1 \circ \Sigma_2^1 = \{0, 1\} \circ \{0, 1\} = \{00, 01, 10, 11\}$

$\Sigma_2^3 = \Sigma_2^1 \circ \Sigma_2^2 = \{0, 1\} \circ \{00, 01, 10, 11\} = \{000, 001, 010, 011, 100, 101, 110, 111\}$

…

$\Sigma_2^n = \Sigma_2^1 \circ \Sigma_2^{n-1} = \{0, 1\} \circ \{$palabras de largo **n - 1** formadas con ceros y unos$\}$
$= \{$palabras de largo **n** formadas con ceros y unos$\}$

Se define recursivamente la ***potenciación de un alfabeto*** (lo que permite obtener conjuntos de palabras de largo **n** formadas con los símbolos del mismo), de la siguiente forma:

$$\Sigma^n = \begin{cases} \{\lambda\} & , \sin = 0 \\ \Sigma \circ \Sigma^{n-1} & , \sin > 0 \end{cases}$$

Se denomina además, **universo de discurso de un alfabeto** Σ, al conjunto de todas las palabras que pueden formarse con sus símbolos, sean del largo que sean. Este conjunto, también llamado lenguaje universal de Σ, suele denotarse $\mathbf{W(\Sigma)}$ y con mayor frecuencia Σ^* (que se lee sigma estrella o estrella de Kleene[1] de Σ). Usando las potencias sucesivas del alfabeto Σ, el universo

[1] En honor a Stephen Kleene, lógico que introdujo esta operación a mediados del siglo XX.

de discurso está compuesto por todas las cadenas de símbolos de sigma de largo cero (Σ^0), todas las de largo uno (Σ^1), todas las de largo dos (Σ^2), … y así sucesivamente. Se puede escribir entonces:

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots \cup \Sigma^i \cup \ldots$$

o resumidamente:

$$\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$$

Por sus propiedades, la estrella de Kleene de un alfabeto también recibe el nombre de ***cierre*** o ***clausura*** (reflexiva y transitiva) del alfabeto; si se quita de la unión la potencia cero, esto es, se elimina la palabra vacía del conjunto Σ^*, se obtiene:

$$\Sigma^+ = \bigcup_{i=1}^{\infty} \Sigma^i$$

$$\Sigma^+ = \bigcup_{i=0}^{\infty} \Sigma^i - \Sigma^0$$

$$\Sigma^+ = \Sigma^* - \{\lambda\}$$

el conjunto de todas las palabras de símbolos de Σ de largo uno o mayor, que es denominado cierre positivo del alfabeto Σ.

Ejemplo 2.8

Usando nuevamente el alfabeto $\Sigma_2 = \{0, 1\}$, su cierre positivo y clausura pueden presentarse como:

$\sum_2^+ = \{0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, \ldots\}$

$\sum_2^* = \sum_2^+ \cup \{\lambda\}$

$= \{\lambda, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, \ldots\}$

Por otro lado, si se piensa en el alfabeto de las letras minúsculas y mayúsculas españolas $\Sigma_e = \{a, b, \ldots, z, A, B, \ldots, Z\}$, salvo acentos y diéresis, todas las palabras del español incluidas en el diccionario, constituyen un subconjunto del universo de discurso de Σ_e^*, que además contendrá como secuencias de símbolos, palabras sin significado para el idioma.

Lenguajes y operaciones

Se ha estado tratando con conjuntos de palabras obtenidos como concatenaciones y potencias de alfabetos; estos conjuntos no son alfabetos ya que no tienen símbolos sino palabras como elementos.

Estos conjuntos, aún desprovistos de un nombre particular, conformarán un concepto fundamental de nuestro estudio de la lingüística matemática.

Lenguaje

La idea más general, y a la vez la más sencilla de lenguaje, puede expresarse como sigue: un **lenguaje** definido sobre un alfabeto, es un conjunto de palabras construidas con los símbolos de ese alfabeto.

En símbolos, si Σ es un alfabeto y **L** es un lenguaje definido sobre Σ, entonces:

$$\mathbf{L} \subseteq \Sigma^*$$

Al ser los lenguajes conjuntos de palabras, puede aplicarse toda la teoría de conjuntos a ellos. Algunos conceptos de conjuntos adaptados al contexto de lenguajes son los que siguen:

- Un lenguaje $\mathbf{L_1}$ definido sobre Σ es subconjunto de otro $\mathbf{L_2}$ si toda palabra de $\mathbf{L_1}$ es también palabra de $\mathbf{L_2}$:

$$L_1 \subseteq L_2 \Leftrightarrow \forall \alpha \in \Sigma^*: \alpha \in L_1 \rightarrow \alpha \in L_2$$

- Un lenguaje $\mathbf{L_1}$ definido sobre Σ es subconjunto propio de otro $\mathbf{L_2}$ si toda palabra de $\mathbf{L_1}$ es también palabra de $\mathbf{L_2}$ y existe al menos una palabra del segundo que no es elemento del primero:

$$L_1 \subset L_2 \Leftrightarrow (\forall \alpha \in \Sigma^*: \alpha \in L_1 \rightarrow \alpha \in L_2) \wedge (\exists \beta \in \Sigma^* / \beta \in L_2 \wedge \beta \notin L_1)$$

- Un lenguaje $\mathbf{L_1}$ definido sobre Σ es igual a otro lenguaje $\mathbf{L_2}$ si tienen exactamente las mismas palabras:

$$L_1 = L_2 \Leftrightarrow [\forall \alpha \in \Sigma^*: \alpha \in L_1 \leftrightarrow \alpha \in L_2]$$
$$\Leftrightarrow [L_1 \subseteq L_2 \wedge L_2 \subseteq L_1]$$

- El conjunto vacío $\varnothing$ es un lenguaje llamado lenguaje vacío, tiene cardinalidad cero y es el único con esta propiedad, independiente del alfabeto sobre el cual esté definido.
- $\mathbf{\{\lambda\}}$, el conjunto cuyo único elemento es la cadena vacía, es también un lenguaje único e independiente del alfabeto de definición. Debe notarse que su cardinalidad es uno, por lo cual es bien distinto del lenguaje vacío definido anteriormente.

Operaciones con lenguajes

Si $\mathbf{L_1}$ y $\mathbf{L_2}$ son lenguajes definidos sobre el alfabeto Σ, entonces también lo son la unión $\mathbf{L_1 \cup L_2}$, intersección $\mathbf{L_1 \cap L_2}$, resta $\mathbf{L_1 - L_2}$, concatenación $\mathbf{L_1 \circ L_2}$ y el complemento $\overline{L_1}$, definidos de la siguiente forma:

$L_1 \cup L_2 = \{\alpha / \alpha \in L_1 \vee \alpha \in L_2\}$

= {palabras pertenecientes al menos a uno de los dos lenguajes}

$L_1 \cap L_2 = \{\alpha / \alpha \in L_1 \wedge \alpha \in L_2\}$

= {palabras comunes a ambos lenguajes}

$L_1 - L_2 = \{\alpha / \alpha \in L_1 \wedge \alpha \notin L_2\}$

= {palabras del primer lenguaje que no están en el segundo}

$L_1 \circ L_2 = \{\mu = \alpha\beta / \alpha \in L_1 \wedge \beta \in L_2\}$

= {palabras obtenidas al concatenar cada una de las palabras del primer lenguaje con cada una de las del segundo, en ese orden}

$\overline{L_1} = \{\alpha / \alpha \in \Sigma^* \wedge \alpha \notin L_1\}$

= {todas las palabras del universo de discurso de Σ que no estén en el lenguaje L_1}

Si los lenguajes no tuvieran alfabetos comunes, esto es $\mathbf{L_1}$ está definido sobre $\mathbf{\Sigma_1}$ y $\mathbf{L_2}$ sobre $\mathbf{\Sigma_2}$, siempre se puede pensar en la unión $\mathbf{\Sigma_1 \cup \Sigma_2}$ como un alfabeto común a ambos, sobre el cual estarán definidos los lenguajes compuestos como resultado de operar entre $\mathbf{L_1}$ y $\mathbf{L_2}$.

Ejemplo 2.9

Considérense los siguientes lenguajes, definidos sobre el alfabeto de dígitos decimales Σ_{dec} = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}:

L_1 = {10, 11, 12, 13,…, 98, 99}

L_2 = {numerales de los números naturales de dos dígitos}

L_3 = {10, 11, 12, 13, 14, 15, 16, 17, 18, 19}

L_4 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

Entonces, se verifica lo siguiente:

a) $L_1 = L_2$ ya que, si bien las definiciones de ambos lenguajes son distintas, se están refiriendo al mismo conjunto de cadenas.

b) $L_3 \subseteq L_1$ porque las diez cadenas del primer lenguaje también son elementos del segundo lenguaje. Inclusive puede afirmarse que $L_3 \subset L_1$ porque además L_1 tiene cadenas que no están en L_3.

c) $L_3 \cup L_4$ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19} esto es, todas las cadenas comunes y no comunes a ambos lenguajes.

d) $L_3 \cup L_1 = L_1$ porque el primer lenguaje está incluido en el segundo.

e) $L_3 \cap L_2 = L_3$ ya que el primer lenguaje está incluido en el segundo.

f) $L_4 \cap L_2$ = {10}, 10 es la única cadena común a ambos lenguajes.

g) $L_1 - L_2$ = {} ya que siendo iguales los lenguajes, al quitar todas las cadenas del segundo lenguaje a las del primero, se obtiene el conjunto vacío.

h) $L_4 - L_3$ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} se quita del primer lenguaje la única cadena común a ambos: 10.

i) $L_3 - L_4$ = {11, 12, 13, 14, 15, 16, 17, 18, 19}, ídem anterior.

j) $L_4 \circ L_3$ = {010, 011, 012, 013, 014, 015, 016, 017, 018, 019,
110, 111, 112, 113, 114, 115, 116, 117, 118, 119,
210, 211, 212, 213, 214, 215, 216, 217, 218, 219,
310, 311, 312, 313, 314, 315, 316, 317, 318, 319,
410, 411, 412, 413, 414, 415, 416, 417, 418, 419,
510, 511, 512, 513, 514, 515, 516, 517, 518, 519,
610, 611, 612, 613, 614, 615, 616, 617, 618, 619,
710, 711, 712, 713, 714, 715, 716, 717, 718, 719,
810, 811, 812, 813, 814, 815, 816, 817, 818, 819,
910, 911, 912, 913, 914, 915, 916, 917, 918, 919,
1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017,
1018, 1019}

Si se considera el universo de discurso Σ^*_{dec}, esto es, todas las posibles cadenas de dígitos decimales, entonces:

k) $\overline{L_1}$ = {todas las cadenas de dígitos decimales de longitud $\neq 2$}

= $\{\alpha \in \Sigma^*_{dec} / \mid \alpha \mid \neq 2\}$

Procediendo en forma similar a lo hecho con alfabetos y usando la concatenación de un lenguaje **L** consigo mismo, se puede definir la ***potenciación*** de lenguajes recursivamente, de la siguiente forma:

$$L^n = \begin{cases} \{\lambda\} & , sin = 0 \\ L \circ L^{n-1} & , sin > 0 \end{cases}$$

Además, pueden extenderse a lenguajes los operadores estrella de Kleene y cierre positivo basados en potenciación y unión, haciendo:

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

conjunto denominado estrella de Kleene del lenguaje **L** o también cierre o clausura (reflexiva y transitiva); si se quita de la unión la potencia cero se obtiene:

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

Debe notarse que, a diferencia de lo visto con alfabetos, en general, se tiene:

$$L^+ \neq \bigcup_{i=0}^{\infty} L^i - L^0$$

$$L^+ \neq L^* - \{\lambda\}$$

ya que si originariamente el lenguaje **L** tuviera como elemento a la cadena vacía ($\lambda \in \mathbf{L}$), entonces la primer potencia de L también la tendría como elemento. Sin embargo, siempre podremos decir que la clausura se obtiene agregando la cadena vacía al cierre positivo:

$$L^* = L^+ \cup \{\lambda\}$$

Ejemplo 2.10

Sea L = {mar, casa} un lenguaje de solo dos palabras y veamos cómo se opera con la potenciación antes definida. Se tienen para las primeras potencias:

L^0 = {λ}

L^1 = {mar, casa}

L^2 = {marmar, marcasa, casamar, casacasa }

L^3 = {marmarmar, marmarcasa, casacasamar, casacasacasa,
marcasamar, marcasacasa, casamarmar, casamarcasa}

L^4 = {marmarmarmar, marmarmarcasa, marmarcasamar,
marmarcasacasa, casacasamarmar, casacasamarcasa,
casacasacasamar, casacasacasacasa, marcasamarmar,
marcasamarcasa, marcasacasamar, marcasacasacasa,
casamarmarmar, casamarmarcasa, casamarcasamar,
casamarcasacasa}

y así sucesivamente. Nótese que $\mathbf{L^*}$ se formará de la unión de todos estos conjuntos y siguiendo hasta el infinito (conjunto de infinitas cadenas finitas), conteniendo la cadena vacía. $\mathbf{L^+}$ será el mismo conjunto pero sin la potencia cero (la cadena vacía).

Descripción de lenguajes

Se estableció el concepto más general y simple de lenguaje como un conjunto de palabras. Como todo conjunto, un lenguaje puede definirse o determinarse:

- Por ***extensión*** o ***enumeración***, indicando una por una todas las palabras que son elementos del mismo, o
- Por ***comprensión***, explicitando propiedades de las cadenas definidas sobre algún alfabeto, que solo las palabras del lenguaje verificarán.

La determinación por extensión es simple y clara, pero solo es útil para lenguajes finitos y que tengan un reducido número de elementos, ya que hay que escribir *todas* las palabras del lenguaje.

Ejemplo 2.11

Los siguientes lenguajes están determinados por extensión:

L_1 = {espn, sports, espnmas, teledeportes}; sobre $\sum_3$ de letras minúsculas españolas.

L_2 = {00, 01, 10, 11}; sobre $\sum_2$ = {0, 1}.

L_3 = {torre, caballo, alfil, rey, reina, peón}; sobre $\sum_3$ de letras minúsculas españolas.

En la definición por comprensión, la especificación de las propiedades que deben tener las palabras del lenguaje, puede hacerse de diversas formas: coloquialmente, por fórmulas, por conjunto con una propiedad, y de muchas otras maneras.

Ejemplo 2.12

Coloquialmente: propiedades explicitadas en lenguaje natural.

T = {usuarios de números de teléfono fijo válidos en la ciudad de Córdoba, Argentina, al 31/12/2012}

P = {programas correctos escritos en lenguaje C}

Conjunto con una propiedad: propiedades dadas en símbolos.

$L_2 = \{\alpha \,/\, \alpha \in \{0,1\}^* \wedge |\alpha| = 2\}$

$A = \{\alpha \,/\, \alpha \in \{0,1\}^+ \wedge |\alpha| = 3 \wedge \alpha = \beta 00\delta \wedge \beta, \delta \in \{0,1\}^*\}$

Fórmula lingüística o algebraicamente: propiedad expresada utilizando potenciaciones, uniones, intersecciones, concatenaciones, etcétera.

$B = \{\alpha = a^n bc^m \,/\, n \geq 1, 0 < m < 4\}$

$C = \{\alpha = Begin^n x^m End^n \,/\, n \geq 1, m \geq 0\}$

(*Begin* y *End* son tomados como un solo símbolo cada uno)

El lenguaje **T** del Ejemplo 2.12, es un lenguaje finito, como lo demuestra la existencia de una guía telefónica para la ciudad indicada en la fecha establecida, pero claramente su determinación por comprensión es la más indicada para ser incluida aquí. La definición del lenguaje **A,** describe en símbolos las cadenas de ceros y unos de largo tres que tengan al menos una subpalabra **00** (esto es, el lenguaje {001, 100}), mientras que las cadenas de los lenguajes **B** y **C**, se especifican utilizando patrones que se conforman apelando a la concatenación y la potenciación de símbolos.

Con estas notaciones, se pueden determinar infinitos lenguajes, y según el objetivo con el que se lo esté haciendo, será más útil expresar el lenguaje en una de ellas que hacerlo con otra.

Considérese ahora el conjunto **P** del anterior ejemplo, de *todos los programas correctos escritos en lenguaje C*; su descripción coloquial parece concisa y clara a primera vista, pero el conjunto ¿está bien definido? Primero debe preguntarse ¿qué se entiende por *programa correcto*?; y aun suponiendo que se llega a un acuerdo sobre esto (por ejemplo, solo pidiendo que no tenga errores, sin tener en cuenta su objetivo o funcionamiento) luego aparece el interrogante: ¿cómo se determina si un programa fuente escrito en lenguaje C pertenece o no al conjunto **P**?

Lo primero que se piensa es pasar el programa fuente por el procesamiento de algún compilador C que se tenga a la mano, y si al finalizar dice ***0 errors - 0 warnings*** o algo por el estilo, diremos que el programa está en el conjunto **P**. De acuerdo, pero ahora debemos preguntarnos ¿cómo sabe el compilador que el programa es correcto?, esto es, que no tiene errores según el anterior convenio.

Un programa fuente, es una larga tira de símbolos formada por la concatenación de caracteres definidos como válidos para el lenguaje de programación (*el alfabeto del lenguaje*); pero no cualquier cadena de caracteres válidos constituye un programa fuente; hay reglas que cumplir que son establecidas en los manuales del lenguaje por sus diseñadores y son ellas las que definen qué es un programa correcto y qué no lo es; estas reglas deben ser conocidas por el compilador y, claro está, por todos aquellos que deseen escribir programas en C.

El compilador es un programa de computación que necesita para hacer su trabajo, poder determinar si una cadena (*el programa fuente*) cumple las reglas definidas por el lenguaje fuente y,

para ello, requiere una descripción bastante más precisa y detallada de lo que es un programa C correcto, que la que pueden dar las anteriores notaciones.

Debe utilizarse algo *más expresivo* que la descripción coloquial, la dada por fórmulas o por un grupo sencillo de propiedades. Además, estamos fuertemente interesados en alguna herramienta descriptiva que no solo ayude a determinar la pertenencia o no pertenencia de una cadena a un lenguaje, sino que permita generar procedimientos automáticos para la tarea de determinar la corrección de un programa fuente. Para elaborar esta nueva forma de determinar un conjunto de cadenas, se deberán primero establecer algunos nuevos conceptos, centrales a la Teoría de Lenguajes Formales.

Gramáticas formales

Reglas de reescritura o producciones

Se iniciará el estudio, con un operador que trabaja sobre cadenas para generar nuevas cadenas al aplicarlo convenientemente.

Para un alfabeto Σ dado, diremos que una ***producción*** o ***regla de reescritura*** es un par ordenado de palabras **(α, β)** definidas sobre Σ. Escribiremos las producciones utilizando una notación similar a la denominada **BNF**[2] de la siguiente forma:

$$\alpha := \beta$$

y diremos que la cadena α es el lado izquierdo de la producción, **:=** el símbolo de producción, la cadena β su lado derecho y la leeremos *alfa produce beta* o también *alfa puede reescribirse como beta*.

Puede pensarse en β como una redefinición del concepto α, que muchas veces tomará el formato de una definición recursiva.

En su forma más general, una producción $\alpha{:=}\beta$ constituye un *operador unario* que puede ser aplicado a una palabra $\delta = \omega\alpha\rho$, si y solo si el lado izquierdo de la producción (la cadena α) coincide con una subpalabra de δ; el efecto de aplicar la producción $\alpha{:=}\beta$ a la palabra δ es el reemplazo o sustitución de α por β en δ para obtener una nueva palabra $\varphi{=}\omega\beta\rho$. Veremos luego que esta forma general de producción se restringe para que el lado izquierdo contenga un solo símbolo que se reemplaza por el lado derecho, dentro de un contexto o sin él.

Uso de las producciones

Se llama derivación directa a la operación que aplica una sola producción a una palabra obteniendo una nueva palabra y se simboliza:

$$\delta \rightarrow \varphi$$

Se dice que δ se deriva directamente en φ, o que φ se obtiene por derivación directa desde δ, por la aplicación de una producción.

Ejemplo 2.13

Sea δ**=casablanca** una cadena definida sobre el alfabeto de letras minúsculas españolas y **blanca:=negra** una producción dada. Como el lado izquierdo de la producción (*blanca*), coincide, en este caso, con un sufijo de la cadena δ, entonces puede escribirse:

$$\text{casablanca} \rightarrow \text{casanegra}$$

donde la cadena **casablanca** se deriva directamente en **casanegra** por la aplicación de la producción. Si además, existiera otra producción **a:=e** disponible junto con la anterior, se podrían efectuar en secuencia, las siguientes derivaciones directas:

[2] John Backus, quien dirigía el grupo que creó el lenguaje ALGOL en la década del cincuenta, fue quien utilizó esta notación para describir la sintaxis del lenguaje. Peter Naur, que escribió un influyente informe sobre ALGOL en 1963, llamó a la notación BNF por *Forma Normal de Backus*. Debido a una recomendación posterior de Donald Knuth (autor de la obra clásica *The Art of Computer Programming*), se cambió su significado a *Forma de Backus-Naur*. Luego, en este capítulo, se describirá BNF con más detalle.

$$\text{casablanca} \rightarrow \text{casanegra} \rightarrow \text{cesanegra} \rightarrow \text{cesenegra} \rightarrow \text{cesenegre}$$

donde se han aplicado la primera producción una vez y luego, la segunda producción tres veces consecutivas.

Cada aplicación de una producción en el ejemplo anterior, constituye una derivación directa que transforma una cadena en otra, pero claramente la cadena inicial **casablanca** se ha transformado en una nueva cadena **cesenegre** al final del proceso.

Ejemplo 2.14

Muchas frases del idioma español, denominadas bimembres, tienen dos miembros principales denominados *sujeto* y *predicado*: el predicado es *lo que se dice* en la frase y el sujeto es *de quién se dice* lo que expresa el predicado. Hay muchas formas en español para cada una de estas construcciones sintácticas; sin pretender ser riguroso ni exhaustivo al respecto, y solo a modo de ejemplo, se presenta una sencilla muestra utilizando el concepto de producciones:

<Frase> := <Sujeto> <Predicado>
<Sujeto> := <Artículo> <Sustantivo>
<Predicado> := <Verbo> <ObjetoDirecto>
<ObjetoDirecto> := <Preposición> <Artículo> <Sustantivo>
<ObjetoDirecto> := <Artículo> <Sustantivo>
<ObjetoDirecto> := <Sustantivo>
<Artículo> := la
<Artículo> := el
<Sustantivo> := gato
<Sustantivo> := rata
<Verbo> := corre
<Verbo> := come
<Preposición> := a
<Preposición> := con

En las anteriores producciones, todas las palabras son consideradas símbolos individuales y no concatenaciones de letras. Entre estos símbolos, pueden distinguirse claramente dos tipos distintos a saber:

- Aquellos que se encuentran entre corchetes angulares, que como puede verse se reescriben de distintas formas según las producciones que los tienen en su lado izquierdo (se denominan símbolos auxiliares o no terminales) y
- Los símbolos que no están entre corchetes angulares y para los cuales no hay reglas de reescritura que los tengan en su lado izquierdo (se denominan símbolos terminales).

Con solo las producciones dadas, pueden derivarse desde el símbolo inicial <Frase> una cantidad de palabras distintas. Por ejemplo:

<Frase> → <Sujeto> <Predicado> →

… → <Artículo> <Sustantivo> <Predicado> →

… → el <Sustantivo> <Predicado> → el gato <Predicado> →

… → el gato <Verbo> <ObjetoDirecto> →

… → el gato corre <ObjetoDirecto> →

… → el gato corre <Artículo> <Sustantivo> →

… → el gato corre la <Sustantivo> → **el gato corre la rata**

En resumen, el símbolo ***<Frase>*** se ha transformado en la palabra ***el gato corre la rata*** aplicando en secuencia algunas de las producciones posibles.

También se podrían haber derivado otras muchas frases de símbolos terminales: *el gato come la rata, el gato corre a la rata, la rata corre el gato,* etcétera, y démonos cuenta de que todas las palabras intermedias que contienen tanto símbolos terminales como no terminales, también son derivadas desde el símbolo inicial **<Frase>**.

Se llama sencillamente ***derivación***, a la operación de aplicar una secuencia finita de producciones a una cadena δ dada para obtener otra cadena φ, y se la simboliza:

$$\delta \rightarrow^{*} \varphi$$

que se lee δ *se deriva en* φ, o φ *se deriva de* δ. Esto equivale a decir que existen cadenas $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}, \alpha_n$ tales que:

$$\delta = \alpha_0 \rightarrow \alpha_1 \rightarrow \ldots \rightarrow \alpha_{n-1} \rightarrow \alpha_n = \varphi$$

Usualmente, se trabajará con varias producciones que se podrán aplicar a distintas cadenas; por ello, a veces será esclarecedor enumerar las producciones de que se dispone, no para ordenarlas sino al solo fin de etiquetarlas, para luego indicar efectivamente cómo se hizo la derivación indicando en cada derivación directa, la regla que se utilizó para transformar una cadena α_i en la siguiente α_{i+1}.

Ejemplo 2.15

Supóngase tener a disposición las siguientes producciones:

1) blanca := negra 2) a := e 3) casa := cosa

Entonces, puede efectuarse la siguiente derivación:

$$casablanca \rightarrow^{*} cosanegre$$

resultado de aplicar las producciones en cierto orden a la cadena ***casablanca***. Algunas posibles secuencias podrían ser:

$$casablanca \overset{1}{\rightarrow} casanegra \overset{2}{\rightarrow} casanegre \overset{3}{\rightarrow} cosanegre$$

$$casablanca \overset{3}{\rightarrow} cosablanca \overset{1}{\rightarrow} cosanegra \overset{2}{\rightarrow} cosanegre$$

donde los números que etiquetan las flechas indican qué producción se utilizó para efectuar cada una de las derivaciones directas indicadas.

Si se dispone de varias producciones entre las cuales optar para efectuar derivaciones de cadenas, suele ocurrir que más de una producción es aplicable a una cadena dada en cada paso del proceso, por lo cual se debe decidir qué producción aplicar primero y cuál después.

El Ejemplo 2.15 muestra que, gracias a esto, pueden existir distintos caminos a seguir para obtener una cadena por derivación desde otra; en particular, pueden existir caminos sin salida, esto es, una secuencia de derivaciones directas que no lleve a la cadena final buscada, por lo que

la derivación de cadenas se convierte en un proceso de búsqueda de las producciones a aplicar y del orden en el cual hacerlo, para lograr obtener una cadena a partir de otra.

Si durante el proceso de derivación, cada vez que puede optarse por una producción a aplicar se efectúa el reemplazo posible más a la derecha en la cadena, se dice que se ha efectuado una ***derivación por la derecha***. Si el reemplazo que siempre se elige es el de más a la izquierda posible, se dice que se ha hecho una ***derivación por la izquierda***. En los casos en los que las opciones se toman mezcladas, la derivación puede denominarse ***mixta***.

Ejemplo 2.16

Consideremos las siguientes producciones (donde se consideran como símbolos a **Número** y a **Dígito**) que especifican cómo escribir el numeral de un número entero sin signo.[3]

1) Número := Dígito
2) Número := Dígito Número
3) Dígito := 0
4) Dígito := 1
5) Dígito := 2
6) Dígito := 3
7) Dígito := 4
8) Dígito := 5
9) Dígito := 6
10) Dígito := 7
11) Dígito := 8
12) Dígito := 9

Para transformar el símbolo **Número** en la cadena **272**, se puede proceder a efectuar las siguientes derivaciones:

Por derecha: **Número** $\rightarrow$ Dígito Número $\rightarrow$ Dígito Dígito Número $\rightarrow$ …

… $\rightarrow$ Dígito Dígito Dígito $\rightarrow$ Dígito Dígito 2 $\rightarrow$ Dígito 72 $\rightarrow$ **272**

Por izquierda: **Número** $\rightarrow$ Dígito Número $\rightarrow$ 2 Número $\rightarrow$ …

… $\rightarrow$ 2 Dígito Número $\rightarrow$ 27 Número $\rightarrow$ 27 Dígito $\rightarrow$ **272**

Mixta: **Número** $\rightarrow$ Dígito Número $\rightarrow$ 2 Número $\rightarrow$ …

… $\rightarrow$ 2 Dígito Número $\rightarrow$ 2 Dígito Dígito $\rightarrow$ 27 Dígito $\rightarrow$ **272**

Queda como ejercicio colocar el número de la producción usada sobre cada una de las flechas en las anteriores derivaciones.

En ocasiones, es necesario y muy productivo proceder al revés de como se lo ha estado haciendo; desde una cadena dada, se pueden "desandar" los posibles pasos efectuados para derivarla desde otra, haciendo lo que se denominan *reducciones*. Se llama ***reducción directa*** a la operación inversa de una derivación directa. Así, se dice que φ *puede reducirse a* δ, si existe una producción que aplicada a δ la transforma en φ, y se simboliza:

$$\delta \leftarrow \varphi$$

Si lo que se hacen son múltiples reducciones directas para descubrir o revertir una derivación completa, simplemente se dice que se hizo una ***reducción*** y se la denota:

$$\delta \overset{*}{\leftarrow} \varphi$$

que equivale a decir que existen cadenas $\alpha_0, \alpha_1, \ldots, \alpha_{n-1}, \alpha_n$ tales que:

$$\delta = \alpha_0 \leftarrow \alpha_1 \leftarrow \ldots \leftarrow \alpha_{n-1} \leftarrow \alpha_n = \varphi$$

Estamos ahora en condiciones de comprender una nueva forma de especificar lenguajes, más expresiva que las anteriormente vistas y capaz de inducir algoritmos para la detección de cadenas correctamente escritas según sus especificaciones.

[3] Usando BNF se escribirían estas producciones luego en una forma mucho más concisa, utilizando: <Número>:=<Dígito>|<Dígito><Número>, <Dígito> := 0|1|2|3|4|5|6|7|8|9.

Gramática formal

En 1956, el lingüista norteamericano Noam Chomsky presentó su Teoría de las Gramáticas Transformacionales-Generativas donde, de alguna forma, puede decirse que *aritmetizó la lengua*, en el mismo sentido que George Boole cien años antes lo había hecho con la lógica.

Chomsky propuso una nueva forma de estudiar lenguajes usando símbolos y notaciones *al estilo matemático* (usando los sistemas de producción desarrollados anteriormente por Emil Post), construyendo un álgebra para el tratamiento de los lenguajes; si bien inicialmente su trabajo trataba sobre los lenguajes naturales (en particular el inglés), sus ideas fueron rápidamente adaptadas y utilizadas por los diseñadores de lenguajes de computadoras, como John Backus y Peter Naur que en la misma época desarrollaban los primeros lenguajes de programación de computadoras de alto nivel: Fortran y Algol.

Una ***gramática formal G*** es una cuádrupla **(Σ_T, Σ_N, S, P)**, en la cual sus cuatro componentes representan:

- Σ_T es el alfabeto de los símbolos que formarán las cadenas del lenguaje que se está describiendo y es denominado ***alfabeto de símbolos terminales***,
- Σ_N es un conjunto de variables o símbolos auxiliares llamado ***alfabeto de símbolos no terminales***,
- $S \in \Sigma_N$ es un símbolo no terminal distinguido denominado ***axioma*** o ***símbolo inicial*** de la gramática, y
- **P** es un ***conjunto de producciones*** de la forma $\alpha:=\beta$ donde ambas palabras están compuestas de símbolos terminales y símbolos no terminales, pero en α al menos debe encontrarse un símbolo no terminal.

Este formalismo requiere algunas aclaraciones. En primer lugar, el orden en el que se expresan los cuatro componentes de una gramática no es importante, aunque una vez establecido debe respetarse en todas las definiciones que se hagan de una gramática en particular, para evitar confusiones. Siempre se seguirá en este texto el orden antes indicado (alfabeto de símbolos terminales, alfabeto de símbolos no terminales, axioma y conjunto de producciones) con este fin.

Refiriéndonos ahora a sus componentes, el alfabeto de símbolos terminales es fundamental ya que todas las cadenas descriptas por la gramática estarán construidas, como se indicó, con estos símbolos. Por su lado, los símbolos no terminales son también llamados *auxiliares* ya que sirven para armar las producciones pero nunca aparecerán en las cadenas del lenguaje definido por la gramática. Por ello, se exige que el alfabeto de símbolos no terminales sea disjunto con el de terminales:

$$\Sigma_T \cap \Sigma_N = \varnothing$$

Para un mismo lenguaje puede existir más de una gramática que lo describa, posiblemente con distintos símbolos no terminales, axioma y producciones, pero todas estas descripciones tendrán exactamente los mismos símbolos terminales.

En un lenguaje cuidadosamente diseñado, los símbolos no terminales suelen representar *categorías sintácticas* bien definidas que, en forma conjunta con las producciones, forman la estructura gramatical del lenguaje.

Las producciones del conjunto **P** son las reglas que permiten, a partir de derivaciones desde el símbolo inicial **S** de la gramática, generar las palabras del lenguaje descripto. En el contexto de las gramáticas formales de Chomsky, las producciones siempre deben tener en el lado izquierdo al menos un símbolo no terminal, porque es ese símbolo el que se reescribe o remplaza en general, en una derivación directa.

BNF

La forma de Backus-Naur es una notación para describir, con cierta economía, la sintaxis de los lenguajes formales (en particular, los llamados libres o independientes del contexto) y consiste de las siguientes reglas sencillas:

a) Una producción se escribe como un símbolo no terminal **A** en el lado izquierdo encerrado entre corchetes angulares, el símbolo de producción y una cadena β de terminales y no terminales de cualquier largo como lado derecho:

$$<A> := \beta$$

b) En caso de existir varias producciones con el mismo lado izquierdo $<A>:=\beta_1$, $<A>:=\beta_2$,…, $<A>:=\beta_n$ y distintos lados derechos, se las escribe todas juntas utilizando la barra vertical como separador:

$$<A> := \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

y se la lee como **A** produce β_1 o bien β_2 o bien … β_n. Recuérdese que siguen siendo **n** producciones, solo se las ha escrito en una forma más conveniente.

Siempre que sea posible se utilizará la notación **BNF** (descripta brevemente en el cuadro anterior) para la escritura de las producciones de una gramática y, en caso de no producir confusiones, se omitirán por claridad los corchetes angulares utilizando letras minúsculas **a, b, c, …, z**, dígitos **0, 1, 2, …, 9** y símbolos especiales necesarios como **+, /, *, …** para representar los símbolos terminales, y letras mayúsculas **A, B, C, …, Z** o palabras descriptivas del concepto, para los símbolos no terminales.

Lenguaje generado

Pero ¿en qué forma describe la gramática formal un lenguaje? y ¿cómo se determina que una palabra pertenece o no pertenece al lenguaje descripto? La idea es sencilla y se explica a continuación.

Se tiene en la gramática un símbolo no terminal especial y distinguido, el axioma **S**, y un conjunto de producciones que permiten transformar unas cadenas en otras a través de derivaciones. Si se aplica alguna producción al axioma para transformarlo en una cadena de símbolos terminales y no terminales, y luego al resultado obtenido se le vuelve a aplicar una producción reescribiendo algún no terminal como una cadena de terminales y no terminales y se sigue con este proceso hasta que todos los símbolos que queden sean terminales (que no pueden ser reescritos ya que no pueden existir producciones únicamente con terminales en el lado izquierdo), se obtiene una cadena de terminales como una derivación desde el axioma. Todas las combinaciones y órdenes posibles de aplicación de producciones crearán de esta forma, un conjunto de cadenas de símbolos terminales bien definido.

Dada una gramática formal $\mathbf{G = (\Sigma_T, \Sigma_N, S, P)}$, se llama ***lenguaje formal generado por*** **G** *al conjunto de todas las cadenas de símbolos terminales que puedan derivarse desde el axioma* **S** *utilizando las reglas de producción de* **P**. En símbolos:

$$\mathbf{L(G) = \{ \alpha \in \Sigma_T^* \,/\, S \rightarrow^* \alpha \}}$$

Durante la derivación de una cadena del lenguaje descripto por **G**, se generarán eventualmente cadenas intermedias. Se denomina ***forma sentencial*** o ***metapalabra*** a una cadena de terminales y no terminales $\alpha_i \in (\Sigma_T \cup \Sigma_N)^*$ que puede derivarse desde el axioma de la gramática. La cadena final de terminales a la que se arriba con la derivación desde el axioma, pertenece al lenguaje generado por la gramática y recibe también el nombre de ***sentencia*** o ***palabra generada*** por la gramática.

Ejemplo 2.17

Considérense nuevamente las producciones del Ejemplo 2.16, para crear una gramática basada en ellas que describa los numerales de los números enteros sin signo (recuérdese que **Número** y **Dígito** son, en este caso, considerados como un símbolo, específicamente como símbolos no terminales):

G_{ess} = ({0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, {Número, Dígito}, Número, P_{ess})

donde:

P_{ess} = {Número:=Dígito|Dígito Número, Dígito:=0|1|2|3|4|5|6|7|8|9}

Los números enteros sin signo se construyen utilizando solo los dígitos del cero al nueve, por lo que estos símbolos constituyen el alfabeto de símbolos terminales.

Las producciones del conjunto **P** indican que se necesitan dos símbolos no terminales o auxiliares: **Número** y **Dígito**. Claramente, en ningún número entero sin signo (sentencias), deben aparecer los símbolos **Número** o **Dígito** sino solo los dígitos del cero al nueve, y si lo hacen no son sentencias sino solo formas sentenciales intermedias.

Las producciones para el símbolo no terminal **Número**, dicen que un número entero sin signo es o solo un dígito, o un número al cual se le antepone un dígito; las producciones para **Dígito** dicen que solo se consideran como dígitos los diez símbolos 0, 1, 2, …, 9.

El axioma de esta gramática es el símbolo no terminal **Número**.

Usando la gramática entonces, se pueden derivar:

Número $\rightarrow$ Dígito $\rightarrow$ 0

Número $\rightarrow$ Dígito $\rightarrow$ 1

...

Número $\rightarrow$ Dígito $\rightarrow$ 9

Número $\rightarrow$ Dígito Número $\rightarrow$ Dígito Dígito $\rightarrow$ Dígito 0 $\rightarrow$ 10

Número $\rightarrow$ Dígito Número $\rightarrow$ Dígito Dígito $\rightarrow$ Dígito 1 $\rightarrow$ 11

…

Así puede generarse el numeral de cualquier número entero sin signo y el conjunto de todos ellos constituye el lenguaje generado por la gramática $\mathbf{G_{ess}}$:

$$L(G_{ess}) = \{0, 1, 2, \ldots, 9, 10, 11, \ldots\}$$

Nótese que esta gramática también generará 00, 01,… y otras infinitas palabras con uno o más ceros como prefijo, cadenas que no constituyen la forma usual de escribir un entero no negativo, aunque son válidas en la mayoría de los lenguajes de programación. Se deja como ejercicio, la modificación de esta gramática para evitar que se generen estas formas no usuales de escribir enteros.

Debería notarse en el Ejemplo 2.17, que no es importante cuál es el nombre o la forma de los símbolos no terminales, sino solo que se necesitan dos distintos; esto refuerza las denominaciones de *variables* o *auxiliares* que también reciben los símbolos no terminales. Si por ejemplo se cambian los símbolos no terminales *Número* por **N** y *Dígito* por **D,** y modificamos las producciones en forma acorde:

$$P'_{ess} = \{N := D \mid DN,\ D := 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\}$$

obtenemos otra gramática distinta:

$$G'_{ess} = (\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}, \{N, D\}, N, P'_{ess})$$

que genera exactamente el mismo lenguaje. Aún más, se puede formular un conjunto de producciones distinto, usando por ejemplo tres símbolos no terminales y aun así, seguir generando las mismas cadenas, esto es, el mismo lenguaje:

$$\{N := D \mid A,\ A := DA \mid D,\ D := 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\}$$

Esto brinda, por un lado, flexibilidad y cierta riqueza a la teoría de lenguajes pero, por otro, muestra que gramáticas muy distintas podrían utilizarse para describir exactamente lo mismo.

Equivalencia de gramáticas

Surgen así varias cuestiones de interés: en primer lugar, poder determinar cuándo dos gramáticas distintas sirven para describir el mismo lenguaje y luego, cómo es posible transformar una gramática en otra que genere el mismo lenguaje, pero que presente algunas propiedades deseables prefijadas (menor número de no terminales, menor número de producciones, producciones fácil-

mente utilizables en derivaciones, etc.). Al respecto, se verán luego algunas transformaciones útiles de las gramáticas *independientes del contexto*, con cierto detalle.

Digamos por ahora que *dos gramáticas* $G_1=(\Sigma_{T1}, \Sigma_{N1}, S_1, P_1)$ y $G_2=(\Sigma_{T2}, \Sigma_{N2}, S_2, P_2)$ *son* ***equivalentes*** *si y solo si generan exactamente el mismo lenguaje*. En símbolos:

$$\mathbf{G_1 \equiv G_2 \;\Leftrightarrow\; L(G_1) = L(G_2)}$$

Claramente, para que esto ocurra, los alfabetos de símbolos terminales $\mathbf{\Sigma_{T1}}$ y $\mathbf{\Sigma_{T2}}$ deben ser iguales. Nótese que basada en la igualdad de lenguajes, la equivalencia de gramáticas es reflexiva, simétrica y transitiva, una relación de equivalencia bien definida.

Ejemplo 2.18

Una gramática sencilla para las cadenas de paréntesis apareados, esto es, palabras formadas solamente por paréntesis abiertos y cerrados, en las que a una cantidad de paréntesis abiertos le sigue la misma cantidad de paréntesis cerrados (como se usan en expresiones aritméticas), es $G_{(1)} = (\{(,)\}, \{S, A\}, S, \{S := (A) \mid \lambda, A := (A) \mid \lambda\})$.

Veamos cómo genera cadenas esta gramática:

$S \rightarrow \lambda$ Se aplica la segunda producción.

$S \rightarrow (A) \rightarrow ( )$ Se aplica la primera y luego la cuarta.

$S \rightarrow (A) \rightarrow ((A)) \rightarrow (( ))$ Se aplica la primera producción, luego la tercera y finalmente la cuarta.

$S \rightarrow (A) \rightarrow ((A)) \rightarrow (((A))) \rightarrow ((( )))$

$S \rightarrow (A) \rightarrow ((A)) \rightarrow (((A))) \rightarrow ((((A)))) \rightarrow (((( ))))$

…

$S \rightarrow (A) \rightarrow ((A)) \rightarrow (((A))) \rightarrow \ldots \rightarrow (^n)^n$

Es claro en las últimas derivaciones que el axioma se transforma en una forma sentencial usando la primera producción y luego se aplica la tercera regla de reescritura tantas veces como se quiera para generar capas apareadas de paréntesis abiertos y cerrados. Cuando se quiere cortar el ciclo para obtener la cadena resultante (sentencia), se utiliza la cuarta producción que elimina el no terminal **A** al cambiarlo por la cadena vacía. Pueden eliminarse el símbolo **A** y sus producciones cambiando las reglas de **S** como sigue, para generar exactamente el mismo lenguaje:

$$G_{(2)} = (\{(,)\}, \{S\}, S, \{S := (S) \mid \lambda\})$$

donde se aplicará la primera producción tantas veces como se quiera para generar la cadena de paréntesis apareados deseada y se cortará el ciclo recursivo aplicando finalmente la segunda producción.

Como $\mathbf{L(G_{(1)}) = L(G_{(2)})}$, las gramáticas resultan ser equivalentes aunque los alfabetos de no terminales y las producciones sean distintas.

Jerarquía de Chomsky

Se han llamado *lenguajes formales*, a aquellos que pueden ser generados por gramáticas formales. En su trabajo, Chomsky estableció que todos los lenguajes formales podían clasificarse en cuatro tipos (denominados *lenguajes tipo 0, 1, 2 y 3*) que solo se distinguen por el formato de las producciones de las gramáticas que los generan. Mientras más restricciones se le ponen a las producciones, en cuanto al formato de las cadenas de sus lados derecho e izquierdo, menos lenguajes pueden describir, por lo que la clasificación es inclusiva.

Clasificación de gramáticas y lenguajes formales

Las cuatro gramáticas, y sus respectivos lenguajes, conforman la jerarquía de Chomsky y tienen, además de su número de tipo, nombres especiales que describen características específicas de los conjuntos de cadenas que los constituyen y de los formatos de las producciones que los generan.

TIPO	DENOMINACIÓN
0	Estructurados por frases o recursivamente enumerables.
1	Dependientes del contexto o sensibles al contexto.
2	Independientes del contexto o de contexto libre.
3	Regulares o lineales.

Tabla 2.2: Gramáticas y lenguajes de la Jerarquía de Chomsky.

Como se dijo, las gramáticas mismas toman los nombres y tipo de los lenguajes que generan. Así, los lenguajes tipo 0, 1, 2, 3 se dice que son generados por gramáticas tipo 0, 1, 2 y 3, respectivamente.

Tipo 0: Lenguajes estructurados por frases

También denominados lenguajes *recursivamente enumerables*, los lenguajes tipo 0 son los más generales en la jerarquía de Chomsky y están descriptos por las reglas de reescritura menos restrictivas, por lo que a veces también se dice que son *lenguajes irrestrictos*.

Las producciones pueden contener cualquier cadena de terminales y no terminales tanto en lado izquierdo como en el lado derecho, con al menos un símbolo no terminal en el lado izquierdo. En símbolos:

$$\alpha := \gamma \quad \alpha \in (\Sigma_T \cup \Sigma_N)^* \, \Sigma_N \, (\Sigma_T \cup \Sigma_N)^*, \, \gamma \in (\Sigma_T \cup \Sigma_N)^*$$

Otra forma de describir el hecho de que el lado izquierdo deba tener al menos un símbolo no terminal, es la siguiente:

$$\alpha A \beta := \gamma \qquad \alpha, \beta, \gamma \in (\Sigma_T \cup \Sigma_N)^*, \, A \in \Sigma_N$$

Tipo 1: Lenguajes dependientes del contexto

O *sensibles al contexto*, son lenguajes que permiten el reemplazo *contextual* de símbolos no terminales. Sus producciones tienen la forma:

$$S := \lambda \quad \text{ó} \quad \alpha A \beta := \alpha \gamma \beta \qquad \alpha, \beta \in (\Sigma_T \cup \Sigma_N)^*, \, A \in \Sigma_N, \, \gamma \in (\Sigma_T \cup \Sigma_N)^+$$

Esto dice que el símbolo no terminal **A**, solo puede ser reemplazado por la cadena γ de terminales y no terminales si se encuentra flanqueada por α a la izquierda y por β a la derecha, es decir, en el *contexto alfa-beta*.

Debe notarse que la cadena γ debe por lo menos tener largo unitario (pertenece a la cerradura positiva de la unión de alfabetos de terminales y no terminales), por lo cual en estas reglas siempre la cadena del lado izquierdo es de largo igual o menor que la cadena del lado derecho (*reglas no compresoras*); sin embargo, el lenguaje podría contener como palabra a la cadena vacía, por lo cual ésta debe poder ser generada por la gramática. Por esto, se permite la *regla lambda* **S := λ**, como única regla compresora permitida.

Tipo 2: Lenguajes independientes del contexto

O *de contexto libre*, son los lenguajes sobre los que más esfuerzo e investigación se ha efectuado a la fecha, ya que la sintaxis de la gran mayoría de los lenguajes de programación de computadoras se describe con *gramáticas independientes del contexto*. Sus producciones pueden adoptar las siguientes formas:

$$S := \lambda \quad \text{ó} \quad A := \alpha \qquad A \in \Sigma_N, \, \alpha \in (\Sigma_T \cup \Sigma_N)^+$$

Puede verse que el símbolo no terminal **A**, puede ser reemplazado por la cadena α de terminales y no terminales en cualquier lugar donde aparezca durante el proceso de derivación, *sin tener en cuenta el contexto* donde se encuentra, y de allí el nombre del lenguaje. Nótese que estas reglas son también no compresoras, salvo la regla lambda que tiene idéntica justificación que en el tipo 1.

Tipo 3: Lenguajes regulares o lineales

Son los lenguajes que tienen producciones más restringidas dentro de la jerarquía de Chomsky, pero de ninguna forma son por ello menos útiles. Los elementos de los que se compone un lenguaje de programación (identificadores, constantes, palabras clave, operadores, etc.) conforman lenguajes regulares y pueden ser especificados utilizando *gramáticas regulares* (y expresiones regulares que luego se discutirán); además, éstas describen en forma sencilla importante cantidad de patrones por lo que se las utiliza ampliamente en herramientas de desarrollo de sistemas y para, por ejemplo, las operaciones de administración de los sistemas operativos.

Las producciones de las gramáticas que generan estos lenguajes tienen un solo símbolo no terminal del lado izquierdo (como las *tipo 2* anteriores), pero su lado derecho está compuesto por un solo símbolo terminal, o por un símbolo terminal y un símbolo no terminal, aparte de poder tener la regla lambda.

Este formato de reglas de reescritura puede presentarse de dos formas, totalmente equivalentes:

Regular por derecha: $S:=\lambda$ ó $A:=aB$ ó $A:=a$ $A, B\in\Sigma_N, a\in\Sigma_T$

Regular por izquierda: $S:=\lambda$ ó $A:=Ba$ ó $A:=a$ $A, B\in\Sigma_N, a\in\Sigma_T$

Los formatos derecho e izquierdo no pueden mezclarse en una misma gramática y seguir siendo regular (sería en este caso una gramática de *tipo 2*). Pero puede demostrarse que *para cada formulación de una gramática regular por derecha, existe una gramática regular por izquierda equivalente* (esto se verá en el capítulo 4).

Jerarquía inclusiva

Si se revisa cuidadosamente la estructura de las producciones de cada tipo de gramática, se verá claramente que las gramáticas regulares resultan ser un caso particular de las gramáticas independientes del contexto, éstas un caso particular de las dependientes del contexto y finalmente, todas son casos especiales de las gramáticas irrestrictas *tipo 0*. Los lenguajes generados (o sus gramáticas) mostrados en un diagrama de Venn, deben verse por lo tanto como subconjuntos propios.

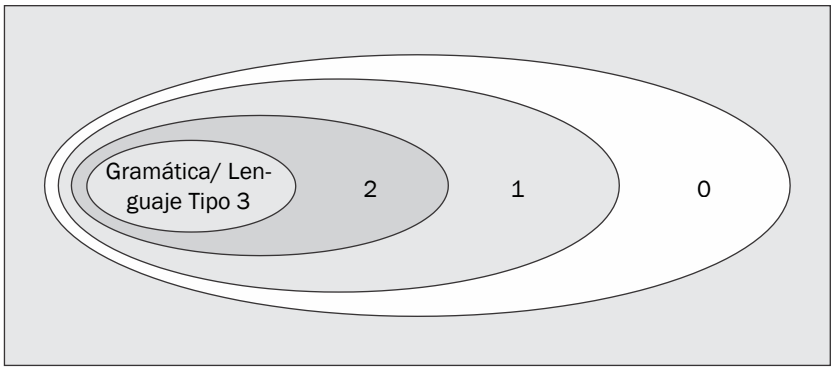


Figura 2.1: Jerarquía de Chomsky: $\mathbf{L_3 \subset L_2 \subset L_1 \subset L_0}$.

Las gramáticas de Chomsky se diferencian entonces, unas de otras, solo por el formato de sus producciones.

Ejemplo 2.19

Si en una gramática existe al menos una producción $\alpha:=\beta$, siendo la cadena del lado izquierdo α de mayor longitud que la del lado derecho β (*regla compresora*), entonces la gramática será **tipo 0** (salvo la excepción de la regla lambda **$S:=\lambda$**, que es siempre permitida). Por ejemplo:

AZBj := ABj

Si por el contrario, todas las producciones son no compresoras (salvo la regla lambda) y en ellas siempre un símbolo no terminal del lado izquierdo puede ser reemplazado por una cadena no vacía del lado derecho (dentro o fuera de un contexto), entonces la gramática es dependiente del contexto o **tipo 1**:

A**B**j := A**ZB**j o A**B**C := A**Z**C

La primera regla dice que el símbolo no terminal **B** puede reescribirse como **ZB** siempre que se encuentre en el contexto **A-j**; la segunda que **B** se puede reemplazar por **Z** en el contexto **A-C**.

Si en una gramática, el lado izquierdo de *todas* las producciones solo tiene un símbolo no terminal, la gramática es **tipo 2** o de contexto libre.

A := AZBj o B := AZB o B := x

Finalmente, si todas las producciones de la gramática tienen un no terminal en el lado izquierdo y solo un terminal o un símbolo terminal y un no terminal en el lado derecho, la gramática será regular o de **tipo 3**:

A := 0B o B := 1A o C:=1

Por ejemplo, las siguientes producciones corresponden a una gramática regular:

TREN := locomotora | TREN vagón

que con **locomotora** y **vagón** como símbolos terminales, y **TREN** como único símbolo no terminal, dan una definición recursiva de tren.

Actividades prácticas

Ejercicios propuestos de cadenas y lenguajes

Dados los alfabetos $\Sigma_1 = \{a, b, c, d\}$, $\Sigma_2 = \{c, d, e\}$, $\Sigma_3 = \{0, 1\}$ y sobre ellos definidas las cadenas: $\alpha = aab$, $\beta = cdee$, $\gamma = eddcc$ y $\delta = aacddd$, se requiere:

Ejercicio 1

Determinar la longitud de las cadenas: $|\boldsymbol{\alpha}|$, $|\boldsymbol{\beta}|$, $|\boldsymbol{\gamma}|$, $|\boldsymbol{\delta}|$.

Ejercicio 2

Realizar las siguientes operaciones con las palabras dadas:

a) α^2

b) $\alpha \,.\, \delta$

c) β^{-1}

d) $(\delta \,.\, \alpha)^{-1}$

e) $[(\beta \,.\, \gamma)^{-1}]^2$

Ejercicio 3

Definir por enumeración los siguientes lenguajes sobre $\boldsymbol{\Sigma_1}$, $\boldsymbol{\Sigma_2}$ y $\boldsymbol{\Sigma_3}$.

a) $L_1 = \{ d^n e^m \;/\; n \geq 0 \;\wedge\; m \geq 2 \}$

b) $L_2 = \{ a^r c\, d^r \;/\; r \geq 1 \}$

c) $L_3 = \{ 0^a 1^{2a} \;/\; a \geq 0 \}$

Ejercicio 4

Determinar los prefijos y sufijos propios de la cadena $\boldsymbol{\delta}$ y aquellas subcadenas que no sean prefijos ni sufijos.

Ejercicio 5

Defina por extensión, mostrando al menos diez cadenas en orden creciente de longitud, los siguientes conjuntos:

a) $(\Sigma_1 \cap \Sigma_2)^+$

b) $(\Sigma_1 \cup \Sigma_2)^*$

c) $(L_1 \circ L_2)$

d) $(L_2 \circ L_1)$

e) L_3^2

Ejercicios propuestos de derivaciones y tipos de gramáticas

Ejercicio 6

Obtener todas las derivaciones posibles de las siguientes gramáticas y determine el tipo de lenguaje generado en cada caso:

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

$P_1 = \{S := 0B \mid 0A1,\ A := 0B \mid 0,\ B := 1\}$

b) $G_2 = (\{c, d\}, \{D, E\}, D, P_2)$

$P_2 = \{D := cE \mid d,\ E := cd\}$

c) $G_3 = (\{0, 1, 2\}, \{P, Q, R, S\}, P, P_3)$

$P_3 = \{P:= 1R \mid 2Q, Q:= 0R \mid 0, R:= 1S \mid 2, S:= 0\}$

d) $G_4 = (\{a, b, c\}, \{P, Q, R\}, P, P_4)$

$P_4 = \{ P:= aQ, Q:= ab \mid caR, R:= c \mid ac \}$

e) $G_5 = (\{a, b, c\}, \{S, A, B\}, S, P_5)$

$P_5 = \{S:= aAb \mid bBa, aAb:= aBb, aBb:= abb \mid acb, bBa:= bca\}$

f) $G_6 = (\{0, 1\}, \{S, A, B, C, D\}, S, P_6)$

$P_6 = \{S:= CA0, A:= 0B, A0:= 00B0 \mid 10, 0B:= 11, C:= 0BD \mid 01,$

$D:= 0\}$

Ejercicio 7

Siendo **S** el axioma, los símbolos terminales representados por letras minúsculas y los no terminales por letras mayúsculas, considere los siguientes conjuntos de producciones como las reglas de alguna gramática. Luego, determine a qué tipo de gramática corresponde según la clasificación de Chomsky, indique las derivaciones sucesivas directas para cada cadena en particular y si se trata de una gramática tipo 2 o tipo 3, construya el árbol de derivación:

a) Considere la cadena **abbc**, con

$P = \{ S := aab \mid aAc, aAc := aBc, aBc := acc \mid abbc \}$

b) Considere la cadena **abcde**, con

$P = \{ S := aB, B:= bcAe, A := d \}$

c) Considere la cadena **zzaazaz**, con

$P = \{ S := zMz, M := zA , zA := zaazB, B := a, A := aza \}$

d) Considere la cadena **xxyyyy**, con

$P = \{ S := xX, X := xY, Y := yY, Y := y \}$

Ejercicio 8

Considerando **G = ({a, b}, {A, S}, S, P)**, establezca si las reglas de producción dadas, corresponden a gramáticas equivalentes para los siguientes casos:

a) $P_1 = \{ S := aS, S := aA, A := ab \}$
$P_2 = \{ S := aS, S := aab \}$

b) $P_1 = \{ S := aS, S := b \}$
$P_2 = \{ S := aSb, S := b \}$

Ejercicio 9

Para cada uno de los siguientes conjuntos de producciones, y considerando la gramática G_i **= ({a, b}, {S, A, B}, S,** P_i**)**, investigar si se obtienen sentencias ambiguas representando los árboles de derivación.

a) $P_1 = \{ S := AB \mid aA, B := AB \mid b, A := a \mid b \}$

b) $P_2 = \{ S := AB, A := aB \mid b \mid ab, B := b \mid a \}$

Ejercicio 10

Establezca si las siguientes gramáticas son recursivas, y en caso afirmativo, indique cuáles son las reglas recursivas y el tipo de recursividad que presentan (directa, en más de un paso, por izquierda, por derecha, mixta):

$G = (\{a, b, c, d\}, \{S, A, B, C, D\}, S, P)$

a) $P = \{ S:= aAb, A := aB \mid a, B := c \}$

b) $P = \{ S := abS \mid aA, A := a \}$

c) P = { S := Sa | aB, A := ad | a, B := b | aA }

d) P = { S := AB | c, A := aC, C := bS, B := aD, D := b }

Ejercicio 11

Dados Σ_T **= {a, b, c}** y Σ_N **= {S, A, B}**, y siendo el axioma **S**, para cada uno de los siguientes conjuntos de producciones, establezca si las gramáticas construidas son equivalentes, si son recursivas y si son ambiguas (en este caso, muestre construyendo árboles de derivación):

a) P = { S := aSA, S := AB, A := b, B := c }

b) P = { S := aA, A := aBb, B := aBb, B := c }

c) P = { S := AB, A := aA, A := a, B := AB, B := b }

d) P = { S := bBa, B := bBa, B := c }

e) P = { S := AB, A := aA, A := b, B := a, B := bB }

Ejercicios resueltos de cadenas y lenguajes

Ejercicio 30

Dados los siguientes alfabetos:

$$\Sigma_1 = \{a, b, c, d, e, f, g\} \qquad \Sigma_2 = \{x \in \mathbb{N} / 1 \leq x < 5\}$$

Determinar si las siguientes cadenas están definidas sobre alguno de los anteriores alfabetos e indique sobre cuál:

α_1 = 1432 α_2 = 5bax α_3 = agfdbc

α_4 = @ab α_5 = 5332 α_6 = badeg

Solución:

$\alpha_1 \in W(\Sigma_2)$ $\alpha_3 \in W(\Sigma_1)$ $\alpha_6 \in W(\Sigma_1)$

Ejercicio 31

Indique cuatro cadenas que pertenezcan al universo de discurso de cada uno de los alfabetos dados a continuación:

a) Σ_1 = {a, b, c, d, e, f, g, h, i, j, k, l, m}

b) $\Sigma_2 = \{x \in \mathbb{N} / 0 \leq x \leq 9\}$

Solución:

Σ_1: α_1 = abcd

α_2 = cama

α_3 = fghi

α_4 = fila

Σ_2: β_1 = 0123

β_2 = 3939

β_3 = 8888

β_4 = 7654

Ejercicio 32

Determine la potencia indicada a cada lenguaje para los siguientes casos:

a) $[L(\Sigma_1)]^2$ siendo: $L(\Sigma_1) = \{pe, sa\}$

b) $[L(\Sigma_2)]^3$ siendo: $L(\Sigma_2) = \{11, 00\}$

Solución:

$L_1^2 = \{ pepe, pesa, sape, sasa \}$

$L_2^3 = \{111111, 111100, 110011, 110000, 001111, 001100, 000011, 000000\}$

Ejercicio 33

Sean los alfabetos $\Sigma_A = \{a, b, c\}$, $\Sigma_B = \{b, c, d\}$ y los lenguajes:

$L_1 = \{a^i b^m / i \geq 1 \wedge m \geq 1\}$

$L_2 = \{b^i c^r / i \geq 0 \wedge r \geq 1\}$

$L_3 = \{a^i b^n c^i d^n / i \geq 1 \wedge n \geq 2\}$" , siendo i, m, n y r enteros

Establezca si las siguientes afirmaciones se cumplen o no:

a) L_1 es un lenguaje sobre el alfabeto Σ_A

b) L_1 es un lenguaje sobre el alfabeto Σ_B

c) L_2 es un lenguaje sobre el alfabeto $\Sigma_A \cup \Sigma_B$

d) L_2 es un lenguaje sobre el alfabeto $\Sigma_A \cap \Sigma_B$

e) L_3 es un lenguaje sobre el alfabeto $\Sigma_A \cup \Sigma_B$

f) L_3 es un lenguaje sobre el alfabeto $\Sigma_A \cap \Sigma_B$

g) $L_1 \cup L_2$ es un lenguaje sobre el alfabeto Σ_A

h) $L_1 \cup L_2$ es un lenguaje sobre el alfabeto $\Sigma_A \cup \Sigma_B$

i) $L_1 \cup L_2$ es un lenguaje sobre el alfabeto $\Sigma_A \cap \Sigma_B$

j) $L_1 \cup L_2$ es un lenguaje sobre el alfabeto Σ_B

Solución:

$L_1 = \{ ab, aab, abb, aaabb, ...\}$

$L_2 = \{ c, cc, bbccc, bccc, bbbbc, ...\}$

$L_3 = \{ abbcdd, aabbbccddd, ...\}$

a	b	c	d	e	f	g	h	i	j
SI	NO	SI	SI	SI	NO	SI	SI	NO	NO

Ejercicios resueltos de derivaciones y tipos de gramáticas

Ejercicio 34

Dados los siguientes conjuntos de producciones, donde se supone que las minúsculas son símbolos terminales, las mayúsculas son símbolos no terminales y **S** el axioma:

$P_1 = \{ S := AB, A := aA , A := a, B := b, B := bB \}$

$P_2 = \{ S := aA, A := bA, A := bC, C := cC, C := c \}$

$P_3 = \{ S := cA, A := Aa, A := a, B := Bb, B := b, A := B \}$

$P_4 = \{ S := aaA, A := aa, A := aaA, A := B, B := b, B := bB \}$

Se pide para cada uno de ellos:

a) Generar al menos cuatro cadenas de símbolos terminales, mediante el proceso de derivación desde el axioma.

b) Establecer la fórmula algebraica del lenguaje que conforman las cadenas obtenidas.

c) Determinar los componentes formales de la gramática.

Solución:

P_1

C_1 = { ab, aab, abb, aabb, aaaab, abbbb, ... }

a) $L_1 = \{ a^n b^p \} / n \geq 1, p \geq 1 \}$

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

P_2

a) C_2 = { abc, abbcc, abcc, abbc, ... }

b) $L_2 = \{ a\ b^p\ c^r \} / \ p \geq 1, r \geq 1 \}$

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

P_3

a) C_3 = { ca, caa, cb, cab, cbb, cbbb, ... }

b) $L_3 = \{ c\ a^p\ b^r \ / p \geq 0, r \geq 0 \}$

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

P_4

a) C_4 = { aaaa, aaaaaa, aab, aabb, ... }

b) $L_4 = \{ a^{2n} b^p \} / (n \geq 2 \wedge p = 0) \vee (n \geq 1 \wedge p \geq 1) \}$

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

Ejercicio 35

Por cada grupo de producciones, realizar todas las derivaciones posibles e indicar si hay cadenas ambiguas:

$$G = (\Sigma_T = \{a, b, c\}, \Sigma_N = \{A, B, C, S\}, S, P_i),$$

Los conjuntos P_i de producciones se muestran en la siguiente tabla:

1) S:= aB | bA A:= aB | a B:=b	2) S:= aA A:= bC C:= ca |c	3) S:= caBA A:= aC | b B:= bA | a C:= ab | c
4) S:= BAa A:= Ca | a B:= Ab | c C:= ab	5) S:= bAB | c A:= aC C:= bD B:= aD D:= b	6) S:= aaA A:= aa | aaB | cB B:= b | bC C:= c | λ

Solución:

W_1 = { ab, ba, bab }

W_2 = { abca, abc }

W_3 = { cabacb, cabaabb, cabbb, caab, cabaabac, cabacac,
cabbac, caaac, caaaab, cabbaab, cabacaab, cabaabaab }

W_4 = {abababaa, ababaa (cadena ambigua), abaa, caa, cabaa}

W_5 = { c, babbab }

W_6 = { aaaa, aaaab (cadena ambigua), aaaabc,
aacb (cadena ambigua), aacbc }

Ejercicio 36

Identificar en el ejercicio anterior la clasificación de Chomsky que le corresponde a cada uno de los conjuntos de producciones dadas.

Solución:

1-TIPO 3	2- TIPO 2
3-TIPO 2	4- TIPO 2
5-TIPO 2	6- TIPO 0

Ejercicio 37

Dada la siguiente gramática:

G = ({a, b}, {A, B, S}, S, P), con

P = { S := aB | bA, B := bS | b, A := aS | a }

Indique si las siguientes cadenas pueden ser generadas o no por esta gramática (demostrarlo mediante la derivación):

α = abbaaab, β = baabbaaaabb, γ = baabab, δ = abbaab

Solución:

S → aB → abS → abbA → abbaS → abbaaB → no tiene solución, solo por **B** obtengo **b**. Luego α no puede ser generada por la gramática.

S → bA → baS → baaB → baabS → baabbA → baabbaS → baabbaaB → baabbaabS → no tiene solución, porque por **S** no se llega a una derivación en **b**. Luego β no puede ser generada por la gramática.

S → bA → baS → baaB → baabS → baabaB → baabab. La cadena γ sí puede ser generada por la gramática.

$S \rightarrow aB \rightarrow abS \rightarrow abbA \rightarrow abbaS \rightarrow abbaaB \rightarrow abbaab$. La cadena δ sí puede ser generada por la gramática.

Ejercicio 38

Para cada una de las siguientes gramáticas, determine la definición algebraica del lenguaje generado y muestre los árboles de derivación correspondientes a algunas de las cadenas de ejemplo investigadas:

$G_1 = (\{x, y, z\}, \{S, X, Y, Z\}, S, P_1)$ con

$P_1 = \{ S:= XSZ \mid Y \mid yY, X:= x, Y:= yY \mid y, Z:= z \}$

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

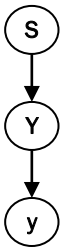
$P_2 = \{ S:= xSz \mid \lambda \mid y \}$

$G_3 = (\{x, y\}, \{M, S, Y\}, S, P_3)$ con

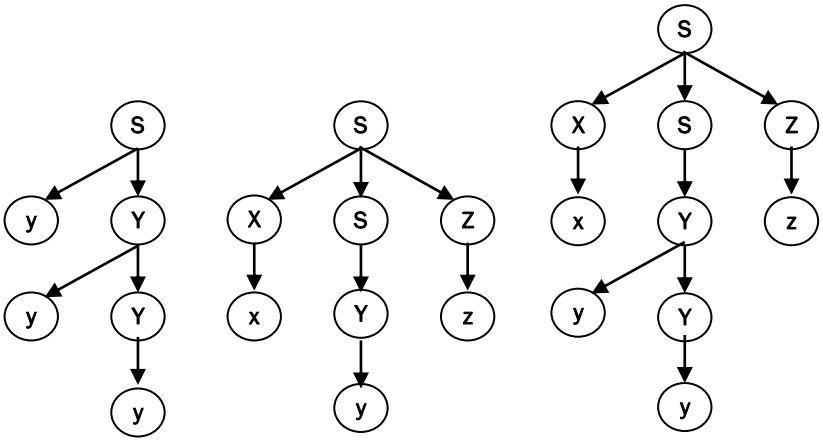
$P_3 = \{ S:= xM \mid \lambda, M:= SY, Y:= y \}$

Solución:

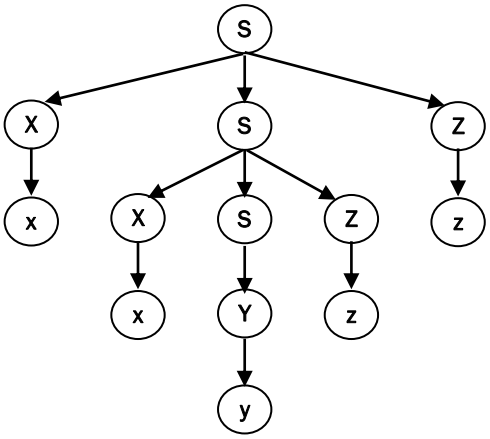
G_1, P_1: Se presenta recursividad en esta gramática.



Cadena: y



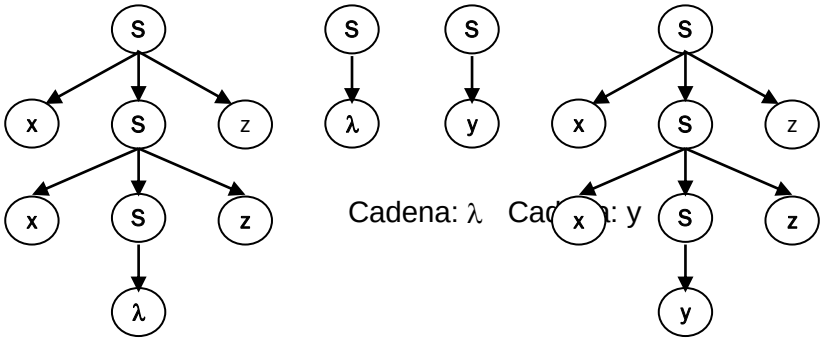
Cadena: yyy Cadena: xyz Cadena: xyyz



Cadena: xxyzz

$$L_1 = \{ x^n y^r z^n \;/\; n \geq 0 \wedge r \geq 1 \}$$

G_2, P_2: Se presenta recursividad en esta gramática.

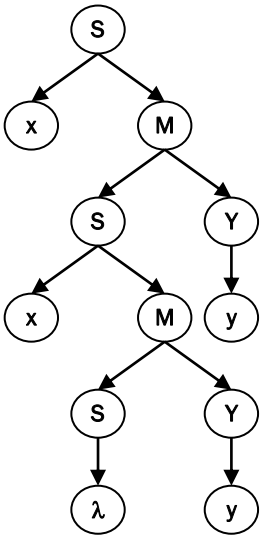


Cadena: λ Cadena: y

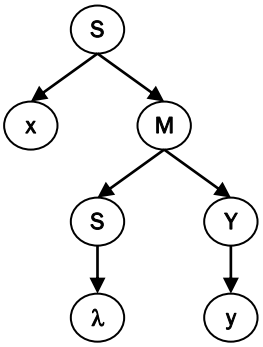
Cadena: xxzz Cadena: xxyzz

$$L_2 = \{ x^n y^p z^n \;/\; n \geq 1 \wedge (p = 0 \vee p = 1) \} \cup \{ \lambda, y \}$$

G_3, P_3: Se presenta recursividad en esta gramática.



Cadena: xxyy



Cadena: xy

$$L_3 = \{ x^n y^n / n \geq 0 \}$$

Ejercicio 39

Las siguientes reglas de producción son algunas de las reglas del lenguaje Java (los terminales y no terminales se pueden distinguir sabiendo que estos últimos figuran entre paréntesis):

a) (WhileDeclaración) := while ((Expresión)) (Declaración)
b) (Expresión) := (ExpresiónCondicional)|(Asignación)
c) (Asignación) := (LadoIzquierdo)(OperadorAsignación)(ExpresiónAsignada)
d) (Declaración) := (WhileDeclaración) | (IfThenDeclaración) | (Bloque) |…
e) (Bloque) := (ExpresiónIncremental) | (ExpresiónDecremental) | …
f) (LadoIzquierdo) := IdentificadorJava
g) (OperadorAsignación) := = | > =| < = | = = | ++ |…
h) (ExpresiónIncremental) := IdentificadorJava ++
i) (ExpresionAsignada) := (Dígitos)
j) (Dígitos) := (Dígito) | (Dígito) (Dígitos)
k) (Dígito) := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Mediante derivaciones sucesivas, obtenga el siguiente bloque de programa:

```
while  (IdentificadorJava <= 10)
        IdentificadorJava ++
```

Solución:

(WhileDeclaración) → while ((Expresión)) (Declaración)

→ while ((Asignación)) (Declaración)

→ while ((LadoIzquierdo)(OperadorAsignación)(ExpresiónAsignada)) (Declaración)

→ while (IdentificadorJava(OperadorAsignación)(ExpresiónAsignada))(Declaración)

→ while (IdentificadorJava <= (ExpresiónAsignada))(Declaración)

→ while (IdentificadorJava <= (Dígitos))(Declaración)

→ while (IdentificadorJava <= (Dígito) (Dígitos))(Declaración)

→ while (IdentificadorJava <= 1 (Dígitos)) (Declaración)

→ while (IdentificadorJava <= 1 (Dígito)) (Declaración)

→ while (IdentificadorJava <= 10) (Declaración)

→ while (IdentificadorJava <= 10) (Bloque)

→ while (IdentificadorJava <= 10) (ExpresiónIncremental)

→ while (IdentificadorJava <= 10) IdentificadorJava ++

Nótese que se trata de derivaciones sucesivas. Todas ellas deberían haber sido expresadas una a continuación de otra, pero por razones de espacio y claridad, se efectuaron en líneas de texto separadas.

Ejercicio 40

Considerando Σ_T **= {a, b, c},** Σ_N **= {S, A, B}** y axioma **S**, establecer si las reglas de producción dadas a continuación, corresponden a gramáticas equivalentes. Justifique su respuesta:

Caso 1: P_1 = { S:= aSA, S:= aB, A:= b, B:= c }

P_2 = { S:= aA, A:= aBb, B:= aBb, B:= c }

Caso 2: P_1 = { S:= aAb, A:= aAb, A:= c }

P_2 = { S:= bBa, B:= bBa, B:= c }

Solución:

Caso 1: G_1 y G_2 no son equivalentes ya que la primera genera la cadena **ac** y la segunda no puede hacerlo.

Caso 2: G_1 y G_2 no son equivalentes ya que todas las cadenas de la primera inician con el símbolo **a** y las de la segunda con **b**.

Ejercicio 41

Considere la siguiente gramática independiente del contexto:

G = ({a, b, c, d}; {S, A, B, C, D}; S; P)

donde el conjunto de producciones está dado por:

P = {S:=AB|C, A:=aAb |ab, B:=cBd | cd, C:=aCd | aDd, D:=bDc | bc}.

Esta gramática genera el lenguaje inherentemente ambiguo:

$L = \{a^n b^n c^m d^m / n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n / n \geq 1, m \geq 1\}$ con m y n enteros.

Demuestre que la cadena **aabbccdd** puede construirse con árboles de derivación distintos.

Solución:

$S \rightarrow AB \rightarrow aAbB \rightarrow aabbB \rightarrow aabbcBd \rightarrow aabbccdd$

$S \rightarrow C \rightarrow aCd \rightarrow aaDdd \rightarrow aabDcdd \rightarrow aabbccdd$

Claramente, estas dos derivaciones de la cadena en cuestión generarán árboles de análisis sintáctico distintos (queda a cargo del lector hacer los dibujos de estos árboles para comprobarlo); esto ocurrirá para cualquier cadena en la que los exponentes **n** y **m** sean iguales.