Cuadernos Didácticos

Ingeniería Informática

Lenguajes, Gramáticas, y Autómatas En Procesadores De Lenguaje

Cuaderno Nº 36

Juan Manuel Cueva Lovelle
Raúl Izquierdo Castanedo
Aquilino Adolfo Juan Fuente
Mª Cándida Luengo Díez
Francisco Ortín Soler
José Emilio Labra Gayo

Profesores de la Universidad de Oviedo

Oviedo, Noviembre 2003

Cuadernos Didácticos
Ingeniería Informática

Cuaderno Nº 36

Lenguajes, Gramáticas y Autómatas en Procesadores de Lenguaje

Autores:

Juan manuel Cueva Lovelle
Raúl Izquierdo Castanedo
Aquilino Adolfo Juan Fuente
Mª Cándida Luengo Díez
Francisco Ortín Soler
Jose Emilio Labra Gayo

Universidad de Oviedo - España

Editorial:

SERVITEC

ISBN: 84-688-4211-7

1ª Edición : Oviedo, Noviembre 2003

Consultor Editorial
Juan Manuel Cueva Lovelle
cueva@lsi.uniovi.es

CUADERNO DIDÁCTICO Nº 36

LENGUAJES GRAMÁTICAS Y AUTÓMATAS EN PROCESADORES DE LENGUAJE

Juan Manuel Cueva Lovelle
Raúl Izquierdo Castanedo
Aquilino Adolfo Juan Fuente
Mª Cándida Luengo Díez
Francisco Ortín Soler
José Emilio Labra Gayo
Profesores de Lenguajes y Sistemas Informáticos
Universidad de Oviedo

Oviedo, Noviembre 2003

TABLA DE CONTENIDOS

TABLA DE FIGURAS

CAPÍTULO 1: INTRODUCCIÓN

El objetivo de este libro de texto es introducir los conceptos teóricos necesarios sobre Teoría de Lenguajes Formales, Gramáticas y Autómatas para un curso universitario de Traductores, Procesadores, Compiladores e Intérpretes de lenguajes de programación.

En este texto se presenta la Teoría de Gramáticas y Lenguajes Formales, como una herramienta matemática que permite abordar con rigor el diseño de lenguajes de programación. Además se desarrollan los conceptos necesarios para la construcción de Autómatas para el reconocimiento de lenguajes de programación.

La Teoría de los Lenguajes Formales tiene su origen en un campo aparentemente bastante alejado de la Informática: la Lingüistica.

Los lingüistas de la llamada *escuela estructuralista americana* habían elaborado por los años 50 algunas ideas informales acerca de la gramática universal. Se entiende por gramática universal, una gramática que caracteriza las propiedades generales de cualquier lenguaje humano.

El primer trabajo que desarrolló teorías formales sobre gramáticas y lenguajes fue obra de *Avram Noam Chomsky (1928-),* quien es sin duda la figura más destacada de la lingüistica moderna, tanto por desarrollar sus fundamentos matemáticos, como por sus teorías sobre el origen y la naturaleza de los lenguajes naturales, aunque éstas últimas son más discutidas (*Chomsky, 1956; 1959; 1962; y 1963).*

En el campo de la Informática, poco después de las primeras publicaciones de *Chomsky,* el concepto de Gramática Formal adquirió gran importancia para la especificación de lenguajes de programación; concretamente, se definió con sus teorías la sintaxis del lenguaje ALGOL 60 (con ligeras modificaciones sobre su versión primitiva), usándose una gramática libre de contexto. Ello condujo rápidamente al diseño riguroso de algoritmos de traducción y compilación.

Finalmente, y enlazando con el campo de la lingüistica, la Teoría de Lenguajes Formales es de gran utilidad para el trabajo en otros campos de la Informática por ejemplo en Informática Teórica, Inteligencia Artificial, Procesamiento de lenguajes naturales (comprensión, generación, y traducción) y Reconocimiento del Habla.

La Teoría de los Lenguajes y Gramáticas Formales tiene una relación directa con la Teoría de Autómatas, siendo posible establecer entre ambas una correspondencia denominada en Algebra isomorfismo.

La Teoría de los Autómatas proviene del campo de la Ingeniería Eléctrica. El científico estadounidense *Claude Elwood Shannon (1916-2001)*, publicó varios trabajos, donde mostraba las bases para la aplicación de la Lógica Matemática a los circuitos combinatorios y secuenciales. A lo largo de las décadas siguientes, las ideas de *Shannon* se desarrollaron considerablemente, dando lugar a la Teoría de Autómatas (*Shannon 1949; 1954 y 1956*).

Los autómatas son sistemas que reciben información, la transforman y producen otra información que se transmite al entorno.

La Teoría de Autómatas tiene aplicación en campos muy diversos :

- Lógica de los Circuitos Secuenciales
- Teoría de Control de Sistemas
- Teoría de la Comunicación
- Arquitectura de Ordenadores
- Redes Conmutadoras y Codificadoras
- Teoría de los Sistemas Evolutivos y Auto-reproductivos
- Reconocimiento de patrones
- Redes Neuronales
- Reconocimiento y procesado de lenguajes de programación
- Traducción de lenguajes
- Teoría de Lenguajes Formales

En este texto la Teoría de Autómatas se aplicará principalmente los tres últimos campos enumerados. Dentro del campo de los Traductores, Procesadores, Compiladores e Intérpretes se aplicarán los lenguajes, gramáticas y autómatas de tipo 3 para la construcción de analizadores léxicos, y los de tipo 2 para la construcción de analizadores sintácticos.

CAPÍTULO 2: DEFINICIONES PREVIAS

En este capítulo se introducen un conjunto de definiciones elementales necesarias para los desarrollos teóricos posteriores. Se definen de forma intuitiva, especificándose algunas de ellas de manera formal en capítulos posteriores.

2.1 Símbolo

Es una entidad abstracta, que no se va a definir, pues se dejará como axioma. Al igual que no se define punto en Geometría. Normalmente los símbolos son letras (a, b, c, . . . ,z), dígitos (0, 1, . . ., 9), y otros caracteres (+, -, *, /, ?, . . .). Los símbolos también pueden estar formados por varias letras o caracteres, así por ejemplo las palabras reservadas de un lenguaje de programación son símbolos de dicho lenguaje.

2.1.1 Ejemplos

a , b , c , # , 0 , 1 , + , * ,**then**, **begin**, **end**, **else**

2.2 Vocabulario o alfabeto

Es un conjunto finito de símbolos, no vacio. Para definir que un símbolo *a* pertenece a un alfabeto *V* se utiliza la notación $a \in V$. Los alfabetos se definen por enumeración de los símbolos que contienen, así por ejemplo se presentan a continuación varios alfabetos.

Ejemplos 2.2.1

$V_1 = \{ A , B , C , D , E , F , G , H , \ldots , X , Y , Z \}$
$V_2 = \{ a , b , c , d , 0 , 1 , 2 , 3 , 4 , * , # , + \}$
$V_3 = \{ 0 , 1 \}$
$V_4 = \{$**if**, **then**, **begin**, **end**, **else**, *a, b, ; , =, >* $\}$

También se puede definir las tablas ASCII y EBCDIC como los alfabetos de distintos ordenadores.

2.3 Cadena

Una cadena es una secuencia finita de símbolos de un determinado alfabeto.

Ejemplos 2.3.1

Se utilizan los vocabularios de los ejemplos del epígrafe 2.2.1.

abcb es una cadena del alfabeto V_2
*a+2*b* es una cadena del alfabeto V_2
000111 es una cadena del alfabeto V_3
if *a>b* **then** *b=a;* es una cadena del alfabeto V_4

2.4 Longitud de cadena

La longitud de una cadena es el número de símbolos que contiene. La notación empleada es la que se indica en los siguientes ejemplos.

Ejemplos 2.4.1

Se utilizan las cadenas de los ejemplos del epígrafe 2.3.1.

$| abcb |\rightarrow 4$
$| a + 2*b |\rightarrow 5$
$| 000111 |\rightarrow 6$
$|$ **if** $a > b$ **then** $a = b$; $|\rightarrow 9$

2.5 Cadena vacía

Existe una cadena denominada *cadena vacía*, que no tiene símbolos y se denota con λ , entonces su longitud es :

$$| \lambda |\rightarrow 0$$

2.6 Concatenación de cadenas

Sean α y β dos cadenas cualesquiera, se denomina concatenación de α y β a una nueva cadena $\alpha\beta$ constituida por los símbolos de la cadena α seguidos por los de la cadena β.

El elemento neutro de la concatenación es λ :

$$\alpha\lambda = \lambda\alpha = \alpha$$

2.7 Universo del discurso

El conjunto de todas las cadenas que se pueden formar con los símbolos de un alfabeto V se denomina *universo del discurso* de V y se representa por W(V). Evidentemente W(V) es un conjunto infinito. La cadena vacia pertenece a W(V).

Ejemplo 2.7.1

Sea un alfabeto con una sola letra V = { *a* }, entonces el universo del discurso es :

$$W(V) = \{\lambda , a, aa, aaa, aaaa, \ldots \}$$

que contiene infinitas cadenas.

2.8 Lenguaje

Se denomina lenguaje sobre un alfabeto V a un subconjunto del universo del discurso. También se puede definir como un conjunto de palabras de un determinado alfabeto.

Alguien puede pensar que los lenguajes se pueden definir porenumeración de las cadenas que pertenecen a dicho lenguaje, pero este método además de ineficiente, es en muchos casos imposible (habitualmente un lenguaje tiene infinitas cadenas). Así los lenguajes se defienen por las propiedades que cumplen las cadenas del lenguaje.

Ejemplo 2.8.1

El conjunto de *palíndromos* (cadenas que se leen igual hacia adelante, que hacia atrás) sobre el alfabeto {*0,1*}. Evidentemente este lenguaje tiene infinitas cadenas.

Algunas cadenas de este lenguaje son:

λ
0
1
00
11
010
0110
000000
101101
111111

100001
001100
1101011
0010100

2.9 Lenguaje vacio

Existe un lenguaje denominado el *lenguaje vacío*, que es un conjunto vacío y que se denota por {∅}. El lenguaje vacío no debe confundirse con un lenguaje que contenga una sola cadena, y que ésta sea la cadena vacia, es decir {λ}, ya que el número de elementos (cardinalidad) de estos dos conjuntos es diferente.

Cardinal ({∅}) = 0
Cardinal ({λ}) = 1

2.10 Gramática

La gramática es un ente formal para especificar, de una manera finita, el conjunto de cadenas de símbolos que constituyen un lenguaje.

2.11 Autómata

Un autómata es una construcción lógica que recibe una entrada y produce una salida en función de todo lo recibido hasta ese instante.

En el caso de los *Procesadores de Lenguaje* un autómata es una construcción lógica que recibe como entrada una cadena de símbolos y produce una salida indicando si dicha cadena pertenece o no a un determinado lenguaje.

CAPÍTULO 3: DEFINICIÓN FORMAL DE GRAMÁTICA

Una gramática es una cuádrupla :

$$G = (VT , VN , S , P)$$

donde :

VT = {conjunto finito de símbolos terminales}
VN = {conjunto finito de símbolos no terminales}
S es el *símbolo inicial* y pertenece a VN.
P = {conjunto de producciones o de reglas de derivación}

Todas las cadenas del lenguaje definido por la gramática están formados con símbolos del *vocabulario terminal* VT. El vocabulario terminal se define por enumeración de los símbolos terminales.

El *vocabulario no terminal* VN es el conjunto de símbolos introducidos como elementos auxiliares para la definición de la gramática, y que no figuran en las sentencias del lenguaje. El vocabulario no terminal se define por enumeración de los símbolos no terminales.

La intersección entre el vocabulario terminal y no terminal es el conjunto vacio :

$$\{VN\} \cap \{VT\} = \{\varnothing\}$$

La unión entre el vocabulario terminal y no terminal es el *vocabulario* :

$$\{VN\} \cup \{VT\} = \{V\}$$

En ocasiones es importante distinguir si un determinado vocabulario incluye o no la cadena vacia, indicándose respectivamente con superíndice + o superíndice *, tal como se muestra a continuación :

$$V^{+} = V - \{\lambda\}$$

$$V^{*} = V + \{\lambda\}$$

El *símbolo inicial* S es un símbolo no terminal a partir del cual se aplican las reglas de la gramática para obtener las distintas cadenas del lenguaje.

Las *producciones* P son las reglas que se aplican desde el símbolo inicial para obtener las cadenas del lenguaje. El conjunto de producciones P se define por medio de la enumeración de las distintas producciones, en forma de reglas o por medio de un metalenguaje por ejemplo BNF (*Backus Naur Form*) o EBNF (*Extended Backus Naur Form*).

Ejemplo 3.1

Sea la gramática : G = (VT, VN, *S*, P) donde VT = {*a*, *b*}, VN = {*S*}, y el conjunto de producciones es :

$S \rightarrow ab$
$S \rightarrow aSb$

Ejemplo 3.2

Sea la gramática G = ({*a, b, c, d*} , {*S, A, B*} , *S*, P) donde P son las producciones :

$S \rightarrow ASB$
$A \rightarrow b$
$aaA \rightarrow aaBB$
$S \rightarrow d$
$A \rightarrow aA$
$B \rightarrow dcd$

Ejemplo 3.3

Sea la gramática G = (VN, VT, S, P) donde :

VN = { <número> , <dígito> }
VT = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
S = <número>

Las reglas de producción P son :

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

Ejemplo 3.4

Sea la gramática G = (VN, VT, S, P) donde :

VN = { <dígito> , <otroDígito> , <Base2>, <vacio>}
VT = { 0, 1 }
S = <Base2>

Las reglas de producción P son :

<Base2> ::= <dígito> <otroDígito>
<otroDígito> ::= <dígito> <otroDígito> | <vacio>
<dígito> ::= 0 | 1
<vacio> ::=

3.5 Notación

Se usará la que se describe a continuación, por ser la más extendida en la bibliografía *Aho y Ullman (1973a, 1973b), Hopcroft y Ullman (1979), Aho et al. (1986), Sanchís y Morales (1986), Alfonseca et al. (1987), y Sánchez y Valverde (1989).*

3.5.1 Vocabulario terminal

Los elementos del vocabulario terminal se representan por :

- letras minúsculas de comienzo del abecedario : *a, b, c, . . . , g.*
- operadores tales como : + , - , * , / , . . .
- caracteres especiales : # , @ , (,) , . , ; , . . .
- los dígitos : 0, 1, . . . , 9
- las palabras reservadas de lenguajes de programación con letras minúsculas y en negrita : **if**, **then**, **else**, . . .

3.5.2 Vocabulario no terminal

Los elementos del vocabulario no terminal se representan por :

- letras mayúsculas de comienzo del abecedario : A, B, . . . , G. La única excepción suele ser el símbolo inicial que se representa con S.
- nombres en minúscula, pero encerrados entre paréntesis angulares : <expresión>, <operador>, . . .

3.5.3 Vocabulario

Los elementos indiferenciados del vocabulario terminal y no terminal se denotan con :

- las letras mayúsculas del final del abecedario : U, V, W, X, Y, Z.

3.5.4 Cadenas terminales

Las cadenas compuestas totalmente por símbolos terminales se representan como :

- las letras minúsculas del final del abecedario : t, u, v, x, y, z.

3.5.5 Cadenas

Las cadenas que contienen símbolos terminales y no terminales indiferenciados se representan por :

- letras minúsculas griegas : α, β, γ, δ, ε, . . .

CAPÍTULO 4: RELACIONES ENTRE CADENAS

En este capítulo se muestran las relaciones de derivación directa y de derivación entre las cadenas de un determinado lenguaje descrito por una gramática.

4.1 Relacion de derivación directa

Sea una gramática G = (VN,VT,S,P), si $\alpha \rightarrow \beta$ es una producción, es decir

$$(\alpha \rightarrow \beta) \in P$$

y $\gamma \alpha \rho$ es una cadena, es decir $\gamma \alpha \rho \in V^+$, entonces las cadenas $\gamma \alpha \rho$ y $\gamma \beta \rho$ están en la relación de derivación directa de la gramática G, que se puede expresar por:

$$\gamma \alpha \rho \rightarrow \gamma \beta \rho$$

y se puede decir que la cadena $\gamma \beta \rho$ deriva directamente de la $\gamma \alpha \rho$, o bien que $\gamma \alpha \rho$ produce directamente $\gamma \beta \rho$ en la gramática G. De ahí el nombre de producciones para los elementos de P.

Ejemplo 4.1.1

Sea la gramática : G = (VT, VN, *S*, P) donde VT = {*a*, *b*}, VN = {*S*}, y el conjunto de producciones es :

$S \rightarrow ab$
$S \rightarrow aSb$

Se obtiene la siguiente derivación directa, al sustituir la primera regla en la segunda :

$S \rightarrow aabb$

4.2 Relacion de derivación

Sean α_1 y α_m cadenas pertenecientes a V^+, se dice que están en relación de derivación en la gramática G si existen $\alpha_1, \alpha_2, \alpha_3, ..., \alpha_m$ tales que :

$$\alpha_1 \rightarrow \alpha_2$$

$$\alpha_2 \rightarrow \alpha_3$$

$$\alpha_3 \rightarrow \alpha_4$$

$$\ldots$$

$$\alpha_{(m-1)} \rightarrow \alpha_m$$

se escribirá entonces:

$$\alpha_1 \rightarrow \alpha_m$$

diciéndose que α_m deriva de α_1 , o que α_1 produce α_m .

Ejemplo 4.2.1

Sea la gramática G = ({*S, A, B*}, {*a, b, c, d*}, *S*, P) donde P son las siguientes reglas de producción, que en este caso se numeran para su posterior identificación cuando se usen.

(1) $S \rightarrow ASB$
(2) $A \rightarrow b$
(3) $aaA \rightarrow aaBB$
(4) $S \rightarrow d$
(5) $A \rightarrow aA$
(6) $B \rightarrow dcd$

Por aplicación de derivaciones inmediatas a partir del símbolo inicial se obtiene la derivación :

$$S \rightarrow abddcd$$

Las derivaciones inmediatas necesarias para llegar a la derivación anterior se muestran a continuación, indicándose en cada paso el número de la regla aplicada.

$$S \overset{(1)}{\rightarrow} ASB \overset{(5)}{\rightarrow} aASB \overset{(2)}{\rightarrow} abSB \overset{(4)}{\rightarrow} abdB \overset{(6)}{\rightarrow} abddcd$$

CAPÍTULO 5: SENTENCIAS O INSTRUCCIONES

Se denominan sentencias o instrucciones de un lenguaje a cualquier cadena que sea el resultado último de una derivación a partir del símbolo inicial S y que está compuesta únicamente por símbolos terminales, es decir

$$S \rightarrow \alpha_m \qquad \text{y} \qquad \alpha_m \in \text{VT}$$

Ejemplo 5.1

Utilizando la relación de derivación directa del ejemplo 4.1.1, la derivación da lugar a la sentencia :

aabb

Ejemplo 5.2

Continuando con el ejemplo 4.2.1, la derivación da lugar a la sentencia :

abddcd

CAPÍTULO 6: DEFINICIÓN FORMAL DE LENGUAJE

El lenguaje L(G) generado por una gramática G es el conjunto de todas las sentencias que puede generar G. Es decir expresado formalmente :

$$L(G) = \{\eta \in VT^*/S \rightarrow \eta\}$$

Una sentencia pertenece a L(G) si :

- está compuesta de símbolos terminales
- la sentencia puede derivarse del símbolo inicial S aplicando las reglas de producción de la gramática.

6.1 Propiedad

Dos gramáticas son equivalentes si ambas generan el mismo lenguaje.

G_1 y G_2 son equivalentes si $L(G_1)=L(G_2)$

Ejemplo 6.2

Sea la gramática definida por $G_2 = (\{S\}, \{0,1\}, S, P)$ donde $P=\{(S \rightarrow 000S111), (0S1 \rightarrow 01)\}$. Determinar el lenguaje que genera.

Solución : La única forma de generar sentencias es aplicando cualquier nº de veces la primera producción y terminando con la aplicación de la segunda, así se obtiene el lenguaje.

$$S \rightarrow 000S111 \rightarrow 000000S111111 \rightarrow \cdots \rightarrow 0^{(3n-1)}0S11^{(3n-1)} \rightarrow 0^{(3n)}1^{(3n)}$$

Por consiguiente el lenguaje que genera esta gramática es el conjunto infinito de instrucciones que se indica a continuación :

$$L(G_2) = \{0^{(3n)}1^{(3n)}/n \geq 1\}$$

Ejemplo 6.3

Si la 2ª producción de la gramática del ejemplo 6.2 fuese $S \rightarrow 01$ el lenguaje sería :

$$L(G_3) = \{0^{(3n+1)}1^{(3n+1)}/n \geq 0\}$$

Ejemplo 6.4

Sea la gramática $G_4 = (\{S\}, \{a,b\}, S, P)$ donde $P=\{(S \rightarrow aSb), (S \rightarrow ab)\}$. Determinar el lenguaje que genera.

Solución : Aplicando la primera producción n-1 veces, seguida por la aplicación de la segunda producción, se tiene que :

$$S \rightarrow aSb \rightarrow aaSbb \rightarrow a^3Sb^3 \rightarrow \cdots \rightarrow a^{(n-1)}Sb^{(n-1)} \rightarrow a^n b^n$$

El lenguaje generado :

$$L(G_4)=\{a^n\, b^n / n \geq 1\}$$

Ejemplo 6.5

Dada la gramática $G_5 = (\{S,A\}, \{a,b\}, S, P)$ donde $P=\{(S \rightarrow abAS), (abA \rightarrow baab), (S \rightarrow a), (A \rightarrow b)\}$. Determinar el lenguaje que genera.

Solución : Se generan sentencias del lenguaje aplicando las reglas hasta que se pueda ver la forma general del lenguaje.

$S \rightarrow abAS \rightarrow baabS \rightarrow baaba$

$S \rightarrow a$

$S \rightarrow abAS \rightarrow abbS \rightarrow abba$

$S \rightarrow abAS \rightarrow abAabAS \rightarrow \cdots \rightarrow (abA)^nS \rightarrow (abb)^na$

$S \rightarrow abAS \rightarrow abAabAS \rightarrow \cdots \rightarrow (abA)^nS \rightarrow (baab)^na$

$S \rightarrow abAS \rightarrow abAabAS \rightarrow abbbaaba$

$S \rightarrow abAS \rightarrow abAabAS \rightarrow baababba$

$S \rightarrow abAS \rightarrow abAabAS \rightarrow abAabAabAS \rightarrow baababbbaaba$

$L(G_5)$ = {cadenas que contienen *abb* y *baab* intercambiándose y reproduciéndose cualquier número de veces, y terminando siempre con el símbolo *a*}

Se puede observar que la forma de expresar este lenguaje no es simple, y surge la necesidad de tener una herramienta que permita describir los lenguajes de otra forma.

Ejemplo 6.6

Sea la gramática $G_6 = (\{S,A,B\}, \{a,b\}, S, P)$ donde las producciones P son :

$S \rightarrow aB$ $\quad\quad$ $A \rightarrow bAA$

$S \rightarrow bA$ $\quad\quad$ $B \rightarrow b$

$A \rightarrow a$ $\quad\quad$ $B \rightarrow bS$

$A \rightarrow aS$ $\quad\quad$ $B \rightarrow aBB$

Determinar el lenguaje que genera.

Solución : Se generan algunas instrucciones.

$S \rightarrow aB \rightarrow ab$

$S \rightarrow bA \rightarrow ba$

$S \rightarrow aB \rightarrow abS \rightarrow abbA \rightarrow abba$

$S \rightarrow bA \rightarrow bbAA \rightarrow bbaa$

$S \rightarrow aB \rightarrow abS \rightarrow abaB \rightarrow ababS \rightarrow ababaB \rightarrow ababab$

Se puede demostrar (*Hopcroft y Ullman (1979)*, pp. 81-82) que el lenguaje generado es :

$$L(G_6) = \{\text{cadenas que tienen igual nº de } a \text{ que de } b\}$$

La demostración no es inmediata.

Ejemplo 6.7

Sea la gramática G_7 = (VN, VT, S, P) donde :

VN = { <número> , <dígito> }

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

S = <número>

Las reglas de producción P son :

<número> ::= <dígito> <número>

<número> ::= <dígito>

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

Determinar el lenguaje que genera.

Solución : A continuación se muestran algunas sentencias del lenguaje generado por esta gramática.

$< número > \rightarrow < dígito >< número > \rightarrow 7 < número > \rightarrow 72$

$< número > \rightarrow < dígito > \rightarrow 7$

$< número > \rightarrow < dígito > \rightarrow 0$

$< número > \rightarrow < dígito >< número > \rightarrow < dígito >< dígito >< número > \rightarrow \cdots \rightarrow 235$

$L(G_7)$ = {conjunto de los números naturales en base diez}.

Ejemplo 6.8

Sea la gramática $G_8 = (\{A,S\}, \{a,b\}, S, P)$ donde las reglas de producción son :

$S \rightarrow aS$
$S \rightarrow aA$
$A \rightarrow bA$
$A \rightarrow b$

Determinar el lenguaje que genera esta gramática.

Solución : Se muestran algunas sentencias del lenguaje generado por la gramática.

$S \rightarrow aS \rightarrow aaA \rightarrow aab$

$S \rightarrow aA \rightarrow ab$

$S \rightarrow aS \rightarrow aaS \rightarrow aaaS \rightarrow \cdots \rightarrow a^n S \rightarrow a^n aA \rightarrow a^{n+1} b$

$S \rightarrow aA \rightarrow abA \rightarrow abbA \rightarrow abbbA \rightarrow \cdots \rightarrow a\,b^n A \rightarrow a\,b^{n+1}$

El lenguaje generado se puede definir con la siguiente expresión regular, cuya definición se estudiará en el capítulo 9.

$$L(G_8) = a\ a^*\ b\ b^*$$

CAPÍTULO 7: JERARQUÍA DE LAS GRAMÁTICAS

Chomsky definió cuatro tipos distintos de gramáticas en función de la forma de las reglas de derivación P (*Chomsky, 1959*). La clasificación comienza con un tipo de gramáticas que pretende ser universal, aplicando restricciones a sus reglas de derivación se van obteniendo los otros tres tipos de gramáticas. Esta clasificación es jerárquica, es decir cada tipo de gramáticas engloba a todos los tipos siguientes.

7.1 Gramáticas de tipo 0

También llamadas *gramáticas no restringidas* o *gramáticas con estructura de frase*. Las reglas de derivación son de la forma:

$$\alpha \rightarrow \beta$$

siendo $\alpha \in (VN \cup VT)^+$ y $\beta \in (VN \cup VT)^*$, es decir la única restricción es que no puede haber reglas de la forma $\lambda \rightarrow \beta$ donde λ es la cadena vacia.

7.1.1 Ejemplos

Todas las gramáticas mostradas en los ejemplos del capítulo 6 son de tipo 0, pues en ninguna de ellas existe la producción $\lambda \rightarrow \beta$ siendo λ la cadena vacia.

7.2 Gramáticas de tipo 1

Tambien llamadas *gramáticas sensibles al contexto* (en inglés *context sensitive*). En ellas las reglas de producción son de la forma :

$$\alpha A \beta \quad \rightarrow \quad \alpha \gamma \beta$$

siendo $A \in VN$; $\alpha, \beta \in (VN \cup VT)^*$ y $\gamma \in (VN \cup VT)^+$.

Estas gramáticas se llaman sensibles al contexto, pues se puede reemplazar A por γ siempre que estén en el contexto $\alpha \ldots \beta$.

7.2.1 Ejemplos de gramáticas de tipo 1

A continuación se muestran varios ejemplos de gramáticas de tipo 1, que se adaptan a la definición anterior.

Ejemplo 7.2.1.1

La gramática G = ({*S*,*A*,*B*}, {*a*,*b*}, *S*, P) cuyas producciones P se muestran a continuación es de tipo 1.

$S \rightarrow aB$ $\quad\quad$ $A \rightarrow bAA$

$S \rightarrow bA$ $\quad\quad$ $B \rightarrow b$

$A \rightarrow a$ $\quad\quad$ $B \rightarrow bS$

$A \rightarrow aS$ $\quad\quad$ $B \rightarrow aBB$

Ejemplo 7.2.1.2

La gramática G = (VN, VT, S, P) donde VN = { <número> , <dígito> }; VT = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }; S = <número> y las reglas de producción P que se muestran a continuación es de tipo 1.

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

Ejemplo 7.2.1.3

La gramática G = ({*a*,*b*}, {*A*,*S*}, *S*, P) donde P son las producciones que se muestran a continuación es de tipo 1.

$S \rightarrow aS$
$S \rightarrow aA$
$A \rightarrow bA$
$A \rightarrow b$

7.2.2 Ejemplos de gramáticas que No son de tipo 1

A continuación se muestran algunos ejemplos de gramáticas que no son de tipo 1, y que pueden ilustrar mejor la definición de estas gramáticas.

Ejemplo 7.2.2.1

La gramática definida como G = ({*S*}, {*a*,*b*}, *S*, P) donde P son las siguientes producciones :

$S \rightarrow aaaaSbbbb$
$aSb \rightarrow ab$

La producción $aSb \rightarrow ab$ no es del tipo 1, pues se sustituye *S* por vacío en el contexto *a...b*.

Sin embargo si se esta producción fuera $S \rightarrow ab$ o $aSb \rightarrow abb$, entonces sería de tipo 1.

Ejemplo 7.2.2.2

La gramática G = ({*S*,*A*}, {*a*,*b*}, *S*, P) con las producciones P siguientes :

$S \rightarrow abAS$
$abA \rightarrow baab$
$S \rightarrow a$
$A \rightarrow b$

No es del tipo 1, ya que la producción $abA \rightarrow baab$ no es sensible al contexto. Lo sería si fuese $abA \rightarrow abab$.

7.2.3 Propiedades de las gramáticas de tipo 1

A continuación se presenta la propiedad de no decrecimiento de las gramáticas de tipo 1, que se presenta en forma directa e inversa, lo cual permite intercambiar dicha propiedad con la definición dada anteriormente.

7.2.3.1 Propiedad de no decrecimiento

Las cadenas que se obtienen en cualquier derivación de una gramática de tipo 1 son de longitud no decreciente, es decir :

$$\alpha \rightarrow \beta \quad \Rightarrow \quad |\beta| \geq |\alpha|$$

y que se puede enunciar como *la longitud de la parte derecha de la producción es mayor o igual a la de la parte izquierda*.

La demostración es inmediata. Si se define una producción de un lenguaje tipo 1 como :

$$\alpha A \beta \quad \rightarrow \quad \alpha \gamma \beta$$

siendo $\gamma \in (VN \cup VT)^+$, es decir γ nunca puede ser la cadena vacia, lo que implica que $|\gamma| \geq 1$ y como $|A|$ como mínimo vale 1, queda demostrada la propiedad :

$$|\alpha A \beta| \quad \leq \quad |\alpha \gamma \beta|$$

7.2.3.2 Propiedad de sensibilidad al contexto

También se puede demostrar (*Fernández y Sáez Vacas, 1987, pág. 442*) que si todas las reglas de una gramática cumplen la condición de no decrecimiento, se puede hallar una gramática equivalente con las producciones sensibles al contexto. Esta segunda propiedad combinada con la primera hace que se pueda intercambiar la característica de no decrecimiento con la definición.

Ejemplo 7.2.3.3

Sea la gramática G = ({*S,B,C*}, {*a,b,c*}, *S*, P) donde P son las producciones :

$S \rightarrow aSBC$
$S \rightarrow aBC$
$CB \rightarrow BC$
$bB \rightarrow bb$
$bC \rightarrow bc$
$cC \rightarrow cc$
$aB \rightarrow ab$

La gramática anterior no es de tipo 1 según la definición dada, ya que la regla $CB \rightarrow BC$ no respeta el contexto. Sin embargo puede apreciarse que todas las reglas de esta gramática son no decrecientes, por lo tanto es posible encontrar una gramática equivalente que genere el mismo lenguaje. Se puede sustituir la regla $CB \rightarrow BC$ por :

$$CB \rightarrow XB$$
$$XB \rightarrow XY$$
$$XY \rightarrow BY$$
$$BY \rightarrow BC$$

Puede observarse que ambas gramáticas son equivalentes y que generan el lenguaje :

$$L(G) = \{ a^n b^n c^n / n \geq 1 \}$$

7.3 Gramáticas de tipo 2

Las gramáticas de tipo 2 también se denominan *gramáticas de contexto libre* o *libres de contexto* (en inglés *context free*). Sus reglas de producción tan sólo admiten tener un símbolo no terminal en su parte izquierda, es decir son de la forma :

$$A \rightarrow \alpha$$

siendo $A \in VN$ y $\alpha \in (VN \cup VT)^+$.

Si cada regla se representa como un par ordenado (A, α), el conjunto P es un subconjunto del conjunto producto cartesiano $VN \times (\{VN \cup VT\})^+$, es decir :

$$P \subset \{N \times (\{VN\} \cup \{VT\})^+\}$$

La denominación contexto libre se debe a que se puede cambiar A por α, independientemente del contexto en que aparezca A.

Ejemplo 7.3.1

La gramática G = ({*S*,*A*,*B*}, {*a*,*b*}, *S*, P) cuyas producciones P se muestran a continuación es de tipo 2.

$S \rightarrow aB$
$S \rightarrow bA$
$A \rightarrow a$
$A \rightarrow aS$
$A \rightarrow bAA$
$B \rightarrow b$
$B \rightarrow bS$
$B \rightarrow aBB$

Ejemplo 7.3.2

La gramática G = (VN, VT, S, P) donde VN = { <número> , <dígito> }; VT = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }; S = <número> y las reglas de producción P que se muestran a continuación es de tipo 2.

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

Ejemplo 7.3.3

La gramática G = ({*a*,*b*}, {*A*,*S*}, *S*, P) donde P son las producciones que se muestran a continuación es de tipo 2.

$S \rightarrow aS$
$S \rightarrow aA$
$A \rightarrow bA$
$A \rightarrow b$

7.4 Gramáticas de tipo 3

Las gramáticas de tipo 3 tambien denominadas *regulares* o *gramáticas lineales a la derecha* comienzan sus reglas de producción por un símbolo terminal, que puede ser seguido o no por un símbolo no terminal, es decir son de la forma :

$A \rightarrow aB$
$A \rightarrow a$

donde A,B $\in$ VN y $\alpha \in$ VT.

Ejemplo 7.4.1

La gramática G = ({*a*,*b*}, {*A*,*S*}, *S*, P) donde P son las producciones que se muestran a continuación es de tipo 3.

$S \rightarrow aS$
$S \rightarrow aA$
$A \rightarrow bA$
$A \rightarrow b$

7.5 Lenguajes con la cadena vacia

Según las definiciones anteriores la cadena vacia no puede aparecer en ningún lenguaje de tipo 1, 2 o 3. Supongamos que deseamos añadir la cadena vacia a un lenguaje.

Se pretende crear un nuevo lenguaje L', a partir del lenguaje L de tal forma que:

$$L = L' \cup \{\lambda\}$$

Bastará añadir λ de algún modo a la descripción del lenguaje L.

Una forma de hacer esto es añadir la siguiente regla de producción $S \rightarrow \lambda$ a las reglas de la gramática que describe L. Pero se había impuesto a las reglas de las gramáticas de

tipo 1 $\alpha A \beta \rightarrow \alpha \gamma \beta$, con la condición de $\gamma \neq \lambda$, para cumplir la propiedad de no decrecimiento. Si se añade $S \rightarrow \lambda$ será necesario imponer la condición de que no aparezca en la parte derecha S, tal y como se verá en el teorema 7.5.1

Teorema 7.5.1

Si G es una gramática de tipo 1, 2, o 3 puede encontrarse una gramática equivalente G' de tipo 1, 2 o 3 respectivamente, tal que L(G')=L(G) y además su símbolo inicial S', no aparece en el segundo término de ninguna regla de G'. Es decir si $\exists$ G=(VT,VN,P,S) se puede encontrar G'=(VT,VN*,P',S') donde P'=P$\cup\{S' \rightarrow \alpha / (S' \rightarrow \alpha) \in P\}$. La demostración se puede encontrar en las páginas 437-438 del libro *Fundamentos de Informática*, Fernández G. y Sáez Vacas F.).

Corolario 7.5.2

Si L es un lenguaje de tipo 1, 2 o 3 entonces $L \cup \{\lambda\}$ y $L - \{\lambda\}$ son lenguajes de tipo 1, 2, o 3 respectivamente.

Corolario 7.5.3

Dada una gramática G cualquiera de tipo 1, 2 o 3 se puede obtener otra G', con $S' \rightarrow \lambda$ de forma que $L(G') = L(G) \cup \{\lambda\}$.

7.6 Relación de inclusión

Los cuatro tipos de gramáticas estudiados anteriormente (tipo 0, tipo 1, tipo 2, y tipo 3), cada una de ellas tiene restricciones más fuertes que las anteriores. Las gramáticas de tipo 0, contienen a todas las demás. Las de tipo 1 contienen a las de tipo 2 y tipo 3. Y por último las de tipo 2 contienen a las de tipo 3. Es decir una gramática de tipo 3 es de tipo 2, tipo 1 y tipo 0. Por lo tanto se define una *jerarquía de gramáticas respecto de la relación de inclusión*, que se puede representar gráficamente mediante el diagrama de la figura 1.

JERARQUIA DE LAS GRAMATICAS

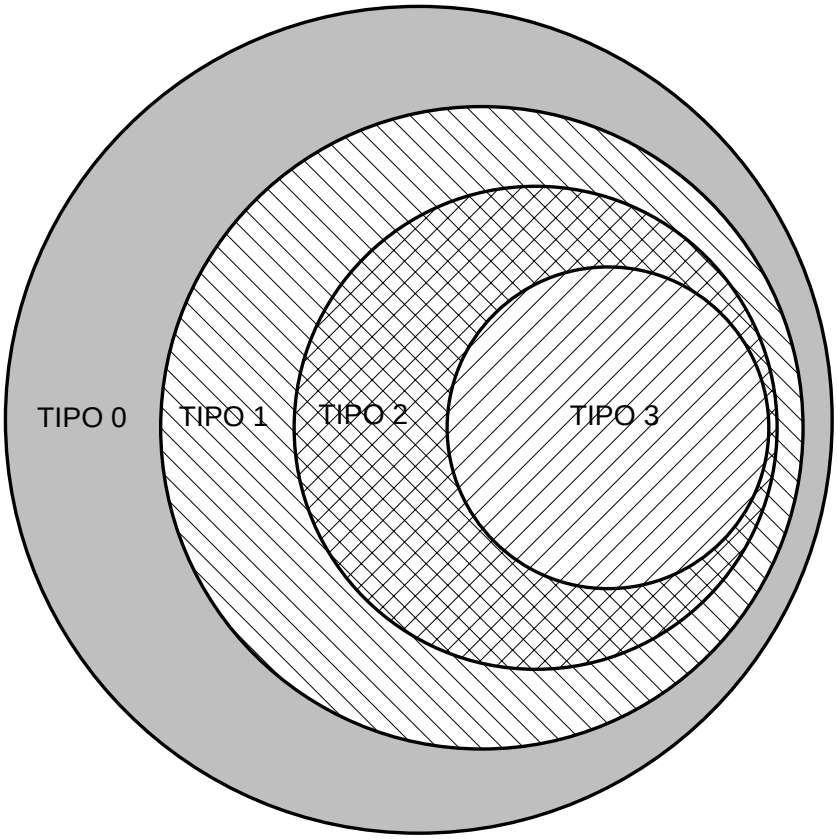


Fig. 1 : Relación de inclusión entre los distintos tipos de gramáticas

CAPÍTULO 8: CORRESPONDENCIA ENTRE GRAMÁTICAS Y LENGUAJES

Se denomina *lenguaje de tipo 0* al generado por una gramática de tipo 0. De la misma forma, se denominan *lenguajes de tipo 1, tipo 2, y tipo 3*, a los generados por las gramáticas de tipo 1, tipo 2, y tipo 3, respectivamente.

Si los lenguajes generados por los distintos tipos de gramáticas se se relacionan entre sí con respecto a la relación de inclusión se obtiene :

$$\{L(G_3)\} \subset \{L(G_2)\} \subset \{L(G_1)\} \subset \{L(G_0)\}$$

Según lo visto anteriormente existe una correspondencia entre las gramáticas y los lenguajes de tal forma que se genera una jerarquía de lenguajes análoga a la mostrada para las gramáticas, que se puede representar gráficamente mediante el diagrama de la figura 2.

CORRESPONDENCIA ENTRE LOS LENGUAJES Y LAS GRAMATICAS

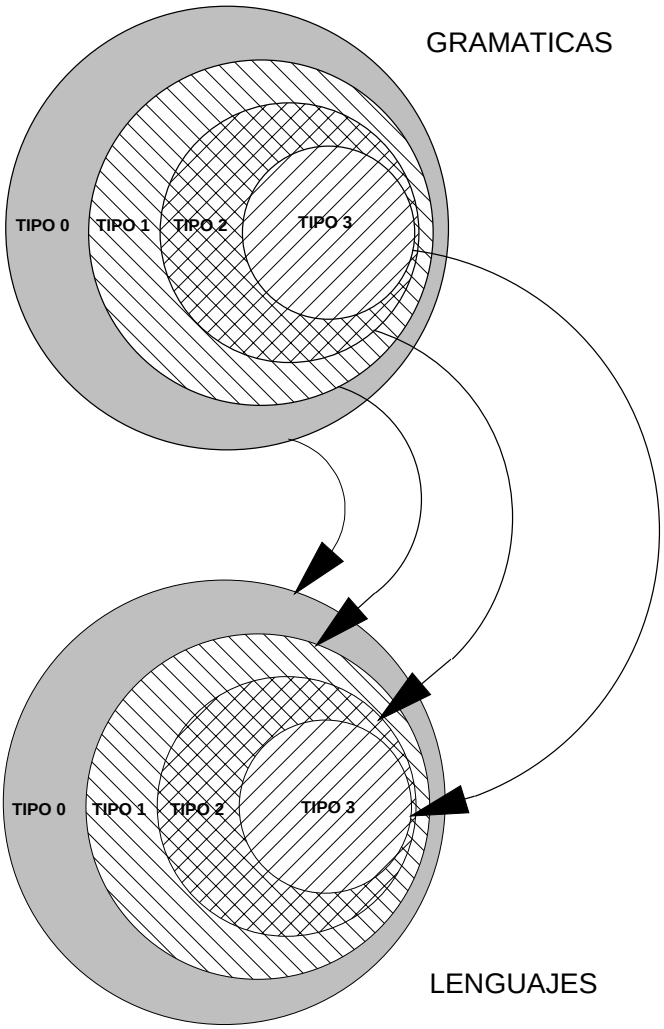


Fig. 2 : Correspondencia entre gramáticas y lenguajes

CAPÍTULO 9: EXPRESIONES REGULARES

Anteriormente se estudiaron los lenguajes formales, y se vio como en algunos ejemplos no era fácil representar los lenguajes de una manera condensada. En este capítulo se va a presentar una herramienta, las expresiones regulares, para describir lenguajes de tipo 3 o lenguajes regulares.

Las expresiones regulares se introducen para describir los lenguajes regulares, entonces las expresiones regulares serán metalenguajes. Es decir *las expresiones regulares son un metalenguaje para describir los lenguajes regulares.*

9.1 Operaciones con los lenguajes regulares

a) **Unión o alternativa** : Sean dos lenguajes definidos sobre un mismo alfabeto, se denomina unión de los dos lenguajes al conjunto formado por las cadenas que pertenezcan indistintamente a uno u otro de los dos lenguajes. Formalmente se puede expresar :

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

b) **Concatenación** : Sean dos lenguajes definidos sobre el mismo alfabeto, se denomina concatenación de los dos lenguajes al conjunto de todas las cadenas formadas concatenando una palabra del primer lenguaje con otra del segundo. Formalmente se puede expresar :

$$L_1 L_2 = \{x_1 x_2 / x_1 \in L_1 \wedge x_2 \in L_2\}$$

c) **Potencia de un lenguaje** : Desde un punto de vista estricto esta no es una nueva operación, sino un caso particular de la anterior. Se denomina potencia i-ésima de un lenguaje a la operación que consiste en concatenarlo consigo mismo i-veces. En el caso de i=0, el resultado es el conjunto vacio.

d) **Cierre u operación estrella** : La operación cierre de un lenguaje L es otro lenguaje L^* obtenido uniendo el lenguaje L con todas sus potencias posibles, incluso L^0. Formalmente se puede expresar como :

$$L^* = \{\varnothing\} \cup \{L\} \cup \{LL\} \cup \{LLL\} \cdots = \bigcup_{n=0}^{\infty} L^n$$

e) **Cierre positivo** : La operación cierre positivo de un lenguaje L es otro lenguaje L^+ obtenido uniendo el lenguaje L con todas sus potencias posibles, excepto L^0. Formalmente se puede expresar como :

$$L^+ = \{L\} \cup \{LL\} \cup \{LLL\} \cdots = \bigcup_{n=1}^{\infty} L^n$$

9.2 Operaciones con las expresiones regulares

Si α es una expresion regular, entonces $\{\alpha\}$ es el conjunto descrito por la expresión regular α. También se puede decir que α denota el lenguaje de la cadena α.

Las expresiones regulares describen los lenguajes regulares, luego sus operaciones corresponderan a las indicadas para los lenguajes regulares.

a) **Unión o alternativa** : Si α y β son expresiones regulares, $\alpha \mid \beta$ es una expresion regular tal que :

$$\{\alpha \mid \beta\} \quad = \quad \{\alpha\} \cup \{\beta\}$$

es decir puede aparecer α o β indistintamente.

b) **Concatenación** : Si α y β son expresiones regulares, $\alpha\,\beta$ es una expresión regular tal que $\{\alpha\,\beta\} = \{\alpha\}\,\{\beta\}$

c) **Cierre u operación estrella** : Si α es una expresión regular, entonces α^* es una expresión regular que denota $\{\ \alpha\ \}^*$. Es decir denota las cadenas :

λ
α
$\alpha\alpha$
$\alpha\alpha\alpha$
$\alpha\alpha\alpha\alpha$
. . .
$\alpha\alpha\alpha\alpha \cdots \alpha$

d) **Cierre positivo** : Si α es una expresión regular, entonces α^+ es una expresión regular que denota $\{\ \alpha\ \}^+$. Es decir denota las cadenas :

α

$\alpha\alpha$

$\alpha\alpha\alpha$

$\alpha\alpha\alpha\alpha$

. . .

$\alpha\alpha\alpha\alpha \cdots \alpha$

9.3 Precedencia de las operaciones

Se permite el uso de paréntesis para indicar la precedencia de las operaciones, pero cuando no se utilizan paréntesis para evaluar una expresión regular, hay que tener en cuenta el siguiente orden de precedencia :

1ª.- Uso de paréntesis
2ª.- Operación cierre y cierre positivo
3ª.- Operación concatenación
4ª.- Alternativa

9.4 Teorema

Dos expresiones regulares son iguales, si designan al mismo conjunto regular.

9.5 Propiedades

A partir del teorema anterior se pueden enunciar las siguientes propiedades :

a) **Asociatividad de la operación concatenación**

$$\alpha(\beta\gamma) = (\alpha\beta)\gamma$$

b) **Distributividad de la operación alternativa respecto de la concatenación**

$$\alpha\beta \mid \alpha\gamma \quad = \quad \alpha(\beta \mid \gamma)$$

c) λ es el **elemento neutro de la concatenación**, es decir

$$\alpha\lambda \quad = \quad \lambda\alpha \quad = \quad \alpha$$

d) **Propiedades de la operación cierre**

d.1) $(\alpha \mid \beta)^* = (\alpha^* \mid \beta^*)^* = (\alpha^* \beta^*)^*$

d.2) $(\alpha \mid \lambda)^* = (\alpha^* \mid \lambda) = \alpha^*$

d.3) $\alpha\alpha^* \mid \lambda = \alpha^*$

d.4) $\lambda^* = \lambda$

Ejemplo 9.6

Sea el vocabulario {*a,b*} y la expresión regular aa^*bb^*. Indicar el lenguaje que denota, y algunas cadenas de dicho lenguaje.

Solución : Algunas cadenas son:

ab
aab
aaaab
abbbb
abb
aaaab

El lenguaje que se describe es L={cadenas que comienzan por una *a* y continuan con varias o ninguna *a*, y siguen con una *b* y continuan con varias o ninguna *b*}

Ejemplo 9.7

Sea el vocabulario {*0,1*}, la expresión regular $1(01)^*$ denota el conjunto de cadenas que empiezan por *1* y van seguidas por *(01)* cualquier nº de veces o ninguna.

Ejemplo 9.8

Sea el vocabulario {*0,1*}, la expresión regular $(0 \mid 1)^+$ denota el conjunto de números en base 2.

Ejemplo 9.9

Sea el vocabulario {*0, 1, 2*}, la expresión regular $(0 \mid 1 \mid 2)^+$ denota el conjunto de números en base 3.

Ejemplo 9.10

Dada la expresión regular $(a \mid b)^*$, el lenguaje que denota es el que se puede formar con todas las cadenas compuestas por *a* y *b* incluida la cadena vacia. Algunos ejemplos de sentencias de este lenguaje son :

λ
aaa
bbb
aba
abaaa
abbaa

Ejemplo 9.11

Sea el vocabulario {*1*,*2*,*3*}, la expresión regular $(1 \mid 2)^*$ 3 indica el conjunto de todas las cadenas formadas con los símbolos *1* y *2*, sucediéndose cualquier nº de veces (y en cualquier orden), y siempre terminando la cadena en el símbolo *3*. Ejemplos de sentencias :

3	*23*
13	*223*
123	*113*
11113	*121211223*
221113	*111212213*

Ejemplo 9.12

Sea el vocabulario {*a*,*b*,*c*}, la expresión regular *a* | *bc* denota el lenguaje formado por las sentencias *a* y *bc*.

Ejemplo 9.13

Sea el vocabulario {*a*,*b*}, la expresión regular $((a \mid b)(a \mid b))^*$ denota el lenguaje compuesto por todas las cadenas cuya longitud es cero o un nº par, y están compuestas solamente por el símbolo *a*, el símbolo *b*, o por símbolos *a* y *b*.

Ejemplo 9.14

Sea el vocabulario $\{a,b\}$, la expresión regular $(a \mid b)(a \mid b)$ denota el lenguaje compuesto por todas las cadenas de longitud dos formadas con los símbolos *a* y *b*. Se pueden definir por enumeración $\{aa, ab, ba, bb\}$.

Ejemplo 9.15

Dar una expresión regular para identificador:

$$\text{<letra>}\ (\text{<letra>} \mid \text{<dígito>})^*$$

También se puede definir identificador como:

$$(a \mid b \mid c \mid \cdots \mid z)(a \mid b \mid c \mid d \mid \cdots \mid z \mid 0 \mid 1 \mid \cdots \mid 9)^*$$

Ejemplo 9.16

Dar una expresión regular para los números reales sin exponente del lenguaje Pascal estándar.

Solución :

$$(\lambda \mid + \mid -)(< dígito >< dígito >^* \cdot < dígito >^* < dígito >)$$

Ejemplo 9.17

La expresión regular a^*b^* denota el lenguaje $\{a^m b^n / m \geq 0 \quad \text{y} \quad n \geq 0\}$.

CAPÍTULO 10: AUTÓMATAS

La palabra *autómata* evoca algo que pretende imitar las funciones propias de los seres vivos, especialmente relacionadas con el movimiento, por ejemplo el típico robot antropomorfo. En el campo de los Traductores, Procesadores, Compiladores e Intérpretes, lo fundamental no es la simulación del movimiento, sino la simulación de procesos para tratar información.

La información se codifica en cadenas de símbolos, y un autómata es un dispositivo que manipula cadenas de símbolos que se le presentan a su entrada, produciendo otras tiras o cadenas de símbolos a su salida.

El autómata recibe los símbolos de entrada, uno detrás de otro, es decir secuencialmente. El símbolo de salida que en un instante determinado produce un autómata, no sólo depende del último símbolo recibido a la entrada, sino de toda la secuencia o cadena, que ha recibido hasta ese instante.

Todo lo anterior conduce a definir un concepto fundamental : **estado de un autómata**. *El estado de un autómata es toda la información necesaria en un momento dado, para poder deducir, dado un símbolo de entrada en ese momento, cual será el símbolo de salida*. Es decir, conocer el estado de un autómata, es lo mismo que conocer toda la historia de símbolos de entrada, así como el **estado inicial**, estado en que se encontraba el autómata al recibir el primero de los símbolos de entrada.

El autómata tendrá un determinado *número de estados* (pudiendo ser infinitos), y se encontrará en uno u otro según sea la historia de símbolos que le han llegado.

Se define *configuración de un autómata* a su situación en un instante. Se define *movimiento de un autómata* como el transito entre dos configuraciones.

Si un autómata se encuentra en un estado determinado, recibe un símbolo también determinado, producirá un símbolo de salida y efectuará un cambio o *transición* a otro estado (también puede quedarse en el mismo estado).

El campo de estudio de los Traductores, Procesadores e Intérpretes son los lenguajes y las gramáticas que los generan. Los elementos del lenguaje son sentencias, palabras, etc... formadas a partir de un alfabeto o vocabulario, que no es otra cosa que un conjunto finito de símbolos. Establecidas las reglas gramaticales, una cadena de símbolos pertenecerá al correspondiente lenguaje si tal cadena se ha formado obedeciendo esas reglas. Entonces un **autómata reconocedor de ese lenguaje**, funciona de tal forma que

cuando reciba a su entrada una determinada cadena de símbolos indica si dicha cadena pertenece o no al lenguaje. También se mostrará como existe un tipo de autómata para reconocer cada uno de los tipos de lenguajes generados por las correspondientes gramáticas.

10.1 Definición formal de autómata

Un autómata es una quíntupla A = (E, S, Q, f, g) donde :

E = {conjunto de entradas o vocabulario de entrada}
S = {conjunto de salidas o vocabulario de salida}
Q = {conjunto de estados}
$f : E \times Q \rightarrow Q$
$g : E \times Q \rightarrow S$

E es un conjunto finito, y sus elementos se llaman entradas o símbolos de entrada.

S es un conjunto finito, y sus elementos se llaman salidas o símbolos de salida.

Q es el conjunto de estados posibles, puede ser finito o infinito.

f es la *función de transición* o función del estado siguiente, y para un par del conjunto $E \times Q$ devuelve un estado perteneciente al conjunto Q. $E \times Q$ es el conjunto producto cartesiano de E por Q.

g es la *función de salida*, y para un par del conjunto $E \times Q$, devuelve un símbolo de salida del conjunto S.

10.2 Representación de autómatas

Los autómatas se pueden representar mediante :

- Tabla de transiciones
- Diagrama de Moore

10.2.1 Tabla de transiciones

Las funciones f y g pueden representarse mediante una tabla, con tantas filas como estados y tantas columnas como entradas. Así por ejemplo se puede representar el autómata A = (E, S, Q, f, g) donde E = {*a,b*}, S = {*0,1*}, Q = $\{q_1, q_2, q_3\}$ y las funciones f y g se pueden representar por :

f	*a*	*b*
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_3	q_1

g	*a*	*b*
q_1	*0*	*1*
q_2	*0*	*0*
q_3	*1*	*0*

Así se tiene que $f(a, q_1) = q_1$; $g(a, q_1) = 0$; o también $f(a, q_2) = q_3$; y $g(a, q_3) = 1$.

Ambas funciones también se pueden representar en una misma tabla de la siguiente forma :

f / g	*a*	*b*
q_1	q_1/*0*	q_2/*1*
q_2	q_3/*0*	q_2/*0*
q_3	q_3/*1*	q_1/*0*

10.2.2 Diagramas de Moore

Los diagramas de Moore son otra forma de representar las funciones de transición y salida de un autómata.

El diagrama de Moore es un grafo orientado en el que cada nodo corresponde a un estado; y si $f(\varepsilon, q_i) = q_j$ y $g(\varepsilon, q_i) = s$ existe un arco dirigido del nodo q_i al correspondiente q_j, sobre el que se pone la etiqueta ε / s, tal y como se muestra en la figura 3.

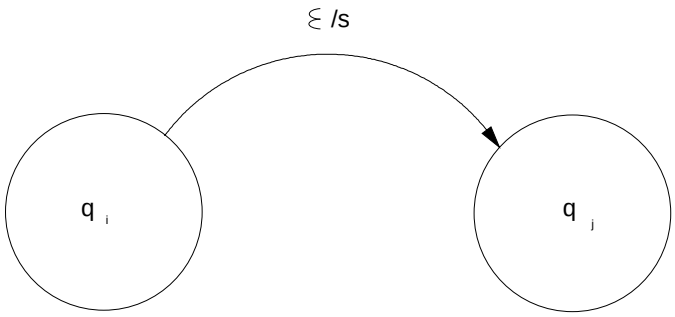


Fig. 3 : Diagrama de Moore.

Así continuando con el ejemplo del apartado 10.2.1, el autómata se representa con el diagrama de Moore de la figura 4.

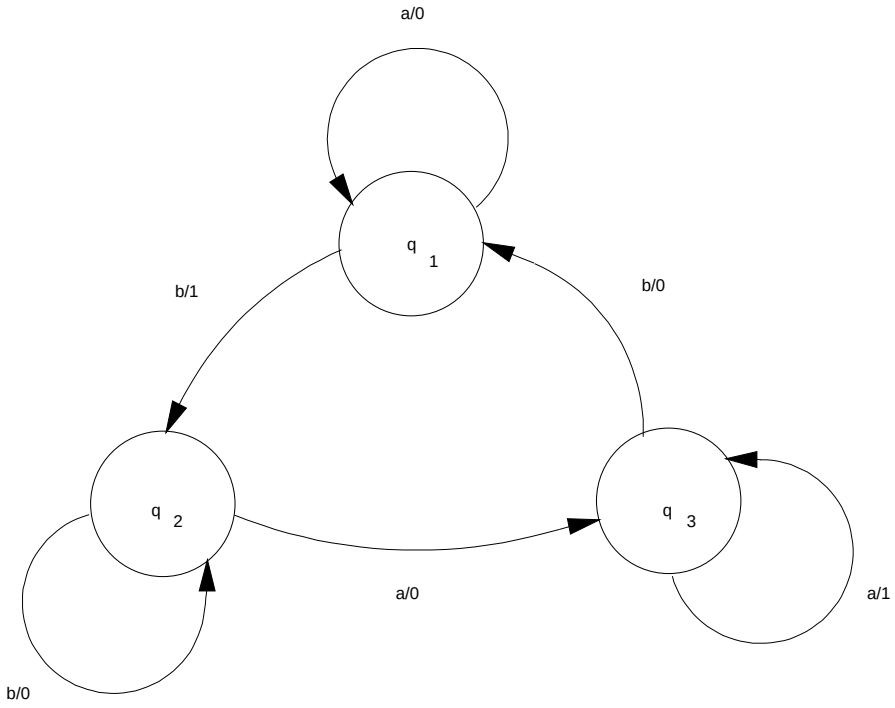


Fig. 4 : Ejemplo de diagrama de Moore.

10.3 Máquinas de Moore y Mealy

El modelo general de autómata que se ha definido se llama *máquina de Mealy*.

Se puede plantear el siguiente problema : si se considera que además de los elementos del vocabulario de entrada E, *un elemento vacio* λ, que físicamente indica que no hay entrada, se han de ampliar los dominios de las definiciones de la siguiente forma :

$$f : \{E \cup \{\lambda\}\} \times Q \rightarrow Q$$
$$g : \{E \cup \{\lambda\}\} \times Q \rightarrow S$$

La ampliación del dominio f no plantea ningún problema, pues se puede convenir que f(λ ,q)=q, es decir *si no hay entrada, no se cambia de estado.*

No ocurre lo mismo con la ampliación del dominio de g, ya que g(λ ,q), *produce una salida indeterminada*, pues depende del estado anterior al q. Así por ejemplo en el autómata del apartado 10.2.1, se observa que :

g(λ ,q_2)=*1*, si el estado anterior es q_1.

g(λ ,q_2)=*0*, si el estado anterior es q_2.

ya que si se llega a q_2 desde q_1 la salida es *1*, mientras que si se alcanza desde el propio q_2 la salida es *0*.

Entonces *sólo se puede ampliar el dominio de g, si a cada estado q se le puede asociar una salida y sólo una*.

Cuando esto ocurre para todo $q \in Q$, se puede definir una aplicación inyectiva :

$$h : Q \rightarrow S$$

tal que g se obtiene como composición de h por f :

$$g = h \circ f$$

es decir g(e,q) = h(f(e,q)), donde $e \in \{E \cup \{\lambda\}\}$ y $q \in Q$. En este caso, se puede decir, que *la salida sólo depende del estado*, y el autómata se llama **máquina de Moore**.

En una máquina de Mealy las salidas están asociadas a las transiciones, mientras que en una máquina de Moore las salidas están asociadas a los estados, o, lo que es lo mismo, todas las transiciones que conducen a un mismo estado tienen asociada la misma salida. También se puede decir que una máquina de Mealy, en el instante de efectuar una transición necesita conocer una entrada $e \in E$, ya que en general $g(\lambda, q)$ no está definida. Sin embargo en las máquinas de Moore la entrada puede ser λ.

Puesto que por definición toda máquina de Moore es una máquina de Mealy que cumple la condición de que a cada estado se le puede asociar una salida y sólo una, parece en principio que las máquinas de Moore son un subconjunto de las máquinasde Mealy. Sin embargo, se va a demostrar que, *dada una máquina de Mealy, siempre se puede encontrar una máquina de Moore equivalente*, normalmente, a costa de aumentar el número de estados.

En efecto sea la máquina de Mealy A = (E, S, Q, f, g) siempre se puede definir un nuevo autómata :

$$\hat{A} = (E, S, \hat{Q}, \hat{f}, \hat{g})$$

en el que $\hat{Q}$ se obtiene escindiendo $q \in Q$ en tantos estados q^s como salidas s pueden asociarse a q, es decir

$$\hat{Q} = \{q^s / \exists\, (q' \in Q \quad y \quad e \in E) \text{ tales que } f(e, q') = q \quad y \quad g(e, q') = s\}$$

y en el que $\hat{f}$ y $\hat{g}$ se definen de la siguiente forma :

$\hat{f}(e, q^s) = [f(e,q)]^{g(e,q)}$

$\hat{g}(e, q^s) = g(e,q)$

De este modo, a cada $q^s \in \hat{Q}$ se le puede asociar una sola salida s, y así tendra una función de salida :

$$\hat{h} : \hat{Q} \to S$$

tal que $\hat{g}(e, q^s) = \hat{h}(\hat{f}(e, q^s))$, por lo tanto $\hat{A}$ es una máquina de Moore.

En lo sucesivo siempre que sólo se tratarán autómatas que son máquinas de Moore.

Ejemplo 10.3.1

Construir una máquina de Moore equivalente a la mostrada en el ejemplo del apartado 10.2.1.

Solución : Se construye la siguiente tabla de transiciones combinada.

$g = h \circ f$	a	b
q_1^0	q_1^0	q_2^1
q_2^0	q_3^0	q_2^0
q_2^1	q_3^0	q_2^0
q_3^0	q_3^1	q_1^0
q_3^1	q_3^1	q_1^0

Se observa que al estado q_1 siempre se le asocia la salida 0; sin embargo q_2 puede tener la salida *0* ó *1*, se crean entonces los estados q_2^0 y q_2^1; de la misma forma q_3^0 y q_3^1.

Esta máquina de Moore se puede representar por el diagrama de la figura 5. Las salidas están indicadas en los nodos como superíndices de los estados.

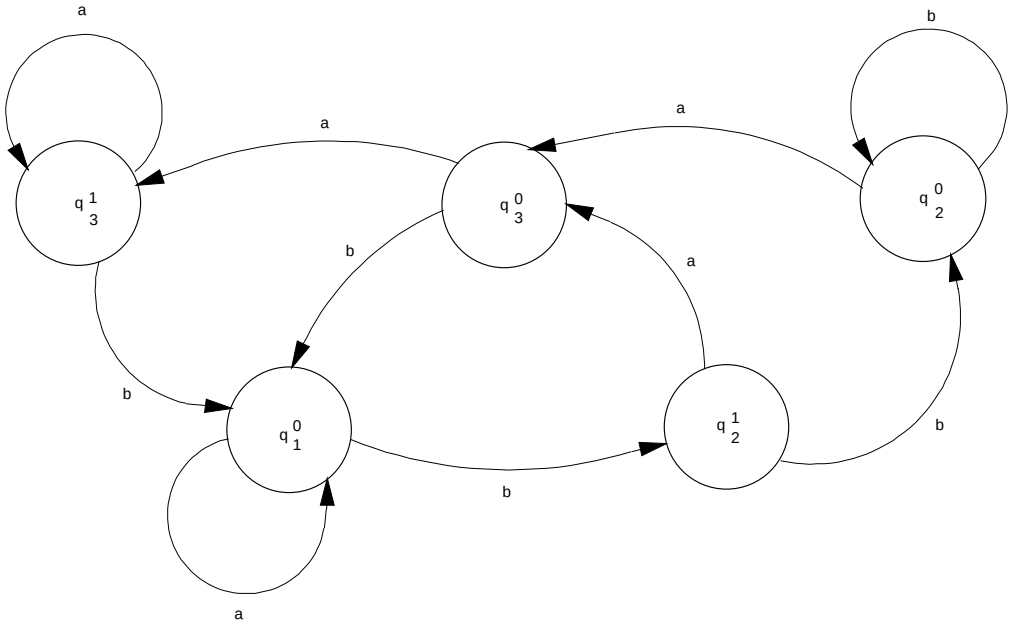


Fig. 5 : Ejemplo de máquina de Moore.

10.4 Estados accesibles de un autómata

Sea un autómata A=(E,S,Q,f,g), se dice que un estado q_j es accesible desde otro estado q_i, si existe una entrada $e \in E^*$ tal que $f(q_i,e)=q_j$.

Evidentemente todo estado es accesible desde sí mismo, puesto que $f(q_i,\lambda)=q_i$.

10.5 Autómatas conexos

Sea un autómata A=(E,S,Q,f,g), se dice que es *conexo* si todos los estados de Q son accesibles desde el estado inicial q_0.

Dado un autómata no conexo se puede encontrar otro equivalente y conexo eliminando los estados inaccesibles. Es evidente que los dos autómatas aceptarán el mismo lenguaje.

10.6 Autómatas deterministas y no deterministas

Se denomina autómata determinista cuando la función de cambio de estado f es determinista. En caso contrario se dice no determinista.

CAPÍTULO 11: JERARQUÍA DE LOS AUTÓMATAS

En el campo de los Traductores, Procesadores, Compiladores e Intérpretes, la teoría de autómatas interesa desde el punto de vista de las relaciones entre el tipo de lenguaje y la estructura de la máquina capaz de reconocerlo. Estas relaciones pueden ser en dos sentidos :

a) Dada una gramática G ¿Qué estructura deberá tener una máquina, M, tal que el lenguaje reconocido por la máquina M, es igual al lenguaje generado por G?

b) Dada una máquina M ¿Cuál será la gramática G, tal que el lenguaje generado por G, es igual al lenguaje reconocido por M?

En los apartados siguientes se expondrá como paralelamente a la jerarquía de gramáticas y de lenguajes, aparece una jerarquía de autómatas, cuya correspondencia puede verse en el diagrama de la figura 6.

CORRESPONDENCIA ENTRE LOS LENGUAJES LAS GRAMATICAS Y LOS AUTOMATAS

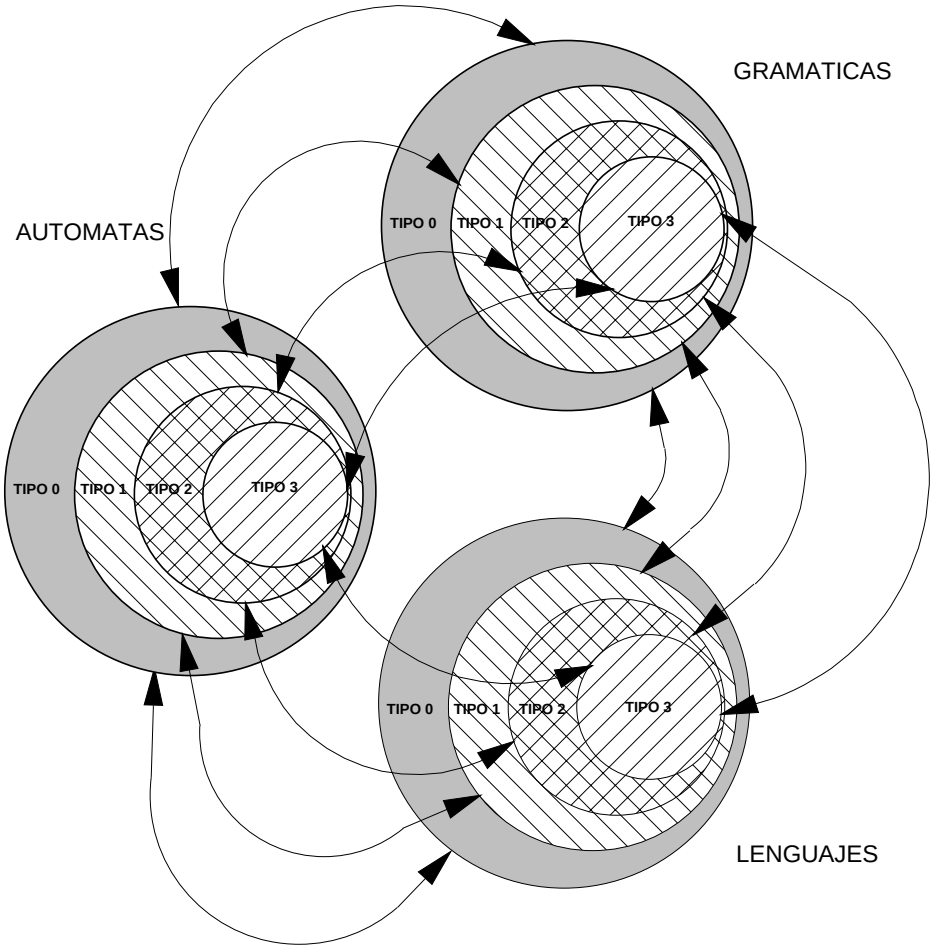


Fig. 6 : Correspondencia entre gramáticas, lenguajes y autómatas.

CAPÍTULO 12: MÁQUINAS DE TURING

Su nombre se debe a *Alan Mathison Turing*, que fué quien introdujo el concepto en 1936 (*A. M. Turing, 1936; C. Conde, 1969; J.E. Hopcroft, 1984*). Una máquina de Turing o autómata de tipo 0 es una construcción lógica, que se puede representar intuitivamente como un dispositivo mecánico (fig. 7), formado por una cinta infinita, dividida en celdas, y un cabezal de lectura/escritura que se mueve sobre dicha cinta, avanzando una celda de cada vez. En la figura 7 se representa el caso particular de un conjunto de símbolos en la cinta formados por 0 y 1.

Un movimiento de la máquina de Turing, depende del símbolo explorado con la cabeza, y del estado actual en el que se encuentra la máquina, el resultado puede ser :

a) Cambio de estado
b) Imprime un símbolo en la cinta reemplazando el símbolo leído.
c) Se mueve la cabeza de la cinta a la izda, a la derecha o se para.

Pueden darse los tres fenómenos anteriores, juntos o separados.

Formalmente una máquina de Turing es un autómata, y como todo autómata está formado por una quíntupla MT = (E, S, Q, f, g) sin embargo suele usarse la notación equivalente :

$$MT = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

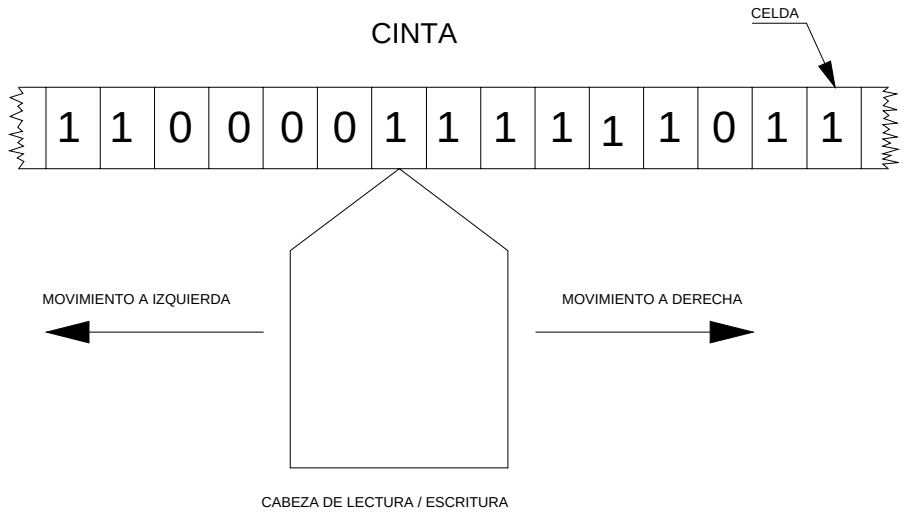


Fig. 7 : Esquema de máquina de Turing.

donde :

- Q = {conjunto de estados}
- Γ = {conjunto de símbolos permitidos en la cinta}
- B $\in \Gamma$ es el símbolo blanco.
- $\Sigma \in \Gamma$ es el subconjunto de símbolos de entrada no incluyendo el blanco.
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{I, D, S\}$ donde δ es la función del siguiente movimiento, I significa movimiento a izquierda, D significa movimiento a derecha, y S parada (stop).
- $q_0 \in Q$ es el estado inicial.
- $F \subset Q$ es el subconjunto de estados finales

El lenguaje aceptado o reconocido por una máquina de Turing, que denotaremos por L(MT), es el conjunto de palabras formadas con el alfabeto Σ^* , que hace que la máquina de Turing se pare al alcanzar un estado final.

En un principio la cabeza de la máquina de Turing está situada a la izquierda de la cadena a reconocer, y su estado es el estado inicial q_0.

Formalmente, el lenguaje aceptado por una máquina de Turing MT = $(Q, \Sigma, \Gamma, \delta, q_0, B, F)$ es :

$$L(MT) = \{W / W \in \Sigma^* \quad y \quad q_0 W \rightarrow \alpha_1 p \alpha_2 \quad p \in F, \quad \alpha_1, \alpha_2 \in \Gamma^*\}$$

Una máquina de Turing reconoce un lenguaje L si es capaz de reconocer todas las sentencias de dicho lenguaje. Si una sentencia de L es aceptada, la maquina se para, es decir alcanza un estado final. Pero si no se acepta la sentencia, la máquina de Turing no se parará nunca.

12.1 Teorema

Para toda gramática de tipo 0, existe una máquina de Turing que reconoce el lenguaje generado por dicha gramática.

12.2 Teorema

Para toda máquina de Turing, existe una gramática de tipo 0 que genera un lenguaje igual al reconocido por la máquina de Turing.

12.3 Corolario

Existe una correspondencia entre gramáticas, lenguajes y autómatas de tipo 0, tal y como se mostró en el diagrama de la figura 6.

Ejemplo 12.4

Diseñar una máquina de Turing que acepte el lenguaje $L = \{0^n 1^n / n \geq 1\}$.

Solución : Inicialmente la cinta contendrá $0^n 1^n$, sequido por ambos lados por un número infinito de blancos.

El algoritmo de reconocimiento es el siguiente : la cabeza de lectura/escritura se coloca en el 0 más a la izquierda y lo reemplaza por una X, se mueve a la derecha hasta encontrar el 1 más a la izquierda, reemplazándolo por una Y, después se mueve a la izquierda hasta encontrar la X más a la derecha, entonces se mueve una celda a la derecha, y repite el ciclo.

La máquina de Turing que reconoce el lenguaje L es $MT = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ donde :

$Q=\{q_0, q_1, q_2, q_3, q_4\}$
$\Sigma =\{0,1\}$
$\Gamma =\{0,1,X,Y,B\}$
$F=\{q_4\}$
q_0 es el estado inicial
La función δ viene dada por la tabla siguiente.

δ	0	1	X	Y	B
q_0	q_1,X,D	-	-	q_3,Y,D	-
q_1	q_1,0,D	q_2,Y,I	-	q_1,Y,D	-
q_2	q_2,0,I	-	q_0,X,D	q_2,Y,I	-
q_3	-	-	-	q_3,Y,D	q_4,B,D
q_4	S	S	S	S	S

Los guiones (-) en la tabla significan estados imposibles. La máquina primero escribe, luego cambia de estado y por último se mueve. Utilizando la máquina anterior se puede reconocer la cadena 0011 :

$$q_0 0011 \rightarrow X q_1 011 \rightarrow X 0 q_1 11 \rightarrow X q_2 0 Y 1 \rightarrow q_2 X 0 Y 1 \rightarrow X q_0 0 Y 1 \rightarrow$$
$$X X q_1 Y 1 \rightarrow X X Y q_1 1 \rightarrow X X q_2 Y Y \rightarrow X q_2 X Y Y \rightarrow X X q_0 Y Y \rightarrow$$
$$X X Y q_3 Y \rightarrow X X Y Y q_3 \rightarrow X X Y Y B q_4 \rightarrow S$$

CAPÍTULO 13: AUTÓMATAS LINEALES ACOTADOS

Los autómatas de tipo 1 son los autómatas limitados linealmente o autómatas lineales acotados, en inglés *linear bounded automaton*. Un autómata lineal acotado es una máquina de Turing que satisface las siguientes condiciones :

a) El alfabeto de entrada incluye dos símbolos especiales # y $, que son las marcas fin de cinta, izquierda y derecha respectivamente.

b) La cabeza del autómata no puede desplazarse fuera de los limites izquierdo y derecho de la cinta, y no puede imprimir otro símbolo sobre # y $.

Un autómata lineal acotado se puede representar por el diagrama de la figura 8. Se representa el caso particular de que los símbolos de la cinta son 0 y 1.

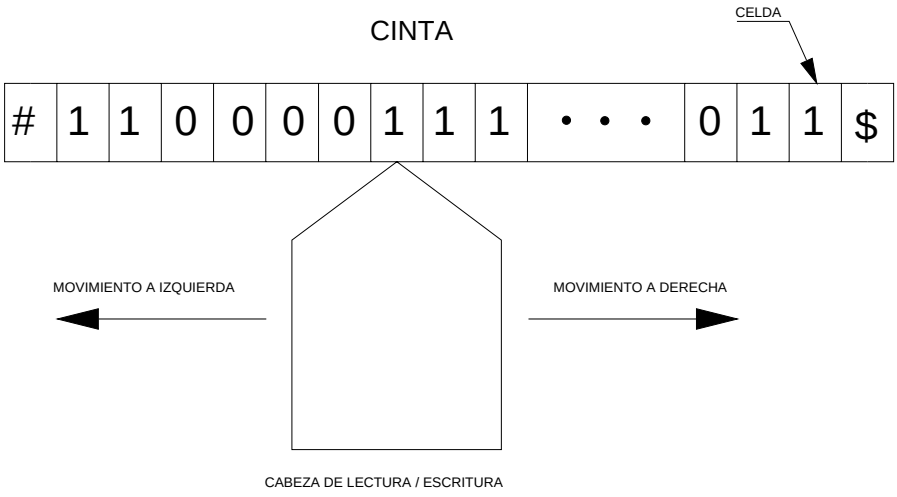


Fig. 8 : Esquema de autómata lineal acotado.

Un autómata lineal acotado se puede definir formalmente como una máquina de Turing con dos símbolos límite de la cinta.

$$ALA = (Q, \Sigma, \Gamma, \delta, q_0, \#, \$, F\}$$

donde Q, Σ, Γ, δ , q_0, y F significan lo mismo que en la máquina de Turing; # y $ son símbolos de Σ , correspondientes a la marca izquierda y derecha de la cinta.

El lenguaje aceptado por un autómata lineal acotado, es :

$$L(ALA) = \{ W / W \in \{ \Sigma - \{\#,\$\}\}^* \text{ y } q_0\#W\$ \rightarrow \alpha q \beta \text{ para algún } q \in F\}$$

Nótese que las marcas fin de cinta y comienzo de cinta no son consideradas como parte de la sentencia a reconocer. Un autómata lineal acotado no puede moverse fuera de la cadena de entrada.

13.1 Teorema

Para toda gramática sensible al contexto G_1 existe un autómata reconocedor lineal acotado R_{ALA}, tal que el lenguaje generado por la gramática $L(G_1)$, es reconocido por el autómata R_{ALA}.

$$L(G_1)=L(R_{ALA})$$

13.2 Teorema

Si $L(R_{ALA})$ es el lenguaje reconocido por un autómata lineal acotado, existe una gramática sensible al contexto G_1, tal que el lenguaje reconocido por el autómata es igual al generado por la gramática.

$$L(R_{ALA})=L(G_1)$$

13.3 Corolario

De los dos teoremas anteriores se deduce que existe una correspondencia entre las gramáticas de tipo 1, los lenguajes de tipo 1, y los autómatas lineales acotados.

CAPÍTULO 14: AUTÓMATAS DE PILA

Un autómata de pila AP, en inglés *pushdown automata*, es un autómata capaz de reconocer los lenguajes libres de contexto, o de tipo 2. Los autómatas de pila se pueden representar como una máquina de Turing, que sólo puede leer de una cinta, y que puede guardar resultados intermedios en una pila. De hecho, su capacidad de procesamiento es inferior a los ALA, debido a las siguientes restricciones sobre las posibles operaciones con la cinta y la pila :

- La cinta se desplaza en un sólo sentido, y su cabeza sólo puede leer.
- La pila, está limitada en un extremo por definición, cuando se lee un elemento de la pila, este desaparece o se saca, y cuando se escribe en la pila, se introduce un elemento.

Las operaciones elementales, que se pueden realizar con un AP, son de dos tipos :

- *Dependientes de la entrada* : se lee $e_i \in \Sigma$, y se desplaza la cinta, y en función de e_i, q_j (el estado en que se encuentra la cinta), y Z (el valor de la pila), el control pasa a otro estado q_l, y en la pila se introduce Z', o se extrae Z, o no se hace nada.
- *Independientes de la entrada* : puede ocurrir lo mismo que en el caso anterior, sólo que e_i no interviene, la cinta no se mueve, lo que permite manejar la pila sin las informaciones de entrada.

En cualquier caso, si se vacia la pila (es decir se extrae todas las Z) el autómata se para.

Un autómata de pila se puede representar intuitivamente según el esquema de la figura 9.

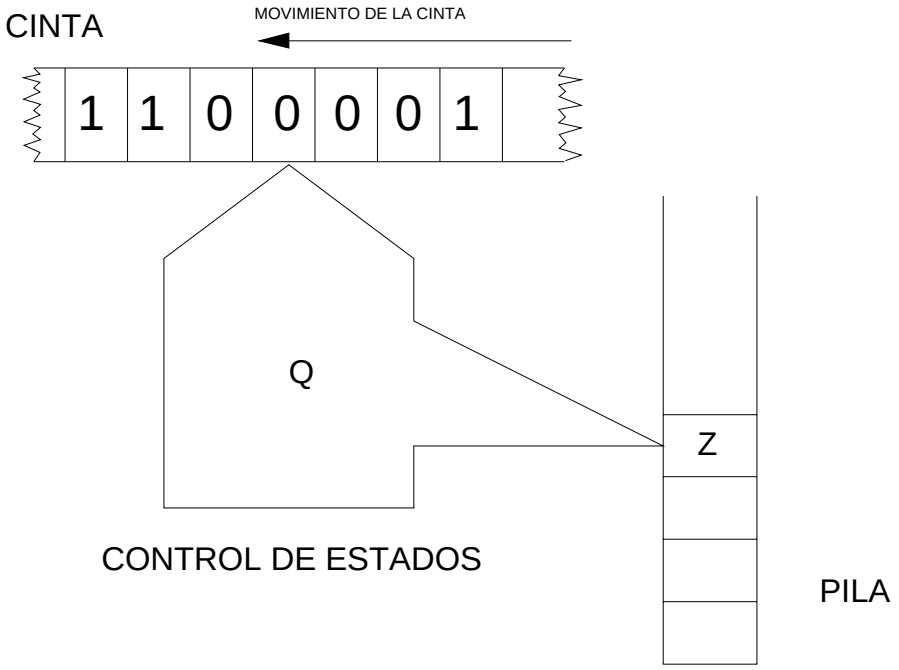


Fig. 9 : Esquema de un autómata de pila.

Un autómata de pila se puede definir formalmente como una séptupla :

$$AP = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

donde :

- Q es el *conjunto finito de estados*.
- Σ es el *alfabeto de entrada*, es finito.
- Γ es el *alfabeto de pila*.
- δ es la *función de transición*, y es una aplicación de la forma :

$$\delta : Q \times \{\Sigma \cup \{\lambda\}\} \times \Gamma \rightarrow Q \times \Gamma^*$$

- q_0 es el *estado inicial*, y cumple $q_0 \in Q$.
- Z_0 es el *símbolo inicial que contine la pila* antes de comenzar, evidentemente $Z_0 \in \Gamma$.
- F es el *conjunto de estados finales*, evidentemente $F \subset Q$.

Tal y como ha sido definida la función de transición δ, el autómata es en general *no determinista*. Se entiende por no determinista, cuando el resultado de la función no está determinado, es decir pueden resultar dos o más valores, sin que la función precise cual va a tomar de ellos.

Se define **configuración** de un autómata de pila a su situación en un instante, que se puede expresar formalmente mediante el terceto :

$$(q, W, \alpha)$$

donde :

- q representa el *estado actual del autómata*, y evidentemente $q \in Q$.

- W es la *cadena de entrada que resta por analizar*, siendo $W \in \Sigma^*$; si $W = \lambda$, se asume que toda la cadena de entrada ya ha sido leída.

- α es el *contenido de la pila*, en el instante considerado, y $\alpha = \lambda$ indica que la pila está vacia. Por supuesto $\alpha \in \Gamma^*$.

Se entiende por **movimiento** de un autómata a una transición entre configuraciones, y se representa por el operador binario $\rightarrow$. Así por ejemplo sea el siguiente movimiento :

$$(q, aW, z\,\alpha) \rightarrow (q', W, \gamma\alpha)$$

donde la función de transición para esta entrada toma el valor $\delta(q,aW,Z) \rightarrow (q',\gamma)$, siendo $q \in Q$, $a \in \Sigma \cup \{\lambda\}$, $W \in \Sigma^*$, $Z \in \Gamma$, y $\alpha \in \Gamma^*$.

Es decir que el autómata se encuentra en el estado *q*, que la cabeza de lectura de la cinta se encuentra sobre el símbolo *a*, y que la pila contiene determinados símbolos representados por la concatenación de Z y α, siendo el situado más a la izquierda Z el que se encuentra en cabeza de la pila (fig. 10).

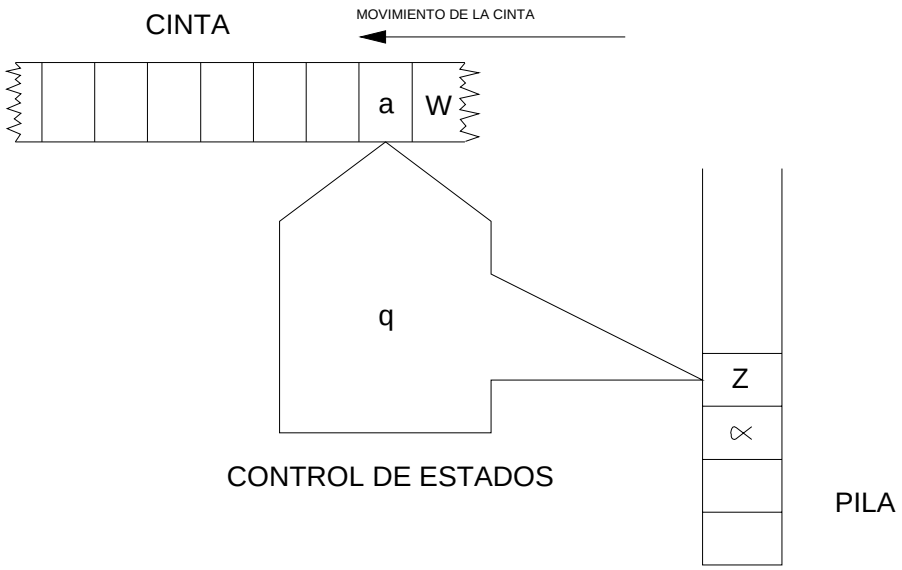


Fig. 10 : Transición en un autómata de pila.

El autómata pasa de la configuración $(q, aW, z\alpha)$ a la configuración $(q', W, \gamma\alpha)$, es decir pasa a un estado q', avanzando la cabeza de lectura al siguiente símbolo y procediendo a realizar determinadas sustituciones en la cabeza de la pila (fig. 11).

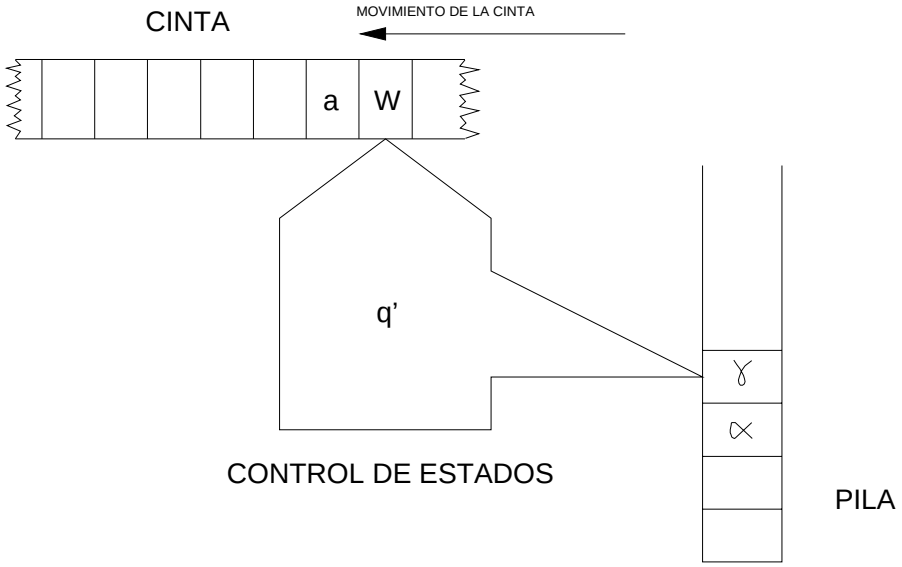


Fig. 11 : Transición en un autómata de pila.

14.1 Lenguaje reconocido por un autómata de pila

Se puede definir de dos formas :

a) El autómata reconoce una cadena de entrada, cuando alcanza el estado final, es decir el lenguaje reconocido por un autómata de pila se puede expresar de la siguiente forma :

$$L(AP) = \{W/W \in \Sigma^* \quad y \quad (q_{0,}W, Z_0) \rightarrow (q_f, \lambda, \alpha)\}$$

b) El autómata reconoce la cadena cuando la pila queda vacia, independientemente del estado al que se llegue, entonces el lenguaje reconocido por el autómata es :

$$L(AP) = \{W/W \in \Sigma^* \quad y \quad (q_{0,}W, Z_0) \rightarrow (q', \lambda, \lambda)\}$$

Se puede demostrar que ambas definiciones de L(AP), son equivalentes, en el sentido de que la clase de lenguajes aceptados por los autómatas de pila es la misma en ambos casos.

14.1.1 Teorema

Para toda gramática libre de contexto G_2, existe un reconocedor constituido por un autómata de pila R_{AP}, tal que el lenguaje generado por la gramática $L(G_2)$ es reconocido por el autómata R_{AP}.

$$L(G_2)=L(R_{AP})$$

14.1.2 Teorema

Para todo reconocedor constituido por un autómata de pila R_{AP}, existe una gramática libre de contexto G_2, tal que el lenguaje reconocido por el autómata es igual al generado por la gramática.

$$L(R_{AP})=L(G_2)$$

14.1.3 Corolario

De los dos teoremas anteriores se deduce que el conjunto de lenguajes reconocidos por los autómatas de pila, son los lenguajes de tipo 2 y que todo lenguaje de tipo 2 se puede reconocer por un autómata de pila. También se puede deducir que existe una correspondencia entre las gramáticas, los lenguajes y los autómatas de tipo 2.

Ejemplo 14.1.4

Construir un autómata de pila que reconozca el lenguaje $L = \{0^n 1^n / n \geq 0\}$.

Solución : Se puede definir un autómata de pila, P, de la forma:

$$P = (\{q_0, q_1, q_2\}, \{0,1\}, \{Z,0\}, \delta, q_0, Z, \{q_0\})$$

donde δ se define :

$$\delta(q_0, 0, Z) \rightarrow (q_1, 0Z)$$

$$\delta(q_1, 0, 0) \rightarrow (q_1, 0, 0)$$

$$\delta(q_1, 1, 0) \rightarrow (q_2, \lambda)$$

$$\delta(q_2, 1, 0) \rightarrow (q_2, \lambda)$$

$$\delta(q_2, \lambda, Z) \rightarrow (q_0, \lambda)$$

$$\delta(q_0, \lambda, Z) \rightarrow (q_0, \lambda)$$

El autómata va copiando todos los 0 de la cinta en la pila, y cuando va encontrando los 1, va sacando los ceros de la pila. Puede observarse que la cadena vacia también pertenece al lenguaje.

También se puede definir δ por la tabla, donde los guiones (-) representan estados imposibles.

δ	0	1	λ
q_0,Z	q_1,0Z	-	q_0,λ
q_1,0	q_1,00	q_2,λ	-
q_2,0	-	q_2,λ	-
q_2,Z	-	-	q_0,λ

Por ejemplo para reconocer la sentencia 0011 del lenguaje, se realizan las siguientes transiciones :

$$(q_0,0011,Z) \rightarrow (q_1,011,0Z) \rightarrow (q_1,11,00Z) \rightarrow (q_2,1,0Z) \rightarrow (q_2,\lambda,Z) \rightarrow (q_0,\lambda,\lambda)$$

En el caso general de i≥1, las transiciones son las siguientes :

$$(q_0,00^i11^i,Z) \rightarrow (q_1,0^i11^i,0Z) \overset{i}{\rightarrow} (q_1,11^i,00^iZ) \rightarrow (q_2,1^i,0^iZ) \overset{i}{\rightarrow} (q_2,\lambda,Z) \rightarrow (q_0,\lambda,\lambda)$$

y para i=0 la transición es :

$$(q_0,00^i11^i,Z) \rightarrow (q_0,\lambda,\lambda)$$

14.2 Algoritmo de transformación de una gramática de tipo 2 en un autómata de pila

Dada una gramática de tipo 2 se puede construir un autómata de pila (por lo general no determinista) que reconozca el mismo lenguaje que genera la gramática.

Sea la gramática G=(VN,VT,S,P) se desea construir el autómata de pila AP=(Q,Σ,Γ,δ, q_0, z_0, F) que reconozca el mismo lenguaje generado por G. Así se puede obtener el autómata AP de la siguiente forma:

$\Sigma = VT$ El vocabulario o alfabeto de entrada del autómata Σ coincide con el vocabulario terminal VT de la gramática.

$\Gamma = VN \cup VT$ El alfabeto de la pila coincide con la union de los vocabularios terminal y no terminal de la gramática.

$z_0 = S$ El símbolo inicial que contiene la pila es el símbolo inicial de la gramática

$Q = \{q\}$ El conjunto de estados del autómata tiene un estado único q

$F = \{\varnothing\}$ El conjunto de estados finales del autómata de pila está vacio. El autómata sólo se parará si se vacia la pila.

δ La función de transición δ se construye de la siguiente forma:

a) Para todo símbolo terminal x se construye la siguiente relación $\delta(q, x, x) \rightarrow (q, \lambda)$ que se puede interpretar como:
   - Saca x de la pila
   - Avanza el símbolo x de la cinta
   - No escribe nada en la pila
   - No cambia de estado

b) Para toda regla de producción $A \rightarrow \alpha$ perteneciente a P se construye la siguiente relación $\delta(q, \lambda, A) \rightarrow (q, \alpha)$ que se puede interpretar como:
   - No avanza la cinta
   - Saca A de la pila
   - Mete α en la pila
   - No cambia de estado

Ejemplo 14.2.1

Sea la gramática G=(VN,VT,S,P) que representa el manejo de expresiones aritméticas, siendo VN={E,T,F} donde E es la abreviatura de expresión, T la de término y F la de factor. VT={a,+,*,(,)} donde *a* representa a los identificadores. El símbolo inicial S=E. Las reglas de producción son las siguientes:

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T*F \mid F$$
$$F \rightarrow (E) \mid a$$

Construir un autómata de pila que reconozca el mismo lenguaje generado por la gramática G.

Solución: AP=(Q,Σ,Γ,δ, q_0, z_0, F) donde

$$Q = \{q\}$$
$$\Sigma = \{a, +, *, (,)\}$$
$$\Gamma = \{a, +, *, (,), E, T, F\}$$
$$q_0 = q$$
$$z_0 = E$$
$$F = \{\varnothing\}$$

a) Símbolos terminales

(1) $\delta(q, a, a) \rightarrow (q, \lambda)$

(2) $\delta(q, +, +) \rightarrow (q, \lambda)$

(3) $\delta(q, *, *) \rightarrow (q, \lambda)$

(4) $\delta(q, (, () \rightarrow (q, \lambda)$

(5) $\delta(q,),)) \rightarrow (q, \lambda)$

b) Reglas de producción

(6) $\delta(q, \lambda, E) \rightarrow (q, T)$

(7) $\delta(q, \lambda, E) \rightarrow (q, E + T)$

(8) $\delta(q, \lambda, T) \rightarrow (q, F)$

(9) $\delta(q, \lambda, T) \rightarrow (q, T*F)$

(10) $\delta(q,\lambda,F) \rightarrow (q,(E))$

(11) $\delta(q,\lambda,F) \rightarrow (q,a)$

Se puede observar que si se desea reconocer una cadena por ejemplo $a+a$ se producen retrocesos, debido a que el autómata no es determinista. Así se pone la cadena a reconocer $a+a$ en la cinta, el autómata en el estado inicial q, y la pila con su valor inicial E. Las reglas de transición entre configuraciones están numeradas y el autómata aplica en primer lugar la regla con número más bajo.

$$(q,a+a,E) \overset{(6)}{\rightarrow} (q,a+a,T) \overset{(8)}{\rightarrow} (a,a+a,F) \overset{(10)}{\rightarrow} (a,a+a,(E))$$

Aquí se alcanza una configuración imposible. El símbolo terminal en la cima de la pila no coincide con el símbolo terminal de la cinta. Se debe retroceder hasta donde se eligió la última regla alternativa que fue la (10). Se observa que hay otra regla la (11).

$$(q,a+a,F) \overset{(11)}{\rightarrow} (q,a+a,a) \overset{(1)}{\rightarrow} (q,+a,\lambda)$$

Se alcanza una configuración imposible la pila está vacia pero queda una parte de la cadena en la cinta. Se tiene que retroceder hasta aplicar otra regla alternativa. Se retrocede hasta donde se aplicó la regla (8) y ahora se utiliza la regla (9).

$$(q,a+a,T) \overset{(9)}{\rightarrow} (q,a+a,T*F) \overset{(8)}{\rightarrow} (q,a+a,F*F) \overset{(10)}{\rightarrow} (q,a+a,(E)*F)$$

Se alcanza una configuración imposible. Se retrocede y se aplica la regla (11) en vez de la (10).

$$(q,a+a,F*F) \overset{(11)}{\rightarrow} (q,a+a,a*F) \overset{(1)}{\rightarrow} (q,+a,*F)$$

Se alcanza una configuración imposible. Se retrocede hasta donde se aplicó la regla (6) y ahora se utiliza la regla (7). Estamos otra vez en la configuración inicial.

$$(q,a+a,E) \overset{(7)}{\rightarrow} (q,a+a,E+T) \overset{(6)}{\rightarrow} (q,a+a,T+T) \overset{(8)}{\rightarrow} (q,a+a,F+T) \overset{(10)}{\rightarrow} (q,a+a,(E)+T)$$

Se alcanza una configuración imposible y se retrocede hasta donde se aplicó la regla (10) y ahora se usa la (11).

$$(q,a+a,F+T) \overset{(11)}{\rightarrow} (q,a+a,a+T) \overset{(1)}{\rightarrow} (q,+a,+T) \overset{(2)}{\rightarrow} (q,a,T) \overset{(8)}{\rightarrow} (q,a,F) \overset{(10)}{\rightarrow} (q,a,(E))$$

Se alcanza una configuración imposible y se retrocede hasta donde se aplicó la regla (10) y ahora se usa la (11).

$$(q,a,F) \overset{(11)}{\rightarrow} (q,a,a) \overset{(1)}{\rightarrow} (q,\lambda,\lambda)$$

La cadena $a+a$ ha sido reconocida por el autómata.

Ejercicio 14.2.2

Sea la gramática G=(VN,VT,S,P) que representa el manejo de expresiones tipo Lisp, siendo VN={S,R} y VT={x,(,), , }. Las reglas de producción son las siguientes:

$S \rightarrow x$

$S \rightarrow (SR$

$R \rightarrow ,SR$

$R \rightarrow)$

Determinar el lenguaje que genera y construir el autómata que lo reconoce. Poner un ejemplo de reconocimiento de una cadena del lenguaje.

CAPÍTULO 15: AUTÓMATAS FINITOS

Los autómatas finitos reconocen los lenguajes regulares, o de tipo 3 y se pueden representar intuitivamente por una cinta y una cabeza de lectura (fig. 12).

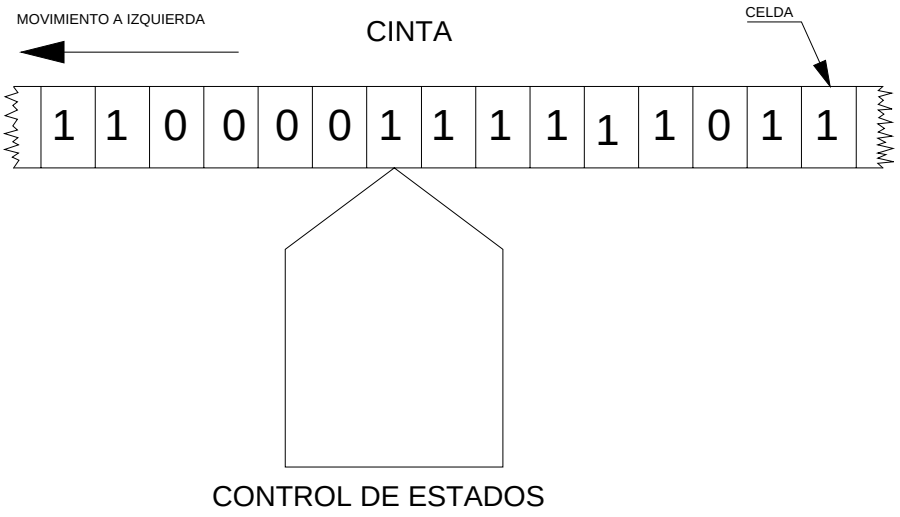


Fig. 12 : Esquema intuitivo de un autómata finito.

La *cinta de entrada*, sólo contiene símbolos de un determinado alfabeto, y se mueve en una sóla dirección.

El *control de estados*, determina el funcionamiento del autómata.

Una sentencia de un lenguaje determinado, colocada en la cinta, y leída por el autómata finito, es reconocida por éste, si el control de estados llega a un estado final.

15.1 Definición formal de autómata finito

Se puede definir como una quíntupla $AF = (E, Q, f, q_1, F)$ donde :

E = {conjunto finito de símbolos de entrada, que constituye el vocabulario}

Q = {conjunto finito de estados}

$f: E^* \times Q \rightarrow Q$ es la *función de transición*

$q_1 \in Q$, es el *estado inicial*

$F \subset Q$ es el *conjunto de estados finales*

Se entiende por *configuración* de un autómata finito, a un par de la forma (q, W) donde q, es el estado actual, y W la cadena que queda por leer en ese instante. Según la definición anterior, se puede afirmar que la *configuración inicial* de un autómata finito es el par (q_1, t) siendo t la sentencia o cadena de entrada a reconocer. La *configuración final* se representa por el par (q_i, λ) donde $q_i \in$ F, y λ indica que no queda nada por entrar de la cinta.

Un *movimiento* de un autómata finito, puede definirse como el tránsito entre dos configuraciones, y se representa por (q, aW) → (q',W) y se debe de cumplir que f(q,a)=q'.

15.2 Lenguaje reconocido por un autómata finito

Cuando un autómata transita a una configuración final partiendo de la configuración inicial, en varios movimientos, se dice que se ha producido aceptación o *reconocimiento de la cadena* de entrada. Es decir que dicha cadena, pertenece al lenguaje reconocido por el autómata.

Por el contrario, cuando el autómata finito no es capaz de llegar a un estado final, se dice que el autómata no reconoce dicha cadena y que por tanto no pertenece al lenguaje.

El lenguaje reconocido por un autómata finito, es:

$$L(AF) = \{t/t \in E^*, \quad (q_{1,}t) \rightarrow (q_i, \lambda), \quad q_i \in F\}$$

15.2.1 Teorema

Para toda gramática regular, G_3, existe un autómata finito, AF, tal que el lenguaje reconocido por el autómata finito es igual al lenguaje generado por la gramática.

$$L(AF) = L(G_3)$$

15.2.2 Teorema

Para todo autómata finito, AF, existe una gramática regular, G_3, tal que el lenguaje generado por la gramática es igual al lenguaje reconocido por el autómata finito

$$L(G_3) = L(AF)$$

15.2.3 Corolario

Según el teorema 15.2.1, se tiene que $\{L(G3)\} \subset \{L(AF)\}$ y por el teorema 15.2.2, $\{L(AF) \subset \{L(G3)\}$, luego $\{L_{regulares}\} = \{L(AF)\} = \{L(G_3)\}$

La forma habitual de representar los autómatas finitos es mediante un grafo o diagrama de estados (fig. 13), donde los nodos son los estados y las ramas están marcadas con los símbolos del alfabeto de entrada. Las ramas se construyen según la función de transición, así debe de cumplir $f(q_1, a) \rightarrow q_2$.

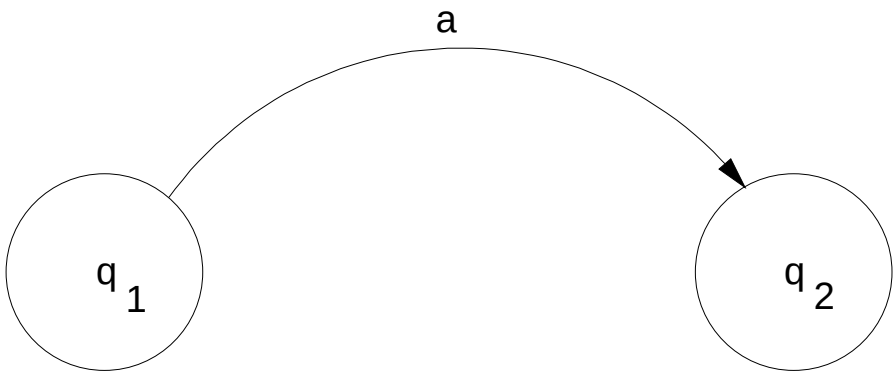


Fig. 13 : Transición entre dos estados.

Los nodos que representan los *estados finales*, suelen marcarse con un doble círculo, y el *estado inicial* también se marca con una flecha, encima de la cual se coloca la palabra INICIO.

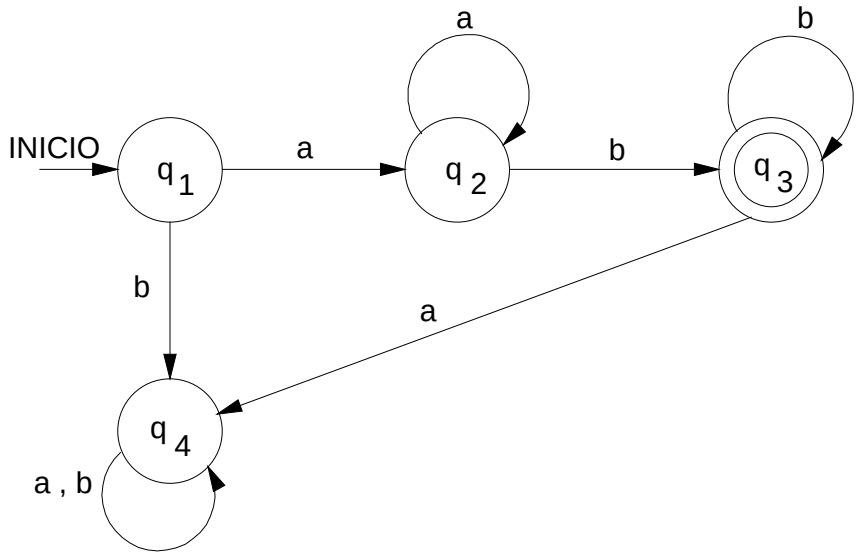
Ejemplo 15.2.4

Sea el autómata finito $A_1 = (E, Q, f, q_1, F)$ donde $E = \{a, b\} \cup \{\lambda\}$; $Q = \{q_1, q_2, q_3, q_4\}$ y la función f viene dada por la tabla siguiente y el conjunto de estados finales es F={q_3}

f	*a*	*b*
q_1	q_2	q_4
q_2	q_2	q_3
q_3	q_4	q_3
q_4	q_4	q_4

Determinar el lenguaje que reconoce, representar el diagrama de Moore, e indicar la expresión regular que representa al lenguaje.

Solución : Se construye el diagrama de Moore, colocando en primer lugar todos los estados dentro de círculos, marcando con doble círculo el estado final. El estado inicial se indica con una flecha que lo señala con la palabra INICIO encima. Para construir las ramas, nos situamos en el primer estado de la tabla de transiciones y se observa que $f(q_1,a)=q_2$, entonces se traza una flecha entre q_1 y q_2, apuntando a q_2, y se coloca encima de la flecha el símbolo del vocabulario de entrada a. De igual forma se recorre la tabla de transiciones para cada estado y entrada completándose el diagrama de Moore.



q

Fig. 14 : Diagrama de Moore del ejemplo 15.2.4

El lenguaje generado se obtiene partiendo del estado inicial y recorriendo todos los caminos posibles para alcanzar el estado final. Así se obtiene que este autómata reconoce el lenguaje :

$$L(A_1) = \{ab, aab, \ldots, abbb, \ldots, aabb, \ldots\}$$

$$L(A_1) = \{a^n b^m / n \geq 1 m \geq 1\}$$

La expresión regular que denota el lenguaje es a^+b^+ o también aa^*bb^*.

Ejemplo 15.2.5

Sea el autómata finito $A_2 = (E, Q, f, q_1, F)$ donde E = {0,1} , $Q = \{q_1, q_2, q_3, q_4\}$ y f se define por la tabla siguiente, y F={q_2}.

f	0	1
q_1	q_4	q_2
q_2	q_3	q_4
q_3	q_4	q_2
q_4	q_4	q_4

Construir el diagrama de Moore, y determinar el lenguaje que reconoce, denotándolo con su expresión regular.

Solución : Se construye el diagrama de Moore de forma análoga al ejemplo anterior (fig. 15).

El lenguaje generado es el siguiente :

$$L(A_2) = \{1,101,10101,\ldots\} = \{1(01)^n / n \geq 0\}$$

La expresión regular $1(01)^*$.

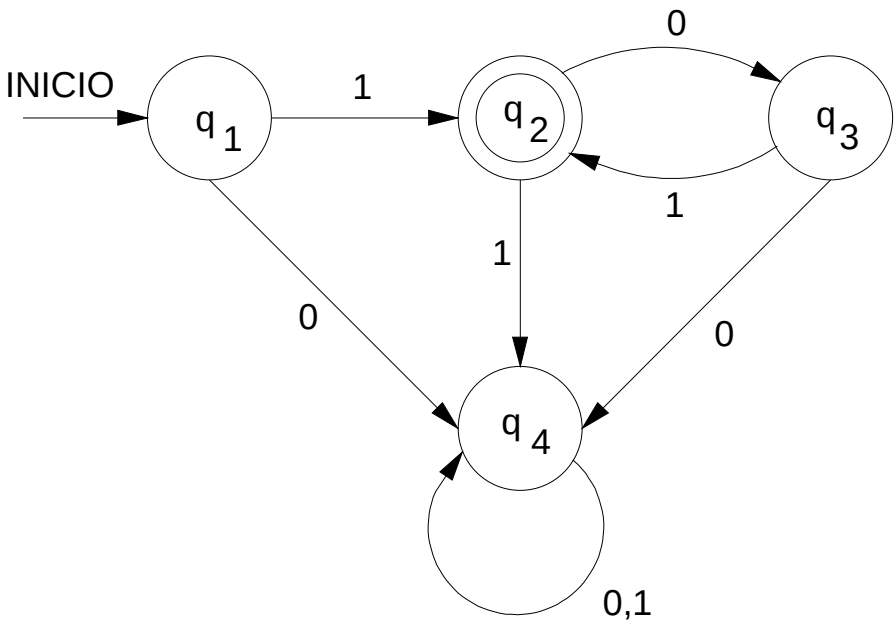


Fig. 15 : Diagrama de Moore del ejemplo 15.2.5.

Ejemplo 15.2.6

Sea el autómata finito A_3 = (E, Q, f, q_1, F) donde E = {*a,b,c*}, Q = {q_1, q_2, q_3, q_4, q_5}, f se representa por la tabla siguiente y F={q_2, q_4}.

Representar el diagrama de Moore, determinar el lenguaje que reconoce, y denotarlo con una expresión regular.

f	*a*	*b*	*c*
q_1	q_2	q_3	q_5
q_2	q_5	q_5	q_5
q_3	q_5	q_5	q_4
q_4	q_5	q_5	q_5
q_5	q_5	q_5	q_5

Solución : El diagrama de Moore se construye al igual que en los ejemplos anteriores (fig. 16).

El lenguaje generado es L(A_3)={*a,bc*}, se puede denotar con la expresión regular *a* | *bc*.

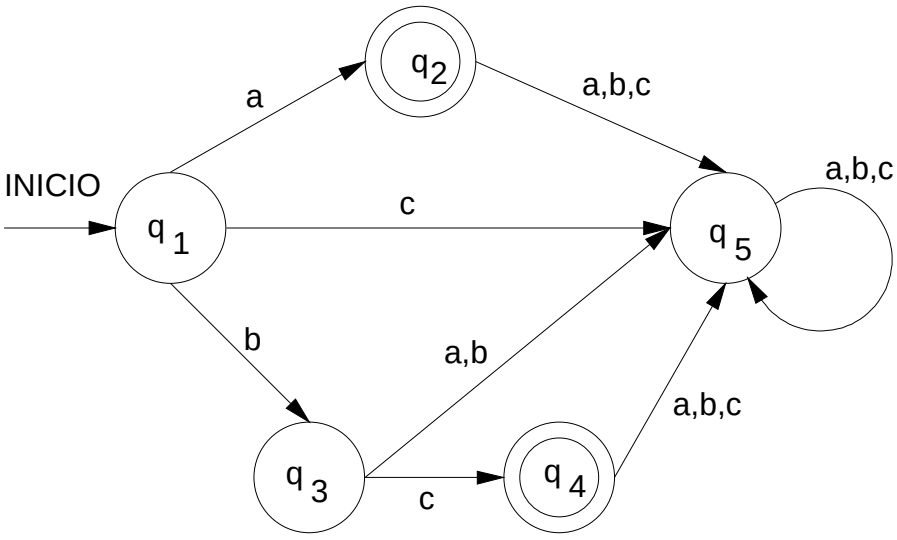


Fig. 16 : Diagrama de Moore del ejemplo 15.2.6.

Ejemplo 15.2.7

Sea el autómata $A_4 = (E = \{1,2,3\}, Q = \{q_1, q_2, q_3\}, f, q_1, F = \{q_2\})$, donde f viene dada por la tabla siguiente.

f	1	2	3
q_1	q_1	q_1	q_2
q_2	q_3	q_3	q_3
q_3	q_3	q_3	q_3

Determinar el diagrama de Moore, el lenguaje que genera y la expresión regular que lo describe.

Solución : El diagrama de Moore se construye al igual que en los ejemplos anteriores (fig. 17).

El lenguaje reconocido es el siguiente :

$$L(A_4) = \{3,13,113,1113,\ldots,123,12223,\ldots,213,21113,\ldots\}$$

$$L(A_4) = \left\{1^{(n_1)}2^{(n_2)}1^{(n_3)}2^{(n_4)}\ldots 3 / n_1, n_2, \ldots \geq 0\right\}$$

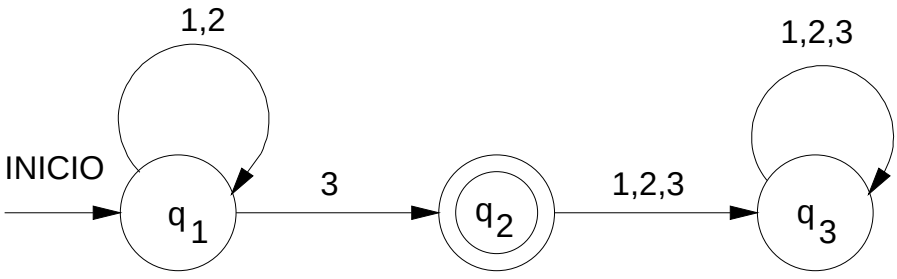


Fig. 17 : Diagrama de Moore del ejemplo 15.2.7.

La expresión regular es $(1 \mid 2)^*3$.

Ejemplo 15.2.8

Construir un autómata finito que reconozca un identificador de un lenguaje de programación, definido en EBNF de la forma :

<identificador> ::= <letra> {<letra>|<dígito>}
<letra> ::= a|b|...|z
<dígito> ::= 0|1|2|...|9

Solución : Este ejemplo es inverso a los anteriores, pues se da un lenguaje y se pide el autómata que lo reconoce. En primer lugar se construye un diagrama de Moore, de tal forma que a partir del estado inicial, después de leer una letra, acepte letras o dígitos de forma variable, y cuando encuentre un carácter diferente de letra o dígito alcance el estado final. El diagrama de Moore es el que se muestra en la figura 18.

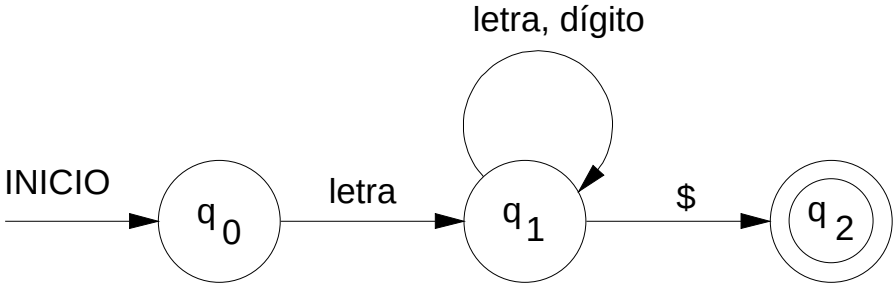


Fig. 18 : Diagrama de Moore del ejemplo 15.2.8.

$ representa a todos los caracteres diferentes de letra o dígito.

El autómata finito se deduce del diagrama de Moore y es el siguiente :

$$AF = (E = \{a, b, \ldots, z, 0, 1, \ldots 9, \$\}, Q = \{q_0, q_1, q_2\}, f, q_0, F = \{q_2\})$$

donde f se define por :

f	<letra>	<dígito>	$
q_0	q_1	-	-
q_1	q_1	q_1	q_2
q_2	-	-	-

15.3 Clasificación de los autómatas finitos

Cuando se definió autómata finito, la función $f: E^* \times Q \to Q$, es en general no determinista. Así en función de f, se hablará de *autómatas finitos deterministas AFD* y *autómatas finitos no deterministas AFND.*

Un autómata finito no determinista AFND se caracteriza por la posibilidad de que dada una entrada *e* en un estado q_i, se pueda pasar a un estado q_j, q_k,...,q_n sin saber a ciencia cierta, a cual de esos estados pasará. Existiendo la misma probabilidad de que pase a cualquiera de dichos estados.

15.3.1 Autómatas finitos no deterministas

La definición de autómata finito no determinista AFND coincide con la de autómata finito :

$$AFND = (E, Q, f, q_{1,} F)$$

con la salvedad de que $f{:}E^{*}xQ \rightarrow Q$ es no determinista.

Ejemplo 15.3.1.1

Sea el autómata finito no determinista $AFND = (E, Q, f, q_{1,} F)$ donde E={*a*,*b*}, $Q = \{q_1, q_2, q_3, q_4\}$, $F = \{q_4\}$ y la función f viene dada por la siguiente tabla :

f	*a*	*b*
q_1	$\{q_2,q_3\}$	λ
q_2	λ	$\{q_2,q_4\}$
q_3	q_3	q_4
q_4	q_4	λ

Determinar el lenguaje que reconoce, y dar su expresión regular.

Solución : El diagrama de Moore se construye al igual que en los ejemplos anteriores de autómatas finitos, con la salvedad de que para una entrada a un estado puede salir más de una flecha de un determinado estado (fig. 19).

El lenguaje reconocido es el siguiente :

$$a(b^{*}b \mid a^{*}b)a^{*}$$

o también

$$a(b^{*} \mid a^{*})ba^{*}$$

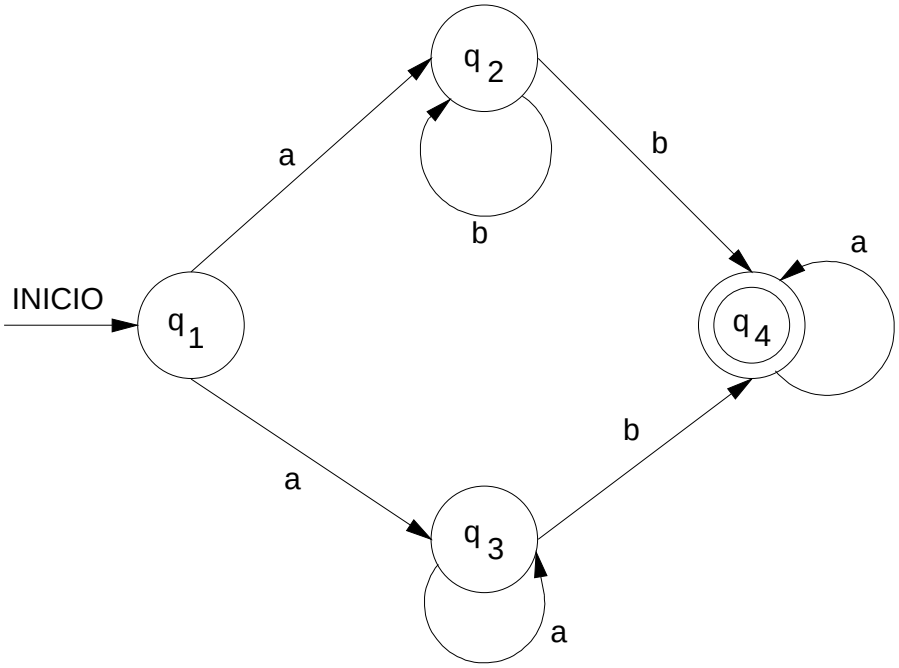


Fig. 19 : Diagrama de Moore del ejemplo 15.3.1.1.

15.3.2 Autómatas finitos deterministas

Un autómata finito determinista AFD es un caso particular de los autómatas finitos, en el que la función de transición no presenta ningúna ambigüedad en las transiciones de estados para una entrada dada.

Un autómata finito determinista es una quíntupla AFD=(E, Q, f, q_1, F) donde la función $f{:}E^* \times Q \rightarrow Q$ es determinista.

15.3.3 Teorema sobre la transformación de AFND en AFD

El teorema dice así : "Para todo autómata finito no determinista AFND=(E, Q, f, q_1, F) se puede construir un autómata finito determinista AFD=(E, Q', f', q'_1, F') tal que el lenguaje reconocido por el autómata finito determinista AFD coincida con el lenguaje reconocido por el autómata finito no determinista AFND, es decir L(AFD) = L(AFND)".

Demostración :

Se determina en primer lugar Q' que es el conjunto de las partes del conjunto de estados Q.

Q' = P(Q) = { conjunto de las partes de Q }

El cardinal de Q' o número de estados del conjunto Q' es :

$$\text{cardinal } (Q') = 2^{\text{cardinal}(Q)}$$

Al estado de Q' que corresponde a $\{q_a,q_b,...q_l\}$ se denotará por $[q_a,q_b,...q_l]$ es decir que se define f' de la forma :

$$f'(e,[q_a,q_b,...q_l])=[q_m,q_n,...q_k] \text{ si y sólo si } f(e,\{q_a,q_b,...q_l\})=\{q_m,q_n,...q_k\}$$

Es decir se calcula f'(e,q') aplicando f a cada estado q de los que figuran en q' y haciendo la unión de todos los resultantes.

$$q'_1=[q_1]$$

$$F'=\{q' \in Q'/q_f \in q \text{ y } q_f \in F\}$$

Es decir, para que q' sea estado final basta que uno o más de los estados de Q que lo componen sea final.

Con esto se ha construido un autómata finito determinista, ahora hace falta demostrar que reconocen el mismo lenguaje, para ello bastará comprobar que, para todo $x \in E^*$, $f'(x,q_1) \in F'$ si y solo si $f(x,q_1)$ contiene un estado (o varios) $q_f \in F$, y teniendo en cuenta la definición de F' esto será evidentemente cierto si se demuestra que :

$$f'(x,q'_1)=[q_a,...,q_l] \text{ si y sólo si } f(x,q_1)=\{q_a,...,q_l\}$$

Tal demostración puede hacerse por inducción sobre la longitud de x : para longitud de x nula, $x = \lambda$, es inmediato puesto que $f'(\lambda,q'_1)=q'_1=[q_1]$, y $f(\lambda,q_1)=\{q_1\}$. Supóngase que es cierto para longitud de x <= 1; entonces para $e \in E$ se tiene que

$$f'(xe,q'_1)=f'(x,q'_1)$$

Pero por hipótesis de inducción :

$$f'(x,q'_1)=[q_a,...,q_l] \text{ si y sólo si } f(x,q_1)=\{q_a,...,q_l\}$$

y por definición de f'

$$f'(e,[q_a,...,q_l])= [q_m,q_n,...q_k] \text{ si y sólo si } f(e,\{q_a,...,q_l\})=\{q_m,q_n,...q_k\}$$

Por tanto

$$f'(x,q'_1)=[q_m,...,q_k] \text{ si y sólo si } f(xe,q_1)=\{q_m,...,q_k\}$$

con lo que queda demostrado.

Ejemplo 15.3.3.1

Sea el autómata finito no determinista del ejemplo 15.3.1.1, determinar un autómata finito determinista equivalente.

Solución : Siguiendo la construcción del teorema 15.3.3, el AFD tendrá en un principio 2^4 estados, es decir Q' conjunto de las partes de Q tiene en un principio 16 estados. También se define el estado inicial y el conjunto de estados finales F'.

$Q'=\{\lambda,[q_1],[q_2],[q_3],[q_4],[q_1q_2],...,[q_1q_2q_3q_4]\}$

$q'_1=[q_1]$

$F'=\{[q_4],[q_1q_4],[q_2q_4],[q_3q_4],[q_1q_2q_4],...,[q_1q_2q_3q_4]\}$

y f' se construye a partir de f resultando la siguiente tabla :

	f'	*a*	*b*
	λ	λ	λ
	$[q_1]$	$[q_2q_3]$	λ
→	$[q_2]$	λ	$[q_2q_4]$
	$[q_3]$	$[q_3]$	$[q_4]$
	$[q_4]$	$[q_4]$	λ
→	$[q_1q_2]$	$[q_2q_3]$	$[q_2q_4]$
→	$[q_1q_3]$	$[q_2q_3]$	$[q_4]$
→	$[q_1q_4]$	$[q_2q_3q_4]$	λ
	$[q_2q_3]$	$[q_3]$	$[q_2q_4]$
	$[q_2q_4]$	$[q_4]$	$[q_2q_4]$
→	$[q_3q_4]$	$[q_3q_4]$	$[q_4]$
→	$[q_1q_2q_3]$	$[q_2q_3]$	$[q_2q_4]$
→	$[q_1q_2q_4]$	$[q_2q_3q_4]$	$[q_2q_4]$
→	$[q_1q_3q_4]$	$[q_2q_3q_4]$	$[q_4]$
→	$[q_2q_3q_4]$	$[q_3q_4]$	$[q_2q_4]$
→	$[q_1q_2q_3q_4]$	$[q_2q_3q_4]$	$[q_2q_4]$

Ahora bien, en un AF los estados que no son accesibles desde el inicial pueden eliminarse, así se eliminan los marcados en la tabla con flechas:

$[q_2]$, $[q_1q_2]$, $[q_1q_3]$, $[q_1q_4]$, $[q_1q_2q_3]$, $[q_1q_2q_4]$, $[q_1q_3q_4]$, y $[q_1q_2q_3q_4]$ por no aparecer dentro de la tabla.

[$q_2q_3q_4$] por no aparecer en la tabla como transición de un estado eliminado anteriormente.

[q_3q_4] por aparecer en la tabla como transición de un estado eliminado anteriormente, y también en su propio estado, no es accesible por no aparecer en otro estado.

Evidentemente [q_1] nunca puede eliminarse como estado, por ser el estado inicial.

Entonces f' puede resumirse según la tabla :

f'	*a*	*b*
λ	λ	λ
[q_1]	[q_2q_3]	λ
[q_3]	[q_3]	[q_4]
[q_4]	[q_4]	λ
[q_2q_3]	[q_3]	[q_2q_4]
[q_2q_4]	[q_4]	[q_2q_4]

El estado vacio [λ] no puede eliminarse en este caso, pues es accesible desde [q_1] y [q_4].

De los 16 estados posibles sólo han quedado 6, con los que se construye el diagrama de Moore de la figura 20.

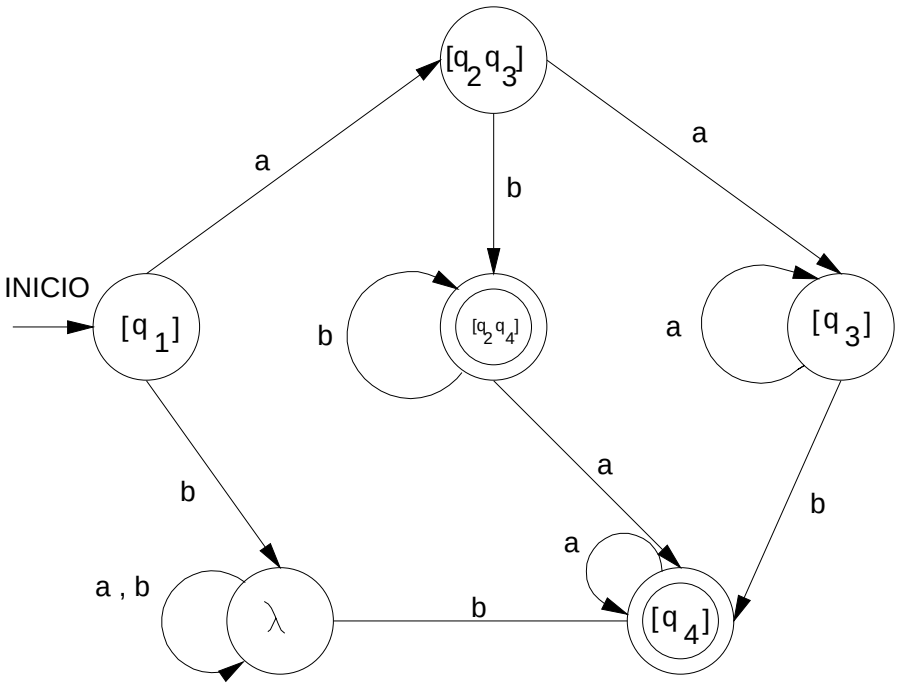


Fig. 20 : Diagrama de Moore del ejemplo 15.3.3.1.

Se puede comprobar que el lenguaje reconocido por el AFD es el mismo que el reconocido por el AFND del ejemplo 15.3.1.1. Así el lenguaje reconocido por el AFD es :

$$abb^* \mid abb^*aa^* \mid aaa^*ba^*$$

$$= abb^*(\lambda \mid aa^*) \mid aaa^*ba^*$$

$$= abb^*a^* \mid aaa^*ba^*$$

$$= ab^*ba^* \mid aaa^*ba^*$$

$$= a(b^* \mid aa^*)ba^*$$

$$= a(b^* \mid a^*)ba^*$$

La última igualdad se ha obtenido teniendo en cuenta que:

$$b^* \mid aa^* = b^* \mid \lambda \mid aa^* = b^* \mid a^*$$

15.4 Algoritmo de transformación de una gramática de tipo 3 en un autómata finito

Sea una gramática de tipo 3 o regular G=(VT,VN,S,P) y se desea obtener un autómata finito AF=(E,Q,f,q_1,F) que reconozca el lenguaje generado por la gramática. El autómata obtenido, en general será no determinista.

Solución: Se determinan los distintos elementos del autómata de la siguiente forma:

E=VT

Q=VN $\cup\{q_f\}$ A cada símbolo no terminal de la gramática se le asocia un estado del autómata. Además se introduce un nuevo estado, denominado q_f, que será el único estado final del autómata.

q_1=S

F={q_f}

La función de transición f se determina a partir de la forma de las reglas de producción P, de la manera siguiente:

a) Para reglas de la forma A $\rightarrow$ a B se obtiene f(A,a)=B siendo A y B los estados correspondientes a los no terminales A y B.

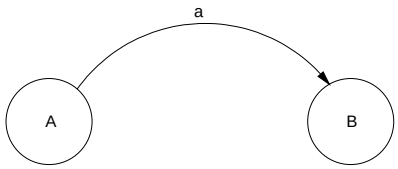


Fig. 21 : Diagrama de Moore de f(A,a)=B

b) Para reglas de la forma A $\rightarrow$ a se obtiene f(A,a)=q_f.

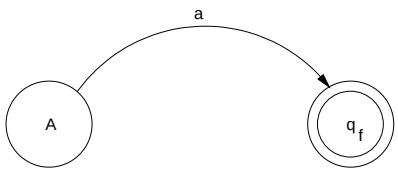


Fig. 22 : Diagrama de Moore de f(A,a)=q_f

Ejemplo 15.4.1

Sea la gramática de tipo 3 siguiente :

$$G = (VN=\{A,S\},\ VT=\{a,b\},\ S,\ P)$$

donde P son las reglas :

$S \rightarrow aS$

$S \rightarrow aA$

$A \rightarrow bA$

$A \rightarrow b$

Obtener un AFND y otro AFD que reconozcan el mismo lenguaje que esta gramática genera.

Solución : Se define el AFND a partir de la gramática como $AFND = (E, Q, f, q_1, F)$ donde E={*a*,*b*}, Q={A,S,X}, q_1=S, F={X}, y f viene dada por la tabla siguiente :

f	*a*	*b*
S	{S,A}	-
A	-	{A,X}
X	-	-

El diagrama de Moore se representa en la figura 23.

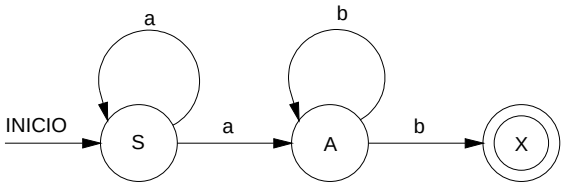


Fig. 23 : Diagrama de Moore del ejemplo 15.4.1

El lenguaje que reconoce :

$$a^*ab^*b = aa^*bb^* = a^+b^+$$

El AFD se define como AFD=(E, Q',f', q_1, F') donde f' viene dado por la tabla :

	f’	*a*	*b*
	λ	λ	λ
	[S]	[S,A]	λ
→	[A]	λ	[A,X]
→	[X]	λ	λ
	[S,A]	[S,A]	[A,X]
→	[S,X]	[S,A]	λ
	[A,X]	λ	[A,X]
→	[S,A,X]	[A,X]	[A,X]

Se eliminan de la tabla los estados inaccesibles desde el estado inicial [S], y se representa el diagrama de Moore de la figura 24.

El lenguaje que reconoce es :

$$a a^{*} b b^{*} = a^{+} b^{+}$$

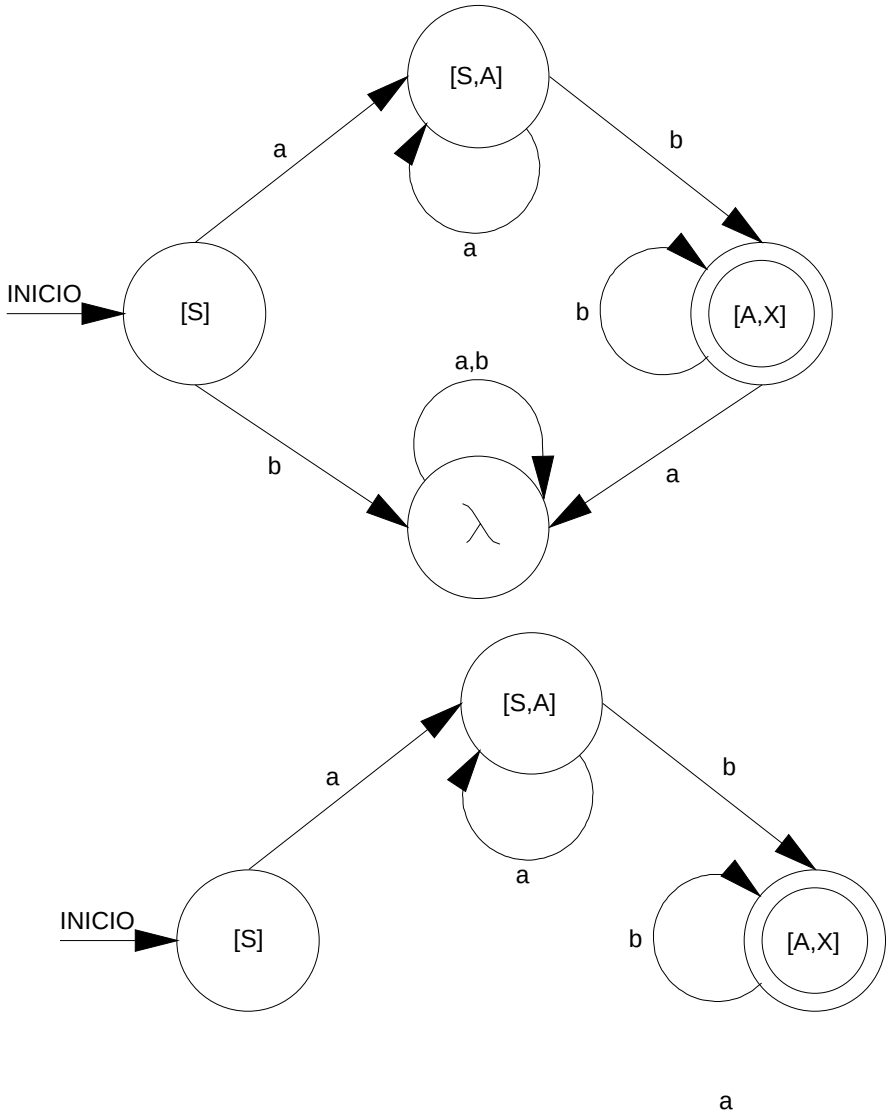


Fig. 24 : Diagrama de Moore AFD del ejemplo 15.4.1

15.5 Transformación de una expresión regular en un autómata finito

Dada una expresión regular existe un autómata finito capaz de reconocer el lenguaje que ésta define. Recíprocamente, dado un autómata finito, se puede expresar mediante una expresión regular el lenguaje que reconoce.

Para la transformación de una expresión regular en un autómata finito, se definirán en un principio las equivalencias entre las expresiones regulares básicas y sus autómatas finitos. Posteriormente se mostrará la construcción de Thompson que genera automáticamente un autómata finito a partir de una o varias expresiones regulares de cualquier complejidad.

15.5.1 Equivalencia entre expresiones regulares básicas y autómatas finitos

Se muestran equivalencias entre expresiones regulares simples y autómatas finitos expresados mediante un diagrama de Moore

15.5.1.1 Expresión regular λ

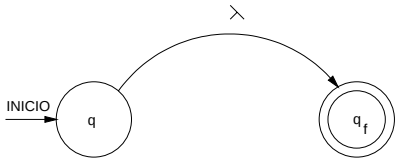


Fig. 25 : Diagrama de Moore de λ

15.5.1.2 Expresión regular *a*

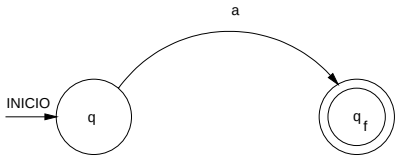


Fig. 26 : Diagrama de Moore de *a*

15.5.1.3 Expresión regular a^*

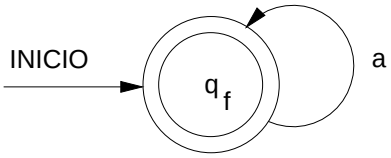


Fig. 27 : Diagrama de Moore de a^*

15.5.1.4 Expresión regular a^+

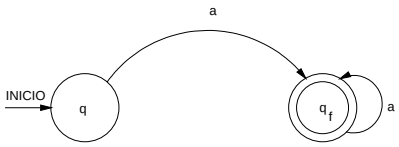


Fig. 28 : Diagrama de Moore de a^+

15.5.1.5 Expresión regular *a|b*

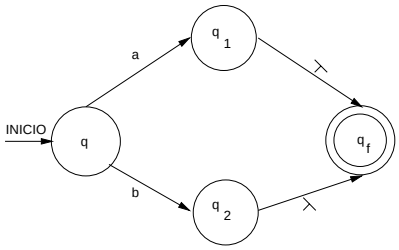


Fig. 29 : Diagrama de Moore de *a|b*

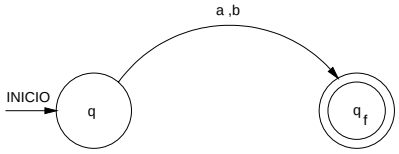


Fig. 30 : Otro diagrama de Moore de *a|b*

15.5.1.6 Expresión regular *(a|b)**

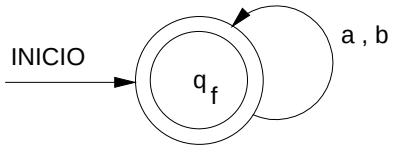


Fig. 31 : Diagrama de Moore de $(a|b)^*$

15.5.1.7 Expresión regular *(ac|b)**

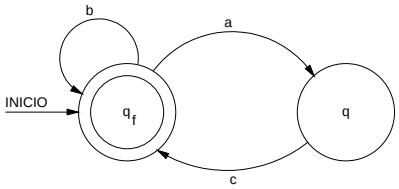


Fig. 32 : Diagrama de Moore de *(ac|b)**

15.5.1.8 Expresión regular *(acd|b)**

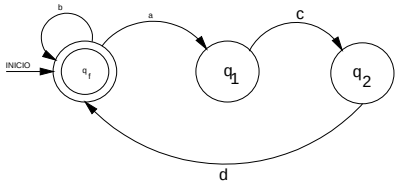


Fig. 33 : Diagrama de Moore de *(acd|b)**

15.5.2 Construcción de Thompson

La construcción de Thompson construye un AFND a partir de cualquier expresión regular. La herramienta *lex* utiliza esta construcción para obtener en sucesivos pasos un AFD (*Aho et al.*, 1986, capítulo 3).

Supongamos que N(s) y N(t) son AFND para las expresiones regulares s y t.

a) Para la expresión regular *s|t* se construye el AFND *N(s|t)* que se muestra en la figura 34.

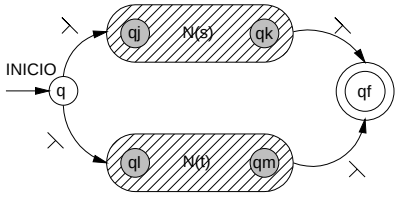


Fig. 34 : Construcción de Thompson para *N(s|t)*

b) Para la expresión regular *st* se construye el AFND *N(st)* que se muestra en la figura 35.

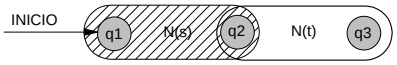


Fig. 35 : Construcción de Thompson para *st*

c) Para la expresión regular s^* se construye el AFND $N(s^*)$ que se muestra en la figura 36.

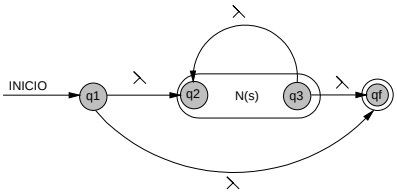


Fig. 36 : Construcción de Thompson para s^*

d) Para la expresión regular *(s)* se utiliza directamente *N(s)*

Ejemplo 15.5.2.1

Utilizando la construcción de Thompson construir un autómata finito AF que reconozca la expresión regular $(0|1)^*\ 0\ (0|1)\ (0|1)$

Solución: Se descompone la expresión regular en subexpresiones, siguiendo la precedencia de operadores, tal y como se muestra en la figura 37.

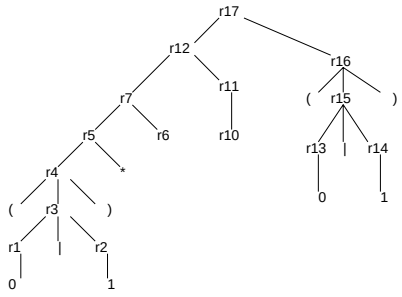


Fig. 37 : Descomposición sintáctica de la expresión regular

Comenzando por r1, r2,... r7 se llega que el AFND de la expresión r7 es el representado en la figura 38.

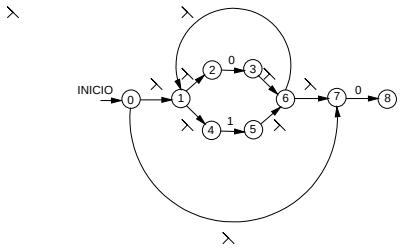


Fig. 38 : Construcción de Thompson para r_7

Continuando se puede observar que para la expresión r17 el AFND es el que se presenta en la figura 39.

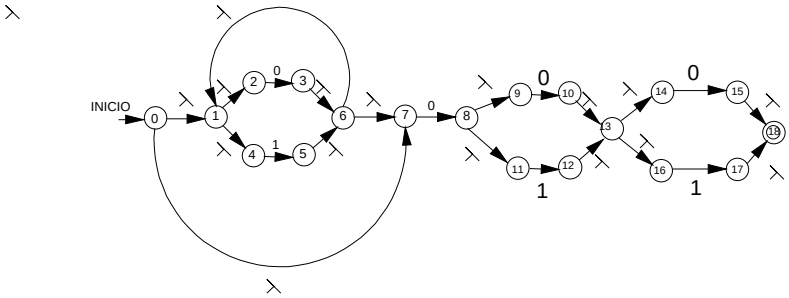


Fig. 39 : Construcción de Thompson para (0|1)*0(0|1)(0|1)

15.6 Minimización de estados de un AFD

Se definió anteriormente que dos AF son equivalentes si reconocen el mismo lenguaje. En este apartado se trata de encontrar un AF equivalente en forma mínima. Es importante poder saber si un AF está en forma mínima, y si no lo está hallar un AF equivalente en forma mínima. Se puede demostrar que siempre existe una forma mínima de un AF (*Hopcroft y Ullman, 1979, pp. 67-68*). Para determinar el AF en forma mínima no es preciso realizar infinitos ensayos con cadenas de entrada. Existen algoritmos para minimizar AF, es decir, hallar otro AF en forma mínima equivalente.

Algoritmo 15.6.1

Entrada. Un autómata finito determinista A=(E,Q,f,q_0,F).

Salida. Un autómata finito determinista A'=(E,Q',f',q'_0,F') que acepta el mismo lenguaje que A, y que tiene el menor número de estados posible.

Metodo. El algoritmo para minimizar el número de estados de un AFD funciona encontrando todos los grupos de estados que pueden ser diferenciados por una cadena de entrada. Cada grupo de estados que no puede diferenciarse se fusiona entonces en un único estado. El algoritmo opera manteniendo y refinando una partición del conjunto de estados. Cada grupo de estados dentro de la partición está formado por estados que aún no han sido distinguidos unos de otros, y todos los pares de estados escogidos de entre grupos diferentes han sido considerados distinguibles por una entrada.

1. Se construye una partición inicial Π del conjunto de estados Q en dos grupos : los estados finales F y los estados no finales Q-F.

2. Se determina una nueva partición Π_{nueva} a partir de la partición anterior Π, con el siguiente procedimiento :

FOR cada grupo G de Π **DO**

BEGIN
Partición de G en subgrupos tales que dos estados q_i y q_j de G están en el mismo subgrupo si, y sólo si, para todos los símbolos de entrada *e*, los estados q_i y q_j tienen transiciones en *e* hacia estados del mismo grupo de Π;

/* en el peor caso, un estado estará sólo en un subgrupo */

sustituir G en Π_{nueva} por el conjunto de todos los subgrupos formados;
END

3. Se realizan las siguientes comprobaciones.

IF $\Pi_{nueva}=\Pi$ **THEN**

BEGIN
$\Pi_{final}:=\Pi$;
GOTO 4; /* Ir al paso 4 */
END

ELSE

BEGIN
$\Pi := \Pi_{nueva}$;
GOTO 2; /* Volver al paso 2 */
END

4. Se escoge un estado en cada grupo de la partición Π_{final} como *representante* de este grupo.

Los representantes seran los estados reducidos Q' de A'. Sea un estado q_i representante, y sea una entrada *a* que produce una transición de q_i a q_j en A. Sea q_k el representante del grupo de q_j (q_k puede ser q_j). Entonces A' tiene una transición desde q_i a q_k con la entrada *a*. Sea el estado inicial q'_0 de A' el representante del grupo que contiene al estado inicial q_0 de A, y sean los estados finales F' de A' los representantes que están en F. Se puede observar que cada grupo de Π_{final} consta únicamente de estados en F o no tiene ningún estado en F.

5. Se eliminan los estados pasivos q_p de Q', es decir estados que no son finales, y que tienen transiciones hacia ellos, pero no desde ellos hacia otros. Todas las transiciones a q_p desde otros estados se convierten en indefinidas. También se eliminan los estados inaccesibles de Q', es decir todos los estados que no se pueden alcanzar desde el estado inicial. Se obtiene Q' y la función f'.

Ejemplo 15.6.2

Sea el autómata finito determinista A=(E,Q,f,q_1,F), donde E={*a*,*b*}, Q={q_1,q_2,q_3,q_4,q_5}, F={q_5}, y f viene dada por la tabla :

f	*a*	*b*
q_1	q_2	q_3
q_2	q_2	q_4
q_3	q_2	q_3
q_4	q_2	q_5
q_5	q_2	q_3

Se desea determinar un autómata equivalente con un número de estados mínimo.

Solución

(1).La partición inicial Π consta de dos grupos : (q_5) conjunto de estados finales, y ($q_1q_2q_3q_4$) los estados no finales.

(2). Se aplica el procedimiento de partición a cada grupo. El grupo (q_5) consta de un sólo estado, y no se puede dividir más, se coloca directamente en Π_{nueva}. El otro grupo ($q_1q_2q_3q_4$), con la entrada *a* tiene una transición a q_2, así que todos podrían permanecer en un mismo grupo en lo que a la entrada *a* se refiere. Sin embargo, con la entrada *b*, q_1, q_2, y q_3 van a miembros del grupo ($q_1q_2q_3q_4$) de Π mientras que q_4 va al grupo (q_5). Por lo tanto Π_{nueva}, tiene tres grupos :($q_1q_2q_3$), (q_4) y (q_5).

(3). Como Π_{nueva} no es igual a Π, se repite el paso (2) con $\Pi:=\Pi_{nueva}$.

(2). Se aplica el procedimiento de partición a los grupos de Π que constan de más de un estado. Para una entrada *a* en el grupo ($q_1q_2q_3$), no hay división, sin embargo para la entrada *b* se produce una división en (q_1q_3) y (q_2), puesto que (q_1q_3) tienen una transición a q_3, mientras que q_2 tiene una transición a q_4, que es miembro de un grupo distinto de ($q_1q_2q_3$). Por lo tanto Π_{nueva}, tiene cuatro grupos :(q_1q_3), (q_2), (q_4) y (q_5).

(3). Como Π_{nueva} no es igual a Π, se repite el paso (2) con $\Pi:=\Pi_{nueva}$.

(2). Se aplica el procedimiento de partición a los grupos de Π que constan de más de un estado. La única posibilidad es intentar dividir (q_1q_3). Sin embargo, q_1 y q_3 van al mismo estado q_2 para la entrada *a*, y al mismo estado q_3 para la entrada *b*. Por lo tanto Π_{nueva}, tiene cuatro grupos :(q_1q_3), (q_2), (q_4) y (q_5).

(3). Como Π_{nueva} es igual a Π, se va al paso (4) con $\Pi_{final}:=\Pi$.

(4). Se escoge q_1 como representante del grupo (q_1q_3), y q_2, q_4 y q_5 como representantes de los grupos de un sólo estado.

(5). El automata con un número de estados mínimo A=(E,Q',f',q_1,F), donde E={a,b}, Q={q_1,q_2,q_4,q_5}, F={q_5}, y f' viene dada por la tabla :

f'	*a*	*b*
q_1	q_2	q_1
q_2	q_2	q_4
q_4	q_2	q_5
q_5	q_2	q_1

En el autómata reducido A', el estado q_5 tiene una transición al estado q_1 con la entrada *b*, puesto que q_1 es el representante del grupo (q_1q_3) y hay una transición de q_5 a q_3 con la entrada *b* en el autómata original. Una modificación similar se realiza para el estado q_1 y la entrada *b*. Todas las demás transiciones están tomadas de *f*. No hay ningún estado pasivo, y todos los estados son accesibles desde el estado inicial q_1.

Se puede observar que el lenguaje reconocido por los autómatas A y A' es $(a|b)^*abb$.

CAPÍTULO 16: EJERCICIOS RESUELTOS

Ejercicio 16.1

Dada la gramática G=(VN={S,A},VT={0,1},S,P) donde P son las producciones :

$S \rightarrow 0A$

$A \rightarrow 0A$

$A \rightarrow 1S$

$A \rightarrow 0$

Determinar :
a) ¿De qué tipo es?
b) Expresar de algún modo el lenguaje que genera.

Solución :

a) Es una gramática de tipo 3, donde las reglas de producción son de la forma : $A \rightarrow aB$ o también $A \rightarrow a$, donde A,B pertenecen al VN y a pertenece al VT.

b) Se generan algunas sentencias del lenguaje :

$S \rightarrow 0A \rightarrow 00A \rightarrow 001S \rightarrow 0010A \rightarrow 00100$
$S \rightarrow 0A \rightarrow 01S \rightarrow 010A \rightarrow 0100$

L(G) = { Cadena que empieza por 0 y termina con dos 0. Entre ellos puede haber un nº indefinido de 0 y 1, con la restricción de que no puede haber dos 1 seguidos}

Ejercicio 16.2

Definir una gramática que permita generar todos los números racionales escritos en decimal con el formato :

<signo> <parte entera> . <parte fraccionaria>

Construir un autómata que reconozca dichos números.

Solución : Sea la gramática G = (VN={<racional>, <cadena dígitos>, <dígito>, <signo>, <vacio>}, VT={0,1,2,3,4,5,6,7,8,9,+,-}, S=<racional>, P), donde las reglas de producción P son las siguientes :

<racional> → <signo><cadena dígitos>.<cadena dígitos>

<cadena dígitos> → <dígito><cadena dígitos>

<cadena dígitos> → <dígito>

<dígito> → 0|1|2|3|4|5|6|7|8|9

<signo> → +|-|<vacio>

<vacio> → λ

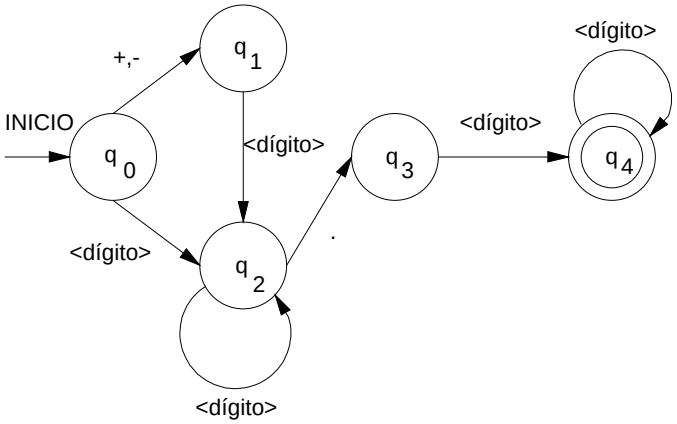


Fig. 40 : Solución del ejercicio 16.2

Ejercicio 16.3

Definir una gramática que permita generar identificadores, es decir secuencias de letras y dígitos que empiezan siempre por una letra.

Solución : Sea la gramática G=(VN={<identificador>,<letra>,<resto identificador>, <dígito>, <vacio>}, VT={a,b,...,A,...,Z,0,1,...,9}, S=<identificador>, P), donde P son las producciones :

<identificador> → <letra><resto identificador>

<letra> → a | b | c | ... | z | A | B | C | ... |Z

<resto identificador> → <dígito> <resto identificador>

<resto identificador> → <letra> <resto identificador>

<resto identificador> → <letra>

<resto identificador> → <dígito>

<resto identificador> → <vacio>

<dígito > → 0 | 1 | 2 | 3 | ... |9

<vacio> → λ

Ejercicio 16.4

Escribir una gramática que defina el lenguaje $L = \{a b^n a / n \geq 0\}$.

Solución : Sea G=(VN={S,B}, VT={a,b}, S, P) y las producciones P de la gramática son :

S → aB

B → bB

B → a

Ejercicio 16.5

Escribir una gramática libre de contexto para un número real del lenguaje C++, y escribir las derivaciones, para las siguientes sentencias:

(a) 251.
(b) -61.
(c) -75.25
(d) 1.73
(e) 14.25E+02
(f) -25.2E-23
(g) .23E12
(h) .23

Solución :

<Tipo REAL> → <signo> <número> <exp>

<Tipo REAL> → <signo> <número> <exponente>

<Tipo REAL> → <signo> <fracción> <exponente>

<Tipo REAL> → <signo> <número> <fracción> <exponente>

<signo> → +|-|<vacio >

<número> → <dígito><número>|<dígito>

<fracción> → .< número >

<exp> → E <signo> <número>

<exponente> → <exp> | <vacio>

<dígito> → 0|1|2|3|4|5|6|7|8|9

<vacio> → λ

b) -61.

< Tipo REAL > → <signo> <num> . <exponente>
→ - < num > . < exponente >
→ - < dígito > < num > . < exponente >
→ - < dígito > < num >.

→ - 6 < num >.
→ - 6 < dígito >.
→ - 61.

c) -72.25

< Tipo REAL > → < signo > < num > < fracción > < exponente >
→ < signo > < num > < fracción >
→ - < num > < fracción >
→ - < dígito > < num > < fracción >
→ - 7 < num > <fracción >
→ - 7 < dígito > < fracción >
→ - 72 < fracción >
→ - 72. < num >
→ - 72. < dígito > < num >
→ - 72.2 < num >
→ - 72.2 < dígito >
→ - 72.25

e) 14.25E+02

< Tipo REAL > → < signo > < num > < fracción > < exponente >
→ < num > < fracción > < exponente >
→ < dígito > < num > < fracción > < exponente >
→ < num > < fracción > < exponente >
→ 1 < dígito > < fracción > < exponente >
→ < fracción > < exponente >
→ 14. < num > < exponente >
→ 14. < dígito > < num > < exponente >
→ 14.2 < num > < exponente >
→ 14.25 < exponente >
→ 14.25 < exp >
→ 14.25E < signo > < num >
→ 14.25E+ < num >
→ 14.25E+ < dígito > < num >
→ 14.25E+0 < num >
→ 14.25E+0 < dígito >
→ 14.25E+02

f) -25.2E-23

< Tipo REAL > → < signo > < num > < fracción > < exponente >
→ - < num > < fracción > < exponente >
→ - < dígito > < num > < fracción > < exponente >
→ -2 < num > < fracción > < exponente >
→ -2 < dígito > < fracción > < exponente >
→ -25 < fracción > < exponente >
→ -25. < num > < exponente >
→ -25. < dígito > < exponente >
→ -25.2 < exponente >
→ -25.2 < exp >
→ -25.2E < signo > < num >
→ -25.2E- < num >
→ -25.2E- < dígito > < num >
→ -25.2E-2 < num >
→ -25.2E-2 < dígito >
→ -25.2E-23

Ejercicio 16.6

Escribir una gramática libre de contexto para la sentencia de FORTRAN FORMAT. Supongase que no se admiten constantes definidas con H (Hollerith). Dar las derivaciones de las sentencias.

(a) 10 FORMAT (I7, F5.1)
(b) 200 FORMAT (E12.5)
(c) 99015 FORMAT (I5/F7.1)
(d) 40 FORMAT (1X,I4,3X,F7.2)
(e) 50 FORMAT (I3,2(F7.2))

Solución :

< sentencia FORMAT > → < etiqueta > < blanco > {< blanco >} FORMAT {<blanco>} ({< blanco >} < argumento > {< blanco >})
< etiqueta > → < dígito > < número1 >
< número1 > → < dígito > < número2 > | < vacio >
< número2 > → < dígito > < número3 > | < vacio >

< número3 > → < dígito > < número4 > | < vacio >
< número4 > → < dígito > | < vacio >
< dígito > → 0 | 1 | 2 | .. | 9
< blanco > → ƀ
< argumentos > → < especificación > < resto argumentos >
< resto argumentos > → , < especificación > < resto argumentos >
< resto argumentos > → <diagonal> <especificación> <resto argumentos>
< resto argumentos > → < vacio >
< diagonal > → / < otra diagonal >
< otra diagonal > → < diagonal >
< otra diagonal > → < vacio >
< especificación > → < número > (< argumentos >)
< especificación > → < formato >
< formato > → I < número real >
< formato > → I < número >
< formato > → Q
< formato > → A < número >
< formato > → A
< formato > → R < número >
< formato > → L < número >
< formato > → K < número >
< formato > → F < número real >
< formato > → Z < número >
< formato > → E < número real >
< formato > → E < número real > E.
< formato > → D < número real >
< formato > → D < número real > E.
< formato > → G < número real >
< formato > → G < número real > E.
< formato > → T < número >
< formato > → TL < número >
< formato > → TR < número >
< formato > → < número > X
< formato > → < número > P
< formato > → S

< formato > $\rightarrow$ SP

< formato > $\rightarrow$ SS

< formato > $\rightarrow$ BN

< formato > $\rightarrow$ BZ

< número real > $\rightarrow$ < número > . < número >

< número > $\rightarrow$ < dígito > < número >

< número > $\rightarrow$ < vacio >

< vacio > $\rightarrow \lambda$

CAPÍTULO 17: EJERCICIOS PROPUESTOS

Ejercicio 17.1

Construir una gramática que describa el lenguaje constituido por los números romanos. Diseñar un autómata que dada una cadena de entrada indique si es un número romano o no lo es.

Ejercicio 17.2

Construir una gramática que describa el lenguaje constituido por todas las cadenas de a y b, que no contienen la subcadena abb. Se entiende por subcadena de una cadena s como una cadena que se obtiene suprimiendo cero o más símbolos desde la derecha o la izquierda de la cadena s. Ejemplo : *ver* es una subcadena de *conversión*. Sin embargo *cosión* no es una subcadena de *conversión*.

Ejercicio 17.3

Construir una gramática que describa el lenguaje constituido por todas las cadenas de a y b, que no contienen la subsecuencia abb. Se entiende por subsecuencia de una cadena s como una cadena que se obtiene suprimiendo cero o más símbolos no necesariamente contiguos de la cadena s. Ejemplo : *ovesi* es una subsecuencia *conversión*.

Ejercicio 17.4

Construir un AFD con mínimo de estados para que reconozca el lenguaje descrito por la expresión regular $(0|1)^*0(0|1)(0|1)$.

Ejercicio 17.5

Escribir un ejemplo de lenguaje, gramática y autómata de tipo 0, pero que no sea de tipo 1, 2 o 3. No incluido en este texto.

Ejercicio 17.6

Escribir un ejemplo de lenguaje, gramática y autómata de tipo 1, pero que no sea de tipo 2 o 3. No incluido en este texto.

Ejercicio 17.7

Escribir un ejemplo de lenguaje, gramática y autómata de tipo 2, pero que no sea de tipo 3. No incluido en este texto.

Ejercicio 17.8

Escribir un ejemplo de lenguaje, gramática y autómata de tipo 3. No incluido en este texto.

Ejercicio 17.9

Escribir un autómata que reconozca si un número es un real válido en Pascal estándar o no.

Ejercicio 17.10

Diseñar una gramática que genere el lenguaje formado por las expresiones con o sin paréntesis, debiendo estar siempre los paréntesis emparejados. Los símbolos terminales son VT={(,e,)}. Ejemplos de instrucciones del lenguaje : *e, ee, (e)(e), (((e))), ((e)(e))*, y *(eee)*. Construir un autómata que reconozca dicho lenguaje.

CAPÍTULO 18: EJERCICIOS DE PROGRAMACIÓN

Ejercicio 18.1

Escribir un programa que simule el funcionamiento de una máquina de Turing genérica. El programa pedirá al usuario bien por teclado o por fichero los componentes de la definición de la máquina de Turing reconocedora de un lenguaje de tipo 0. Dada una cadena de entrada indicará si pertenece o no al lenguaje.

Ejercicio 18.2

Escribir un programa que simule el funcionamiento de un autómata lineal acotado genérico. El programa pedirá al usuario bien por teclado o por fichero los componentes de la definición del autómata lineal acotado reconocedor de un lenguaje de tipo 1. Dada una cadena de entrada indicará si pertenece o no al lenguaje.

Ejercicio 18.3

Escribir un programa que simule el funcionamiento de un autómata de pila genérico. El programa pedirá al usuario bien por teclado o por fichero los componentes de la definición del autómata de pila reconocedor de un lenguaje de tipo 2. Dada una cadena de entrada indicará si pertenece o no al lenguaje.

Ejercicio 18.4

Escribir un programa que simule el funcionamiento de un autómata finito genérico. El programa pedirá al usuario bien por teclado o por fichero los componentes de la definición del autómata finito reconocedor de un lenguaje de tipo 3. Dada una cadena de entrada indicará si pertenece o no al lenguaje.

Ejercicio 18.5

Escribir un programa que tome como entrada la definición de un autómata finito no determinista y escriba como salida un autómata finito determinista equivalente.

Ejercicio 18.6

Escribir un programa que dado un autómata finito determine otro equivalente con un número de estados mínimo.

BIBLIOGRAFÍA

(1) Aho A.V. y Ullman J.D. (1973b). *The Theory of Parsing, Translation and Compiling. Vol I: Parsing*. Prentice-Hall.

(2) Aho A.V. y Ullman J.D. (1973a). *The Theory of Parsing, Translation and Compiling. Vol II: Compiling*. Prentice-Hall.

(3) Aho A.V., Sethi R. y Ullman J.D (1986). *Compilers : Principles, Techniques, and Tools*. Addison-Wesley. Edición en Castellano (1990) *Compiladores : Principios, Técnicas y Herramientas*. Addison-Wesley Iberoamericana.

(4) Alfonseca M., Sancho J. y Martínez Orga M. (1987). *Teoría de lenguajes, gramáticas y autómatas.* Ediciones Universidad y Cultura.

(5) Alvarez-Uría Alvarez, E. (1974). *Algoritmos de Markov y lenguajes de programación*. Dto. de Matemáticas. Escuela Técnica Superior de Ingenieros de Minas. Universidad de Oviedo.

(6) Barber F., Botti V.J. y Pérez T.A. (1986). *Introducción a los traductores, compiladores e intérpretes.* Dto. de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia.

(7) Chomsky N. (1956). Three models for the description of language. *IRE Trans. on Information Theory* 2:3, pp 113-124.

(8) Chomsky N. (1959). On certain formal properties of grammars. *Information and Control 2:2, pp. 137-167.*

(9) Chomsky N. (1962). Context-free grammars and pushdown storage. *Quaterly Prog. Rept. No. 65, pp. 187-194, MIT REs. Lab. Elect.,* Cambridge, Mass.

(10) Chomsky N. (1963). Formal properties of grammars. *Handbook of Math. Psych.*, Vol. 2, pp. 323-418, John Wiley and Sons.

(11) Chomsky N, y Miller G.A. (1958). Finite state languages. *Information and Control 1:2, pp 91-112.*

(12) Chomsky N., y M.P. Schutzenberger (1963). The algebraic theory of context free languages. *Computer Programming and Formal Systems,* pp. 118-161, North Holland.

(13) Conde Sánchez C. (1969). *Máquinas de Turing y computabilidad*. Discurso inagural del año académico 1969-70 en la Universidad de Oviedo. Dto. de Matemáticas. Escuela Técnica Superior de Ingenieros de Minas. Universidad de Oviedo.

(14) Cueva Lovelle J.M. (1998). *Conceptos básicos de Procesadores de Lenguaje*. Cuaderno Didáctico de Ingeniería Informática nº10. SERVITEC.

(15) Fernández G. y Sáez Vacas F. (1987). *Fundamentos de informática*. Alianza.

(16) Hopcroft J.E. y Ullman J.D. (1979). *Introduction to automata theory, languages and computation*. Addison-Wesley.

(17) Hopcroft J.E. (1984). Máquinas de Turing. *Investigación y Ciencia nº 94, pp 8-19, Julio 1984*.

(18) Isasi P., Martínez P., Borrajo D. *Lenguajes, Gramáticas y Autómatas. Un enfoque práctico.* Addison-Wesley (1997)

(19) Katrib Mora M. (1988). *Lenguajes de programación y técnicas de compilación*. Editorial Pueblo y Educación (Cuba).

(20) Minsky M.L. (1967). *Computation: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs.

(21) Sánchez Dueñas G. y Valverde Andreu J.A. (2ª Edición, 1989). *Compiladores e intérpretes. Un enfoque pragmático*. Ed. Díaz de Santos.

(22) Sanchís Llorca F.J. y Galán Pascual C. (1986). *Compiladores. Teoría y construcción*. Paraninfo.

(23) Shannon C.E. (1949). The synthesis of two-terminal switching circuits. *Bell System Tech. J.*, vol. 28, pp. 59-98.

(24) Shannon C.E. (1954). *A symbolic analysis of relay and switching circuits*. Van Nostrand.

(25) Shannon C.E. (1956). A universal Turing machine with two internal states. *Automata Studies*, pp. 129-153, Princeton Univ. Press.

(26) Turing A.M. (1936). On computable numbers with an application to the Entscheidungs-problem. *Proc. London Math. Soc.* . 2:42, pp 230-265. Una corrección *ibid*, 43, pp- 544-546.

Índice

Títulos de la Colección

Ingeniería Informática

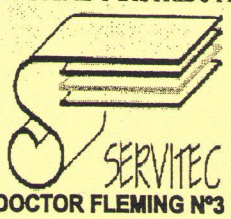
Nº CUADERNO	TITULO	ISBN
1	ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS	84-8497-802-8
2	DISEÑO DE PÁGINAS WEB USANDO HTML	84-8497-803-6
3	FORMATOS GRÁFICOS	84-8497-804-4
4	PROGRAMACIÓN EN LENGUAJE C	84-8497-805-2
5	ADMINISTRACIÓN DE WINDOWS N 4.0	84-8497-899-0
6	GUÍA DE USUARIO DE DERIVE PARA WUIDOWS	84-8497-919-9
7	GUÍA DE REFERÊNCIA DE MULTIBASE COSMOS	84-8416-001-7
8	GESTIÓN DE UN TALLER MULTIBASE CÓSMOS	84-8416-034-3
9	EJERCICIOS DE LÓGICA INFOMÁTICA	84-8416-357-1
10	CONEPTOS BÁSICOS DE PROCESADORES DE LENGUAJE	54-8416-889-1
11	INTRODUCCIÓN A LA ADMINISTRACIÓN DE UNIX	84-8416-570-1
12	LÓGICA PROPOSICIONAL PARA LA INFOMÁTICA	84-8416-613-9
13	PROGRAMACIÓN PRÁCTICA EN PROLOG	84-8416-612-0
14	LA HERRAMIENTA CASE META COSMOS	84-699-0054-4
15	GUIA DE LENGUAJE COOL DE MULTIBASE COSMOS	84-699-0053-6
16	GUÍA DE REFERÊNCIA PROGRESS	84-699-2083-9
17	GESTIÓN DE DEPOSITO DENTAL CON PROGRESS	84-699-2082-0
18	GUIA DE DISEÑO Y CONS. REPOSICIÓN MUL. COSMOS	84-699-2081-2
19	GESTIÓN Y ADMIN. DE UN COLEGIO MAYOR MUL-COSMOS	84-699-2080-4
20	MODELADO DE SOFWARE EN UML	84-699-2079-0
21	EL LENGUAJE DE PROGRAMACIÓN JAVA	84-699-3880-0
22	PRINCIPIOS DE ALGORITMIA	84-699-6537-9
23	LISTAS, PILAS Y COLAS	84-699-6536-0
24	RECURSIVIDAD	84-699-6535-2
25	ARBOLES	84-699-6849-1
26	CONJUNTOS Y TABLAS DE DISPERSIÓN	84-699-7398-3
27	ALGORITMOS DE ORDENACIÓN	84-699-7397-5
28	TEORÍA DE GRAFOS	84-699-8362-8
29	IINTR. AL PROCES. EFEC. DE TEXTOS CON MICROSOFT WORD	84-699-9618-5
30	ORACLE SQL	84-688-0420-7
31	TÉCNICAS DE DISEÑO DE ALGORITMOS	84-688-1764-3
32	LA PLATAFORMA . NET	84-688-3449-1
33	EJERC. DE HOJAS DE CÁL. EXCEL APLICADOS A LA GES.EMPRES.	84-688-3450-5
34	TIPOS ABSTRACTOS DE DATOS	84-688-4209-5
35	INTERPRETES T DISEÑO DE LENGUAJES DE PROGRAMACIÓN	84-688-4210-9
36	LENGUAJES Y AUTOMATAS EN PROCESADORES DE LENGUAJE	84-688-4211-7
37	METODOLOGIA DE LA PROGRAMACIÓN: GUIA DEL ALUMNO	84-688-5901-4
38	ANÁLISIS SEMÁNTICO EN PROCESADORES DE LENGUAJE	84-688-6208-8
39	ARQUITECTURA WEB EN APLICACIONES JAVA/J2EE	84-688-6580-9
40	ALGORITMICA CON FORTRAN 90	84-688-7250-4
41	TABLAS DE SÍMBOLOS EN PROCESADORES DE LENGUAJES	84-688-7631-3
42	INTRODUCCIÓN A LA COMINICACIÓN PERSONA MÁQUINA	84-688-8362-X
43	INTRODUC. A LA PROG. ORIENTADA A OBJETOS CON JAVA	84-688-8826-6
44	DESARR. APLIC. EN SISTEM. DISTRIBUIDOS E INTERNET	84-689-3379-1
45	LÓGICA DE PREDICADOS	84-689-3380-5
46	LENGUAJE C#	84-689-5893-X
47	INTEGRACIÓN DE APLICACIONES OFIMÁTICAS	84-689-5892-1
48	INFORMÁTICA GENERAL	84-689-7033-6
49	PROYECTOS INFORMÁTICOS	

IMPRIME Y DISTRIBUYE

C/DOCTOR FLEMING Nº3
33005 OVIEDO
TLF-FAX 985 250581
E-mail:copisteriaservitec@fade.es

Consultor Editorial
Juan Manuel Cueva Lovelle
cueva@lsi.uniovi.es

IMPRIME Y DISTRIBUYE



C/DOCTOR FLEMING Nº3
33005 OVIEDO
TLF-FAX 985 250581
www.fade.es/copisteriaservitec
E-mail:copisteriaservitec@fade.es