PRINCIPIOS DE AUTOMATAS FINITOS

Guillermo Morales-Luna
Sección de Computación
CINVESTAV-IPN
gmorales@cs.cinvestav.mx

27 de junio de 2000

Contenido

Capítulo 0

Introducción a la teoría de autómatas

0.1 Gramáticas formales

Las gramáticas formales son sistemas de manipulación simbólica que permiten generar cadenas de símbolos, llamadas por esto *bien formadas*, o bien reconocer cuándo una cadena dada está, en efecto, bien formada. En este primer capítulo, meramente introductorio, ilustraremos con varios ejemplos estos tipos de sistemas e inclusive algunos otros sistemas un tanto más generales.

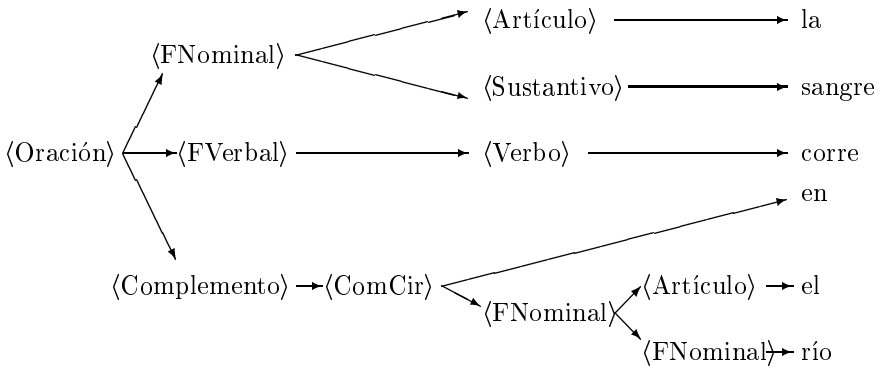
0.1.1 Una gramática sencilla del castellano

Presentamos en las figuras (1) y (2) una gramática muy sencilla del español siguiendo la conocida representación de los *diagramas sintácticos*.

Y, equivalentemente, en la tabla (1) la presentamos en su *Forma Normal de Backus*:

Como ejemplos de cadenas bien formadas derivadas en esta gramática están los siguientes:

1. *la sangre corre en el río*



2. *Cervantes escribió "El Quijote" para gloria de las letras españolas en una cárcel.*

No es difícil ver que efectivamente esta cadena se deriva con las reglas descritas.

0.1.2 Otras gramáticas formales

Cadenas de 1's separadas por 0's únicos Consideremos las siguientes dos gramáticas:

$$a)\quad \begin{array}{lcl} S & \to & 1S|1T|1 \\ T & \to & 0S \end{array} \qquad b)\quad \begin{array}{lcl} S & \to & 0T|1S|1 \\ T & \to & 0U|1S|1 \\ U & \to & 0U|1U \end{array}$$

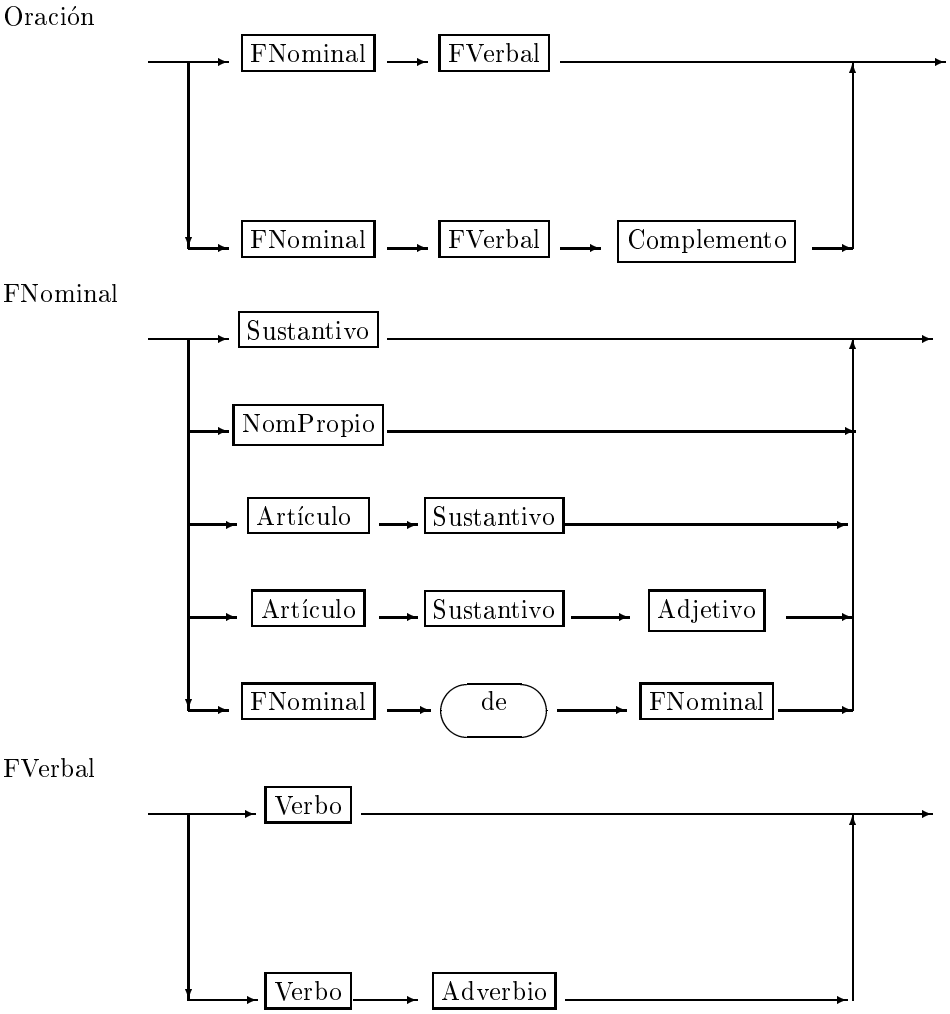


Figura 1: Una gramática sencilla de castellano.

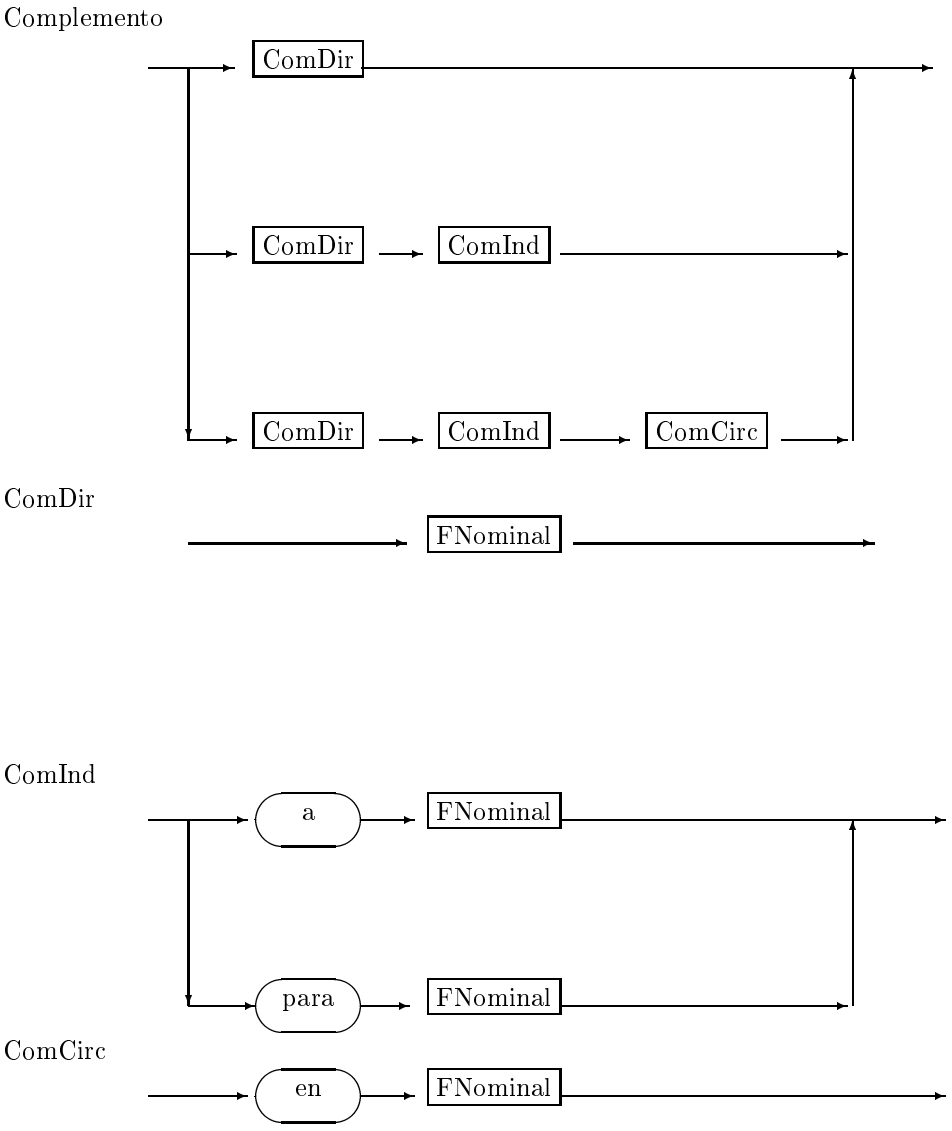


Figura 2: Una gramática sencilla de castellano (cont').

⟨Oración⟩	::=	⟨FNominal⟩⟨FVerbal⟩|
		⟨FNominal⟩⟨FVerbal⟩⟨Complemento⟩
⟨FNominal⟩	::=	⟨Sustantivo⟩|
		⟨NomPr⟩|
		⟨Artículo⟩⟨Sustantivo⟩|
		⟨Artículo⟩⟨Sustantivo⟩⟨Adjetivo⟩|
		⟨FNominal⟩de⟨FNominal⟩
⟨FVerbal⟩	::=	⟨Verb⟩|
		⟨Verb⟩⟨Adverbio⟩
⟨Complemento⟩	::=	⟨ComDir⟩|
		⟨ComInd⟩|
		⟨ComCirc⟩|
		⟨ComDir⟩⟨ComInd⟩⟨ComCirc⟩
⟨ComDir⟩	::=	⟨FNominal⟩
⟨ComInd⟩	::=	a⟨FNominal⟩|
		para⟨FNominal⟩|....
⟨ComCir⟩	::=	en⟨FNominal⟩|
		desde⟨FNominal⟩|
		cuando⟨FNominal⟩|...
⟨Sustantivo⟩ ⟨Artículo⟩ ⟨NomProp⟩ ⟨Adjetivo⟩ ⟨Verbo⟩ ⟨Adverbio⟩	}	Toman como valores palabras constantes propias de esas categorías gramaticales.

Tabla 1: La gramática sencilla en forma normal de Backus.

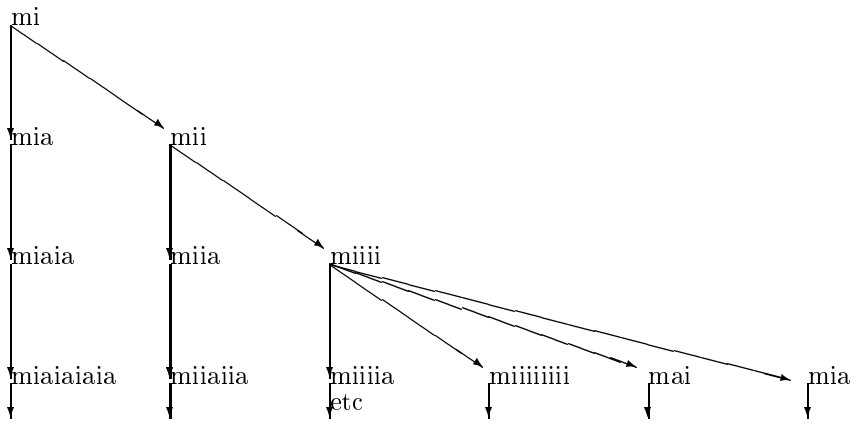


Figura 3: Derivaciones en el sistema *MAI*.

Como ejemplos de derivaciones de una misma palabra tenemos:

Derivación en a)			Derivación en b)		
S	$\to$	$1S$	S	$\to$	$1S$
	$\to$	$11T$		$\to$	$11S$
	$\to$	$110S$		$\to$	$110T$
	$\to$	$1101S$		$\to$	$1101S$
	$\to$	$11011S$		$\to$	$11011S$
	$\to$	110111		$\to$	110111

Vemos pues que un mismo lenguaje puede ser generado por varias gramáticas.

Cualquier gramática de un lenguaje de alto nivel

Como un último ejemplo de gramáticas formales mencionamos tan solo que cualquier lenguaje de programación de alto nivel está generado por una gramática formal.

0.2 Reglas de transformación

Además de las gramáticas formales existen otros procedimientos de transformación simbólica. A manera de ejemplos presentamos algunos de ellos.

0.2.1 MAI : Modos Mecánico, Anti e Inteligente (Zen)

Este ejemplo aparece en el capítulo 1 del libro de Hofstadter ([15]). Consideremos las reglas siguientes:

I.	σi	$\to$	σia
II.	$m\sigma$	$\to$	$m\sigma\sigma$
III.	$\sigma iii\tau$	$\to$	$\sigma a\tau$
IV.	$\sigma aa\tau$	$\to$	$\sigma\tau$

Los caracteres griegos representan a palabras en el alfabeto $MAI = \{m, a, i\}$. La primera regla dice que a toda palabra que termina con i puede añadírsele una a, la segunda, que toda palabra que comience con m puede repetir su "resto", la tercera, que cualquier cadena de tres i-es consecutivas puede cambiarse por una a, y, finalmante, que cualesquiera dos a-es consecutivas pueden ser suprimidas.

Como ejemplos de derivaciones están los mostrados en la figura (3).

Para cada palabra σ sea $MAI(\sigma)$ el conjunto de palabras en el alfabeto MAI que pueden ser derivadas a partir de la palabra σ mediante una sucesión finita de aplicaciones de las reglas de transformación. Naturalmente, surgen los problemas siguientes:

Problema de la palabra. Dadas dos palabras σ, τ decidir si acaso $\tau \in MAI(\sigma)$, es decir, decidir si acaso una palabra es o no derivable desde alguna otra.

Problema de equivalencia. Dadas dos palabras σ, τ decidir si acaso $MAI(\sigma) = MAI(\tau)$, es decir, decidir si acaso cualquier palabra es derivable desde una de las dos palabras si y sólo si es derivable desde la otra.

Problema de derivación óptima. Dadas dos palabras σ, τ tales que $\tau \in MAI(\sigma)$, localizar la sucesión de transformaciones más corta que transforma a σ en τ.

0.2.2 KENNINGS: Poesía antigua islandesa

Este es un género de poesía islandesa de los siglos IX-XIII, construído mediante el remplazo de frases sustantivales por otras equivalentes. Los kennings pueden ser bellas metáforas o irresolubles acertijos. Ciertamente, esta poesía posee dos tipos de encantos: Uno sintáctico y otro semántico, y, por su naturaleza, ambos se mezclan indisolublemente. Como meros ejemplos consideremos las siguientes "equivalencias":
Sustituciones:

arpón	$\rightarrow$	*bruja de los escudos*
barco	$\rightarrow$	*bridón de las olas*
batalla	$\rightarrow$	*tormenta de arpones*
escudo	$\rightarrow$	*luna de barcos*
espada	$\rightarrow$	*fuego de la batalla*
guerrero	$\rightarrow$	*lanzador de espadas*

Ejemplos

En un primer ejemplo, ilustramos la sustitución reiterada que se hace en los kennings, y en el segundo citamos un poema mucho más acabado.

a) *guerrero*
lanzador de espadas
lanzador del fuego de la batalla
lanzador del fuego de la tormenta de arpones
lanzador del fuego de la tormenta de lunas de barcos
lanzador del fuego de la tormenta de lunas de bridones de olas
$\vdots$

Fonéticamente, en castellano es muy desagradable la repetición de la conjunción *de* seguida de un artículo casi obligatorio. En los lenguajes nórdicos esto no aparece pues concatenando los vocablos, los sustantivos pueden realizar funciones de adjetivos, tal como sucede en inglés.

b) El siguiente verso es un kenning traducido por Jorge Luis Borges.

El aniquilador de la prole de gigantes
quebró al fuerte bisonte de la pradera de la gaviota.
Así los dioses, mientras el guardián de la campana se
lamentaba, destrozaron el halcón de la ribera.
De poco le valió el rey de los griegos
al caballo que corre por los arrecifes

0.2.3 Algoritmos de Markov

Estos sistemas formalizan procedimientos de cálculo.

Multiplicación por 3 en representación binaria

En la tabla (2) presentamos a las *producciones*, es decir, a las sustituciones que constituyen este sistema de transformaciones.

Como ejemplos de cálculo presentamos en la tabla (3) dos multiplicaciones. En cada una, la derivación en un renglón proviene de la anterior mediante la aplicación de la regla correspondiente al símbolo *de estado* y las dos literales que lo flanquean.

Ob	$\rightarrow$	$A0$	Inicio		
$1b$	$\rightarrow$	$A1$			
$xA0$	$\rightarrow$	$Bx0$	$xE0$	$\rightarrow$	$Dx1$
$xA1$	$\rightarrow$	$Ex1$	$xE1$	$\rightarrow$	$Fx0$
xAb	$\rightarrow$	$x00$	xEb	$\rightarrow$	$xb1$
$xB0$	$\rightarrow$	$Ax0$	$xF0$	$\rightarrow$	$Gx0$
$xB1$	$\rightarrow$	$Cx1$	$xF1$	$\rightarrow$	$Fx1$
xBb	$\rightarrow$	$x00$	xFb	$\rightarrow$	$x10$
$xC0$	$\rightarrow$	$Dx1$	$xG0$	$\rightarrow$	$Ax1$
$xC1$	$\rightarrow$	$Fx0$	$xG1$	$\rightarrow$	$Hx0$
xCb	$\rightarrow$	$xb1$	xGb	$\rightarrow$	xbb
$xD0$	$\rightarrow$	$Ax0$	xHU	$\rightarrow$	$Gx0$
$xD1$	$\rightarrow$	$Cx1$	$xH1$	$\rightarrow$	$Fx1$
xDb	$\rightarrow$	xbb	xHb	$\rightarrow$	$x10$

con b = blanco y $x \in \{0, 1, b\}$.

Tabla 2: Algoritmo de Markov para triplicar números binarios.

a) $19 \times 3 = 57$	b) $11 \times 3 = 33$
10011	1011
$1001A1$	$101A1$
$100E11$	$10E11$
$10F001$	$1F001$
$1G0001$	$G1001$
$A11001$	$Hb0001$
$Eb11001$	100001
111001	

Tabla 3: Dos ejemplos de la triplicación de números binarios.

Los algoritmos de Markov son equivalentes a otros sistemas de transformación como son las gramáticas formales *irrestrictas*, las funciones recursivas y las máquinas de Turing. Posteriormente presentaremos con todo detalle estos sistemas.

Para concluir esta sección enunciamos la

Tesis 0.2.1 (de Church) *Cualquier procedimiento efectivamente computable corresponde a un conjunto de transformaciones sintácticas sobre un alfabeto.*

Así pues, puesto de manera muy simplificada, se tiene que el concepto de *computabilidad efectiva* ha de coincidir con el de *transformación sintáctica*.

0.3 Una formalización de gramáticas

Sea T un conjunto de símbolos *terminales* y sea V un conjunto de símbolos *variables*. La unión de ellos, $A = V \cup T$, es un *alfabeto* de gramática. A^* es el *diccionario* sobre A y consta de todas las palabras, de longitud finita, con símbolos en A. A^+ coincide con A^*, salvo en que no posee a la palabra vacía, *nil*.

Una *regla de producción* es un elemento del producto cartesiano $A^+ \times A^*$. Si $(\mathbf{a}, \mathbf{c}) \in A^+ \times A^*$ escribimos $\mathbf{a} \to \mathbf{c}$ y decimos que $\mathbf{a}$ es el *antecedente* y $\mathbf{c}$ el *consecuente* de la regla $\mathbf{a} \to \mathbf{c}$.

Sea $P \subset A^+ \times A^*$ un conjunto de reglas de producción. Sea $S \in V$ un símbolo variable distinguido, llamado *inicial*.

El sistema $G = (V, T, P, S)$ se dice ser una *gramática formal*.

Las reglas de producción transforman palabras en otras:

$$\forall \mathbf{x}, \mathbf{y} \in A^* : \mathbf{x} \text{ da } \mathbf{y} \quad \Leftrightarrow \quad \exists \mathbf{p}, \mathbf{q} \in A^*, (\mathbf{a} \to \mathbf{c}) \in P : (\mathbf{x} = \mathbf{paq}) \& (\mathbf{y} = \mathbf{pcq}).$$

La *cerradura reflexivo-transitiva* de la relación "*da*" define la relación de *derivación*

$$\forall \mathbf{x}, \mathbf{y} \in A^* : \mathbf{x} \text{ deriva } \mathbf{y} \text{ si } \mathbf{x}(\text{da})^*\mathbf{y}.$$

El *lenguaje generado* por la gramática consta de todas las palabras que se derivan del símbolo inicial y que sólo contienen símbolos terminales:

$$\text{Lenguaje}(G) = \{\mathbf{x} \in T^* | S \text{ deriva } \mathbf{x}\}.$$

0.4 Jerarquía de Chomsky

En función de la forma de sus producciones, se puede caracterizar qué tan compleja es una gramática formal. Noam Chomsky mostró que esta caracterización clasifica jerárquicamente a las gramáticas formales: Gramáticas en un nivel están incluidas en los siguientes niveles y la inclusión entre niveles es propia.

Se puede dar varios refinamientos de la Jerarquía de Chomsky. En la tabla (4) presentamos esquemáticamente uno de tales refinamientos. Lo esquemático de esta tabla se suprimirá en el capítulo 3 de este curso, en el que se presentará estas gramáticas con todo detalle.

En toda gramática formal aparecen dos problemas fundamentales:

Problema de la palabra:

Instancia: Una gramática G, sobre un alfabeto A, y una palabra $\mathbf{x} \in A$.

Solución: $\begin{cases} 1 & \text{si } \mathbf{x} \in \text{Lenguaje}(L), \\ 0 & \text{en otro caso.} \end{cases}$

Es decir, este problema consiste en decidir, para una gramática y una palabra dadas, si acaso la palabra está generada por la gramática.

Problema de la derivación:

Instancia: Una gramática G, sobre un alfabeto A, y una palabra $\mathbf{x} \in A$.

Tipo	Nombres	Forma de producciones
0	Irrestricta	$\mathbf{p} \to \mathbf{q}, \quad \mathbf{p} \in A^+, \mathbf{q} \in A^*$
1	Irrestricta con memoria limitada	existe una función (computable) tal que la longitud de cualquier cadena en una derivación que dé una palabra $\mathbf{x}$ ha de estar acotada por el valor de la función en la longitud de $\mathbf{x}$,
2	Sensibles al contexto con borro	$\mathbf{p}B\mathbf{r} \to \mathbf{pqr}, \quad \mathbf{q} \in A^*$
3	Sensibles al contexto no reductivas	$S \to nil$, $\mathbf{p}B\mathbf{r} \to \mathbf{pqr}, \quad \mathbf{q} \in A^+$,
4	Libres de contexto	$B \to \mathbf{q}, \quad \mathbf{q} \in A^*$
5	Libres de contexto deterministas	producciones libres de contexto con la particularidad de que una vez que se ha derivado prefijos de un cierto tamaño entonces se tendrá determinada la palabra a derivarse,
6	Lineales	$B \to \mathbf{p}U\mathbf{r}, \quad \mathbf{p}, \mathbf{r} \in T^*$
7	Regulares	$B \to \mathbf{p}U, \quad \mathbf{p} \in A^*$

la tipología empleada denota lo siguiente:

negritas	cadenas de caracteres;	A	alfabeto, $A = T \cup V$,
MAYÚSCULAS	símbolos variables;	T	conjunto de símbolos terminales,
		V	conjunto de símbolos variables.

Tabla 4: Jerarquía gramatical de Chomsky.

Solución: $\begin{cases} [\mathbf{x}_1, \ldots, \mathbf{x}_1] : (\mathbf{x}_1 = S) \& (\mathbf{x}_n = \mathbf{x}) \& \forall i : (\mathbf{x}_i \text{ da } \mathbf{x}_{i+1}) & \text{si } \mathbf{x} \in \text{Lenguaje}(L), \\ nil & \text{en otro caso.} \end{cases}$

Es decir, este problema consiste en encontrar, cuando exista, una derivación, en una gramática dada, de una palabra dada también.

Los problemas mencionados pueden ser resueltos efectivamente en algunos niveles de la Jerarquía de Chomsky. En otros niveles superiores puede ser irresolubles estos problemas.

0.5 Autómatas

Los autómatas vienen a ser mecanismos formales que "realizan" derivaciones en gramáticas formales. La manera en que las realizan es mediante la noción de *reconocimiento*. Una palabra será generada en una gramática si y sólo si la palabra hace transitar al autómata correspondiente a sus condiciones terminales. Por esto es que los autómatas son *analizadores léxicos* (llamados en inglés "*parsers*") de las gramáticas a que corresponden.

0.5.1 Autómatas regulares

Estos son los autómatas finitos más sencillos. Se construyen a partir de un conjunto de *estados* Q y de un conjunto de *símbolos de entrada* T. Su funcionamiento queda determinado por una *función de transición* $t : Q \times T \to Q$. Si $t(q, s) = p$ esto se interpreta como que el autómata transita del estado q al estado p cuando arriba el símbolo s.

En todo autómata finito se cuenta con un estado *inicial*, $q_0 \in Q$ y un conjunto de estados *finales* $F \subset Q$.

Con todo esto definido, la estructura $\text{AutoReg} = (Q, T, t, q_0, F)$ es un *autómata regular*.

De manera natural, t se extiende a una función de transición $T^* \to Q$: Toda palabra se aplica al autómata y éste, partiendo del estado inicial, transita con cada símbolo de la palabra dada según lo especifique t,

correspondiendo a ese símbolo y al estado actual en el autómata. Una palabra es *reconocida* por el autómata si lo hace arribar a un estado final. El *lenguaje* del autómata consta de todas las palabras reconocidas.

Ejemplo: Sea AutoReg $= (Q, T, t, q_0, F)$ el autómata cuyo conjunto de estados es $Q = \{a, b, c\}$, el de símbolos de entrada es $T = \{0, 1\}$, su estado inicial es $q_0 = a$ y el conjunto de estados finales es $F = \{a\}$. Su transición queda determinada por la tabla

t	0	1
a	b	a
b	c	a
c	c	c

Observamos que, partiendo del estado a, mientras lleguen 1's se está en el estado inicial, con un 0 se pasa a b, con un segundo 0 se pasa a c y de ahí no se sale más. En b, al llegar un 1 se regresa al estado inicial. Así pues, para arribar al estado a desde a mismo la cadena de entrada ha de ser una sarta de varias de 1's separadas éstas por únicos 0's. En otras palabras, el autómata reconoce al lenguaje $(1^+0)^*1^+$.

0.5.2 Autómatas de pila

Estos autómatas finitos cuentan con un dispositivo de memoria muy elemental, del tipo *pila*, el cual es un almacenamiento lineal que funciona bajo el principio PEUS[1]: *Primero en Entrar, Ultimo en Salir.*

Sea Q un conjunto de estados, sea T el alfabeto de entrada y sea V un alfabeto de pila. La función de *transición* es de la forma $t : Q \times T \times V \to Q \times V^*$, donde la relación $t(q, a, v) = (p, \mathbf{v})$ se interpreta como sigue: "Si se está en el estado q, arriba el símbolo a y en el tope de la pila está el símbolo b entonces se pasa al estado p y se empila la palabra $\mathbf{v}$".

Un autómata de pila reconoce a una palabra si, tras haberla leído, termina con su pila vacía.

Ejemplo: Las *cadenas equilibradas de paréntesis* son reconocidas por un autómata de pila determinista. Recordamos que

1. () es una cadena equilibrada de paréntesis, (CEP).
2. Si σ es una CEP entonces (σ) es una CEP.
3. La concatenación de dos CEP's es una CEP.

Para describir a un autómata que reconozca CEP's, representemos al paréntesis que *abre* "(" con el símbolo a, al paréntesis que *cierra* ")" con c, y con b al "blanco", es decir, al fin de la cadena de entrada.

Consideremos el autómata de pila cuyas componentes son las siguientes:

$$\begin{array}{lll} Q = \{\text{Seguir}, \text{Exito}, \text{Fracaso}\} & : & \text{estados,} \\ T = \{a, b, c\} & : & \text{símbolos de entrada,} \\ V = \{A, C\} & : & \text{símbolos de pila,} \\ q_0 = \text{Seguir} & : & \text{símbolo inicial,} \end{array}$$

y cuya función de transición actúa como sigue,

$$\begin{array}{rlcll} & (\text{Seguir}, a, y) & \mapsto & (\text{Seguir}, Ay) & \text{empila paréntesis que abren,} \\ & (\text{Seguir}, c, A) & \mapsto & (\text{Seguir}, nil) & \text{suprime paréntesis empatados,} \\ t: & (\text{Seguir}, c, nil) & \mapsto & (\text{Fracaso}, C) & \text{no hay equilibrio,} \\ & (\text{Seguir}, b, A) & \mapsto & (\text{Fracaso}, A) & \text{no hay equilibrio,} \\ & (\text{Seguir}, b, nil) & \mapsto & (\text{Exito}, nil) & \text{equilibrio verificado.} \end{array}$$

Es claro que este autómata de pila reconoce al lenguaje CEP.

[1] Esto es una pésima seudotraducción del inglés *FILO: FirstIn LastOut.*

0.5.3 Autómatas lineales

Los *autómatas lineales* son autómatas de pila deterministas que a lo largo de su computación sólo hacen un "cambio de turno". A grandes rasgos, esto significa que toda computación consiste de un procedimiento de empilar consecutivamente para después pasar a desempilar.

Ejemplo: Consideremos el lenguaje de palíndromas con una marca central:

$$L = \{\mathbf{y}M\mathbf{x} | \mathbf{x} \in (0+1)^*, \mathbf{y} = \mathit{reverso}(\mathbf{x})\}.$$

Consideremos el autómata de pila cuyas componentes son las siguientes:

$$\begin{array}{lcl} Q = \{\text{Meter}, \text{Sacar}, \text{Exito}, \text{Fracaso}\} & : & \text{estados,} \\ T = \{0, 1, M\} & : & \text{símbolos de entrada,} \\ V = \{C, U\} & : & \text{símbolos de pila,} \\ q_0 = \text{Meter} & : & \text{símbolo inicial,} \end{array}$$

y cuya función de transición actúa como sigue,

$$t: \begin{array}{llll} (\text{Meter}, 0, y) & \mapsto & (\text{Meter}, Cy) & \text{empila 0's,} \\ (\text{Meter}, 1, y) & \mapsto & (\text{Meter}, Uy) & \text{empila 1's,} \\ (\text{Meter}, M, y) & \mapsto & (\text{Sacar}, y) & \text{con } M \text{ pasa a desempilar,} \\ (\text{Sacar}, 0, C) & \mapsto & (\text{Sacar}, \mathit{nil}) & \text{desempila si hay empatamiento de 0's,} \\ (\text{Sacar}, 1, U) & \mapsto & (\text{Sacar}, \mathit{nil}) & \text{desempila si hay empatamiento de 1's,} \\ (\text{Sacar}, b, \mathit{nil}) & \mapsto & (\text{Exito}, \mathit{nil}) & \text{estado de éxito,} \end{array}$$

donde $y \in V$ y b es el símbolo "blanco". En cualquier otra instancia de t, ésta transitará al estado de fracaso.

Es claro que este autómata de pila reconoce al lenguaje L.

0.5.4 Autómatas de pila no-deterministas

Los *autómatas de pila no-deterministas* coinciden con sus homólogos de pila salvo en que su transición no es propiamente una función. Aquí se tiene que la *transición* es un subconjunto $t \subset (Q \times T \times V) \times (Q \times V^*)$.

Ejemplo: El lenguaje de palíndromas sin marca central

$$L = \{\mathbf{x} \in (0+1)^* | \mathbf{x} = \text{reverso}(\mathbf{x})\}$$

es reconocido por un autómatas de pila no-determinista que funciona de acuerdo con el sigiente procedimiento:

> *Avanzando a la derecha, se almacena primeramente cada símbolo y cuando se "cree" estar a la mitad, se compara cada símbolo leído con el tope de la pila. Si coinciden, se continúa. En otro caso se marca un error.*

El no-determinismo del autómata está en que no se precisa en qué momento pasará al estado de desempilar. Transita a ése de manera indeterminada.

0.5.5 Máquinas de Turing

Estas son autómatas finitas con dos pilas que tienen un *tope* común. O equivalentemente, son autómatas que poseen una memoria dada por una *cinta* la cual es un almacenamiento lineal, infinito a ambos lados, con acceso a cualquier localidad en ella.

El tope común es la *casilla leída ("scanned cell")*, una pila es la parte de la cinta a la derecha de la casilla leída y otra pila es su parte izquierda.

Las transiciones de la máquina quedan determinadas por una función $t : Q \times T \to Q \times T \times \mathit{Mov}$, donde $\mathit{Mov} = \{\mathit{Der}, \mathit{Izq}\}$. Esta vez, la relación $t(q, a) = (p, b, \mu)$ se interpreta como sigue: "Si se está en el estado q y se lee el símbolo a entonces se escribe b, se pasa a p y se va a examinar la casilla al lado μ de la leída".

$1, x; 1, x, Der$	:	con cualquier cosa, salvo ")", avánce se a la derecha, $x \in \{(, b, A, B\}$,
$1,); 2, B, Izq$	:	con el primer ")", márquese y retrocédase,
$1, b; 3, b, Izq$	:	si la lista se acaba, revísese que todo haya sido marcado,
$2, y; 2, y, Izq$	:	con cualquier cosa, salvo "(", continúese el retroceso, $y \in \{), b, A, B\}$,
$2, (; 1, A, Der$	:	márquese el "(" que empata y repítase el ciclo,
$2, b; 4, No, Alt$	:	se termina la cadena y no existe el correspondiente "(". No hay equilibrio.
$3, x; 3, x, Izq$	:	retrocédase ignorando las marcas, $x \in \{A, B\}$,
$3, (; 3, No, Alt$	:	si quedare un "(", éste ya no podría empatarse. No hay equilibrio.
$3, b; 3, Sí, Alt$	:	se agotó la cadena y todo se marcó. Sí hay equilibrio.

Tabla 5: Máquina de Turing para reconocer cadenas equilibradas de paréntesis.

De hecho, la relación $t(q, a) = (p, b, \mu)$ puede escribirse como $(q, a; p, b, \mu)$, y por esto, decimos que una máquina de Turing queda especificada por su lista de *quíntuplas*. O, abusando aún más del lenguaje, que una lista de quíntuplas es el *programa* correspondiente a una máquina de Turing.

Ejemplo: Cadenas equilibradas de paréntesis

Para tener una cadena equilibrada, cada ")" en ella ha de "empatar" con un correspondiente "(" que lo anteceda.

Para reconocer cadenas equilibradas procederemos como sigue:

- Cada "(" ya empatado se marcará como ya revisado por una A y cada ")" se marcará por una B.
- Inicialmente, se busca el primer ")", yendo de izquierda a derecha.
- Por cada ")",
  - se marca con B,
  - se busca hacia la izquierda el primer "(" que lo empate,
  - si no se encontrare tal "(" la cadena está desequilibrada. En otro caso, el "(" se marca con A y se repite este ciclo.
- Si quedaren "(" no empatados, la cadena está desequilibrada. En otro caso se tiene equilibrio y se termina el proceso.

El procedimiento queda descrito por la lista de quíntuplas mostrada en la tabla (5).

Los cuatro estados de la máquina tienen una interpretación evidente:

1	:	avanza a la derecha buscando ")",
2	:	retrocede a la izquierda buscando "(",
3	:	revisa que todo haya sido marcado,
4	:	estado terminal.

Como mero ejemplo, en la tabla (6) mostramos la computación correspondiente a una cadena dada. En cada *instante*, la casilla leída es la adyacente a la derecha del estado actual, el cual se escribe en negritas.

0.5.6 Jerarquía de Chomsky en autómatas

La complejidad de un autómata está determinada por sus capacidades de transición, de su dispositivo de memoria y las capacidades de inspección en su memoria. De manera natural la jerarquía gramatical se traduce en una jerarquía equivalente en los autómatas. En la tabla (7) presentamos la equivalente a la mostrada en la tabla (4).

$()(())$	$AB(A\mathbf{1}B)$	$ABAAB\mathbf{3}B$
$\mathbf{1}()(())$	$AB(AB\mathbf{1})$	$ABAA\mathbf{3}BB$
$(\mathbf{1})(())$	$AB(A\mathbf{2}BB$	$ABA\mathbf{3}ABB$
$\mathbf{2}(B(())$	$AB(\mathbf{2}ABB$	$AB\mathbf{3}AABB$
$A\mathbf{1}B(())$	$AB\mathbf{2}(ABB$	$A\mathbf{3}BAABB$
$AB\mathbf{1}(())$	$ABA\mathbf{1}ABB$	$\mathbf{3}ABAABB$
$AB(\mathbf{1}())$	$ABAA\mathbf{1}BB$	$\mathbf{3}bABAABB$
$AB((\mathbf{1}))$	$ABAAB\mathbf{1}B$	[Sí]$ABAABB$
$AB(\mathbf{2}(B)$	$ABAABB\mathbf{1}b$	

Tabla 6: Un ejemplo del funcionamiento de la máquina de Turing que reconoce cadenas equilibradas de paréntesis.

Tipo	Nombres	Tipo de autómata reconocedor
0	Irrestricta	Máquinas de Turing
1	Irrestricta con memoria limitada	Máquinas de Turing con cinta acotada
2	Sensibles al contexto con borro	Máquinas de Turing con dominio total
3	Sensibles al contexto no reductivas	Máquinas de Turing con "espacio lineal"
4	Libres de contexto	Autómatas de pila no-deterministas
5	Libres de contexto deterministas	Autómatas de pila deterministas
6	Lineales	Autómatas lineales
7	Regulares	Autómatas regulares

Tabla 7: Jerarquía de Chomsky en autómatas.

Capítulo 1

Fundamentos matemáticos

1.1 Cadenas y lenguajes

El *producto cartesiano* de dos conjuntos A y B consta de las parejas ordenadas cuyo primer elemento está en A y cuyo segundo elemento está en B. Usaremos la notación de *yuxtaposición* para denotar al producto cartesiano:

$$AB = \{ab | a \in A, b \in B\}.$$

Cualquier conjunto finito Σ es un *alfabeto*. Entre los alfabetos más comunes están:

$$\begin{array}{rcl} Latino & = & \{a, b, c, \ldots, x, y, z\} \\ (0+1) & = & \{0, 1\} \\ Diez & = & \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\ ASCII & = & \{(00)_{16}, \ldots, (FF)_{16}\} \\ & : & \text{símbolos del ASCII extendido} \end{array}$$

Sea $\Sigma^1 = \Sigma$ y, para $n \geq 1$, sea $\Sigma^{n+1} = \Sigma\Sigma^n$. Cada elemento $\sigma \in \Sigma^n$ se dice ser una *palabra* sobre Σ de *longitud* n. La palabra *nil* que no tiene símbolo alguno es la palabra *vacía*. Definimos $\Sigma^0 = \{nil\}$.

El *diccionario* sobre Σ es $\Sigma^* = \bigcup_{n \geq 0} \Sigma^n$, y, por ende, consta de todas las palabras de longitud finita con símbolos en el alfabeto Σ. El conjunto de palabras no-vacías, se denota como $\Sigma^+ = \Sigma^* - \{nil\}$. Para cada $n \leq 0$, $\Sigma^{\leq n} = \bigcup_{\nu=0}^{n} \Sigma^\nu$ es el conjunto de palabras de longitud a lo sumo n.

Ejemplos:

1. $nil \in \Sigma^*$ para cualquier alfabeto Σ.

2. $2000 \in Diez^4 \subset Diez^*$.

3. $(2000)_2 = 11111010000 \in (0+1)^{11} \subset (0+1)^*$.

4. $(259430127)_2 = 11110111011010010111011101111 \in (0+1)^{28} \subset (0+1)^*$.

5. $soyunapalabradetreintayseisliterales \in Latino^{36} \subset Latino^*$.

6. Cualquier programa en C es una palabra en $ASCII^*$.

Un *lenguaje* es un subconjunto de cualquier diccionario. $\emptyset$ y Σ^* son los lenguajes *vacío* y *total*, respectivamente.

1.2 Monoide de cadenas

1.2.1 La operación de concatenación

Dotemos al diccionario Σ^* de un alfabeto Σ de la operación de *concatenación*:

$$\begin{array}{rcl} \cdot : \Sigma^* \times \Sigma^* & \to & \Sigma^* \\ (\sigma_1, \sigma_2) & \mapsto & \sigma_1 \cdot \sigma_2 \end{array}$$

que a cada dos palabras $\sigma_1 = s_{10} \cdots s_{1,l_1-1}$, $\sigma_2 = s_{20} \cdots s_{2,l_2-1}$ le asocia la palabra que se obtiene de pegar, tras los símbolos de la primera palabra, a los símbolos de la segunda. Nos referiremos indistintamente a esta operación como de concatenación o de *yuxtaposición*.

Se ve inmediatamente que se cumplen las proposiciones siguientes:

Proposición 1.2.1 *La concatenación es asociativa:*

$$\forall \sigma_1, \sigma_2, \sigma_3 \in \Sigma^* : \quad (\sigma_1\sigma_2)\sigma_3 = \sigma_1(\sigma_2\sigma_3).$$

Consecuentemente, Σ^* con la concatenación, es decir, la estructura $(\Sigma^*, \cdot)$, es un *semigrupo*.

Proposición 1.2.2 *La palabra vacía nil es una* unidad*, por ambos lados, para la concatenación:*

$$\forall \sigma \in \Sigma^* : \quad nil \cdot \sigma = \sigma = \sigma \cdot nil.$$

Por tanto, la estructura $(\Sigma^*, \cdot, nil)$ es un *monoide*.

Proposición 1.2.3 *Las leyes de* cancelación *se cumplen por ambos lados:*

$$\begin{array}{rlcl} \forall \sigma_1, \sigma_2, \sigma_3 \in \Sigma^* : & \sigma_1\sigma_2 = \sigma_1\sigma_3 & \Rightarrow & \sigma_2 = \sigma_3 \\ & \sigma_2\sigma_1 = \sigma_3\sigma_1 & \Rightarrow & \sigma_2 = \sigma_3 \end{array}$$

Sin embargo, como un hecho, digamos "negativo", se tiene que la concatenación no es conmutativa.

El diccionario sobre un alfabeto, bien que es infinito, es numerable.

Proposición 1.2.4 *Para todo alfabeto Σ, su diccionario Σ^* es un conjunto numerable.*

En efecto, supongamos que $\Sigma = \{s_0, \ldots, s_{m-1}\}$. A cada palabra de longitud fija, digamos k, la podemos ver como la representación en base m de un único número entero entre 0 y $m^k - 1$, inclusive. Esto da una enumeración efectiva de cada bloque de palabras de longitud fija. Colocamos los bloques, ya enumerados, según el orden de k. Esto da una enumeración de Σ^*.

Más precisamente, definamos $c : \Sigma^* \to I\!N$ como sigue:

$$\begin{array}{rcl} c(nil) & = & 0 \\ c(s_{n_1}s_{n_2}\cdots s_{n_k}) & = & \dfrac{m^k - 1}{m - 1} + \displaystyle\sum_{j=1}^{k} n_j m^{k-j} \end{array}$$

Así, por ejemplo, para el alfabeto *Latino* de $m = 26$ caracteres se tiene

$$\begin{array}{rcl} c(zapatista) & = & \dfrac{26^9 - 1}{26 - 1} + 25 \cdot 26^8 + 0 \cdot 26^7 + 15 \cdot 26^6 + 0 \cdot 26^5 + \\ & & 19 \cdot 26^4 + 8 \cdot 26^3 + 20 \cdot 26^2 + 19 \cdot 26^1 + 0 \cdot 26^0 \\ & = & 135737591974375 + 5225319188206 \\ & = & 140962911162581 \end{array}$$

$$\begin{array}{rcl} c(mexico) & = & \dfrac{26^6 - 1}{26 - 1} + 12 \cdot 26^5 + 4 \cdot 26^4 + 23 \cdot 26^3 + 8 \cdot 26^2 + \\ & & 2 \cdot 26^1 + 14 \cdot 26^0 \\ & = & 7722894375 + 144814138 = 7867708513 \end{array}$$

1.2.2 Relaciones de orden

Orden de prefijo

Primeramente presentamos condiciones necesarias y suficientes para comparar cadenas y subcadenas.

Teorema 1.2.1 (Levi) *Sean $\sigma_1, \sigma_2, \tau_1, \tau_2 \in \Sigma^*$ tales que $\sigma_1\tau_1 = \sigma_2\tau_2$. Entonces se cumplen las implicaciones siguientes:*

1. $long(\sigma_1) \geq long(\sigma_2) \Rightarrow \exists! \upsilon \in \Sigma^* \, (\sigma_1 = \sigma_2\upsilon \;\&\; \upsilon\tau_1 = \tau_2)$
2. $long(\sigma_1) = long(\sigma_2) \Rightarrow (\sigma_1 = \sigma_2 \;\&\; \tau_1 = \tau_2)$
3. $long(\sigma_1) \leq long(\sigma_2) \Rightarrow \exists! \upsilon \in \Sigma^* \, (\sigma_2 = \sigma_1\upsilon \;\&\; \upsilon\tau_2 = \tau_1)$

Gráficamente resulta muy clara la validez del teorema y por eso omitimos su demostración formal.
Como una consecuencia directa de este teorema resulta el siguiente

Corolario 1.2.1 *Si $\sigma_1, \sigma_2, \tau_1, \tau_2 \in \Sigma^*$ son tales que $\sigma_1\tau_1 = \sigma_2\tau_2$ entonces σ_1 es un prefijo de σ_2 o, recíprocamente, σ_2 es un prefijo de σ_1.*

Recordamos que una relación de *orden* "$\leq$" en un conjunto S es una relación binaria $(\leq) \subset S^2$ tal que es reflexiva, transitiva y antisimétrica. Esto último significa que

$$\forall s, t \in S: \;\; (s \leq t) \& (t \leq s) \;\Rightarrow\; (s = t).$$

Del último corolario se ve fácilmente que la relación *de prefijo*:

$$\forall \sigma_1, \sigma_2 \in \Sigma^*: \;\; (\sigma_1 \leq_p \sigma_2) \Leftrightarrow (\sigma_1 \text{ es un prefijo de } \sigma_2),$$

es, efectivamente, un orden en Σ^*.

Hemos visto que Σ^* no es conmutativo. Veremos a continuación algunas proposiciones relativas al orden de prefijo y a la contención, en general, entre cadenas.

Proposición 1.2.5 *Si $\sigma_1, \sigma_2, \tau_1, \tau_2 \in \Sigma^*$ son tales que $\sigma_1 \leq_p \tau_1$ y $\sigma_2 \leq_p \tau_1\tau_2$ entonces $\sigma_1 \leq_p \sigma_2$ o $\sigma_2 \leq_p \sigma_1$.*

En efecto, las siguientes implicaciones son todas verdaderas:

$$\left.\begin{array}{l} \sigma_1 \leq_p \tau_1 \\ \sigma_2 \leq_p \tau_1\tau_2 \end{array}\right\} \;\Rightarrow\; \exists \upsilon_1, \upsilon_2 \in \Sigma^*: \left\{\begin{array}{rcl} \tau_1 & = & \sigma_1\upsilon_1 \\ \tau_1\tau_2 & = & \sigma_2\upsilon_2 \end{array}\right.$$
$$\Rightarrow \;\; \sigma_1(\upsilon_1\tau_2) = \sigma_2\upsilon_2$$
$$\overset{\text{Cor}}{\Rightarrow} \;\; (\sigma_1 \leq_p \tau_1) \vee (\sigma_2 \leq_p \sigma_1)$$

En el siguiente teorema veremos una condición necesaria para que una palabra sea a la vez prefijo y sufijo de alguna otra. Es evidente que la condición que ahí se establece es también suficiente.

Teorema 1.2.2 *Sean $\sigma_1, \sigma_2 \in \Sigma^*$ tales que σ_1 es prefijo y sufijo de σ_2, es decir*

$$\exists \tau_1, \tau_2 \in \Sigma^+ : (\tau_2\sigma_1 = \sigma_2) \& (\sigma_2 = \sigma_1\tau_1).$$

Entonces,

$$\exists \upsilon_1, \upsilon_2 \in \Sigma^*, k \geq 0: \left\{\begin{array}{rcl} \tau_1 & = & \upsilon_1\upsilon_2 \\ \tau_2 & = & \upsilon_2\upsilon_1 \end{array}\right. \;\&\; \left\{\begin{array}{rcl} \sigma_1 & = & (\upsilon_1\upsilon_2)^k \upsilon_1 \;=\; \upsilon_1 (\upsilon_2\upsilon_1)^k \\ \sigma_2 & = & (\upsilon_1\upsilon_2)^{k+1} \upsilon_1 \;=\; \upsilon_1 (\upsilon_2\upsilon_1)^{k+1} \end{array}\right.$$

En efecto, consideremos dos casos, según se comparen entre sí las longitudes de las palabras σ_1 y τ_2.

$\log(\sigma_1) \leq \log(\tau_2)$: Ya que $\tau_2\sigma_1 = \sigma_1\tau_1$, por el teorema de Levy se tiene que σ_1 es un prefijo de τ_2. Se ve pues que $\exists \upsilon_1 \in \Sigma^+$ tal que $\tau_2 = \sigma_1\upsilon_1$ y $\tau_1 = \upsilon_1\sigma_1$. El teorema se cumple tomando $\upsilon_2 = \sigma_1$ y $k = 0$.

$\log(\sigma_1) > \log(\tau_2)$**:** Probemos el teorema por inducción en $\log(\sigma_1)$.

Para $\log(\sigma_1) = \log(\tau_2)$ el teorema se cumple pues se está en el caso anterior.

Supongamos ahora que $\log(\sigma_1) > \log(\tau_2)$ y que el teorema se cumple para toda palabra σ_{11} tal que $\log(\sigma_1) > \log(\sigma_{11}) \geq \log(\tau_2)$.

Ya que $\tau_2\sigma_1 = \sigma_1\tau_1$, por el teorema de Levy se tiene que, para alguna $\sigma_{11} \in \Sigma^*$

$$\sigma_1 = \tau_2\sigma_{11} \quad \& \quad \sigma_1 = \sigma_{11}\tau_1.$$

Por la hipótesis de inducción $\exists \upsilon_1, \upsilon_2 \in \Sigma^*, k \geq 0$:

$$\left\{ \begin{array}{rcl} \tau_1 & = & \upsilon_1\upsilon_2 \\ \tau_2 & = & \upsilon_2\upsilon_1 \end{array} \right. \quad \text{y} \quad \left\{ \begin{array}{rcll} \sigma_{11} & = & (\upsilon_1\upsilon_2)^k\,\upsilon_1 & = \quad \upsilon_1\,(\upsilon_2\upsilon_1)^k \\ \sigma_1 & = & (\upsilon_1\upsilon_2)^{k+1}\,\upsilon_1 & = \quad \upsilon_1\,(\upsilon_2\upsilon_1)^{k+1} \end{array} \right.$$

y con esto queda demostrado el teorema.

Con el siguiente teorema vemos cuándo dos palabras dadas son ambas múltiplos de una misma partícula.

Teorema 1.2.3 *Sean $\sigma_1, \sigma_2 \in \Sigma^*$. Las siguientes aseveraciones son equivalentes:*

1. *Ambas σ_1, σ_2 son múltiplos de una misma partícula, es decir,*

$$\exists \tau \in \Sigma^*,\, n_1, n_2 \in I\!N : \ (\sigma_1 = \tau^{n_1})\,\&\,(\sigma_2 = \tau^{n_2})\,.$$

2. *Las palabras σ_1 y σ_2 poseen sendos múltiplos con un prefijo común de longitud*

$$\log(\sigma_1) + \log(\sigma_2) - \mathrm{mcd}\,(\log(\sigma_1), \log(\sigma_2))\,.$$

En símbolos,

$$\begin{array}{rcl} \exists \upsilon \in \Sigma^*,\, m_1, m_2 \in I\!N & : & (\upsilon \leq_p \sigma_1^{m_1})\,\&\,(\upsilon \leq_p \sigma_2^{m_2})\,\& \\ & & \log(\upsilon) = \log(\sigma_1) + \log(\sigma_2) - \mathrm{mcd}\,(\log(\sigma_1), \log(\sigma_2)) \end{array}$$

Escribamos $k_i = \log(\sigma_i)$, $i = 1, 2$.

$1 \Rightarrow 2$**:** Supongamos que $\sigma_i = \tau^{n_i}$, $i = 1, 2$. Para $i = 1, 2$, hagamos

$$m_i = \left\lceil \frac{n_1 + n_2 - \mathrm{mcd}(n_1, n_2)}{n_i} \right\rceil,$$

donde $\lceil x \rceil$ denota al entero más chico por encima de x.

Tenemos pues que, para $i = 1, 2$, $\sigma_i^{m_i} = \tau^{m_i n_i}$. Puesto que $m_i n_i \geq n_1 + n_2 - \mathrm{mcd}(n_1, n_2)$, se tiene que $\sigma_i^{m_i}$ contiene a la palabra $\upsilon = \tau^{n_1+n_2-\mathrm{mcd}(n_1,n_2)}$, la cual tiene longitud

$$\log(\upsilon) = \log(\tau)\,(n_1 + n_2 - \mathrm{mcd}(n_1, n_2)) = k_1 + k_2 - \mathrm{mcd}(k_1, k_2)$$

$2 \Rightarrow 1$**:** Supongamos que υ es un prefijo común de $\sigma_i^{m_i}$, $i = 1, 2$, de longitud $k_1 + k_2 - \mathrm{mcd}(k_1, k_2)$.

Supongamos, sin pérdida de generalidad, que $k_1 \leq k_2$, e, inicialmente, que $\mathrm{mcd}(k_1, k_2) = 1$. Escribamos $\sigma_i = s_{i0} \cdots s_{i,k_i-1}$. Como ambos $\sigma_1^{m_1}$ y $\sigma_2^{m_2}$ poseen un prefijo común de longitud $k_1 + k_2 - 1$, se tiene que

$$\forall k < k_1 + k_2 - 1 : s_{1,k \bmod k_1} = s_{2,k \bmod k_2}.$$

En particular, $s_{1,(\kappa_2+k_2) \bmod k_1} = s_{2,\kappa_2}$, $\forall \kappa_2 < k_2$. Y por tanto, $s_{1,(\kappa_1 k_2) \bmod k_1} = s_{2,0}$, $\forall \kappa_1 < k_1$.

Ahora bien, como $\mathrm{mcd}(k_1,k_2)=1$, tenemos que el conjunto de índices $\{(\kappa_1 k_2) \bmod k_1 | \kappa_1 < k_1\}$ coincide con todo el intervalo $[0,k_1-1]$. Por tanto, tenemos que $\forall \kappa_1 < k_1 : s_{1,\kappa_1} = s_{2,0}$.

Así pues, resulta válida la condición 1., con $\tau = s_{2,0}$, la cual es una palabra de longitud 1.

En el caso de que $\mathrm{mcd}(k_1,k_2)=d>1$, consideremos el alfabeto $\Sigma' = \Sigma^d$. Entonces σ_1 y σ_2, vistas como palabras en $\left(\Sigma^d\right)^*$, tienen longitud $k_1' = \frac{k_1}{d}$ y $k_2' = \frac{k_2}{d}$ respectivamente, y aquí, $\mathrm{mcd}(k_1',k_2')=1$. Así, este caso se reduce al anterior y consecuentemente también en este caso resulta válida la condición 1.

Para concluir esta sección veamos una condición necesaria para que dos palabras conmuten bajo la operación de concatenación.

Proposición 1.2.6 *Sean $\sigma_1,\sigma_2 \in \Sigma^*$ dos palabras tales que $\sigma_1\sigma_2 = \sigma_2\sigma_1$. Entonces*

$$\exists \tau \in \Sigma^*,\, n_1,n_2 \in \mathbb{N} :\ (\sigma_1 = \tau^{n_1}) \,\&\, (\sigma_2 = \tau^{n_2})\,.$$

En efecto, supongamos que $\sigma_1\sigma_2 = \sigma_2\sigma_1$ y que $\mathrm{long}(\sigma_1) \geq \mathrm{long}(\sigma_2)$. Por el Teorema de Levy, existe una palabra $\sigma_{11} \in \Sigma^*$ tal que $(\sigma_1 = \sigma_2\sigma_{11}) \,\&\, (\sigma_1 = \sigma_{11}\sigma_2)$. Consecuentemente, σ_{11} y σ_2 conmutan. Así, que por la misma razón, $\exists \sigma_{12} \in \Sigma^* : (\sigma_{11} = \sigma_2\sigma_{12}) \,\&\, (\sigma_{11} = \sigma_{12}\sigma_2)$.

Consecutivamente, en tanto que $\mathrm{long}(\sigma_{1,\kappa-1}) \geq \mathrm{long}(\sigma_2)$, se tendrá que ha de existir una $\sigma_{1,\kappa} \in \Sigma^*$ tal que $(\sigma_{1,\kappa-1} = \sigma_2\sigma_{1,\kappa}) \,\&\, (\sigma_{1,\kappa-1} = \sigma_{1,\kappa}\sigma_2)$.

Así pues, $\exists q_1 \in \mathbb{N} :\ (\sigma_1 = \sigma_2^{q_1}\tau_1) \,\&\, (\sigma_2\tau_1 = \tau_1\sigma_2) \,\&\, (\mathrm{long}(\tau_1) < \mathrm{long}(\sigma_2))$, donde $\tau_1 = \sigma_{1,q_1}$.

Igualmente, $\exists q_2 \in \mathbb{N} :\ (\sigma_2 = \tau_1^{q_2}\tau_2) \,\&\, (\tau_1\tau_2 = \tau_2\tau_1) \,\&\, (\mathrm{long}(\tau_1) < \mathrm{long}(\tau_1))$, donde $\tau_2 = \sigma_{2,q_2}$.

Reiterando este procedimiento construimos una sucesión de palabras $(\tau_k)_k$ tal que para todo k,

$$\left(\tau_{k-1} = \tau_k^{q_{k+1}}\tau_{k+1}\right) \,\&\, (\tau_k\tau_{k+1} = \tau_{k+1}\tau_k) \,\&\, (\mathrm{long}(\tau_k) < \mathrm{long}(\tau_k))\,.$$

Observe el lector la similitud que existe entre este procedimiento y el cálculo del máximo común divisor de $\mathrm{long}(\sigma_1), \mathrm{long}(\sigma_2)$ mediante el Algoritmo de Euclides. Como las longitudes de las palabras τ_i están decreciendo, en un momento se harán cero. Sea τ la última palabra no nula en esa sucesión. Pues bien, se tiene que $\exists n_1,n_2 \in \mathbb{N} :\ (\sigma_1 = \tau^{n_1}) \,\&\, (\sigma_2 = \tau^{n_2})$.

Orden lexicográfico

Sea Σ un alfabeto dotado de un orden "$\leq$", por ejemplo, para el alfabeto ASCII consideramos el orden usual. Extendemos el orden "$\leq$" de Σ a un orden "$\leq_{lg}$" de Σ^* mediante la definición siguiente:

$\forall \sigma_1,\sigma_2 \in \Sigma^* :\ \sigma_1 \leq_{lg} \sigma_2 \quad \Leftrightarrow$ i) σ_1 es un prefijo de σ_2, o bien

ii) la primera posición en la que difieren σ_1 y σ_2, el símbolo de σ_1 es estrictamente menor, según el orden de Σ, que el de σ_2. En otras palabras, para cualesquiera tres palabras $\tau_0,\tau_1,\tau_2 \in \Sigma^*$ y cualesquiera símbolos $s_1,s_2 \in \Sigma$ rige la implicación

$$\left.\begin{array}{rcl} \sigma_1 &=& \tau_0 s_1 \tau_1 \\ \sigma_2 &=& \tau_0 s_2 \tau_2 \end{array}\right\} \Rightarrow s_1 < s_2$$

La siguiente proposición resume las propiedades más elementales de este orden, llamado *lexicográfico*. Su demostración es más tediosa que instructiva y por esa razón la omitimos.

Proposición 1.2.7 *El orden lexicográfico satisface las propiedades siguientes:*

1. *Es, en efecto, un orden, es decir, cualesquiera que sean las palabras σ,τ,υ,*

$$\begin{array}{rcl} & & \sigma \leq_{lg} \sigma \\ (\sigma \leq_{lg} \tau)\&(\tau \leq_{lg} \upsilon) & \Rightarrow & \sigma \leq_{lg} \upsilon \\ (\sigma \leq_{lg} \tau)\&(\tau \leq_{lg} \sigma) & \Rightarrow & \sigma = \tau \end{array}$$

2. *Es lineal, es decir, $\forall \sigma,\tau$: $(\sigma \leq_{lg} \tau) \vee (\sigma = \tau) \vee (\tau \leq_{lg} \sigma)$.*

3. *Es congruente por la izquierda, es decir, $\forall \sigma,\tau : \upsilon$, $(\sigma \leq_{lg} \tau \Rightarrow \upsilon\sigma \leq_{lg} \upsilon\tau)$. Mas no lo es por la derecha.*

1.2.3 Homomorfismos

Sean $(S_1, *_1, u_1)$ y $(S_2, *_2, u_2)$ dos monoides con operaciones respectivas $*_1$ y $*_2$ y unidades u_1 y u_2. Una función $h : S_1 \to S_2$ es un *homomorfismo* si

$$\begin{aligned} \forall x, y \in S_1 : \ h(x *_1 y) &= h(x) *_2 h(y) \\ h(u_1) &= u_2 \end{aligned}$$

En otras palabras, un homomorfismo es una correspondencia entre monoides que preserva las operaciones.

Observación 1.2.1 *El tomar* longitudes *da un homomorfismo* $(\Sigma^*, \cdot) \to (\mathbb{N}, +)$*:*

$$\forall \sigma_1, \sigma_2 \in \Sigma^* : \quad long(\sigma_1\sigma_2) = long(\sigma_1) + long(\sigma_2).$$

En efecto, sea $(\mathbb{N}, +, 0)$ el monoide que consta de los números naturales con la suma como operación y el 0 como unidad. Sea Σ un alfabeto. Ya que $\forall \sigma_1, \sigma_2 \in \Sigma^* : long(\sigma_1\sigma_2) = long(\sigma_1) + long(\sigma_2)$, y además $long(nil) = 0$ tenemos que $long : (\Sigma^*, \cdot, nil) \to (\mathbb{N}, +, 0)$ es un homomorfismo.

Ejemplos de homomorfismos.

1. Sea $(\mathbb{Z}_2, +, 0)$ el monoide sobre el conjunto $\mathbb{Z}_2 = \{0, 1\}$ con la operación suma "módulo 2", o, en otras palabras, el complemento de la disyunción exclusiva, XOR. Sea Σ un alfabeto y $par : (\Sigma^*, \cdot, nil) \to (\mathbb{Z}_2, +, 0)$ la función definida como

$$par : \sigma \mapsto \begin{cases} 0 & \text{si } long(\sigma) \text{ es par,} \\ 1 & \text{si } long(\sigma) \text{ es impar.} \end{cases}$$

par es, en efecto, un homomorfismo.

2. Sea Σ un alfabeto y sea $f : \Sigma \to \Sigma^*$ cualquier función. Extendemos f a una función $f^* : \Sigma^* \to \Sigma^*$ haciendo

$$\begin{aligned} f^*(nil) &= nil \\ \forall \sigma \in \Sigma^*, s \in \Sigma : \ f^*(\sigma s) &= f^*(\sigma) f(s) \end{aligned}$$

f^* es un homomorfismo y es, propiamente, una sustitución de símbolos: en cada palabra, cada símbolo s se sustituye por la partícula $f(s)$. El homomorfismo f^* se dice *libre de nil* si para ningún $s \in \Sigma$ se tiene $f(s) = nil$.

3. Sea Σ un alfabeto. Dadas dos palabras $\sigma_1, \sigma_2 \in \Sigma^*$ diremos que

- σ_1 es un *prefijo* de σ_2 si $\exists \sigma \in \Sigma^* : \ \sigma_2 = \sigma_1\sigma$.
- σ_1 es un *sufijo* de σ_2 si $\exists \sigma \in \Sigma^* : \ \sigma_2 = \sigma\sigma_1$.
- σ_1 es un *enfijo* de σ_2 si $\exists \sigma', \sigma'' \in \Sigma^+ : \ \sigma_2 = \sigma'\sigma_1\sigma''$.

Sea $n > 0$. Consideremos la operación $suf_n : \Sigma^* \to \Sigma^{\leq n}$ que actúa sobre cualquier palabra suprimiendo, si fuera necesario, un prefijo para quedarse únicamente con el sufijo de longitud n. Más precisamente,

$$suf_n : s_{i_1} s_{i_2} \ldots s_{i_k} \mapsto s_{i_{k-n+1}} \ldots s_{i_k}.$$

$\Sigma^{\leq n}$ tiene una estructura de monoide con la operación de concatenación, seguida de suf_n,

$$(\sigma_1, \sigma_2) \mapsto suf_n(\sigma_1\sigma_2).$$

Bajo esta transformación que es, en efecto, un homomorfismo, todas las palabras de longitud a lo sumo n permanecen fijas.

Sea $(S, \cdot, u)$ un monoide cualquiera. Consideremos el monoide aditivo sobre el conjunto de números naturales con la suma usual, $(\mathbb{N}, +, 0)$. Entonces se tiene que para todo $s \in S$ existe un único homomorfismo $f_s : \mathbb{N} \to S$ tal que $f_s(1) = s$.

La imagen de tal homomorfismo es el submonoide $(s^n)_{n \geq 0}$ y se dice ser el *generado* por el elemento $s \in S$.

Un *antihomomorfismo* entre dos monoides $(S_1, *_1, u_1)$ y $(S_2, *_2, u_2)$ es una función $h : S_1 \to S_2$ tal que

$$\begin{aligned} \forall x, y \in S_1 : \ h(x *_1 y) &= h(y) *_2 h(x) \\ h(u_1) &= u_2 \end{aligned}$$

es decir, es una operación que en el contradominio "conmuta" a la operación.

Ejemplos.

1. Si S_2 es conmutativo, entonces cualquier homomorfismo es también un antihomomorfismo.

2. Sea Σ un alfabeto y sea $reverso : \Sigma^* \to \Sigma^*$ la función que intercambia de derecha a izquierda los símbolos que conforman una palabra. Inductivamente, se tiene

$$\begin{aligned} reverso(nil) &= nil \\ \forall \sigma \in \Sigma^*, s \in \Sigma : \ reverso(\sigma s) &= s\, reverso(\sigma) \end{aligned}$$

Se ve fácilmente que

$$\forall \sigma_1, \sigma_2 \in \Sigma^* : \ reverso(\sigma_1 \sigma_2) = reverso(\sigma_2) reverso(\sigma_1)$$

y consecuentemente esta función es un antihomomorfismo.

Las palabras fijas bajo este operador son *palíndromas*.

Como meros ejemplos, se tiene a los siguientes:

dabale arroz a la zorra el abad
la marlene ama en el ramal
elba mas amable
elba rodas adorable
atomo oro o mota
a man a plan a canal panama

1.2.4 Relaciones de equivalencia congruentes

En un conjunto A, una *relación de equivalencia* R es un subconjunto $R \subset A^2$ tal que es

- reflexiva: $\forall a \in A, \ aRa$,
- simétrica: $\forall a, b \in A, \ aRb \Rightarrow bRa$, y
- transitiva: $\forall a, b, c \in A, \ aRb \& bRc \Rightarrow aRc$.

Una relación de equivalencia induce una partición de manera natural: Para cada elemento $a \in A$ la *clase de equivalencia de* a bajo la relación R es el conjunto $[a]_R = \{b \in A | aRb\}$ que consta de los elementos equivalentes a a. Se tiene

$$\begin{aligned} [a]_R \neq [b]_R \ &\Rightarrow \ [a]_R \cap [b]_R = \emptyset \\ \forall b \in A \ \exists a \in A \ &: \ b \in [a]_R \end{aligned}$$

Así pues, la colección de clases de equivalencia $A/R = \{[a]_R | a \in A\}$ es una partición de A. A/R se dice ser el *cociente* de A bajo la relación R. La cardinalidad del cociente A/R es el *índice* de la relación R. La función $\pi_R : a \mapsto [a]_R$, que a cada elemento le asocia su clase de equivalencia, es la *proyección canónica* de A sobre el cociente A/R.

Toda relación de equivalencia induce pues una partición en su conjunto de definición. Recíprocamente, si $\mathcal{R}$ es una *partición* en A, es decir, si $\mathcal{R}$ es una colección de subconjuntos de A tal que

$$\begin{aligned} \forall R, S \in \mathcal{R} \ &: \ R \neq S \Rightarrow R \cap S = \emptyset \\ & A = \bigcup_{R \in \mathcal{R}} R \end{aligned}$$

entonces la relación $\equiv_{\mathcal{R}}$ definida como

$$\forall a, b \in A : a \equiv_{\mathcal{R}} b \Leftrightarrow \exists R \in \mathcal{R}(a, b \in R),$$

es una relación de equivalencia que determina como cociente precisamente a la partición $\mathcal{R}$.

A manera de ejemplos, citemos a las relaciones siguientes en un conjunto $A \neq \emptyset$:

Discreta. Sea $R = (=)$ la relación que coincide con la identidad en A: $\forall a, b \in A,\ aRb \Leftrightarrow a = b$. Entonces, para toda a, la clase de equivalencia $[a]$ es la mónada cuyo único elemento es a, y el cociente A/R se identifica con A. En este caso, la proyección canónica π "es" la función identidad.

Tosca. Sea $R = A^2$ la relación que identifica a cualesquiera dos elementos en A. La única clase de equivalencia es A^2 y la proyección canónica π es una función constante.

Una manera de introducir relaciones de equivalencia en un conjunto es transportando, mediante funciones, las definidas en otros conjuntos. Si S es una relación de equivalencia en un conjunto B y $f : A \to B$ es una función con dominio en un conjunto A y contradominio en el conjunto B entonces la relación R en A definida como

$$\forall a_1, a_2 \in A :\ a_1 R a_2 \Leftrightarrow f(a_1) S f(a_2)$$

es también una relación de equivalencia.

Si S es la identidad en B, entonces diremos que R es el *núcleo* de la función f, y lo denotaremos $núc(f)$. Así pues

$$núc(f) = \{(a_1, a_2) \in A^2 | f(a_1) = f(a_2)\}.$$

Ejemplo. Consideremos a $\mathbb{R}$ con la identidad y sea $A = \mathbb{R}^2$. La función $f : (x, y) \mapsto x^2 + y^2$ tiene como gráfica a un paraboloide de revolución con vértice en el origen. El núcleo de f consta de todas las parejas de puntos equidistantes al origen, y esta relación de equivalencia determina como espacio cociente a la colección de todos los círculos concéntricos con centro en el origen. Para cada punto $\mathbf{x} \in \mathbb{R}^2$, su clase de equivalencia es el círculo que pasa por él y tiene centro en el origen. El índice de la relación de equivalencia es infinito y coincide con la cardinalidad de $\mathbb{R}$.

Supongamos ahora que $(A, *, u)$ es un monoide. Una relación de equivalencia R en A se dice ser una *congruencia* si se cumple lo siguiente:

$$\forall x_1, x_2, y_1, y_2 \in A :\ (x_1 R x_2)\ \&\ (y_1 R y_2)\ \Rightarrow\ (x_1 * y_1 R x_2 * y_2)\,,$$

es decir, si "factores equivalentes dan productos equivalentes".

Proposición 1.2.8 *Sean $(S_1, *_1, u_1)$ y $(S_2, *_2, u_2)$ dos monoides y sea $h : S_1 \to S_2$ un homomorfismo. Entonces $núc(h)$ es una congruencia en S_1.*

En efecto, supongamos que $(x_1, x_2) \in núc(h)$ y $(y_1, y_2) \in núc(h)$. Hemos de probar que los respectivos productos coinciden, es decir, que $(x_1 *_1 y_1, x_2 *_1 y_2) \in núc(h)$. Pero esto es inmediato pues

$$h(x_1 *_1 y_1) = h(x_1) *_2 h(y_1) = h(x_2) *_2 h(y_2) = h(x_2 *_1 y_2).$$

Observación 1.2.2 *Si $(S, *, u)$ es un monoide y R es una congruencia en S, entonces podemos "calcar" la operación $*$ de S en una nueva operación en el cociente S/R de manera que este cociente sea un monoide y la proyección canónica un homomorfismo.*

En efecto, para cualesquiera dos clases de equivalencia $[a]_R, [b]_R$ definamos $[a]_R *_{S/R} [b]_R = [a * b]_R$. Observamos inmediatamente que

1. la definición de la operación $*_{S/R}$ no depende de los representantes que se elija en las clases $[a]_R, [b]_R$ pues R es una congruencia:

$$a_1 \in [a]_R, b_1 \in [b]_R\ \Rightarrow\ a_1 * b_1 \in [a * b]_R\ \Rightarrow\ [a_1 * b_1]_R = [a * b]_R,$$

2. la clase de equivalencia de u, $[u]_R$, es la unidad en el cociente S/R, y

3. la definición $[a]_R *_{S/R} [b]_R = [a*b]_R$ puede ser reescrita como $\pi_R(a*b) = \pi_R(a) *_{S/R} \pi_R((b)$, lo que a su vez indica que la proyección canónica es, efectivamente, un homomorfismo.

Ejemplos.

1. Se vió que $long : \Sigma^* \to I\!N$ es un homomorfismo. Dos palabras están en el núcleo de $long$, llamémoslo R, si y sólo si son de igual longitud. Por tanto, para cada palabra σ su clase de equivalencia $[\sigma]_R$ es el conjunto de palabras que tienen la misma longitud que σ. Si $n = long(\sigma)$ entonces podemos identificar a la clase $[\sigma]_R$ con el número n.

Para dos palabras $\sigma_1, \sigma_2 \in \Sigma^*$, con $n_1 = long(\sigma_1)$ y $n_2 = long(\sigma_2)$, tenemos que

$$n_1 *_{\Sigma^*/R} n_2 = long(\sigma_1\sigma_2) = n_1 + n_2,$$

es decir, la operación que induce la congruencia coincide con la suma usual de $I\!N$.

Así pues, podemos identificar naturalmente al monoide $(\Sigma^*/R, *_{\Sigma^*/R})$ con el monoide $(I\!N, +)$.

2. La función de paridad $par : (\Sigma^*, \cdot, nil) \to (Z\!\!Z_2, +, 0)$ es un homomorfismo. Una pareja de palabras está en el núcleo de par si y sólo si sus dos componentes son de igual paridad: O bien ambas son de longitud par, o bien ambas son de longitud impar. Escribamos

$$\begin{aligned} \overline{0} &= \{\sigma \in \Sigma^* | \text{la longitud de } \sigma \text{ es par}\}, \\ \overline{1} &= \{\sigma \in \Sigma^* | \text{la longitud de } \sigma \text{ es impar}\}. \end{aligned}$$

Entonces $\Sigma^*/n\acute{u}c(par) = \{\overline{0}, \overline{1}\}$, por lo que $n\acute{u}c(par)$ es de índice 2, y la operación que induce la concatenación es, precisamente

$*_{\Sigma^*/ker(par)}$	$\overline{0}$	$\overline{1}$
$\overline{0}$	$\overline{0}$	$\overline{1}$
$\overline{1}$	$\overline{1}$	$\overline{0}$

Así, se identifica $(\Sigma^*/n\acute{u}c(par), *_{\Sigma^*/ker(par)})$ con el monoide $(Z\!\!Z_2, +)$.

3. Sea $n > 0$ y sea $suf_n : \Sigma^* \to \Sigma^*$ el homomorfismo que trunca una palabra para quedarse con el sufijo de longitud a los sumo n.

Una pareja de palabras estará en el núcleo de suf_n sólo si ambas palabras poseen un sufijo común de longitud a lo más n. Así pues hay $\sum_{i=0}^{n} m^i = \frac{m^{n+1}-1}{m-1}$ clases de equivalencia para la relación $n\acute{u}c(suf_n)$, donde m es el número de símbolos en el alfabeto Σ.

La operación que induce la concatenación en el cociente $\Sigma^*/n\acute{u}c(suf_n)$ coincide con la operación definida en el monoide $\Sigma^{\leq n}$: Dadas dos palabras, las concatena primero y luego toma el sufijo de longitud n.

$(\Sigma^*/n\acute{u}c(suf_n), *_{\Sigma^*/ker(suf_n)})$ se identifica así con el monoide $\Sigma^{\leq n}$.

Sean $(S_1, *_1, u_1)$ y $(S_2, *_2, u_2)$ dos monoides y sea $h : S_1 \to S_2$ un homomorfismo. h se dice ser

- un *monomorfismo* si h es inyectiva, es decir, si $\forall s, t \in S_1 : h(s) = h(t) \Rightarrow s = t$,
- un *epimorfismo* si h es suprayectiva, es decir, si $\forall s_2 \in S_2 \exists s_1 \in S_1 : h(s_1) = s_2$, y
- un *isomorfismo* si h es a la vez un monomorfismo y un epimorfismo.

La *imagen* de un homomorfismo $h : S_1 \to S_2$ es $h(S_1) = \{s_2 \in S_2 | \exists s_1 \in S_1 : h(s_1) = s_2\}$.

Escribiremos $S_1 \equiv S_2$ para denotar el hecho de que los monoides S_1 y S_2 son *isomorfos*, es decir, de que existe un isomorfismo $h : S_1 \to S_2$.

Observación 1.2.3 *Las siguientes relaciones se siguen inmediatamente para un homomorfismo $h : S_1 \to S_2$:*

1. *$h(S_1)$ es un submonoide de S_2.*
2. *h es un monomorfismo si y sólo si $n\acute{u}c(h) = \{(s, s) | s \in S_1\}$, es decir, el núcleo de h coincide con la identidad.*

3. *h es un epimorfismo si y sólo si $S_2 = h(S_1)$*

Teorema 1.2.4 (del homomorfismo.) *Para cualquier epimorfismo $h : S_1 \to S_2$ se tiene $S_1/núc(h) \equiv S_2$.*

En efecto, siendo h un homomorfismo, se tiene que $núc(h)$ es una congruencia. Por tanto, el cociente $S_1/núc(h)$ posee una estructura de monoide. Consideremos la función $H : S_1/núc(h) \to S_2$, $[s] \mapsto h(s)$. Es claro que H está bien definida (es decir, $H([s])$ no depende del representante que se tome para la clase $[s]$), y que es un isomorfismo.

Ejemplo. Sea Σ un alfabeto y sea $f : \Sigma \to \Sigma^*$ una función de sustitución. Sea $T = (f(s))_{s\in\Sigma}$ el conjunto de palabras correspondientes a símbolos de Σ mediante la función f. Sea $f^* : \Sigma^* \to \Sigma^*$ el homomorfismo que extiende a f. Entonces $f^*(\Sigma^*) = T^* \subset \Sigma^*$. Ahora, una pareja (σ_1, σ_2) está en el núcleo de f^* si y sólo si bajo f^* ambas se convierten en una misma palabra de T^*. De acuerdo con el Primer Teorema Fundamental de Homomorfismos se tiene $\Sigma^*/núc(f^*) \equiv T^*$.

Proposición 1.2.9 *Sea $h : S_1 \to S_2$ un homorfismo de monoides. Entonces:*

1. *Si S_{11} es un submonoide de S_1 entonces $h(S_{11})$ es un submonoide de S_2 y $S_{11}/núc(h)$ es un submonoide isomorfo a $h(S_{11})$.*
2. *Si S_{21} es un submonoide de S_2 entonces $h^{-1}(S_{21}) = \{s_1 \in S_1 | h(s_1) \in S_{21}\}$ es un submonoide de S_1 y $h^{-1}(S_{21})/núc(h)$ es un submonoide isomorfo a S_{21}.*

1.3 Propiedades de lenguajes

1.3.1 Particiones en base a un lenguaje

Congruencia lateral inducida por un lenguaje

Sea Σ un alfabeto. Construimos en esta sección una relación de congruencia lateral en Σ^* que utilizaremos frecuentemente en este curso.

Sea L un lenguaje de Σ^*. La relación R_L que *induce* el lenguaje L se define como sigue:

$$\forall \sigma_1, \sigma_2 \in \Sigma^* : \ \sigma_1 R_L \sigma_2 \ \Leftrightarrow \ \forall \sigma \in \Sigma^* (\sigma_1\sigma \in L \Leftrightarrow \sigma_2\sigma \in L). \tag{1.1}$$

Es decir, dos palabras están relacionada según R_L si y sólo si, al añadírseles un mismo sufijo, o bien ambas palabras resultantes quedan en el lenguaje, o bien ambas dejan de pertenecer al lenguaje.

Es fácil ver que R_L es una relación de equivalencia. De hecho, es una congruencia *derecha* pues rige la implicación:

$$\forall \sigma_1, \sigma_2 \in \Sigma^* : \ \sigma_1 R_L \sigma_2 \ \Rightarrow \ \forall \sigma \in \Sigma^* (\sigma_1\sigma R_L \sigma_2\sigma).$$

El *índice del lenguaje* L es, por definición, el índice de la relación R_L.

La noción de índice es de particular importancia para analizar la *complejidad* del lenguaje.

El Teorema de Myhill-Nerode, que veremos más adelante, asevera que un lenguaje será regular si y sólo si es de índice finito.

Algunas variantes

Sea Σ un alfabeto, L un lenguaje de Σ^* y sea $k > 0$. La relación $R_{L,k}$ se define como sigue:

$$\forall \sigma_1, \sigma_2 \in \Sigma^* : \ \sigma_1 R_L \sigma_2 \ \Leftrightarrow \ \forall \sigma \in \Sigma^* \left(\text{long}(\sigma) \leq k \Rightarrow (\sigma_1\sigma \in L \Leftrightarrow \sigma_2\sigma \in L)\right). \tag{1.2}$$

Es decir, dos palabras están relacionada según $R_{L,k}$ si y sólo si, al añadírseles un mismo sufijo de longitud a lo sumo k, o bien ambas palabras resultantes quedan en el lenguaje, o bien ambas dejan de pertenecer al lenguaje.

Es fácil ver que $R_{L,k}$ es una relación de equivalencia, aunque no es una congruencia. Esta relación puede ser de índice finito, sin que ello implique que el lenguaje L sea regular.

1.3.2 Propiedades de cerradura

Una clase de lenguajes $\mathcal{L}$ es un *álgebra* si es cerrada por unión y complemento y si contiene al conjunto vacío. Es decir, si se cumplen las propiedades

$$\begin{aligned} & \emptyset \in \mathcal{L} \\ L \in \mathcal{L} \quad &\Rightarrow \quad L^c = \Sigma^* - L \in \mathcal{L} \\ L_1, L_2 \in \mathcal{L} \quad &\Rightarrow \quad L_1 \cup L_2 \in \mathcal{L} \end{aligned}$$

En una clases de lenguajes, además de caracterizar la solubilidad de los problemas de la palabra y de derivación, mencionados en la Introducción del curso, también es importante decidir si acaso una clase dada de lenguajes en un álgebra.

Por ejemplo, la clase de lenguajes regulares sobre un alfabeto finito forma un álgebra de conjuntos.

La clase de lenguajes recursivamente enumerables, es decir, aquellos que son generados por gramáticas irrestrictas, contiene al conjunto vacío, y la unión y la intersección de cualesquiera dos lenguajes en la clase están en la clase, sin embargo no ocurre lo mismo para la operación de complementación.

En general, si Φ es un operador de aridad k, que transforma k lenguajes dados en algún otro lenguaje, un problema importante para una clase $\mathcal{L}$ de lenguajes es decidir cuándo esa clase es *cerrada* bajo el operador, es decir, cuándo rige la implicación:

$$L_1, \ldots, L_k \in \mathcal{L} \;\Rightarrow\; \Phi(L_1, \ldots, L_k) \in \mathcal{L}.$$

La siguiente observación se sigue inmediatamente de las conocidas Leyes de De Morgan, válidas en el álgebra de conjuntos:

Observación 1.3.1 *Una clase de lenguajes $\mathcal{L}$ es un álgebra si y sólo si*

1. *contiene al lenguaje vacío, $\emptyset$,*
2. *es cerrada bajo la unión, y*
3. *es cerrada bajo la operación de complementación.*

Hemos ya visto algunos de los operadores más usuales entre lenguajes:

Unión. $L_1 + L_2 = L_1 \cup L_2 = \{\sigma | (\sigma \in L_1) \text{ o } (\sigma \in L_2)\}$: palabras que están en L_1 o en L_2.

Intersección. $L_1 \cap L_2 = \{\sigma | (\sigma \in L_1) \text{ y } (\sigma \in L_2)\}$: palabras que están en ambos lenguajes L_1 y L_2.

Diferencia. $L_1 - L_2 = \{\sigma | (\sigma \in L_1) \text{ pero } (\sigma \notin L_2)\}$: palabras que están en L_1 pero no en L_2.

Complemento. $L^c = \Sigma^* - L$: palabras en el diccionario que no están en L.

Concatenación. $L_1 \cdot L_2 = \{\sigma | \exists \sigma_1 \in L_1, \sigma_2 \in L_2 : \sigma = \sigma_1 \cdot \sigma_2\}$: palabras formadas al concatenar partículas en L_1 y en L_2.

0-ésima potencia. $L^0 = \{nil\}$: mónada consistente de la palabra vacía.

m-ésima potencia ($m > 0$). $L^m = \{\sigma | \exists \sigma_1, \ldots, \sigma_m \in L : \sigma = \sigma_1 \cdots \sigma_m\}$: palabras que se forman al concatenar m partículas en L.

Operador "estrella" (de Kleene). $L^* = \bigcup_{n \geq 0} L^n$: palabras que se forman al concatenar un conjunto finito de partículas en L, incluída la palabra vacía.

Operador "más". $L^+ = \bigcup_{n > 0} L^n = L^* - L^0$: palabras que se forman al concatenar un conjunto finito de partículas en L.

Mas también los siguientes serán utilizados en este curso:

i-ésima m-ava parte. A cada palabra del lenguaje se la divide en partes iguales y se toma la i-ésima:

$$\left(i, \frac{1}{m}\right) L = \{\sigma_i | \forall j \in ([\![1,m]\!] - \{i\}) \exists \sigma_j \in L : \mathrm{long}(\sigma_j) = \mathrm{long}(\sigma_i) \ \& \ \sigma_1 \cdots \sigma_i \cdots \sigma_m \in L\} :$$

palabras que se forman al "dividir" en m partes de igual longitud a palabras en L, $1 \leq i \leq m$.

m-ésimas repeticiones. $L^{(m)} = \{\sigma | \exists \sigma_1 \in L : \ \sigma = \sigma_1^m\}$: palabras que se forman al concatenar m copias de una misma partícula en L.

Raíz m-ésima. $\sqrt[m]{L} = \{\sigma | \sigma^m \in L\}$: palabras tales que al concatenar m copias de ellas producen una palabra en L. Por tanto, $L = \left(\sqrt[m]{L}\right)^{(m)}$.

Cociente izquierdo. $L_1 \backslash L_2 = \{\sigma | \exists \sigma_2 \in L_2 : \sigma\sigma_2 \in L_1\}$: prefijos que con sufijos en L_2 dan palabras en L_1.

Derivada izquierda. $\partial_\tau^{izq}(L) = L \backslash \{\tau\} = \{\sigma | \sigma\tau \in L\}$: prefijos que con τ dan palabras en L.

Cociente derecho. $L_1 / L_2 = \{\sigma | \exists \sigma_2 \in L_2 : \sigma_2\sigma \in L_1\}$: sufijos que con prefijos en L_2 dan palabras en L_1.

Derivada derecha. $\partial_\tau^{der}(L) = L / \{\tau\} = \{\sigma | \tau\sigma \in L\}$: sufijos que con τ dan palabras en L.

Reverso. $L^{rev} = \{\sigma^{rev} | \sigma \in L\}$.

1.4 Ejercicios

1. Proporcione tres ejemplos de monoides finitos mostrando sus respectivas tablas de operación.

2. En cada una de las operaciones definidas a continuación, sobre el conjunto de número naturales $\mathbb{N} = \{0, 1, 2, \ldots\}$, decida si es acaso asociativa o si posee una unidad lateral:

$$\begin{array}{rlcl}
i. & x * y & = & \mathrm{Max}\{x, y\} \\
ii. & x * y & = & \mathrm{Min}\{x, y+2\} \\
iii. & x * y & = & x + y + 3 \\
iv. & x * y & = & x + 2y \\
v. & x * y & = & \left\{ \begin{array}{ll} \mathrm{Min}\{x, y\} & \text{si } \mathrm{Min}\{x, y\} < 10, \\ \mathrm{Max}\{x, y\} & \text{si } \mathrm{Min}\{x, y\} \geq 10. \end{array} \right.
\end{array}$$

3. Sea S un conjunto no vacío dotado de una operación binaria $* : S^2 \to S$ tal que

$$\forall x, y \in S : \quad x * y = x.$$

Muestre que $*$ es una operación asociativa.

4. Suponga que en un monoide $(S, *)$ se cumplen las relaciones siguientes:

$$\begin{array}{rlrcl}
i. & \forall x & x * x & = & x \quad (* \text{ es } \textit{idempotente},) \\
ii. & \forall x, y, z, w \quad (x * y) * (z * w) & = & & (x * z) * (y * w)
\end{array}$$

Muestre que entonces se cumple la relación

$$\forall x, y, z : \quad x * (y * z) = (x * y) * (x * z).$$

Dé un ejemplo que ilustre este ejercicio.

5. Sea $(S, *)$ un monoide y $a \in S$ un elemento cualquiera. Se define una nueva operación "$\bullet$" haciendo

$$\forall x, y : \quad x \bullet y = x * a * y.$$

Muestre que esta nueva operación es asociativa. Dé condiciones sobre a suficientes para que esta operación posea unidad, y en tal caso caracterice a las unidades.

6. Muestre que si $(S, *)$ es una estructura asociativa finita, entonces posee un *elemento idempotente*, es decir,

$$\exists a \in S : \quad a * a = a.$$

7. Sea $n > 1$ un entero positivo. Para cada entero x sea $\log_n(x)$ el mínimo entero k tal que $n^k > x$. $\log_n(x)$ se dice ser la *longitud* de x en base n.

Dados $x, y \in \mathbb{N}$ definimos $x * y = x \cdot n^{\log_n(x)} + y$.

Pruebe que $(\mathbb{N}, *)$ es un monoide.

8. Sean R y S dos relaciones de equivalencia sobre un conjunto dado. Sea $T = R \cap S$, es decir,

$$\forall x, y : \quad xTy \Leftrightarrow (xRy)\&(xSy).$$

a) Pruebe que T es una relación de equivalencia.

b) Exprese al índice de T en términos de los índices de R y S.

c) Sea $U = R \cup S$. Decida si acaso U es una relación de equivalencia y justifique su decisión.

9. En $\mathbb{N}$, considere la relación,

$$R = \{(x, y) | x - y \text{ es positivo e impar } \}.$$

Decida si acaso R es reflexiva, o simétrica, o transitiva, o antisimétrica, o de equivalencia o de orden.

10. En $(0 + 1)^*$, considere la relación,

$$R = \{(\sigma, \tau) | \sigma \text{ y } \tau \text{ poseen el mismo número de 0's } \}.$$

Decida si acaso R es reflexiva, o simétrica, o transitiva, o antisimétrica, o de equivalencia o de orden.

11. En un conjunto de 10 elementos,

a) cuente el número de relaciones simétricas y bosqueje un procedimiento para generar a todas ellas, una a una,

b) cuente el número de relaciones reflexivas y bosqueje un procedimiento para generar a todas ellas, una a una,

c) cuente el número de relaciones transitivas y bosqueje un procedimiento para generar a todas ellas, una a una.

12. Sea R una relación de equivalencia y sea

$$S = \{(x, y) | \exists z \in S : \ (xRz)\&(zRy)\}.$$

Pruebe que S es una relación de equivalencia. Compare el índice de S con el de R.

13. Dé un ejemplo de una relación definida sobre algún conjunto que sea simétrica y transitiva mas no reflexiva.

14. Dos palabras $\sigma, \tau \in \Sigma^*$ se dicen *conjugadas* si $\exists \upsilon \in \Sigma^* : \ \sigma\upsilon = \upsilon\tau$.

Pruebe que dos palabras $\sigma, \tau \in \Sigma^*$ son conjugadas si y sólo si

$$\exists \alpha, \beta \in \Sigma^* : \ (\sigma = \alpha\beta)\&(\tau = \beta\alpha).$$

15. Muestre que la relación de *conjugación* es una relación de equivalencia en el diccionario de cualquier alfabeto.

16. Muestre que si $\sigma = \alpha^n$, donde $\sigma, \alpha \in \Sigma^*$ y $n \geq 0$, entonces cualquier conjugado de σ es de la forma $\tau = \beta^n$, donde β es un conjugado de α.

17. Sea Σ un alfabeto con $m > 0$ símbolos. Sea $\omega : \Sigma \to [1, m]$ una enumeración de Σ. Se tiene un orden natural en Σ: $s \leq t \Leftrightarrow \omega(s) \leq \omega(t)$.

a) Extendemos ω a una función $\omega^* : \Sigma^* \to \mathbb{Q}^+$, donde $\mathbb{Q}^+$ es el conjunto de números racionales positivos, haciendo

$$\begin{aligned} \omega^*(nil) &= 0 \\ \omega^*(s_1 \cdots s_k) &= \sum_{i=1}^{k} \frac{\omega(s_i)}{(m+1)^{i-1}}. \end{aligned}$$

Decida si acaso ω^* es una biyección.

b) Pruebe que la relación $\leq_\omega$ en Σ^* definida como

$$\forall \sigma, \tau : \quad \sigma \leq_\omega \tau \Leftrightarrow \omega^*(\sigma) \leq \omega^*(\tau),$$

es una relación de orden en Σ^* que coincide con el orden lexicográfico inducido por ω en Σ^*.

c) ¿Si se definiera $\omega^*(s_1 \cdots s_k) = \sum_{i=1}^{k} \frac{\omega(s_i)}{(m)^{i-1}}$, podría concluirse lo mismo que en el inciso anterior ?

18. Decida cuáles de las siguientes relaciones se cumplen para cualesquiera tres lenguajes $L_1, L_2, L_3 \subset \Sigma^*$:

$$\begin{aligned} L_1 \cap L_2 &= L_2 \cap L_1 \\ L_1(L_2 \cup L_3) &= L_1L_2 \cup L_1L_3 \\ L_1L_2 &= L_2L_1 \\ L_1(L_2 \cap L_3) &= L_1L_2 \cap L_1L_3 \end{aligned}$$

19. Sea $h : \Sigma^* \to \Sigma^*$ un homomorfismo. Decida cuáles de las siguientes relaciones se cumplen para cualesquiera dos lenguajes $L_1, L_2 \subset \Sigma^*$:

$$\begin{aligned} h(h(L_1)) &= h(L_1) \\ h(L_1L_2) &= h(L_1)h(L_1) \\ h(L_1 \cap L_2) &= h(L_1) \cap h(L_1) \\ h(L_1 \cup L_2) &= h(L_1) \cup h(L_1) \end{aligned}$$

20. Sea $h : (0+1)^* \to (0+1)^*$ el homomorfismo tal que $h(0) = 1, h(1) = 01$. Sea $f : \mathbb{N} \to \mathbb{N}$ la función $n \mapsto f(n) = \text{long}(h^n(01))$. Dé una expresión aritmética para calcular f. Demuestre la validez de su expresión.

Capítulo 2

Gramáticas formales

2.1 Conceptos básicos de gramáticas

Una *gramática* es una estructura $G = (V, T, P, s_0)$ donde

V	:	es un conjunto de símbolos *variables*,
T	:	es un conjunto de símbolos *terminales*, el conjunto $\Sigma = V \cup T$ se dice ser el *alfabeto* de la gramática,
$s_0 \in V$	:	es el símbolo inicial, y
$P \in V$	:	es un conjunto de parejas $(\alpha, \beta) \in (\Sigma^* - T^*) \times \Sigma^*$ llamadas *producciones* o *reglas sintácticas*.

Una pareja $(\alpha, \beta) \in P$ se escribe como $\alpha \to \beta$, o bien $\alpha ::= \beta$. Se dice que α es el *antecedente* de la regla y que β es su *consecuente*. Si $(\alpha, \beta_1), (\alpha, \beta_2), \ldots, (\alpha, \beta_k) \in P$ es un conjunto de reglas con un antecedente común, se simplifica la notación y se escribe $\alpha \to \beta_1|\beta_2|\cdots|\beta_k$, o alternativamente $\alpha ::= \beta_1|\beta_2|\cdots|\beta_k$.

Dadas dos palabras $\sigma, \tau \in \Sigma^*$ decimos que τ *se sigue* de σ en G, y escribimos $G \vdash (\sigma \to \tau)$, o simplemente $\sigma \to \tau$, cuando τ resulte de σ al intercambiar una partícula de σ, que sea el antecedente de una regla, por el respectivo consecuente de la regla. En símbolos,

$$G \vdash \sigma \to \tau \;\Leftrightarrow\; \exists \sigma_1, \sigma_2, \alpha, \beta : (\alpha, \beta) \in P \;\&\sigma = \sigma_1 \alpha \sigma_2 \;\&\tau = \sigma_1 \beta \sigma_2. \tag{2.1}$$

Decimos que τ *se deriva* de σ en G, y escribimos $G \vdash (\sigma \overset{*}{\to} \tau)$, o simplemente $\sigma \overset{*}{\to} \tau$, si existe una sucesión finita de palabras, cuyos primero y último elementos coinciden con σ y τ respectivamente, y cualquier elemento en la sucesión se sigue del anterior. En símbolos,

$$G \vdash \sigma \overset{*}{\to} \tau \;\Leftrightarrow\; \exists \sigma_1, \sigma_2, \ldots, \sigma_n : \sigma = \sigma_1 \;\&\tau = \sigma_n \;\&\forall i < n : G \vdash (\sigma_i \to \sigma_{i+1}) \tag{2.2}$$

En tal caso, la sucesión de palabras y producciones $\sigma_1 P_1 \sigma_2 \cdots \sigma_{n-1} P_{n-1} \sigma_n$, donde σ_{i+1} se sigue de σ_i precisamente por la aplicación de la producción P_i, es una *derivación* de τ a partir de σ. Así pues, la relación "*se_deriva*" es la cerradura reflexivo-transitiva de la relación "*se_sigue*"

El *lenguaje* de la gramática es el conjunto de palabras formadas con símbolos terminales que se derivan del símbolo inicial,

$$L(G) = \{\sigma \in T^* | G \vdash (s_0 \overset{*}{\to} \sigma)\}.$$

2.2 Ejemplos de gramáticas

2.2.1 Proposiciones bien formadas

En el Cálculo Proposicional, las *proposiciones* se construyen a partir de un conjunto de *variables proposicionales* siguiendo un conjunto de *reglas gramaticales*:

i) Una variable proposicional es una proposición.

ii) La negación de una proposición es una proposición también.

iii) La conjunción, la disyunción, la implicación y la equivalencia de dos proposiciones es también una proposición.

iv) Las proposiciones se obtienen sólo mediante la aplicación sucesiva de las reglas anteriores.

Para describir esta construcción mediante una gramática formal, podemos considerar como conjunto de símbolos terminales al conjunto unión de los siguientes:

$$\begin{array}{lcll} \mathit{VarsProp} & = & \{x_0, x_1, x_2, \ldots\} & : \text{ variables proposicionales,} \\ \mathit{Especial} & = & \{(,), \neg, \wedge, \vee, \rightarrow, \leftrightarrow\} & : \text{ especiales.} \end{array}$$

Como es convencional, los conectivos tienen asociada una *prioridad*:

Prior	Conec
1	$\neg$
2	$\wedge, \vee$
3	$\rightarrow, \leftrightarrow$

donde un valor menor de *Prior* significa que el correspondiente conectivo se aplica más rápido, y, en igualdad de prioridades, el orden de izquierda a derecha determina el de aplicación de los conectivos. Así,

$$\neg x_0 \vee x_2 \wedge x_3 \vee x_5 \rightarrow x_6 \leftrightarrow x_7$$

se ha de interpretar como

$$[((\neg x_0 \vee x_2) \wedge x_3) \vee x_5 \rightarrow x_6] \leftrightarrow x_7.$$

Introduzcamos entonces los siguientes símbolos variables para la gramática formal:

$$\begin{array}{ll} \langle \mathit{VarsProp} \rangle & : \text{ variables proposicionales,} \\ \langle \mathit{Prop} \rangle & : \text{ proposiciones bien formadas,} \\ \langle \mathit{PropNeg} \rangle & : \text{ proposiciones cuyo conectivo principal es la negación,} \\ \langle \mathit{PropCon} \rangle & : \text{ proposiciones cuyo conectivo principal es la conjunción,} \\ \langle \mathit{PropDis} \rangle & : \text{ proposiciones cuyo conectivo principal es la disyunción,} \\ \langle \mathit{PropImp} \rangle & : \text{ proposiciones cuyo conectivo principal es la implicación,} \\ \langle \mathit{PropEqu} \rangle & : \text{ proposiciones cuyo conectivo principal es la equivalencia.} \\ \langle \mathit{Prop1} \rangle & : \text{ proposiciones cuyo conectivo principal tiene prioridad 1,} \\ \langle \mathit{Prop2} \rangle & : \text{ proposiciones cuyo conectivo principal tiene prioridad 2,} \\ \langle \mathit{Prop3} \rangle & : \text{ proposiciones cuyo conectivo principal tiene prioridad 3,} \end{array}$$

En la tabla (2.1) presentamos las producciones de la gramática PBF (de Proposiciones Bien Formadas).

El lenguaje generado pr PBF consta de todas las proposiciones bien formadas a partir de las variables proposicionales $x_1, \ldots, x_n$.

En la presentación de las siguientes gramáticas, utilizaremos símbolos mayúsculos para denotar a los símbolos variables y minúsculos para los terminales.

2.2.2 Tercetas de igual longitud

Sea $L_3 = \{a^k b^k c^k | k \geq 1\}$ el lenguaje que consta de tres bloques consecutivos de a's, b's y c's de iguales longitudes. Construiremos una gramática para generar este lenguaje.

Consideremos las producciones siguientes:

1.	S	$\rightarrow$	A	*inicio*
2.	A	$\rightarrow$	$aABC$	*añádase a-es y por cada una, ha de añadirse una b y una c*
3.	A	$\rightarrow$	abC	*no se añada más a-es y equilíbrese a la primera con una b y una c*
4.	CB	$\rightarrow$	BC	*córranse las c-es a la derecha de las b-es*
5.	bB	$\rightarrow$	bb	*cámbiese las B-es variables por b-es terminales*
6.	bC	$\rightarrow$	bc	*comiéncese a cambiar las C-es*
7.	cC	$\rightarrow$	cc	*cámbiese las C-es variables por c-es terminales*

$$
\begin{array}{rrll}
\forall i: & \langle VarsProp\rangle & ::= & x_i \\
 & \langle Prop\rangle & ::= & \langle Prop3\rangle|\langle Prop2\rangle|\langle Prop1\rangle \\
 & \langle Prop3\rangle & ::= & \langle PropImp\rangle|\langle PropEqu\rangle \\
 & \langle Prop2\rangle & ::= & \langle PropCon\rangle|\langle PropDis\rangle \\
 & \langle Prop1\rangle & ::= & \langle VarsProp\rangle|\langle PropNeg\rangle|(\langle Prop\rangle) \\
 & \langle PropNeg\rangle & ::= & \neg\langle Prop1\rangle|\neg(\langle Prop\mathrm{J}\rangle) \qquad \text{con } J \leq 2, \\
 & \langle PropCon\rangle & ::= & \langle Prop\mathrm{K}\rangle \wedge \langle Prop1\rangle| \qquad \text{con } K \leq 2, \\
 & & & (\langle Prop3\rangle) \wedge \langle Prop1\rangle| \\
 & & & \langle Prop\mathrm{K}\rangle) \wedge (\langle Prop\mathrm{J}\rangle)| \qquad \text{con } K \leq 2, J \geq 2, \\
 & & & (\langle Prop3\rangle) \wedge (\langle Prop\mathrm{J}\rangle)| \qquad \text{con } J \geq 2, \\
 & \langle PropDis\rangle & ::= & \langle Prop\mathrm{K}\rangle \vee \langle Prop1\rangle| \qquad \text{con } K \leq 2, \\
 & & & (\langle Prop3\rangle) \vee \langle Prop1\rangle| \\
 & & & \langle Prop\mathrm{K}\rangle) \vee (\langle Prop\mathrm{J}\rangle)| \qquad \text{con } K \leq 2, J \geq 2, \\
 & & & (\langle Prop3\rangle) \vee (\langle Prop\mathrm{J}\rangle)| \qquad \text{con } J \geq 2, \\
 & \langle PropImp\rangle & ::= & \langle Prop\mathrm{K}\rangle \to \langle PropJ\rangle| \qquad \text{con } K \leq 3, J < 3, \\
 & & & \langle Prop\mathrm{K}\rangle \to (\langle Prop3\rangle)| \qquad \text{con } K \leq 3, \\
 & \langle PropEqu\rangle & ::= & \langle Prop\mathrm{K}\rangle \leftrightarrow \langle PropJ\rangle| \qquad \text{con } K \leq 3, J < 3, \\
 & & & \langle Prop\mathrm{K}\rangle \leftrightarrow (\langle Prop3\rangle)| \qquad \text{con } K \leq 3,
\end{array}
$$

Tabla 2.1: Gramática de proposiciones bien formadas.

En la tabla (2.2) vemos, a manera de ejemplo, la generación de la palabra $a^3b^3c^3 = aaabbbccc$. Con la gramática descrita, tenemos, efectivamente, que

i) toda palabra generada en ella tiene una longitud múltiplo de 3, y

ii) toda palabra generada es de la forma $a^kb^kc^k$ para algún $k \geq 0$.

Así pues, el lenguaje generado por la gramática es $L(\text{gramática}) = L_3$.

$\underline{S}$	$aaab\ \underline{CB}\ CBC$	$aaab\ \underline{bB}\ CCC$
$\uparrow$	$\uparrow$	$\uparrow$
1	4	5
$\underline{A}$	$aaabBC\ \underline{CB}\ C$	$aaabb\ \underline{bC}\ CC$
$\uparrow$	$\uparrow$	$\uparrow$
2	4	6
$a\ \underline{A}\ BC$	$aaabB\ \underline{CB}\ CC$	$aaabbb\ \underline{cC}\ C$
$\uparrow$	$\uparrow$	$\uparrow$
2	4	6
$aa\ \underline{A}\ BCBC$	$aaa\ \underline{bB}\ BCCC$	$aaabbbc\ \underline{cC}$
$\uparrow$	$\uparrow$	$\uparrow$
3	5	6
		$aaabbbccc$

Tabla 2.2: Un ejemplo para la gramática de L_3.

1.	S	$\to$	ABC	*inicio*
2.	AB	$\to$	$0AD$	*añádase un 0*
3.	DC	$\to$	$B0C$	*repítase un 0*
4.	$D0$	$\to$	$0D$	*avánce se a la derecha, recordando un 0*
5.	$D1$	$\to$	$1D$	*avánce se a la derecha, recordando un 0*
6.	AB	$\to$	$1AE$	*añádase un 1*
7.	EC	$\to$	$B1C$	*repítase un 1*
8.	$E0$	$\to$	$0E$	*avánce se a la derecha, recordando un 1*
9.	$E1$	$\to$	$1E$	*avánce se a la derecha, recordando un 1*
10.	$0B$	$\to$	$B0$	*retrocédase a la izquierda*
11.	$1B$	$\to$	$B1$	*retrocédase a la izquierda*
12.	AB	$\to$	*nil*	*termínese*
13.	C	$\to$	*nil*	*termínese*

Tabla 2.3: Gramática que genera repeticiones de palabras.

2.2.3 Parejas de igual longitud

Sea $L_2 = \{a^k b^k | k \geq 1\}$. Construiremos una gramática para este lenguaje siguiendo el esquema de la gramática anterior. Consideremos las siguientes producciones:

$$\left.\begin{array}{llll} 1. & S \to A & & \\ 2. & A \to aABC & & \\ 3. & A \to abC & : & \textit{actúan como en la gramática anterior,} \\ 4. & CB \to BC & & \\ 5. & bB \to bb & & \end{array}\right\}$$

$$6.\quad cC \to b \qquad : \quad \textit{omite las c-es.}$$

Efectivamente, esta nueva gramática genera a L_2: En cada generación, genera una palabra de L_3 y luego suprime el último bloque de c's.

Evidentemente, L_2 se genera también por la gramática $S \to aSb|ab$.

Así pues, un mismo lenguaje puede ser generado por más de una gramática.

2.2.4 Palabras dobles

Sea $L = (0+1)^{(2)} = \{\sigma\sigma | \sigma \in (0+1)^*\}$ el lenguaje que consta de las palabras formadas por la repetición de una partícula en el alfabeto $\{0, 1\}$.

Una gramática que genera al lenguaje L es la gramática G mostrada en la tabla (2.3).

Una derivación de la palabra $1001\,1001 \in L$ se muestra en la tabla (2.4).

Del ejemplo mostrado y de las producciones en la tabla (2.3) podemos ver que los símbolos tienen los efectos siguientes:

S : símbolo inicial,
A : delimitador derecho del primer bloque,
B : "cursor" para seguir añadiendo símbolos en los bloques generados,
C : delimitador derecho del segundo bloque,
D : recordatorio de que se ha generado un 0,
E : recordatorio de que se ha generado un 1.

El procedimiento seguido por la gramática es el siguiente:

1. Del símbolo inicial genera un área de trabajo, ABC: a la izquierda de A está la palabra vacía y a la izquierda de C también está la palabra vacía.

2. Cada vez que A y B sean adyacentes,

$\underset{\uparrow}{\underline{S}}$	$10A\ \underset{\uparrow}{\underline{1B}}\ 0C$	$100A\ \underset{\uparrow}{\underline{1B}}\ 00C$	$1001A10\ \underset{\uparrow}{\underline{0B}}\ 1C$
1	11	11	10
$\underset{\uparrow}{\underline{AB}}\ C$	$10\ \underset{\uparrow}{\underline{AB}}\ 10C$	$100\ \underset{\uparrow}{\underline{AB}}\ 100C$	$1001A1\ \underset{\uparrow}{\underline{0B}}\ 01C$
6	2	6	10
$1A\ \underset{\uparrow}{\underline{EC}}$	$100A\ \underset{\uparrow}{\underline{D1}}\ 0C$	$1001A\ \underset{\uparrow}{\underline{E1}}\ 00C$	$1001A\ \underset{\uparrow}{\underline{1B}}\ 001C$
7	5	9	11
$1\ \underset{\uparrow}{\underline{AB}}\ 1C$	$100A1\ \underset{\uparrow}{\underline{D0}}\ C$	$1001A1\ \underset{\uparrow}{\underline{E0}}\ 0C$	$1001\ \underset{\uparrow}{\underline{AB}}\ 1001C$
2	5	8	12
$10A\ \underset{\uparrow}{\underline{D1}}\ C$	$100A10\ \underset{\uparrow}{\underline{DC}}$	$1001A10\ \underset{\uparrow}{\underline{E0}}\ C$	$10011001\ \underset{\uparrow}{\underline{C}}$
5	3	8	13
$10A1\ \underset{\uparrow}{\underline{DC}}$	$100A1\ \underset{\uparrow}{\underline{0B}}\ 0C$	$1001A100\ \underset{\uparrow}{\underline{EC}}$	10011001
3	10	7	

Tabla 2.4: Un ejemplo en la generación de repeticiones.

(a) genera, inmediatamente a la izquierda de A, un símbolo $x \in \{0,1\}$ y su correspondiente recordatorio X, $X = D$ si $x = 0$ y $X = E$ si $x = 1$,

(b) traslada X hacia la izquierda de C,

(c) en esa posición escribe x y recupera a B,

(d) traslada B a ser adyacente a A,

3. o bien suprime AB cuando ya no vaya a generar más símbolos.

4. Suprime C.

2.2.5 Elevación al cuadrado

Construiremos una gramática G que genere a las cadenas de 1's cuya longitud es el cuadrado de algún número natural positivo. Es decir, G ha de ser tal que $L(G) = L$ donde $L = \{1^{n^2} | n \geq 1\}$.

Ya que para todo n, $(n+1)^2 = n^2 + 2n + 1 = n^2 + n + (n+1)$ y, al inicio, $1^2 = 1$, la gramática, una vez que genera una cadena de longitud n^2 ha de concatenarla con una de longitud n y otra de longitud $n+1$.

Con esta motivación en mente, sea G la gramática cuyas producciones se presentan en la tabla (2.5) y cuyo símbolo inicial es S.

Observaciones:

1. En la tabla (2.6) se muestra la derivación de la cadena de longitud 25.

2. Z es un delimitador derecho de cualquier cadena generada.

3. Un *ciclo de generación* se obtiene entre dos derivaciones con la cadena WZ como sufijo.

4. La generación de las cadenas de longitudes n^2, $2 \leq n \leq 7$, se muestra en la tabla (2.7).

5. Ahí las cadenas generadas son de la forma $1V(VC^+)^*WZ$. Sea σ_n la cadena intermedia de la forma $V(VC^+)^*$. Para $n > 2$ escribamos $\sigma_n = \tau_n VC^{2(n-2)}$. Entonces

$$\begin{array}{lcllcl} \tau_2 & = & nil & ; \sigma_2 & = & \tau_2 VC^0 \\ \tau_{n+1} & = & \tau_n VC^{n-3} VC^{n-2} & ; \sigma_{n+1} & = & \tau_{n+1} VC^{2(n-1)} \end{array}$$

Se ve que $\forall n: \quad \text{long}(\sigma_n) = n^2 - 3$.

$S \to 1$ $S \to 1VWZ$	:	inicio
$V1 \to 11$ $C1 \to 11$ $WZ \to 11$	:	terminación
$WZ \to ACVZ$	:	inicio reiteración
$VT \to TCV$ $CT \to ACV$	:	incrementación
$VA \to TC$ $CA \to AC$	:	transposición de A's
$WC \to VB$ $BC \to CB$ $BV \to CW$	:	transposición de B's
$1T \to 1VVCW$	:	fin semireiteración

Tabla 2.5: Gramática para elevar al cuadrado en expresión unaria.

6. $G \vdash \sigma_n WZ \stackrel{*}{\to} \sigma_{n+1} WZ$.
7. Una producción inicial genera: $1 \cdot \sigma_2 WZ$.
8. Consecutivamente, se forma cualquier cadena de la forma $1 \cdot \sigma_n WZ$.
9. Al final a WZ se le cambia por 11 y a todas las V's y C's se les cambia también por 1.

2.3 Equivalencia de gramáticas

Sean G_1 y G_2 dos gramáticas con un mismo conjunto de símbolos terminales. Diremos que G_2 *subsume* a G_1 si toda palabra generada en G_1 es también generada en G_2, es decir, si $L(G_1) \subset L(G_2)$. Escribiremos en este caso $G_1 \leq G_2$.

Proposición 2.3.1 *La relación de* subsunción *es una relación reflexiva y transitiva en la clase de gramáticas con un alfabeto terminal fijo.*

La subsunción no es antisimétrica pues dos gramáticas distintas pueden generar a un mismo lenguaje.

Una condición suficiente, mas no necesaria, para revisar si acaso una gramática queda subsumida por otra la da la siguiente

Observación 2.3.1 *Sean $G_1 = (V, T, P_1, s_0)$ y $G_2 = (V, T, P_2, s_0)$ dos gramáticas con un mismo alfabeto. Si para cada producción $(\alpha, \beta) \in P_1$ se tiene que $G_2 \vdash (\alpha \to \beta)$, entonces $G_1 \leq G_2$.*

Y, en efecto, si $\sigma \in L(G_1)$ y $\sigma_0 P_{i_1} \sigma_1 \cdots \sigma_{k-1} P_{i_k} \sigma_k$ es una derivación de σ a partir del símbolo inicial s_0 en G_1, y para cada j, σ_j se deriva de σ_{j-1} mediante una derivación $\tau_{j0} Q_{j_1} \tau_{j1} \cdots \tau_{j,k_j-1} Q_{j_{k_j}} \tau_{j,k_j}$ en G_2, entonces la concatenación de estas últimas derivaciones ha de dar una derivación de σ a partir del símbolo inicial s_0 en G_2.

Dos gramáticas son *equivalentes* si cualquiera de ellas subsume a la otra, es decir,

$$G_1 \equiv G_2 \;\Leftrightarrow\; L(G_1) = L(G_2).$$

	1VVCVCVCCVCCCCACVZ
	1VVCVCVCCVCCCACCVZ
	1VVCVCVCCVCCCACCCVZ
	1VVCVCVCCVCACCCCVZ
	1VVCVCVCCVACCCCCVZ
	1VVCVCVCCTCCCCCCVZ
	1VVCVCVCACVCCCCCCVZ
	1VVCVCVACCVCCCCCCVZ
S	1VVCVCTCCCVCCCCCCVZ
1VWZ	1VVCVACVCCCVCCCCCCVZ
1VACVZ	1VVCTCCVCCCVCCCCCCVZ
1TCCVZ	1VVACVCCVCCCVCCCCCCVZ
1VVCWCCVZ	1VTCCVCCVCCCVCCCCCCVZ
1VVCVBCVZ	1TCVCCVCCVCCCVCCCCCCVZ
1VVCVCBVZ	1VVCWCVCCVCCVCCCVCCCCCCCVZ
1VVCVCCWZ	1VVCVBVCCVCCVCCCVCCCCCCCVZ
1VVCVCCACVZ	1VVCVCWCCVCCVCCCVCCCCCCCVZ
1VVCVCACCVZ	1VVCVCVBCVCCVCCCVCCCCCCCVZ
1VVCVACCCVZ	1VVCVCVCBVCCVCCCVCCCCCCCVZ
1VVCTCCCCVZ	1VVCVCVCCWCCVCCCVCCCCCCCVZ
1VVACVCCCCVZ	1VVCVCVCCVBCVCCCVCCCCCCCVZ
1VTCCVCCCCVZ	1VVCVCVCCVCBVCCCVCCCCCCCVZ
1TCVCCVCCCCVZ	1VVCVCVCCVCCWCCCVCCCCCCCVZ
1VVCWCVCCVCCCCCVZ	1VVCVCVCCVCCVBCCVCCCCCCCVZ
1VVCVBVCCVCCCCCVZ	1VVCVCVCCVCCVCBCVCCCCCCCVZ
1VVCVCWCCVCCCCCVZ	1VVCVCVCCVCCVCCBVCCCCCCCVZ
1VVCVCVBCVCCCCCVZ	1VVCVCVCCVCCVCCCWCCCCCCCVZ
1VVCVCVCBVCCCCCVZ	1VVCVCVCCVCCVCCCVBCCCCCCVZ
1VVCVCVCCWCCCCCVZ	1VVCVCVCCVCCVCCCVCBCCCCCVZ
1VVCVCVCCVBCCCCVZ	1VVCVCVCCVCCVCCCVCCBCCCCVZ
1VVCVCVCCVCBCCCVZ	1VVCVCVCCVCCVCCCVCCCCBCCVZ
1VVCVCVCCVCCBCCVZ	1VVCVCVCCVCCVCCCVCCCCCBCVZ
1VVCVCVCCVCCCBCVZ	1VVCVCVCCVCCVCCCVCCCCCCBVZ
1VVCVCVCCVCCCCWZ	1VVCVCVCCVCCVCCCVCCCCCCCWZ

Tabla 2.6: Derivación completa de longitud 5^2.

$$1VWZ$$
$$1VVCVCCWZ$$
$$1VVCVCVCCVCCCCWZ$$
$$1VVCVCVCCVCCVCCCVCCCCCCWZ$$
$$1VVCVCVCCVCCVCCCVCCCVCCCCVCCCCCCCCWZ$$
$$1VVCVCVCCVCCVCCCVCCCVCCCCVCCCCVCCCCCCVCCCCCCCCCCWZ$$

$$1\sigma_2WZ{=}1VWZ$$
$$1\sigma_3WZ{=}1V(VC)VC^2WZ$$
$$1\sigma_4WZ{=}1V(VC)VC^1VC^2VC^4WZ$$
$$1\sigma_5WZ{=}1V(VC)VC^1VC^2VC^2VC^3VC^6WZ$$
$$1\sigma_6WZ{=}1V(VC)VC^1VC^2VC^2VC^3VC^3VC^4VC^8WZ$$
$$1\sigma_7WZ{=}1V(VC)VC^1VC^2VC^2VC^3VC^3VC^4VC^4VC^5VC^{10}WZ$$

$$\begin{array}{lllll} 3 & & & & \\ 3 & + & 3V & + & 3C \\ 3 & + & 5V & + & 8C \\ 3 & + & 7V & + & 15C \\ 3 & + & 9V & + & 24C \\ 3 & + & 11V & + & 35C \end{array}$$

Tabla 2.7: Derivación de longitudes n^2, $2 \leq n \leq 7$.

$\underset{\uparrow}{S}$	$()(\ \underset{\uparrow}{S}\ )$	$()(()())$
3	3	
$\underset{\uparrow}{S}\ S$	$()(\ \underset{\uparrow}{S}\ S)$	
1	1	
$()\ \underset{\uparrow}{S}$	$()(()\ \underset{\uparrow}{S}\ )$	
2	1	

(a)

$\underset{\uparrow}{S}$	$()(\ \underset{\uparrow}{A}$	$()(()()\ \underset{\uparrow}{A}$
s	$a.4$	$a.2$
$(\ \underset{\uparrow}{A}$	$()((\ \underset{\uparrow}{A}\ A$	$()(()())$
$a.2$	$a.3$	
$()\ \underset{\uparrow}{S}$	$()(()(\ \underset{\uparrow}{A}\ A$	
s	$a.2$	

(b)

Tabla 2.8: Ejemplo de una CEP generada en sendas gramáticas equivalentes. (Se indica con una flecha sobre cuál símbolo variable se aplica la regla indicada por el número de abajo.)

La relación de *equivalencia* es, efectivamente, una relación de equivalencia en la clase de gramáticas con un alfabeto terminal fijo.

La relación de subsunción es en efecto una relación de orden en el cociente de las gramáticas con un mismo conjunto de símbolos terminales partido por la relación de equivalencia.

Ejemplos

Palíndromas

Sea $G_1 = (\{s_0\}, \{a, b\}, P_1, s_0)$ la gramática con producciones $s_0 \to nil|as_0a|bs_0b$

Observamos que con la aplicación de cada regla,

> se añade un par de símbolos, aquí iguales, o bien no se añade símbolo alguno, y el símbolo que se añade al inicio se ha de "equilibrar" con otro igual al final.

Así pues, es muy fácil ver que el lenguaje generado por esta gramática consta de los palíndromas de longitud par sobre $(a + b)$,

$$L(G_1) = \{\sigma \in (a+b)^* | \text{long}(\sigma) \equiv 0 \bmod 2 \ \&\ \text{reverso}(\sigma) = \sigma\}.$$

Sea G_2 la gramática que coincide con G_1 salvo en que contiene además las producciones $s_0 \to a|b$. Entonces G_2 genera a todos los palíndromas en $(a + b)^*$. G_2 subsume a la gramática G_1.

Cadenas equilibradas de paréntesis

Recordamos que las cadenas equilibradas de paréntesis (CEP) quedan caracterizadas por las siguientes proposiciones:

a) () es una CEP.

b) Si E, F son dos CEP's entonces tanto (E) como EF son CEP's.

c) Las CEP's son únicamente las cadenas obtenidas por las reglas anteriores.

Así pues consideremos la gramática $G_1 = (\{S\}, \{(,)\}, P_1, S)$, donde P_1 consta de las reglas

$$S \to ()|(S)|SS.$$

De acuerdo con las reglas descritas arriba, G_1 genera a las CEP's. Por ejemplo, una derivación de la cadena $()(()())$ se muestra en la tabla (2.8-(a)).

Consideremos ahora la gramática G_2 con símbolo inicial S y producciones

$$\begin{array}{lrcl} s) & S & \to & (A \\ a) & A & \to &)|)S|)(A|(AA \end{array}$$

Esta gramática también genera a las cadenas equilibradas de paréntesis. Tan solo como un ejemplo, se muestra la derivación de la cadena anterior en la tabla (2.8-(b)).

Para ver que G_1 subsume a G_2, es decir, que toda palabra terminal generada en G_2 es generada también en G_1, basta observar que para cualquier $\sigma \in \{(,)\}^*$:

$i)$ $G_2 \vdash (S \overset{*}{\to} \sigma) \Leftrightarrow \sigma$ es una cadena de paréntesis equilibrada,
$ii)$ $G_2 \vdash (A \overset{*}{\to} \sigma) \Leftrightarrow \exists \sigma_1 : \ \sigma = [\sigma_1)]$ y σ_1 es una cadena de paréntesis equilibrada, que incluso puede ser vacía.

De hecho, estas relaciones pueden ser demostradas simultáneamente por inducción en el número de reglas aplicadas en G_2 para obtener la palabra σ.

En efecto, supongamos que la relación $ii)$ es verdadera. Entonces la $i)$ también lo es, tan solo por la aplicación de la regla s). Ahora, si σ se obtiene de A mediante una sola regla, esta regla ha de ser la $a.1$) y en este caso $ii)$ se cumple. Si σ se obtiene de $n+1$ reglas entonces la primera regla ha de ser cualquiera de $a.2), a.3), a.4)$ las cuales son congruentes con la hipótesis $ii)$.

Ahora, para ver que G_2 subsume a G_1 hay que ver que toda cadena generada en G_1 es generada también en G_2. Para esto se procede también por inducción en el número de reglas aplicadas para generar una palabra terminal σ.

Si σ se obtiene mediante una sola regla en G_1, entonces esa regla ha de ser la 1. y $\sigma = ()$. Una derivación de ella en G_2 es

$$S \overset{s}{\to} (A \overset{a.1}{\to} ()$$

Ahora, supongamos que σ se obtiene mediante la aplicación de $n+1$ reglas en G_1.

Supongamos que la primera es de la forma 2.: $S \to (S)$. Entonces $\exists \sigma_1 \in$ CEP tal que $\sigma = (\sigma_1)$ y σ_1 se deriva con a lo sumo n aplicaciones de reglas en G_1. Por inducción, tenemos que σ_1 se deriva en G_2 y por la relación $ii)$ tendremos $G_2 \vdash [A \overset{*}{\to} \sigma_1)]$, de hecho, $G_2 \vdash [A \overset{*}{\to} \sigma_1 A]$. Así pues, una derivación de σ en G_2 es

$$S \overset{s}{\to} (A \overset{*}{\to} (\sigma_1 A \overset{a.1}{\to} (\sigma_1) = \sigma$$

Si la primera regla aplicada es 3.: $S \to SS$, entonces se construye de manera análoga una derivación en G_2.

Así pues, las dos gramáticas dadas son equivalentes.

2.4 Tipos de gramáticas

En el estudio de las gramáticas formales y de sus correspondientes lenguajes aparecen varios problemas, entre los que se cuentan a los siguientes:

Problema 2.4.1 (de la Palabra) *Dada una gramática G y una palabra σ sobre su alfabeto de símbolos terminales, decidir si acaso la palabra pertenece o no al lenguaje generado por G.*

Problema 2.4.2 (de Derivación) *Dada una gramática G y una $\sigma \in L(G)$ encontrar una derivación en G de σ a partir del símbolo inicial de G.*

Problema 2.4.3 (Formalización de un lenguaje) *Dado un lenguaje L encontrar una gramática G tal que $L(G) = L$, es decir, tal que genere al lenguaje L.*

Estos problemas pueden ser muy complejos, e irresolubles, en general, si consideramos como "*decidir*" el aplicar un procedimiento efectivo, en el sentido de Church, que en un número finito de transformaciones simbólicas nos dé una respuesta positiva o negativa al problema planteado.

En toda clase de gramáticas es de suma importancia decidir la resolubilidad de los anteriores problemas y, en el caso de que sean resolubles, calcular la complejidad de los procedimientos para resolverlos.

En la definición siguiente presentamos algunas clases de gramáticas, las cuales coinciden con las ya presentadas en la tabla (4).

Una gramática $G = (V, T, P, s_0)$ se dice ser

irrestricta si $P \subset V^+ \times (V \cup T)^*$, es decir, las producciones de la gramática tienen como antecedentes palabras no-vacías de símbolos variables,

sensible al contexto con borro si las producciones de la gramática son de la forma $\sigma A\tau \to \sigma\alpha\tau$, donde $A \in V$ y $\alpha, \sigma, \tau \in (V \cup T)^*$,

sensible al contexto si las producciones de la gramática son de una cualquiera de las formas siguientes,

1. $\sigma A\tau \to \sigma\alpha\tau$, donde $A \in V$, $\sigma, \tau \in (V \cup T)^*$ y $\alpha \in (V \cup T)^+$,
2. $s_0 \to nil$,

y además s_0 no aparece como consecuente en ninguna producción,

libre de contexto si las producciones de la gramática son de la forma $A \to \alpha$, donde $A \in V$ y $\alpha \in (V \cup T)^*$,

$LR(k)$ si la gramática es libre de contexto y además satisface las restricciones enunciadas a continuación.

i) El símbolo inicial no se deriva de sí mismo mediante reglas que preserven la longitud de palabras derivadas.

ii) Para cualesquiera variables $A_1, A_2 \in V$ y cualesquiera palabras $\sigma_1, \sigma_2, \tau_1, \tau_2 \in (V+T)^*$ y $\upsilon_1, \upsilon_2 \in T^*$ rige la implicación siguiente:

$$\left.\begin{array}{rcl} S & \overset{*}{\to} & \sigma_1 A_1 \upsilon_1 \to \sigma_1\tau_1\upsilon_1 \\ S & \overset{*}{\to} & \sigma_2 A_2 \upsilon_2 \to \sigma_2\tau_2\upsilon_2 \\ \sigma_1\tau_1\upsilon_1|_{\mathrm{long}(\sigma_1\tau_1)+k} & = & \sigma_2\tau_2\upsilon_2|_{\mathrm{long}(\sigma_1\tau_1)+k} \end{array}\right\} \Rightarrow \left\{\begin{array}{rcl} A_1 & = & A_2 \\ \sigma_1 & = & \sigma_2 \\ \tau_1 & = & \tau_2 \end{array}\right.$$

donde para cada palabra σ denotamos por $\sigma|_n$ al mayor prefijo de σ con longitud a lo sumo n.

$LL(k)$ si la gramática es libre de contexto y además satisface la restricción enunciada a continuación.

Para cualquier variable $A_1 \in V$ y cualesquiera palabras $\tau_1, \tau_2, \upsilon_1, \upsilon_2 \in (V+T)^*$ y $\sigma, \phi_1, \phi_2 \in T^*$ rige la implicación siguiente:

$$\left.\begin{array}{rcl} S & \overset{*}{\to} & \sigma A\upsilon_1 \to \sigma\tau_1\upsilon_1 \overset{*}{\to} \sigma\phi_1 \\ S & \overset{*}{\to} & \sigma A\upsilon_2 \to \sigma\tau_2\upsilon_2 \overset{*}{\to} \sigma\phi_2 \\ & & \phi_1|_k = \phi_2|_k \end{array}\right\} \Rightarrow \quad \tau_1 = \tau_2.$$

Las gramáticas del tipo $LL(k)$ o $LR(k)$ se dicen ser *gramáticas libres de contexto deterministas.*

lineal si las producciones de la gramática son de la forma $A \to \alpha B\beta$ o $A \to \alpha$, donde $A, B \in V$ y $\alpha, \beta \in T^*$,

lateral si las producciones de la gramática son todas de la forma $A \to \alpha B$ o $A \to \alpha$, donde $A, B \in V$ y $\alpha, \beta \in T^*$, en cuyo caso la gramática es *derecha*, o bien son todas de la forma $A \to B\alpha$ o $A \to \alpha$, en cuyo caso la gramática es *izquierda.*

Puede verse que las clases de gramáticas así definidas forman una jerarquía en el siguiente sentido:

$$\left.\begin{array}{c} \textit{lineal derecha} \\ \Updownarrow \\ \textit{lineal izquierda} \end{array}\right\} \Rightarrow \begin{array}{ccccccc} \textit{lineal} & \Rightarrow & LR(k) & \Rightarrow & LC & \Rightarrow \\ SC & \Rightarrow & SCB & \Rightarrow & \textit{tipo 0} \end{array}$$

Cada una de las implicaciones mostradas es estricta pues sus recíprocos no son válidos según lo atestiguan contraejemplos apropiados que iremos presentando a lo largo de este curso.

Un lenguaje L se dice ser del mismo tipo que una gramática que lo genere.

Para concluir esta sección presentamos en la proposición siguiente una manera de re-escribir gramáticas que será de utilidad posteriormente.

Proposición 2.4.1 *Para toda gramática $G_1 = (V_1, T_1, P_1, s_{01})$ y exite una gramática equivalente $G_2 = (V_2, T_2, P_2, s_{02})$, con el mismo conjunto de símbolos terminales $T_2 = T_1$, tal que en la nueva gramática las producciones son de la forma $\alpha \to \beta$, donde $\alpha \in V_2^*$, y bien $\beta \in V_2^*$ o bien $\beta \in T_2$, es decir, consta de un único símbolo terminal. Además, la gramática G_2 es del mismo tipo que la gramática G_1.*

En efecto, dada $G_1 = (V_1, T_1, P_1, s_{01})$, para cada símbolo terminal $t \in T_1$, sea v_t un nuevo símbolo variable. Hagamos $V_2 = V_1 \cup \{v_t | t \in T_1\}$, $s_{02} = s_{01}$ y en el conjunto de producciones P_2 colocamos cada una de las producciones en P_1, habiendo sutituído toda aparición de caracteres $t \in T_1$ por sus respectivos $v_t \in V_2$, más las producciones de la forma $v_t \to t$.

Por un lado, G_1 está subsumida por G_2 pues toda derivación en G_1 da una derivación en G_2 y, por otro lado, de la observación (2.3.1) se sigue que G_2 está subsumida por G_1. Es claro que la gramática G_2 es del mismo tipo que G_1.

2.5 Ejercicios

1. Para la gramática G cuyas producciones son

$$S \to SS|aSb|bSa|ab|ba$$

describa al lenguaje generado por G.

2. Considere las gramáticas G_1, G_2 cuyas producciones son

$$\begin{array}{lcll} G_1 & : & S & \to & nil|A \\ & & A & \to & cA|Ad|c|d \\ G_2 & : & S & \to & nil|A \\ & & A & \to & cAd|cd \end{array}$$

   a. Derive, en cada una de las gramáticas, dos palabras de longitud 4.
   b. Calcule $L(G_1)$ y $L(G_2)$

3. Considere la gramática G cuyas producciones son

$$\begin{array}{lcl} S & \to & aB|ba \\ A & \to & a|aS|bAA \\ B & \to & b|bS|aBB \end{array}$$

y la palabra $\sigma = a^3b^2ab^3a$.
   a. Encuentre una derivación diestra de σ.
   b. Encuentre una derivación siniestra de σ.
   c. Bosqueje un procedimiento tal que dada una palabra decida si acaso está generada en la gramática y, en tal caso, encuentre una derivación diestra.

4. Construya una gramática que genere al lenguaje

$$L = \{a^ib^jc^k | (i \neq j) \vee (j \neq k)\}.$$

Explique su estrategia.

5. Construya una gramática que genere al lenguaje

$$L = \{a^ib^ja^ib^j | i, j \geq 1\}.$$

Explique su estrategia.

6. Construya una gramática que genere al lenguaje

$$L = \{\sigma \in (0+1)^* | \sigma \text{ posee el mismo número de 0's y 1's}\}.$$

Explique su estrategia.

7. Construya una gramática que genere al lenguaje

$$L = \{0^m1^n | m > n \geq 0\}.$$

Explique su estrategia.

8. Construya una gramática que genere al lenguaje

$$L = \{0^n 1^k | 1 \leq n \leq 2^k\}.$$

Explique su estrategia.

9. Construya una gramática que genere al lenguaje

$$L = \{0^n 1^m 0^k | k = n + m\}.$$

Explique su estrategia.

10. Construya una gramática que genere al lenguaje

$$L = \{1^x 0 1^y 0 1^z | z = xy,\ x, y \geq 0\}.$$

Explique su estrategia.

11. Construya una gramática libre de contexto que genere a las expresiones aritméticas de C, considerando las prioridades usuales.

12. Construya una gramática lineal izquierda que genere al lenguaje 10^*.

13. Construya una gramática lineal derecha que genere al lenguaje $ab^* + c^*$.

14. Dadas dos gramáticas G_1, G_2 construya una gramática para generar cada uno de los siguientes lenguajes: $L(G_1) \cup L(G_2), L(G_1)L(G_2), L(G_1)^*$.

Muestre que si ambas gramáticas son libres de contexto entonces también pueden serlo las gramáticas de los anteriores lenguajes.

15. Sea G la gramática con producciones

$$\left.\begin{array}{rcl} S & \to & YXY \end{array}\right\} : \text{inicio}$$

$$\left.\begin{array}{rcl} YX & \to & YZ \end{array}\right\} : Z \text{ es apuntador para duplicar,}$$

$$\left.\begin{array}{rcl} ZX & \to & XXZ \\ ZY & \to & XXY \end{array}\right\} : \text{duplica}$$

$$\left.\begin{array}{rcl} X & \to & 1 \\ Y & \to & nil \end{array}\right\} : \text{terminación}$$

y símbolo inicial S.

a) Derive a las palabras $1^4, 1^8, 1^{16}$.

b) Demuestre que el lenguaje de la gramática es $\{1^{2^n} | n \geq 1\}$.

16. Sea G la gramática con producciones

$$\left.\begin{array}{rcl} S & \to & 1 \\ S & \to & 1VUZ \end{array}\right\} : \text{inicio}$$

$$\left.\begin{array}{rcl} UZ & \to & 11 \\ V1 & \to & 11 \\ C1 & \to & 11 \end{array}\right\} : \text{terminación}$$

$$\left.\begin{array}{rcl} UZ & \to & ACVZ \\ VT & \to & TCV \\ CT & \to & ACV \end{array}\right\} :$$

$$\left.\begin{array}{rcl} VA & \to & TC \\ CA & \to & AC \\ UC & \to & VB \\ BC & \to & CB \\ BV & \to & CU \end{array}\right\} : \text{transposiciones}$$

$$\left.\begin{array}{rcl} 1T & \to & 1VVCU \end{array}\right\} : \text{reiteración}$$

y símbolo inicial S.

a) Derive a las palabras $1^2, 1^4, 1^9, 1^{16}$.

b) Demuestre que el lenguaje de la gramática es $\{1^{n^2} | n \geq 1\}$.

Sugerencia: Recuerde que para todo $n \geq 1$, $\sum_{i=1}^{n}(2i-1) = n^2$.

17. Sea G la gramática con producciones

1.	S	$\rightarrow$	$IMaD$	*inicio*
2.	Ma	$\rightarrow$	aaM	*duplíquese cada a,*
3.	MD	$\rightarrow$	RD	*al concluir una vuelta, regrésese,*
4.	aR	$\rightarrow$	Ra	*prosígase regresando,*
5.	IR	$\rightarrow$	IM	*repítase el procedimiento,*
6.	MD	$\rightarrow$	T	*para terminar, suprímase al delimitador derecho,*
7.	aT	$\rightarrow$	Ta	*regrésese a suprimir el delimitador izquierdo,*
8.	IT	$\rightarrow$	nil	*termínese.*

y símbolo inicial S.

a. Genere al menos tres palabras en esa gramática.

b) Conjeture cuál conjunto es $L(G)$ y demuestre su conjetura.

18. Para cada $m \geq 0$ calcule el número de cadenas equilibradas de de paréntesis de longitud $4n = 2m$.

19. Construya todos los árboles de derivación de la palabra *abababa* en la gramática $S \rightarrow SbS|a$.

20. Dado un triángulo (no-degenerado) su *centro* es el punto donde se cruzan las bisectrices de cada uno de los ángulos del triángulo.

Sea Φ el procedimiento que dado un triángulo, une su centro con cada uno de sus vértices de manera que el triángulo queda dividido en tres "subtriángulos".

Las *triangulaciones* de un triángulo inicial se definen como sigue:

- El triangulo inicial es en sí una triangulación.
- Si $\mathcal{T}$ es una triangulación, al elegir un subtriángulo cualquiera de ella y aplicarle el procedimiento Φ se obtiene una nueva triangulación.

a. Muestre que un número n es impar si y sólo existe una triangulación de n subtriángulos a partir de cualquier triángulo inicial.

b. Describa una gramática formal que genere al conjunto de triangulaciones.

c. Cuente el número de triangulaciones.

d. Diremos que dos triangulaciones, de un triángulo equilátero, son *equivalentes* si una se obtiene de la otra mediante una isometría del triángulo[1]

Para una triangulación dada, calcule el número de triangulaciones que le son equivalentes.

2.6 Programas

1. Considere la regla de transformación *MAI* visto en la sección 1.2.1:

[1] Es decir, rotaciones de $\frac{\pi}{3}$ radianes, reflexiones a lo largo del pie ortogonal en un lado que pasa por el vértice opuesto y composiciones de esas transformaciones.

I.	σi	$\to$	σia
II.	$m\sigma$	$\to$	$m\sigma\sigma$
III.	$\sigma iii\tau$	$\to$	$\sigma a\tau$
IV.	$\sigma aa\tau$	$\to$	$\sigma\tau$

Los caracteres griegos representan a palabras en el alfabeto $MAI = \{m, a, i\}$. La primera regla dice que a toda palabra que termina con i puede añadírsele una a, la segunda, que toda palabra que comience con m puede repetir su "resto", la tercera, que cualquier cadena de tres i-es consecutivas puede cambiarse por una a, y, finalmante, que cualesquiera dos a-es consecutivas pueden ser suprimidas.

Si $\sigma_0 \in MAI^*$, el *árbol de derivación* de σ_0 es el árbol (tal vez infinito, pero tal que cada nodo posee un conjunto finito de hijos) $Ar_{MAI}(\sigma_0)$ cuyos nodos están etiquetados por palabras en MAI^* y cuyas aristas lo están por las producciones I, II., III. o IV. definido recursivamente como sigue:

1. La raíz de $Ar_{MAI}(\sigma_0)$ tiene como etiqueta a σ_0.
2. Para cada nodo de $Ar_{MAI}(\sigma_0)$, digamos que etiquetado con la palabra σ, si no se le puede aplicar ninguna producción a σ entonces este nodo se declara como *hoja*, en otro caso,
   (a) pongamos un *contador de hijos* h igual a 0, y apliquemos, en tanto sea posible, las reglas siguientes en el orden en que las presentamos:
   (b) si σ termina con i, $\sigma = \sigma_1 i$, hacemos $h := h + 1$ y le añadimos un h-ésimo hijo a σ etiquetándolo con $\sigma_1 ia$, y a la arista que los une con I.,
   (c) si σ comienza con m, $\sigma = m\sigma_1$, hacemos $h := h+1$ y le añadimos un h-ésimo hijo a σ etiquetándolo con $m\sigma_1\sigma_1$, y a la arista que los une con II.,
   (d) si σ contiene la cadena iii, para cada vez que aparezca esa cadena, es decir siempre que podamos escribir $\sigma = \sigma_1 iii\tau_1$, hacemos $h := h + 1$ y le añadimos un h-ésimo hijo a σ etiquetándolo con $\sigma_1 a\tau_1$, y a la arista que los une con III.,
   (e) si σ contiene la cadena aa, para cada vez que aparezca esa cadena, es decir siempre que podamos escribir $\sigma = \sigma_1 aa\tau_1$, hacemos $h := h + 1$ y le añadimos un h-ésimo hijo a σ etiquetándolo con $\sigma_1\tau_1$, y a la arista que los une con IV.

Escriba un programa que reciba una cadena $\sigma_0 \in MAI^*$ y un número natural n y enliste los n primeros nodos de $Ar_{MAI}(\sigma_0)$ en un recorrido "a lo ancho" (si no los hubiere ha de indicar que éste es el caso).

2. Considere la gramática del recuadro siguiente:

I.	S	$\to$	$0AB$	II.	$A1$	$\to$	$SB1$	IV.	$1B$	$\to$	0
				III.	$A0$	$\to$	$S0B$	V.	B	$\to$	SA
								VI.	B	$\to$	01

El *árbol de derivación* de S es el árbol (tal vez infinito, pero tal que cada nodo posee un conjunto finito de hijos) Ar_G cuyos nodos están etiquetados por palabras en $(S + A + B + 0 + 1)^*$ y cuyas aristas lo están por las producciones I, II., III., IV., V., VI. definido recursivamente como sigue:

1. La raíz de Ar_G tiene como etiqueta a S.
2. Para cada nodo de Ar_G, digamos que etiquetado con la palabra σ, si $\sigma \in (0 + 1)^*$ entonces este nodo se declara como *hoja*, en otro caso, se aplica la producción que corresponda, examinando σ por la izquierda y las producciones en el orden enlistado.

Escriba un programa que partiendo del símbolo inicial S y un número natural n, enliste los n primeros nodos de Ar_G en un recorrido "a lo ancho" (si no los hubiere ha de indicar que éste es el caso). Decida si el lenguaje de esta gramática es o no vacío.

3. Siguiendo la idea del algoritmo de Markov mostrado en la sección 1.2.3, diseñe un algoritmo de Markov para multiplicar por el entero 5 números enteros en binario. Escriba un programa que reciba como entrada

la representación en base 2 de un entero n y muestre visualmente la aplicación sobre él del algoritmo que lo "quintuplica".

4. *Enumeración con representación en octetos*: Utilice la representación en base $256 = 2^8$ de los números enteros. Cada dígito en tal representación es un *byte*. Un arreglo de k *bytes* $\mathbf{a} = [a_{k-1}, a_{k-2}, \ldots, a_1, a_0]$, $0 \leq a_i \leq 255$ representa al número

$$m_{\mathbf{a}} = a_{k-1}256^{k-1} + a_{k-2}256^{k-2} + \cdots + a_1 256^1 + a_0.$$

Imprima a cada dígito a_i como una pareja de dígitos hexadecimales: $00 = 0, \ldots, FF = 255$

Para un alfabeto Σ de m caracteres considere la enumeración $c : \Sigma^* \to \mathbb{N}$ definida en la Proposición 2.2.4.

Escriba sendos programas para los problemas siguientes:

1. Dada m y una cadena $\sigma \in \Sigma^*$, calcule $c(\sigma)$ y lo exprese representado en octetos.
2. Dada m y un número entero x representado en octetos, calcule $\sigma \in \Sigma^*$ tal que $x = c(\sigma)$.

5. Sea $A = \{a_1, \ldots, a_n\}$ un conjunto de n elementos. Para cada relación binaria $R \subset A \times A$, consideremos la matriz $M_R = (m_{ij})_{i,j \leq n} \in \{0,1\}^{n \times n}$, llamada *de adyacencia*, tal que

$$\forall i, j \leq n : \; m_{ij} = \begin{cases} 1 & \text{si } R(a_i, a_j), \\ 0 & \text{en otro caso.} \end{cases}$$

Escriba un programa que dados $n \in \mathbb{N}$ y una matriz $M \in \{0,1\}^{n \times n}$ decida (de manera efectiva y eficiente) si la relación R cuya matriz de adyacencia es M es:

1. reflexiva,
2. simétrica,
3. antisimétrica,
4. transitiva,
5. de equivalencia,
6. de orden.

De la evidencia que dé este programa, conjeture cuántas relaciones que sean a la vez reflexivas, simétricas, antisimétricas y transitivas puede haber en un conjunto de n elementos. Demuestre que su conjetura es correcta.

6. *Experimento de conteo a la Monte Carlo de relaciones de equivalencia*: Escriba un programa que dado n, genere de manera aleatoria una relación que sea a la vez reflexiva y simétrica.

Escriba un programa que dados n y k, realice k veces el experimento siguiente: Genera de manera aleatoria una relación que sea a la vez reflexiva y simétrica. Si ésta fuese también transitiva incrementa un contador. Luego de las k repeticiones, el programa ha de calcular la razón del contador entre k.

¿A cuáles valores, en función de n, esperaría usted que converjan esas razones, cuando k se incrementa arbitrariamente?

Pruebe este programa para valores de n entre 20 y 100 y de k no menor que 10^5.

7. *Experimento de conteo a la Monte Carlo de relaciones de orden*: Escriba un programa que dado n, genere de manera aleatoria una relación que sea a la vez reflexiva y antisimétrica.

Escriba un programa que dados n y k, realice k veces el experimento siguiente: Genera de manera aleatoria una relación que sea a la vez reflexiva y antisimétrica. Si ésta fuese también transitiva incrementa un contador. Luego de las k repeticiones, el programa ha de calcular la razón del contador entre k.

¿A cuáles valores, en función de n, esperaría usted que converjan esas razones, cuando k se incrementa arbitrariamente?

Pruebe este programa para valores de n entre 20 y 100 y de k no menor que 10^5.

8. Escriba un programa que dado n, genere de manera aleatoria una relación de equivalencia en un conjunto A de n elementos. Para cada tal relación R, describa al cociente A/R y calcule el índice de R.

Repita este experimento y grafique una tabla de frecuencias considerando los índices.

9. Escriba un programa que dado n, genere de manera aleatoria una relación de orden en un conjunto A de n elementos. Para cada tal relación R, dibuje a su gráfica dirigida $G_{(A,R)}$: los nodos son los elementos de A y entre dos nodos x, y, $x \neq y$, habrá una arista de x a y si xRy y no hay un tercer elemento z tal que xRz y zRy.

10. Sea Σ un alfabeto. Considere la relación de orden:

Prefijo: Una palabra es menor que otra si es un prefijo de ella. $\forall \sigma_1, \sigma_2 \in \Sigma^*$:

$$\sigma_1 \leq_{pre} \sigma_2 \;\Leftrightarrow\; \exists \tau \in \Sigma^* : \sigma_2 = \sigma_1 \tau.$$

Escriba programas para resolver cada uno de los problemas siguientes:

1. Dadas dos palabras σ_1, σ_2 decida si acaso una es menor que la otra.
2. Para un conjunto dado de palabras $S = \{\sigma_i\}_{1 \leq i \leq m}$ calcule la matriz de orden $M = (m_{ij})_{i,j \leq n}$: $m_{ij} = 1 \Leftrightarrow \sigma_i \leq \sigma_j$.

   Una *cadena* es un subconjunto ordenado linealmente, es decir, cualesquiera dos elementos de ella se comparan entre sí. La cadena es maximal si no está contenida propiamente en ninguna otra.

   En este caso encuentre todas las cadenas maximales posibles.

11. Sea Σ un alfabeto. Considere la relación de orden:

Monotonía: Una palabra es menor que otra si todos sus símbolos aparecen, en el mismo orden, en ella. $\forall \sigma_1 = s_{11} \cdots s_{1k_1}, \sigma_2 = s_{21} \cdots s_{2k_2} \in \Sigma^*$:

$$\sigma_1 \leq_{mon} \sigma_2 \;\Leftrightarrow\; \exists \phi : [1, k_1] \to [2, k_2] \text{ creciente }, \forall j \leq k_1 : s_{1j} = s_{2\phi(j)}.$$

Escriba programas para resolver cada uno de los problemas siguientes:

1. Dadas dos palabras σ_1, σ_2 decida si acaso una es menor que la otra.
2. Para un conjunto dado de palabras $S = \{\sigma_i\}_{1 \leq i \leq m}$ calcule la matriz de orden $M = (m_{ij})_{i,j \leq n}$: $m_{ij} = 1 \Leftrightarrow \sigma_i \leq \sigma_j$.

   Encuentre todas las cadenas maximales posibles.

12. Sea $A = \{a_1, \ldots, a_n\}$ un conjunto de n elementos. Para cada operación binaria $\cdot : A \times A \to A$, consideremos su *tabla de multiplicación* $M = (m_{ij})_{i,j \leq n} \in [\![1, n]\!]^{n \times n}$, tal que

$$\forall i, j \leq n : \; a_{m_{ij}} = a_i \cdot a_j$$

Escriba un programa que dados $n \in I\!N$ y una matriz $M \in [\![1, n]\!]^{n \times n}$ decida (de manera efectiva y eficiente) si la operación cuya tabla de multiplicación es M es:

1. conmutativa,
2. posee una unidad derecha,
3. posee una unidad izquierda,
4. asociativa,
5. dado que existe una unidad, decidir si cada elemento posee un inverso,
6. un monoide,

7. un grupo.

13. Consideraremos conjuntos $A = \{a_1, \ldots, a_n\}$ de n elementos con una operación binaria $\cdot : A \times A \to A$, dada por su *tabla de multiplicación* $M = (m_{ij})_{i,j \leq n} \in [\![1, n]\!]^{n \times n}$ en los que se define una relación de equivalencia mediante su matriz de adyacencia, $R \in \{0, 1\}^{n \times n}$.

Escriba un programa que dados $n \in \mathbb{N}$, una tabla de multiplicación $M \in [\![1, n]\!]^{n \times n}$ y una equivalencia $R \in \{0, 1\}^{n \times n}$ decida (de manera efectiva y eficiente) si la equivalencia es congruente con la multiplicación. En tal caso, calcule la tabla de multiplicación del cociente A/R.
Sugerencia: Ejemplifique considerando dos números primos p, q. Haga $n = p \cdot q$. Considere la operación $(i, j) \mapsto (i - 1)(j - 1) \bmod n + 1$ y la relación $iRj \Leftrightarrow i \equiv j \bmod p$. Ejemplifique también considerando la tabla de mutiplicación del grupo de permutaciones S_k, con $k!$ elementos. Considere como relación $\phi R \psi$ si y sólo si la composición $\phi \circ \psi$ es una permutación par. (Revise un libro de *Algebra Moderna*, si fuera necesario, para aclarar la terminología que uso.)

14. *Producto de monoides*: Dados dos monoides $A_1 = (A_1, \cdot_1, u_1)$ y $A_2 = (A_2, \cdot_2, u_2)$ el monomio *producto* de ellos dos es $A_3 = (A_3, \cdot_3, u_3)$, donde $A_3 = A_1 \times A_2$, la operación está definida "componente-a-componente"

$$\cdot_3 : ((x_{11}, x_{12}) \, , \, (x_{21}, x_{22})) \mapsto (x_{11}, x_{12}) \cdot_3 (x_{21}, x_{22}) = (x_{11} \cdot_1 x_{21}, x_{12} \cdot_2 x_{22})$$

y $u_3 = (u_1, u_2)$. Si n_1 es el número de elementos en A_1 y n_2 el de A_2 entonces su producto tendrá $n_3 = n_1 n_2$ elementos. Enumeremos a los elementos de A_3 como $(a_{k3})_{k=1}^{n_3}$, donde $\forall i, j : \; a_{\phi(i,j)\ 3} = (a_{i1}, a_{j2})$, y $\phi : (i, j) \mapsto (i - 1)n_2 + j$.

Escriba un programa que dados dos monoides A_1, A_2 con sendas tablas de multiplicación $M_1 \in [\![1, n_1]\!]^{n_1 \times n_1}$, $M_2 \in [\![1, n_2]\!]^{n_2 \times n_2}$, escriba la tabla de multiplicación del producto A_3, de acuerdo con la enumeración ϕ.

15. *Homomorfismo de monoides*: Escriba un programa que dados dos monoides A_1, A_2 con sendas tablas de multiplicación $M_1 \in [\![1, n_1]\!]^{n_1 \times n_1}$, $M_2 \in [\![1, n_2]\!]^{n_2 \times n_2}$, decida si existe o no un homomorfismo $\phi : A_1 \to A_2$. En caso de que exista, localice un homomorfismo, y de hecho a todos los posibles homomorfismos.

16. Sea Z_m el anillo de enteros módulo m. Sea $Mat_{n \times n}(Z_m)$ el conjunto de matrices $n \times n$ con entradas en Z_m. Con la multiplicación de matrices, $Mat_{n \times n}(Z_m)$ forma un monoide. Si $A, B \in Mat_{n \times n}(Z_m)$ son dos matrices, cualquier palabra $\sigma \in (A + B)^*$ determina una matriz en $Mat_{n \times n}(Z_m)$, a saber, la que se obtiene al interpretar la conjunción de dos símbolos como el producto de matrices. Introduzcamos la relación de equivalencia en $(A + B)^*$ siguiente:

$$\sigma \equiv \tau \quad \Leftrightarrow \quad \sigma = \tau \text{ vistas como matrices en } Mat_{n \times n}(Z_m).$$

El cociente $(A + B)^*/\equiv$ resulta ser un submonoide de $Mat_{n \times n}(Z_m)$.

Escriba un programa que dados m y n y dos matrices $A, B \in Mat_{n \times n}(Z_m)$ calcule a los elementos en $(A + B)^*/\equiv$ y a su tabla de multiplicación.

17. Dada una palabra $\sigma \in \Sigma$, un *período* es un prefijo τ de σ tal que σ es un prefijo de alguna potencia de τ, es decir, $\sigma \leq_{pre} \tau^k$ para alguna k.

Escriba un programa que dada una palabra σ calcule todos sus períodos y escriba una lista de las longitudes de los períodos.

18. Una palabra $\sigma \in \Sigma$ se dice ser *libre-de-cuadrados* si no contiene ningún enfijo de la forma $\tau\tau$, con $\tau \neq nil$.

Escriba un programa que dados n y m escriba una lista de todas las palabras de longitud a lo sumo n que están libres de cuadrados sobre el alfabeto de m símbolos $\{a, b, c, ..., m\text{-ésima letra}\}$.

19. Sea $A_1 = (a + b)$ el alfabeto consistente de dos símbolos, y sea $A = A_1 \cap A_1^2 \cap A_1^3$ el lenguaje consistente de las 14 palabras de longitud a lo más 3 sobre A_1.

1. Escriba un programa que dados n y un subconjunto no vacío $X \subset A$ enliste al conjunto X_n conformado por las palabras de longitud a lo más n formadas al concatenar partículas en X.

2. Para cada n defina la relación de equivalencia: $X \equiv_n Y \Leftrightarrow X_n = Y_n$.

   Escriba un programa que dado n escriba la matriz de adyacencia de $\equiv_n$.

20. Dos palabras σ, τ se dicen *conjugadas* si existe una tercer palabra υ tal que $\sigma\upsilon = \upsilon\tau$. Esta relación es de equivalencia.

Escriba un programa que dados m y n dé las clases de equivalencia de la relación de conjugación restringida a las palabras de longitud n sobre un alfabeto de m símbolos.

Conjeture cuántas clases ha de haber, en términos de m y n.

21. Escriba un programa que dados m, y dos palabras σ y τ, sobre un alfabeto de m símbolos, donde σ antecede a τ en el orden lexicográfico, calcule cuántas palabras están entre σ y τ (contando a τ pero excluyendo a σ, según ese orden).

22. *Derivación guiada en gramáticas formales*: Escriba un programa que dada una gramática formal $G = (V, T, P, s_0)$ le permita a un usuario derivar de manera interactiva palabras, seleccionando producciones a aplicarse en partículas de la palabra actual.

En otras palabras, divida la pantalla en dos sectores, en cada uno de los cuales ha de poder hacerse "scrolling". En el primero se muestra las producciones enumeradas. En el segundo se deriva palabras. La *palabra actual* es la última línea de este segundo sector. El usuario ha de "remarcar" una partícula en la palabra actual, ha de indicar cuál producción quiere aplicar y el programa aplicará esa producción dejando el resultado como palabra actual.

23. *Problema de la palabra en gramáticas libres de contexto*: Escriba un programa que resuelva el *Problema de la palabra* en gramáticas libres de contexto, buscando derivaciones siniestras ("leftmost").

Para una gramática libre de contexto $G = (V, T, P, s_0)$ y una palabra dada σ decida si acaso $\sigma \in L(G)$.

Si $\sigma' = \tau A \tau_1$ es una palabra que posee un prefijo común con σ, τ, consistente de símbolos terminales y A es un símbolo variable, entonces vea cuáles producciones con antecedente A tienen consecuentes que "empaten" con σ. Opte por las primeras. Proceda de esta forma hasta reconocer a σ. Si fallara entonces realice "backtracking" considerando otras producciones que empaten con σ.

24. Diseñe una gramática que genere al lenguaje

$$L_{PotenciasDeDos} = \{1^{2^n} | n \geq 0\}.$$

Escriba un programa que dado n muestre visualmente la manera en la que hay que aplicar las reglas para generar 1^{2^n}.

25. Diseñe una gramática que genere al lenguaje

$$L_{Cuadrados} = \{1^{n^2} | n \geq 2\}.$$

Escriba un programa que dado n muestre visualmente la manera en la que hay que aplicar las reglas para generar 1^{n^2}.
Sugerencia: Observe que para todo $n \geq 0$: $(n+1)^2 = n^2 + 2n + 1$.

26. Diseñe una gramática que genere al lenguaje

$$L_{ijij} = \{a^i b^j a^i b^j | i, j \geq 1\}.$$

Escriba un programa que dados i, j muestre visualmente la manera en la que hay que aplicar las reglas para generar $a^i b^j a^i b^j$.

27. Considere a la gramática con producciones

$$S \to DAI|a \;\; ; \;\; D \to S|d \;\; ; \;\; A \to S|a \;\; ; \;\; I \to S|i$$

Escriba un programa que dado $n \geq 0$ genere una palabra terminal de longitud $2n + 1$.

Escriba un programa que dado $n \geq 0$ cuente el número total de derivaciones posibles para arribar a una palabra de longitud $2n + 1$.

28. *Cuenta de palabras en gramáticas libres de contexto*: Escriba un programa que para una gramática libre de contexto $G = (V, T, P, s_0)$ y para un número n cuente cuántas palabras de longitud n se derivan en G.

Capítulo 3

Autómatas finitos y expresiones regulares

3.1 Máquinas secuenciales

3.1.1 Máquinas de Mealy

Una *máquina de Mealy* es una estructura de la forma

$$MMe = (Q, Ent, Sal, tran, res, q_0)$$

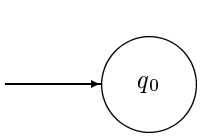
donde

$$\begin{array}{rcl} Q & : & \text{es el conjunto de } \textit{estados}, \\ Ent & : & \text{es el alfabeto de } \textit{entrada}, \\ Sal & : & \text{es el alfabeto de } \textit{salida}, \\ tran & : & Q \times Ent \to Q, \quad \text{es la función de } \textit{transición}, \\ res & : & Q \times Ent \to Sal, \quad \text{es la función de } \textit{respuesta}, \text{ y} \\ q_0 \in Q & : & \text{es el estado } \textit{inicial}. \end{array}$$

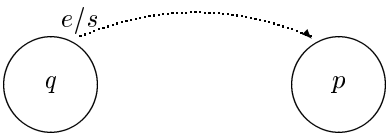
La *semántica procedimental* de la máquina de Mealy es la siguiente:

> Al inicio de cualquier computación, la máquina se encuentra en el estado q_0. Posteriormente, cuando la máquina se encuentra en un estado $q \in Q$, y recibe una literal de entrada $e \in Ent$, entonces emite el símbolo de salida $s = res(q, e)$ y transita al nuevo estado $p = tran(q, e)$.

Gráficamente, representamos esto de la siguiente manera:



q_0 es el estado *inicial*.

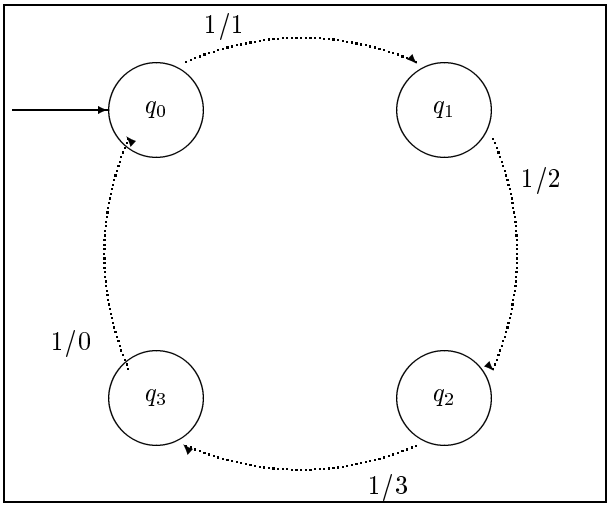


Si se está en q y llega e entonces se emite $s = res(q, e)$ y se transita a $p = tran(q, e)$.

Ejemplos

1. **Residuos módulo 4**: Si $n \in \mathbb{N}$ entonces $\overline{n}^1 = 1^{(n)}$ es la *representación unaria* de n.

Presentaremos una máquina que calcula el residuo módulo 4, de una cadena de 1's, cuando se ve a esa cadena como la representación unaria de un número no-negativo.



tran	1
q_0	q_1
q_1	q_2
q_2	q_3
q_3	q_0

(a)

res	1
q_0	1
q_1	2
q_2	3
q_3	0

(b)

Figura 3.1: Máquina de Mealy para el cálculo de residuos módulo 4 en representación unaria.

Representamos gráficamente a la máquina en la figura (3.1-a).

Esta máquina es $M_{\mathrm{mod}\ 4} = (\{q_0, q_1, q_2, q_3\}, \{1\}, \{0, 1, 2, 3\}, tran, res, q_0)$ donde las funciones *tran* y *res* están dadas como sendas tablas en la figura (3.1-b).

Aquí se puede confundir el conjunto de estados con el alfabeto de salida de manera muy natural: el i-ésimo estado es un i-ésimo símbolo de salida.

2. *Repetición final de un mismo símbolo*: Construyamos una máquina de Mealy que reconozca a las palabras en $(0+1)$ que terminan con la repetición de un mismo símbolo. Es decir, que reconozca a palabras en el alfabeto $L = (0+1)^*(00+11)$. Gráficamente, presentamos a la máquina en la figura (3.2).

La interpretación de cada estado es natural:

q_0 : estado inicial,

p_0 : estado de "*haber llegado un* 0",

p_1 : estado de "*haber llegado un* 1".

Se tiene una respuesta afirmativa cuándo se permanece en un mismo estado.

Las componentes de la máquina son pues $Q = \{q_0, p_0, p_1\}$, $Ent = \{0, 1\}$, $Sal = \{n, s\}$ y

tran	0	1
q_0	p_0	p_1
p_0	p_0	p_1
p_1	p_0	p_1

res	0	1
q_0	n	n
p_0	s	n
p_1	n	s

3. **Máquina expendedora de golosinas**: Consideremos una máquina expendedora de golosinas, de \$4 pesos cada una, que recibe monedas de \$1, \$2, \$5 y \$10 pesos. Supongamos que la máquina funciona bajo los siguientes supuestos:

- el costo de las golosinas puede cubrirse con cualquier combinación de monedas aceptables,
- la máquina sólo da cambio en monedas de \$1 peso, las cuales están almacenadas en una alcancía. Si no puede dar cambio, es decir, si el contenido de la alcancía no es suficiente, regresa la moneda insertada, y

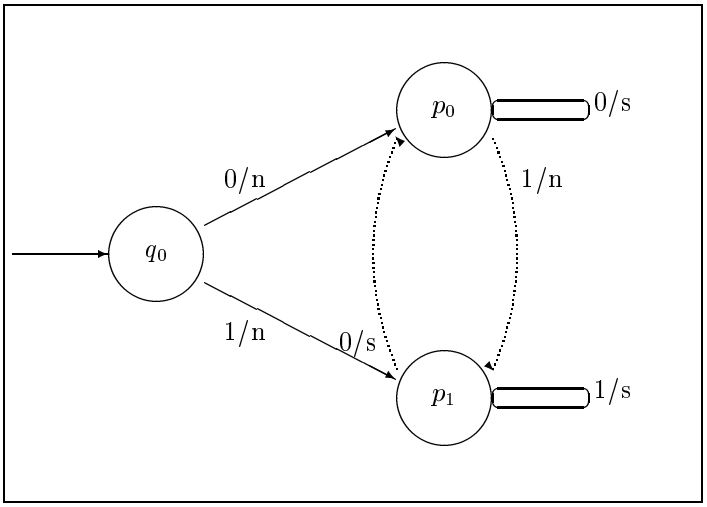


Figura 3.2: Máquina de Mealy para reconocer palabras que terminan con un símbolo repetido.

- sólo se puede insertar monedas en orden inverso a su denominación.

Codifiquemos el funcionamiento de la máquina con los conjuntos siguientes:

- *Monedas a insertarse*:

$$\begin{aligned} m_0 &: \text{ninguna moneda se inserta,}\\ m_1 &: \text{moneda de un peso,}\\ m_2 &: \text{moneda de dos pesos,}\\ m_5 &: \text{moneda de cinco pesos,}\\ m_{10} &: \text{moneda de diez pesos.} \end{aligned}$$

- *Respuestas de la máquina*:

$$\begin{aligned} s_0 &: \text{continúa sin más,}\\ s_1 &: \text{entrega una golosina,}\\ s_2 &: \text{da un peso de cambio,}\\ s_3 &: \text{devuelve la moneda insertada.} \end{aligned}$$

- *Estados de la máquina*:

$$\begin{aligned} q_0 &: \text{estado inicial,}\\ \forall i \in [0,5] : a_i &: \text{resta por devolver } i \text{ pesos,}\\ \forall j \in [1,2] : b_i &: \text{falta por pagar } j \text{ pesos cuando se inició el pago con \$2,}\\ \forall k \in [1,3] : c_i &: \text{falta por pagar } k \text{ pesos cuando se inició el pago con \$1.} \end{aligned}$$

- *Depósito en la alcancía*:

$$\begin{aligned} \forall i \in [1,6] : p_i &: \text{NO alcanza a haber } i \text{ pesos,}\\ p_7 &: \text{al menos hay \$6 pesos.} \end{aligned}$$

La máquina de Mealy que modela el funcionamiento de la máquina expendedora tiene como alfabeto de entrada el producto cartesiano del conjunto de monedas aceptables con el conjunto que codifica a los depósitos de la alcancía. Hay pues $5 \times 7 = 35$ símbolos de entrada $m_\mu p_\nu$. El alfabeto de salida está dado por las 4 posibles respuestas que da la máquina expendedora. Hay $1 + 6 + 2 + 3 = 12$ estados.

A grandes rasgos las transiciones se definen como se muestra en las tablas (3.1) y (3.2).

$\forall j \leq 6$: $tran(q_0, m_{10}p_j) = q_0$ $res(q_0, m_{10}p_j) = s_3$	si se inserta una moneda de \$10 pesos y no hay cambio suficiente, se devuelve la moneda y se reinicia el proceso,
$tran(q_0, m_{10}p_7) = a_5$ $res(q_0, m_{10}p_7) = s_2$	ya que lo hay, procédase a dar cambio,
$\forall k = 5,4,3,2,1$: $tran(a_k, m_0P) = a_{k-1}$ $res(a_k, m_0P) = s_2$	para $P = p_j$, cualquiera que sea j, continúese devolviendo un peso hasta completar el cambio. Obsérvese que aquí, en principio, puede haber combinaciones (a_k, p_j) contradictorias. Sin embargo, la interpretación que se está construyendo excluye que aparezcan esas inconsistencias.
$tran(a_0, m_0P) = q_0$ $res(a_0, m_0P) = s_1$	al terminar de dar el cambio, se entrega la golosina y se reinicia el proceso.

Tabla 3.1: Transiciones y repuestas de la máquina expendedora.

$tran(q_0, m_5p_1) = q_0$ $res(q_0, m_5p_1) = s_3$	si se inserta una moneda de \$5 pesos y no hay cambio, se devuelve la moneda y se reinicia el proceso,
$tran(q_0, m_5P) = a_0$ $res(q_0, m_5P) = s_2$	si hay monedas en la alcancía, i.e. $P \neq p_1$, entonces se da el peso de cambio,
$tran(q_0, m_2P) = b_2$ $res(q_0, m_2P) = s_0$	se insertan \$2 pesos y se espera a completar el importe de \$4 pesos,
$tran(b_2, m_2P) = q_0$ $res(b_2, m_2P) = s_1$	habiéndose completado el costo de la golosina, se lo entrega y se reinicia el proceso,
$tran(b_2, m_1P) = c_1$ $res(b_2, m_1P) = s_0$	se inserta un peso más y hay que esperar a que llegue el último,
$tran(b_2, MP) = b_2$ $res(b_2, MP) = s_3$	si llega una moneda con denominación mayor $M = m_5, m_{10}$ entonces se la devuelve y se continúa la espera,
$tran(q_0, m_1P) = c_3$ $res(q_0, m_1P) = s_0$	si se inicia el pago con una moneda de un peso hya que esperar los otros tres pesos,
$\forall k = 3,2,1$: $tran(c_k, m_1P) = c_{k-1}$ $res(c_k, m_1P) = s_0$	se continúa el pago, recibiendo un peso a la vez. Aquí $c_0 = a_0$. Si se recibe monedas de mayor denominación, se develve éstas.
	cualquier otra posibilidad (Estado,Entrada) es inconsistente e inalcanzable en la máquina.

Tabla 3.2: Transiciones y repuestas de la máquina expendedora (cont).

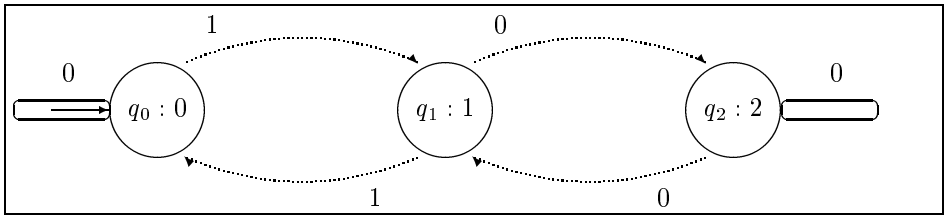


Figura 3.3: Máquina de Moore para calcular congruencias módulo 3 de números dados en binario.

3.1.2 Máquinas de Moore

Una *máquina de Moore* es similar a una de Mealy, salvo en que la respuesta sólo depende del estado actual de la máquina y es independiente de la entrada. Precisamente, una máquina de Moore es una estructura de la forma

$$MMo = (Q, Ent, Sal, tran, res, q_0)$$

donde

$$\begin{array}{rcl} Q & : & \text{es el conjunto de } \textit{estados}, \\ Ent & : & \text{es el alfabeto de } \textit{entrada}, \\ Sal & : & \text{es el alfabeto de } \textit{salida}, \\ tran & : & Q \times Ent \to Q, \quad \text{es la función de } \textit{transición}, \\ res & : & Q \to Sal, \quad \text{es la función de } \textit{respuesta}, \\ q_0 \in Q & : & \text{es el estado } \textit{inicial}. \end{array}$$

La *semántica procedimental* de la máquina de Moore es la siguiente:

> Al inicio de cualquier computación, la máquina se encuentra en el estado q_0. Posteriormente, cuando la máquina se encuentra en un estado $q \in Q$, y recibe una literal de entrada $e \in Ent$, entonces transita al nuevo estado $p = tran(q, e)$ y emite el símbolo de salida $s = res(p)$.

Ejemplos

1. **Congruencias módulo 3:** Supongamos que se da un número $n \in I\!N$ en su representación binaria y se quiere calcular su residuo módulo 3.

Consideremos la máquina cuya representación gráfica se muestra en la figura (3.3).

Las funciones de transición y de respuesta quedan especificadas de manera tabular como sigue:

$tran$	0	1
q_0	q_0	q_1
q_1	q_2	q_0
q_2	q_1	q_2

res	
q_0	0
q_1	1
q_2	2

Por inducción en la longitud n de cualquier palabra $\sigma \in (0+1)^*$, que sea la representación en binario de un número x_σ se puede ver que la respuesta final obtenida al aplicar σ es x_σ mod 3.

En efecto, para $n = 1$, con las palabras '0' y '1' se tiene las respuestas correctas 0 y 1. Sea $n > 0$. Supongamos que para una palabra σ, de longitud $n-1$, se tiene como respuesta final i, donde $x \equiv i \bmod 3$ y x es el número representado en binario por σ. Para $s \in (0+1)$ el número representado por la concatenación de σ con s, σs es $2x + s$, el cual es congruente módulo 3 con $2i + s$ mod 3. Al tabular estos últimos valores se tiene

$i \backslash s$	0	1
0	0	1
1	2	0
2	1	2

$MMo^{10}_{\mathrm{mod}\ 5}$:

tran	0	1	2	3	4	5	6	7	8	9
q_0	q_0	q_1	q_2	q_3	q_4	q_0	q_1	q_2	q_3	q_4
q_1	q_0	q_1	q_2	q_3	q_4	q_0	q_1	q_2	q_3	q_4
q_2	q_0	q_1	q_2	q_3	q_4	q_0	q_1	q_2	q_3	q_4
q_3	q_0	q_1	q_2	q_3	q_4	q_0	q_1	q_2	q_3	q_4
q_4	q_0	q_1	q_2	q_3	q_4	q_0	q_1	q_2	q_3	q_4

$MMo^{10}_{\mathrm{mod}\ 7}$:

tran	0	1	2	3	4	5	6	7	8	9
q_0	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_0	q_1	q_2
q_1	q_3	q_4	q_5	q_6	q_0	q_1	q_2	q_3	q_4	q_5
q_2	q_6	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_0	q_1
q_3	q_2	q_3	q_4	q_5	q_6	q_0	q_1	q_2	q_3	q_4
q_4	q_5	q_6	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_0
q_5	q_1	q_2	q_3	q_4	q_5	q_6	q_0	q_1	q_2	q_3
q_6	q_4	q_5	q_6	q_0	q_1	q_2	q_3	q_4	q_5	q_6

$MMo^{10}_{\mathrm{mod}\ 13}$:

tran	0	1	2	3	4	5	6	7	8	9
q_0	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9
q_1	q_A	q_B	q_C	q_0	q_1	q_2	q_3	q_4	q_5	q_6
q_2	q_7	q_8	q_9	q_A	q_B	q_C	q_0	q_1	q_2	q_3
q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_A	q_B	q_C	q_0
q_4	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_A
q_5	q_B	q_C	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7
q_6	q_8	q_9	q_A	q_B	q_C	q_0	q_1	q_2	q_3	q_4
q_7	q_5	q_6	q_7	q_8	q_9	q_A	q_B	q_C	q_0	q_1
q_8	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_A	q_B
q_9	q_C	q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8
q_A	q_9	q_A	q_B	q_C	q_0	q_1	q_2	q_3	q_4	q_5
q_B	q_6	q_7	q_8	q_9	q_A	q_B	q_C	q_0	q_1	q_2
q_C	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_A	q_B	q_C

Tabla 3.3: Cálculo de residuos módulo 5, 7 y 13 en notación decimal.

lo que corresponde naturalmente a la tabla de transiciones del autómata construído.

De hecho, éste es un caso particular del siguiente ejemplo más general: Sea $n > 1$ una base de representación de números naturales y sea $k > 0$ un número natural. Sea $MMo^{n}_{\mathrm{mod}\ k}$ la máquina de Moore tal que

- posee n símbolos de entrada $\{0, \ldots, n-1\}$,
- posee k estados $\{q_0, \ldots, q_{k-1}\}$, y k símbolos de salida, uno por cada estado.
- tiene como transición a la función $(q_i, s) \mapsto q_{(i \cdot n) \bmod k}$, y
- tiene como respuesta $q_i \mapsto i$.

Entonces $MMo^{n}_{\mathrm{mod}\ k}$ calcula el residuo módulo k de cualquier número en base n. En la tabla (3.3) presentamos las tablas de transición de las máquinas $MMo^{10}_{\mathrm{mod}\ k}$, para $k = 5, 7, 13$. El lector no ha de tener dificultad en visualizar, a partir de esos ejemplos, las transiciones de cualquier máquina $MMo^{n}_{\mathrm{mod}\ k}$.

2. **Problema de botes:** Supongamos dados $k > 1$ botes. Para cada $i \leq k$, sea $c_i \in I\!N$ la capacidad, en litros, del i-ésimo bote. Los botes pueden ser llenados de agua o bien ser vaciados de acuerdo con las siguientes reglas:

L_i : llénese el i-ésimo bote,
V_i : vacíese el i-ésimo bote,
$M_{i_1 i_2}$: viértase el contenido del i_1-ésimo bote en el i_2-ésimo hasta que aquel se vacíe o éste se llene.

Si se considera a los dos primeros botes como distinguidos, se trata de caracterizar a las cantidades de agua "constructibles" como suma de los contenidos de esos dos primeros botes.

Sean pues Estados : $Q = \{\mathbf{x} = (x_1, \ldots, x_k) | \forall i \leq k : 0 \leq x_i \leq c_i\}$
Entradas : $Ent = \{L_i, V_i\}_{i \leq k} \cup \{M_{i_1 i_2}\}_{i_1 i_2 \leq k; i_1 \neq i_2}$
Salidas : $Sal = [0, c_1 + c_2]$

Las transiciones quedan caracterizadas de la siguiente forma:

$$\forall i \leq n \quad : \quad tran(\mathbf{x}, L_i) = \mathbf{y} \text{ donde } \forall j : \; y_j = \begin{cases} c_i & \text{si } j = i, \\ x_j & \text{en otro caso.} \end{cases}$$

$$\forall i \leq n \quad : \quad tran(\mathbf{x}, V_i) = \mathbf{y} \text{ donde } \forall j : \; y_j = \begin{cases} 0 & \text{si } j = i, \\ x_j & \text{en otro caso.} \end{cases}$$

$$\forall i_1, i_2 \leq n \quad : \quad tran(\mathbf{x}, M_{i_1 i_2}) = \mathbf{y} \text{ donde } \forall j : \begin{cases} j \neq i_1, i_2 & \Rightarrow \quad y_j = x_j \\ j = i_1 & \Rightarrow \quad y_{i_1} = \begin{cases} 0 & \text{si } x_{i_1} \leq c_{i_2} - x_{i_2}, \\ x_{i_1} + x_{i_2} - c_{i_2} & \text{en otro caso.} \end{cases} \\ j = i_2 & \Rightarrow \quad y_{i_2} = \begin{cases} x_{i_1} + x_{i_2} & \text{si } x_{i_1} \leq c_{i_2} - x_{i_2}, \\ c_{i_2} & \text{en otro caso.} \end{cases} \end{cases}$$

La respuesta es la función $res : \mathbf{x} \mapsto x_1 + x_2$.

3.1.3 Equivalencia e indistinguibilidad

Sea $M = (Q, Ent, Sal, tran, res, q_0)$ una máquina, ya sea de Mealy o de Moore. Extendemos la función de transición $tran : Q \times Ent \to Q$ a una función $tran^* : Q \times Ent^* \to Q$, haciendo, para cada estado $q \in Q$:

$$\begin{aligned} tran^*(q, nil) &= q, \\ \forall \sigma \in Ent^*, s \in Ent : \; tran^*(q, \sigma s) &= tran(tran^*(q, \sigma), s) \end{aligned}$$

Así pues, para cada palabra σ, $tran^*(q, \sigma)$ es el estado al que se llega cuando, a partir del estado q, se va aplicando, uno a uno, cada uno de los símbolos de σ, de izquierda a derecha.

De manera similar se puede extender la función de respuesta a todo el diccionario Ent^*.

Si M es una máquina de Mealy, definimos $res^* : Q \times Ent^* \to Sal^*$, haciendo, para cada estado $q \in Q$ y para cada palabra $\sigma \in Ent^*$, $res^*(q, \sigma) = \tau \in Sal^*$ donde,

$$\begin{aligned} \sigma = nil \quad &\Rightarrow \quad \tau = nil, \\ \sigma = s_1 \cdots s_k \quad &\Rightarrow \quad \tau = t_1 \cdots t_k \text{ con } \begin{array}{llllll} t_1 &= res(q, s_1) & , q_1 &= tran(q, s_1) \\ t_2 &= res(q_1, s_2) & , q_2 &= tran(q_1, s_2) \\ \vdots & \vdots & , \; \vdots & \vdots \\ t_k &= res(q_{k-1}, s_k) & , q_k &= tran(q_{k-1}, s_k) \end{array} \end{aligned}$$

en otras palabras, se tiene

$$\begin{aligned} res^*(q, nil) &= nil, \\ \forall \sigma \in Ent^*, s \in Ent : \; res^*(q, \sigma s) &= res^*(q, \sigma) res(tran^*(q, \sigma), s) \end{aligned}$$

Si M es una máquina de Moore, la función de respuesta $res^* : Q \times Ent^* \to Sal^*$ depende únicamente del estado visitado: para cada estado $q \in Q$

$$\begin{aligned} res^*(q, nil) &= nil, \\ \forall \sigma \in Ent^*, s \in Ent : \; res^*(q, \sigma s) &= res^*(q, \sigma) res(trans^*(q, \sigma)) \end{aligned}$$

En cualquier caso, sea en máquinas de Mealy o de Moore, la función $trad : \sigma \mapsto res^*(q_0, \sigma)$, donde q_0 es el estado inicial, es la función de *traducción* que realiza la máquina. Por las semánticas procedimentales introducidas, se tiene que $\forall\sigma$: $\log(trad(\sigma)) = \log(\sigma)$.

Dos máquinas M y N se dicen ser *equivalentes*, $M \equiv N$, si $trad_M = trad_N$. En otras palabras, dos máquinas son equivalentes si ambas traducen de idéntica manera a cualquier palabra de entrada.

Ya que las máquinas de Moore son casos particulares de las máquinas de Mealy, se tiene que toda máquina de Moore es equivalente a una de Mealy. Veamos que el recíproco también se cumple:

Proposición 3.1.1 *Toda máquina de Mealy es equivalente a una de Moore: Para cada máquina de Mealy* $MMe = (Q, Ent, Sal, tran, res, q_0)$ *existe una máquina de Moore* $MMo = (Q', Ent, Sal, tran', res', q'_0)$ *tal que* $MMe \equiv MMo$.

En efecto, dada una máquina de Mealy $MMe = (Q, Ent, Sal, tran, res, q_0)$, realicemos la siguiente construcción:

estados: sea $Q' = Q \times Sal \cup \{(q_0, nil)\}$. Se "desdobla" cada estado "viejo" $q \in Q$ en $\mathrm{card}(Sal)$ estados "nuevos" de la forma (q, t), $t \in Sal$;

transición: sea $tran' : ((q, t), e) \mapsto (tran(t, e), res(t, e))$, donde $tran$ y res son las funciones de transición y de respuesta "viejas";

respuesta: sea $res' : (q, t) \mapsto t$; y

estado inicial: sea $q'_0 = (q_0, nil)$.

Se ve directamente que la máquina de Moore construida es equivalente a la de Mealy dada.

Ejemplo

Consideremos la máquina de Mealy del ejemplo 2. anterior que "reconoce a repeticiones finales de un mismo símbolo en $\{0 + 1\}$". Ahí, la máquina tiene transición y respuesta,

$trans$	0	1
q_0	p_0	p_1
p_0	p_0	p_1
p_1	p_0	p_1

res	0	1
q_0	n	n
p_0	s	n
p_1	n	s

La máquina de Moore equivalente consiste de $7 = 1 + 6$ estados

$$q_0 nil, q_0 n, p_0 n, p_1 n, q_0 s, p_0 s, p_1 s$$

y sus correspondientes transición y respuesta son

$trans'$	0	1
$q_0 nil$	$p_0 n$	$p_1 n$
$q_0 n$	$p_0 n$	$p_1 n$
$p_0 n$	$p_0 s$	$p_1 n$
$p_1 n$	$p_0 n$	$p_1 s$
$q_0 s$	$p_0 n$	$p_1 n$
$p_0 s$	$p_0 s$	$p_1 n$
$p_1 s$	$p_0 n$	$p_1 s$

res'	
$q_0 nil$	nil
$q_0 n$	n
$p_0 n$	n
$p_1 n$	n
$q_0 s$	s
$p_0 s$	s
$p_1 s$	s

Observamos aquí que los estados $q_0 nil, q_0 n, q_0 s$ no aparecen en la imagen de la función de transición nueva. Por tanto, los restantes cuatro estados, junto con el inicial, definen una máquina de Moore de 5 estados equivalente a la máquina de Mealy dada.

En lo que resta de esta sección, consideraremos únicamente máquinas de Moore.

Sea $MMo = (Q, Ent, Sal, tran, res, q_0)$ una máquina de Moore. Se dice que MMo es una *máquina-*(n, m, k) si $n = \mathrm{card}(Q)$ es el número de estados, $m = \mathrm{card}(Ent)$ es el número de símbolos de entrada y $k = \mathrm{card}(Sal)$ es el número de símbolos de salida, que son efectivamente asumidos bajo la función de respuesta *res*.

Sea $f_{MMo} = res \circ tran^*$ la función que, para un estado q y una palabra σ, da el último símbolo de respuesta cuando se aplica σ a partir de q.

Diremos que dos estados q_1, q_2 son *indistinguibles*, $q_1 \sim_{Ind} q_2$, si para cualquier palabra $\sigma \in Ent^*$ se tiene $f(q_1, \sigma) = f(q_2, \sigma)$. Intuitivamente, dos estados son indistinguibles si no se los puede distinguir mediante una sucesión de estímulos, pues ambos estados ofrecen mismas respuestas ante mismas entradas. Los estados son *distinguibles* si para alguna palabra σ se tiene $f(q_1, \sigma) \neq f(q_2, \sigma)$, y en tal caso, se dice que σ *los distingue*.

Proposición 3.1.2 *Cualesquiera dos estados distinguibles en una máquina-*(n, m, k) *lo son mediante una palabra de longitud a lo sumo* $n - k$.

En efecto, para cada $i \geq 0$ sea I_i el conjunto de parejas de estados que no pueden ser distinguidos por palabras de longitud i,

$$I_i = \{(q_1, q_2) \in Q^2 | \forall \sigma \in Ent^* : \mathrm{long}(\sigma) \leq i \Rightarrow f(q_1, \sigma) = f(q_2, \sigma)\}.$$

I_i es una relación de equivalencia. Sea ι_i el índice de la relación I_i.

Ya que la sucesión de relaciones $\{I_i\}_i$ es decreciente, o sea,

$$\forall (q_1, q_2) \in Q^2 : \quad (q_1, q_2) \in I_i \Rightarrow (q_1, q_2) \in I_{i-1},$$

se tiene que la correspondiente sucesión de índices $\{\iota_i\}_i$ es creciente,

$$\mathrm{card}(Sal') = \iota_0 \leq \iota_1 \leq \cdots \leq \iota_i \leq \iota_{i+1} \leq \cdots \tag{3.1}$$

Naturalmente, $\mathrm{Max}_{i \geq 0}\{\iota_i\} \leq \iota_{+\infty} \leq \mathrm{card}(Q)$, donde $\iota_{+\infty}$ es el índice de la relación "$\sim_{Ind}$".

Por tanto, necesariamente, $\exists i_0 : \ \iota_{i_0} = \iota_{i_0+1}$, y, de hecho, $\forall i \geq i_0 : \ I_i = I_{i_0}$. De aquí puede verse que las desigualdades intermedias en la serie de relaciones 3.1 son estrictas, es decir

$$\mathrm{card}(Sal') = \iota_0 < \iota_1 < \cdots < \iota_{i_0} \leq \mathrm{card}(Q),$$

y, en particular, $\mathrm{card}(Sal') + i_0 \leq \mathrm{card}(Q)$. Por tanto, el número de relaciones distintas de la forma I_i está mayorizado por la desigualdad $i_0 \leq \mathrm{card}(Q) - \mathrm{card}(Sal')$, *quod erat demonstratum*.

La proposición anterior proporciona un algoritmo elemental para calcular, de manera exhaustiva, al cociente $Q/\sim_{Ind}$:

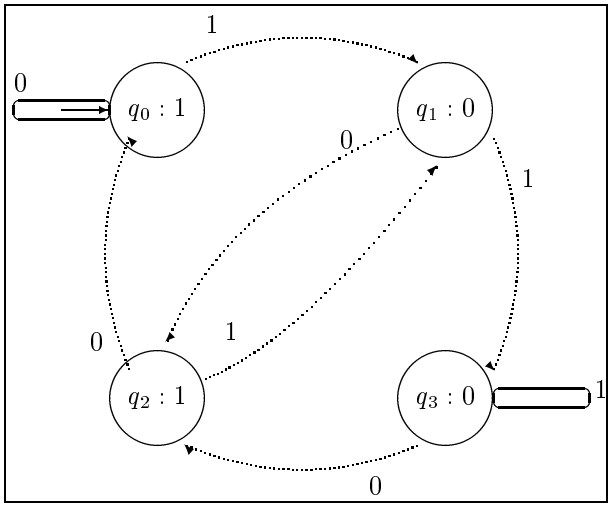
1. Sean $ne = \mathrm{card}(Ent)$, $nq = \mathrm{card}(Q)$ y $ns = \mathrm{card}(Sal')$ las cardinalidades de los conjuntos de símbolos de entrada, estados y símbolos de salida asumidos.
2. Sea $k = \dfrac{ne^{nq-ns+1} - 1}{ne - 1}$ el número de palabras de longitud a lo más $nq - ns$.
3. Fórmese la matriz $F = (f_{ij})_{1 \leq i \leq k, 1 \leq j \leq nq}$ tal que $\forall i, j : \ f_{ij} = f(q_j, \sigma_i)$.
4. Dos estados son indistinguibles entre sí si los correspondientes vectores columnas en F coinciden.

Ejemplo. Residuos módulo 4: Una máquina que reconoce números binarios congruentes con 2 o con 4, módulo 4, se muestra en la figura (3.4).

Se tiene $ne = \mathrm{card}(Ent) = 2$, $nq = \mathrm{card}(Q) = 4$ y $ns = \mathrm{card}(Sal') = 2$, luego $k = 2^{4-2+1} - 1 = 7$.

La tabla para reconocer estados indistinguibles queda:

palabra\estado	q_0	q_1	q_2	q_3
nil	1	0	1	0
0	1	1	1	1
1	0	0	0	0
00	1	1	1	1
01	0	0	0	0
10	1	1	1	1
11	0	0	0	0



tran	0	1
q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_0	q_1
q_3	q_2	q_3

res	1
q_0	1
q_1	0
q_2	1
q_3	0

Figura 3.4: Reconocedor de números binarios congruentes con 2 o 4 módulo 4.

Por tanto, las parejas $[q_0] = \{q_0, q_2\}$ y $[q_1] = \{q_1, q_3\}$ constan de estados indistinguibles entre sí.

Se ve directamente que la relación "$\sim_{Ind}$" es de equivalencia en el conjunto de estados Q. Por tanto, el cociente $Q/\sim_{Ind}$ es una partición de Q. Más aún, si dos estados son indistinguibles, lo son también los estados a los que transitan bajo cualquier estímulo,

$$q_1 \sim_{Ind} q_2 \;\Rightarrow\; \forall e \in Ent : tran(q_1, e) \sim_{Ind} tran(q_2, e),$$

en otras palabras, la noción de indistinguibilidad es congruente con las transiciones de la máquina *MMo*.

Observación 3.1.1 *El espacio cociente $Q/\sim_{Ind}$ puede ser dotado de una estructura de máquina de Moore.*

En efecto, la construcción es la siguiente:

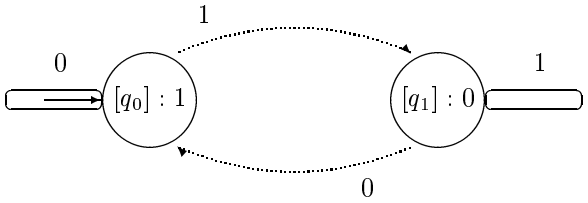
estados: clases de equivalencia $[q] \in Q/\sim_{Ind}$, con $q \in Q$,

transición: $tran_{Ind} : ([q], e) \mapsto [tran(q, e)]$, o sea, la clase de indistinguibilidad de q transita, bajo e a la clase del estado al que transita q. Esta definición tiene sentido pues la indistinguibilidad es congruente con las transiciones,

respuesta: $res_{Ind} : [q] \mapsto res(q)]$, la cual función también está bien definida, y

estado inicial: $q_{0,Ind} = [q_0]$, es decir, el nuevo estado inicial es la clase de equivalencia del estado inicial original. En esta clase están incluidos todos los estados indistinguibles respecto a q_0.

Así por ejemplo, la máquina cociente del último ejemplo es la siguiente:



Observación 3.1.2 *La máquina cociente tiene un número de estados que no excede al de la máquina dada. De hecho, si hubiera una pareja de estados indistinguibles entonces el número de estados de la máquina cociente es estrictamente menor.*

Además, la máquina cociente es equivalente a la máquina dada.

En efecto, veamos que para todo $\sigma \in Ent^*$, $res^*_{Ind}(\sigma) = res^*(\sigma)$.

Para $\sigma = nil$ se tiene

$$res^*_{Ind}(nil) = res_{Ind}([q_0]) = res(q_0) = res^*(nil).$$

Ahora, para $\sigma \in Ent^*$ y $e \in Ent$, al suponer que $res^*_{Ind}(\sigma) = res^*(\sigma)$, se tiene

$$\begin{aligned} res^*_{Ind}(\sigma e) &= res^*_{Ind}(\sigma) res_{Ind}(tran^*_{Ind}([q_0], \sigma e)) \\ &= res^*(\sigma) res_{Ind}([tran^*(q_0, \sigma e)]) \\ &= res^*(\sigma) res(tran^*(q_0, \sigma e)) \\ &= res^*(\sigma e) \end{aligned}$$

3.2 Autómatas finitos

3.2.1 Conceptos básicos

En la sección anterior se vió que basta considerar a las máquinas de Moore en el estudio de las propiedades relativas a las máquinas finitas. Más aún, en las máquinas de Moore se puede omitir a la función de respuesta y considerar que en cada estado, el símbolo que se emite al llegar a él es precisamente la etiqueta con que se "distingue" a ese estado. Por tanto, si se está interesado en reconocer a todas las palabras que al final emiten un símbolo determinado, bastará con distinguir como finales a los estados que emiten ese símbolo. Las palabras reconocidas son todas aquellas que llegan a los estados finales a partir del estado inicial.

Un *autómata finito (determinista)* es pues una estructura de la forma

$$AF = (Q, Ent, tran, q_0, F)$$

donde

$$\begin{aligned} Q &: \text{es el conjunto de } \textit{estados}, \\ Ent &: \text{es el alfabeto de } \textit{entrada}, \\ tran &: Q \times Ent \to Q, \quad \text{es la función de } \textit{transición}, \\ q_0 \in Q &: \text{es el estado } \textit{inicial}, \text{ y} \\ F \subset Q &: \text{es el conjunto de estados } \textit{finales}. \end{aligned}$$

Un *semiautómata finito* es una estructura de la forma

$$SAF = (Q, Ent, tran, q_0)$$

es decir, es un "autómata finito" en el que no se ha especificado estados finales. Todo autómata finito puede ser visto como un semiautómata con estados finales distinguidos. El semiautómata determinado por un autómata se dice ser el semiautómata *subyacente* del autómata. Todas las nociones y aseveraciones hechas sobre semiautómatas serán válidas también en los autómatas de los que son subyacentes.

Como en las máquinas finitas, ya sea de Mealy o de Moore, en cada semiautómata extendemos la función de transición $tran : Q \times Ent \to Q$ a una función $tran^* : Q \times Ent^* \to Q$, haciendo, para cada estado $q \in Q$:

$$\begin{aligned} tran^*(q, nil) &= q, \\ \forall \sigma \in Ent^*, s \in Ent : \ tran^*(q, \sigma s) &= tran(tran^*(q, \sigma), s) \end{aligned}$$

Sea $T : Ent^* \to Q$ la función $\sigma \mapsto T(\sigma) = tran^*(q_0, \sigma)$. Un estado $q \in Q$ se dice ser *accesible* si está en la imagen de T, es decir, si $\exists \sigma \in Ent^* : T(\sigma) = q$. La *parte accesible* de AF es la imagen de T, es decir, consta de todos los estados accesibles a partir del estado inicial.

Lema 3.2.1 *Sea $SAF = (Q, Ent, tran, q_0)$ un semiautómata finito. Cualquier estado accesible se alcanza mediante una palabra de longitud a lo sumo el número total de estados, $n = \mathrm{card}(Q)$. En otras palabras, la parte accesible del semiautómata coincide con el conjunto $Acc(SAF) = \{T(\sigma) | \mathrm{long}(\sigma) \leq \mathrm{card}(Q)\}$.*

En efecto, para cada $m \in \mathbb{N}$ sea $Ent^{(m)} = \bigcup_{\mu \leq m} Ent^{\mu}$ el conjunto de palabras de longitud a lo sumo m. La colección de conjuntos es un recubrimiento (creciente) del diccionario Ent^* mediante conjuntos anidados:

$$Ent^* = \bigcup_{m \geq 0} Ent^{(m)} \qquad \&$$
$$\forall m \geq 0 \quad : \quad [Ent^{(m)} \subset Ent^{(m+1)}] \wedge [Ent^{(m)} \neq Ent^{(m+1)}]$$

Consecuentemente, $\{T(Ent^{(m)})\}_{m \geq 0}$ es también un recubrimiento de Q mediante conjuntos anidados. Por ser Q finito, necesariamente para algún índice m_0 se ha de tener que $T(Ent^{(m_0)}) = T(Ent^{(m_0+1)})$, y, de hecho, para todo $m \geq m_0$, $T(Ent^{(m)}) = T(Ent^{(m_0)})$. Así pues, se tiene una cadena finita de inclusiones,

$$\{q_0\} = T(Ent^{(0)}) \subset \cdots \subset T(Ent^{(m_0)}) \subset Q.$$

Como cualesquiera dos conjuntos consecutivos $T(Ent^{(k)})$, $T(Ent^{(k+1)})$ pueden diferir en al menos un elemento en Q se tiene que $m_0 \leq n$. Además, $T(Ent^*) = T(Ent^{(m_0)})$. De aquí se sigue la tesis del lema, *quod erat demonstratum (q.e.d.).*

La parte *accesible*, SAF^{acc}, de un semiautómata SAF consta de todos sus estados accesibles.

Naturalmente, se tiene un algoritmo elemental para construir la parte accesible de cualquier semiautómata finito:

1. Consideremos dos listas: una de estados *ya revisados* y otra de estados *por revisar*. Inicialmente la primera está vacía y la segunda consta sólo del estado inicial.
2. Para cada estado por revisar,
   (a) se toma a ese estado como *actual* q,
   (b) para cada símbolo de entrada $e \in Ent$ sea $p = tran(q, e)$. Si p aparece en alguna de las dos listas se pasa al siguiente símbolo, en otro caso se lo coloca al final de los estados a revisar,
   (c) se coloca el estado actual en la lista de los ya revisados.

En la figura 3.5 se presenta un seudocódigo de este algoritmo.

El lema anterior implica que el número de iteraciones en el ciclo principal del algoritmo anterior no excede al número de estados en el autómata.

Ejemplo. Si $Ent = \{1\}$ consta de un único símbolo entonces el algoritmo 3.5 muestra que la parte accesible tiene forma de la letra griega "rho", ρ, es decir, existen $n, m \geq 0$ tales que

$$q_0 \xrightarrow{1} q_1 \xrightarrow{1} \cdots \xrightarrow{1} q_n \xrightarrow{1} \underbrace{q_{n+1} \xrightarrow{1} \cdots \xrightarrow{1} q_{n+m} \xrightarrow{1} q_n}_{\text{ciclo}}$$

Sea $AF = (Q, Ent, tran, q_0, F)$ un autómata finito. Decimos que una palabra $\sigma \in Ent^*$ es *reconocida* por A si $T(\sigma) \in F$, es decir, σ es reconocida si al aplicarla a AF desde el estado inicial se arriba a uno de los estados finales. El *lenguaje reconocido* por AF consta de todas las palabras reconocidas por AF:

$$L(AF) = \{\sigma \in Ent^* | T(\sigma) \in F\} = T^{-1}(F).$$

Diremos que un autómata AF_1 *subsume* a otro autómata AF_2 si $L(AF_2) \subset L(AF_1)$. La relación de "subsunción" es reflexiva y transitiva.

Input: A finite semiautomaton $SAF = (Q, Ent, tran, q_0)$.
Output: The accesible part SAF^{acc}.

Tst: List of *tested* states.
TBTst: List of *to-be-tested* states.

```
{   Tst := nil ; TBTst := [q0] ;
    while TBTst ≠ nil do
    {   Pop(q, TBTst) ;
        for each e ∈ Ent do
        {   p := tran(q, e) ;
            if p ∉ Tst ∪ TBTst ∪ {q} then TBTst := Append(TBTst, [p])} ;
        Tst := Append(Tst, [q])
    } ;
    SAF^acc consists of the states in Tst
}
```

Figura 3.5: Cálculo de la *parte accesible*.

Diremos que dos autómatas son *equivalentes* si uno subsume al otro, es decir, si coinciden los lenguajes reconocidos por ellos. Esta es una relación de equivalencia entre autómatas.

Diremos que un lenguaje $L \subset Ent^*$ es *regular-AF* si existe un autómata finito AF tal que $L = L(AF)$.

Ejemplos. Sea $Ent = \{0, 1\}$.

1. Construyamos un autómata que reconozca cadenas binarias con números pares de 0's y de 1's. Consideremos los estados siguientes:

$$\begin{aligned} q_0 &: \text{número par de 0's y número par de 1's,} \\ q_1 &: \text{número par de 0's y número impar de 1's,} \\ q_2 &: \text{número impar de 0's y número par de 1's,} \\ q_3 &: \text{número impar de 0's y número impar de 1's.} \end{aligned}$$

La tabla de transición queda definida de manera natural:

$tran$	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

El estado inicial es q_0 y el conjunto de estados finales es $F = \{q_0\}$.

Es fácil ver que el lenguaje reconocido por este autómata es

$$L = \{\sigma \in Ent^* | (\text{No. de 0's en } \sigma) \bmod 2 = 0 = (\text{No. de 1's en } \sigma) \bmod 2\}.$$

El lenguaje L es pues regular-AF.

En este ejemplo, es también muy fácil ver que para cada $\sigma \in Ent^*$, $T(\sigma)$ queda determinada por las paridades de 0's y de 1's en σ.

2. Consideremos el autómata con tabla de transición

tran	0	1
q_0	q_1	q_3
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_3	q_3

y estado inicial q_0. Observemos que

- si se arriba al estado q_3 ya no se sale de ahí,
- se arriba a q_3 si inicialmente aparece un 1 y no hay 0's que lo precedan, o bien, habiendo llegado un bloque de 0's y luego uno de 1's, reaparece un 0.

Así pues, si el conjunto de estados finales es $F = \{q_2\}$ entonces el lenguaje reconocido por este autómata es $L = \{0^n 1^m | n, m > 0\}$.

3.2.2 Homomorfismos

Sean $SAF_1 = (Q_1, Ent, tran_1, q_{01})$, $SAF_2 = (Q_2, Ent, tran_2, q_{02})$ dos semiautómatas finitos. Un *homomorfismo*, escrito formalmente $h : SAF_1 \to SAF_2$, es una función $h : Q_1 \to Q_2$ que cumple con las condiciones siguientes

- h "preserva" a las transiciones, es decir,

$$\forall q_1 \in Q_1, e \in Ent : \quad h(tran_1(q_1, e)) = tran_2(h(q_1), e),$$

en otras palabras, el siguiente diagrama es conmutativo

$$\begin{array}{ccc} q_1 & \stackrel{h}{\to} & q_2 \\ e \downarrow & & \downarrow e \\ tran_1(q_1, e) & \stackrel{h}{\to} & tran_2(q_2, e) \end{array}$$

y

- el estado inicial de SAF_1 se aplica en el de SAF_2, es decir, $h(q_{01}) = q_{02}$.

Si $AF_1 = (SAF_1, F_1)$ y $AF_2 = (SAF_2, F_2)$ son dos autómatas con semiautómatas adyacentes SAF_1 y SAF_2 entonces un homomorfismo $h : AF_1 \to AF_2$ es un homomorfismo $h : SAF_1 \to SAF_2$ tal que los estados finales de AF_1 se aplican en finales de AF_2, es decir, $h(F_1) \subset F_2$.

Si existe un homomorfismo $h : AF_1 \to AF_2$ decimos que AF_2 es una *imagen homomorfa* de AF_1.

Ejemplo. Consideremos el semiautómata *SA35* de 35 estados siguiente:

tran	0	1
q_0	q_1	q_2
q_1	q_3	q_4
q_2	q_5	q_6
q_3	q_7	q_6
q_4	q_8	q_9
q_5	q_{10}	q_{11}
q_6	q_{12}	q_2

tran	0	1
q_7	q_{13}	q_9
q_8	q_{14}	q_{15}
q_9	q_{16}	q_4
q_{10}	q_{17}	q_2
q_{11}	q_{18}	q_{19}
q_{12}	q_{20}	q_{11}
q_{13}	q_1	q_2

tran	0	1
q_{14}	q_{21}	q_4
q_{15}	q_{22}	q_{23}
q_{16}	q_{24}	q_{15}
q_{17}	q_{25}	q_{19}
q_{18}	q_{26}	q_{15}
q_{19}	q_{27}	q_{11}
q_{20}	q_{28}	q_6

tran	0	1
q_{21}	q_{29}	q_{23}
q_{22}	q_{27}	q_{11}
q_{23}	q_{26}	q_{15}
q_{24}	q_{30}	q_9
q_{25}	q_5	q_6
q_{26}	q_{31}	q_{11}
q_{27}	q_{32}	q_{15}

tran	0	1
q_{28}	q_{33}	q_{19}
q_{29}	q_8	q_9
q_{30}	q_{34}	q_{23}
q_{31}	q_{11}	q_{23}
q_{32}	q_{15}	q_{19}
q_{33}	q_{12}	q_2
q_{34}	q_{16}	q_4

Sea *SA5* el semiautómata de 5 estados siguiente:

tran	0	1
p_0	p_1	p_2
p_1	p_2	p_3
p_2	p_4	p_2
p_3	p_0	p_1
p_4	p_0	p_1

Sea $h : SA35 \to SA5$ la aplicación definida como sigue:

q_0	$\mapsto$	p_0	q_7	$\mapsto$	p_4	q_{14}	$\mapsto$	p_1	q_{21}	$\mapsto$	p_2	q_{28}	$\mapsto$	p_1
q_1	$\mapsto$	p_1	q_8	$\mapsto$	p_0	q_{15}	$\mapsto$	p_2	q_{22}	$\mapsto$	p_4	q_{29}	$\mapsto$	p_4
q_2	$\mapsto$	p_2	q_9	$\mapsto$	p_1	q_{16}	$\mapsto$	p_2	q_{23}	$\mapsto$	p_2	q_{30}	$\mapsto$	p_0
q_3	$\mapsto$	p_2	q_{10}	$\mapsto$	p_0	q_{17}	$\mapsto$	p_1	q_{24}	$\mapsto$	p_4	q_{31}	$\mapsto$	p_0
q_4	$\mapsto$	p_3	q_{11}	$\mapsto$	p_1	q_{18}	$\mapsto$	p_2	q_{25}	$\mapsto$	p_2	q_{32}	$\mapsto$	p_1
q_5	$\mapsto$	p_4	q_{12}	$\mapsto$	p_4	q_{19}	$\mapsto$	p_3	q_{26}	$\mapsto$	p_4	q_{33}	$\mapsto$	p_2
q_6	$\mapsto$	p_2	q_{13}	$\mapsto$	p_0	q_{20}	$\mapsto$	p_0	q_{27}	$\mapsto$	p_0	q_{34}	$\mapsto$	p_1

Entonces h es un homomorfismo de semiautómatas.

Si consideramos como estados finales en $SA35$ a los estados $q_2, q_3, q_6, q_{15}, q_{16}, q_{18}, q_{21}, q_{23}, q_{25}, q_{33}$ y en $SA5$ consideramos únicamente a p_2 como estado final, se tiene que h es también un homomorfismo de autómatas.

Proposición 3.2.1 *Si existe un homomorfismo $h : AF_1 \to AF_2$ entonces el autómata AF_2 subsume al autómata AF_2.*

Para esto, hay que ver que toda palabra reconocida en AF_1 también ha de ser reconocida en AF_2. Se tiene las implicaciones siguientes:

$$\begin{array}{lcll} \sigma \in L(AF_1) & \Rightarrow & tran_1^*(q_{01}, \sigma) \in F_1 & \\ & \Rightarrow & h(tran_1^*(q_{01}, \sigma)) \in h(F_1) & : \text{ sólo notación,} \\ & \Rightarrow & tran_2^*(h(q_{01}), \sigma) \in h(F_1) & : \ h \text{ preserva transiciones,} \\ & \Rightarrow & tran_2^*(q_{02}, \sigma) \in F_2 & : \ h \text{ preserva estados inicial y finales,} \\ & \Rightarrow & \sigma \in L(AF_2) & \end{array}$$

qed.

Es evidente que vale la

Observación 3.2.1 *Si el autómata $AF_2 = (Q_2, Ent, tran_2, q_{20}, F_2)$ es una imagen homomorfa del autómata $AF_1 = (Q_1, Ent, tran_1, q_{10}, F_1)$ mediante un homomorfismo $h : AF_1 \to AF_2$ entonces las siguientes tres proposiciones son lógicamente equivalentes a pares:*

1. *AF_1 y AF_2 son equivalentes.*
2. *$\forall q \in Q_1 : \ q \in F_1 \ \Leftrightarrow \ h(q) \in F_2$.*
3. *$h^{-1}(F_2) = F_1$.*

Un homomorfismo $h : AF_1 \to AF_2$ es un *isomorfismo* si h es inyectivo y suprayactivo. Si existe un isomorfismo de un autómata AF_1 a un autómata AF_2, se dice que ambos son *isomorfos*, y en tal caso difieren tan solo por la manera en la que se nombra a sus estados.

3.2.3 Monoide de un semiautómata

Sea $SAF = (Q, Ent, tran, q_0)$ un semiautómata finito. Consideremos la relación "$\equiv_{SAF}$" en el diccionario Ent^* definida como sigue,

$$\forall \sigma_1, \sigma_2 \in Ent^* : \ \sigma_1 \equiv_{SAF} \sigma_2 \ \Leftrightarrow \ \forall q \in Q [tran^*(q, \sigma_1) = tran^*(q, \sigma_2)], \tag{3.2}$$

es decir, dos palabras están relacionadas si ambas "actúan" de igual manera en todo el conjunto de estados. "$\equiv_{SAF}$" es una relación de equivalencia que es además "congruente con la concatenación":

$$\sigma_1 \equiv_{SAF} \sigma_2, \tau_1 \equiv_{AF} \tau_2 \ \Rightarrow \ \sigma_1\tau_1 \equiv_{SAF} \sigma_2\tau_2.$$

Por tanto el cociente $S_{SAF} = (Ent^* / \equiv_{SAF})$ tiene una estructura de monoide, y se dice ser el *monoide del semiautómata SAF.*

Cada elemento en S_{SAF} determina una función $Q \to Q$. Si n es el número de estados en Q, entonces hay n^n funciones de Q en Q y, consecuentemente, la cardinalidad de S_{SAF}, es decir, el índice de la relación "$\equiv_{AF}$", no puede exceder n^n. De hecho, esta es una cota que, aunque muy grande, llega a alcanzarse. Como un mero ejemplo de esto, sea Q un conjunto de n estados y $q_0 \in Q$ un estado distinguido como inicial. Sea $(f_\nu)_{\nu=1}^{n^n}$ una enumeración de Q^Q, es decir , del conjunto de todas las funciones $Q \to Q$. Ahora, consideremos un alfabeto $E = (e_\nu)_{\nu=1}^{n^n}$ de n^n símbolos distintos y sea SAF el semiautómata sobre Q que a cada símbolo e_ν le asocia la función f_ν. Es claro que el monoide determinado por SAF coincide con Q^Q, dotado de la composición de funciones como operación, y tiene consecuentemente n^n elementos.

Ejemplos. Sea $Ent = \{0, 1\}$.

1. Para el semiautómata con tabla de transición

tran	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

(3.3)

se tiene que "$\equiv_{AF}$" se resume como sigue:

palabras equivalentes	función definida
$nil, 00, 11$	$\begin{pmatrix} 0123 \\ 0123 \end{pmatrix}$
$0, 011$	$\begin{pmatrix} 0123 \\ 2301 \end{pmatrix}$
$1, 010$	$\begin{pmatrix} 0123 \\ 1032 \end{pmatrix}$
$01, 10$	$\begin{pmatrix} 0123 \\ 3210 \end{pmatrix}$

aquí, por $\begin{pmatrix} 0 & 1 & 2 & 3 \\ i_0 & i_1 & i_2 & i_3 \end{pmatrix}$ denotamos a la función $q_j \mapsto q_{i_j}$.

De la tabla anterior es posible calcular la clase de equivalencia de cualquier palabra $\sigma \in Ent^*$. Dada una palabra σ, cada vez que se encuentra en ella una de las partículas de la primera columna de la tabla en la ec. (3.3), sustituímos a esa partícula por la primera en dicha columna. Procedemos recursivamente en tanto esto sea posible. Por ejemplo, la clase de 1010 se calcula como sigue:

$$1010 = \underline{10}\,\underline{10} \to 0\,\underline{10}\,1 \to \underline{00}\,\underline{11} \to nil\ nil = nil$$

Tenemos pues que en nuestro ejemplo hay 4 clases de equivalencia: $[nil], [0], [1], [01]$. La tabla de multiplicación del monoide de este semiautómata es la siguiente:

$\cdot$	$[nil]$	$[0]$	$[1]$	$[01]$
$[nil]$	$[nil]$	$[0]$	$[1]$	$[01]$
$[0]$	$[0]$	$[nil]$	$[01]$	$[1]$
$[1]$	$[1]$	$[01]$	$[nil]$	$[0]$
$[01]$	$[01]$	$[1]$	$[0]$	$[nil]$

2. Para el semiautómata con tabla de transición

tran	0	1
q_0	q_1	q_3
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_3	q_3

se tiene que "$\equiv_{AF}$" se resume como sigue:

palabras equivalentes	función definida
nil	$\begin{pmatrix} 0123 \\ 0123 \end{pmatrix}$
$0, 00$	$\begin{pmatrix} 0123 \\ 1133 \end{pmatrix}$
$1, 11$	$\begin{pmatrix} 0123 \\ 3223 \end{pmatrix}$
$01, 011$	$\begin{pmatrix} 0123 \\ 2233 \end{pmatrix}$
$10, 010, 100, 101$	$\begin{pmatrix} 0123 \\ 3333 \end{pmatrix}$

En este ejemplo hay 5 clases de equivalencia: $[nil]$, $[0]$, $[1]$, $[01]$, $[10]$. La tabla de multiplicación del monoide de este semiautómata es la siguiente:

$\cdot$	$[nil]$	$[0]$	$[1]$	$[01]$	$[10]$
$[nil]$	$[nil]$	$[0]$	$[1]$	$[01]$	$[10]$
$[0]$	$[0]$	$[0]$	$[01]$	$[01]$	$[10]$
$[1]$	$[1]$	$[10]$	$[1]$	$[10]$	$[10]$
$[01]$	$[01]$	$[10]$	$[01]$	$[10]$	$[10]$
$[10]$	$[10]$	$[10]$	$[10]$	$[10]$	$[10]$

Presentamos a continuación un procedimiento para calcular el monoide de un semiautómata dado.

Para cada palabra $\sigma \in Ent^*$, sea $\mathbf{t}_\sigma \in Q^Q$ el vector con entradas en Q, indicado por índices en Q, tal que para todo $q \in Q$, $(\mathbf{t}_\sigma)_q = tran^*(q, \sigma)$. Se tiene la recurrencia,

$$\forall \sigma \in Ent^*, e \in Ent, q \in Q : \ (\mathbf{t}_{\sigma e})_q = tran((\mathbf{t}_\sigma)_q, e).$$

Utilizaremos un arreglo de palabras a revisar y otro de palabras ya revisadas.

1. Inicialmente se coloca la palabra vacía en la lista de palabras a revisar y la de revisadas es vacía.
2. Para cada palabra por revisar,
   (a) se toma la primera palabra σ a revisar,
   (b) si el vector $\mathbf{t}_\sigma$ coincide con el vector correspondiente a una palabra ya revisada, se asocia σ a esa palabra y se continúa este ciclo,
   (c) en otro caso, se incorpora σ a las ya revisadas y sus palabras hijas, σe, $e \in Ent$, se colocan al final de la lista de palabras a revisar.
3. La lista de palabras revisadas ha de contener al final el monoide del semiautómata.

De manera más precisa, presentamos un seudocódigo en la figura 3.6.

Ejemplos

1. Para el ejemplo 1 anterior, el algoritmo 3.6 genera la tabla mostrada en la tabla (3.4).
Ahí vemos que el monoide posee 4 elementos y la tabla generada por el algoritmo posee $2 \cdot 4 = 8$ renglones.

2. Para el ejemplo 2 anterior, el algoritmo (3.6) genera la tabla mostrada en la tabla (3.5).

Input: A semiautomaton $SAF = (Q, Ent, tran, q_0)$.
Output: The semiautomaton's monoid.

Tst: List of *tested* words.
TBTst: List of *to-be-tested* words.

{ $Tst := nil$; $TBTst := [nil]$;
while $TBTst \neq nil$ **do**
{ $Pop(\sigma, TBTst)$;
if $[\forall \tau \in Tst : \mathbf{t}_\sigma \neq \mathbf{t}_\tau]$ **then**
{ $Tst := Append(Tst, [\sigma])$;
$TBTst := Append(TBTst, [\sigma e | e \in Ent])$ }
else connect σ with $\tau \in Tst$ such that $\mathbf{t}_\sigma = \mathbf{t}_\tau$
} ;
the monoid consists of the words in *Tst*
}

Figura 3.6: Cálculo del *monoide* de un semiautómata.

σ	q_0	q_1	q_2	q_3	$\tau \equiv_{AF} \sigma$
nil	q_0	q_1	q_2	q_3	–
0	q_2	q_3	q_0	q_1	–
1	q_1	q_0	q_3	q_2	–
00	q_0	q_1	q_2	q_3	*nil*
01	q_3	q_2	q_1	q_0	–
10	q_3	q_2	q_1	q_0	01
11	q_0	q_1	q_2	q_3	*nil*
010	q_1	q_0	q_3	q_2	1
011	q_2	q_3	q_0	q_1	0

Tabla 3.4: Cálculo del monoide del ejemplo 1.

σ	q_0	q_1	q_2	q_3	$\tau \equiv_{AF} \sigma$
nil	q_0	q_1	q_2	q_3	–
0	q_1	q_1	q_3	q_3	–
1	q_3	q_2	q_2	q_3	–
00	q_1	q_1	q_3	q_3	0
01	q_2	q_2	q_3	q_3	–
10	q_3	q_3	q_3	q_3	–
11	q_3	q_2	q_2	q_3	1
010	q_3	q_3	q_3	q_3	10
011	q_2	q_2	q_3	q_3	01
100	q_3	q_3	q_3	q_3	10
101	q_3	q_3	q_3	q_3	10

Tabla 3.5: Cálculo del monoide del ejemplo 2.

σ	q_0	q_1	q_2	q_3	q_4	$\tau \equiv_{AF} \sigma$
nil	q_0	q_1	q_2	q_3	q_4	$-$
a	q_2	q_2	q_2	q_3	q_3	$-$
b	q_4	q_4	q_2	q_2	q_2	$-$
c	q_3	q_3	q_2	q_2	q_2	$-$
aa	q_2	q_2	q_2	q_3	q_3	a
ab	q_3	q_3	q_2	q_2	q_2	$-$
ac	q_3	q_3	q_2	q_2	q_2	ab
ba	q_3	q_3	q_2	q_2	q_2	c
bb	q_3	q_3	q_2	q_2	q_2	ab
bc	q_3	q_3	q_2	q_2	q_2	ab
ca	q_3	q_3	q_2	q_2	q_2	c
cb	q_3	q_3	q_2	q_2	q_2	ab
cc	q_3	q_3	q_2	q_2	q_2	ab
aba	q_3	q_3	q_2	q_2	q_2	ab
abb	q_3	q_3	q_2	q_2	q_2	ab
abc	q_3	q_3	q_2	q_2	q_2	ab

(a)

$\cdot$	$[nil]$	$[a]$	$[b]$	$[c]$	$[ab]$
$[nil]$	$[nil]$	$[a]$	$[b]$	$[c]$	$[ab]$
$[a]$	$[a]$	$[a]$	$[ab]$	$[ab]$	$[ab]$
$[b]$	$[b]$	$[c]$	$[ab]$	$[ab]$	$[ab]$
$[c]$	$[c]$	$[c]$	$[ab]$	$[ab]$	$[ab]$
$[ab]$	$[ab]$	$[ab]$	$[ab]$	$[ab]$	$[ab]$

(b)

Tabla 3.6: Cálculo del monoide del ejemplo 3.

3. Consideremos el semiautómata de 5 estados sobre el alfabeto $Ent = \{a, b, c\}$ siguiente:

$tran$	a	b	c
q_0	q_2	q_4	q_3
q_1	q_2	q_4	q_3
q_2	q_2	q_4	q_2
q_3	q_3	q_2	q_2
q_4	q_3	q_2	q_2

El algoritmo (3.6) genera la tabla mostrada en la tabla (3.6-a). En este ejemplo hay 5 clases de equivalencia: $[nil], [a], [b], [c], [ab]$. La tabla de multiplicación del monoide de este semiautómata se muestra en la tabla (3.6-b).

4. Sea $n > 0$ y sea A_n el semiautómata consistente de n estados $q_0, q_1, \ldots, q_{n-1}$, cuya tabla de transición es la siguiente:

$tran$	0	1
q_0	q_1	q_1
q_1	q_0	q_2
q_2	q_2	q_3
$\vdots$	$\vdots$	$\vdots$
q_{n-2}	q_{n-2}	q_{n-1}
q_{n-1}	q_{n-1}	q_0

En otras palabras, la acción del estímulo 0 puede identificarse con la transposición (0 1) y la de 1 puede identificarse con el ciclo $(0\ 1\ \cdots\ n-2\ n-1)$.

Es bien sabido que en el grupo S_n de permutaciones de n objetos se cumple lo siguiente:

- Toda permutación es un producto de ciclos. Por tanto, los ciclos generan a S_n.
- Todo ciclo es un producto de transposiciones, de hecho,

$$(i) \quad = \quad (i\ j)(j\ i)$$

permutación	palabras representativas
0123	$00, 0000, 1111$
0132	11011
0213	$01011, 11101$
0231	$10, 0010, 1000$
0312	$0111, 1010$
0321	01101
1023	$0, 000, 00000, 01111, 10101, 11110$
1032	$011011, 110110$
1203	$010110, 110111, 111010$
1230	$1, 001, 100, 00001, 00100, 10000, 11111$
1302	$101, 00101, 01110, 10001, 10100$
1320	$010111, 011010, 111011$
2013	1011
2031	$010, 00010, 01000$
2103	10110
2130	$01, 0001, 0100$
2301	$11, 0011, 1001, 1100$
2310	$110, 00110, 10010, 11000$
3012	$111, 00111, 01010, 10011, 11001, 11100$
3021	1101
3102	$0101, 1110$
3120	$10111, 11010$
3201	$011, 00011, 01001, 01100$
3210	0110

Una lista de la forma $i_0 i_1 i_2 i_3$, en la primera columna, representa a la permutación $\pi : j \mapsto i_j$.

Tabla 3.7: S_4 como semigrupo de A_4.

$$(i_1\ i_2\ \cdots\ i_r) \quad = \quad (i_1\ i_r)(i_1\ i_{r-1})\cdots(i_1\ i_2)$$

Por tanto, las transposiciones generan a S_n.

- Para cada transposición $(i\ j)$ se tiene $(i\ j) = (0\ i)(i\ j)(0\ i)$. Por tanto, las transposiciones de la forma $(0\ i), i < n$, generan a S_n.
- Para cada transposición $(0\ j)$ se tiene $(0\ j) = (0\ j-1)(j-1\ j)(0\ j-1)$. Por tanto, las transposiciones de la forma $(j-1\ j), j < n$, generan a S_n.
- Si π es una permutación cualquiera, se tiene que $(\pi(0)\ \pi(1)) = \pi(0\ 1)\pi^{-1}$. Así,, si $\gamma_n = (0\ 1\ \cdots\ n-2\ n-1)$ entonces $(j-1\ j) = \gamma_n^{j-1}(0\ 1)\gamma_n^{n-j+1}$. Por tanto, $(0\ 1)$ y γ_n generan a S_n.

Así pues, el monoide generado por el semiautómata coincidirá con el grupo de permutaciones S_n, el cual posee $n!$ elementos. Al aplicar el algoritmo 3.6 sobre un semiautómata de la forma A_n se generará una tabla con un orden de $2 \cdot n!$ renglones. En particular, para $n = 4$, las $24 = 4!$ permutaciones quedan generadas por las palabras mostradas en la tabla (3.7).

Lema 3.2.2 *Sea SAF un semiautómata finito. Entonces su monoide S_{SAF} puede ser dotado de una estructura de semiautómata, y si $AF = (SAF, F)$ es un autómata cuyo semiatautómata adyacente es SAF, entonces puede distinguirse a algunas clases de equivalencia de manera que el autómata resultante queda subsumido por SAF.*

En efecto, como la relación "$\equiv_{SAF}$" es congruente con la concatenación se tiene bien definida la transición

$$\begin{array}{rcl} tran_S : S_{SAF} \times Ent & \to & S_{SAF} \\ ([\sigma], e) & \mapsto & tran_S([\sigma], e) = [\sigma e] \end{array}$$

Consideremos a una clase de equivalencia $[\sigma]$ como un estado final si es que la función $Q \to Q$ que determina esa clase contiene es tal que envía al estado inicial q_0 de AF en un estado final. Es decir, hagamos

$$F_S = \{[\sigma] \in S_{AF} | tran^*(q_0, \sigma) \in F\}.$$

Seguiremos denotando por S_{AF} al autómata así construído.
Es evidente que el autómata S_{AF} queda, en efecto, subsumido por AF, *qed*.

Así, por ejemplo, si AF es un autómata finito y S_{AF} es su monoide, visto como autómata, entonces la aplicación

$$\begin{array}{rcl} h : S_{AF} & \to & AF \\ [\sigma] & \mapsto & h([\sigma]) = T(\sigma) = tran^*(q_0, \sigma) \end{array}$$

es, evidentemente, un homomorfismo. Consecuentemente, el autómata S_{AF} queda subsumido por el autómata AF del que es monoide.

Proposición 3.2.2 *Si existe un homomorfismo $h : AF_1 \to AF_2$ entonces*

1. *existe un homomorfismo de monoides $h_m : S_{AF_1} \to S_{AF_1}$, y*
2. *existe un homomorfismo de autómatas $h_a : S_{AF_1} \to S_{AF_1}$, y, de hecho, los homomorfismos h_m y h_a coinciden.*

En efecto, supongamos que $h : AF_1 \to AF_2$ es un homomorfismo. Definamos

$$\begin{array}{rcl} h_m : S_{AF_1} & \to & S_{AF_2} \\ [\sigma]_{AF_1} & \mapsto & h_m([\sigma]_{AF_1}) = [\sigma]_{AF_2} \end{array}$$

Se tiene,

- h_m está bien definida, pues,

$$\begin{array}{rcl} [\sigma_1]_{AF_1} = [\sigma_2]_{AF_1} & \Rightarrow & [\forall q_1 \in Q_1 : tran_1^*(q_{01}, \sigma_1) = tran_1^*(q_{01}, \sigma_2)] \\ & \Rightarrow & [\forall q_2 \in Q_2 : tran_2^*(q_{02}, \sigma_1) = tran_2^*(q_{02}, \sigma_2)] \\ & \Rightarrow & [\sigma_1]_{AF_2} = [\sigma_2]_{AF_2}, \end{array}$$

- h_m preserva la concatenación, pues

$$\begin{array}{rcccl} h_m([\sigma_1]_{AF_1} \cdot [\sigma_2]_{AF_1}) & = & h_m([\sigma_1 \cdot \sigma_2]_{AF_1}) & = & [\sigma_1 \cdot \sigma_2]_{AF_2} \\ & = & [\sigma_1]_{AF_2} \cdot [\sigma_2]_{AF_2} & = & h_m([\sigma_1]_{AF_1}) \cdot h_m([\sigma_2]_{AF_1}), \end{array}$$

- h_m preserva al estado inicial,

$$h_m([nil]_{AF_1}) = [nil]_{AF_2}.$$

Así pues h_m es un homomorfismo de monoides. Y se ve directamente que $h_a = h_m$ es también un homomorfismo de autómatas. *qed*

Reiteremos, para concluir esta sección, una observación ya hecha anteriormente: para cada estado $q \in Q$ en la parte accesible de un semiautómata $SAF = (Q, Ent, tran, q_0)$ existe una palabra σ_q tal que $T(\sigma_q) = q$. Por tanto, para cualesquiera dos estados p, q en la parte accesible de SAF, rige la implicación,

$$p \neq q \;\Rightarrow\; [\sigma_p]_{SAF} \neq [\sigma_q]_{SAF}.$$

Así pues el monoide de SAF tiene al menos tantos elementos cuantos tiene la parte accesible de SAF.

3.2.4 Acceso en un semiautómata

Dado un semiautómata $SAF = (Q, Ent, tran, q_0)$, definamos ahora la relación de equivalencia $\approx_{SAF}$ en el diccionario Ent^* haciendo

$$\sigma \approx_{SAF} \tau \quad \Leftrightarrow \quad T_{SAF}(q_0, \sigma) = T_{SAF}(q_0, \tau). \tag{3.4}$$

es decir, dos palabras están relacionadas si ambas arriban a un mismo estado cuando se parte desde el inicial. "$\approx_{SAF}$" es una relación de equivalencia que, aunque no es congruente con la concatenación, sí es "invariante bajo concatenaciones a la derecha":

$$\sigma_1 \equiv_{SAF} \sigma_2, \tau \in Ent^* \;\Rightarrow\; \sigma_1\tau \approx_{SAF} \sigma_2\tau.$$

De las relaciones (3.2) y (3.4) se tiene que vale la siguiente:

Observación 3.2.2 *La relación "$\equiv_{SAF}$" es un refinamiento de la relación "$\approx_{SAF}$", es decir,*

$$\forall \sigma_1, \sigma_2 \in Ent^* : \quad \sigma_1 \equiv_{SAF} \sigma_2 \;\Rightarrow\; \sigma_1 \approx_{SAF} \sigma_2.$$

Ahora, es claro que cada estado accesible del semiautómata SAF determina una clase de equivalencia bajo la relación "$\approx_{SAF}$". Por tanto, el cociente $Ent^*/\approx_{SAF}$ puede identificarse naturalmente con la parte accesible de SAF, y, consecuentemente, la estructura de semiautómata de este último se traslada de manera directa al cociente $Ent^*/\approx_{SAF}$.

3.2.5 Cocientes de autómatas

Congruencias de autómatas

Sea $SAF = (Q, Ent, tran, q_0)$ un semiautómata. Una relación de equivalencia "$\equiv$" definida en el conjunto de estados Q se dice ser *congruente* con la función de transición si

$$\forall p, q \in Q : \quad p \equiv q \quad \Rightarrow \quad \forall e \in Ent : \; tran(p, e) \equiv tran(q, e), \tag{3.5}$$

es decir para cada símbolo, los estados a los que transita el semiautómata desde sendos estados equivalentes son también equivalentes.

Si "$\equiv$" es una congruencia, entonces el cociente $Q/\equiv$ puede ser dotado de una estructura de semiautómata:

$$\begin{aligned} Q/\equiv \quad &: \quad \text{estados,} \\ tran' : ([q], e) \mapsto [tran(q, e)] \quad &: \quad \text{transición,} \\ [q_0] \quad &: \quad \text{estado inicial.} \end{aligned}$$

y éste se dice ser el *semiautómata cociente* respecto a la congruencia "$\equiv$", denotado como $SAF/\equiv$. Se tiene pues que la proyección $\pi : q \mapsto [q]$ es un homomorfismo de semiautómatas.

Más aún, si $AF = (SAF, F)$ es un autómata finito y $(\equiv) \subset Q^2$ es una congruencia en el semiautómata SAF, en el cociente $(SAF/\equiv)$ distinguimos como estados finales a las clases de la forma $[q]$ tales que $[q] \cap F \neq \emptyset$, para obtener el *autómata cociente* $(AF/\equiv)$. Aquí, la proyección π es un homomorfismo de autómatas. Consecuentemente, el cociente $(AF/\equiv)$ subsume al autómata AF.

Observación 3.2.3 *Los autómatas AF y $(AF/\equiv)$ son equivalentes si y sólo si el conjunto de estados finales F es la unión de algunas clases de equivalencia respecto a "$\equiv$".*

Para presentar unos primeros ejemplos de congruencias, consideremos homomorfismos de semiautómatas o autómatas. A lo largo de la presente sección, toda vez que hablemos de homomorfismos, supondremos que éstos son de hecho epimorfismos, es decir, como funciones, son suprayectivos.

Sean pues $SAF_1 = (Q_1, Ent, tran_1, q_{10})$ y $SAF_2 = (Q_2, Ent, tran_2, q_{20})$ dos semiautómatas y $h : SAF_1 \to SAF_2$ un homomorfismo. Entonces su núcleo, $núc(h) = \{(p, q) | h(p) = h(q)\}$, es una relación de equivalencia que es de hecho una congruencia. Por tanto, se puede definir el cociente $SAF_1/núc(h)$, y éste es tal que cada clase de equivalencia corresponde biunívocamente a un elemento en la imagen de h. Así pues, lo expuesto aquí lo enunciamos como el

Teorema 3.2.1 (de homomorfismo de semiautómatas) *Si $h : SAF_1 \to SAF_2$ es un epimorfismo de semiautómatas entonces el cociente $SAF_1/núc(h)$ es isomorfo a SAF_2. En símbolos, $SAF_1/núc(h) \equiv SAF_2$.*

Para el caso de autómatas, si $AF_1 = (SAF_1, F_1)$ y $AF_2 = (SAF_2, F_2)$ son dos autómatas y $h : AF_1 \to AF_2$ es un epimorfismo de autómatas, tal que

$$\forall q \in Q : \; q \in F_1 \quad \Leftrightarrow \quad h(q) \in F_2$$

entonces $F_1 = \{[q]_{núc(h)} | q \in F_1\}$ es la unión de algunas clases de equivalencia respecto a $núc(h)$. Por tanto los autómatas $AF_1/núc(h)$ y AF_2 son isomorfos.

Teorema 3.2.2 (de homomorfismo de autómatas) *Si $h : AF_1 \to AF_2$ es un epimorfismo de autómatas tal que F_1 es la unión de clases de equivalencia respecto a $núc(h)$, entonces el cociente $AF_1/núc(h)$ es isomorfo a AF_2. En símbolos, $AF_1/núc(h) \equiv AF_2$.*

Así pues, en cada pareja de semiautómatas o autómatas homomorfos que ya hemos visto, tendremos congruencias de semiautómatas o autómatas y correspondientes estructuras cocientes. Los ejemplos de homomorfismos de semiautómatas y de autómatas que ya hemos visto aquí, han de proporcionar ejemplos de congruencias y de autómatas cocientes.

Indistinguibilidad de estados en autómatas

Otro ejemplo, muy importante, de congruencias de autómatas está dado por la noción de *indistinguibilidad*. Para un autómata $AF = (Q, Ent, tran, q_0, F)$ definimos la siguiente relación en Q:

$$\forall p, q \in Q : \quad p \equiv_{Ind} q \quad \Leftrightarrow \quad \forall \sigma \in Ent^* : \; [tran^*(p, \sigma) \in F \quad \Leftrightarrow \quad tran^*(q, \sigma) \in F] \tag{3.6}$$

es decir, dos estados son *indistinguibles* cuando y sólo cuando ante mismas entradas o bien ambos transitan a estados finales o bien ninguno de los dos lo hace.

Es evidente que "$\equiv_{Ind}$" es una relación de equivalencia. Además, como $Ent^* = Ent \cdot Ent^*$, se tiene,

$$p \equiv_{Ind} q \quad \Leftrightarrow \quad \forall e \in Ent : tran(p, e) \equiv_{Ind} tran(q, e).$$

Por tanto "$\equiv_{Ind}$" es también una congruencia, y en consecuencia se puede construir el autómata cociente $(AF/\equiv_{Ind})$. Más aún,

$$q_1 \in F \;\&\; (q_1 \equiv_{Ind} q_2) \quad \Rightarrow \quad (tran^*(p, nil) \in F \quad \Leftrightarrow \quad tran^*(q, nil) \in F) \;\&\; q_1 \in F \quad \Rightarrow \quad q_2 \in F$$

así que F es una unión de clases de indistinguibilidad. Consecuentemente se tiene la

Observación 3.2.4 *El autómata AF es equivalente a su cociente $(AF/\equiv_{Ind})$.*

Observemos que es posible trasladar la relación de equivalencia "$\equiv_{Ind}$" a una relación "$\equiv_{Ind,\ AF}$" en el diccionario Ent^* haciendo

$$\forall \sigma_1, \sigma_2 \in Ent^* : \quad \sigma_1 \equiv_{Ind,\ AF} \sigma_2 \quad \Leftrightarrow \quad T(\sigma_1) \equiv_{Ind} T(\sigma_2) \tag{3.7}$$

la cual es invariante por la derecha.

Observación 3.2.5 *La relación "$\approx_{AF}$" es un refinamiento de la relación "$\equiv_{Ind,\ AF}$", es decir,*

$$\forall \sigma_1, \sigma_2 \in Ent^* : \quad \sigma_1 \approx_{AF} \sigma_2 \;\Rightarrow\; \sigma_1 \equiv_{Ind,\ AF} \sigma_2.$$

En efecto, si $\sigma_1 \approx_{AF} \sigma_2$ se tiene $T(\sigma_1) = T(\sigma_2)$ y al ser "$\equiv_{Ind,\ AF}$" reflexiva, $T(\sigma_1) \equiv_{Ind} T(\sigma_2)$.

Por tanto, el cociente $(Ent^*/\equiv_{Ind,\ AF})$, que prácticamente coincide con el cociente $(AF/\equiv_{Ind})$, es de cardinalidad menor o igual que la cardinalidad del autómata AF.

Veremos que el cociente $(AF/\equiv_{Ind})$ puede ser visto como "un buen representante" de la clase de autómatas equivalentes al autómata AF.

Sean $AF_1 = (Q_1, Ent, tran_1, q_{10}, F_1)$ y $AF_2 = (Q_2, Ent, tran_2, q_{20}, F_2)$ dos autómatas equivalentes.

Lema 3.2.3 *Las relaciones "$\equiv_{Ind,\ AF_1}$" y "$\equiv_{Ind,\ AF_2}$" coinciden.*

En efecto, las siguientes equivalencias lógicas son evidentes:

$$\begin{aligned}
\sigma_1 \equiv_{Ind,\ AF_1} \sigma_2 &\Leftrightarrow T_1(\sigma_1) \equiv_{Ind} T_1(\sigma_2)\\
&\Leftrightarrow \forall\tau : [tran_1^*(T_1(\sigma_1),\tau) \in F_1 \Leftrightarrow tran_1^*(T_1(\sigma_1),\tau) \in F_1]\\
&\Leftrightarrow \forall\tau : [\sigma_1\cdot\tau \in L(AF_1) \Leftrightarrow \sigma_1\cdot\tau \in L(AF_1)]\\
&\Leftrightarrow \forall\tau : [\sigma_1\cdot\tau \in L(AF_2) \Leftrightarrow \sigma_1\cdot\tau \in L(AF_2)]\\
&\Leftrightarrow \forall\tau : [tran_2^*(T_2(\sigma_1),\tau) \in F_2 \Leftrightarrow tran_2^*(T_2(\sigma_1),\tau) \in F_2]\\
&\Leftrightarrow T_2(\sigma_1) \equiv_{Ind} T_2(\sigma_2)\\
&\Leftrightarrow \sigma_1 \equiv_{Ind,\ AF_2} \sigma_2
\end{aligned}$$

Resulta entonces el

Corolario 3.2.1 *Los autómatas cocientes $(AF_1/\equiv_{Ind})$ y $(AF_2/\equiv_{Ind})$ son isomorfos.*

En efecto, ambos son isomorfos al cociente $\left(Ent^*/\equiv_{Ind,\ AF_i}\right)$, sin importar cuál de los dos $i = 1, 2$.

De todo esto resulta la siguiente proposición que explica porqué es que el autómata cociente es un buen representante de los autómatas equivalentes.

Proposición 3.2.3 (Propiedad universal) *Si AF_1 es equivalente a AF_2 entonces $(AF_1/\equiv_{Ind})$ es una imagen homomorfa de AF_2.*

Corolario 3.2.2 (Minimización) *El cociente $(AF_1/\equiv_{Ind})$ es el autómata mínimo, en cuanto al número de estados, entre todos los equivalentes al autómata AF_1.*

Para finalizar esta sección presentaremos un procedimiento para calcular el autómata cociente $(AF/\equiv_{Ind})$, dado el autómata AF.

Sea pues $AF = (Q, Ent, tran, q_0, F)$ un autómata finito. Dos estados son *distinguibles* si ellos no son indistinguibles. Observamos lo siguiente:

1. Si un estado es final y otro no lo es, esos dos estados son distinguibles, y la palabra vacía los distingue.
2. Si dos estados p_1, p_2 son distinguibles y la palabra τ los distingue, y si para un símbolo $e \in Ent$ y algunos otros estados q_1, q_2 se tiene $tran(q_1, e) = p_1$ y $tran(q_2, e) = p_2$, entonces q_1 y q_2 son distinguibles y la palabra $e\tau$ los distingue.

Con estas observaciones podemos bosquejar un algoritmo para clasificar a las parejas de estados en aquellas que son distinguibles, con respectivas palabras distinguientes, y en aquellas que son indistinguibles. A grandes rasgos, a lo largo del proceso utilizaremos las listas siguientes:

Distinguidos	:	parejas de estados ya reconocidas como distinguibles. A cada una le asociamos la palabra que las distingue,
PorProbar	:	parejas de estados por ser aún probadas,
PorDecidir	:	parejas de estados que, aunque ya han sido probadas, la decisión de si acaso son distinguibles quedó relegada a una palabra aún por revisarse,

y procedemos como sigue:

1. Inicialmente, las primeras parejas distinguidas son las formadas por un estado final y otro no-final y la palabra vacía es la que los distingue. Las demás parejas quedan por ser probadas y no hay parejas con decisión pendiente. Sean pues,

$$\begin{aligned}
Distinguidos &= \{(p, q; nil) | p \in F, q \notin F\}\\
PorProbar &= \{\{p, q\} | (p \in F \& q \in F) \text{ o } (p \notin F \& q \notin F)\}\\
PorDecidir &= \emptyset
\end{aligned}$$

tran	0	1
q_0	q_1	q_2
q_1	q_3	q_4
q_2	q_5	q_6
q_3	q_7	q_8
q_4	q_9	q_{10}
q_5	q_{11}	q_8
q_6	q_{12}	q_2

tran	0	1
q_7	q_{13}	q_4
q_8	q_{14}	q_{15}
q_9	q_{16}	q_4
q_{10}	q_{17}	q_4
q_{11}	q_{18}	q_8
q_{12}	q_{19}	q_8
q_{13}	q_7	q_8

tran	0	1
q_{14}	q_{20}	q_8
q_{15}	q_{18}	q_8
q_{16}	q_{17}	q_4
q_{17}	q_{17}	q_4
q_{18}	q_{18}	q_8
q_{19}	q_{18}	q_8
q_{20}	q_{18}	q_8

Tabla 3.8: Semiautómata *SAF21*.

2. En tanto que haya parejas *PorProbar*: se toma una de estas parejas, digamos $\{p, q\}$. Se tiene dos posibilidades:

   (a) para cada símbolo e la pareja $\{p_e, q_e\} = \{tran(p, e), tran(p, e)\}$ no aparece en *Distinguidos*. Entonces $\{p, q\}$ se incorpora a *PorDecidir* y para cada e, $\{p, q\}$ relega su decisión a la pareja $\{p_e, q_e\}$, o bien,

   (b) existe un símbolo e tal que para $\{p_e, q_e\} = \{tran(p, e), tran(p, e)\}$, hay una terceta $\{p_e, q_e; \sigma\} \in$ *Distinguidos*. En tal caso, incorporamos $\{p, q; e\sigma\}$ a *Distinguidos*, y recursivamente todas las parejas que hayan relegado su decisión a $\{p, q\}$ se reconocen como distinguibles, y con las palabras que las distinguen se incorporan a *Distinguidos*,

3. al final, las parejas indistinguibles son las que quedan en la lista *PorDecidir*.

Ejemplo. Consideremos el autómata $AF21 = (SAF21, F)$ de 21 estados cuyo semiautómata adyacente posee la función de transición que aparece en la tabla (3.8). y cuyo conjunto de estados finales es

$$F = \{q_0, q_3, q_{12}, q_{13}, q_{18}, q_{19}\}.$$

Al proceder de acuerdo con el algoritmo bosquejado para el cálculo de parejas de estados indistinguibles se reconoce a las parejas de estados distinguibles, con respectivas palabras distinguientes, mostradas en las tablas (3.9, 3.10).

En cada renglón $i < 21$ de la tabla (3.9) aparecen parejas de la forma $(q_{i'}, q_i; nil)$, donde $i' < i$ y $(q_{i'}, q_i) \in [F \times (Q - F)] \cup [(Q - F) \times F]$. Por tanto, es la palabra vacía la que distinge a esas parejas de estados.

Ahora bien, en cada renglón $i < 21$ de la tabla (3.10) aparecen por lo general parejas de la forma $(q_{i'}, q_i; \sigma)$, donde $i' < i$ y σ distinge a esa pareja de estados. Cuando llega a aparecer en ese renglón una pareja de la forma $(q_{j'}, q_j; \sigma')$ con $j < i$, es porque en su momento, la pareja $(q_{j'}, q_j)$ relegó su decisión, respecto a indistinguibilidad, en una pareja de la forma $(q_{i'}, q_i)$. Por ejemplo en el renglón correspondiente a $i = 15$ aparece la pareja (q_8, q_{10}) con la palabra 10, porque $tran(q_8, 1) = q_{15}$, $tran(q_{10}, 1) = q_4$ y recién se ha descubierto que la palabra 0 distingue a la pareja (q_4, q_{15}). También aquí aparece la pareja (q_4, q_{14}) con la palabra 110, porque $tran(q_4, 1) = q_{10}$, $tran(q_{14}, 1) = q_8$ y se acaba de descubrir que la palabra 10 distingue a la pareja (q_8, q_{10}).

Las parejas de estados indistinguibles quedan enlistadas en la tabla (3.11). Estas determinan una relación de equivalencia, cuyas clases aparecen, renombradas, en la tabla (3.12-a).

Al considerar en ese espacio cociente la esctructura de semiautómata inducida por el semiautómata original se obtiene la función de transición de la tabla (3.12-b).

Finalmente, ha de marcarse como estados finales a las clases p_0 y p_6, consistentes de estados finales en *AF21*, para obtener un autómata equivalente a *AF21*, que de acuerdo con el corolario 3.2.2 es el mínimo entre los equivalentes a *AF21*.

$(q_0, q_1; nil)$,
$(q_0, q_2; nil)$,
$(q_1, q_3; nil), (q_2, q_3; nil)$,
$(q_0, q_4; nil), (q_3, q_4; nil)$,
$(q_0, q_5; nil), (q_3, q_5; nil)$,
$(q_0, q_6; nil), (q_3, q_6; nil)$,
$(q_0, q_7; nil), (q_3, q_7; nil)$,
$(q_0, q_8; nil), (q_3, q_8; nil)$,
$(q_0, q_9; nil), (q_3, q_9; nil)$,
$(q_0, q_{10}; nil), (q_3, q_{10}; nil)$,
$(q_0, q_{11}; nil), (q_3, q_{11}; nil)$,
$(q_1, q_{12}; nil), (q_2, q_{12}; nil), (q_4, q_{12}; nil), (q_5, q_{12}; nil), (q_6, q_{12}; nil),$ $(q_7, q_{12}; nil), (q_8, q_{12}; nil), (q_9, q_{12}; nil), (q_{10}, q_{12}; nil), (q_{11}, q_{12}; nil),$
$(q_1, q_{13}; nil), (q_2, q_{13}; nil), (q_4, q_{13}; nil), (q_5, q_{13}; nil), (q_6, q_{13}; nil),$ $(q_7, q_{13}; nil), (q_8, q_{13}; nil), (q_9, q_{13}; nil), (q_{10}, q_{13}; nil), (q_{11}, q_{13}; nil),$
$(q_0, q_{14}; nil), (q_3, q_{14}; nil), (q_{12}, q_{14}; nil), (q_{13}, q_{14}; nil)$,
$(q_0, q_{15}; nil), (q_3, q_{15}; nil), (q_{12}, q_{15}; nil), (q_{13}, q_{15}; nil)$,
$(q_0, q_{16}; nil), (q_3, q_{16}; nil), (q_{12}, q_{16}; nil), (q_{13}, q_{16}; nil)$,
$(q_0, q_{17}; nil), (q_3, q_{17}; nil), (q_{12}, q_{17}; nil), (q_{13}, q_{17}; nil)$,
$(q_1, q_{18}; nil), (q_2, q_{18}; nil), (q_4, q_{18}; nil), (q_5, q_{18}; nil), (q_6, q_{18}; nil), (q_7, q_{18}; nil), (q_8, q_{18}; nil),$ $(q_9, q_{18}; nil), (q_{10}, q_{18}; nil), (q_{11}, q_{18}; nil), (q_{14}, q_{18}; nil), (q_{15}, q_{18}; nil), (q_{16}, q_{18}; nil), (q_{17}, q_{18}; nil),$
$(q_1, q_{19}; nil), (q_2, q_{19}; nil), (q_4, q_{19}; nil), (q_5, q_{19}; nil), (q_6, q_{19}; nil), (q_7, q_{19}; nil), (q_8, q_{19}; nil),$ $(q_9, q_{19}; nil), (q_{10}, q_{19}; nil), (q_{11}, q_{19}; nil), (q_{14}, q_{19}; nil), (q_{15}, q_{19}; nil), (q_{16}, q_{19}; nil), (q_{17}, q_{19}; nil),$
$(q_0, q_{20}; nil), (q_3, q_{20}; nil), (q_{12}, q_{20}; nil), (q_{13}, q_{20}; nil), (q_{18}, q_{20}; nil), (q_{19}, q_{20}; nil)$,

Tabla 3.9: Primer grupo de parejas de estados distinguibles en el autómata *AF21*.

$(q_1, q_2; 0)$,
$(q_1, q_4; 0)$,
$(q_1, q_5; 0)$,
$(q_2, q_6; 0), (q_4, q_6; 0), (q_5, q_6; 0)$,
$(q_2, q_7; 0), (q_4, q_7; 0), (q_5, q_7; 0)$,
$(q_1, q_8; 0), (q_6, q_8; 0), (q_2, q_5; 10)$,
$(q_7, q_8; 0)$,
$(q_1, q_9; 0), (q_2, q_9; 10), (q_6, q_9; 0), (q_7, q_9; 0)$,
$(q_1, q_{10}; 0), (q_2, q_{10}; 10), (q_6, q_{10}; 0), (q_2, q_4; 10), (q_1, q_6; 110), (q_6, q_7; 110), (q_7, q_{10}; 0)$,
$(q_2, q_{11}; 0), (q_4, q_{11}; 0), (q_5, q_{11}; 0), (q_8, q_{11}; 0), (q_9, q_{11}; 0), (q_4, q_5; 00), (q_{10}, q_{11}; 0)$,
$(q_0, q_{12}; 0), (q_3, q_{12}; 0)$,
$(q_{12}, q_{13}; 0)$,
$(q_1, q_{14}; 0), (q_2, q_{14}; 10), (q_6, q_{14}; 0), (q_7, q_{14}; 0), (q_{11}, q_{14}; 0), (q_5, q_8; 00)$,
$(q_2, q_{15}; 0), (q_4, q_{15}; 0), (q_8, q_9; 10), (q_8, q_{10}; 10), (q_4, q_{14}; 110)$, $(q_5, q_{15}; 0), (q_8, q_{15}; 0), (q_8, q_{14}; 10), (q_9, q_{15}; 0), (q_{10}, q_{15}; 0)$, $(q_4, q_8; 10), (q_5, q_9; 110), (q_5, q_{10}; 110), (q_1, q_{11}; 110), (q_7, q_{11}; 110)$, $(q_9, q_{14}; 110), (q_{10}, q_{14}; 110), (q_1, q_{15}; 110), (q_7, q_{15}; 110), (q_{14}, q_{15}; 0)$,
$(q_1, q_{16}; 0), (q_2, q_{16}; 10), (q_5, q_{16}; 110), (q_6, q_{16}; 0)$, $(q_7, q_{16}; 0), (q_8, q_{16}; 10), (q_{11}, q_{16}; 0), (q_{14}, q_{16}; 110), (q_{15}, q_{16}; 0)$,
$(q_1, q_{17}; 0), (q_2, q_{17}; 10), (q_5, q_{17}; 110), (q_6, q_{17}; 0)$, $(q_7, q_{17}; 0), (q_8, q_{17}; 10), (q_{11}, q_{17}; 0), (q_{14}, q_{17}; 110), (q_{15}, q_{17}; 0)$,
$(q_0, q_{18}; 0), (q_3, q_{18}; 0), (q_{13}, q_{18}; 0)$,
$(q_0, q_{19}; 0), (q_3, q_{19}; 0), (q_{13}, q_{19}; 0)$,
$(q_1, q_{20}; 00), (q_2, q_{20}; 0), (q_4, q_{20}; 0), (q_5, q_{20}; 0), (q_7, q_{20}; 00), (q_8, q_{20}; 0)$, $(q_9, q_{20}; 0), (q_{10}, q_{20}; 0), (q_{14}, q_{20}; 0), (q_{16}, q_{20}; 0), (q_{17}, q_{20}; 0)$

Tabla 3.10: Segundo grupo de parejas de estados distinguibles en el autómata *AF21*.

$(q_0, q_0), (q_0, q_3), (q_0, q_{13})$,
$(q_1, q_1), (q_1, q_7)$,
$(q_2, q_2), (q_2, q_8)$,
$(q_3, q_3), (q_3, q_{13})$,
$(q_4, q_4), (q_4, q_9), (q_4, q_{10}), (q_4, q_{16}), (q_4, q_{17})$,
$(q_5, q_5), (q_5, q_{14})$,
$(q_6, q_6), (q_6, q_{11}), (q_6, q_{15}), (q_6, q_{20})$,
(q_7, q_7),
(q_8, q_8),
$(q_9, q_9), (q_9, q_{10}), (q_9, q_{16}), (q_9, q_{17})$,
$(q_{10}, q_{10}), (q_{10}, q_{16}), (q_{10}, q_{17})$,
$(q_{11}, q_{11}), (q_{11}, q_{15}), (q_{11}, q_{20})$,
$(q_{12}, q_{12}), (q_{12}, q_{18}), (q_{12}, q_{19})$,
(q_{13}, q_{13}),
(q_{14}, q_{14}),
$(q_{15}, q_{15}), (q_{15}, q_{20})$,
$(q_{16}, q_{16}), (q_{16}, q_{17})$,
(q_{17}, q_{17}),
$(q_{18}, q_{18}), (q_{18}, q_{19})$,
(q_{19}, q_{19}),
(q_{20}, q_{20})

Tabla 3.11: Parejas de estados indistinguibles en el autómata *AF21*.

$p_0:$	$\{q_0, q_3, q_{13}\}$,
$p_1:$	$\{q_1, q_7\}$,
$p_2:$	$\{q_2, q_8\}$,
$p_3:$	$\{q_4, q_9, q_{10}, q_{16}, q_{17}\}$,
$p_4:$	$\{q_5, q_{14}\}$,
$p_5:$	$\{q_6, q_{11}, q_{15}, q_{20}\}$,
$p_6:$	$\{q_{12}, q_{18}, q_{19}\}$

(a)

tran	0	1
p_0	p_1	p_2
p_1	p_0	p_3
p_2	p_4	p_5
p_3	p_3	p_3
p_4	p_5	p_2
p_5	p_6	p_2
p_6	p_6	p_2

(b)

Tabla 3.12: Autómata cociente bajo la noción de indistinguibilidad en el autómata *AF21*.

3.3 Autómatas no-deterministas

3.3.1 Nociones básicas

Los autómatas no-deterministas se conforman como los autómatas finitos ya vistos, salvo que sus transiciones, en lugar de ser funciones, son relaciones que a cada pareja (estado,estímulo) le asocian varios, uno o ningún estado. Más precisamente:

Un *semiautómata no-determinista* es una estructura de la forma $SAFND = (Q, Ent, tran, q_0)$ donde

$$\begin{array}{rcl} Q & : & \text{es el conjunto de } \textit{estados}, \\ Ent & : & \text{es el alfabeto de } \textit{entrada}, \\ tran & \subset & [Q \times Ent] \times Q, \quad \text{es una relación de } \textit{transición}, \\ q_0 \in Q & : & \text{es el estado } \textit{inicial}. \end{array}$$

Un *autómata no-determinista* es una pareja $AFND = (SAFND, F)$ donde $SAFND$ es un semiautómata no-determinista y $F \subset Q$ es un conjunto de estados *finales*.

Si $(q, e, p) \in tran$ decimos que *se puede transitar* a p desde el estado q cuando arriba un símbolo e. Para cada pareja $(q, e) \in Q \times Ent$ su *imagen* bajo la transición es el conjunto $tran(q, e) = \{p \in Q | (q, e, p) \in tran\}$, es decir, es el conjunto de estados a los que se puede transitar desde q con e. De manera reiterada, para $(q, \sigma) \in Q \times Ent^*$, definimos la *imagen* $tran^*(q, \sigma)$ como sigue:

$$\begin{array}{rcl} & tran^*(q, nil) & = \{q\} \\ \sigma \in Ent^*, e \in Ent \quad \Rightarrow & tran^*(q, \sigma e) & = \displaystyle\bigcup_{p \in tran^*(q,\sigma)} tran(p, e) = \{o \in Q | \exists p \in tran^*(q, \sigma) : (p, e, o) \in tran\} \end{array}$$

Para cada $\sigma \in Ent^*$ definimos $T(\sigma) = tran^*(q_0, \sigma)$.

Una palabra $\sigma \in Ent^*$ es *reconocida* por el autómata $AFND$ si algún estado en $T(\sigma)$ es final. El *lenguaje* del autómata $AFND$ consiste de todas las palabras que reconoce, $L(AFND) = \{\sigma \in Ent^* | T(\sigma) \cap F \neq \emptyset\}$.

Ejemplo. Sea $AFND = (Q, Ent, tran, q_0, F)$ el autómata no-determinista tal que

$$\begin{array}{rcl} Q & = & \{q_0, q_1, q_2, q_3, q_4\} \\ Ent & = & \{0, 1\} \\ tran & = & \left\{ \begin{array}{ccccc} (q_0\ 0\ q_1) & (q_1\ 0\ q_0) & (q_2\ 0\ q_1) & (q_2\ 0\ q_2) & (q_3\ 0\ q_2) \\ (q_0\ 1\ q_3) & (q_2\ 1\ q_3) & (q_3\ 1\ q_1) & (q_3\ 1\ q_2) & (q_3\ 1\ q_4) \end{array} \right\} \\ q_0 & : & \text{estado inicial}, \\ F & = & \{q_4\} \end{array}$$

En la siguiente tabla presentamos el cálculo de la correspondiente función T en algunas palabras:

σ	$T(\sigma)$
nil	$\{q_0\}$
1	$\{q_3\}$
10	$\{q_2\}$
100	$\{q_2, q_1\}$
1000	$\{q_2, q_1, q_0\}$
10001	$\{q_3\}$
100011	$\{q_1, q_2, q_4\}$

Así pues, $T(100011) \cap F = \{q_4\}$ y consecuentemente $100011 \in L(AFND)$.

Observación 3.3.1 *Todo autómata finito (determinista) es también un autómata finito no-determinista.*

En efecto, las funciones son casos particulares de relaciones. Por tanto, toda función de transición, es una relación de transición.

Representación de transiciones mediante matrices booleanas

Sea $Dos = \{0, 1\}$ el álgebra booleana de dos elementos, dotada de sus operaciones usuales de *conjunción*, "$\wedge$" y *disyunción*, "$\vee$": $x \wedge y$ es 1 sólo si ambos x e y son 1; $x \vee y$ es 0 sólo si ambos x e y son 0.

Para cada símbolo de entrada $e \in Ent$ definamos la matriz $M^{AFND}(e) = (m_{qp})_{q,p \in Q} \in Dos^{Q \times Q}$ tal que para todos $q, p \in Q$:

$$m_{qp} = \begin{cases} 1 & \text{si } (q\ e\ p) \in tran, \\ 0 & \text{si } (q\ e\ p) \notin tran. \end{cases}$$

Similarmente, para $\sigma \in Ent^*$ definamos la matriz $M^{AFND}(\sigma) = (m_{qp})_{q,p \in Q} \in Dos^{Q \times Q}$ tal que para todos $q, p \in Q$:

$$m_{qp} = \begin{cases} 1 & \text{si } p \in tran^*(q, \sigma), \\ 0 & \text{si } p \notin tran^*(q, \sigma). \end{cases}$$

Así pues, $\forall \sigma \in Ent^*$ se tiene la relación,

$$\forall q \in Q : \ tran^*(q, \sigma) = \{p \in Q | \left(M^{AFND}(\sigma)\right)_{qp} = 1\}.$$

Ahora bien, la colección $Dos^{Q \times Q}$ de matrices booleanas con índices en Q tiene una estructura de anillo con la operación suma dada por la disyunción entrada a entrada,

$$A + B = C \ \Leftrightarrow \ \forall q, p \in Q : c_{qp} = a_{qp} \vee b_{qp},$$

y el producto booleano de matrices,

$$AB = C \ \Leftrightarrow \ \forall q, p \in Q : c_{qp} = \bigvee_{o \in Q} (a_{qo} \wedge b_{op}).$$

Lema 3.3.1 *Si $\sigma = \sigma_1 \sigma_2$ entonces $M^{AFND}(\sigma) = M^{AFND}(\sigma_1) M^{AFND}(\sigma_2)$.*

En particular, si $\sigma = e_{i_0} \cdots e_{i_{k-1}}$ entonces $M^{AFND}(\sigma) = \prod_{j=0}^{k-1} M^{AFND}(e_{i_j})$.

Ejemplo. Para el *AFND* del ejemplo anterior tenemos

$$M^{AFND}(0) = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad , \quad M^{AFND}(1) = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

σ	$\tau \equiv_{AFND}$	σ	$\tau \equiv_{AFND}$	σ	$\tau \equiv_{AFND}$
		nil	–		
0	–	0000	–	00100	–
1	–	0001	01	00101	001
00	–	0010	–	00110	1000
01	–	0011	–	00111	001
10	–	0100	–	01000	0110
11	–	0101	01	01001	01
000	–	0110	–	01100	0110
001	–	0111	01	01101	01
010	–	1000	–	01000	1000
011	–	1001	001	01001	001
100	–	1100	–	11000	1000
101	001	1101	001	11001	001
110	–	00000	000	001000	1000
111	1	00001	001	001001	001

Tabla 3.13: Cálculo del monoide del autómata no-determinista.

3.3.2 Monoides de autómas no-deterministas

El *monoide* de un autómata no-determinista se construye de manera similar a como se hizo en el caso determinista: Dos palabras σ_1, σ_2 son *equivalentes*, $\sigma_1 \equiv_{AFND} \sigma_2$, si $M^{AFND}(\sigma_1) = M^{AFND}(\sigma_2)$, es decir, si ambas definen a la misma relación entre estados.

Esta relación[1], además de ser de equivalencia, es congruente con la concatenación. Por tanto, el cociente $S_{AFND} = (Ent/\equiv_{AFND})$ es un monoide, dicho *del autómata AFND*.

S_{AFND} se construye también siguiendo el algoritmo (3.6).

Ejemplo. Aplicando el algoritmo (3.6) al ejemplo anterior, se obtiene las palabras mostradas en la tabla (3.13). Se ve que exactamente 21 clases de equivalencia conforman el monoide del autómata. En la tabla (3.14) se muestra cada una de las 21 matrices correspondientes. Ahí, observamos que las palabras 11 y 0011 son reconocidas por el autómata (sus entradas $(M)_{04}$, correspondientes a un arribo al estado final q_4 a partir del inicial q_0, asumen el valor 1). Por tanto, cualquier palabra equivalente a una de ellas dos también ha de ser reconocida. Se tiene pues que el lenguaje reconocido por el autómata no-determinista es precisamente la unión de las dos clases de equivalencia [11] y [0011].

Por otro lado, el monoide del autómata no-determinista puede ser dotado, como se hizo anteriormente, de una estructura de autómata finito. Si aquí se declara como finales a las clases [11] y [0011] entonces el autómata resultante será uno finito que reconoce al mismo lenguaje que el autómata no-determinista. Esta propiedad de ser equivalente a uno finito no es exclusiva del autómata en este ejemplo según veremos en el lema (3.3.2) más abajo.

3.3.3 Indeterminismo y determinismo

Diremos que un lenguaje $L \subset Ent^*$ es *regular-N* si coincide con el lenguaje reconocido por algún autómata no-determinista.

Ya que todo autómata finito es en sí mismo un autómata no-determinista se tiene que todo lenguaje regular es también un lenguaje regular-N. El recíproco también es cierto.

Lema 3.3.2 (Equivalencia de determinismo e indeterminismo) *Todo lenguaje regular-N es regular. Es decir, para todo autómata no-determinista AFND existe un autómata finito AF tal que* $L(AFND) = L(AF)$.

[1] reconocemos aquí lo desafortunado del lenguaje: *esta relación* se refiere a la definida entre palabras, la cual relaciona a palabras que definen iguales *relaciones* entre estados.

$M(nil) = \begin{pmatrix} 1&0&0&0&0\\ 0&1&0&0&0\\ 0&0&1&0&0\\ 0&0&0&1&0\\ 0&0&0&0&1 \end{pmatrix}$	$M(0) = \begin{pmatrix} 0&1&0&0&0\\ 1&0&0&0&0\\ 0&1&1&0&0\\ 0&0&1&0&0\\ 0&0&0&0&0 \end{pmatrix}$	$M(1) = \begin{pmatrix} 0&0&0&1&0\\ 0&0&0&0&0\\ 0&0&0&1&0\\ 0&1&1&0&1\\ 0&0&0&0&0 \end{pmatrix}$
$M(00) = \begin{pmatrix} 1&0&0&0&0\\ 0&1&0&0&0\\ 1&1&1&0&0\\ 0&1&1&0&0\\ 0&0&0&0&0 \end{pmatrix}$	$M(01) = \begin{pmatrix} 0&0&0&0&0\\ 0&0&0&1&0\\ 0&0&0&1&0\\ 0&0&0&1&0\\ 0&0&0&0&0 \end{pmatrix}$	$M(10) = \begin{pmatrix} 0&0&1&0&0\\ 0&0&0&0&0\\ 0&0&1&0&0\\ 1&1&1&0&0\\ 0&0&0&0&0 \end{pmatrix}$
$M(11) = \begin{pmatrix} 0&1&1&0&1\\ 0&0&0&0&0\\ 0&1&1&0&1\\ 0&0&0&1&0\\ 0&0&0&0&0 \end{pmatrix}$	$M(000) = \begin{pmatrix} 0&1&0&0&0\\ 1&0&0&0&0\\ 1&1&1&0&0\\ 1&1&1&0&0\\ 0&0&0&0&0 \end{pmatrix}$	$M(001) = \begin{pmatrix} 0&0&0&1&0\\ 0&0&0&0&0\\ 0&0&0&1&0\\ 0&0&0&1&0\\ 0&0&0&0&0 \end{pmatrix}$
$M(010) = \begin{pmatrix} 0&0&0&0&0\\ 0&0&1&0&0\\ 0&0&1&0&0\\ 0&0&1&0&0\\ 0&0&0&0&0 \end{pmatrix}$	$M(011) = \begin{pmatrix} 0&0&0&0&0\\ 0&1&1&0&1\\ 0&1&1&0&1\\ 0&1&1&0&1\\ 0&0&0&0&0 \end{pmatrix}$	$M(100) = \begin{pmatrix} 0&1&1&0&0\\ 0&0&0&0&0\\ 0&1&1&0&0\\ 1&1&1&0&0\\ 0&0&0&0&0 \end{pmatrix}$
$M(110) = \begin{pmatrix} 1&1&1&0&0\\ 0&0&0&0&0\\ 1&1&1&0&0\\ 0&0&1&0&0\\ 0&0&0&0&0 \end{pmatrix}$	$M(0000) = \begin{pmatrix} 1&0&0&0&0\\ 0&1&0&0&0\\ 1&1&1&0&0\\ 1&1&1&0&0\\ 0&0&0&0&0 \end{pmatrix}$	$M(0010) = \begin{pmatrix} 0&0&1&0&0\\ 0&0&0&0&0\\ 0&0&1&0&0\\ 0&0&1&0&0\\ 0&0&0&0&0 \end{pmatrix}$
$M(0011) = \begin{pmatrix} 0&1&1&0&1\\ 0&0&0&0&0\\ 0&1&1&0&1\\ 0&1&1&0&1\\ 0&0&0&0&0 \end{pmatrix}$	$M(0100) = \begin{pmatrix} 0&0&0&0&0\\ 0&1&1&0&0\\ 0&1&1&0&0\\ 0&1&1&0&0\\ 0&0&0&0&0 \end{pmatrix}$	$M(0110) = \begin{pmatrix} 0&0&0&0&0\\ 1&1&1&0&0\\ 1&1&1&0&0\\ 1&1&1&0&0\\ 0&0&0&0&0 \end{pmatrix}$
$M(1000) = \begin{pmatrix} 1&1&1&0&0\\ 0&0&0&0&0\\ 1&1&1&0&0\\ 1&1&1&0&0\\ 0&0&0&0&0 \end{pmatrix}$	$M(1100) = \begin{pmatrix} 1&1&1&0&0\\ 0&0&0&0&0\\ 1&1&1&0&0\\ 0&1&1&0&0\\ 0&0&0&0&0 \end{pmatrix}$	$M(00100) = \begin{pmatrix} 0&1&1&0&0\\ 0&0&0&0&0\\ 0&1&1&0&0\\ 0&1&1&0&0\\ 0&0&0&0&0 \end{pmatrix}$

Tabla 3.14: Matrices correspondientes a los elementos del monoide del autómata no-determinista.

En efecto, sea $AFND = (Q_1, Ent, tran_1, q_{01}, F_1)$ un autómata no-determinista. Podemos presentar dos construcciones de autómatas finitos equivalentes a $AFND$.

Primera construcción. Construyamos el monoide S_{AFND} del autómata no-determinista $AFND$ y consideremos su estructura de autómata finito: cada uno de sus elementos $[\sigma]$ es un estado, para cada símbolo $e \in Ent$ definamos la transición $([\sigma], e) \mapsto [\sigma e]$ y definamos como estados finales a las clases de equivalencia $[\sigma]$ tales que $[\sigma] \cap F \neq \emptyset$. Una palabra será reconocida en este último autómata cuando y sólo cuando lo sea por $AFND$.

Segunda construcción. Construyamos el autómata finito $AF = (Q_2, Ent, tran_2, q_{02}, F_2)$ como sigue:

estados: Todo subconjunto de estados "viejos" será un "nuevo" estado, $Q_2 = \mathcal{P}(Q_1) = \{Q | Q \subset Q_1\}$.

transición: Todo subconjunto de estados "viejos" se transforma en su imagen bajo la función de transición "vieja", $tran_2 : Q_2 \times Ent \to Q_2, (Q, e) \mapsto tran_2(Q, e) = \bigcup_{q \in Q} tran(q, e)$, es decir, para cada $p \in Q_1$, $p \in tran_2(Q, e)$ si y sólo si $\exists q \in Q : (q, e, p) \in tran_1$.

estado inicial: Hagamos $q_{02} = \{q_{01}\}$, la mónada que consta sólo del estado inicial "viejo".

estados finales: Todo subconjunto de estados "viejos" que contenga alguno final de ésos será un nuevo estado final: $Q' \in F_2 \Leftrightarrow Q' \cap F_1 \neq \emptyset$.

Observamos que rige cada una de las siguientes equivalencias para cualquier palabra $\sigma \in Ent$:

$$\begin{aligned} \sigma \in L(AFND) &\Leftrightarrow tran^*_{AFND}(q_{01}, \sigma) \cap F_1 \neq \emptyset \\ &\Leftrightarrow tran^*_{AF}(\{q_{01}\}, \sigma) \cap F_1 \neq \emptyset \\ &\Leftrightarrow tran^*_{AF}(q_{02}, \sigma) \in F_2 \\ &\Leftrightarrow \sigma \in L(AF) \end{aligned}$$

así pues, $AFND$ y AF son equivalentes. Observemos también aquí que el nuevo conjunto de estados ha de tener 2^n elementos, donde n es el número de estados "viejos". Esto hace crecer mucho el tamaño del autómata finito equivalente construído de esta forma. Bien que en algunos casos tal cota superior al número de estados nuevos puede alcanzarse, en muchos otros casos la parte accesible del autómata construído incluirá sólo una cantidad mucho menor de estados. Por tanto, en la práctica es muy conveniente construir tan solo la

Q	0	1	Q	0	1	Q	0	1	Q	0	1
0	0	0	8	16	0	16	8	2	24	24	2
1	0	0	9	16	0	17	8	2	25	24	2
2	4	13	10	20	13	18	12	15	26	28	15
3	4	13	11	20	13	19	12	15	27	28	15
4	12	2	12	28	2	20	12	2	28	28	2
5	12	2	13	28	2	21	12	2	29	28	2
6	12	15	14	28	15	22	12	15	30	28	15
7	12	15	15	28	15	23	12	15	31	28	15

Tabla 3.15: Transición en el autómata finito equivalente al no-determinista.

parte accesible del autómata AF siguiendo la estrategia del algoritmo (3.5) de cálculo de estados accesibles.

Ejemplo. Consideremos el mismo ejemplo tratado en esta sección. Cada subconjunto Q del conjunto de estados $Q_1 = \{q_0, q_1, q_2, q_3, q_4\}$ puede ser codificado por una cadena de 5 caracteres $c_Q = c_{0Q}c_{1Q}c_{2Q}c_{3Q}c_{4Q} \in \{0,1\}^5$ de manera evidente,

$$\forall i \leq 5 : c_{iQ} = \begin{cases} 1 & \text{si } q_i \in Q, \\ 0 & \text{si } q_i \notin Q. \end{cases}$$

y cada una de tales cadenas puede ser vista como la representación binaria de un número entero entre 0 y 31. Nombremos pues con números de 0 a 31 a los elementos del conjunto Q_2 de nuevos estados. Así por ejemplo "7" que en binario es 00111 representa al conjunto $\{q_2, q_3, q_4\}$ y "16", $16 = (10000)_2$, es el nuevo estado inicial $q_{02} = \{q_0\}$. Los nuevos estados finales son todos aquellos que contegan a q_4, es decir, que tengan el último bit "prendido". Los nuevos estados finales son entonces todos los números impares.

Con ayuda de la tabla (3.14), se ve que la función de transición del nuevo autómata es la mostrada en la tabla (3.15). Observamos en este ejemplo que hay muchos estados inaccesibles tan sólo por el hecho de que la imagen de la función de transición no incluye a todos los estados. Con el estímulo 0 sólo se puede arribar a los estados 0, 4, 8, 12, 16, 20, 24 y 28. Con el estímulo 1 sólo se puede arribar a los estados 0, 2, 13 y 15. Si se aplica el algoritmo (3.5) se obtiene el autómata de 8 estados cuya tabla de transición es la siguiente:

Q	0	1	Q	0	1	Q	0	1	Q	0	1
0	0	0	4	12	2	12	28	2	16	8	2
2	4	13	8	16	0	13	28	2	28	28	2

en el que "16" es el estado inicial y "13" es el único estado final.

3.4 Gráficas de transición

3.4.1 Nociones básicas

Sea *Ent* un alfabeto finito y sea λ un símbolo que no esté en *Ent*. Sea $Ent_\lambda = \{\lambda\} \cup Ent$ la extensión, mediante la añadidura del símbolo λ, del alfabeto dado. Naturalmente, el diccionario Ent^* está incluido en el diccionario Ent^*_λ. Por otro lado, sea

$$\begin{array}{rcl} \pi : Ent_\lambda & \to & Ent \\ nil & \mapsto & \pi(nil) = nil \\ \sigma_1 s = \sigma & \mapsto & \pi(\sigma) = \begin{cases} \pi(\sigma_1)s & \text{si } s \in Ent, \\ \pi(\sigma_1) & \text{si } s = \lambda. \end{cases} \end{array}$$

la transformación que en cada palabra de Ent_λ^* suprime todas las apariciones del símbolo λ. Acaso vale la pena decir aquí que π es el homomorfismo entre diccionarios que extiende a las sustitución

$$f_\pi : \xi \mapsto \begin{cases} nil & \text{si } \xi = \lambda, \\ \xi & \text{si } \xi \in Ent. \end{cases}$$

Una *gráfica de transición* sobre el alfabeto Ent es una estructura $GT = (Q, Ent, tran, q_0, F)$ que coincide con un autómata no-determinista $AFND = (Q, Ent_\lambda, tran, q_0, F)$ sobre el alfabeto Ent_λ. Se dice que todas las transiciones de la forma (q, λ, p) son *transiciones-λ*, o *transiciones-vacías*.

Para cualquier estado $q \in Q$ y cualquier palabra $\sigma \in Ent^*$ el conjunto de estados accesibles por σ desde q en GT es

$$tran^*_{GT}(q, \sigma) = \bigcup_{\sigma_1 \in Ent^*_\lambda, \pi(\sigma_1) = \sigma} tran^*_{AFND}(q, \sigma_1),$$

es decir, un estado es accesible si hay una sucesión de transiciones, que puede incluir transiciones-λ, correspondiente a los símbolos de σ desde el estado q hasta ese estado.

Introduzcamos también en este caso, la siguiente relación en Ent^*:

$$\sigma_1 \equiv_{GT} \sigma_2 \quad \Leftrightarrow \quad \forall q \in Q : tran^*_{GT}(q, \sigma_1) = tran^*_{GT}(q, \sigma_2)$$

Como antes, se tiene que $\equiv_{GT}$ es una relación de equivalencia congruente con la concatenación. Por tanto, el cociente $S_{GT} = (Ent^* / \equiv_{GT})$ es un monoide, dicho de la gráfica de transición.

3.4.2 Supresión de transiciones vacías

Sea también $T_{GT}(\sigma) = tran^*_{GT}(q_0, \sigma)$ el conjunto de estados accesibles en la gráfica de transición mediante σ partiendo del estado inicial.

Una palabra $\sigma \in Ent^*$ es *reconocida* por la gráfica de transición GT si para alguna palabra $\sigma_1 \in Ent^*_\lambda$ tal que $\sigma = \pi(\sigma_1)$, se tiene que σ_1 es reconocida por el autómata no-determinista $AFND$. El *lenguaje* de la gráfica está formado por todas las palabras que reconoce.

Observación 3.4.1 *Las siguientes dos relaciones se cumplen:*

- $L(GT) = \pi(L(AFND))$.
- $\forall \sigma \in Ent^* : \sigma \in L(GT) \Leftrightarrow T_{GT}(\sigma) \cap F \neq \emptyset$.

Un lenguaje $L \subset Ent^*$ se dice *regular-G* si existe una gráfica de transición GT sobre el alfabeto Ent tal que $L = L(GT)$.

Todo autómata no-determinista sin transiciones-λ es una gráfica de transición. Por tanto, todo lenguaje regular-N es también regular-G.

En cuanto al recíproco, se puede ver inmediatamente que todo lenguaje regular-G en el diccionario Ent^* es regular-N en el diccionario Ent^*_λ y, debido al lema 3.3.2, es de hecho regular en ese mismo diccionario. Así pues, toda gráfica de transición es equivalente a un autómata finito con transiciones-λ. Sin embargo, con esta construcción se ha aumentado un símbolo al alfabeto. Veamos que se puede eliminar de manera equivalente a las transiciones-λ.

Lema 3.4.1 (Equivalencia de GT's y AF's) *Todo lenguaje regular-G es regular. Es decir, para cada gráfica de transición GT sobre el alfabeto Ent existe un autómata finito AF sobre el alfabeto Ent tal que $L(GT) = L(AF)$.*

En efecto, para construir un autómata finito equivalente a una gráfica de transición dada, se puede seguir cualquiera de las dos construcciones presentadas en el lema 3.3.2 partiendo del correspondiente autómata no-determinista, sólo que en vez de considerar las relaciones $tran^*_{AFND}(q, \sigma)$ se considerará las relaciones $tran^*_{GT}(q, \sigma)$. También, para evitar considerar estados irrelevantes se puede "filtrar" la segunda construcción con el algoritmo (3.5) de cálculo de partes accesibles.

Input: A transition graph $GT = (Q, Ent, tran, q_0, F)$
Output: A finite automaton $AF = (Q_1, Ent, tran_1, q_{01}, F_1)$ such that

$$L(GT) = L(AF).$$

Tst: List of *tested* new states.
TBTst: List of *to-be-tested* new states.

{ $Tst := nil$; $TBTst := [\mathbf{q}_0 = 10\cdots 0]$; $tran_1 = nil$;
while $TBTst \neq nil$ **do**
{ $Pop(\mathbf{q}, TBTst)$;
for each $e \in Ent$ **do**
{ $\mathbf{p} := \bigcup_{q \in \mathbf{q}} tran_{GT}(q, e)$; $tran_1 := Append(tran_1, (\mathbf{q}, e, \mathbf{p}))$;
if $\mathbf{p} \notin Tst \cup TBTst \cup \{\mathbf{q}\}$ **then** $TBTst := Append(TBTst, [\mathbf{p}])$ } ;
$Tst := Append(Tst, [\mathbf{q}])$
} ;
AF consists of the states in Tst
}

Figura 3.7: Conversión de una gráfica de transición a un autómata finito equivalente.

Presentamos un seudocódigo de este procedimiento en la figura (3.7). Si $Q = \{q_0, \ldots, q_{n-1}\}$ es el conjunto de estados de la gráfica, la lista $\mathbf{q} = i_0 \cdots i_{n-1} \in Dos^n$ denota al subconjunto Q' tal que para todo j, $q_{i_j} \in Q' \Leftrightarrow i_j = 1$. El nuevo estado final es la mónada que consiste del estado inicial q_0, $q_{01} = 10\cdots 0 = \mathbf{q}_0$.
Los estados finales serán todos aquellos nuevos estados que incluyan alguno final de los viejos estados.

Ejemplo. Sea $GT = (Q, Ent, tran, q_0, F)$ la gráfica de transición definida como sigue:

estados: $Q = \{0, 1, 2, 3\}$,

entradas: $Ent = \{0, 1\}$,

estado inicial: $q_0 = 0$,

estados finales: $F = \{0\}$,

transición: $tran = \left\{ \begin{array}{lll} (0\ \lambda\ 1) & (2\ 0\ 1) & (2\ 1\ 2) \\ (1\ \lambda\ 2) & (1\ 0\ 3) & (3\ 1\ 3) \\ (3\ \lambda\ 0) & & \end{array} \right\}$.

La palabra $\lambda\lambda 1001\lambda$ realiza la transición 01221330 y consecuentemente la palabra 1001 es reconocida. Una palabra que consista sólo de 1's no puede ser reconocida.

Se puede ver que "partir del estado 0" es como si se "partiera del estado 1 o del estado 2" pues a ambos se llega desde el estado 0 con palabras de la forma λ^*. Por la misma razón, si se arriba al estado 0 se arriba también a los estados 1 y 2, si se arriba al estado 1 se arriba también al estado 2 y si se arriba al estado 3 se arriba también a los estados 0, 1, 2.

Al aplicar el algoritmo (3.7) obtenemos la tabla siguiente:

q	0	1
1000	1111	0010
1111	1111	1111
0010	0100	0010
0100	1111	0100

El estado inicial del autómata equivalente consiste de la mónada formada por el estado 0, es pues 1000, y los estados finales son los que tienen prendido el "bit" de la extrema izquierda, a saber 1000 y 1111.

3.5 Autómatas bidireccionales

Hasta ahora los autómatas que hemos considerado leen su entrada de izquierda a derecha y no reculan sobre ella para realizar transiciones alternativas en función de la "historia" de la entrada. Los autómatas bidireccionales en cambio, sí pueden revisar la cadena de entrada y efectuar transiciones en función de una misma entrada leída varias veces. Veamos una formalización de esta noción de autómatas finitos.

Un *autómata bidireccional* es una estructura de la forma

$$AB = (Q, Ent, tran, q_0, F)$$

donde

$$\begin{array}{rcl} Q & : & \text{es el conjunto de } \textit{estados}, \\ Ent & : & \text{es el alfabeto de } \textit{entrada}, \\ tran & : & Q \times Ent \to Q \times [Izq, Der], \text{ es la función de } \textit{transición}, \\ q_0 \in Q & : & \text{es el estado } \textit{inicial}, \\ F \subset Q & : & \text{es el conjunto de estados } \textit{finales}. \end{array}$$

La semántica del autómata es muy intuitiva: Si se está leyendo la i-ésima posición de una palabra de entrada, en la cual aparece el símbolo e, y se está en el estado q entonces,

- si $tran(q, e) = (p, Izq)$ se pasa al estado p y a revisar la posición $i - 1$ de la cadena de entrada,
- si $tran(q, e) = (p, Der)$ se pasa al estado p y a revisar la posición $i + 1$ de la cadena de entrada.

Para llevar un recuento de las transiciones del autómata utilizaremos *descripciones instantáneas* (DI's). Una *DI* es una cadena

$$\sigma_{Izq} q e \sigma_{Der} \in Ent^* \times Q \times Ent \times Ent^*$$

que indica que la palabra de entrada actual es $\sigma = \sigma_{Izq} e \sigma_{Der}$, se está leyendo el símbolo e y se está en el estado q: $\sigma_{Izq} \underset{\uparrow}{e} \sigma_{Der}$, o bien de la forma $\sigma_{Izq} q \in Ent^* \times Q$, que indica que se ha arribado al estado q y se ha concluido la lectura de la palabra de entrada.

Diremos que una DI $d_1 = \sigma_{1,Izq} q_1 e_1 \sigma_{1,Der}$ *se sigue* de otra $d_0 = \sigma_{0,Izq} q_0 e_0 \sigma_{0,Der}$, $AB \vdash (d_0 \to d_1)$, si es el resultado de aplicar la transición correspondiente a d_0. Formalmente, $AB \vdash (d_0 \to d_1) \Leftrightarrow$

$$\left\{ \begin{array}{l} \left[\begin{array}{rcl} (\sigma_{0,Izq} & = & \sigma_{1,Izq} e_1) \wedge \\ (\sigma_{1,Der} & = & e_0 \sigma_{0,Der}) \end{array} \right] \quad \text{si } tran(q_0, e_0) = (q_1, Izq), \\ \left[\begin{array}{rcl} (\sigma_{0,Der} & = & e_1 \sigma_{1,Der}) \wedge \\ (\sigma_{1,Izq} & = & \sigma_{0,Izq} e_0) \end{array} \right] \quad \text{si } tran(q_0, e_0) = (q_1, Der). \end{array} \right.$$

$$\vee \left\{ \begin{array}{rcl} (d_0 & = & \sigma_{0,Izq} q_0 e_0) \wedge \\ (d_1 & = & \sigma_{0,Izq} e_0 q_1) \wedge \\ tran(q_0, e_0) & = & (q_1, Izq) \end{array} \right.$$

La cerradura reflexivo-transitiva de la relación "se sigue" es la relación "*se deriva*", denotada como $AB \vdash (d_0 \overset{*}{\to} d_1)$.

Una palabra $\sigma \in Ent^*$ es *reconocida* por el autómata bidireccional AB si para algún estado final $p \in F$ se tiene $AB \vdash (q_0 \sigma \overset{*}{\to} \sigma p)$. Es decir, una palabra es reconocida si para cuando se concluya su lectura se haya arribado a un estado final. El *lenguaje reconocido* por el autómata bidireccional AB es

$$L(AB) = [\sigma \in Ent^* | \exists p \in F : AB \vdash (q_0 \sigma \overset{*}{\to} \sigma p)].$$

Un lenguaje es *regular-B* si es reconocido por un autómata bidireccional.

$$
\begin{array}{ll}
DI[1]: & q_0 aabbbabababbbb \\
DI[2]: & aq_3 abbbababbbb \\
DI[3]: & aaq_5 bbbababbbb \\
DI[4]: & aabq_1 bbababbbb \\
DI[5]: & aabbq_3 babababbb \\
DI[6]: & aabbbq_5 ababbbb \\
DI[7]: & aabbq_4 babababbb \\
DI[8]: & aabq_5 bbababbbb \\
DI[9]: & aabbq_1 babababbb \\
DI[10]: & aabbbq_3 ababbbb \\
DI[11]: & aabbbaq_5 babbbb \\
DI[12]: & aabbbabq_1 abbbb \\
DI[13]: & aabbbaq_2 babbbb \\
DI[14]: & aabbbabq_2 abbbb \\
DI[15]: & aabbbabaq_1 bbbb \\
DI[16]: & aabbbababq_3 bbb \\
DI[17]: & aabbbababbq_5 bb \\
DI[18]: & aabbbababbbq_1 b \\
DI[19]: & aabbbababbbbq_3
\end{array}
$$

Tabla 3.16: Acción de SAB_6 sobre α_1.

Ejemplo

El autómata bidireccional SAB_3 que

- avanza a la derecha hasta que hayan aparecido dos 1's, y
- retrocede a la derecha hasta encontrar un 0, en cuyo caso reentra al estado inicial,

tiene como transición a la siguiente:

Estado	0	1
q_0	(q_0, Der)	(q_1, Der)
q_1	(q_1, Der)	(q_2, Izq)
q_2	(q_0, Der)	(q_2, Izq)

El estado inicial es q_0.

Si q_1 se designa como estado final, entonces se reconocerá a palabras de la forma $0^*(010^*)^*010^*$.

Ejemplo

Consideremos el semiautómata bidireccional SAB_6 de 6 estados sobre el alfabeto de dos símbolos $A = [a, b]$:

	q_0	q_1	q_2	q_3	q_4	q_5
a	(q_3, Der)	(q_2, Izq)	(q_1, Der)	(q_5, Der)	(q_3, Izq)	(q_4, Izq)
b	(q_4, Der)	(q_3, Der)	(q_2, Der)	(q_5, Der)	(q_5, Izq)	(q_1, Der)

Una palabra que termina de leerse es $\alpha_1 = aabbbababbbb$. En la tabla (3.16) se ve la acción de SAB_6 sobre α_1. Una palabra con un prefijo de 4 a's hace que se salga por la izquierda de la cadena de entrada. En efecto, consideremos $\alpha_2 = aaaabbbbaaaa$. En la tabla (3.17) se ve la acción de SAB_6 sobre α_2. Una palabra que cicla es $\alpha_3 = aabbabaabbbb$. En la tabla (3.18) se ve la acción de SAB_6 sobre α_3.

Si se ve la entrada del autómata como inscrita en una cinta con casillas contiguas, en cada una de las cuales se inscribe un símbolo, entonces en una palabra de k símbolos habrá $k+1$ *separadores* de casillas. Si la palabra de entrada fuese $\sigma = e_1 \dots e_k$, al escribir explícitamente a los separadores, podríamos re-escribir a σ como

$$\sigma = s_0 e_1 s_1 \dots s_{k-1} e_k s_k.$$

$DI[1]:$	$q_0 aaaabbbbaaaa$
$DI[2]:$	$aq_3 aaabbbbaaaa$
$DI[3]:$	$aaq_5 aabbbbaaaa$
$DI[4]:$	$aq_4 aaabbbbaaaa$
$DI[5]:$	$q_3 aaaabbbbaaaa$
$DI[6]:$	$aq_5 aaabbbbaaaa$
$DI[7]:$	$q_4 aaaabbbbaaaa$
$DI[8]:$	q_3 *¡Salida por la izquierda!*

Tabla 3.17: Acción de SAB_6 sobre α_2.

$DI[1]:$	$q_0 aabbabaabbbb$
$DI[2]:$	$aq_3 abbabaabbbb$
$DI[3]:$	$aaq_5 bbabaabbbb$
$DI[4]:$	$aabq_1 babaabbbb$
$DI[5]:$	$aabbq_3 abaabbbb$
$DI[6]:$	$aabbaq_5 baabbbb$
$DI[7]:$	$aabbabq_1 aabbbb$
$DI[8]:$	$aabbaq_2 baabbbb$
$DI[9]:$	$aabbabq_2 aabbbb$
$DI[10]:$	$aabbabaq_1 abbbb$
$DI[11]:$	$aabbabq_2 aabbbb$

$DI[9] = DI[11]$ *¡Ciclo!*

Tabla 3.18: Acción de SAB_6 sobre α_3.

En la figura (3.8) presentamos un diagrama de esta presentación.

La aplicación de la palabra de entrada σ en el autómata determina una *sucesión de cruce* S_i en cada uno de los separadores s_i, de acuerdo con la construcción siguiente:

1. Inicialmente hacemos $S_0 = [(q_0, Der)]$ y para cada $i > 0$, $S_i = nil$.
2. Posteriormente, si $d = \sigma_{Izq} q e \sigma_{Der}$ es la descripción instantánea actual, $d' = \sigma'_{Izq} q' e' \sigma'_{Der}$ es tal que $AB \vdash (d \to d')$ y e está en la posición j, entonces

$$\begin{aligned} tran(q,e) = (q', Izq) &\quad\Rightarrow\quad S_{j-1} = S_{j-1} * [(q', Izq)], \\ tran(q,e) = (q', Der) &\quad\Rightarrow\quad S_j \quad = S_j \quad * [(q', Der)], \end{aligned}$$

Con estas reglas, toda sucesión de cruce ha de comenzar con una pareja de la forma (q', Der) correspondiente a un movimiento a la derecha, y los movimientos correspondientes a dos parejas sucesivas en una misma sucesión de cruce habrán de alternarse. Por tanto, al cabo de la lectura de una palabra de entrada, en

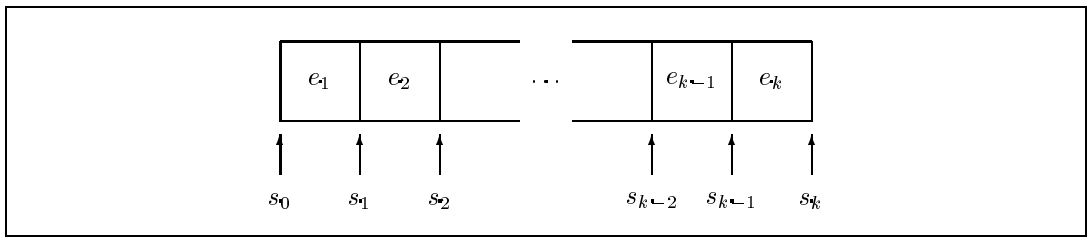


Figura 3.8: Cadena de entrada y separadores en un AB.

s_0	$1\ s_1$	$0\ s_2$	$1\ s_3$	$0\ s_4$	$0\ s_5$	$1\ s_6$
$(q_0, Der) : 1$	$(q_1, Der) : 2$	$(q_1, Der) : 3$				
		$(q_2, Izq) : 4$				
		$(q_0, Der) : 5$	$(q_1, Der) : 6$	$(q_1, Der) : 7$	$(q_1, Der) : 8$	
					$(q_2, Izq) : 9$	
					$(q_0, Der) : 10$	$(q_1, Der) : 11$

En cada entrada, los números de la derecha son un recuento secuencial de la computación.

Tabla 3.19: Sucesiones de cruce para el autómata bidireccional SAB_3.

s_0	$a\ s_1$	$a\ s_2$	$a\ s_3$	$a\ s_4$	$a\ s_5$	$a\ s_6$
$(q_0, Der) : 1$	$(q_3, Der) : 2$	$(q_5, Der) : 3$	$(q_1, Der) : 4$	$(q_3, Der) : 5$	$(q_5, Der) : 6$	
				$(q_5, Izq) : 8$	$(q_4, Izq) : 7$	
				$(q_1, Der) : 9$	$(q_3, Der) : 10$	$(q_5, Der) : 11$

$b\ s_7$	$a\ s_8$	$b\ s_9$	$b\ s_{10}$	$b\ s_{11}$	$b\ s_{12}$
$(q_1, Der) : 12$					
$(q_2, Izq) : 13$					
$(q_2, Der) : 14$	$(q_1, Der) : 15$	$(q_3, Der) : 16$	$(q_5, Der) : 17$	$(q_1, Der) : 18$	$(q_3, Der) : 19$

Tabla 3.20: Sucesiones de cruce para el autómata bidireccional SAB_6.

cada separador, la correspondiente sucesión de cruce ha de tener un número impar de elementos. Además, en el caso de que se haya aplicado una palabra que es efectivamente reconocida, ninguna sucesión de cruce puede tener dos apariciones de una misma pareja (en otro caso la "computación" se ciclaría).

Ejemplo
Para la palabra de longitud 6, 101001, las 7 sucesiones de cruce en el semiautómata bidireccional SAB_3 son las mostradas en la tabla 3.19.

Ejemplo
Para la palabra de longitud 12, *aabbbababbbb*, las 13 sucesiones de cruce en el semiautómata bidireccional SAB_6 son las mostradas en la tabla 3.20.

De manera general, diremos que una *sucesión de cruce* es una lista de parejas de la forma $(q, m) \in Q \times [Izq, Der]$ tal que

- los movimientos en la sucesión se van alternando, y
- ninguna pareja se repite en la sucesión.

Por razones que se aclararán más adelante, diremos que una sucesión de cruce es *derecha* si comienza con un movimiento a la derecha y es de longitud impar. Simétricamente, diremos que es *izquierda* si comienza con un movimiento a la izquierda y es de longitud par.

Si hay n estados en el autómata bidireccional, entonces ninguna sucesión de cruce puede tener más de $2n$ elementos. Además, para cada $j \leq n$ hay $c_{n,j} = \left(\frac{n!}{(n-j)!}\right)^2$ sucesiones de cruce de longitud $2j-1$ que comiencen con el movimiento Der. Por tanto, con n estados, hay

$$C_n = \sum_{j=1}^{n} c_{n,j} = (n!)^2 \sum_{j=1}^{n} \frac{1}{((n-j)!)^2}$$

posibles sucesiones de cruce derechas. El crecimiento de C_n es muy rápido como se ve en la tabla (3.21).

n	C_n	n	C_n
1	1	8	3705927296
2	8	9	300180111057
3	81	10	30018011105800
4	1312	11	3632179343801921
5	32825	12	523033825507476768
6	1181736	13	88392716510763573961
7	57905113	14	17324972436109660496552

Tabla 3.21: Crecimiento del número máximo de sucesiones de cruce.

En resumen, el conjunto de sucesiones de cruce para un autómata bidireccional siempre es finito. Por esto, tendremos que todo autómata bidireccional será equivalente a uno finito no-determinista. Los estados de éste último serán precisamente las sucesiones de cruce derechas posibles. En lo que resta de esta sección, veremos cómo convertir un autómata bidireccional a uno finito equivalente.

Observemos en las tablas (3.19) y (3.20) que cualesquiera dos sucesiones de cruces contiguas "concuerdan" entre sí en el sentido de que sus construcciones son consistentes con el funcionamiento del correspondiente autómata bidireccional.

Ahora bien, diremos que dos sucesiones de cruce S_1, S_2 *concuerdan* para un símbolo $e \in Ent$ si para alguna palabra $\sigma = \sigma_1 e \sigma_2$, en la cual e aparezca en una posición, digamos i, se tiene que S_1 aparece como subcadena de la sucesión de cruce S_{i-1}, S_2 aparece como subcadena de S_i y ambas cadenas forman la "traza" del funcionamiento del autómata bidireccional sobre la palabra σ. Diremos que S_1 *concuerda yendo a la derecha* con S_2 si la traza inicia con el primer elemento de S_1 y el movimiento correspondiente es a la derecha. Similarmente, diremos que S_2 *concuerda yendo a la izquierda* con S_1 si la traza inicia con el primer elemento de S_2 y el movimiento correspondiente es a la izquierda.

Para poner esto en términos más precisos:

D.1 La sucesión vacía concuerda yendo a la derecha consigo misma para cualquier símbolo e.

I.1 La sucesión vacía concuerda yendo a la izquierda consigo misma para cualquier símbolo e.

D.2 Si S_1 concuerda yendo a la derecha con S_2 para el símbolo e y $tran(q_1, e) = (q_2, Izq)$ entonces la sucesión $[(q_1, Der), (q_2, Izq)] * S_1$ concuerda yendo a la derecha con S_2 para el símbolo e.

En efecto, la pareja (q_1, Der) indica que se atraviesa al separador s_{i-1} hacia la derecha para leer el símbolo e, y luego, por la acción del AB, se vuelve a atravesar s_{i-1} pero ahora hacia la izquierda.

I.2 Si S_2 concuerda yendo a la izquierda con S_1 para el símbolo e y $tran(p_1, e) = (p_2, Der)$ entonces la sucesión $[(p_1, Izq), (p_2, Der)] * S_2$ concuerda yendo a la izquierda con S_1 para el símbolo e.

En efecto, estamos en una situación simétrica de la anterior.

D.3 Si S_2 concuerda yendo a la izquierda con S_1 para el símbolo e y $tran(q_1, e) = (p_1, Der)$ entonces $[(q_1, Der)] * S_1$ concuerda yendo a la derecha con $[(p_1, Der)] * S_2$ para el símbolo e.

En efecto, la pareja (q_1, Der) indica que se atraviesa a s_{i-1} hacia la derecha para leer el símbolo e, y luego, por la acción del AB, se atraviesa s_i hacia la derecha también. Posteriormente para mantener la concordancia es necesario que S_2 concuerde yendo a la izquierda con S_1.

I.3 Si S_1 concuerda yendo a la derecha con S_2 para el símbolo e y $tran(q_1, e) = (p_1, Izq)$ entonces $[(q_1, Izq)] * S_2$ concuerda yendo a la izquierda con $[(p_1, Izq)] * S_1$ para el símbolo e.

En efecto, estamos en una situación simétrica de la anterior.

Se sigue inmediatamente lo siguiente:

1. Si S_1 concuerda yendo a la derecha con S_2, entonces ambas sucesiones S_1 y S_2 son derechas.
2. Si S_2 concuerda yendo a la izquierda con S_1, entonces ambas sucesiones S_1 y S_2 son derechas.

3. Para cualquier palabra σ que se lea por completo en el autómata bidireccional, si $(S_i)_i$ es su sucesión de sucesiones de cruce, entonces:
   (a) cada S_i es una sucesión de cruce derecha,
   (b) cada S_{i-1} concuerda yendo a la derecha con S_i, para el i-ésimo símbolo de la palabra σ.

Proposición 3.5.1 (Equivalencia de los autómatas bidireccionales con los no-deterministas) *Todo lenguaje regular-B es regular-N, y consecuentemente es también regular. Es decir, todo autómata bidireccional es equivalente a un autómata no-determinista, y consecuentemente a un autómata finito.*

En efecto, sea $AB = (Q, Ent, tran, q_0, F)$ un autómata bidireccional. Construyamos el autómata no-determinista siguiente:

estados: $Q_1 = [\text{sucesiones de cruce derechas}]$,

estado inicial: $q_{01} = [(q_0, Der)]$,

estados finales: $F_1 = [[(q, Der)]|q \in F]$,

transiciones: $(S_1\ e\ S_2) \in tran_1 \Leftrightarrow S_1$ concuerda yendo a la derecha con S_2 para el símbolo e

Es fácil ver que ambos autómatas son equivalentes pues una "computación reconocedora", en el autómata no-determinista, corresponde a una lectura por columnas como las presentadas en las tablas 3.19 y 3.20, correspondientes a sendas "computaciones reconocedoras", en sus respectivos autómatas bidireccionales. El recíproco también vale.

El autómata descrito en la demostración anterior es muy extenso y seguramente ha de contener una gran cantidad de estados inaccesibles. Como una primera aproximación para reducir el tamaño del autómata equivalente podemos calcular sólo aquellas sucesiones de cruce que potencialmente pueden concordar, según lo permita la función de transición del autómata bidireccional.

Describamos un procedimiento que dado un símbolo a produce sendas listas de parejas de sucesiones de cruce que concuerdan yendo por la derecha, abreviado *ConcDer*, y yendo por la izquierda, abreviado *ConcIzq*, respecto al símbolo a. Para cada una de las dos relaciones, *ConcDer* y *ConcIzq*, utilizaremos un arreglo de palabras a revisar y otro de palabras ya revisadas.

1. Inicialmente consideramos la sucesión de cruce vacía λ. La pareja (λ, λ) se coloca en la lista de parejas a revisar, en tanto que la lista de parejas revisadas es vacía, para cada una de las dos relaciones de concordancia.

2. Por cada pareja de sucesiones que quede por revisar,
   (a) se toma la primera pareja $(SDer_1, SDer_2)$ a revisar según *ConcDer* para pasarla a las ya revisadas tras el procesamiento que aquí describimos,
   (b) para cada estado q sea (p, mov) el resultado de leer a en el estado q y para cada pareja $(SIzq_1, SIzq_2)$ entre las revisadas o por revisar de la relación *ConcIzq*:
      - si $mov = Izq$ entonces consideramos la pareja $([[q, Der], [p, Izq]] * SDer_1, SDer_2)$, para colocarla en la lista de las parejas a revisar según la relación *ConcDer*, si es que es la primera vez que aparece,
      - si $mov = Der$ entonces consideramos la pareja $([[q, Der]] * SIzq_1, [[p, Der]] * SIzq_2)$, para colocarla en la lista de las parejas a revisar según la relación *ConcDer*, si es que es la primera vez que aparece,
   (c) luego procedemos de manera dual, es decir, se toma la primera pareja $(SIzq_1, SIzq_2)$ a revisar según *ConcIzq* para pasarla a las ya revisadas tras el procesamiento que aquí describimos,
   (d) para cada estado q sea (p, mov) el resultado de leer a en el estado q y para cada pareja $(SDer_1, SDer_2)$ entre las revisadas o por revisar de la relación *ConcDer*:
      - si $mov = Der$ entonces consideramos la pareja $(SIzq_1, [[q, Izq], [p, Der]] * SIzq_2)$, para colocarla en la lista de las parejas a revisar según la relación *ConcIzq*, si es que es la primera vez que aparece,

- si $mov = Izq$ entonces consideramos la pareja $([[p, Izq]] * SDer_1, [[q, Izq]] * SDer_2)$, para colocarla en la lista de las parejas a revisar según la relación $ConcIzq$, si es que es la primera vez que aparece,

3. Las listas de parejas revisadas han de concordar.

El procedimiento descrito produce aún una gran cantidad de sucesiones de cruce, aunque efectivamente está descartando a muchísimas sucesiones irrelevantes.

Ejemplo

Volvamos al autómata SAB_3. El algoritmo anterior produce 48 parejas de sucesiones de cruce derechas, de manera que la primera concuerda con la segunda yendo a la derecha, respecto al símbolo 0. En la tabla (3.22(a)) las enlistamos en su orden de aparición con el algoritmo descrito. Sin embargo, ahí aparecen también sucesiones que tienen elementos repetidos. En la tabla (3.22(b)) suprimimos esas sucesiones.

En la tabla (3.23) enlistamos las parejas de sucesiones de cruce derechas, de manera que la primera concuerda con la segunda yendo a la derecha, respecto al símbolo 1, también en el orden de aparición con el algoritmo descrito.

Para el autómata SAB_6, de 6 estados, este algoritmo da un número extremadamente grande de sucesiones relacionadas.

Como otra alternativa, en vez de generar todas las sucesiones de cruce plausibles, con cierta noción de plausibilidad, partiendo de la sucesión de cruce primera $[(q_0, Der)]$, dada una sucesión de cruce actual calcularemos a todas aquellas que concuerdan con ella yendo a la derecha para cada uno de los símbolos en el alfabeto de entrada.

Para esto, definimos dos relaciones $HaciaDer_a$ y $HaciaIzq_a$ de manera tal que para cualesquiera dos sucesiones de cruce S y T:

$$\begin{aligned} SHaciaDer_a T &\Leftrightarrow S \text{ concuerda yendo a la derecha con } T \text{ respecto a } a \\ SHaciaIzq_a T &\Leftrightarrow T \text{ concuerda yendo a la izquierda con } S \text{ respecto a } a \end{aligned}$$

Diremos que una pareja $[(q, Izq), (p, Der)]$ es "de zig-zag", respecto al símbolo a si en la transición del autómata bidireccional se tiene $tran(q, a) = [p, Der]$. Diremos también que un estado q es un "antecedente" de $[p, mov]$ si para algún símbolo b se tuviese que $tran(q, b) = [p, mov]$.

De manera recursiva, tenemos:

HD.1 $[]HaciaDer_a[]$.

HD.2 $[(q, Der)] * SHaciaDer_a[(p, Der)] * T \Leftrightarrow (tran(q, a) = (p, Der))\&(SHaciaIzq_a T)$.

HD.3 $[(q, Der), (p, Izq)] * SHaciaDer_a T \Leftrightarrow (tran(q, a) = (p, Izq))\&(SHaciaDer_a T)$.

HI.1 $[]HaciaIzq_a T \Leftrightarrow T$ es de zig-zag, respecto al símbolo a.

HI.2 $[(q, Izq)] * SHaciaDer_a[(p, Izq)] * T \Leftrightarrow (q$ es antecedente de $(p, Izq))\&(SHaciaIzq_a T)$.

Utilizaremos un arreglo de sucesiones de cruce a revisar y otro de sucesiones de cruce ya revisadas.

1. Inicialmente se coloca la sucesión de cruce $[(q_0, Der)]$ en la lista de sucesiones a revisar y la de revisadas es vacía.

2. En tanto haya sucesiones por revisar,

   (a) se toma la primera sucesión S a revisar, para pasarla luego a las ya revisadas,

   (b) para cada símbolo a,

      i. se calcula la lista de sucesiones T tales que $SHaciaDer_a T$,

      ii. se pone una transición, marcada con a de S a cada tal T,

      iii. la primera vez que aparezca una tal T, se la pone en la lista de sucesiones a revisar.

$[(0, Der)]$	$[(0, Der)]$
$[(1, Der)]$	$[(1, Der)]$
$[(1, Der)]$	$[(1, Der), (0, Izq), (0, Der)]$
$[(2, Der)]$	$[(0, Der)]$
$[(2, Der)]$	$[(0, Der), (0, Izq), (0, Der)]$
$[(2, Der)]$	$[(0, Der), (1, Izq), (1, Der)]$
$[(2, Der)]$	$[(0, Der), (1, Izq), (1, Der), (0, Izq), (0, Der)]$
$[(0, Der)]$	$[(0, Der), (0, Izq), (0, Der)]$
$[(0, Der)]$	$[(0, Der), (1, Izq), (1, Der)]$
$[(0, Der)]$	$[(0, Der), (1, Izq), (1, Der), (0, Izq), (0, Der)]$
$[(0, Der)]$	$[(0, Der), (2, Izq), (0, Der)]$
$[(0, Der)]$	$[(0, Der), (2, Izq), (0, Der), (0, Izq), (0, Der)]$
$[(0, Der)]$	$[(0, Der), (2, Izq), (0, Der), (1, Izq), (1, Der)]$
$[(0, Der)]$	$[(0, Der), (2, Izq), (0, Der), (1, Izq), (1, Der), (0, Izq), (0, Der)]$
$[(1, Der)]$	$[(1, Der), (1, Izq), (1, Der)]$
$[(1, Der)]$	$[(1, Der), (1, Izq), (1, Der), (0, Izq), (0, Der)]$
$[(1, Der)]$	$[(1, Der), (2, Izq), (0, Der)]$
$[(1, Der)]$	$[(1, Der), (2, Izq), (0, Der), (0, Izq), (0, Der)]$
$[(1, Der)]$	$[(1, Der), (2, Izq), (0, Der), (1, Izq), (1, Der)]$
$[(1, Der)]$	$[(1, Der), (2, Izq), (0, Der), (1, Izq), (1, Der), (0, Izq), (0, Der)]$
$[(1, Der)]$	$[(1, Der), (0, Izq), (0, Der), (1, Izq), (1, Der)]$
$[(1, Der)]$	$[(1, Der), (0, Izq), (0, Der), (2, Izq), (0, Der)]$
$[(1, Der)]$	$[(1, Der), (0, Izq), (0, Der), (2, Izq), (0, Der), (1, Izq), (1, Der)]$
$[(2, Der)]$	$[(0, Der), (2, Izq), (0, Der)]$
$[(2, Der)]$	$[(0, Der), (2, Izq), (0, Der), (0, Izq), (0, Der)]$
$[(2, Der)]$	$[(0, Der), (2, Izq), (0, Der), (1, Izq), (1, Der)]$
$[(2, Der)]$	$[(0, Der), (2, Izq), (0, Der), (1, Izq), (1, Der), (0, Izq), (0, Der)]$
$[(2, Der)]$	$[(0, Der), (0, Izq), (0, Der), (1, Izq), (1, Der)]$
$[(2, Der)]$	$[(0, Der), (0, Izq), (0, Der), (2, Izq), (0, Der)]$
$[(2, Der)]$	$[(0, Der), (0, Izq), (0, Der), (2, Izq), (0, Der), (1, Izq), (1, Der)]$
$[(2, Der)]$	$[(0, Der), (1, Izq), (1, Der), (2, Izq), (0, Der)]$
$[(2, Der)]$	$[(0, Der), (1, Izq), (1, Der), (2, Izq), (0, Der), (0, Izq), (0, Der)]$
$[(2, Der)]$	$[(0, Der), (1, Izq), (1, Der), (0, Izq), (0, Der), (2, Izq), (0, Der)]$
$[(0, Der)]$	$[(0, Der), (0, Izq), (0, Der), (1, Izq), (1, Der)]$
$[(0, Der)]$	$[(0, Der), (0, Izq), (0, Der), (2, Izq), (0, Der)]$
$[(0, Der)]$	$[(0, Der), (0, Izq), (0, Der), (2, Izq), (0, Der), (1, Izq), (1, Der)]$
$[(0, Der)]$	$[(0, Der), (1, Izq), (1, Der), (2, Izq), (0, Der)]$
$[(0, Der)]$	$[(0, Der), (1, Izq), (1, Der), (2, Izq), (0, Der), (0, Izq), (0, Der)]$
$[(0, Der)]$	$[(0, Der), (1, Izq), (1, Der), (0, Izq), (0, Der), (2, Izq), (0, Der)]$
$[(0, Der)]$	$[(0, Der), (2, Izq), (0, Der), (0, Izq), (0, Der), (1, Izq), (1, Der)]$
$[(1, Der)]$	$[(1, Der), (1, Izq), (1, Der), (2, Izq), (0, Der)]$
$[(1, Der)]$	$[(1, Der), (1, Izq), (1, Der), (2, Izq), (0, Der), (0, Izq), (0, Der)]$
$[(1, Der)]$	$[(1, Der), (1, Izq), (1, Der), (0, Izq), (0, Der), (2, Izq), (0, Der)]$
$[(1, Der)]$	$[(1, Der), (2, Izq), (0, Der), (0, Izq), (0, Der), (1, Izq), (1, Der)]$
$[(1, Der)]$	$[(1, Der), (0, Izq), (0, Der), (1, Izq), (1, Der), (2, Izq), (0, Der)]$
$[(2, Der)]$	$[(0, Der), (2, Izq), (0, Der), (0, Izq), (0, Der), (1, Izq), (1, Der)]$
$[(2, Der)]$	$[(0, Der), (0, Izq), (0, Der), (1, Izq), (1, Der), (2, Izq), (0, Der)]$
$[(0, Der)]$	$[(0, Der), (0, Izq), (0, Der), (1, Izq), (1, Der), (2, Izq), (0, Der)]$

(a)

$[(0, Der)]$	$[(0, Der)]$
$[(1, Der)]$	$[(1, Der)]$
$[(1, Der)]$	$[(1, Der), (0, Izq), (0, Der)]$
$[(2, Der)]$	$[(0, Der)]$
$[(2, Der)]$	$[(0, Der), (1, Izq), (1, Der)]$
$[(0, Der)]$	$[(0, Der), (1, Izq), (1, Der)]$
$[(1, Der)]$	$[(1, Der), (2, Izq), (0, Der)]$

(b)

Tabla 3.22: (a) Las 48 parejas de sucesiones de cruce derechas, de manera que la primera concuerda con la segunda yendo a la derecha, respecto al símbolo 0. (b) Las 7 parejas de la tabla (a) sin sucesiones con elementos repetidos.

$[(0, Der)]$	$[(1, Der)]$
$[(1, Der), (2, Izq), (0, Der)]$	$[(1, Der)]$
$[(2, Der), (2, Izq), (0, Der)]$	$[(1, Der)]$
$[(2, Der), (2, Izq), (1, Der), (2, Izq), (0, Der)]$	$[(1, Der)]$
$[(0, Der)]$	$[(1, Der), (0, Izq), (1, Der)]$
$[(1, Der), (2, Izq), (2, Der), (2, Izq), (0, Der)]$	$[(1, Der)]$
$[(1, Der), (2, Izq), (0, Der)]$	$[(1, Der), (0, Izq), (1, Der)]$
$[(2, Der), (2, Izq), (0, Der)]$	$[(1, Der), (0, Izq), (1, Der)]$
$[(2, Der), (2, Izq), (1, Der), (2, Izq), (0, Der)]$	$[(1, Der), (0, Izq), (1, Der)]$
$[(1, Der), (2, Izq), (2, Der), (2, Izq), (0, Der)]$	$[(1, Der), (0, Izq), (1, Der)]$

Tabla 3.23: Las 10 parejas, de manera que la primera concuerda con la segunda yendo a la derecha, respecto al símbolo 1.

3. La lista de sucesiones revisadas define al autómata no-determinista equivalente.

Ejemplo

Para el autómata bidireccional SAB_3, al aplicar el algoritmo anterior se distingue a las sucesiones de cruce

0	$[(0, Der)]$
1	$[(0, Der), (1, Izq), (1, Der)]$
2	$[(1, Der)]$
3	$[(1, Der), (0, Izq), (0, Der)]$
4	$[(1, Der), (2, Izq), (0, Der)]$
5	$[(1, Der), (1, Izq), (0, Der)]$

y con la enumeración de la primer columna, la transición del autómata no-determinista es

	0	1
0	$\{0,1\}$	$\{2\}$
1	$\{\}$	$\{\}$
2	$\{2,3,4\}$	$\{\}$
3	$\{\}$	$\{\}$
4	$\{5,4\}$	$\{2\}$
5	$\{\}$	$\{\}$

y sobre éste, al calcular el autómata determinista equivalente, obtenemos el autómata

NvoEdo	*CtoViejosEdos*
0	$\{0\}$
1	$\{0,1\}$
2	$\{2\}$
3	$\{2,3,4\}$
4	$\{\}$
5	$\{2,3,4,5\}$

	0	1
0	1	2
1	1	2
2	3	4
3	5	2
4	4	4
5	5	2

donde la tabla del lado izquierdo representa una nueva enumeración ahora de los conjuntos de estados del autómata no-determinista, y la tabla de la derecha, la transición del nuevo autómata determinista. Este es el equivalente a SAB_3.

Ejemplo

Para el autómata bidireccional SAB_6, al aplicar el algoritmo anterior se distingue a 177 sucesiones de cruce para formar el autómata no-determinista *AND*. Cuando a éste se le convierte en uno determinista,

digamos AD, aparecen 63 estados. El estado 0 de AND corresponde a la sucesión de cruce $[(0, Der)]$. El estado 0 de AD corresponde al conjunto de estados $\{0\}$.

Vimos anteriormente que la palabra $\sigma = aabbbababbbb$ se lee por completo en el autómata SAB_6 y arriba a su estado 3. Cuando aplicamos σ a AD se llega a su estado 49, el cual corresponde a la colección de estados de AND

$$\{1, 3, 23, 24, 25, 26, 27, 28, 29, 30, 34, 35, 36, 37, 38, 39, 40,$$
$$41, 42, 43, 44, 46, 47, 48, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67,$$
$$68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84,$$
$$85, 86, 87, 88, 89, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126,$$
$$127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 149, 150, 151, 152, 153,$$
$$154, 155, 156, 157, 158, 159\}$$

entre los cuales está el estado 1 que corresponde a la sucesión de cruce $[(3, Der)]$.

3.6 Producto de autómatas

Sean $AF_1 = (Q_1, Ent, tran_1, q_{01}, F_1)$ y $AF_2 = (Q_2, Ent, tran_2, q_{02}, F_2)$ dos autómatas finitos. Su *semiautómata producto* se define sobre el producto cartesiano como sigue:

estados: $Q = Q_1 \times Q_2 = [(q_1, q_2) | q_1 \in Q_1, q_2 \in Q_2]$,

estado inicial: $q_{01} = (q_{01}, q_{02})$,

transiciones:
$$\begin{array}{rcl} tran : Q \times Ent & \to & Q \\ ((q_1, q_2), e) & \mapsto & tran\,((q_1, q_2), e) = (tran_1(q_1, e), tran_2(q_2, e)) \end{array}$$

(se tiene un semiautómata porque aún no hay estados finales). Denotemos por $AF = Prod(AF_1, AF_2)$ a este semiautómata.

Es evidente que

$$\begin{array}{rcl} \sigma \in Ent^*, (q_1, q_2) \in Q & \Rightarrow & tran^*\,((q_1, q_2), \sigma) = (tran_1(q_1, \sigma), tran_2(q_2, \sigma)) \\ \sigma \in Ent^* & \Rightarrow & T(\sigma) = (T_1(\sigma), T_2(\sigma)) \end{array}$$

Así pues, si definimos como conjunto de estados finales al conjunto

$$F_\vee = [(q_1, q_2) \in Q | (q_1 \in F_1) \vee (q_2 \in F_2)]$$

entonces una palabra $\sigma \in Ent^*$ será reconocida por el producto si y sólo si lo es por alguno de los autómatas AF_1 o AF_2, es decir, en este caso,

$$L(AF) = L(AF_1) \cup L(AF_2).$$

Ahora, si definimos como conjunto de estados finales al conjunto

$$F_\wedge = [(q_1, q_2) \in Q | (q_1 \in F_1) \wedge (q_2 \in F_2)]$$

entonces una palabra $\sigma \in Ent^*$ será reconocida por el producto si y sólo si lo es por ambos autómatas AF_1 o AF_2, es decir, en este caso,

$$L(AF) = L(AF_1) \cap L(AF_2).$$

Naturalmente, un autómata producto puede reducirse de manera equivalente considerando unicamente su parte accesible, es decir, aplicándole el algoritmo 3.5.

3.7 Propiedades de cerradura

Para dos lenguajes $L_1, L_2 \subset Ent^*$ definimos los operadores siguientes:

$$\begin{array}{lrcl}
\text{intersección:} & L_1 \cap L_2 & = & [\sigma | (\sigma \in L_1) \wedge (\sigma \in L_2)] \\
\text{suma:} & L_1 + L_2 & = & L_1 \cup L_2 \\
 & & = & [\sigma | (\sigma \in L_1) \vee (\sigma \in L_1)] \\
\text{producto:} & L_1 \cdot L_2 & = & [\sigma_1\sigma_2 | (\sigma_1 \in L_1) \wedge (\sigma_2 \in L_2)] \\
\text{estrella:} & & & L_1^0 = [nil] \\
 & & & \forall n \geq 0 : \; L_1^{n+1} = L_1^n \cdot L_1 \\
 & L_1^* & = & \bigcup_{n \geq 0} L_1^n \\
\text{complemento:} & L_1^c & = & \{\sigma \in Ent^* | \sigma \notin L_1\}
\end{array}$$

Proposición 3.7.1 *La clase de lenguajes regulares es cerrada bajo cada uno de los anteriores operadores.*

En efecto, veámoslo para cada operador.
Sean AF_1 y AF_2 sendos autómatas finitos que reconocen a L_1 y a L_2.

Intersección: Sea AF el autómata producto de AF_1 y AF_2 dotado de los estados finales $F_\wedge$. AF reconoce al lenguaje $L_1 \cap L_2$ y por tanto éste es regular.

Suma: Sea AF el autómata producto de AF_1 y AF_2 dotado de los estados finales $F_\vee$. AF reconoce al lenguaje $L_1 + L_2$ y por tanto éste es regular.

Alternativamente, sea GT la gráfica de transición que consiste de la estructura de AF_1 más la estructura de AF_2 más un estado inicial q_0 con correspondientes transiciones-λ $(q_0\ \lambda\ q_{01})$ y $(q_0\ \lambda\ q_{02})$. Esta gráfica reconoce a $L_1 + L_2$ y por tanto éste es regular-G. Como todo regular-G es también regular, $L_1 + L_2$ es regular.

Producto: Sea GT la gráfica de transición que consiste de la estructura de AF_1 más la estructura de AF_2 más un estado inicial q_0 con la transición-λ $(q_0\ \lambda\ q_{01})$. Añadamos también una transición-λ de cada estado final de AF_1 al estado inicial q_{02} de AF_2: $(q_1\ \lambda\ q_{02})$, $q_1 \in F_1$. Esta gráfica reconoce a $L_1 \cdot L_2$ y por tanto éste es regular-G. Así pues, $L_1 \cdot L_2$ es regular.

Estrella: Sea GT la gráfica de transición que consiste de la estructura de AF_1 más un estado inicial q_0 con la transición-λ $(q_0\ \lambda\ q_{01})$ y una transición-λ de cada estado final de AF_1 a q_0: $(q_1\ \lambda\ q_0)$, $q_1 \in F_1$. Declaremos también como final a q_0. Esta gráfica reconoce a L_1^* y por tanto éste es regular-G. Así pues, L_1^* es regular.

Complemento: Sea AF el autómata que tiene la misma estructura de AF_1 salvo en que un estado será final en AF si y sólo si ese estado no lo es en AF_1, es decir, $F = Q - F_1$. Evidentemente, AF reconoce a L_1^c y por tanto éste es un lenguaje regular.

3.8 Ejercicios

1. Construya máquinas de Mealy y de Moore con el desempeño siguiente: Para entradas en $(0+1)^*$, si la entrada termina con el sufijo 101 emite A; si la entrada termina con el sufijo 110 emite B; en otro caso emite C.

2. Construya máquinas de Mealy y de Moore con el desempeño siguiente: Para entradas en $(0+1+2)^*$, escribe el residuo en base 5 del número que, en base 3 está dado por la entrada.

3. Sea $tran : Q \times Ent \to Q$ la función de transición de un autómata finito. Demuestre que para cada $q \in Q$ y cada $\sigma_1, \sigma_2 \in Ent^*$ se tiene

$$tran^*(q, \sigma_1\sigma_2) = tran^*(tran^*(q, \sigma_1), \sigma_2).$$

4. Construya un autómata finito sobre el alfabeto $Ent = \{0,1\}$ que reconozca al lenguaje consistente de las cadenas de caracteres con tres 0's consecutivos.

5. Construya un autómata finito sobre el alfabeto $Ent = \{0,1\}$ que reconozca al lenguaje consistente de las cadenas de caracteres tales que cada bloque de cinco caracteres consecutivos contiene al menos dos 0's.

6. Construya un autómata finito sobre el alfabeto $Ent = \{1\}$ que reconozca al lenguaje $L = \{1111^{6k}1 | k \geq 0\}$.

7. Construya un autómata finito sobre el alfabeto $Ent = \{a,b\}$ que reconozca al conjunto de palabras cuyos cuatro últimos símbolos forman la partícula $bbab$.

8. Construya un autómata finito sobre el alfabeto $Ent = \{a,b\}$ que reconozca al conjunto de palabras cuyos cinco últimos símbolos incluyen dos a's y tres b's.

9. *Revisor de sumas binarias*: Construya un autómata finito sobre el alfabeto $Ent = \{0,1\}^3$ que reconozca al lenguaje L_{sum} tal que

$$\begin{array}{rcll} (a_1b_1c_1)(a_2b_2c_2)\cdots(a_kb_kc_k) & \in & L_{sum} & \Leftrightarrow \\ (a_1a_2\cdots a_k)_2 + (b_1b_2\cdots b_k)_2 & = & (c_1c_2\cdots c_k)_2 & \end{array}$$

Así, por ejemplo, como $6 + 14 = 20$ se tiene

$$\left[\begin{array}{c}0\\0\\1\end{array}\right]\left[\begin{array}{c}0\\1\\0\end{array}\right]\left[\begin{array}{c}1\\1\\1\end{array}\right]\left[\begin{array}{c}1\\1\\0\end{array}\right]\left[\begin{array}{c}0\\0\\0\end{array}\right] \in L_{sum}$$

10. Construya un autómata finito sobre el alfabeto $Ent = \{a,b,c\}$ que reconozca al lenguaje consistente de los palíndromas de longitud a lo sumo 6.

11. Construya un autómata finito sobre el alfabeto $Ent = \{a,b\}$ que reconozca al lenguaje consistente de las palabras cuyo **antepenúltimo** símbolo es b.

12. Demuestre que el autómata cuya transición es

$tran$	0	1
q_0	q_1	q_2
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_3	q_3

y tiene a q_0 como estado inicial y a él mismo como único estado final, reconoce a todas las cadenas con un mismo número de 0's y 1's que además son tales que cualquier prefijo posee a lo más un 1 más que 0's o bien a lo más un 0 más que 1's.

13. Construya un autómata no-determinista sobre el alfabeto $Ent = \{0,1\}$ que reconozca al lenguaje consistente de las cadenas de caracteres tales que dos 0's están separados por $4i$ caracteres, para algún $i \geq 0$.

14. Construya un autómata no-determinista sobre el alfabeto $Ent = \{a,b,c\}$ que reconozca al lenguaje consistente de las cadenas de caracteres tales que dan el mismo valor evaluadas de derecha a izquierda o de izquierda a derecha según la operación (no-asociativa) siguiente:

	a	b	c
a	a	a	c
b	c	a	b
c	b	c	a

15. Construya un autómata no-determinista sobre el alfabeto $Ent = \{0,1\}$ que reconozca al lenguaje consistente de las cadenas de caracteres tales que el décimo símbolo, contado desde la derecha, es un 1.

16. Pruebe que para todo autómata no-determinista A existe un autómata no-determinista B con un único estado final tal que bien $L(B) = L(A)$ o bien $L(B) = L(A) - \{nil\}$.

17. a) Construya un autómata finito determinista equivalente al autómata no-determinista cuyo estado inicial es q_0, su único estado final es q_3 y cuya tabla de transición es la siguiente:

tran	0	1
q_0	q_0, q_1	q_0
q_1	q_2	q_2
q_2	q_3	$-$
q_3	q_3	q_3

b) Construya un autómata finito determinista equivalente al autómata no-determinista cuyo estado inicial es q_0, sus estados finales son q_1, q_3 y cuya tabla de transición es la siguiente:

tran	0	1
q_0	q_1, q_3	q_1
q_1	q_2	q_1, q_2
q_2	q_3	q_0
q_3	$-$	q_2

18. Construya un autómata finito determinista equivalente al autómata bidireccional cuyo estado inicial es q_0, su estado final es q_2 y cuya tabla de transición es la siguiente:

tran	0	1
q_0	(q_0, Der)	(q_1, Der)
q_1	(q_1, Der)	(q_2, Der)
q_2	(q_2, Der)	(q_3, Izq)
q_3	(q_4, Izq)	(q_3, Izq)
q_4	(q_0, Der)	(q_4, Izq)

19. Un autómata *bidireccional no-determinista* se define de manera similar a uno bidireccional determinista con la salvedad de que para cada pareja (estado_actual, símbolo_leído) se tiene un conjunto de posibles movimientos. Pruebe que todo autómata bidireccional no-determinista es equivalente a un autómata finito determinista.
Sug. La observación de que ningún estado puede repetirse con la misma dirección en una sucesión de cruce ya no es válida en esta caso. Sin embargo, para cada cadena aceptada puede considerarse la *computación* más corta que conduce a reconocimiento.

20. Muestre que al permitir que en los autómatas bidireccionales no-deterministas la cabeza lectora permanezca inmóvil aún cuando haya cambios de estados "no se incrementa la computabilidad", es decir, todo tal autómata bidireccional es equivalente a uno finito determinista.

Capítulo 4

Expresiones regulares

4.1 Presentación conjuntista

Sea Σ un alfabeto. Para dos lenguajes cualesquiera $K, L \subset \Sigma^*$ definimos

$$\begin{array}{lllll} \textit{yuxtaposición o producto} & : K \cdot L & = & \{\sigma\tau | \sigma \in K, \tau \in L\} \\ \textit{unión o suma} & : K + L & = & \{\sigma | \sigma \in K \text{ o } \sigma \in L\} = K \cup L \\ \textit{potencias} & : K^0 & = & \{nil\} \\ & : K^{n+1} & = & K \cdot K^n \\ \textit{potencia "estrella"} & : K^* & = & \bigcup_{n \geq 0} K^n \end{array}$$

Denotemos por $\mathcal{P}(\Sigma^*)$ al conjunto de "partes" de Σ^*, es decir, a la colección de todos los lenguajes sobre el alfabeto Σ. A algunos lenguajes particulares se los distingue desde la manera en la que se los denota. Algunos de éstos son los siguientes:

$$\begin{array}{lll} \text{lenguaje vacío} & : & \mathbf{0} = \emptyset \\ \text{palabra vacía} & : & \mathbf{1} = \{nil\} \\ \text{mónada } \sigma, & : & \sigma = \{\sigma\} \\ \text{lenguaje total} & : & \Sigma^* \end{array}$$

Proposición 4.1.1 *La estructura algebraica* $(\mathcal{P}(\Sigma^*), +, \cdot, \mathbf{0}, \mathbf{1})$ *posee las propiedades siguientes:*

$$\begin{array}{lllllllll} \textit{conmutatividad} & : & & & & ; & K + L & = & L + K \\ \textit{asociatividad} & : & (K \cdot L) \cdot M & = & K \cdot (L \cdot M) & ; & (K + L) + M & = & K + (L + M) \\ \textit{unidades por la izquierda} & : & \mathbf{1} \cdot K & = & K & ; & \mathbf{0} + K & = & K \\ \textit{unidades por la derecha} & : & K \cdot \mathbf{1} & = & K & ; & K + \mathbf{0} & = & K \\ \textit{distributividad} & : & (K + L) \cdot M & = & K \cdot M + L \cdot M & ; & K \cdot (L + M) & = & K \cdot L + K \cdot M \end{array} \tag{4.1}$$

Además se cumplen también las propiedades siguientes:

Producto

$$\textit{nulidad} : K \cdot \mathbf{0} = \mathbf{0} \; ; \; \mathbf{0} \cdot K = \mathbf{0} \tag{4.2}$$

Suma

$$\textit{idempotencia} : K + K = K \; ; \tag{4.3}$$

La verificación de las relaciones anteriores es rutinaria.

Tenemos pues que $(\mathcal{P}(\Sigma^*), +, \cdot, \mathbf{0}, \mathbf{1})$ tiene una estructura de *semianillo*[1], idempotente respecto a la adición.

[1] Lo único que la falta para ser un anillo es la propiedad de existencia de inversos aditivos.

Proposición 4.1.2 (Operación estrella) *En $\mathcal{P}(\Sigma^*)$ se cumplen las relaciones siguientes:*

$$K^* = K \cdot K^* + \mathbf{1} \tag{4.4}$$

$$\forall n \geq 1: \quad K^* = K^n \cdot K^* + \left(\sum_{i=0}^{n-1} K^i\right) \tag{4.5}$$

$$\mathbf{0}^* = \mathbf{1} \tag{4.6}$$

$$(KL)^*K = K(LK)^* \tag{4.7}$$

$$(K+L)^* = (K^*L)^*K^* \tag{4.8}$$

$$(KL)^* = \mathbf{1} + K(LK)^*L \tag{4.9}$$

$$(\mathbf{1}+K)^* = K^* \tag{4.10}$$

$$\mathbf{1} + K^* = K^* \tag{4.11}$$

$$(K^*)^* = K^* \tag{4.12}$$

$$\mathbf{1}^* = \mathbf{1} \tag{4.13}$$

$$K^*K^* = K^* \tag{4.14}$$

En efecto, la ecuación (4.4) se sigue de las ecuaciones siguientes:

$$\sum_{n\geq 0} K^n = \sum_{n>0} K^n + K^0 = K \sum_{n\geq 0} K^n + K^0.$$

La ecuación (4.5) se sigue al reiterar la ecuación (4.4).

La ecuación (4.6) se sigue de la ecuación (4.4) y de la condición de nulidad (4.2).

La ecuación (4.7) se sigue al desarrollar como sumas cada uno de sus miembros para ver que ambas constan de sumandos de la forma $K(LK)^i = (KL)^iK$ con $i \geq 0$.

La ecuación (4.8) se sigue al desarrollar como sumas cada uno de sus miembros para ver que todo sumando en una suma aparece en la otra.

La ecuación (4.9) se sigue de las ecuaciones (4.4) y (4.7).

La ecuación (4.10) se sigue de la ecuación (4.4) y de que $\mathbf{1}$ es una unidad multiplicativa, (ec's. 4.1).

La ecuación (4.11) se sigue de la ecuación (4.4) y de la idempotencia, (ec. 4.3).

La ecuación (4.12) se sigue de las siguientes ecuaciones, justificadas ya anteriormente:

$$\begin{array}{ccccccc} (K^*)^* & \overset{(1)}{=} & K^* \cdot (K^*)^* + \mathbf{1} & \overset{(2)}{=} & (K+\mathbf{1})^* + \mathbf{1} & \overset{(3)}{=} & K^* + \mathbf{1} \\ & \overset{(4)}{=} & KK^* + \mathbf{1} + \mathbf{1} & \overset{(5)}{=} & KK^* + \mathbf{1} & \overset{(6)}{=} & K^* \end{array}$$

pues la igualdad (1) es una forma de la ec. (4.4), la igualdad (2) lo es de la ec. (4.8), la igualdad (3) lo es de la ec. (4.10), la igualdad (4) lo es de la ec. (4.4), la igualdad (5) lo es de la ec. (4.3) y la igualdad (6) lo es de la ec. (4.4).

La ecuación (4.13) se sigue de la ecuación (4.12) y de la ecuación (4.6) al tomar $K = \mathbf{0}$.

La ecuación (4.14) se sigue de las siguientes ecuaciones, justificadas ya anteriormente:

$$\begin{array}{ccccc} K^* & \overset{(1)}{=} & (K+\mathbf{1})^* & \overset{(2)}{=} & (K^*\mathbf{1})^*K^* \\ & \overset{(3)}{=} & (K^*)^*K^* & \overset{(4)}{=} & K^*K^* \end{array}$$

pues la igualdad (1) es una forma de la ec. (4.10), la igualdad (2) lo es de la ec. (4.8), la igualdad (3) lo es de que $\mathbf{1}$ es unidad multiplicativa y la igualdad (4) lo es de la ec. (4.12).

Diremos que un lenguaje $K \in \mathcal{P}(\Sigma^*)$ *posee la condición de la palabra vacía* (CPV) si $nil \in K$.

Lema 4.1.1 *Sea K un lenguaje que* **no** *posee la CPV. Las siguientes dos condiciones se cumplen, cualesquiera que sean los lenguajes L y X:*

$$\left.\begin{array}{r} \textit{Operador estrella} \\ \textit{como supremo.} \end{array}\right] \quad : \quad \text{Si para todo } n \in I\!N \text{ se tiene } K^nK^* + L = K^* + L, \text{ entonces } K^* + L = L. \tag{4.15}$$

$$\textit{Arden.}] \quad : \quad \text{Si } X = KX + L \text{ entonces } X = K^*L. \tag{4.16}$$

En efecto, para la primera implicación, supongamos que $\sigma \in K^*$, y sea $m = long(\sigma)$ su longitud. Sea $n > m$. Puesto que $\sigma \in (K^* + L)$, necesariamente $\sigma \in (K^n K^* + L)$, pero todas las palabras de $K^n K^*$ son de longitud estrictamente mayor que la de σ, pues K no posee la CPV, por tanto *a fortiori* $\sigma \in L$. Así pues, $K^* \subset L$ y $K^* + L = L$.

Ahora, para la segunda implicación, si se supone que $X = KX + L$ entonces, al reiterar esta ecuación, resulta $X = K^{n+1}X + \left(\sum_{i=0}^{n} K^i\right) L$. Sea $\sigma \in \Sigma^*$, y sea $n \geq long(\sigma)$. Bajo esta condición, rigen las siguientes equivalencias:

$$\sigma \in X \quad \Leftrightarrow \quad \sigma \in \left(\sum_{i=0}^{n} K^i\right) L \quad \Leftrightarrow \quad \sigma \in K^* L.$$

Por tanto, $X = K^* L$.

Ahora bien, la relación de inclusión define un "orden parcial" en la clase $\mathcal{P}(\Sigma^*)$, y es tal que para cualesquiera dos lenguajes $K, L \in \mathcal{P}(\Sigma^*)$:

$$K \subset L \Leftrightarrow K \cup L = L.$$

Así pues, atendiendo únicamente a las operaciones definidas arriba se tiene que la relación

$$K \leq L \Leftrightarrow K + L = L,$$

es un orden parcial en $\mathcal{P}(\Sigma^*)$ (que de hecho coincide con la inclusión de conjuntos).

4.2 Presentación formal

4.2.1 Sintaxis de las expresiones regulares

Sea Σ un alfabeto. La clase de *expresiones regulares* sobre Σ, $ER(\Sigma)$, se define recursivamente:

$$\begin{aligned} & \mathbf{0}, \mathbf{1} \in ER(\Sigma) \\ s \in \Sigma \quad & \Rightarrow \quad s \in ER(\Sigma) \\ K, L \in ER(\Sigma) \quad & \Rightarrow \quad K + L, K \cdot L \in ER(\Sigma) \\ K \in ER(\Sigma) \quad & \Rightarrow \quad K^* \in ER(\Sigma) \end{aligned}$$

Las expresiones regulares que *poseen la CPV* (condición de la palabra vacía) se determinan también recursivamente:

$$\begin{aligned} K = \mathbf{1} \quad & \Rightarrow \quad K \in CPV(\Sigma) \\ K = L^*, L \in ER(\Sigma) \quad & \Rightarrow \quad K \in CPV(\Sigma) \\ K = K_1 + K_2,\ K_1 \in CPV(\Sigma), K_2 \in ER(\Sigma) \quad & \Rightarrow \quad K \in CPV(\Sigma) \\ K = K_1 K_2,\ K_1, K_2 \in CPV(\Sigma) \quad & \Rightarrow \quad K \in CPV(\Sigma) \end{aligned}$$

Identificaremos entre sí a las expresiones regulares de acuerdo con las igualdades formuladas en las proposiciones 4.1.1 y 4.1.2. En otras palabras, si K, L, M son expresiones regulares entonces supondremos válidos los *axiomas*[2] enlistados en la tabla (4.1). y, por supuesto, éstos implican una a una todas las ecuaciones (4.1-4.14).

Ahora bien las ecuaciones vistas de las expresiones regulares son propiamente *reglas de transformación*: Si una expresión regular coincide con uno de los miembros en uno de los miembros de esas ecuaciones entonces se puede transformar en el otro miembro. Dos expresiones son entonces *equivalentes* si una se puede llevar a la otra mediante la aplicación de un número finito de esas reglas de transformación.

En particular, si se buscase expresiones equivalentes con el menor número de operadores, dándole prioridad a las acciones "más a la derecha", de entre las ecuaciones (4.1-4.14) se podría escoger las "producciones" enlistadas en la tabla (4.2).

[2] Presentamos los axiomas sin poner atención en el "orden" de la lógica en que se formulan, el cual orden es superior y no meramente primero.

$K + L \equiv L + K$	$(K \cdot L) \cdot M \equiv K \cdot (L \cdot M)$
$(K + L) + M \equiv K + (L + M)$	$K \cdot \mathbf{1} \equiv K \equiv \mathbf{1} \cdot K$
$K + K \equiv K$	$K \cdot \mathbf{0} \equiv \mathbf{0} \equiv \mathbf{0} \cdot K$
$K + \mathbf{0} \equiv K$	
	$K \cdot K^* \equiv K^* \cdot K$
	$K^* \equiv \mathbf{1} + K \cdot K^*$
$(K + L) \cdot M \equiv K \cdot M + L \cdot M$	$(KL)^*K = K(LK)^*$
$K \cdot (L + M) \equiv K \cdot L + K \cdot M$	$(K + L)^* \equiv (K^*L)^*K^*$
$K \notin CPV \& [\forall n : \ K^* + L = K^n K^* + L] \ \Rightarrow \ K^* + L = L$	

Tabla 4.1: Axiomas de las expresiones regulares.

	$K + L$	$\to$	$L + K$	$\mathbf{0} + K$	$\to$ K
	$(K \cdot L) \cdot M$	$\to$	$K \cdot (L \cdot M)$	$K + \mathbf{0}$	$\to$ K
	$(K + L) + M$	$\to$	$K + (L + M)$	$K \cdot \mathbf{0}$	$\to$ $\mathbf{0}$
	$(KL)^*K$	$\to$	$K(LK)^*$	$\mathbf{0} \cdot K$	$\to$ $\mathbf{0}$
	$K + K$	$\to$	K	$\mathbf{1} \cdot K$	$\to$ K
	$K \cdot K^* + \mathbf{1}$	$\to$	K^*	$K \cdot \mathbf{1}$	$\to$ K
$\forall n \geq 1 :$	$K^n \cdot K^* + \left(\sum_{i=0}^{n-1} K^i\right)$	$\to$	K^*	$(K^*L)^*K^*$	$\to$ $(K + L)^*$
	$K \cdot M + L \cdot M$	$\to$	$(K + L) \cdot M$	$\mathbf{1} + K(LK)^*L$	$\to$ $(KL)^*$
	$K \cdot L + K \cdot M$	$\to$	$K \cdot (L + M)$	$(\mathbf{1} + K)^*$	$\to$ K^*
	$\mathbf{0}^*$	$\to$	$\mathbf{1}$	$(K^*)^*$	$\to$ K^*
	$\mathbf{1}^*$	$\to$	$\mathbf{1}$	K^*K^*	$\to$ K^*

Tabla 4.2: Reglas de producción para reducir expresiones regulares.

Ejemplo. Sobre el alfabeto $Dos = (0 + 1)$, consideremos la expresión regular:

$$exp_1 = 1^* + (1^* + 1^*0^*)(1^* + 1^*0^*)^*1^*$$

Las producciones anteriores realizan la transformación siguiente:

$$\begin{array}{rclcrcl} exp_1 & = & [\mathbf{1} + (1^* + 1^*0^*)(1^* + 1^*0^*)^*]\,1^* & = & (1^* + 1^*0^*)^*\,1^* & = & (1^*\,(\mathbf{1} + 0^*))^*\,1^* \\ & = & (1 + (\mathbf{1} + 0^*))^* & = & (\mathbf{1} + (1 + 0^*))^* & = & (1 + 0^*)^* \\ & = & (0^* + 1)^* & & & & \end{array}$$

y sirva el ejemplo para ilustrar que el símbolo $\mathbf{1}$ que denota a la palabra vacía es distinto del símbolo $1 \in Dos$.

Como un segundo ejemplo, de manera inductiva en la caracterización de las expresiones con la CPV se demuestra la siguiente:

Observación 4.2.1 *Para toda expresión regular K que posea la CPV existe una expresión regular L tal que L no posee la CPV y $K = \mathbf{1} + L$.*

4.2.2 Interpretación estándar

Toda expresión regular *se interpreta* como un lenguaje:

$$\begin{array}{rcl} & & Int(\mathbf{0}) = \emptyset \\ & & Int(\mathbf{1}) = \{nil\} \\ s \in \Sigma & \Rightarrow & Int(s) = \{s\} \\ K, L \in ER(\Sigma) & \Rightarrow & Int(K + L) = Int(K) + Int(L), \\ & & Int(K \cdot L) = Int(K) \cdot Int(L), \\ K \in ER(\Sigma) & \Rightarrow & Int(K^*) = (Int(K))^* \end{array}$$

Observación 4.2.2 *Una expresión regular K posee la CPV si y sólo si $nil \in Int(K)$.*

Diremos que un lenguaje $L \subset \Sigma^*$ es *formalmente regular* si existe una expresión regular $K \in ER(\Sigma)$ que se interpreta como L, es decir, K es tal que $L = Int(K)$.

Ya que, vistos como conjuntos de palabras, los axiomas de las expresiones regulares, se cumplen en el conjunto $\mathcal{P}(\Sigma^*)$ se tiene, evidentemente, la

Proposición 4.2.1 (Coherencia) *Si dos expresiones regulares son equivalentes entonces se interpretan como un mismo lenguaje.*

El recíproco también se cumple, pero su demostración excede los límites del presente curso (véase, por ej. [21]).

Proposición 4.2.2 (Completitud) *Si dos expresiones regulares se interpretan como un mismo lenguaje entonces son equivalentes.*

El espacio cociente $(ER(\Sigma)/\equiv)$ se identifica naturalmente con la clase de los lenguajes formalmente regulares.

4.2.3 De expresiones regulares a autómatas

Proposición 4.2.3 *Todo lenguaje formalmente regular es regular.*

En efecto, veamos que si L es un conjunto formalmente regular entonces existe una gráfica de transición que lo reconoce. Para esto, procederemos inductivamente según las construcciones de las expresiones regulares.

Casos básicos:

1. Cualquier gráfica de transición sin estados finales reconoce al conjunto vacío.

2. Para reconocer a la palabra vacía consideremos dos estados q_0, q_1. El primero, q_0, es a la vez inicial y final. Bajo cualquier símbolo se pasa a q_1 y ahí se queda cuando llega cualquier otro símbolo. Evidentemente, esta gráfica de transición sólo reconoce a la palabra vacía.

3. De manera similar, para un símbolo $s \in \Sigma$ dado, consideremos una gráfica de tres estados q_0, q_1, q_2 con las transiciones:

$$(q_0, t) \mapsto \begin{cases} q_1 & \text{si } t = s, \\ q_2 & \text{en otro caso.} \end{cases} \quad ; \quad \text{y para } i = 1, 2 \text{ hagamos } (q_i, t) \mapsto q_2.$$

Al hacer inicial a q_0, y final sólo a q_1, se reconoce únicamente al símbolo s.

Casos inductivos:

Supongamos construídas sendas gráficas de transición G_A y G_B que reconozcan a los lenguajes que interpretan a las expresiones regulares A y B. Sean q_{0A} y q_{0B} los respectivos estados iniciales y sean F_A y F_B los respectivos conjuntos de estados finales.

1. Para reconocer $A + B$ introduzcamos un nuevo estado inicial q_0 y la unión de ambas gráficas G_A y G_B. Tendamos transiciones vacías de q_0 a cada uno de los estados iniciales q_{0A} y q_{0B}: $(q_0, nil) \mapsto q_{0A}$, $(q_0, nil) \mapsto q_{0B}$; y como conjunto de estados finales consideremos a la unión $F_A \cup F_B$.

Esta nueva gráfica reconoce efectivamente a la suma $A + B$.

2. Para reconocer $A \cdot B$ "pongamos a G_A antes de G_B": Consideremos la unión de ambas gráficas G_A y G_B, como estado inicial tomamos a q_{0A}, tendamos transiciones vacías de cada uno de los estados finales en F_A al estado inicial q_{0B}: $(q, nil) \mapsto q_{0B}$, $q \in F_A$, y como conjunto de estados finales consideremos a F_B.

Esta nueva gráfica reconoce efectivamente al producto $A \cdot B$.

3. Para reconocer A^* consideramos G_A más un nuevo estado q_0 que será el inicial de la nueva gráfica y su único estado final. Tendemos una transición vaciía de q_0 al inicial anterior q_{0A} y de cada estado final anterior tendemos una transición vacía a q_0.

Esta nueva gráfica reconoce efectivamente a la potencia estrella A^*.

Veremos más adelante que el recíproco también es cierto: Todo lenguaje regular puede representarse mediante una expresión regular.

4.3 Estructura algebraica de las expresiones regulares

4.3.1 Orden

Sea Σ un alfabeto. En la clase de expresiones regulares $ER(\Sigma)$ introducimos la relación

$$\forall A, B : \ A \leq B \ \Leftrightarrow \ A + B \equiv B.$$

Tenemos con ésta, una relación reflexiva y transitiva. Esta relación aunque no es antisimétrica en $ER(\Sigma)$ sí lo es en el cociente $(ER(\Sigma)/\equiv)$ en donde, como ya vimos, coincide con la relación de orden de inclusión en los lenguajes formalmente regulares.

Proposición 4.3.1 *Las siguientes relaciones son verdaderas:*

1. $\mathbf{0} \in ER(\Sigma)$ *es el elemento mínimo, es decir,* $\forall A \in ER(\Sigma) : \ \mathbf{0} \leq A$.
2. A *posee la CPV si y sólo si* $\mathbf{1} \leq A$.
3. $\forall A \in ER(\Sigma) : \ \mathbf{1} \leq A^*$.
4. $\forall A, B \in ER(\Sigma) : \ A \leq A + B$.
5. $\forall A \in ER(\Sigma), \forall n \in I\!N : \ \mathbf{1} + A + \cdots + A^n \leq A^*$.
6. *Si* $A \leq B$ *entonces* $\forall C \in ER(\Sigma)$*:*

- $A \in CPV \Rightarrow B \in CPV$,
- $A + C \leq B + C$,
- $A \cdot C \leq B \cdot C$ y $C \cdot A \leq C \cdot B$,
- $A^* \leq B^*$.

En efecto, verifiquemos cada una de las aseveraciones:

1. Como $\mathbf{0}$ es la unidad aditiva, $\mathbf{0} + A = A$, por tanto, $\mathbf{0} \leq A$.
2. Por inducción en la caracterización de las expresiones que poseen la CPV se ve que $A \in CPV \Leftrightarrow \mathbf{1} + A = A$.
3. Esta desigualdad se sigue inmediatamente de que A^* posee la CPV.
4. Por ser la suma idempotente, $A + (A + B) = A + B$, por tanto, $A \leq A + B$.
5. Por la ec. (4.5) $\forall A \in ER(\Sigma), \forall n \in \mathbb{N}$ se tiene:
$$A^* + A^{n+1}A^* = \left(\sum_{i=0}^{n} A^n + A^{n+1}A^*\right) + A^{n+1}A^* = \sum_{i=0}^{n} A^n + A^{n+1}A^* = A^*$$
luego
$$\mathbf{1} + A + \cdots + A^n + A^* = \mathbf{1} + A + \cdots + A^n + (A^{n+1}A^* + A^*) = (A^* + A^*) = A^*,$$
por tanto, $\mathbf{1} + A + \cdots + A^n \leq A^*$.
6. Si $A \leq B$ entonces $A + B = B$, luego $\forall C \in ER(\Sigma)$:
   - como $B = (A + B)$, $B \in CPV$,
   - $(A + C) + (B + C) = (A + B) + C = B + C$, por tanto, $A + C \leq B + C$,
   - $A \cdot C + B \cdot C = (A + B) \cdot C = B \cdot C$, por tanto, $A \cdot C \leq B \cdot C$,
   - Ya que $A \leq B$ y $B \leq B^*$ se tiene $A \leq B^*$. Luego, $\forall n : \left(\sum_{i=0}^{n-1} A^i\right) \leq B^*$. En consecuencia, $A^* + B^* = A^nA^* + B^*$. Si A no posee la CPV, del último de los axiomas enlistados en la tabla (4.1) se sigue $A^* \leq B^*$. Si A posee la CPV, escribamos $A = \mathbf{1} + A_1$ donde A_1 no posee la CPV. Entonces, $A^* = (\mathbf{1} + A_1)^* = A_1^* \leq B^*$.

Tenemos pues, en conclusión, que las operaciones regulares son monótonas respecto a la relación de orden.

4.3.2 Puntos fijos de ecuaciones lineales

Dadas dos expresiones regulares $A, B \in ER(\Sigma)$ sea $P_{A,B} = \{X \in ER(\Sigma) | X = AX + B\}$ el conjunto de expresiones regulares que son puntos fijos de la "función" $X \mapsto AX + B$.

Observación 4.3.1 *La siguientes relaciones son verdaderas:*

1. *La expresión A^*B está en $P_{A,B}$.*
2. *A^*B es una cota inferior de $P_{A,B}$. Es decir, $\forall X \in ER(\Sigma) : X \in P_{A,B} \Rightarrow A^*B \leq X$.*
3. *Si A no posee la CPV, entonces $P_{A,B}$ consta del único elemento A^*B. Es decir, A^*B es el único punto fijo de la "transformación lineal" $X \mapsto AX + B$.*

En efecto:

1. $A^*B = (\mathbf{1} + AA^*)B = A(A^*B) + B$. Así pues, $A^*B \in P_{A,B}$.

2. Si $X \in P_{A,B}$ entonces $X = AX + B$ y, mediante sustituciones sucesivas, se tiene

$$\forall n: \ X = A^{n+1}X + (A^n + A^{n-1} + \cdots + A + \mathbf{1})B,$$

y por tanto $\forall n : X \geq A^n B$. Consecuentemente, $X \geq A^* B$.

3. Esto es un replanteamiento del resultado formulado en la ec. (4.16).

Resumimos todo lo anterior, aunque sea de manera reiterada, en el siguiente

Lema 4.3.1 (Arden) $\forall A, B \in ER(\Sigma): \quad \mathbf{1} \not\leq A \quad \Rightarrow \quad (X = AX + B \ \Leftrightarrow \ X = A^*B)$.

La suposición de que A no posea la CPV es importante. Por ejemplo, si A poseyera la CPV y B fuese cualquier expresión regular, entonces siempre que $X \geq B$ se tendrá: $X = X + B = \mathbf{1}X + B$. Sin embargo, no necesariamente se tendrá que $X = \mathbf{1}^* B = B$.

4.3.3 Matrices de expresiones regulares

Una matriz de expresiones regulares de m renglones y n columnas es un arreglo $\mathbf{A} = (A_{ij})_{1 \leq i \leq m}^{1 \leq j \leq n}$ tal que $\forall i, j : A_{ij} \in ER(\Sigma)$. Denotemos por $ER(\Sigma)^{m \times n}$ a la clase de matrices de expresiones regulares de m renglones y n columnas.

Si $\mathbf{A}, \mathbf{B} \in ER(\Sigma)^{m \times n}$ su *suma* es la matriz $\mathbf{C} \in ER(\Sigma)^{m \times n}$ tal que $\forall i, j : C_{ij} = A_{ij} + B_{ij}$.

Si $\mathbf{B} \in ER(\Sigma)^{m \times n}$ y $A \in ER(\Sigma)$ el *producto por un escalar por la izquierda* $A\mathbf{B}$ es la matriz $\mathbf{C} \in ER(\Sigma)^{m \times n}$ tal que $\forall i, j : C_{ij} = A \cdot B_{ij}$. Simétricamente, el *producto por la derecha* $\mathbf{B}A$ es la matriz $\mathbf{C} \in ER(\Sigma)^{m \times n}$ tal que $\forall i, j : C_{ij} = B_{ij} \cdot A$.

También de manera convencional, definimos el producto de dos matrices $\mathbf{A} \in ER(\Sigma)^{m \times n}$ y $\mathbf{B} \in ER(\Sigma)^{n \times p}$ como la matriz $\mathbf{C} \in ER(\Sigma)^{m \times p}$ tal que $\forall i, j : C_{ij} = \sum_{k=1}^{n} A_{ik} \cdot B_{kj}$.

Con estas operaciones las matrices cuadradas $ER(\Sigma)^{n \times n}$ poseen una estructura de semianillo (lo único que les falta para ser un anillo son inversos aditivos) no-conmutativo. La unidad aditiva es la matriz *cero* $\mathbf{0}^{n \times n}$ y la unidad multiplicativa es la matriz *identidad* $\mathbf{Id}_n = diag(\mathbf{1}, \ldots, \mathbf{1})$.

Las operaciones matriciales son congruentes con "particiones por bloques", es decir, si se parte a dos matrices $\mathbf{A}, \mathbf{B} \in ER(\Sigma)^{n \times n}$ como

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \quad , \quad \mathbf{B} = \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{12} \\ \mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix}$$

donde bloques con mismos índices ij son de mismos tamaños, entonces

$$\begin{aligned} \mathbf{A} + \mathbf{B} &= \begin{bmatrix} \mathbf{A}_{11} + \mathbf{B}_{11} & \mathbf{A}_{12} + \mathbf{B}_{12} \\ \mathbf{A}_{21} + \mathbf{B}_{21} & \mathbf{A}_{22} + \mathbf{B}_{22} \end{bmatrix} \\ \mathbf{AB} &= \begin{bmatrix} \mathbf{A}_{11}\mathbf{B}_{11} + \mathbf{A}_{12}\mathbf{B}_{21} & \mathbf{A}_{11}\mathbf{B}_{12} + \mathbf{A}_{12}\mathbf{B}_{22} \\ \mathbf{A}_{21}\mathbf{B}_{11} + \mathbf{A}_{22}\mathbf{B}_{21} & \mathbf{A}_{21}\mathbf{B}_{12} + \mathbf{A}_{22}\mathbf{B}_{22} \end{bmatrix} \end{aligned}$$

Dadas dos matrices $\mathbf{A}, \mathbf{B} \in ER(\Sigma)^{n \times n}$, diremos que $\mathbf{A} \leq \mathbf{B}$ si $\forall i, j \leq n : \ A_{ij} \leq B_{ij}$. Diremos también que una matriz $\mathbf{A}$ posee la CPV si $\mathbf{Id} \leq \mathbf{A}$.

Ahora la operación "estrella" de matrices ha de generalizar la operación "estrella" de las expresiones regulares. Consecuentemente, se ha de definir de manera que

- se satisfaga la ecuación $\mathbf{A}^* = \mathbf{Id} + \mathbf{A}\mathbf{A}^*$, y
- la solución de la ecuación $\mathbf{X} = \mathbf{A}\mathbf{X} + \mathbf{B}$ sea $\mathbf{X} = \mathbf{A}^*\mathbf{B}$, siempre que $\mathbf{A}$ no posea la CPV.

Para esto, supongamos primero que ya se haya construido tal operación "estrella". Denotemos por $\mathbf{Y} = \mathbf{A}^*$ a la "estrella" de $\mathbf{A}$. Si consideramos particiones de tamaños similares

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} \quad , \quad \mathbf{Y} = \begin{bmatrix} \mathbf{Y}_{11} & \mathbf{Y}_{12} \\ \mathbf{Y}_{21} & \mathbf{Y}_{22} \end{bmatrix} \quad , \quad \mathbf{Id} = \begin{bmatrix} \mathbf{Id}' & \mathbf{0} \\ \mathbf{0} & \mathbf{Id}'' \end{bmatrix}$$

de la ecuación $\mathbf{A}^* = \mathbf{Id} + \mathbf{A}\mathbf{A}^*$ resultará

$$\begin{aligned}
\mathbf{Y}_{11} &= \mathbf{A}_{11}\mathbf{Y}_{11} + \mathbf{A}_{12}\mathbf{Y}_{21} + \mathbf{Id}' && (4.17)\\
\mathbf{Y}_{12} &= \mathbf{A}_{11}\mathbf{Y}_{12} + \mathbf{A}_{12}\mathbf{Y}_{22} && (4.18)\\
\mathbf{Y}_{21} &= \mathbf{A}_{21}\mathbf{Y}_{11} + \mathbf{A}_{22}\mathbf{Y}_{21} && (4.19)\\
\mathbf{Y}_{22} &= \mathbf{A}_{21}\mathbf{Y}_{12} + \mathbf{A}_{22}\mathbf{Y}_{22} + \mathbf{Id}'' && (4.20)
\end{aligned}$$

De la ecuación (4.19) resulta:

$$\mathbf{Y}_{21} = \mathbf{A}_{22}^*\mathbf{A}_{21}\mathbf{Y}_{11}. \tag{4.21}$$

De la ecuación (4.18) resulta:

$$\mathbf{Y}_{12} = \mathbf{A}_{11}^*\mathbf{A}_{12}\mathbf{Y}_{22} \tag{4.22}$$

Al sustituir (4.21) en (4.17) obtenemos

$$\mathbf{Y}_{11} = (\mathbf{A}_{11} + \mathbf{A}_{12}\mathbf{A}_{22}^*\mathbf{A}_{21})\,\mathbf{Y}_{11} + \mathbf{Id}'$$

y en consecuencia

$$\mathbf{Y}_{11} = (\mathbf{A}_{11} + \mathbf{A}_{12}\mathbf{A}_{22}^*\mathbf{A}_{21})^*. \tag{4.23}$$

Al sustituir (4.22) en (4.20) obtenemos

$$\mathbf{Y}_{22} = (\mathbf{A}_{22} + \mathbf{A}_{21}\mathbf{A}_{11}^*\mathbf{A}_{12})\,\mathbf{Y}_{22} + \mathbf{Id}''$$

y en consecuencia

$$\mathbf{Y}_{22} = (\mathbf{A}_{22} + \mathbf{A}_{21}\mathbf{A}_{11}^*\mathbf{A}_{12})^*. \tag{4.24}$$

De las ecuaciones (4.24), (4.23), (4.22) y (4.21) obtenemos

$$\mathbf{Y} = \mathbf{A}^* = \begin{bmatrix} (\mathbf{A}_{11} + \mathbf{A}_{12}\mathbf{A}_{22}^*\mathbf{A}_{21})^* & \mathbf{A}_{11}^*\mathbf{A}_{12}\,(\mathbf{A}_{22} + \mathbf{A}_{21}\mathbf{A}_{11}^*\mathbf{A}_{12})^* \\ \mathbf{A}_{22}^*\mathbf{A}_{21}\,(\mathbf{A}_{11} + \mathbf{A}_{12}\mathbf{A}_{22}^*\mathbf{A}_{21})^* & (\mathbf{A}_{22} + \mathbf{A}_{21}\mathbf{A}_{11}^*\mathbf{A}_{12})^* \end{bmatrix}. \tag{4.25}$$

Ahora bien, de la identidad $(X+Y)^* = (X^*Y)^*X^*$ obtenemos la expresión equivalente

$$\mathbf{A}^* = \begin{bmatrix} (\mathbf{A}_{11}^*\mathbf{A}_{12}\mathbf{A}_{22}^*\mathbf{A}_{21})^*\,\mathbf{A}_{11}^* & \mathbf{A}_{11}^*\mathbf{A}_{12}\,(\mathbf{A}_{22}^*\mathbf{A}_{21}\mathbf{A}_{11}^*\mathbf{A}_{12})^*\,\mathbf{A}_{22}^* \\ \mathbf{A}_{22}^*\mathbf{A}_{21}\,(\mathbf{A}_{11}^*\mathbf{A}_{12}\mathbf{A}_{22}^*\mathbf{A}_{21})^*\,\mathbf{A}_{11}^* & (\mathbf{A}_{22}^*\mathbf{A}_{21}\mathbf{A}_{11}^*\mathbf{A}_{12})^*\,\mathbf{A}_{22}^* \end{bmatrix}.$$

También de la identidad $(XY)^*X = X(YX)^*$ obtenemos

$$\begin{aligned}
\mathbf{A}_{11}^*\mathbf{A}_{12}\,(\mathbf{A}_{22}^*\mathbf{A}_{21}\mathbf{A}_{11}^*\mathbf{A}_{12})^* &= (\mathbf{A}_{11}^*\mathbf{A}_{12}\mathbf{A}_{22}^*\mathbf{A}_{21})^*\,\mathbf{A}_{11}^*\mathbf{A}_{12}\,,\\
\mathbf{A}_{22}^*\mathbf{A}_{21}\,(\mathbf{A}_{11}^*\mathbf{A}_{12}\mathbf{A}_{22}^*\mathbf{A}_{21})^* &= (\mathbf{A}_{22}^*\mathbf{A}_{21}\mathbf{A}_{11}^*\mathbf{A}_{12})^*\,\mathbf{A}_{22}^*\mathbf{A}_{21}
\end{aligned}$$

y por tanto,

$$\mathbf{A}^* = \begin{bmatrix} (\mathbf{A}_{11}^*\mathbf{A}_{12}\mathbf{A}_{22}^*\mathbf{A}_{21})^*\,\mathbf{A}_{11}^* & (\mathbf{A}_{11}^*\mathbf{A}_{12}\mathbf{A}_{22}^*\mathbf{A}_{21})^*\,\mathbf{A}_{11}^*\mathbf{A}_{12}\mathbf{A}_{22}^* \\ (\mathbf{A}_{22}^*\mathbf{A}_{21}\mathbf{A}_{11}^*\mathbf{A}_{12})^*\,\mathbf{A}_{22}^*\mathbf{A}_{21}\mathbf{A}_{11}^* & (\mathbf{A}_{22}^*\mathbf{A}_{21}\mathbf{A}_{11}^*\mathbf{A}_{12})^*\,\mathbf{A}_{22}^* \end{bmatrix}. \tag{4.26}$$

Las ecuaciones (4.25) y (4.26) dan sendos métodos de cálculo de la matriz $\mathbf{A}^*$. Así, por ejemplo, de acuerdo con la ecuación (4.26), dada una matriz $\mathbf{A} \in (ER(\Sigma))^{n\times n}$ y una partición de ella por bloques

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$$

entonces al hacer

$$\begin{aligned}
\mathbf{C}_{11} &= \mathbf{A}_{11}^* &,\ \mathbf{C}_{22} &= \mathbf{A}_{22}^*\\
\mathbf{B}_{12} &= \mathbf{A}_{12}\mathbf{C}_{22} &,\ \mathbf{B}_{21} &= \mathbf{A}_{21}\mathbf{C}_{11}\\
\mathbf{D}_{11} &= (\mathbf{C}_{11}\mathbf{B}_{12}\mathbf{A}_{21})^* &,\ \mathbf{D}_{22} &= (\mathbf{C}_{22}\mathbf{B}_{21}\mathbf{A}_{12})^*\\
\mathbf{B}_{11} &= \mathbf{D}_{11}\mathbf{C}_{11} &,\ \mathbf{B}_{22} &= \mathbf{D}_{22}\mathbf{C}_{22}
\end{aligned}$$

hemos de tener que

$$\mathbf{A}^* = \begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{11}\mathbf{B}_{12} \\ \mathbf{B}_{22}\mathbf{B}_{21} & \mathbf{B}_{22} \end{bmatrix}$$

Los cálculos anteriores pueden realizarse para cualquier partición de la matriz $\mathbf{A}$ por bloques. Sin embargo, diversas particiones serán más eficientes respecto a otras.

Como un primer ejemplo, supongamos que formamos bloques "quitando" el último renglón y la última columna. En otras palabras, supongamos

$$\begin{aligned} \mathbf{A}_{11} &\in (ER(\Sigma))^{(n-1)\times(n-1)} \\ \mathbf{A}_{12} &\in (ER(\Sigma))^{(n-1)\times 1} \\ \mathbf{A}_{21} &\in (ER(\Sigma))^{1\times(n-1)} \\ \mathbf{A}_{22} &\in (ER(\Sigma))^{1\times 1} \end{aligned}$$

Sea $T(n)$ el número de potencias "estrellas" escalares necesarias para calcular la potencia "estrella" de $\mathbf{A}$. Siguiendo las relaciones anteriores, necesitamos calcular 2 potencias "estrellas" de matrices $(n-1)\times(n-1)$, las de las matrices $\mathbf{A}_{11}$ y $\mathbf{C}_{11}\mathbf{B}_{12}\mathbf{A}_{21}$, y dos de matrices 1×1, las de las matrices $\mathbf{A}_{22}$ y $\mathbf{C}_{22}\mathbf{B}_{21}\mathbf{A}_{12}$. Por tanto,

$$\begin{aligned} \forall n>0:\ T(n) &= 2T(n-1)+2 \\ T(1) &= 1 \end{aligned}$$

Al resolver la recurrencia anterior obtenemos como solución

$$T(n) = 3\cdot 2^{n-1} - 2,$$

y consecuentemente $T(n) = O(2^n)$.

Ahora, como un segundo ejemplo, supongamos que $n = 2^k$ es una potencia de 2. En este caso, de acuerdo con las relaciones anteriores, para calcular la potencia "estrella" de $\mathbf{A}$ hay que calcular 4 potencias "estrellas" de matrices $\frac{n}{2}\times\frac{n}{2}$. Por tanto,

$$T(2^k) = T(n) = 4T\left(\frac{n}{2}\right) = 4T(2^{k-1}).$$

Esta vez, la recurrencia tiene solución

$$T(n) = 4^k = \left(2^k\right)^2 = n^2.$$

Ya que $\lim_{n\to+\infty}\frac{n^2}{2^n} = 0$ vemos que es mucho más conveniente introducir particiones por "medias" matrices que quitando un renglón y una columna a la vez.

Como en el caso "escalar" tenemos que se cumple el siguiente

Lema 4.3.2 *Si $\mathbf{A}\in(ER(\Sigma))^{n\times n}$ y $\mathbf{b}\in(ER(\Sigma))^n$ entonces la mínima solución de la ecuación $\mathbf{X} = \mathbf{A}\mathbf{X}+\mathbf{b}$ es $\mathbf{X} = \mathbf{A}^*\mathbf{b}$. Más aún, si $\mathbf{A}$ no posee la CPV entonces la solución mínima es la única solución.*

Ejemplo:

Para una matriz $\mathbf{A}_2\in ER(\Sigma)^{2\times 2}$,

$$\mathbf{A}_2 = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

se tiene, aplicando el algoritmo de cálculo de la operación "estrella" en matrices, que

$$\mathbf{A}_2^* = \begin{bmatrix} \left[a_{11}^* a_{12} a_{22}^* a_{21}\right]^* a_{11}^* & a_{12}a_{22}^* \\ a_{21}a_{11}^* & \left[a_{22}^* a_{21} a_{11}^* a_{12}\right]^* a_{22}^* \end{bmatrix} \tag{4.27}$$

Ahora bien para una matriz $\mathbf{A}_4 \in ER(\Sigma)^{4\times 4}$,

$$\mathbf{A}_4 = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

es necesario reiterar desde un segundo nivel el procedimiento descrito de la operación "estrella" en matrices. En este caso tendremos que

$$\mathbf{A}_4^* = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

donde al hacer

$$\begin{array}{lcll} \forall i,j \leq 4 & : & f_{ij} & = a_{ii}^* a_{ij} a_{jj}^* a_{ji} \\ \forall i,j,k \leq 4 & : & g_{ijk} & = a_{ij} f_{jk}^* a_{jj}^* + a_{ik} a_{kj} a_{jj}^* \\ \forall i,j,k,l \leq 4 & : & h_{ijkl} & = g_{ikl} a_{kj} + g_{ilk} a_{lj} \\ \forall i,j,k,l \leq 4 & : & u_{ijkl} & = a_{ii}^* h_{ijkl} \\ \forall i,j,k,l,m \leq 4 & : & w_{ijklm} & = a_{ij} u_{jklm} + f_{ij}^* u_{iklm} \end{array}$$

resulta, por ejemplo,

$$b_{11} = w_{12234} w_{21234}^* a_{21} a_{11}^* + \left[w_{12134}^* w_{12234} w_{21234}^* w_{21134} \right]^* w_{12134}^* f_{12}^* a_{11}^*$$

y los demás términos b_{ij} tienen expresiones similares, en cuanto a la complejidad de sus expresiones.

4.4 El teorema de Kleene

Veremos en esta sección que se cumple el recíproco de la proposición 4.2.3: Todo lenguaje regular ha de ser formalmente regular.

Sea $GT = (Q, Ent, tran, q_0, F)$ una gráfica de transición. Consideremos la siguiente transformación:

$$\begin{array}{rcl} \Gamma : Ent^* & \to & \mathcal{P}(Ent^*) \\ \sigma & \mapsto & \Gamma(\sigma) = \{\tau \in Ent^* | \sigma\tau \in L(GT)\} \end{array}$$

Observación 4.4.1 *El lenguaje de la gráfica de transición $L(GT)$ coincide con el conjunto $\Gamma(nil)$.*

Observación 4.4.2 *$\forall \sigma \in Ent^*$:*

1. *$nil \in \Gamma(\sigma) \Leftrightarrow \sigma \in L(GT)$.*
2. *$\forall t \in Ent, \forall \tau \in Ent^* : t\tau \in \Gamma(\sigma) \Leftrightarrow \tau \in \Gamma(\sigma t)$.*

Por tanto, se tiene la igualdad conjuntista

$$\Gamma(\sigma) = \left(\bigcup_{t \in Ent} t \cdot \Gamma(\sigma t) \right) \cup \gamma(\sigma),$$

donde $\gamma(\sigma) = \begin{cases} \{nil\} & \text{si } \sigma \in L(GT), \\ \emptyset & \text{si } \sigma \notin L(GT). \end{cases}$. Y puesta la igualdad anterior como una expresión regular queda

$$\Gamma(\sigma) = \sum_{t \in Ent} t \cdot \Gamma(\sigma t) + \gamma(\sigma). \tag{4.28}$$

Ahora, de manera similar a como se hizo en la ec. (3.6), definimos la relación de *indistinguibilidad* entre los estados de GT haciendo

$$\forall p, q \in Q : \; p \equiv_{Ind} q \Leftrightarrow \forall \tau \in Ent^* \left[tran^*(p,\tau) \cap F \neq \emptyset \Leftrightarrow tran^*(q,\tau) \cap F \neq \emptyset \right] \tag{4.29}$$

y de hecho la extendemos al conjunto de subconjuntos de estados, $\mathcal{P}(Q)$, haciendo

$$\forall P', Q' \subset Q : \ P' \equiv_{Ind} Q' \ \Leftrightarrow \ \exists p \in P', q \in Q' : p \equiv_{Ind} q \tag{4.30}$$

Y, también a como se hizo en el caso de los autómatas finitos, se tiene inducida una relación de indistinguibilidad en el diccionario de entrada:

$$\forall \sigma_1, \sigma_2 \in Ent^* : \ \sigma_1 \equiv_{Ind,GT} \sigma_2 \ \Leftrightarrow \ T_{GT}(\sigma_1) \equiv_{Ind} T_{GT}(\sigma_2) \tag{4.31}$$

donde $\forall \sigma : T_{GT}(\sigma) = tran^*_{GT}(q_0, \sigma)$.

Observación 4.4.3 *$\forall \sigma_1, \sigma_2 \in Ent^*$ se cumplen las relaciones siguientes:*

1. $\forall \tau \in Ent^* : \tau \in \Gamma(\sigma_1) \cap \Gamma(\sigma_2) \ \Leftrightarrow \ \sigma_1\tau \in L(GT) \& \sigma_2\tau \in L(GT)$.
2. $\sigma_1 \equiv_{Ind,GT} \sigma_2 \Leftrightarrow \Gamma(\sigma_1) = \Gamma(\sigma_2)$.

Observación 4.4.4 *Las aseveraciones siguientes dan una alternativa para construir autómatas mínimos.*

1. *La relación "kernel de Γ" en el diccionario Ent^*,*

   $$\sigma_1 \equiv_\Gamma \sigma_2 \Leftrightarrow \Gamma(\sigma_1) = \Gamma(\sigma_2),$$

   coincide con la relación "$\equiv_{Ind,GT}$" y, consecuentemente, es un refinamiento de la relación que define al monoide del autómata.

2. *El cociente $GT_\Gamma = (Ent^* / \equiv_\Gamma)$ tiene una estructura de autómata finito al asociarle*

   (a) las transiciones $(\Gamma(\sigma), t) \mapsto \Gamma(\sigma t)$, y

   (b) los estados finales $\{\Gamma(\sigma) | nil \in \Gamma(\sigma)\}$.

   GT_Γ es equivalente a la gráfica de transición dada GT. Además, si S_{GT} es el autómata que coincide con el monoide de GT *al dotar a éste de una estructura de autómata, entonces la función*

   $$\begin{array}{rcl} S_{GT} & \to & GT_\Gamma \\ [\sigma] & \mapsto & \Gamma(\sigma) \end{array}$$

   es un homomorfismo de autómatas.

3. *La función $Q \to GT_\Gamma$, $q \mapsto \Gamma(\sigma)$, donde σ es tal que $q \in T_{GT}(\sigma)$ es también un homomorfismo de gráficas de transición.*

4. *El índice de la relación $\equiv_\Gamma$ está acotado por el número de estados en* GT.

Ahora bien, se tiene una correspondencia entre la imagen de Γ, o sea el conjunto de estados de GT_Γ, y las partes de Q que son accesibles desde el estado inicial q_0 en la gráfica.

Por tanto, podemos calcular al autómata cociente GT_Γ de manera similar a como se calcula el autómata equivalente a una gráfica de transición. En la Figura 4.1 presentamos el seudocódigo de este procedimiento. Si $Q = \{q_0, \ldots, q_{n-1}\}$ es el conjunto de estados de la gráfica, la lista $\mathbf{q} = i_0 \cdots i_{n-1} \in Dos^n$ denota al subconjunto Q' tal que para todo j, $q_{i_j} \in Q' \Leftrightarrow i_j = 1$.

Habiendo calculado GT_Γ, supongamos que

$$GT_\Gamma = \{\Gamma_1, \ldots, \Gamma_{n_\Gamma}\},$$

donde $\Gamma_1 = \Gamma(nil)$ corresponde a la palabra vacía. En este caso, las ecuaciones 4.28 dan un sistema de n_Γ ecuaciones con n_Γ "incógnitas" $\Gamma_0, \ldots, \Gamma_{n_\Gamma}$ de la forma

$$\forall i \leq n_\Gamma : \quad \Gamma_i = \sum_{j=1}^{n_\Gamma} \sigma_{ij}\Gamma_j + b_i \tag{4.32}$$

Input: A transition graph $GT = (Q, Ent, tran, q_0, F)$
Output: The set of states of GT_Γ.

Tst: List of *tested* words.
TBTst: List of *to-be-tested* words.
Final: List of *final* states in GT_Γ.

{ $Tst := nil$; $TBTst := [nil]$;
while $TBTst$ is not empty **do**
{ $Pop(\sigma, TBTst)$; $\mathbf{q} := T_{GT}(\sigma)$;
for each $e \in Ent$ **do**
{ $\mathbf{p} := \bigcup_{q \in \mathbf{q}} tran_{GT}(q, e)$;
if $\forall \tau \in Tst \cup TBTst : \mathbf{p} \neq T_{GT}(\tau)$ **then**
{$TBTst := Append(TBTst, [\sigma e])$;
if $\mathbf{p} \cap F \neq \emptyset$ **then** $\sigma e \in Final$ }};
$Tst := Append(Tst, [\sigma])$
} ;
GT_Γ consists of the states represented by words in Tst
}

Figura 4.1: Cálculo de la imagen de Γ para una gráfica de transición dada.

donde $\forall i, j : \sigma_{ij}$ es la suma de símbolos que en GT_Γ hacen transitar a Γ_i hacia Γ_j:

$$\sigma_{ij} = \{t \in Ent | (\Gamma_i, t) \mapsto \Gamma_j\}$$

y

$$b_i = \begin{cases} \mathbf{1} & \text{si } nil \in \Gamma_i, \\ \mathbf{0} & \text{en otro caso.} \end{cases}$$

Puesto de manera matricial, el sistema de ecuaciones 4.32 queda de la forma

$$\mathbf{\Gamma} = \mathbf{A\Gamma} + \mathbf{b},$$

y por la manera en la que se construyó, la matriz $\mathbf{A}$ no posee la CPV, por tanto

$$\mathbf{\Gamma} = \mathbf{A}^*\mathbf{b}.$$

En la primera entrada de este vector, es decir, la correspondiente a Γ_1, se obtiene una expresión regular que describe al lenguaje reconocido por la gráfica de transición.

En resumen, se tiene la

Proposición 4.4.1 (Kleene) *El lenguaje reconocido por cualquier gráfica de transición se puede expresar mediante una expresión regular. Es decir, todo lenguaje regular es formalmente regular.*

Así pues, para una gráfica de transición $GT = (Q, Ent, tran, q_0, F)$ con sólo dos estados q_1 y q_2. Hagamos

$$\begin{aligned} a_{ij} &= \{e \in Ent | tran(q_i, e) = q_j\} \\ b_i &= \begin{cases} \mathbf{1} & \text{si } q_i \in F, \\ \mathbf{0} & \text{si } q_i \notin F. \end{cases} \end{aligned}$$

Entonces, de la ec. (4.27) obtenemos: $L(GT) = [a_{11}^* a_{12} a_{22}^* a_{21}]^* a_{11}^* b_1 + a_{12} a_{22}^* b_2$.

Ejemplo:

Sea $GT = (Q, Ent, tran, q_0, F)$ la gráfica de transición vista anteriormente, donde $Q = \{0,1,2,3\}$, $Ent = \{0,1\}$, $q_0 = 0$, $F = \{0\}$ y las transiciones son $tran = \left\{ \begin{array}{lll} (0\ \lambda\ 1) & (2\ 0\ 1) & (2\ 1\ 2) \\ (1\ \lambda\ 2) & (1\ 0\ 3) & (3\ 1\ 3) \\ (3\ \lambda\ 0) & & \end{array} \right\}$.

Al aplicar el algoritmo 4.1 obtenemos la tabla siguiente:

σ	$T_{GT}(\sigma)$	$\tau:\ \Gamma(\tau) = \Gamma(\sigma)$	*¿Final?*
nil	1110	–	Sí
0	1111	–	Sí
1	0010	–	No
00	1111	0	Sí
01	1111	0	Sí
10	0110	–	No
11	0010	1	No
100	1111	0	Sí
101	0010	1	No

Por tanto,

$$GT_\Gamma = \{\Gamma(nil), \Gamma(0), \Gamma(1), \Gamma(10)\} = \{\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4\},$$

y se obtiene el sistema de ecuaciones

$$\begin{array}{lllllll} \Gamma_1 & = & \Gamma(nil) & = & 0\Gamma(0) + 1\Gamma(1) + nil & = & 0\Gamma_2 + 1\Gamma_3 + nil \\ \Gamma_2 & = & \Gamma(0) & = & 0\Gamma(00) + 1\Gamma(01) + nil & = & 0\Gamma_2 + 1\Gamma_2 + nil \\ \Gamma_3 & = & \Gamma(1) & = & 0\Gamma(10) + 1\Gamma(11) & = & 0\Gamma_4 + 1\Gamma_3 \\ \Gamma_4 & = & \Gamma(10) & = & 0\Gamma(100) + 1\Gamma(101) & = & 0\Gamma_2 + 1\Gamma_3 \end{array}$$

Puesto de manera matricial el anterior sistema de ecuaciones queda de la forma[3]

$$\begin{bmatrix} \Gamma_1 \\ \Gamma_2 \\ \Gamma_3 \\ \Gamma_4 \end{bmatrix} = \begin{bmatrix} \mathbf{0} & 0 & 1 & \mathbf{0} \\ \mathbf{0} & (0+1) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & 1 & 0 \\ \mathbf{0} & 0 & 1 & \mathbf{0} \end{bmatrix} \cdot \begin{bmatrix} \Gamma_1 \\ \Gamma_2 \\ \Gamma_3 \\ \Gamma_4 \end{bmatrix} + \begin{bmatrix} \mathbf{1} \\ \mathbf{1} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}$$

El lenguaje de la gráfica de transición quedará expresado por Γ_1 donde

$$\begin{bmatrix} \Gamma_1 \\ \Gamma_2 \\ \Gamma_3 \\ \Gamma_4 \end{bmatrix} = \begin{bmatrix} \mathbf{0} & 0 & 1 & \mathbf{0} \\ \mathbf{0} & (0+1) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & 1 & 0 \\ \mathbf{0} & 0 & 1 & \mathbf{0} \end{bmatrix}^* \begin{bmatrix} \mathbf{1} \\ \mathbf{1} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}$$

Al calcular la operación "estrella", se obtiene

$$\begin{bmatrix} \mathbf{0} & 0 & 1 & \mathbf{0} \\ \mathbf{0} & (0+1) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & 1 & 0 \\ \mathbf{0} & 0 & 1 & \mathbf{0} \end{bmatrix}^* = \begin{bmatrix} \mathbf{1} & 0\,(1+0)^* + 100\,(1+0)^* & 1\,(1^*01)^*\,1^* & 10 \\ \mathbf{0} & (1+0)^* & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & (1^*01)^*\,1^* & 0 \\ \mathbf{0} & 0\,(1+0)^* & 11^* & (11^*0)^* \end{bmatrix}$$

y consecuentemente, el lenguaje reconocido por la gráfica de transición es

$$\begin{array}{rcl} L(GT) & = & \mathbf{1} + 0\,(1+0)^* + 100\,(1+0)^* \\ & = & \mathbf{1} + (0+100)\,(1+0)^* \end{array}$$

[3] Como mero recordatorio los símbolos en negritas **0** y **1** representan respectivamente al conjunto vacío y a la palabra vacía, y los románicos 0 y 1 representan a los correspondientes símbolos del alfabeto de entrada.

Así pues, para una gráfica de transición $GT = (Q, Ent, tran, q_0, F)$ con sólo dos estados q_1 y q_2. Hagamos

$$\begin{aligned} a_{ij} &= \{e \in Ent | tran(q_i, e) = q_j\} \\ b_i &= \begin{cases} \mathbf{1} & \text{si } q_i \in F, \\ \mathbf{0} & \text{si } q_i \notin F. \end{cases} \end{aligned}$$

Entonces, de la ec. (4.27) obtenemos: $L(GT) = [a_{11}^* a_{12} a_{22}^* a_{21}]^* a_{11}^* b_1 + a_{12} a_{22}^* b_2$.

Ejemplo:

Sea $GT = (Q, Ent, tran, q_0, F)$ la gráfica de transición vista anteriormente, donde $Q = \{0,1,2,3\}$, $Ent = \{0,1\}$, $q_0 = 0$, $F = \{0\}$ y las transiciones son $tran = \left\{ \begin{array}{lll} (0\ \lambda\ 1) & (2\ 0\ 1) & (2\ 1\ 2) \\ (1\ \lambda\ 2) & (1\ 0\ 3) & (3\ 1\ 3) \\ (3\ \lambda\ 0) & & \end{array} \right\}$.

Al aplicar el algoritmo 4.1 obtenemos la tabla siguiente:

σ	$T_{GT}(\sigma)$	$\tau:\ \Gamma(\tau) = \Gamma(\sigma)$	*¿Final?*
nil	1110	–	Sí
0	1111	–	Sí
1	0010	–	No
00	1111	0	Sí
01	1111	0	Sí
10	0110	–	No
11	0010	1	No
100	1111	0	Sí
101	0010	1	No

Por tanto,

$$GT_\Gamma = \{\Gamma(nil), \Gamma(0), \Gamma(1), \Gamma(10)\} = \{\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4\},$$

y se obtiene el sistema de ecuaciones

$$\begin{array}{lllllll} \Gamma_1 &=& \Gamma(nil) &=& 0\Gamma(0) + 1\Gamma(1) + nil &=& 0\Gamma_2 + 1\Gamma_3 + nil \\ \Gamma_2 &=& \Gamma(0) &=& 0\Gamma(00) + 1\Gamma(01) + nil &=& 0\Gamma_2 + 1\Gamma_2 + nil \\ \Gamma_3 &=& \Gamma(1) &=& 0\Gamma(10) + 1\Gamma(11) &=& 0\Gamma_4 + 1\Gamma_3 \\ \Gamma_4 &=& \Gamma(10) &=& 0\Gamma(100) + 1\Gamma(101) &=& 0\Gamma_2 + 1\Gamma_3 \end{array}$$

Puesto de manera matricial el anterior sistema de ecuaciones queda de la forma[4]

$$\begin{bmatrix} \Gamma_1 \\ \Gamma_2 \\ \Gamma_3 \\ \Gamma_4 \end{bmatrix} = \begin{bmatrix} \mathbf{0} & 0 & 1 & \mathbf{0} \\ \mathbf{0} & (0+1) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & 1 & 0 \\ \mathbf{0} & 0 & 1 & \mathbf{0} \end{bmatrix} \cdot \begin{bmatrix} \Gamma_1 \\ \Gamma_2 \\ \Gamma_3 \\ \Gamma_4 \end{bmatrix} + \begin{bmatrix} \mathbf{1} \\ \mathbf{1} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}$$

El lenguaje de la gráfica de transición quedará expresado por Γ_1 donde

$$\begin{bmatrix} \Gamma_1 \\ \Gamma_2 \\ \Gamma_3 \\ \Gamma_4 \end{bmatrix} = \begin{bmatrix} \mathbf{0} & 0 & 1 & \mathbf{0} \\ \mathbf{0} & (0+1) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & 1 & 0 \\ \mathbf{0} & 0 & 1 & \mathbf{0} \end{bmatrix}^* \begin{bmatrix} \mathbf{1} \\ \mathbf{1} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}$$

Al calcular la operación "estrella", se obtiene

$$\begin{bmatrix} \mathbf{0} & 0 & 1 & \mathbf{0} \\ \mathbf{0} & (0+1) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & 1 & 0 \\ \mathbf{0} & 0 & 1 & \mathbf{0} \end{bmatrix}^* = \begin{bmatrix} \mathbf{1} & 0\,(1+0)^* + 100\,(1+0)^* & 1\,(1^*01)^*\,1^* & 10 \\ \mathbf{0} & (1+0)^* & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & (1^*01)^*\,1^* & 0 \\ \mathbf{0} & 0\,(1+0)^* & 11^* & (11^*0)^* \end{bmatrix}$$

[4] Como mero recordatorio los símbolos en negritas **0** y **1** representan respectivamente al conjunto vacío y a la palabra vacía, y los románicos 0 y 1 representan a los correspondientes símbolos del alfabeto de entrada.

y consecuentemente, el lenguaje reconocido por la gráfica de transición es

$$\begin{aligned} L(GT) &= \mathbf{1} + 0\,(1+0)^* + 100\,(1+0)^* \\ &= \mathbf{1} + (0+100)\,(1+0)^* \end{aligned}$$

4.5 Expresiones regulares y sistemas de planeación

Se presenta un problema clásico para los Sistemas de Planeacion: *El Problema de los Bloques*. Se resuelve este problema por completo modelándolo como un autómata finito y se calcula a todas sus posibles soluciones.

4.5.1 Introducción

Los *Sistemas de Planeación* generan automáticamente procedimientos para resolver problemas bien planteados. En ellos , se describe al *Universo del Problema*, el cual consiste de sus objetos y de las relaciones entre ellos. Se describe también a *las acciones legales*, o *movimientos*. Cada una de ellas involucra a un conjunto de *precondiciones* que se deben de cumplir para aplicar la acción, y otro de *postcondiciones* que se cumplirán tras haber aplicado la acción. Una acción sólo se aplica a algunos estados, y al hacerlo, los transforma en otros nuevos. Si la acción a se puede aplicar al estado e escribiremos $a \downarrow e$ y denotaremos por $a(e)$ al estado que se obtiene de e al aplicar a. Una acción a_1 es *sucesiva* de otra a_0 respecto al estado e si se cumple que $a_1 \downarrow a_0(e)$. Un *plan*, aplicado a un estado e, es una sucesión de acciones $a_0, \ldots, a_k$ tal que $a_0 \downarrow e$, a_1 es sucesiva de a_0 respecto a e y además a_{i+1} es sucesiva de a_i respecto a $a_{i-1} \circ a_{i-2} \circ \cdots \circ a_0(e)$, $i = 2, \ldots, k-1$.

El problema de cualquier Sistema de Planeación consiste en encontrar, para un estado *inicial* e y uno *meta* f dados, un plan $a_0, \ldots, a_k$ tal que $f = a_k \circ a_{k-1} \circ \cdots \circ a_0(e)$, siempre que exista un tal plan, o bien indicar que no existe, en otro caso.

El *Problema de los Bloques* tiene como universo a un conjunto de $n+1$ objetos, de los cuales n son *bloques* y el último es el *piso*. Como relaciones sólo tiene a dos:

- $sobre(x, y)$ "el bloque x está sobre el objeto y", y
- $libre(x)$ "el objeto x está libre para recibir un bloque encima de él".

Las acciones legales son instancias de $mueve(x, y, z)$ "mueve el bloque x del objeto y al objeto z", con

- *precondiciones*:
$$libre(x) \& libre(z) \& sobre(x, y)$$
y
- *postcondiciones*:
$$libre(x) \& libre(y) \& sobre(x, z).$$

Cada estado se especifica como una conjunción de enunciados atómicos, o *átomos*, los cuales son instancias de relaciones con objetos.

Un primer criterio para generar planes de manera automática es poner a los átomos de una meta en una lista, considerarlos como "submetas", e ir lográndolos uno a uno. Es necesario preservar lo logrado al tratar de lograr un nuevo átomo. Si acaso no fuera posible, se permuta a la lista de átomos y se repite el proceso.

A veces este criterio no logra estados metas, aún cuando efectivamente fueren alcanzables. La anomalía de Sussman es un ejemplo: Con tres bloques $1, 2, 3$, estado inicial

$$sobre(3, 1) \& sobre(2, piso)$$

y meta

$$sobre(1, 2) \& sobre(2, 3),$$

la estrategia anterior no logra encontrar un plan. En cualquiera de las dos permutaciones de la meta uno se ve obligado a deshacer logros ya obtenidos.

El problema de planeación ha sido tratado con diversos enfoques, basados, en general, en *deducción automática*. Alternativamente, consideraremos aquí un enfoque de tipo combinatorio. Veremos que es posible calcular no sólo una solución, sino a todas las existentes. Primeramente, calcularemos el número de estados en el Problema de Bloques (PB). Presentamos luego una solución del correspondiente problema de planeación con técnicas de autómatas finitos. Finalmente observamos que, bien que el PB (de hecho cualquier otro problema para los Sistemas de Planeación), se resuelve por completo con estas técnicas, los algoritmos utilizados son tan complejos que aún queda lejos su implementación práctica.

4.5.2 El mundo de los bloques

Consideremos n bloques, enumerados con los números naturales $\mathbf{n} = [1, 2, \ldots, n]$. Asociémosle al *piso* el número 0. El *mundo* es $\mathbf{n{+}1} = \{0\} \cup \mathbf{n}$.

En cada *estado*, o *configuración del universo*, tendremos un cierto número de columnas, cada una de ellas constituída por bloques apilados. Ya que cada bloque está bien sobre el piso o bien sobre algún otro bloque, a cada configuración podemos asociarle una función $apoyo : \mathbf{n} \to \mathbf{n{+}1}$. Naturalmente, si $apoyo(i) = j$ tendremos que el bloque i está sobre el objeto j. Así pues,tendremos las siguientes propiedades para una tal función *apoyo*:

1. "Sólo el piso puede ser apoyo de más de un bloque". En símbolos

$$\forall i, j \in \mathbf{n} : apoyo(i) = apoyo(j) \& i \neq j \;\Rightarrow\; apoyo(i) = 0.$$

2. "El número de columnas en un estado coincide con el de bloques apoyados en el piso". En símbolos:

$$número\_columnas = card[apoyo^{-1}0].$$

3. "Las columnas no son aros". En símbolos:

$$\forall i \in \mathbf{n} \forall k \geq 1 : apoyo^k(i) \neq i.$$

4. "Cada columna tiene una base en el piso". En símbolos:

$$\forall i \in \mathbf{n} \exists k \geq 1 : apoyo^k(i) = 0.$$

5. "Cada bloque está sobre su apoyo". En símbolos:

$$\forall i e \mathbf{n} : sobre(i, apoyo(i)).$$

Los estados del universo corresponden de manera biunívoca con las funciones de apoyo. Por tanto, contar los posibles estados en el PB equivale a contar las funciones de apoyo.

Sea pues

$$A_n = \{apoyo : \mathbf{n} \to \mathbf{n{+}1} \text{ cumple con las propiedades } 1 - 5\}.$$

Para una función $apoyo \in A_n$, sea k el número de columnas que define y $\forall j \leq k$ sea h_j el número de bloques en la columna j-ésima. Supondremos a las columnas ordenadas de manera no-decreciente en cuanto a sus alturas. Entonces tendremos que $0 \leq h_1 \leq \ldots \leq h_k$ y además $h_1 + \ldots + h_k = n$. En este sentido, diremos que el estado determina a la sucesión $(h_1, \ldots, h_k)$.

Surge pues la necesidad de analizar al conjunto de sucesiones

$$H_{n,k} = \{(h_1, \ldots, h_k) \in \mathbf{n}^k | 0 \leq h_1 \leq \ldots \leq h_k \& \sum_{j=1}^{k} h_j = n\}.$$

Observamos que dos estados $apoyo_1$ y $apoyo_2$ determinan a una misma sucesión $(h_1, \ldots, h_k) \in H_{n,k}$ si y sólo si el estado $apoyo_1$ difiere del estado $apoyo_2$ tan sólo por una permutación de los n bloques en el Mundo. Tenemos $n!$ permutaciones tales.

Ahora, para una sucesión $(h_1, \ldots, h_k) \in H_{n,k}$ podemos tener varias de sus entradas iguales, en otras palabras, en cada estado que determine a la sucesión, varias de sus columnas pueden tener una misma altura. Tomemos pues el conjunto de valores distintos que aparecen en ella. Sea éste $con(\mathbf{h}) = \{h \in \mathbf{n} | \exists i \leq k : h = h_i\}$. Para cada $h \in con(\mathbf{h})$, contemos sus repeticiones en $\mathbf{h}$, sea pues $repet(h) = card\{i \leq k | h_i = h\}$. Convengamos también en identificar columnas de igual tamaño. De esta manera, $\forall h \in con(\mathbf{h})$, $[repet(h)]!$ permutaciones de columnas de altura h quedarán identificadas. Luego, habrá $c(\mathbf{h}) = \prod_{h \in con(\mathbf{h})} [repet(h)]!$ estados identificados, cada uno de los cuales determina a la sucesión $\mathbf{h}$.

Hechas las observaciones anteriores, un minuto de reflexión basta para convencerse de que vale la relación siguiente:

$$card(A_n) = n! \sum_{k=1} n \sum_{\mathbf{h} \in H_{nk}} \frac{1}{c(\mathbf{h})} \tag{4.33}$$

Pasemos ahora a encontrar una expresión equivalente cuya evaluación sea más sencilla.

Dados $n > 1$ y $k \leq n$, ¿de cuántas formas podemos expresar a n como una suma de k sumandos enteros positivos?. En otras palabras, si definimos

$$S_{nk} = \{(h_1, \ldots, h_k) \in \mathbf{n}^k | \sum_{j=1}^{k} h_j = n\},$$

¿cuál es la cardinalidad de S_{nk}?

Observemos que, en su representación unaria, n se escribe como una sarta de n 1's, es decir, como n "palitos contiguos". En esa sarta tendremos $n-1$ separaciones entre palitos adyacentes. Si elegimos a $k-1$ de tales separaciones para introducir otras tantas marcas separadoras, los n palitos quedarán agrupados en k hatos contiguos, cada uno con al menos un palito, y la suma de todos ellos nos da, en efecto, los n palitos. Así pues, tendremos que $card(S_{nk}) = \binom{n-1}{k-1}$. Ahora, cada sucesión $\mathbf{h} \in H_{nk}$ tiene a sus entradas ordenadas de manera no-decreciente. Si omitimos esta exigencia de orden, un segundo minuto de reflexión nos convencerá de que la sucesión $\mathbf{h}$ dará origen a $\frac{k!}{c(\mathbf{h})}$ elementos distintos en S_{nk}. Al conjuntar estas dos observaciones, tenemos que $\sum_{\mathbf{h} \in H_{nk}} \frac{k!}{c(\mathbf{h})} = card(S_{nk})$ y consecuentemente

$$\sum_{\mathbf{h} \in H_{nk}} \frac{1}{c(\mathbf{h})} = \frac{1}{k!} \binom{n-1}{k-1} \tag{4.34}$$

Una mera manipulación algebraica da $\frac{1}{k!}\binom{n-1}{k-1} = \frac{1}{n(k-1)!}\binom{n}{k}$.

Así pues, logramos nuestro objetivo al sustituir 4.34 en 4.33:

$$card(A_n) = (n-1)! \sum_{k=1}^{n} \frac{1}{(k-1)!} \binom{n}{k}$$

Lo que equivale a que

$$card(A_n) = \sum_{k=0}^{n-1} k! \binom{n-1}{k} \binom{n}{k} =: a_n \tag{4.35}$$

Esta última relación nos da la naturaleza explosiva del PB. En la Tabla 4.3 presentamos algunos números de estados resultantes con diversos números de bloques.

Se ve que $card(A_n) = \Omega(10^n)$. Este crecimiento rápido del número de estados es lo que, en la práctica, hace intratable al PB con este enfoque. Sin embargo, aún nos dedicaremos a él para ilustrar el uso de expresiones regulares.

4.5.3 PB visto como un AF

Sea $n > 1$. Sea $(e_k)_{0 \leq k \leq a_n - 1}$ el conjunto de estados correspondientes al PB con n bloques, PB_n. Según vimos anteriormente $a_n = \sum_{k=0}^{n-1} k! \binom{n-1}{k} \binom{n}{k}$.

No. bloques	*No. estados*	*No. bloques*	*No. estados*
1	1	11	824073141
2	3	12	12470162233
3	13	13	202976401213
4	73	14	3535017524403
5	501	15	65573803186921
6	4051	16	1290434218669921
7	37633	17	26846616451246353
8	394353	18	588633468315403843
9	4596553	19	13564373693588558173
10	58941091	20	327697927886085654441

Tabla 4.3: Estados en el PB en función de los bloques.

Edo	m_{10}	m_{12}	m_{20}	m_{21}
e_0	e_0	e_1	e_0	e_2
e_1	e_0	e_1	e_3	e_3
e_2	e_3	e_3	e_0	e_2
e_3	e_3	e_3	e_3	e_3

Tabla 4.4: Matriz de transición del autómata correspondiente a 2 bloques.

$\forall i \in \mathbf{n} \forall j \in \mathbf{n+1} - \{i\}$ codificaremos a la instrucción "Colóquese el bloque i sobre el objeto j" mediante un símbolo especial m_{ij}. Claramente, un tal símbolo m_{ij} sólo podrá aplicarse a un estado e_k cuando en este estado ni i ni j tienen un bloque encima, a menos que $j = 0$, es decir, que el objeto j sea precisamente el piso. Introduzcamos entonces un estado más, e_{a_n}, de *inconsistencia*. Sea pues $E = (e_k)_{0 \leq k \leq a_n}$ el conjunto de estados y sea $M = (m_{ij})_{ij \in \mathbf{n} \times \mathbf{n+1}}$ el conjunto de símbolos del autómata correspondiente al PB_n. Las transiciones en él, estarán dadas por la función $T : E \times M \to E$ tal que $\forall (e_k, m_{ij}) \in E \times M$, $T(e_k, m_{ij})$ es bien el estado de inconsistencia e_{a_n}, en el caso que m_{ij} no pueda aplicarse a e_k, o bien es el estado que difiere de e_k tan sólo por la aplicación del movimiento m_{ij} a e_k, cuando ésta es posible.

A guisa de ejemplos, en las tablas 4.4 y 4.5 mostramos las tablas de transición de los autómatas correspondientes a 2 y 3 bloques respectivamente.

Ahora, para el (semi-)autómata $AF_n = (E_n, M_n, T_n)$ correspondiente a PB_n consideremos la matriz $\mathbf{A}_n \in ER(M_n)^{E_n \times E_n}$ tal que $\forall i, j \leq a_n$:

$$a_{ij} = \sum \{m \in M_n | T_n(e_i, m) = e_j\}.$$

Para dos bloques, se tiene

$$\mathbf{A}_2 = \begin{bmatrix} m_{20} + m_{10} & m_{12} & m_{21} & 0 \\ m_{10} & m_{12} & 0 & m_{20} + m_{21} \\ m_{20} & 0 & m_{21} & m_{10} + m_{12} \\ 0 & 0 & 0 & m_{20} + m_{10} + m_{12} + m_{21} \end{bmatrix}$$

Para dos bloques, se tiene la matriz de la tabla 4.6.

Para $\mathbf{A}_2$ se puede calcular su potencia "estrella" por el algoritmo descrito anteriormente. Se obtiene que

$$\mathbf{A}_2^* = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \\ b_{41} & b_{42} & b_{43} & b_{44} \end{bmatrix}$$

donde

$$b_{11} = \left(\left((m_{20} + m_{10})^* m_{12} m_{12}^* m_{10} \right)^* (m_{20} + m_{10})^* m_{21} m_{21}^* \right)^* .$$

Edo	m_{10}	m_{12}	m_{13}	m_{20}	m_{21}	m_{23}	m_{30}	m_{31}	m_{32}
e_0	e_0	e_5	e_3	e_0	e_6	e_1	e_0	e_4	e_2
e_1	e_1	e_{12}	e_{13}	e_0	e_6	e_1	e_{13}	e_{13}	e_{13}
e_2	e_2	e_{13}	e_9	e_{13}	e_{13}	e_{13}	e_0	e_4	e_2
e_3	e_0	e_5	e_3	e_3	e_8	e_{13}	e_{13}	e_{13}	e_{13}
e_4	e_{13}	e_{13}	e_{13}	e_4	e_{13}	e_{10}	e_0	e_4	e_2
e_5	e_0	e_5	e_3	e_{13}	e_{13}	e_{13}	e_5	e_{11}	e_{13}
e_6	e_{13}	e_{13}	e_{13}	e_0	e_6	e_1	e_6	e_{13}	e_7
e_7	e_{13}	e_{13}	e_{13}	e_{13}	e_{13}	e_{13}	e_6	e_{13}	e_7
e_8	e_{13}	e_{13}	e_{13}	e_3	e_8	e_{13}	e_{13}	e_{13}	e_{13}
e_9	e_2	e_{13}	e_9	e_{13}	e_{13}	e_{13}	e_{13}	e_{13}	e_{13}
e_{10}	e_{13}	e_{13}	e_{13}	e_4	e_{13}	e_{10}	e_{13}	e_{13}	e_{13}
e_{11}	e_{13}	e_{13}	e_{13}	e_{13}	e_{13}	e_{13}	e_5	e_{11}	e_{13}
e_{12}	e_1	e_{12}	e_{13}	e_{13}	e_{13}	e_{13}	e_{13}	e_{13}	e_{13}
e_{13}	e_{13}	e_{13}	e_{13}	e_{13}	e_{13}	e_{13}	e_{13}	e_{13}	e_{13}

Tabla 4.5: Matriz de transición del autómata correspondiente a 3 bloques.

$$\mathbf{A}_3 = [\mathbf{A}_{31}\ \mathbf{A}_{32}\ \mathbf{A}_{33}]$$

donde

$$\mathbf{A}_{31} = \begin{bmatrix}
m_{10}+m_{20}+m_{30} & m_{23} & m_{32} & m_{13} & m_{31} & m_{12} \\
m_{20} & m_{10}+m_{23} & 0 & 0 & 0 & 0 \\
m_{30} & 0 & m_{10}+m_{32} & 0 & m_{31} & 0 \\
m_{10} & 0 & 0 & m_{13}+m_{20} & 0 & m_{12} \\
m_{30} & 0 & m_{32} & 0 & m_{20}+m_{31} & 0 \\
m_{10} & 0 & 0 & m_{13} & 0 & m_{12}+m_{31} \\
m_{20} & m_{23} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & m_{20} & 0 & 0 \\
0 & 0 & m_{10} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & m_{20} & 0 \\
0 & 0 & 0 & 0 & 0 & m_{30} \\
0 & m_{10} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

$$\mathbf{A}_{32} = \begin{bmatrix}
m_{21} & 0 & 0 & 0 & 0 & 0 \\
m_{21} & 0 & 0 & 0 & 0 & 0 \\
m_{21} & 0 & 0 & m_{13} & 0 & 0 \\
m_{21} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & m_{23} & 0 \\
0 & 0 & m_{21} & 0 & 0 & m_{31} \\
m_{21}+m_{30} & m_{32} & 0 & 0 & 0 & 0 \\
m_{30} & m_{32} & 0 & 0 & 0 & 0 \\
0 & 0 & m_{21} & 0 & 0 & 0 \\
0 & 0 & 0 & m_{13} & 0 & 0 \\
0 & 0 & 0 & 0 & m_{23} & 0 \\
0 & 0 & 0 & 0 & 0 & m_{31} \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

$$\mathbf{A}_{33} = \begin{bmatrix}
0 & 0 \\
m_{12} & m_{13}+m_{30}+m_{31}+m_{32} \\
0 & m_{12}+m_{20}+m_{21}+m_{23} \\
0 & m_{23}+m_{30}+m_{31}+m_{32} \\
0 & m_{10}+m_{12}+m_{13}+m_{21} \\
0 & m_{20}+m_{21}+m_{23}+m_{32} \\
0 & m_{10}+m_{12}+m_{13}+m_{31} \\
0 & m_{10}+m_{12}+m_{13}+m_{20}+m_{21}+m_{23}+m_{31} \\
0 & m_{10}+m_{12}+m_{13}+m_{23}+m_{30}+m_{31}+m_{32} \\
0 & m_{12}+m_{20}+m_{21}+m_{23}+m_{30}+m_{31}+m_{32} \\
0 & m_{10}+m_{12}+m_{13}+m_{21}+m_{30}+m_{31}+m_{32} \\
0 & m_{10}+m_{12}+m_{13}+m_{20}+m_{21}+m_{23}+m_{32} \\
m_{12} & m_{13}+m_{20}+m_{21}+m_{23}+m_{30}+m_{31}+m_{32} \\
0 & m_{10}+m_{12}+m_{13}+m_{20}+m_{21}+m_{23}+m_{30}+m_{31}+m_{32}
\end{bmatrix}$$

Tabla 4.6: Matriz de coeficientes del sistema de derivadas.

$$
\begin{aligned}
& \left((m_{20}+m_{10})^* m_{12} m_{12}^* m_{10}\right)^* (m_{20}+m_{10})^* \\
b_{12} &= \left(\left((m_{20}+m_{10})^* m_{12} m_{12}^* m_{10}\right)^* (m_{20}+m_{10})^* m_{21} m_{21}^*\right)^* \cdot \\
& m_{12} m_{12}^* \\
b_{13} &= m_{21} m_{21}^* \\
b_{14} &= m_{21}(m_{10}+m_{12})\,(m_{20}+m_{10}+m_{12}+m_{21})^* \\
b_{21} &= m_{10}\,(m_{20}+m_{10})^* + \\
& m_{10}\,(m_{20}+m_{10})^* m_{21} m_{21}^* \left(\left((m_{20}+m_{10})^* m_{12} m_{12}^* m_{10}\right)^* (m_{20}+m_{10})^* m_{21} m_{21}^*\right)^* \\
& \left((m_{20}+m_{10})^* m_{12} m_{12}^* m_{10}\right)^* (m_{20}+m_{10})^* \\
b_{22} &= \left(m_{12}^* m_{10}\,(m_{20}+m_{10})^* m_{12}\right)^* m_{12}^* + m_{10}\,(m_{20}+m_{10})^* m_{21} m_{21}^* \left(\left((m_{20}+m_{10})^* m_{12} m_{12}^* m_{10}\right)^* (m_{20}+m_{10})^* m_{21} m_{2}^*\right. \\
b_{23} &= 0 \\
b_{24} &= (m_{20}+m_{21})\,(m_{20}+m_{10}+m_{12}+m_{21})^* \\
b_{31} &= \left((m_{20}+m_{10})^* m_{12} m_{12}^* m_{10}\right)^* (m_{20}+m_{10})^* \\
b_{32} &= m_{12} m_{12}^* \\
b_{33} &= \left(m_{21}^* \left((m_{20}+m_{10})^* m_{12} m_{12}^* m_{10}\right)^* (m_{20}+m_{10})^* m_{21}\right)^* m_{21}^* \\
b_{34} &= \left(m_{21}^* \left((m_{20}+m_{10})^* m_{12} m_{12}^* m_{10}\right)^* (m_{20}+m_{10})^* m_{21}\right)^* (m_{10}+m_{12})\,(m_{20}+m_{10}+m_{12}+m_{21})^* + m_{21}^* m_{12} m_{12}^* (m_{20} \\
b_{41} &= 0 \\
b_{42} &= 0 \\
b_{43} &= 0 \\
b_{44} &= (m_{20}+m_{10}+m_{12}+m_{21})^*
\end{aligned}
$$

Hemos visto que en teoría es posible utilizar los algoritmos presentados, relativos a los autómatas finitos, para encontrar a todos los posibles planes para pasar de un estado a otro en el Mundo de los Bloques y resolver así cualquier instancia del PB. En la práctica, hemos visto que, incluso para el caso trivial de sólo dos bloques, la solución con este enfoque es difícil. De hecho, la complejidad de este algoritmo crece notablemente, en tiempo y en espacio, con el número n de bloques considerados.

4.6 Minimización de autómatas

Sea *Ent* un alfabeto finito. Según vimos anteriormente la noción de *lenguaje regular* puede presentarse mediante el reconocimiento por autómatas finitos, sean deterministas o no, o mediante la representación por expresiones regulares. Otras presentaciones equivalentes de esta noción se dan en la siguiente:

Proposición 4.6.1 (Teorema de Myhill-Nerode) *Sea $L \subset Ent^*$ un lenguaje arbitrario. Las siguientes aseveraciones son equivalentes a pares:*

1. *L es regular.*
2. *L es la unión de algunas clases de equivalencia de una relación de equivalencia de índice finito, congruente por la derecha con la concatenación de palabras.*
3. *La relación $\equiv_L \subset (Ent^*)^2$ tal que*

$$\forall \sigma, \tau \in Ent^* : \ \sigma \equiv_L \tau \Leftrightarrow \forall \upsilon \in Ent^* \, (\sigma \cdot \upsilon \in L \Leftrightarrow \tau \cdot \upsilon \in L)$$

es una relación de índice finito.

Demostración:

1. $\Rightarrow$ 2. Supongamos L regular. Sea $AF = (Q, Ent, tran, q_0, F)$ un autómata finito tal que $L = L(AF$. Consideremos la función

$$T : \sigma \mapsto T(\sigma) = tran^*(q_0, \sigma),$$

y su kernel:

$$\forall \sigma, \tau \in Ent^* : \ \sigma \equiv_K \tau \Leftrightarrow T(\sigma) = T(\tau).$$

Hemos visto que "$\equiv_K$" es una relación de equivalencia de índice finito (de hecho este índice está acotado por la cardinalidad de Q), congruente por la derecha. Se tiene además

$$L = \bigcup_{T(\sigma) \in F} [\sigma]_{\equiv_K}.$$

Así pues, se cumple 2.

2. $\Rightarrow$ 3. Sea $R \subset (Ent^*)^2$ una relación de equivalencia de índice finito, congruente por la derecha con la concatenación de palabras, tal que para un subconjunto $\mathcal{F} \subset Q/R$ se tiene que $L = \bigcup_{[\sigma] \in \mathcal{F}} [\sigma]$.

Afirmamos que

$$\forall \sigma, \tau \in Ent^* : \ (\sigma R \tau) \Leftrightarrow (\sigma \equiv_L \tau) \tag{4.36}$$

En efecto, si $\sigma R \tau$ entonces, $\forall \upsilon \in Ent^*$, al ser R congruente por la derecha, $\sigma \upsilon R \tau \upsilon$. Como L es la unión de algunas clases de R, esto da que $(\sigma \cdot \upsilon \in L \Leftrightarrow \tau \cdot \upsilon \in L)$. Así pues, $\sigma \equiv_L \tau$.

De la relación 4.36 vemos que toda clase de equivalencia respecto a "$\equiv_L$" es la unión de clases de equivalencia respecto a R. Por tanto el índice de "$\equiv_L$" no puede exceder el de R. Como este último es finito el de "$\equiv_L$" es también finito.

3. $\Rightarrow$ 1. "$\equiv_L$" es una relación de equivalencia de índice finito, congruente por la derecha. Definamos $AF = (Q, Ent, tran, q_0, F)$ haciendo $Q = Ent^*/\equiv_L$, $q_0 = [nil]$, $F = \{[\sigma] | \sigma \in L\}$ y $tran : ([\sigma], e) \mapsto [\sigma e]$. Es evidente que una palabra es reconocida por AF si y sólo si esa palabra está en L. Así pues, $L = L(AF)$ y,consecuentemente, es regular.

Para un autómata finito $AF = (Q, Ent, tran, q_0, F)$ diremos que dos estados $q_1, q_2 \in Q$ son *indistinguibles* si para cualquier palabra $\sigma \in Ent^*$,

$$tran^*(q_1, \sigma) \in F \Leftrightarrow tran^*(q_2, \sigma) \in F.$$

Si dos estados $q_1, q_2 \in Q$ no fueran indistinguibles, entonces habría una palabra $\sigma \in Ent^*$ tal que

$$(tran^*(q_1, \sigma), tran^*(q_2, \sigma) \in (F \times (Q - F)) \cup ((Q - F) \times F).$$

Diremos que tal palabra *distingue* a los estados q_1, q_2.

La relación de indistinguibilidad es una relación de equivalencia. La denotaremos como "$\equiv_I$".

Sea $L = L(AF)$ el lenguaje reconocido por el autómata AF. Para cualesquiera dos palabras $\sigma, \tau \in Ent^*$ rige la equivalencia siguiente

$$[T(\sigma) \equiv_I T(\tau)] \Leftrightarrow [\sigma \equiv_L \tau].$$

Por tanto el autómata $AFM_L = (Ent^*/\equiv_L)$ construído en la demostración de la implicación (3. $\Rightarrow$ 1.) del teorema de Myhill-Nerode 4.6.1 es isomorfo al autómata cociente $AFM_I = (Q^{con}/\equiv_I)$ y, por consiguiente, es una imagen homomorfa de la parte conexa del autómata AF, $h : T(\sigma) \mapsto [\sigma]$ es un homomorfismo.

De hecho, AFM_L es una imagen homomorfa de cualquier autómata que reconozca a L. Así pues, resulta de inmediato la

Proposición 4.6.2 *El autómata mínimo que reconoce a un lenguaje regular L es $AFM_L = (Ent^*/\equiv_L)$.*

El cálculo de AFM_L es equivalente al cálculo de AFM_I, es decir, al cálculo de clases de estados indistinguibles en Q. Para esto observemos lo siguiente:

1. Para una pareja q_1 y q_2, si en cada símbolo $e \in Ent$ la pareja $\{tran(q_1, e), tran(q_2, e)\}$ consta de estados indistinguibles entonces $\{q_1, q_2\}$ es de indistinguibles también.

2. De manera recíproca, si σ distingue a la pareja $\{tran(q_1, e), tran(q_2, e)\}$ entonces $e\sigma$ distingue a la pareja $\{q_1, q_2\}$.

Así pues, para calcular las clases de estados "distinguibles" podemos proceder como sigue:

1. Inicialmente consideremos vacías las listas de parejas *distinguibles* y de parejas *marcadas*. Para cada pareja de estados distintos $\{q_1, q_2\}$, consideremos una lista vacía de parejas *asociadas* a esa pareja.

2. Para cada $q_1 \in F$ y $q_2 \notin F$ incluyamos la pareja $\{q_1, q_2\}$ tanto en la lista de distinguibles, pues los estados lo son con la palabra vacía *nil*, como en la de marcadas.

3. Para cada pareja $\{q_1, q_2\}$ que no haya sido marcada,

   (a) para cada símbolo de entrada $e \in Ent$,

      i. si $\{tran(q_1, e), tran(q_2, e)\}$ es distinguible, digamos con la palabra σ, entonces incluyamos la pareja $\{q_1, q_2\}$ en la lista de distinguibles, pues lo es con la palabra $e\sigma$, e incluyamos a todas las parejas asociadas a $\{q_1, q_2\}$ en la lista de distinguibles, con indicadores a sus correspondientes palabras que los distinguen,

      ii. en otro caso, asociemos la pareja actual $\{q_1, q_2\}$ a la pareja $\{tran(q_1, e), tran(q_2, e)\}$, siempre que esta última conste de dos estados distintos,

   (b) coloquemos a la pareja actual $\{q_1, q_2\}$ en la lista de palabras marcadas.

4. Las parejas que no son distinguibles dan las clases de estados indistinguibles.

4.7 Lema de bombeo

Sea $AF = (Q, Ent, tran, q_0, F)$ un autómata finito con n estados. Una palabra σ de longitud k define una sucesión de estados $S(\sigma)$ de longitud $k+1$ consistente de los estados por los que "pasa" la aplicación de la palabra σ al autómata. Se tiene:

$$\begin{aligned} \sigma = nil &\Rightarrow S(\sigma) = [q_0] \\ \sigma = \sigma_1 e &\Rightarrow S(\sigma) = S(\sigma_1) * [T_{AF}(\sigma)] \end{aligned}$$

Si $k \geq n$ necesariamente al menos un estado ha de aparecer repetido en la sucesión $S(\sigma)$. Por tanto, se puede descomponer a $S(\sigma)$ en tres tramos, $S(\sigma) = S_1 \cdot S_2 \cdot S_3$, de manera tal que S_1 y S_2 son no-vacíos y sus extremos derechos coinciden. Esta descomposición de $S(\sigma)$ corresponde a una descomposición de σ de la forma $\sigma = \sigma_1\sigma_2\sigma_3$ con $long(\sigma_2) > 0$ y $T_{AF}(\sigma_1) = T_{AF}(\sigma_1\sigma_2)$. Así pues, la partícula no-vacía σ_2 está definiendo un bucle en el autómata. Por tanto, si $\sigma = \sigma_1\sigma_2\sigma_3$ fuese reconocida por el autómata, también ha de serlo cualquier palabra de la forma $\sigma_1\sigma_2^*\sigma_3$.

Puesto de manera muy esquemática, se tiene que toda palabra reconocida por un autómata finito cuya longitud exceda al número de estados en el autómata necesariamente ha de contener una partícula no-vacía que se "bombea".

Lema 4.7.1 (de Bombeo) *Si L es un lenguaje regular entonces existe un entero $n \geq 1$ tal que toda palabra $\sigma \in L$ con longitud al menos n admite una descomposición de la forma $\sigma = \sigma_1\sigma_2\sigma_3$ tal que*

- $long(\sigma_2) > 0$,
- $long(\sigma_1\sigma_3) \leq n$, *y*
- $\forall k \geq 0 : \ \sigma_1\sigma_2^k\sigma_3 \in L$.

En símbolos:

$$\forall L \in \{Regulares\}\exists n \geq 1 \forall \sigma \in Ent^* :$$
$$(\sigma \in L)\&(long(\sigma) \geq n) \;\Rightarrow\; \exists \sigma_1, \sigma_2, \sigma_3 : \begin{cases} (\sigma = \sigma_1\sigma_2\sigma_3)\& \\ (long(\sigma_2) > 0)\& \\ (long(\sigma_1\sigma_3) \leq n)\& \\ (\forall k \geq 0 :\; \sigma_1\sigma_2^k\sigma_3 \in L) \end{cases} \tag{4.37}$$

En efecto, si L es regular y es reconocido por un autómata finito AF, entonces el n cuya existencia se asevera en el lema es, precisamente, el número de estados en AF.

La proposición 4.37 es una condición necesaria para los lenguajes regulares. Consecuentemente, cualquier lenguaje que no la satisficiere no puede ser regular.

Ejemplo:

El lenguaje de palabras equilibradas

$$L = \{0^n 1^n | n \geq 0\}$$

no es regular.

En efecto, si lo fuera, existiría $n_0 > 0$ que satisficiera 4.37 (con n_0 en lugar de n).

Sea $n > \lceil \frac{n_0}{2} \rceil$. Entonces habrían de existir $\sigma_1, \sigma_2, \sigma_3$ tales que $0^n 1^n = \sigma_1\sigma_2\sigma_3$ y, para cualquier $k \geq 0$, $\sigma_1\sigma_2^k\sigma_3 \in L$. Un minuto de reflexión basta para ver que esto no es posible.

Ejemplo:

Sea L el lenguaje consistente de las representaciones en binario, sin ceros a la izquierda, de los números primos. L no puede ser regular.

En efecto, supongamos que para todo $\sigma = e_0 \cdots e_{k-1} \in (0+1)^*$

$$1\sigma \in L \;\Leftrightarrow\; 2^k + \sum_{j=0}^{k-1} e_j 2^{k-1-j} \text{ es primo,}$$

y, por un momento, supongamos que L es regular.

El teorema de Euclides sobre la infinidad de los números primos muestra que en L hay cadenas arbitrariamente largas.

Elijamos $n > 0$ que satisfaga 4.37, y consideremos un primo $p > 2^n$, es decir, un primo cuya representación en binario tenga longitud superior a n. Entonces podemos encontrar $\sigma_1, \sigma_2, \sigma_3$ tales que para todo $j \geq 0$, $\left(\sigma_1\sigma_2^j\sigma_3\right)_2$ es un número primo.

Sea $n_i = (\sigma_i)_2$, $i = 1, 2, 3$, el número representado en binario por la partícula σ_i y sea k_i la longitud de σ_i.

Para $j = p$, el número $q = (\sigma_1\sigma_2^p\sigma_3)_2$ ha de ser primo. Se tiene, al agrupar a la representación en binario de q en un primer bloque a la derecha de k_1 bits, p bloques contiguos de k_2 bits hacia la izquierda y uno último de k_3 bits, que

$$\begin{aligned} q &= \left[n_1 2^{pk_2+k_3}\right] + \left[\left[n_2 2^{(p-1)k_2+k_3}\right] + \cdots + \left[n_2 2^{k_2+k_3}\right] + \left[n_2 2^{k_3}\right]\right] + [n_3] \\ &= n_1 2^{pk_2+k_3} + n_2 2^{k_3}\left[2^{(p-1)k_2} + \cdots + 2^{k_2} + 1\right] + n_3 \\ &= n_1 2^{pk_2+k_3} + n_2 2^{k_3}\left[\frac{2^{pk_2}-1}{2^{k_2}-1}\right] + n_3 \end{aligned} \tag{4.38}$$

Ahora, por el Teorema Pequeño de Fermat, $2^{p-1} \equiv 1 \bmod p$. Por tanto, $2^{(p-1)k_2} \equiv 1 \bmod p$, y $2^{pk_2} \equiv 2^{k_2} \bmod p$, pues $2^{pk_2} = 2^{k_2}2^{(p-1)k_2}$.

Sea $s = \left[2^{(p-1)k_2} + \cdots + 2^{k_2} + 1\right]$. Entonces $(2^{k_2}-1)s = 2^{pk_2}-1$. Por tanto, $(2^{k_2}-1)s \equiv (2^{k_2}-1) \bmod p$. Así, módulo p, se tiene las igualdades

$$0 \equiv (2^{k_2}-1)s - (2^{k_2}-1) = (2^{k_2}-1)(s-1).$$

Por consiguiente p debe dividir a $(2^{k_2}-1)$ o a $(s-1)$. No puede dividir a $(2^{k_2}-1)$ pues $2^{k_2} \leq 2^n < p$ ya que $1 \leq k_2 \leq n$. Por tanto p debe dividir a $s-1$. Se sigue que $s \equiv 1 \bmod p$.

Al tomar congruencias módulo p en 4.38 se tiene

$$\begin{aligned} q &\equiv \left[n_1 2^{pk_2+k_3} + n_2 2^{k_3} + n_3\right] \bmod p \\ &\equiv p \bmod p \\ &\equiv 0 \bmod p \end{aligned}$$

es decir, el primo q debe ser divisible por el primo p. Esto es una contradicción y por tanto L no puede ser regular.

4.8 Propiedades de lenguajes regulares

4.8.1 Propiedades de cerradura bajo homomorfismos

Sustituciones

Sean *Ent* y *Ent'* dos lenguajes. Una *sustitución* es una función

$$\begin{aligned} f : Ent &\to \mathcal{P}(Ent') \\ s &\mapsto E_s \end{aligned}$$

que a cada símbolo de *Ent* le asocia un lenguaje en *Ent'*. Naturalmente, una sustitución f se extiende a todo Ent^* haciendo

$$\begin{aligned} f^* : Ent^* &\to \mathcal{P}(Ent') \\ f^*(nil) &= \{nil\} \\ f^*(\sigma e) &= f^*(\sigma) f(e) \end{aligned}$$

Ahora, si $L \subset Ent^*$ es un lenguaje, definimos

$$f^{**}(L) = \bigcup_{\sigma \in L} f^*(\sigma).$$

Por abuso de notación denotaremos a las funciones f^{**} y f^* como f simplemente.

Proposición 4.8.1 *Si $L \subset Ent^*$ es un lenguaje regular y $f : Ent \to \mathcal{P}(Ent')$ es una sustitución, entonces $f(L)$ es también un lenguaje regular, siempre que $\forall e \in Ent$, $f(e) = E_e$ lo sea.*

En efecto, basta ver que $f(L)$ se representa mediante una expresión regular. Pero, como L es regular, existe una expresión regular $E(\mathbf{e}) = E(e_1, \ldots, e_m)$ que lo representa. Ahora bien, para cada $e_i \in Ent$, E_{e_i} es una expresión regular. Por tanto $f(L) = E\left((e_i | E_{e_i})_i\right)$ es una expresión regular.

Imágenes homomorfas

Un *homomorfismo* es un caso particular de una sustitución f en la que, para cada $e_i \in Ent$, E_{e_i} es una mónada, consistente de una sola palabra.

Por lo visto en la sección anterior, tenemos que si $L \subset Ent^*$ es regular y f es un homomorfismo entonces $f(L) \subset (Ent')^*$ también es regular.

Ahora bien, si $K \subset (Ent')^*$ es un lenguaje cualquiera, su *imagen inversa* bajo el homorfismo f es el lenguaje

$$f^{-1}(K) = \{\sigma \in Ent^* | f(\sigma) \in K\}.$$

Es claro que se cumplen las inclusiones siguientes:

$$\begin{aligned} \forall L \subset Ent^* &: L \subset f^{-1}(f(L)) \\ \forall K \subset (Ent')^* &: K \supset f(f^{-1}(K)) \end{aligned}$$

(si, por ejemplo, f fuese inyectiva, entonces valdrían las igualdades en las inclusiones anteriores.)

Proposición 4.8.2 *Las imágenes inversas de lenguajes regulares, bajo homomorfismos, son regulares. En otras palabras, si $K \subset (Ent')^*$ es un lenguaje regular y $f : Ent^* \to (Ent')^*$ es un homomorfismo, entonces $L = f^{-1}(K)$ es también un lenguaje regular.*

En efecto, supongamos que $AF' = (Q, Ent', tran', q_0, F)$ es un autómata finito que reconoce al lenguaje K. Sea $AF = (Q, Ent, tran, q_0, F)$ el autómata finito que coincide con el anterior salvo en que $tran : (q, e) \mapsto tran'^*(q, f(e))$. Es claro que

$$\forall \sigma \in Ent^* : \ \sigma \in L(AF) \Leftrightarrow f(\sigma) \in L(AF').$$

Así pues, AF reconoce al lenguaje $f^{-1}(K)$ y por tanto, éste es regular.

4.9 Ejercicios

1. Escriba expresiones regulares para denotar a cada uno de los siguientes conjuntos de palabras:
   1. Palabras con a lo sumo una pareja de 0's consecutivos y a lo sumo una pareja de 1's consecutivos.
   2. Cadenas en las que toda pareja de 0's contiguos aparece antes de cualquier pareja de 1's contiguos.
   3. Cadenas que no contienen a 101 como subcadena.
   4. Cadenas equilibradas con igual número de 0's y de 1's tales que ningún prefijo de cualquiera de ellas posee más de dos 0's que 1's ni más de dos 1's que 0's.

2. Describa en español a los lenguajes denotados por las siguientes expresiones regulares:
   1. $(11 + 0)^*(00 + 1)^*$.
   2. $(1 + 01 + 001)^*(nil + 0 + 00)$.
   3. $(00 + 11 + (01 + 10)(11 + 00)^*(01 + 10))^*$.

3. Para cada una de las siguientes expresiones regulares, construya un autómata finito que reconozca al lenguaje representado por la expresión regular.
   1. $10 + (0 + 11)0^*1$
   2. $01\left(((10)^* + 111)^* + 0\right)^* 1$

4. Sea $L \subset (a + b)^*$ el lenguaje de palabras que no contienen a la subcadena bb.
   Construya un autómata finito que reconozca a L.
   Encuentre una expresión regular que represente a L.

5. Demuestre las siguientes identidades para cualesquiera dos expresiones regulares E, F:
   1. $(E^*)^* = E^*$
   2. $(nil + E)^* = E^*$
   3. $(E^*F^*)^* = (E + F)^*$

6. Pruebe o refute a cada una de las siguientes ecuaciones:
   1. $(EF + E)^*E = E(FE + E)^*$
   2. $F(EF + F)^*E = EE^*F(EE^*F)^*$
   3. $(E + F)^* = E^* + F^*$

7. Encuentre todas las soluciones X de la ecuación $X = AX + B$, donde A y B son dos expresiones regulares cualesquiera.

8. ¿Cuál de los siguientes lenguajes es regular? Justifique su respuesta:

$$\begin{aligned} L_1 &= \{0^{2n} | n \geq 0\} \\ L_2 &= \{\sigma \in (0+1)^* | \sigma^{rev} = \sigma\} \\ L_3 &= \{\sigma \in (0+1)^* | 000 \text{ no parece en } \sigma\} \end{aligned}$$

9. Demuestre que si L es un lenguaje regular también lo es el lenguaje

$$Saltear(L) = \{e_1 e_3 \cdots e_{2i-1} \cdots e_{2n-1} | e_1 e_2 \cdots e_{2n-1} e_{2n} \in L\}.$$

Sugerencia: Si $AF = (Q, Ent, tran, q_0, F)$ es un autómata finito que reconoce a L considere el autómata no-determinista $AND = (Q, Ent, tran', q_0, F)$ tal que

$$\forall p, q \in Q, e \in Ent : \quad (p, e, q) \in tran' \Leftrightarrow \exists f \in Ent : tran^*(p, ef) = q.$$

10. Demuestre que si L es un lenguaje regular también lo es el lenguaje

$$Ciclo(L) = \{\sigma_2 \sigma_1 | \sigma_1 \sigma_2 \in L\}.$$

Sugerencia: Si $AF = (Q, Ent, tran, q_0, F)$ es un autómata finito que reconoce a L, definamos

$$\begin{aligned} S^{-1}(F) &= \{q \in Q | \exists \sigma : tran^*(q, \sigma) \in F\} \ : \text{ estados que arriban a finales,} \\ S(q_0) &= \{q \in Q | \exists \sigma : tran^*(q_0, \sigma) = q\} \ : \text{ estados accesibles desde el inicial.} \end{aligned}$$

Consideremos un nuevo estado inicial q_{00}. Para cada estado $q \in S(q_0) \cap S^{-1}(F)$ consideremos la gráfica de transición G_q que coincide con AF salvo en lo siguiente:

- deja de considerar como finales a los estados de F,
- deja de considerar como inicial a q_0,
- considera como final e inicial a q_{00},
- tiende las siguientes transiciones *nil*

$$\begin{aligned} q_{00} &\overset{nil}{\mapsto} q \\ q &\overset{nil}{\mapsto} q_{00} \\ \forall q_f \in F : q_f &\overset{nil}{\mapsto} q_0 \end{aligned}$$

A partir de las gráficas G_q, $q \in S(q_0) \cap S^{-1}(F)$, construya una gráfica de transición que reconozca a $Ciclo(L)$.

11. Demuestre que si L es un lenguaje regular también lo es el lenguaje L^{rev} consistente de los reversos de palabras en L.

12. Sea L cualquier subconjunto de 0^*. Demuestre que L^* es un conjunto regular.

13. Demuestre la extensión siguiente del *Lema de bombeo*:

Proposición 4.9.1 *Sea L un lenguaje regular. Entonces existe un $n \in \mathbb{N}$ tal que*

$$\forall \sigma_1, \sigma_2, \sigma_3 : \left[\begin{array}{r} \sigma_1 \sigma_2 \sigma_3 \in L \\ long(\sigma_2) = n \end{array} \right\} \Rightarrow \exists \tau_1, \tau_2, \tau_3 : \left\{ \begin{array}{l} \sigma_2 = \tau_1 \tau_2 \tau_3 \\ long(\tau_2) \geq 1 \\ \forall i \geq 0 : \sigma_1 \tau_1 \tau_2^i \tau_3 \sigma_3 \in L \end{array} \right]$$

14. Utilice el ejercicio anterior para demostrar que el lenguaje $\{0^n1^m2^m | m, n \geq 0\}$ no puede ser regular.

15. ¿Cuál es la relación de equivalencia "$\equiv_L$", definida según el Teorema de Myhill-Nerode, que en $(0+1)^*$ define el lenguaje $L = \{0^n1^n | n \geq 0\}$?

Concluya que L no puede ser regular.

16. Demuestre que el lenguaje $L \subset (0+1)^*$ consistente de las cadenas que tiene más 0's que 1's no es regular.

17. Sea $L \subset (0+1+\cdots+9+.)^*$ el lenguaje de palabras que son prefijos (finitos) de la expansión decimal de π: $3.1, 3.14, 3.141, 3.1415$, etc.

Demuestre que L no es regular.

18. Decida si acaso todo subconjunto de un conjunto regular es también regular.

19. Demuestre que el lenguaje formado por prefijos de palabras en un lenguaje regular es también regular.

20. Sea $Ent = \{a, b\}$ y sea $Ent_1 = \{bab, abba\}$.

Demuestre que todo conjunto que sea regular respecto a Ent_1 también lo es respecto a Ent.

Muestre que el recíproco no se cumple: No todo conjunto regular respecto a Ent lo es respecto a Ent_1.

4.10 Programas

1. *Máquinas de Mealy como "traductores"*: Recordamos que una *máquina de Mealy* es una estructura de la forma

$$MMe = (Q, Ent, Sal, tran, res, q_0)$$

donde, para fines de este ejercicio:

$$\begin{array}{rcl} Q & = & [\![1, n]\!] \text{ es el conjunto de } \textit{estados}, \\ Ent & = & \{a, b, ..., m\text{-ésimo símbolo}\} \text{ es el alfabeto de } \textit{entrada}, \\ Sal & = & \{A, B, ..., M\text{-ésimo símbolo}\} \text{ es el alfabeto de } \textit{salida}, \\ tran & : & \textbf{array } n \times m \textbf{ of } [\![1, n]\!] \quad \text{es la función de } \textit{transición}, \\ res & : & \textbf{array } n \times m \textbf{ of } [\![1, M]\!], \quad \text{es la función de } \textit{respuesta}, \text{ y} \\ q_0 & = & 1 \text{ es el estado } \textit{inicial}. \end{array}$$

Escriba un programa que reciba como entrada los números enteros n, m, M y los arreglos *tran* y *res*.

Posteriormente, para cada cadena $\sigma \in Ent^*$ dada, ha de calcular la cadena de respuesta $res^*(\sigma) \in Sal^*$.

2. *Máquinas de Moore como "traductores"*: Recordamos que una *máquina de Moore* es una estructura de la forma

$$MMo = (Q, Ent, Sal, tran, res, q_0)$$

donde, para fines de este ejercicio:

$$\begin{array}{rcl} Q & = & [\![1, n]\!] \text{ es el conjunto de } \textit{estados}, \\ Ent & = & \{a, b, ..., m\text{-ésimo símbolo}\} \text{ es el alfabeto de } \textit{entrada}, \\ Sal & = & \{A, B, ..., M\text{-ésimo símbolo}\} \text{ es el alfabeto de } \textit{salida}, \\ tran & : & \textbf{array } n \times m \textbf{ of } [\![1, n]\!] \quad \text{es la función de } \textit{transición}, \\ res & : & \textbf{array } [\![1, n]\!] \textbf{ of } [\![1, M]\!], \quad \text{es la función de } \textit{respuesta}, \text{ y} \\ q_0 & = & 1 \text{ es el estado } \textit{inicial}. \end{array}$$

Escriba un programa que reciba como entrada los números enteros n, m, M y los arreglos *tran* y *res*.

Posteriormente, para cada cadena $\sigma \in Ent^*$ dada, ha de calcular la cadena de respuesta $res^*(\sigma) \in Sal^*$.

3. *Equivalencia entre máquinas de Mealy y de Moore*: Escriba un programa que dada una máquina de Mealy, en la representación descrita para el programa 1 de esta lista, calcule la máquina de Moore equivalente, tal

que todos sus estados sean accesibles, y la exprese en la representación descrita para el programa 2 de esta lista.

4. *Indistinguibilidad en máquinas de Moore*:

   1. Dada una máquina de Moore M, dos estados q_i, q_j son *indistiguibles* si para cualquier palabra $\sigma \in Ent^*$ las respuestas de los estados a los que se llega con σ, partiendo de q_i y de q_j, coinciden. Esta es una relación de equivalencia entre estados.

      Escriba un programa que dada una máquina de Moore M calcule al cociente M/Ind.

   2. Dada una máquina de Moore M, dos palabras $\sigma_1, \sigma_2 \in Ent^*$ son *indistiguibles* si los estados a los que llegan, partiendo del inicial, $tran^*(q_0, \sigma_1)$ y $tran^*(q_0, \sigma_2)$ son indistinguibles. Esta es una relación de equivalencia entre palabras.

      Escriba un programa que dada una máquina de Moore M calcule al cociente Ent^*/Ind.

   3. ¿Qué relación hay entre los cocientes encontrados para una misma máquina?

Nota: En lo que sigue, un semiautómata de n estados con m símbolos se representará como una matriz S del tipo $[\![1,n]\!]^{m\times n}$, con la intención obvia de que $\forall i,j:\ s_{ij}$ es el índice del estado al que se llega desde el estado j cuando se aplica el símbolo i.

Un autómata se representará como una pareja (S,F), donde $F \subset [\![1,n]\!]$ es el conjunto de índices de estados finales.

5. *Parte accesible de un autómata finito*: Dado un autómata, calcule la parte accesible del autómata, y renombre, si fuera necesario, al conjunto de estados resultante para expresarlo con la convención que aquí hemos adoptado para representar autómatas.

6. *Homomorfismos de autómatas*: Escriba un programa que dados dos autómatas decida si hay un homomorfismo del primer autómata al segundo.

Si $h : AF_1 \to AF_2$ fuese un homomorfismo entre dos autómatas, diga que dos estados q_i^1, q_j^1 son *equivalentes* si $h(q_i^1) = h(q_j^1)$. Esta es, en efecto, una relación de equivalencia. Calcule sus clases de equivalencia.

7. *Monoides de autómatas deterministas*: Escriba un programa que reciba un autómata determinista y calcule la tabla de multiplicación de su monoide. (Cfr. Sec. 4.2.2 de las "Notas del Curso").

8. *Acciones de grupos libres en monoides*: Suponga dado un monoide $M \in [\![1,n]\!]^{n\times n}$, por su tabla de multiplicación, con unidad, digamos $u = 1$. Para un conjunto no-vacío de elementos $I \subset [\![1,n]\!]$ denotamos por $[I]_M$ al mínimo submonoide, del monoide M, que contiene al conjunto I. $[I]_M$ es el *submonoide generado* por I en M.

Si I consta de un único elemento, digamos $I = \{a\}$, $a \neq u$, el submonoide $[I]_M$ se calcula como sigue:

1. Inicialmente hágase $[I]_M = \{1\} = \{a^0\}$, y $PorProbar = I$.
2. Mientras que $PorProbar \neq \emptyset$, sáquese el primer elemento x, digamos $x = a^i$, que quede en $PorProbar$ y añádasele a $[I]_M$. Tómese ahora $y = x \cdot a = a^{i+1}$. Si y no aparece aún en $[I]_M$ colóquese y en la lista $PorProbar$.
3. Cuando no queden elementos por probar, el submonoide generado por a queda en $[I]_M$ y a partir de ahí es posible construir la tabla de multiplicación de $[\{a\}]_M$.

Si I consta de más de un elemento, digamos $I = \{a_1 \ldots, a_{k-1}, a_k\}$, $\forall i \leq k:\ a_i \neq u$, el submonoide $[I]_M$ se calcula como sigue:

1. Inicialmente hágase $[I]_M = [\{a_1 \ldots, a_{k-1}\}]_M$, el submonoide generado por los primeros $k-1$ elementos, y $PorProbar = \{a_k\}$.
2. Mientras que $PorProbar \neq \emptyset$, sáquese el primer elemento x, que quede en $PorProbar$. Si acaso ya estuviese en $[I]_M$ termínese el proceso sin más. En otro caso, añádasele a $[I]_M$ y para cada elemento z en $[I]_M$, tómese $y_{der} = z \cdot x$ e $y_{izq} = x \cdot z$. Si y_{der} no aparece ni en $[I]_M$ ni en $PorProbar$, colóquese y_{der} en la lista $PorProbar$. Si y_{izq} no aparece ni en $[I]_M$ ni en $PorProbar$, colóquese y_{izq} en la lista $PorProbar$.

3. Cuando no queden elementos por probar, el submonoide generado por I queda en $[I]_M$ y a partir de ahí es posible construir la tabla de multiplicación de $[I]_M$.

Escriba un programa que dado un monoide en M encuentre un conjunto mínimo de elementos $I = \{a_1 \ldots, a_{k-1}, a_k\}$ tal que $M = [I]_M$.

9. *Autómatas de máximos monoides*: (Este programa se ha de apoyar de manera esencial en el programa anterior en esta lista.) Dado n considere un conjunto Q de n estados, para fijar ideas, $Q = [\![1, n]\!]$. Sea $\mathcal{F}_n = \left(f_i^{(n)}\right)_{i=1}^{n^n}$ una enumeración de todas las funciones $Q \to Q$. $\mathcal{F}_n$ es un monoide con la composición de funciones como operación. Sea $m = n^n$ y sea A_n un subconjunto de m símbolos.

Consideremos el semiautómata $\mathcal{A}_n = (Q, A_n, tran, 1)$ tal que $tran : (q, a_{in}) \mapsto f_{in}(q)$. Entonces su monoide, naturalmente coincide con $\mathcal{F}_n$ y posee n^n elementos.

Escriba un programa que reduzca el semiautómata $\mathcal{A}_n$ a uno $\mathcal{A}'_n$ sobre un alfabeto mínimo $A'_n \subset A_n$ tal que el monoide del semiautómata reducido $\mathcal{A}'_n$ coincida también con $\mathcal{F}_n$.

10. *Indistinguibilidad de estados en autómatas*: Para un autómata $AF = (Q, Ent, tran, q_0, F)$ se tiene la relación en Q:

$$\forall p, q \in Q : \quad p \equiv_{Ind} q \quad \Leftrightarrow \quad \forall \sigma \in Ent^* : \ [tran^*(p, \sigma) \in F \quad \Leftrightarrow \quad tran^*(q, \sigma) \in F] \tag{4.39}$$

Escriba un programa que dado un autómata calcule el autómata cociente correspondiente a esta relación. Extienda este programa para que dado un autómata calcule el mínimo autómata equivalente.

11. *Monoides de autómatas no-deterministas*: Escriba un programa que reciba un autómata no-determinista y calcule la tabla de multiplicación de su monoide. (Cfr. Sec. 4.3.2 de las "Notas del Curso").

12. *No-Determinismo = Determinismo*: Escriba un programa que reciba un autómata no-determinista o una gráfica de transición y calcule el autómata determinista equivalente al dispositivo dado. (Cfr. Sec. 4.3 y 4.4 de las "Notas del Curso").

13. *Bidireccionalidad = Monodireccionalidad*: Escriba un programa que reciba un autómata bidireccional y calcule el autómata (monodireccional) equivalente al dispositivo dado. (Cfr. Sec. 4.5 de las "Notas del Curso").

14. *Analizador de expresiones regulares*: Escriba un programa que reciba una expresión regular $X \in ER$, calcule una gráfica de transición cuyo lenguaje sea el expresado por X. (Cfr. Sec. 5.2 de las "Notas del Curso").

15. *Seudoinversión matricial con expresiones regulares*: Escriba un programa que reciba una matriz $\mathbf{A} \in ER^{n \times n}$, con $n \leq 5$, y calcule $\mathbf{A}^*$. (Cfr. Sec. 5.4 de las "Notas del Curso").

16. *Caracterización de lenguajes regulares*: Utilizando el programa del ejercicio anterior, escriba un programa que reciba un autómata determinista y calcule una expresión regular que represente al lenguaje del autómata. (Cfr. Sec. 5.2 de las "Notas del Curso").

17. *Reconocedores de lenguajes finitos*: Escriba un programa que reciba un conjunto finito de palabras y construya el autómata finito que reconoce únicamente a las palabras dadas.

18. *Contención de lenguajes regulares*: Escriba un programa que dadas dos expresiones regulares determine si la primera está contenida, y en caso que no, encuentre una palabra en el primer lenguaje que no esté en el segundo.

19. *Reconocedor de subpalabras*: Escriba un programa que dado un conjunto de k palabras, $\{\sigma_1, \ldots, \sigma_n\}$, construya una máquina de Moore, con alfabeto de salida $(a_K)_{K \subset [\![1,n]\!]}$, tal que para cualquier palabra dada τ la respuesta del autómata tras de leer τ será a_K, donde $K = \{i | \sigma_i$ es una subpalabra de $\tau\}$.

20. *Reconocedor de caminos*: Sea $G = (V, D)$, con $D \subset V \times V$, una gráfica dirigida. Una *camino* es una sucesión de aristas $(a_i = (v_{i1}, v_{i2}))_i$ tal que $\forall i : \ v_{i2} = v_{i+1,1}$.

Escriba un programa que dada una gráfica dirigida, construya un autómata sobre el alfabeto de aristas en la gráfica que reconozca exactamente a los caminos en la gráfica.

21. *Reverso de elenguajes regulares*: Escriba un programa que dado un autómata finito AF construya el autómata finito AF^R tal que $L(\mathrm{AF}^R) = L(\mathrm{AF})^R =$ (reverso del lenguaje reconocido por AF).

22. *Cadenas minimales en lenguajes regulares*: Para un lenguaje L sea

$$\mathrm{Min}(L) = \{\sigma \in L | \forall \tau \in L - \{nil\} : \sigma \geq_{prefijo} \tau \ \Rightarrow \ \sigma = \tau\}$$

el lenguaje consistente de las palabras en L que no poseen prefijo propio en L.

Escriba un programa que dado un autómata finito AF construya el autómata finito Min(AF) tal que $L(\mathrm{Min}(\mathrm{AF})) = \mathrm{Min}\, L(\mathrm{AF})$.

23. *Cadenas maximales en lenguajes regulares*: Para un lenguaje L sea

$$\mathrm{Max}(L) = \{\sigma \in L | \forall \tau \in L : \sigma \leq_{prefijo} \tau \ \Rightarrow \ \sigma = \tau\}$$

el lenguaje consistente de las palabras en L que no son prefijo propio de ninguna otra palabra en L.

Escriba un programa que dado un autómata finito AF construya el autómata finito Max(AF) tal que $L(\mathrm{Max}(\mathrm{AF})) = \mathrm{Max}\, L(\mathrm{AF})$.

24. *Regularidad de estrellas en alfabetos mónicos*: Escriba un programa que dado un conjunto finito de índices $I \subset \mathbb{N}$, construya un autómata, sobre el alfabeto $A = \{a\}$ de un único símbolo, que reconozca al lenguaje $L = \left(\sum_{i \in I} a^i\right)^*$.

25. *Palíndromas de lenguajes regulares*: Para un lenguaje L sea

$$L^{especular} = \{\sigma | \exists \tau \in L : \ \sigma = \tau\tau^R\}$$

el lenguaje consistente de palíndormas con una "primera parte" en L, es decir de palabras, e imágenes especulares, en L.

Escriba un programa que dada una expresión regular E construya el autómata finito $\mathrm{AF}^{especular}$ tal que $L(\mathrm{AF}^{especular}) = L(E)^{especular}$, donde $L(E)$ es el lenguaje expresado por E.
Sugerencia: Razonando por inducción en la construcción de la expresión E, tranfórmela en una expresión que exprese al lenguaje $L(E)^{especular}$ y a partir de ella construya al autómata pedido.

26. *Raíces n-ésimas de lenguajes regulares*: Para un lenguaje L y un entero n, sea

$$\sqrt[n]{L} = \{\sigma | \exists \tau \in L : \ \sigma = \tau^n\}$$

el lenguaje consistente de n repeticiones de una "primera parte" en L.

Escriba un programa que dada una expresión regular E construya el autómata finito $\sqrt[n]{\mathrm{AF}}$ tal que $L(\sqrt[n]{\mathrm{AF}}) = \sqrt[n]{L(E)}$, donde $L(E)$ es el lenguaje expresado por E.
Sugerencia: Razonando por inducción en la construcción de la expresión E, transfórmela en una expresión que exprese al lenguaje $\sqrt[n]{L(E)}$ y a partir de ella construya al autómata pedido.

27. *Rotaciones de lenguajes regulares*: Para un lenguaje L sea

$$rota(L) = \{\sigma | \exists \tau, \upsilon : \ \sigma = \tau\upsilon \ \& \ \upsilon\tau \in L\}$$

el lenguaje consistente de transposiciones de dos partes en L.

Escriba un programa que dada una expresión regular E construya el autómata finito $rota(\mathrm{AF})$ tal que $L(rota(\mathrm{AF})) = rota(L(E))$, donde $L(E)$ es el lenguaje expresado por E.

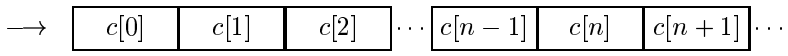
28. *Autómatas no-deterministas-*Π: Un *autómata no-determinista-*Π *(AND-*Π*)* es un autómata no-determinista con la noción de reconocimiento cambiada: Una palabra es *reconocida* si **toda** computación, a partir de ella, en el autómata la conduce a un estado final. Escriba un programa que transforme a un AND-Π en un autómata finito equivalente.

Capítulo 5

Autómatas de pila

5.1 Principios básicos

Una *pila* es un dispositivo de almacenamiento que sigue el principio "*primero-en-entrar-último-en salir*". Lo podemos pensar como un arreglo lineal indicado con los números naturales:



La *casilla* $c[0]$ se dice ser el *tope* de la pila. En todo momento,

- cada casilla puede estar *vacía* o contener un símbolo de un *alfabeto de pila*, *AlfP*, y
- la pila está en blanco salvo en un conjunto finito de casillas, i.e.

$$\exists n_0 \forall n : \ n > n_0 \Rightarrow \mathit{Contenido}(c[n]) = \mathit{vacía}.$$

En este caso, si para cada $i \geq 0$, x_i es el contenido de la casilla $c[i]$, entonces

- podemos establecer que el símbolo más a la derecha, x_{n_0}, sea un símbolo distinguido X en el alfabeto *AlfP* (para marcar el fin de la palabra inscrita en la pila), y
- decimos que la palabra $\sigma = x_0 x_1 \cdots x_{n_0-1}$ es el *contenido* de la pila.

Sobre una pila se pueden ejecutar dos operaciones:

Empilar: Si σ_0 es una palabra en AlfP^* y $\sigma \in \mathit{AlfP}^*$ es el contenido de la pila, entonces la acción $E(\sigma_0, \mathit{Pila})$ hace que el contenido de la pila sea $\sigma_0\sigma$.

Desempilar: Si σ_0 es un prefijo del contenido $\sigma \in \mathit{AlfP}^*$ de la pila, es decir $\sigma = \sigma_0\sigma_1$ para alguna palabra $\sigma_1 \in \mathit{AlfP}^*$, entonces la acción $D(\sigma_0, \mathit{Pila})$ hace que el contenido de la pila sea σ_1.

(Es convencional nombrar a las operaciones E y D por sus denotaciones en inglés: *Push* y *Pop*, respectivamente.)

Ahora bien, estas dos operaciones pueden realizarse mediante una operación de "sustitución" del tope de la pila: $\mathit{Sus}(\sigma_0, \mathit{Pila})$ hace que si el contenido de la pila fuese $\sigma_1 = s\sigma_2$, donde s es el símbolo actual en el tope de la pila, entonces el contenido vendrá a ser $\sigma_0 \cdot \sigma_1$. En otras palabras, el contenido del tope s es sustituído por σ_0.

Así pues la acción $E(\sigma_0, \mathit{Pila})$ equivale a $\mathit{Sus}(\sigma_0 \cdot s, \mathit{Pila})$, donde s es el contenido en el tope; en tanto que la acción $D(\sigma_0, \mathit{Pila})$ es equivalente a reiterar la acción $\mathit{Sus}(\mathit{nil}, \mathit{Pila})$ tantas veces como sea la longitud de σ_0.

En el resto de esta sección consideraremos solamente acciones de sustitución en el tope de la pila.

Un *autómata de pila (no-determinista)* es una estructura de la forma

$$\mathit{AutP} = (Q, \mathit{Ent}, \mathit{AlfP}, \mathit{tran}, q_0, X, F)$$

donde

$$\begin{array}{rcl} Q & : & \text{es el conjunto de } \textit{estados}, \\ \textit{Ent} & : & \text{es el alfabeto de } \textit{entrada}, \\ \textit{AlfP} & : & \text{es el alfabeto de la pila}, \\ \textit{tran} & : & Q \times (\textit{Ent} \cup \{\textit{nil}\}) \times \textit{AlfP} \to \mathcal{P}\left(Q \times \textit{AlfP}^*\right), \text{ es la función de } \textit{transición}, \\ q_0 \in Q & : & \text{es el estado } \textit{inicial}, \\ X \in \textit{AlfP} & : & \text{es el símbolo "inicial" para la pila, y}, \\ F \subset Q & : & \text{es un conjunto de estados } \textit{finales}. \end{array}$$

En cada momento, el autómata funciona como sigue: Si se está en el estado $q \in Q$, se recibe el símbolo $e \in \textit{Ent}$ y en el tope de la pila se encuentra el símbolo $Y \in \textit{AlfP}$,

- si $(q', \sigma) \in \textit{tran}(q, \textit{nil}, Y)$ entonces se pasa al estado q', se sustituye Y por σ y NO se avanza en la cadena de entrada, o bien
- si $(q', \sigma) \in \textit{tran}(q, e, Y)$ entonces se pasa al estado q', se opera en la pila según se describe arriba y se avanza una posición a la derecha en la cadena de entrada.

Ejemplo *(Palíndromas)*: Sea $L = \{\sigma \in (0+1)^* | \sigma = \textit{rev}(\sigma)\}$ el lenguaje que consta de todas las palabras que son palíndromas.

Una estrategia obvia para reconocer palabras en L es la siguiente:

1. al inicio la pila ha de estar vacía,
2. se deposita en la pila una copia de "la primera mitad de la palabra recibida", es decir,
   (a) con cada 0 que llegue se coloca una marca C en la pila,
   (b) con cada 1 que llegue se coloca una marca U en la pila,
3. al (suponer) haber llegado la "primera mitad" se busca que "la segunda" empate con la primera, es decir,
   (a) se pasa a un estado de desempilar,
   (b) con cada 0 que llegue y que empate con una marca C en la pila, se desempila la marca,
   (c) con cada 1 que llegue y que empate con una marca U en la pila, se desempila la marca,
4. se tendrá éxito sólo si al terminar de leer se tiene vacía la pila.

Así pues, consideremos los conjuntos siguientes:

Alfabeto de entrada: $\textit{Ent} = \{0, 1\}$.

Alfabeto de pila: $\textit{AlfP} = \{C, U, A, X\}$. X es la marca del extremo derecho de la pila. C y U son marcas para recontar 0's y 1's, y A es una marca de error.

Estados: Consideremos 4 estados,

$$\begin{array}{rcl} q_0 & : & \text{inicial} \\ q_1 & : & \text{comienza a desempilar}, \\ q_2 & : & \text{"salta" al símbolo del medio en el caso de palíndromas impares}, \\ q_3 & : & \text{final} \end{array}$$

Transiciones: Las siguientes transiciones se explican por sí solas. Las de la izquierda corresponden a "buenas computaciones" de reconocimiento. Las de la derecha marcan errores.

$(q_0, 0, X)$	$=$	$\{(q_0, CX), (q_1, CX), (q_3, nil)\}$	$(q_1, 0, X)$	$=$	$\{(q_3, A)\}$
$(q_0, 0, C)$	$=$	$\{(q_0, CC), (q_1, CC), (q_2, CC)\}$	$(q_1, 0, U)$	$=$	$\{(q_3, A)\}$
$(q_0, 0, U)$	$=$	$\{(q_0, CU), (q_1, CU), (q_2, CU)\}$	$(q_1, 1, X)$	$=$	$\{(q_3, A)\}$
$(q_0, 1, X)$	$=$	$\{(q_0, UX), (q_1, UX), (q_3, nil)\}$	$(q_1, 1, C)$	$=$	$\{(q_3, A)\}$
$(q_0, 1, C)$	$=$	$\{(q_0, UC), (q_1, UC), (q_2, UC)\}$			
$(q_0, 1, U)$	$=$	$\{(q_0, UU), (q_1, UU), (q_2, UU)\}$			
$(q_1, 0, C)$	$=$	$\{(q_1, nil), (q_3, nil)\}$			
$(q_1, 1, U)$	$=$	$\{(q_1, nil), (q_3, nil)\}$			
(q_1, nil, X)	$=$	$\{(q_3, nil)\}$			
(q_2, nil, C)	$=$	$\{(q_1, nil)\}$			
(q_2, nil, U)	$=$	$\{(q_1, nil)\}$			

En cualquier otra configuración, la función de transición asocia el conjunto vacío.

Ejemplo: Sea $L = \{0^{3n}1^n | n \geq 0\}$ el lenguaje que consta de todas las palabras que bien son vacías o bien se inician con 0's, terminan con 1's y contienen el triple de 0's que de 1's.

Una estrategia obvia para reconocer palabras en L es la siguiente:

1. al inicio la pila ha de estar vacía,
2. con cada 0 que llegue se coloca una marca C en la pila,
3. al llegar al primer 1 se pasa a un estado de desempilar,
4. con cada 1 hay que desempilar exactamente 3 marcas C, y
5. se tendrá éxito sólo si al terminar de leer se tiene vacía la pila.

Así pues, consideremos los conjuntos siguientes:

Alfabeto de entrada: $Ent = \{0, 1\}$.

Alfabeto de pila: $AlfP = \{C, A, X\}$. X es la marca del extremo derecho de la pila. C es una marca para recontar 0's y A es una marca de error.

Estados: Consideremos 5 estados,

q_0 : inicial
q_1 : comienza a desempilar secuencias de tres C's,
q_2 : se ha desempilado una C en cada secuencia de tres C's,
q_3 : se ha desempilado dos C's en cada secuencia de tres C's,
q_4 : final

Transiciones: Las siguientes transiciones se explican por sí solas. Las de la izquierda corresponden a "buenas computaciones" de reconocimiento. Las de la derecha marcan errores.

$(q_0, 0, X)$	$=$	$\{(q_0, CX)\}$	$(q_0, 1, X)$	$=$	$\{(q_4, A)\}$
$(q_0, 0, C)$	$=$	$\{(q_0, CC), (q_1, CC)\}$	$(q_1, 1, X)$	$=$	$\{(q_4, A)\}$
$(q_0, 1, C)$	$=$	$\{(q_2, nil)\}$	$(q_1, 0, X)$	$=$	$\{(q_4, A)\}$
(q_2, nil, C)	$=$	$\{(q_3, nil)\}$	$(q_1, 0, C)$	$=$	$\{(q_4, A)\}$
(q_3, nil, C)	$=$	$\{(q_1, nil)\}$	(q_2, nil, X)	$=$	$\{(q_4, A)\}$
$(q_1, 1, C)$	$=$	$\{(q_2, nil)\}$	(q_3, nil, X)	$=$	$\{(q_4, A)\}$
(q_1, nil, X)	$=$	$\{(q_4, nil)\}$			

En cualquier otra configuración, la función de transición asocia el conjunto vacío.

5.2 Reconocimiento de lenguajes

Sea $AutP = (Q, Ent, AlfP, tran, q_0, X, F)$ un autómata de pila.

Una *descripción instantánea (DI)* es una cadena

$$q\sigma;\tau \in Q \times Ent^* \times AlfP^*$$

que indica que el autómata está en el estado q, se está leyendo el primer símbolo a la izquierda de σ y τ es el contenido de la pila. El símbolo en el tope de la pila es el primero de τ.

Diremos que una DI $d_1 = q^1\sigma^1;\tau^1$ *se sigue* de otra $d_0 = q^0\sigma^0;Y^0\tau^0$, $AutP \vdash (d_0 \to d_1)$, si es el resultado de aplicar la transición correspondiente a d_0. Formalmente, $AutP \vdash (d_0 \to d_1)$ si y sólo si

$$\begin{array}{lll}
\left[\sigma^0 = e^0\sigma^1\right] \wedge \left[\tau^1 = \tau\tau^0\right] & \text{si} & (q^1,\tau) \in tran(q^0, e^0, Y^0) \\
\left[\sigma^0 = e^0\sigma^1\right] \wedge \left[\tau^1 = \tau^0\right] & \text{si} & (q^1, nil) \in tran(q^0, e^0, Y^0) \\
\left[\sigma^0 = \sigma^1\right] \wedge \left[\tau^1 = \tau\tau^0\right] & \text{si} & (q^1,\tau) \in tran(q^0, nil, Y^0) \\
\left[\sigma^0 = \sigma^1\right] \wedge \left[\tau^1 = \tau^0\right] & \text{si} & (q^1, nil) \in tran(q^0, nil, Y^0)
\end{array}$$

La cerradura reflexivo-transitiva de la relación "se sigue" es la relación "*se deriva*", denotada como $AutP \vdash (d_0 \overset{*}{\to} d_1)$.

La *descripción inicial de una palabra* $\sigma \in Ent^*$ es $DI_0(\sigma) \equiv [q_0\sigma; X]$.

Una DI $d = q\sigma;\tau$ se dice

- *final* si $q \in F$ y $\sigma = nil$, es decir, si indica que se está en un estado final y no quedan símbolos de entrada,
- *de pila vacía* si $\tau = nil$, es decir, en el tope de la pila no hay símbolo alguno.

Una palabra $\sigma \in Ent^*$ se dice

- *reconocida por estados finales* si desde la descripción inicial de ella se arriba a una final, es decir, si

$$\exists q \in F, \tau \in AlfP^* : \ AutP \vdash (DI_0(\sigma) \overset{*}{\to} q.nil;\tau).$$

- *reconocida por pila vacía* si desde la descripción inicial de ella se arriba a una de pila vacía, es decir, si

$$\exists q \in Q : \ AutP \vdash (DI_0(\sigma) \overset{*}{\to} q.nil; nil).$$

Sea $LF(AutP)$ el lenguaje consistente de las palabras reconocidas por estados finales.

Sea $LPV(AutP)$ el lenguaje consistente de las palabras reconocidas por pila vacía.

Ambas nociones de reconocimiento son equivalentes.

Lema 5.2.1 *Para cada autómata de pila* $AutP = (Q, Ent, AlfP, tran, q_0, X, F)$ *existe otro autómata de pila* $AutP' = (Q', Ent, AlfP', tran', q_0', X', F')$ *tal que el lenguaje reconocido por estados finales en el primer autómata es reconocido por pila vacía en el segundo, es decir*

$$LF(AutP) = LPV(AutP)'$$

En efecto, $AutP'$ se construye a partir de $AutP$ como sigue:

1. en general, actúese en $AutP'$ como se hace en $AutP$,
2. cuando se llegue a un estado final de $AutP$, transítese en $AutP'$ de manera que ahí se vacíe la pila,
3. para evitar reconocimiento de palabras que incidentalmente vacíen la pila de $AutP'$, colóquese ahí un nuevo delimitador "derecho" X' que no pertenezca al alfabeto $AlfP$ y bórrese sólo en el caso del paso 2.

Lema 5.2.2 *Para cada autómata de pila* $AutP = (Q, Ent, AlfP, tran, q_0, X, F)$ *existe otro autómata de pila* $AutP' = (Q', Ent, AlfP', tran', q_0', X', F')$ *tal que el lenguaje reconocido por pila vacía en el primer autómata es reconocido por estados finales en el segundo, es decir*

$$LPV(AutP) = LF(AutP)'$$

En efecto, $AutP'$ se construye a partir de $AutP$ como sigue:

1. colóquese en $AutP'$ un nuevo símbolo inicial X',
2. en general, actúese en $AutP'$ como se hace en $AutP$,
3. cuando se vacíe la pila de $AutP$, es decir, cuando la marca X' acceda al tope de la pila, transítese en $AutP'$ a un nuevo estado final.

De ahora en adelante supondremos, sin pérdida de generalidad, que el reconocimiento de un lenguaje con un autómata de pila se hace por pila vacía, y consecuentemente no consideraremos más estados finales.

5.3 Autómatas de pila y lenguajes libres de contexto

Se tiene que una condición para que un lenguaje sea reconocido por un autómata de pila es que ese lenguaje sea libre de contexto. En esta sección probaremos y ejemplificaremos que la condición es necesaria. Pospondremos la suficiencia de esta condición a la exposición de los lenguajes libres de contexto.

Proposición 5.3.1 *Para cualquier autómata de pila* $AutP = (Q, Ent, AlfP, tran, q_0, X, \emptyset)$ *su lenguaje* $L = LPV(AutP)$ *es libre de contexto.*

En efecto, sea $G = (V, Ent, P, S)$ la gramática tal que

$$\begin{array}{lcl} V & = & (Q \times AlfP \times Q) \cup \{S\}, \\ S & : & \text{(nuevo) símbolo inicial,} \\ P & : & \text{conjunto de producciones definidas por las transiciones de } AutP\text{:} \end{array}$$

1. $\forall q \in Q :\ S \to [q_0, X, q]$
2. $\forall q \in Q, e \in Ent, Y \in AlfP$: si $(q', Y_1 Y_2 \cdots Y_m) \in tran(q, e, Y)$ entonces inclúyase la producción
$$[q, Y, q^{(m+1)}] \to e[q', Y_1, q^{(2)}][q^{(2)}, Y_2, q^{(3)}] \cdots [q^{(m)}, Y_m, q^{(m+1)}]$$
para cualesquiera selecciones de los estados $q^{(2)}, \ldots, q^{(m)}, q^{(m+1)}, \in Q$.
3. $\forall q \in Q, e \in Ent, Y \in AlfP$: si $(q', Y_1 Y_2 \cdots Y_m) \in tran(q, nil, Y)$ entonces inclúyase la producción
$$[q, Y, q^{(m+1)}] \to [q', Y_1, q^{(2)}][q^{(2)}, Y_2, q^{(3)}] \cdots [q^{(m)}, Y_m, q^{(m+1)}]$$
para cualesquiera selecciones de los estados $q^{(2)}, \ldots, q^{(m)}, q^{(m+1)}, q^{(m+2)} \in Q$.
4. $\forall q \in Q, e \in Ent, Y \in AlfP$: si $(q', nil) \in tran(q, e, Y)$ entonces inclúyase la producción $[q, Y, q'] \to e$.
5. $\forall q \in Q, e \in Ent, Y \in AlfP$: si $(q', nil) \in tran(q, nil, Y)$ entonces inclúyase la producción $[q, Y, q'] \to nil$.

La idea subyacente en esta construcción consiste en que toda derivación siniestra en la gramática definida ha de simular una computación reconocedora en el autómata dado.

Veamos que para cualquier palabra $\sigma \in Ent^*$ y para cualesquiera $q, q' \in Q$, $Y \in AlfP$ se cumple

$$G \vdash \left([q, Y, q'] \stackrel{*}{\Rightarrow} \sigma\right) \ \Leftrightarrow\ AutP \vdash \left(q\sigma; Y \stackrel{*}{\Rightarrow} q' nil; X\right) \tag{5.1}$$

$\Leftarrow$) Mostremos esta implicación por inducción en el número de pasos de derivación.

Caso base: Supongamos que la DI $q'nil; X$ se siga inmediatamente de la DI $q\sigma; Y$. Entonces σ consta a lo sumo de un símbolo y necesariamente $(q', nil) \in tran(q, \sigma, Y)$. Por tanto, $[q, Y, q'] \to \sigma$ es una producción legal en G. Eso da el correspondiente miembro izquierdo de la relación 5.1.
Caso inductivo: Sea $k > 0$. Supongamos que si $q'nil; X$ se deriva de $q\sigma; Y$ en a lo sumo k pasos, entonces $G \vdash \left([q, Y, q'] \stackrel{*}{\Rightarrow} \sigma\right)$.

Consideremos ahora el caso en el que $q'nil; X$ se deriva de $q\sigma; Y$ en exactamente $k+1$ pasos.

Escribamos $\sigma = s_1\sigma_1$. Necesariamente, existe una primera DI $q^{(1)}\sigma_1; Y_1 \cdots Y_m$ tal que

$$AutP \vdash q\sigma; Y \Rightarrow \left(q^{(1)}\sigma_1; Y_1 \cdots Y_m\right) \stackrel{*}{\Rightarrow} q'nil; X$$

donde la última derivación se hace en a lo sumo k pasos.

Descompongamos σ_1 en m tramos, $\sigma_1 = \sigma_1^{(1)} \cdots \sigma_1^{(m)}$, donde cada $\sigma_1^{(i)}$ realiza el efecto global de "desempilar" al símbolo Y_i. Se tiene que existen $q^{(2)}, q^{(3)}, \ldots, q^{(m)}, q^{(m+1)} = q' \in Q$ tales que

$$\forall i \leq m: \quad AutP \vdash \left(q^{(i)}\sigma_1^{(i)}; Y_i \stackrel{*}{\Rightarrow} q^{(i+1)}nil; X\right). \tag{5.2}$$

Por la hipótesis de inducción se tiene $G \vdash \left([q^{(i)}, Y_i, q^{(i+1)}] \stackrel{*}{\Rightarrow} \sigma_1^{(i)}\right)$. Ahora, de la primera derivación $AutP \vdash q\sigma; Y \Rightarrow q^{(1)}\sigma_1; Y_1 \cdots Y_m$, se tiene

$$G \vdash \left([q, Y, q'] \to s_1[q^{(1)}, Y_1, q^{(2)}][q^{(2)}, Y_2, q^{(3)}] \cdots [q^{(m)}, Y_m, q^{(m+1)}]\right), \tag{5.3}$$

y consecuentemente, de 5.2 y 5.3 $G \vdash \left([q, Y, q'] \stackrel{*}{\Rightarrow} \sigma\right)$.

$\Rightarrow$) Mostremos ahora la implicación recíproca por inducción en el número de pasos de derivación.
Caso base: Si $[q, Y, q'] \to \sigma$ es una producción legal en G, entonces σ consta a lo sumo de un símbolo y $(q', nil) \in tran(q, \sigma, Y)$. Por tanto la DI $q'nil; X$ se sigue inmediatamente de la DI $q\sigma; Y$. Esto da el correspondiente miembro derecho de la relación 5.1.
Caso inductivo: Sea $k > 0$. Supongamos que si $G \vdash \left([q, Y, q'] \stackrel{*}{\Rightarrow} \sigma\right)$ se deriva en a lo sumo k pasos, entonces $q'nil; X$ se deriva de $q\sigma; Y$.

Consideremos ahora el caso en el que σ se deriva de $[q, Y, q']$ en exactamente $k+1$ pasos.

Escribamos $\sigma = s_1\sigma_1$. Necesariamente, para una primera derivación se ha de tener

$$[q, Y, q'] \to \left(s_1[q^{(1)}, Y_1, q^{(2)}][q^{(2)}, Y_2, q^{(3)}] \cdots [q^{(m)}, Y_m, q^{(m+1)}]\right) \stackrel{*}{\Rightarrow} \sigma$$

donde la última derivación se hace en a lo sumo k pasos.

Escribamos $\sigma_1 = \sigma_1^{(1)} \cdots \sigma_1^{(m)}$, donde cada $\sigma_1^{(i)}$ es tal que $G \vdash [q^{(i)}, Y_1, q^{(i+1)}] \stackrel{*}{\Rightarrow} \sigma_1^{(i)}$.

Por la hipótesis de inducción se tiene que $\forall i \leq m: \quad AutP \vdash \left(q^{(i)}\sigma_1^{(i)}; Y_i \stackrel{*}{\Rightarrow} q^{(i+1)}nil; X\right)$ y de aquí se tiene también que

$$\forall i \leq m: \quad AutP \vdash \left(q^{(i)}\sigma_1^{(i)}; Y_iY_{i+1} \cdots Y_mX \stackrel{*}{\Rightarrow} q^{(i+1)}nil; Y_{i+1} \cdots Y_mX\right). \tag{5.4}$$

Como $[q, Y, q'] \to \left(s_1[q^{(1)}, Y_1, q^{(2)}][q^{(2)}, Y_2, q^{(3)}] \cdots [q^{(m)}, Y_m, q^{(m+1)}]\right)$ es una producción, necesariamente

$$q\sigma; X \to q^{(1)}\sigma_1^{(1)}; Y_1Y_2 \cdots Y_mX \tag{5.5}$$

y consecuentemente, de 5.4 y 5.5 $AutP \vdash \left(q\sigma; X \stackrel{*}{\Rightarrow} q'nil; X\right)$. *q.e.d.*

Ejemplo: En un ejemplo anterior, vimos que el lenguaje $L = \{0^{3n}1^n | n \geq 0\}$ es reconocido por el autómata

$(q_0, 0, X)$	$=$	$\{(q_0, CX)\}$	$(q_0, 1, X)$	$=$	$\{(q_1, A)\}$
$(q_0, 0, C)$	$=$	$\{(q_0, CC), (q_1, CC)\}$	$(q_1, 1, X)$	$=$	$\{(q_1, A)\}$
$(q_0, 1, C)$	$=$	$\{(q_2, nil)\}$	$(q_1, 0, X)$	$=$	$\{(q_1, A)\}$
(q_2, nil, C)	$=$	$\{(q_3, nil)\}$	$(q_1, 0, C)$	$=$	$\{(q_1, A)\}$
(q_3, nil, C)	$=$	$\{(q_1, nil)\}$	(q_2, nil, X)	$=$	$\{(q_1, A)\}$
$(q_1, 1, C)$	$=$	$\{(q_2, nil)\}$	(q_3, nil, X)	$=$	$\{(q_1, A)\}$
(q_1, nil, X)	$=$	$\{(q_1, nil)\}$			

donde en cualquier otra configuración, la función de transición asocia el conjunto vacío.

De acuerdo con la construcción anterior, el conjunto de variables de la gramática que simula computaciones en el autómata es $V = (Q \times AlfP \times Q) \cup \{S\}$ el cual, en este caso, tiene $4 \cdot 3 \cdot 4 + 1 = 49$ símbolos. Construyendo las producciones conforme van apareciendo los símbolos variables, obtenemos lo siguiente:

Producciones para S: Las producciones "iniciales" son $S \to [q_0, X, q_k]$ con $k = 0, 1, 2, 3$.

Producciones para $[q_0, X, q_k]$: Como $(q_0, 0, X) = \{(q_0, CX)\}$, tenemos las producciones

$$[q_0, X, q_k] \to 0[q_0, C, q_j][q_j, X, q_k] \qquad j, k = 0, 1, 2, 3.$$

Producciones para $[q_0, C, q_k]$: Como $(q_0, 0, C) = \{(q_0, CC), (q_1, CC)\}$ y $(q_0, 1, C) = \{(q_2, nil)\}$, tenemos las producciones

$$\begin{aligned}
[q_0, C, q_k] &\to 0[q_0, C, q_j][q_j, C, q_k] \qquad j, k = 0, 1, 2, 3 \\
[q_0, C, q_k] &\to 0[q_1, C, q_j][q_j, C, q_k] \qquad j, k = 0, 1, 2, 3 \\
[q_0, C, q_2] &\to 1
\end{aligned}$$

Producciones para $[q_1, C, q_2]$: Como $(q_1, 1, C) = \{(q_2, nil)\}$, tenemos la producción $[q_1, C, q_2] \to 1$.

Producciones para $[q_2, C, q_3]$ *y* $[q_3, C, q_1]$: Como $(q_2, nil, C) = \{(q_3, nil)\}$ y $(q_3, nil, C) = \{(q_1, nil)\}$, tenemos las producciones $[q_2, C, q_3] \to nil$ y $[q_3, C, q_1] \to nil$.

Producciones para $[q_1, X, q_1]$: Como $(q_1, nil, X) = \{(q_1, nil)\}$, tenemos la producción $[q_1, X, q_1] \to nil$.

Producciones de errores: Como $(q_0, 1, X) = \{(q_1, A)\}$, tenemos la producción

$$[q_0, X, q_k] \to 1[q_1, A, q_k].$$

De manera similar, de las demás condiciones de error tendremos las producciones

$$\begin{aligned}
[q_1, X, q_k] &\to 1[q_1, A, q_k] \\
[q_1, X, q_k] &\to 0[q_1, A, q_k] \\
[q_1, C, q_k] &\to 0[q_1, A, q_k] \\
[q_2, X, q_k] &\to [q_1, A, q_k] \\
[q_3, X, q_k] &\to [q_1, A, q_k]
\end{aligned}$$

y no hay producciones para variables de la forma $[q_i, A, q_k]$, por lo que una vez que se generaron no hay manera de conducirlas a una palabra terminal.

Reescribamos las variables como se indica a continuación:

A_0	:	$[q_0, X, q_0]$	B_0	:	$[q_0, C, q_0]$	D_0	:	$[q_2, C, q_0]$
A_1	:	$[q_0, X, q_1]$	B_1	:	$[q_0, C, q_1]$	D_1	:	$[q_2, C, q_1]$
A_2	:	$[q_0, X, q_2]$	B_2	:	$[q_0, C, q_2]$	D_2	:	$[q_2, C, q_2]$
A_3	:	$[q_0, X, q_3]$	B_3	:	$[q_0, C, q_3]$	D_3	:	$[q_2, C, q_3]$
			C_0	:	$[q_1, C, q_0]$	E_0	:	$[q_3, C, q_0]$
			C_1	:	$[q_1, C, q_1]$	E_1	:	$[q_3, C, q_1]$
F	:	$[q_1, X, q_1]$	C_2	:	$[q_1, C, q_2]$	E_2	:	$[q_3, C, q_2]$
			C_3	:	$[q_1, C, q_3]$	E_3	:	$[q_3, C, q_3]$

En la codificación anterior entre las variables de la forma $[q_j, X, q_k]$ sólo consideramos a $F = [q_1, X, q_1]$ pues las demás no aparecen como antecedentes de ninguna producción. Entonces, las producciones que no corresponden a situaciones de error son las siguientes:

S	$\to$	$A_0 \mid A_1 \mid A_2 \mid A_3$	A_0	$\to$	$0B_0A_0$	B_0	$\to$	$0B_0B_0 \mid 0B_1C_0 \mid 0B_2D_0 \mid 0B_3E_0 \mid$
A_1	$\to$	$0B_1F$	A_1	$\to$	$0B_0A_1$			$0C_0B_0 \mid 0C_1C_0 \mid 0C_2D_0 \mid 0C_3E_0$
B_2	$\to$	1	A_2	$\to$	$0B_0A_2$	B_1	$\to$	$0B_0B_1 \mid 0B_1C_1 \mid 0B_2D_1 \mid 0B_3E_1 \mid$
C_2	$\to$	1	A_3	$\to$	$0B_0A_3$			$0C_0B_1 \mid 0C_1C_1 \mid 0C_2D_1 \mid 0C_3E_1$
D_3	$\to$	nil				B_2	$\to$	$0B_0B_2 \mid 0B_1C_2 \mid 0B_2D_2 \mid 0B_3E_2 \mid$
E_1	$\to$	nil						$0C_0B_2 \mid 0C_1C_2 \mid 0C_2D_2 \mid 0C_3E_2$
F	$\to$	nil				B_3	$\to$	$0B_0B_3 \mid 0B_1C_3 \mid 0B_2D_3 \mid 0B_3E_3 \mid$
								$0C_0B_3 \mid 0C_1C_3 \mid 0C_2D_3 \mid 0C_3E_3$

Ahora, si suprimimos a las producciones que en su consecuente tengan símbolos que no aparecen en el antecedente de ninguna otra producción (pues esos símbolos no podrán sustituirse y por tanto no derivan ninguna palabra terminal) se tiene las producciones:

S	$\to$	$A_0 \mid A_1 \mid A_2 \mid A_3$	A_0	$\to$	$0B_0A_0$	B_0	$\to$	$0B_0B_0$
A_1	$\to$	$0B_1F$	A_1	$\to$	$0B_0A_1$	B_1	$\to$	$0B_0B_1 \mid 0B_3E_1$
B_2	$\to$	1	A_2	$\to$	$0B_0A_2$	B_2	$\to$	$0B_0B_2 \mid 0B_1C_2$
C_2	$\to$	1	A_3	$\to$	$0B_0A_3$	B_3	$\to$	$0B_0B_3 \mid 0B_2D_3 \mid 0C_2D_3$
D_3	$\to$	nil						
E_1	$\to$	nil						
F	$\to$	nil						

Observemos ahora que toda vez que se genera B_0 no hay manera de suprimirla. Por tanto, ninguna derivación terminal puede involucrar a esa variable. Así pues, omitiendo las producciones que involucran a B_0 se tiene la gramática

S	$\to$	A_1	B_1	$\to$	$0B_3E_1$
A_1	$\to$	$0B_1F$	B_2	$\to$	$0B_1C_2$
B_2	$\to$	1	B_3	$\to$	$0B_2D_3 \mid 0C_2D_3$
C_2	$\to$	1			
D_3	$\to$	nil			
E_1	$\to$	nil			
F	$\to$	nil			

la cual efectivamente genera al lenguaje L.

5.4 Autómatas de pila deterministas

Un *autómata de pila determinista* es un autómata de pila

$$AutP = (Q, Ent, AlfP, tran, q_0, X, F)$$

donde se cumplen las siguientes dos propiedades:

- Para cualquier pareja (estado,tope_de_la_pila) se tiene que o bien el autómata tiene definida la transición correspondiente a la palabra de entrada vacía o bien la tiene definida en símbolos del alfabeto de entrada, y
- para cualquier terceta (estado,s,tope_de_la_pila), la transición correspondiente contiene, a lo sumo, una pareja (nuevo_estado,tope_de_la_pila).

En símbolos,

$$\begin{array}{lllll} \forall (q,Y) \in Q \times AlfP: & tran(q,nil,Y) \neq \emptyset & \Rightarrow & \forall s \in Ent : tran(q,s,Y) = \emptyset \\ \forall (q,Y) \in Q \times AlfP & \forall s \in Ent \cup \{nil\} & : & card\,(tran(q,nil,Y)) \leq 1 \end{array}$$

Sea $AutP = (Q, Ent, AlfP, tran, q_0, X, F)$ un autómata de pila determinista. De acuerdo con la construcción vista para obtener la gramática libre de contexto de un autómata de pila dado, tendremos que las producciones de la gramática correspondiente han de ser las siguientes:

1. $\forall q \in Q : \ S \to [q_0, X, q]$
2. $\forall q \in Q, e \in Ent, Y \in AlfP : \{(q', Y_1Y_2 \cdots Y_m)\} = tran(q,e,Y) \ \Rightarrow$

$$[q, Y, q^{(m+1)}] \to e[q', Y_1, q^{(2)}][q^{(2)}, Y_2, q^{(3)}] \cdots [q^{(m)}, Y_m, q^{(m+1)}]$$

para cualesquiera selecciones de los estados $q^{(2)}, \ldots, q^{(m)}, q^{(m+1)}, \in Q$.

3. $\forall q \in Q, e \in Ent, Y \in AlfP : \{(q', Y_1 Y_2 \cdots Y_m)\} = tran(q, nil, Y) \Rightarrow$

$$[q, Y, q^{(m+1)}] \to [q', Y_1, q^{(2)}][q^{(2)}, Y_2, q^{(3)}] \cdots [q^{(m)}, Y_m, q^{(m+1)}]$$

para cualesquiera selecciones de los estados $q^{(2)}, \ldots, q^{(m)}, q^{(m+1)}, q^{(m+2)} \in Q$.

4. $\forall q \in Q, e \in Ent, Y \in AlfP : \{(q', nil)\} = tran(q, e, Y) \Rightarrow ([q, Y, q'] \to e)$.

5. $\forall q \in Q, Y \in AlfP : \{(q', nil)\} = tran(q, nil, Y) \Rightarrow ([q, Y, q'] \to nil)$.

En este caso, si para una pareja $(q, Y) \in Q \times AlfP$ se generasen producciones de acuerdo con la regla 3 entonces no se generará ninguna con la regla 2. Similarmente, si se generase una con la 5, no se generarará ninguna con la 4.

De aquí resulta que cualquier lenguaje reconocido por una autómata de pila determinista ha de ser generado por una gramática libre de contexto cuyas producciones son de la forma

$$\begin{aligned} A &\to \sigma_1 A_1 \cdots \sigma_{k-1} A_{k-1} \sigma_k, \quad \text{donde } \sigma_1, \ldots, \sigma_{k-1}, \sigma_k \in T^*, A_1, \ldots, A_{k-1} \in V, \text{ o bien} \\ A &\to e, \quad e \in AlfP \end{aligned}$$

5.5 Autómatas de pila con escritura

Un autómata de pila contiene un arreglo de entrada y una pila. El arreglo de entrada es donde se inscribe la palabra de entrada, y es recorrido, sin retroceso, de izquierda a derecha por el autómata. En este arreglo, el autómata se detiene en su recorrido cuando se encuentra con una transición de la forma (estado,*nil*,tope_de_la_pila). La pila es un dispositivo de almacenamiento del tipo "el primero en entrar es el último en salir".

Consideremos ahora un segundo arreglo, llamado *de salida*, que es donde se inscribirá la palabra de salida y que el autómata recorre, también sin retroceso, de izquierda a derecha.

Dada una palabra de entrada σ, una correspondiente palabra de salida τ será la que quede inscrita en la cinta tras de que se haya leído toda la palabra σ y, además, en esa situación, se hayan agotado todas las transiciones de la forma (estado_actual,*nil*,tope_de_la_pila).

Formalmente:

Un *autómata de pila con escritura* es una estructura de la forma

$$AutP = (Q, Ent, Sal, AlfP, tran, esc, q_0, X)$$

donde

$$\begin{aligned} Q &: \text{es el conjunto de } estados, \\ Ent &: \text{es el alfabeto de } entrada, \\ Sal &: \text{es el alfabeto de } salida, \\ AlfP &: \text{es el alfabeto de la pila}, \\ tran &: Q \times (Ent \cup \{nil\}) \times AlfP \to \mathcal{P}\left(Q \times AlfP^*\right), \quad \text{es la función de } transición, \\ esc &: Q \times (Ent \cup \{nil\}) \times AlfP \to \mathcal{P}(Sal^*), \quad \text{es la función de } escritura, \text{ y}, \\ q_0 \in Q &: \text{es el estado } inicial, \\ X \in AlfP &: \text{es el símbolo "inicial" para la pila.} \end{aligned}$$

En cada momento, el autómata funciona como sigue: Si se está en el estado $q \in Q$, se recibe el símbolo $e \in Ent$ y en el tope de la pila se encuentra el símbolo $Y \in AlfP$,

- si $(q', \sigma) \in tran(q, nil, Y)$ y $\tau \in esc(q, nil, Y)$ entonces se pasa al estado q', se sustituye Y por σ, se escribe en el arreglo de salida la palabra τ y NO se avanza en la cadena de entrada, o bien
- si $(q', \sigma) \in tran(q, e, Y)$ y $\tau \in esc(q, e, Y)$ entonces se pasa al estado q', se opera en la pila y en el arreglo de salida según se describe arriba y se avanza una posición a la derecha en la cadena de entrada.

Ejemplo *(Transformación de notación de enfijo a notación de sufijo)*:

Las expresiones aritméticas se forman mediante *constantes*, *variables* y *operadores*. Consideremos operadores *unarios*, $Oper_1$, y *binarios*, $Oper_2$. Supongamos además, como es convencional en cualquier lenguaje de programación de alto nivel y en la notación usual de enfijo, que los operadores unarios tienen prioridad de aplicación sobre los binarios, y que $Oper_1$ y $Oper_2$ están jerarquizados internamente por prioridades de aplicación: Un operador con prioridad mayor se aplicará primeramente y operadores con igual prioridad se aplican de derecha a izquierda. Los paréntesis se usan para imponer el orden de aplicación de los operadores pasando por alto las convenciones de prioridades.

Sea T_0 el conjunto formado por las constantes y las variables. El conjunto de *términos* en notación de *enfijo*, y para cada término su *operador principal*, se define como sigue:

$$
\begin{array}{lllll}
\text{i)} & \xi \in T_0 & \Rightarrow [\xi \in T] & \wedge & [OpPrinc(\xi) = nil] \\
\text{ii)} & \xi \in T, o_1 \in Oper_1 & \Rightarrow [o_1(\xi) \in T] & \wedge & [OpPrinc(o_1(\xi)) = o_1] \\
\text{iii)} & \left.\begin{array}{l}\xi_1, \xi_2 \in T, o_2 \in Oper_2, \\ Prior(o_2) < Prior(OpPrinc(\xi_1))\end{array}\right\} & \Rightarrow [\xi_1 o_2 \xi_2 \in T] & \wedge & [OpPrinc(\xi_1 o_2 \xi_2) = o_2] \\
\text{iv)} & \left.\begin{array}{l}\xi_1, \xi_2 \in T, o_2 \in Oper_2, \\ Prior(o_2) \geq Prior(OpPrinc(\xi_1))\end{array}\right\} & \Rightarrow [(\xi_1) o_2 \xi_2 \in T] & \wedge & [OpPrinc(\xi_1 o_2 \xi_2) = o_2] \\
\text{v)} & \left.\begin{array}{l}\xi_1, \xi_2 \in T, o_2 \in Oper_2, \\ Prior(o_2) < Prior(OpPrinc(\xi_2))\end{array}\right\} & \Rightarrow [\xi_1 o_2 \xi_2 \in T] & \wedge & [OpPrinc(\xi_1 o_2 \xi_2) = o_2] \\
\text{vi)} & \left.\begin{array}{l}\xi_1, \xi_2 \in T, o_2 \in Oper_2, \\ Prior(o_2) \geq Prior(OpPrinc(\xi_2))\end{array}\right\} & \Rightarrow [\xi_1 o_2 (\xi_2) \in T] & \wedge & [OpPrinc(\xi_1 o_2 \xi_2) = o_2] \\
\text{vii)} & \xi \in T & \Rightarrow [(\xi) \in T] & \wedge & [OpPrinc((\xi)) = OpPrinc(\xi)]
\end{array}
$$

La notación de *sufijo*, o llamada también *polaca*[1], compone a los términos de la aritmética en el esquema [Unico_Operando][Operador], en el caso de operadores unarios, o bien [Primer_Operando][Segundo_Operando][Operador], en el caso de operadores binarios. Formalmente, los términos se definen de la siguiente manera:

$$
\begin{array}{lll}
\text{i)} & \xi \in T_0 & \Rightarrow [\xi \in T] \\
\text{ii)} & \xi \in T, o_1 \in Oper_1 & \Rightarrow [\xi o_1 \in T] \\
\text{iii)} & \xi_1, \xi_2 \in T, o_2 \in Oper_2 & \Rightarrow [\xi_1 \xi_2 o_2 \in T]
\end{array}
$$

Cada término ξ en notación de enfijo corresponde a uno único equivalente $Tr(\xi)$ en notación de sufijo. De hecho la especificación de la "traducción" Tr queda como sigue:

$$
\begin{array}{lll}
\text{i)} & \xi \in T_0 & \Rightarrow Tr(\xi) = \xi \\
\text{ii)} & \xi \in T, o_1 \in Oper_1 & \Rightarrow Tr(o_1(\xi)) = Tr(\xi) o_1 \\
\text{iii)} & \left.\begin{array}{l}\xi_1, \xi_2 \in T, o_2 \in Oper_2, \\ Prior(o_2) < Prior(OpPrinc(\xi_1))\end{array}\right\} & \Rightarrow Tr(\xi_1 o_2 \xi_2) = Tr(\xi_1)\, Tr(\xi_2) o_2 \\
\text{iv)} & \left.\begin{array}{l}\xi_1, \xi_2 \in T, o_2 \in Oper_2, \\ Prior(o_2) \geq Prior(OpPrinc(\xi_1))\end{array}\right\} & \Rightarrow Tr((\xi_1) o_2 \xi_2) = Tr(\xi_1)\, Tr(\xi_2) o_2 \\
\text{v)} & \left.\begin{array}{l}\xi_1, \xi_2 \in T, o_2 \in Oper_2, \\ Prior(o_2) < Prior(OpPrinc(\xi_2))\end{array}\right\} & \Rightarrow Tr(\xi_1 o_2 \xi_2) = Tr(\xi_1)\, Tr(\xi_2) o_2 \\
\text{vi)} & \left.\begin{array}{l}\xi_1, \xi_2 \in T, o_2 \in Oper_2, \\ Prior(o_2) \geq Prior(OpPrinc(\xi_2))\end{array}\right\} & \Rightarrow Tr(\xi_1 o_2 (\xi_2)) = Tr(\xi_1)\, Tr(\xi_2) o_2 \\
\text{vii)} & \xi \in T & \Rightarrow Tr((\xi)) = Tr(\xi)
\end{array}
$$

La transformación Tr puede calcularse mediante un autómata de pila de escritura: En su arreglo de entrada se da una expresión aritmética de enfijo y en su arreglo de salida ha de quedar la expresión de sufijo equivalente.

En vez de dar explícitamente la especificación del autómata de pila, bosquejaremos un algoritmo describiendo el funcionamiento del autómata. Para ello, utilizaremos los conceptos siguientes:

[1] Se dice "polaca" porque fue utilizada primeramente por el lógico polaco Jan Łukasiewicz (1878-1956).

escribir: poner en la posición actual del arreglo de salida al símbolo que se indique, e incrementar la posición actual una posición a la derecha,

empilar: se coloca el símbolo actual del arreglo de entrada en la pila, es decir ése pasa a ser el contenido del tope de la pila, y se avanza un lugar en la posición actual de la entrada,

desempilar: se escribe el contenido del tope de la pila en el arreglo de salida, y se le suprime de la pila,

e**:** variable que denota al símbolo leído en el arreglo de entrada, en la posición actual,

t**:** variable que denota al símbolo leído en el tope de la pila,

Alfabeto de la pila: Por cada operador unario o binario o utilizaremos dos símbolos o' y o. Ambos "recuerdan" que se ha leído el operador o. El primero indica que aún falta por leerse un argumento. También utilizaremos dos paréntesis izquierdos $('$ y $($.

Algoritmo:

1. Inicialmente, se está en el estado inicial, se está leyendo el extremo izquierdo de la cadena de entrada y en la pila se tiene en su tope al símbolo inicial X. Es decir, e es el primer símbolo de la cadena de entrada y $t = X$. La cadena de salida está en blanco.
2. En función del valor de e, procédase según sea el caso:
   (a) $e = ($: Empílese, en principio, todo paréntesis izquierdo como $('$. **Si** $t = o'$, para algún operador o, **entonces** sustitúyase o' por su respectivo símbolo o.
   (b) $e \in T_0$ (se lee un factor): Escríbase e.
   Si $t = o'$, para algún operador o, **entonces** sustitúyase o' por su respectivo símbolo o.
   Si $t = ('$ **entonces** sustitúyase $('$ por $($.
   (c) $e = o_1 \in Oper_1$ (se lee un operador unario): Procédase según el tope de la pila.
      i. **Si** $t = o'$, para algún operador o, **entonces** sustitúyase o' por o.
   Empílese o'_1.
   (d) $e = o_2 \in Oper_2$: (se lee un operador binario): Procédase según el tope de la pila.
      i. En tanto $t = o$, para algún operador binario o con $Prior(o) \geq Prior(o_2)$: Desempílese o.
      ii. $t = o'$, para algún operador binario o: La cadena de entrada está mal formada. En este caso, termínese el procedimiento.
      iii. $t = ('$: La cadena de entrada está mal formada. En este caso, termínese el procedimiento.
   Empílese o'_2.
   (e) $e =)$: Desempílese uno a uno todos los operadores en la pila hasta encontrar $($ en el tope de la pila y entonces suprímasele.
   Si en el transcurso de tal acción apareciese un símbolo primado o bien no apareciese tal $($ **entonces** la cadena de entrada está mal formada. En este caso, termínese el procedimiento.
   (f) $e = nil$: Desempílese uno a uno todos los operadores en la pila hasta vaciarla, i.e. hasta llegar a X.
   Si en el transcurso de tal acción apareciese un símbolo primado o bien un paréntesis izquierdo $($ **entonces** la cadena de entrada está mal formada. En este caso, termínese el procedimiento.

En el caso de autómatas de pila con escritura, una *descripción instantánea (DI)* es una cadena

$$q\sigma;\tau;\omega \in Q \times Ent^* \times AlfP^* \times Sal^*$$

que indica que el autómata está en el estado q, se está leyendo el primer símbolo a la izquierda de σ, τ es el contenido de la pila y ω es la cadena inscrita en la salida.

En la figura 5.1 presentamos dos computaciones del autómata arriba descrito para sendas cadenas de entrada. La de la izquierda está bien formada mientras que la de la derecha no lo está. En cada rengón sólo ponemos la pareja $\tau;\omega$, correspondientes a la pila (el tope es su estremo izquierdo) y a la salida de cualquier DI. Cada rengón corresponde a un símbolo leído en la cadena de entrada.

$K \uparrow (A \cdot X \uparrow 2 + B \cdot X + (-C))$		
	;	K
$\uparrow'$	;	K
$('\uparrow$	;	K
$(\uparrow$	;	KA
$\cdot'(\uparrow$	;	KA
$\cdot(\uparrow$	;	KAX
$\uparrow'\cdot(\uparrow$	;	KAX
$\uparrow\cdot(\uparrow$	;	$KAX2$
$+'(\uparrow$	;	$KAX2\uparrow\cdot$
$+(\uparrow$	;	$KAX2\uparrow\cdot B$
$\cdot' + (\uparrow$	;	$KAX2\uparrow\cdot B$
$\cdot + (\uparrow$	;	$KAX2\uparrow\cdot BX$
$+'(\uparrow$	;	$KAX2\uparrow\cdot BX\cdot +$
$('+(\uparrow$	;	$KAX2\uparrow\cdot BX\cdot +$
$-'(+(\uparrow$	;	$KAX2\uparrow\cdot BX\cdot +$
$-(+(\uparrow$	;	$KAX2\uparrow\cdot BX\cdot +$
$-(+(\uparrow$	;	$KAX2\uparrow\cdot BX\cdot +C$
$+(\uparrow$	;	$KAX2\uparrow\cdot BX\cdot +C-$
$\uparrow$	;	$KAX2\uparrow\cdot BX\cdot +C-+$
	;	$KAX2\uparrow\cdot BX\cdot +C-+\uparrow$

$A + B + C \uparrow 2-$		
	;	A
$+'$	;	A
$+$	;	AB
$+'$	;	$AB+$
$+$	;	$AB+C$
$\uparrow'+$	;	$AB+C$
$\uparrow+$	;	$AB+C2$
$+$	;	$AB+C2\uparrow$
$-'$	;	$AB+C2\uparrow+$
$-'$	;	$AB+C2\uparrow+$
		¡Mal formada!

Figura 5.1: Conversiones a notación polaca.

Capítulo 6

Lenguajes libres de contexto

6.1 Arboles de derivación

6.1.1 Gráficas y árboles

Recordamos que una gráfica $G = (V, A)$ consta de un conjunto de *vértices* $V = \{v_0, \ldots, v_{n-1}\}$ y de un conjunto de *aristas* A donde cada arista es una pareja de vértices. Para una arista $a = \{v_i, v_j\}$ se dice que la arista *incide* en sus *extremos* v_i, v_j y que éstos son *adyacentes*. Para cada vértice $v \in V$, el *grado* o la *valencia* de v es el número de vértices que le son adyacentes:

$$\partial(v) = \text{card}\{u \in V | \exists a \in A : a = \{v, u\}\}.$$

Un *camino* entre dos vértices u, v es una sucesión de aristas $[a_1, \ldots, a_k]$ tal que u es un extremo de a_1, v lo es de a_k y cualesquiera dos aristas contiguas tienen un extremo común.

Una gráfica *dirigida* es una gráfica tal que cada arista es un elemento del producto cartesiano $V \times V$, es decir, cada arista posee un *inicio* y un *fin*. En tal caso, las valencias *externa* e *interna* cuentan, respectivamente, las aristas *que salen de* y las *que entran a* ese vértice, es decir, para cualquier $v \in V$:

$$\begin{aligned} \partial(v)^+ &= \text{card}\{u \in V | \exists a \in A : a = (v, u)\}, \\ \partial(v)^- &= \text{card}\{u \in V | \exists a \in A : a = (u, v)\}. \end{aligned}$$

Un *árbol* es una gráfica dirigida $Ar = (V, A)$ tal que

- Existe un único vértice $r \in V$ de grado interior cero. Ese vértice se dice ser la *raiz* de Ar.
- Cualquier vértice que no es raiza posee un grado interior igual a 1.

Un vértice $v \in V$ es *interior* si su grado exterior es mayor que 0. Los vértices que no son interiores son *hojas*. Si $(u, v) \in A$ entonces v se dice ser un *hijo* de u y u el *padre* de v. Dos vértices con un mismo padre se dicen ser *hermanos*.

Observación 6.1.1 *1. Cada vértice $v \in V$ tiene un camino único que lo conecta con la raiz. El número de aristas en ese camino es la* altura *del vértice. Los vértices visitados por ese camino son los* ancestros *de v.*

2. Para cada vértice $u \in V$ el conjunto $Ar_u = \{v | u$ ancestro de $v\}$ adquiere naturalmente una estructura de árbol. Se dice ser el subárbol enraizado *en u.*

3. En cada vértice $u \in V$ el subárbol Ar_u es la unión de $\{u\}$ junto con los subárboles enraizados en los hijos de u.

Todo árbol tiene una estructura de conjunto ordenado mediante el orden

$$u \leq_{Ar} v \Leftrightarrow (u \text{ es ancestro de } v).$$

Sin embargo, el árbol se dice *ordenado* sólo si en cada vértice interior el conjunto de sus hijos posee un orden lineal, es decir, a los hijos $v_{i_1}, \ldots, v_{i_k}$ de todo vértice u se los puede ordenar de *menor* a *mayor*. Denotemos por $\leq_u$ al orden de los hijos de u.

Para cada familia $\mathcal{O} = \{\leq_u | u \text{ interior en } Ar\}$ de órdenes en hijos de vértices interiores se puede definir, por ejemplo, los órdenes siguientes en el árbol Ar:

Preorden: Cada padre precede a sus hermanos mayores y a los vértices del que es ancestro.

Entreorden: En cada vértice interior se divide al conjunto de sus hijos en dos conjuntos $Primeros_u$ y $Ultimos_u$ de manera que para cualquier pareja $(v_1, v_2) \in Primeros_u \times Ultimos_u$ se cumpla $v_1 \leq_u v_2$.

En el *entreorden* se tiene que cada padre precede a sus hermanos mayores y a sus últimos hijos, pero sucede a sus hijos primeros.

Postorden: Cada padre sucede a sus hermanos menores y a los vértices del que es ancestro.

Cualquiera de estos órdenes es un orden de *izquierda a derecha*: $\preceq$ es un orden de *izquierda a derecha* si cualesquiera dos vértices con subárboles ajenos son tales que todos los vértices en el subárbol de uno anteceden a cualquier vértice en el subárbol del otro, i. e. $\forall u, v \in V$:

$$Ar_u \cap Ar_v = \emptyset \Rightarrow \begin{cases} (u_1 \in Ar_u \& v_1 \in Ar_v \quad \Rightarrow \quad u_1 \preceq v_1) \\ \text{o bien} \\ (u_1 \in Ar_u \& v_1 \in Ar_v \quad \Rightarrow \quad v_1 \preceq u_1) \end{cases}$$

Todo orden de izquierda a derecha es total: cualesquiera dos vértices en el árbol son comparables.

En un tal orden, la hoja mínima se dice ser la hoja *siniestra (leftmost)* y la máxima es la hoja *diestra (rightmost)*.

6.1.2 Arboles y gramáticas

Sea $G = (V, T, P, s_0)$ una gramática libre de contexto.

Un *árbol gramatical* en G es un árbol ordenado Ar tal que

- los vértices de Ar están etiquetados con etiquetas en el alfabeto de la gramática $V \cup T$ o con la etiqueta vacía, *nil*, y
- cada vértice interior corresponde a una producción en G, es decir, si A es la etiqueta del vértice interior v_0 y $\sigma = [e(v)|v \text{ es hijo de } v_0]$ es la palabra formada por las etiquetas de los hijos de v_0 entonces $(A \to \sigma)$ es una producción en P.

Un árbol gramatical es de *derivación* si su raiz está etiquetada con el símbolo inicial s_0 y todas sus hojas tienen etiquetas terminales o vacía. La *leyenda* de un árbol de derivación es la palabra obtenida al concatenar ordenadamente, con cualquier orden de izquierda a derecha, las etiquetas de sus hojas.

Así pues, todo árbol de derivación tiene como leyenda una palabra en $L(G)$. Recíprocamente, para cada palabra σ en $L(G)$ existe un árbol de derivación cuya leyenda coincide con σ.

Si $\sigma = \sigma_1 A \sigma_2$ y $\tau = \sigma_1 \beta \sigma_2$, donde $(A \to \beta)$ es una producción en P y σ_2 consiste únicamente de símbolos terminales, decimos que τ se sigue de σ por la aplicación *diestra* de una producción. Una derivación es *diestra* si todas las aplicaciones de producciones hechas son diestras. Las derivaciones *siniestras* se definen simétricamente.

Ejemplo.

Consideremos las siguientes producciones:

1.	S	$\to$	a	*termínese con una última "a"*
2.	S	$\to$	aAS	*colóquese una "a" a la izquierda, recuérdese esto y reiníciese,*
3.	A	$\to$	SbA	*antes de cualquier "b" han de ir "a"-es,*
4.	A	$\to$	ba	*colóquese una segunda b y termínese con "a",*
5.	A	$\to$	bb	*el lenguaje es cerrado por concatenación.*

Sendas derivaciones siniestra y diestra de la palabra $a(ba^2)^2 = abaabaa$ son las siguientes:

$\underline{S}$	$\underline{S}$
$\uparrow$	$\uparrow$
2	2
$a\ \underline{A}\ S$	$aA\ \underline{S}$
$\uparrow$	$\uparrow$
4	2
$aba\ \underline{S}$	$aAaA\ \underline{S}$
$\uparrow$	$\uparrow$
2	1
$abaa\ \underline{A}\ S$	$aAa\ \underline{A}\ a$
$\uparrow$	$\uparrow$
4	4
$abaaba\ \underline{S}$	$a\ \underline{A}\ abaa$
$\uparrow$	$\uparrow$
1	4
$abaabaa$	$abaabaa$

En el lenguaje $L(G)$ generado por esta gramática se encuentran incluidos los siguientes lenguajes:

- $\{a^{3n+1} | n \geq 0\}$.
  De hecho, $L(G) \cap a^* = \{a^k | k \equiv 1 \bmod 3\}$.
- $\{a^n bxa | n \geq 0, (n \equiv 1 \bmod 3 \Rightarrow x = a), (n \not\equiv 1 \bmod 3 \Rightarrow x = aa)\}$.
  De hecho, $G \vdash S \stackrel{*}{\rightarrow} a^n bxS$, con $n \geq 0$ y x como antes.
- $\{\pi_n\}_{n \geq 1}$, donde $\pi_n = aba^2b \cdots ba^n bx_n$ y

$$x_n = \begin{cases} aaa & \text{si } n \equiv 0 \bmod 3, \\ aa & \text{en otro caso.} \end{cases}$$

Para concluir esta sección presentaremos las nociones de *ambigüedad* en gramáticas.

Una gramática libre de contexto $G = (V, T, P, s_0)$ se dice ser *ambigua* si alguna palabra de su lenguaje, $\sigma \in L(G)$ posee dos derivaciones diestras.

Ejemplos.

1. La gramática G cuyas producciones son $S \rightarrow SbS | a$ es ambigua pues la palabra $(ab)^2a$ posee dos derivaciones diestras:

$\underline{S}$	$\underline{S}$
$\uparrow$	$\uparrow$
1	1
$\underline{S}\ bS$	$\underline{S}\ bS$
$\uparrow$	$\uparrow$
2	1
$ab\ \underline{S}$	$\underline{S}\ bSbS$
$\uparrow$	$\uparrow$
1	2
$ab\ \underline{S}\ bS$	$ab\ \underline{S}\ bS$
$\uparrow$	$\uparrow$
2	2
$abab\ \underline{S}$	$abab\ \underline{S}$
$\uparrow$	$\uparrow$
2	2
$ababa$	$ababa$

2. Si para la gramática $G = (V, T, P, s_0)$ incluimos una copia V' del conjunto de variables no-iniciales más copias de las producciones de P con símbolos en S' entonces la nueva gramática $G' = (V \cup V', T, P \cup P', s_0)$ es ambigua pues cada derivación diestra puede derivarse considerando símbolos en V o en su contraparte V'.

6.2 Transformaciones equivalentes de gramáticas

Recordamos que una gramática formal $G = (V, T, P, S)$, donde V es el conjunto de símbolos variables, T es el conjunto de símbolos terminales, P es el conjunto de producciones y S es el símbolo inicial, se dice ser *libre de contexto* si sus producciones son de la forma $X \to \sigma$, con $X \in V$ y $\sigma \in (V + T)^*$. Veremos en esta sección que toda gramática libre de contexto puede ser convertida, algorítmicamente, a gramáticas equivalentes que no poseen variables que no se utilicen o que no contengan produccciones con consecuente vacío o cuyas producciones sean una mera sustitución de variables.

A lo largo de esta sección, $G = (V, T, P, S)$ denotará siempre una gramática libre de contexto.

6.2.1 Supresión de símbolos inútiles

Un símbolo variable $X \in V$ se dice ser *útil* si aparece en el transcurso de una derivación terminal, a partir del símbolo inicial. Es decir, si

$$\exists \sigma_{Izq}, \sigma_{Der} \in (V + T)^*, \tau \in T^* : \ G \vdash S \overset{*}{\to} \sigma_{Izq} X \sigma_{Der} \overset{*}{\to} \tau.$$

Un símbolo es *inútil* (¿qué más?) si no es útil.

Proposición 6.2.1 *Toda gramática libre de contexto $G = (V, T, P, S)$ se puede transformar en una gramática libre de contexto $G' = (V', T, P', S')$ equivalente a G sin variables inútiles.*

Para demostrar esta proposición es necesario, primero, reducir la gramática dada a una gramática en la que todo símbolo variable derive una palabra terminal. Se ha de ver luego que tal gramática reducida puede a su vez reducirse a otra en la que cualquier símbolo variable aparece en alguna cadena derivable a partir del símbolo inicial.

Al componer ambas transformaciones se tendrá la gramática cuya existencia se proclama en la proposición.

Lema 6.2.1 (Búsqueda hacia atrás) *Toda gramática libre de contexto $G = (V, T, P, S)$, con $L(G) \neq \emptyset$, se puede transformar en una gramática libre de contexto $G' = (V', T, P', S')$ equivalente a G tal que para cualquier $A \in V'$ existe $\sigma \in T^*$ con la propiedad $G' \vdash A \overset{*}{\to} \sigma$.*

En efecto, primeramente consideremos como *buenas* a las variables que generan palabras terminales en un solo paso, es decir, a aquellas que aparecen como antecedentes de producciones cuyos consecuentes son palabras terminales, incluso la vacía. Reiteremos este procedimiento, considerando como *buenas* a las variables que aparecen como antecedentes de producciones cuyos consecuentes son palabras sobre símbolos terminales o variables ya consideradas como buenas. El total de variables buenas da la gramática reducida que se busca.

En la figura 6.1 presentamos un seudocódigo de este procedimiento.

Lema 6.2.2 (Búsqueda hacia adelante) *Toda gramática libre de contexto $G = (V, T, P, S)$, se puede transformar en una gramática libre de contexto $G' = (V', T', P', S')$ equivalente a G tal que $T' \subset T$ y para cualquier símbolo $\xi \in V' \cup T'$ existen $\sigma_{Izq}, \sigma_{Der} \in (V' \cup T')^*$ con la propiedad $G' \vdash S' \overset{*}{\to} \sigma_{Izq} \xi \sigma_{Der}$.*

En efecto, ahora, primeramente consideremos como *buenos* a los símbolos que se generan a partir del inicial en un solo paso, es decir, a aquellos que aparecen en consecuentes de producciones cuyo antecedente es el símbolo inicial. Reiteremos este procedimiento, considerando como *buenos* a los símbolos que aparecen en consecuentes de producciones cuyos antecedentes son símbolos ya consideradas como buenos. El total de símbolos buenos da la gramática reducida que se busca.

En la figura 6.2 presentamos un seudocódigo de este procedimiento.

Si se aplica primero el lema 6.2.1 y luego el 6.2.2 a una gramática dada, se convierte ésta en una gramática equivalente sin símbolos inútiles. Si el orden de aplicación se invierte, i. e. primero se aplica el lema 6.2.2 y luego el 6.2.1, entonces no necesariamente se eliminan los símbolos inútiles, como se puede ver considerando la gramática

$$\begin{aligned} S &\to AB|s \\ A &\to s \end{aligned}$$

Input: A context-free grammar $G = (V, T, P, S)$.
Output: An equivalent grammar $G' = (V', T, P', S')$ such that

$$\forall A \in V' \exists \sigma \in T^* : \; G' \vdash A \stackrel{*}{\to} \sigma.$$

{ $OldV := \emptyset$;
$NewV := \{X \in V | \exists \sigma \in T^* : (X \to \sigma) \in P\}$;
while $OldV \neq NewV$ **do**
{ $OldV := NewV$;
$NewV := \{X \in V | \exists \sigma \in (T \cup OldV)^* : (X \to \sigma) \in P\}$
} ;
$V' := NewV$;
$P' := P \cap (V' \times (V' + T)^*)$;
$S' := S$
}

Figura 6.1: Búsqueda hacia atrás: Reducción a gramáticas cuyas variables todas derivan palabras terminales.

Input: A context-free grammar $G = (V, T, P, S)$.
Output: An equivalent grammar $G' = (V', T', P', S')$ such that $T' \subset T$ and

$$\forall \xi \in V' \cup T' \exists \sigma_{Left}, \sigma_{Right} \in (V' \cup T')^* : \; G' \vdash S' \stackrel{*}{\to} \sigma_{Left} \xi \sigma_{Right}.$$

{ $OldSym := \{S\}$;
$NewSym := \{\xi \in V \cup T | \exists \, (X \to \sigma) \in P : X \in (V \cap OldSym) \wedge (\xi \text{ appears in } \sigma)\}$;
while $OldSym \neq NewSym$ **do**
{ $OldSym := NewSym$;
$NewSym := \{\xi \in V \cup T | \exists \, (X \to \sigma) \in P : X \in (V \cap OldSym) \wedge (\xi \text{ appears in } \sigma)\}$
} ;
$V' := V \cap NewSym$;
$T' := T \cap NewSym$;
$P' := P \cap (V' \times (V' + T')^*)$;
$S' := S$
}

Figura 6.2: Búsqueda hacia adelante: Reducción a gramáticas cuyas variables todas se derivan a partir del símbolo inicial.

Si aplicamos 6.2.1 obtenemos la gramática $\{S \to s, A \to s\}$ y luego al aplicar 6.2.2 obtenemos $\{S \to s\}$, la cual ya no tiene símbolos inútiles.

Recíprocamente, si aplicamos 6.2.2 obtenemos la gramática $\{S \to AB|s, A \to s\}$ y luego al aplicar 6.2.1 obtenemos $\{S \to s, A \to s\}$, la cual contiene al símbolo inútil A.

6.2.2 Supresión de producciones vacías

Una *producción vacía* o *producción-nil* es una producción de la forma $X \to nil$.

Proposición 6.2.2 *Toda gramática libre de contexto $G = (V, T, P, S)$ se puede transformar en una gramática libre de contexto $G' = (V', T, P', S')$ sin variables inútiles ni producciones vacías que es* casi *equivalente a G, i.e. $L(G') = L(G) - \{nil\}$:*

$$\forall \sigma \in T^* - \{nil\} : \ (\sigma \in L(G') \ \Leftrightarrow \ \sigma \in L(G)) .$$

En efecto, dada una gramática G evitemos las producciones vacías. Definamos a las variables *anulables* de manera recursiva como sigue:

- Si $X \to nil$ es una producción vacía entonces X es *anulable*.
- Si $X_1, \ldots, X_k$ son anulables y $X \to X_1 \cdots X_k$ es una producción en la gramática entonces X es *anulable*.

Sea $V'' = V - \{X \in V | X \text{ es anulable}\}$ y sea P'' el conjunto de producciones construído a partir de P como sigue:

Para cada producción en P de la forma $X \to X_1 \cdots X_k$ y para cada $i \leq k$ hagamos $Y_i = X_i$ si $X_i \in V''$ y $Y_i = nil$ si $X_i \notin V''$. Si $Y_1 \cdots Y_k \neq nil$ entonces incorporemos la producción $Y \to Y_1 \cdots Y_k$ al conjunto P''.

Sea G' la gramática que se obtiene de suprimir símbolos inútiles en G''. G' es la gramática cuya existencia se proclama en la proposición.

6.2.3 Supresión de producciones unitarias

Una producción *unitaria* es una de la forma $X \to Y$ con $X, Y \in V$.

Proposición 6.2.3 *Toda gramática libre de contexto $G = (V, T, P, S)$ que no genere a la palabra vacía se puede transformar en una gramática libre de contexto $G' = (V', T, P', S')$ sin variables inútiles ni producciones vacías ni producciones unitarias equivalente a G.*

En efecto, dada una gramática G, construyamos, mediante la construcción anterior, la gramática G' equivalente a G sin variables inútiles ni producciones vacías. Si hubiese aún producciones unitarias podríamos acomodarlas todas en una lista como la siguiente:

$$\begin{array}{ccccccc} Y_{11} & \to & Y_{12} & \to & \cdots & \to & Y_{1k_1} \\ Y_{21} & \to & Y_{22} & \to & \cdots & \to & Y_{2k_2} \\ \vdots & & \vdots & & & & \vdots \\ Y_{m1} & \to & Y_{m2} & \to & \cdots & \to & Y_{mk_m} \end{array}$$

Así pues $\forall i \leq m \forall 1 \leq j \leq l \leq k_i : \ Y_{ij} \stackrel{*}{\to} Y_{il}$.

Pues bien, ahora para cada par de producciones $X \stackrel{*}{\to} Y$, $Y \to \sigma$ con $X, Y \in V, \sigma \in (V + T)^*$, a esas producciones las sustituimos por la producción $X \to \sigma$.

La gramática G' que así obtenemos es equivalente a G'', y por tanto a G, y es como la anunciada en la proposición.

6.3 Formas normales

6.3.1 Forma normal de Chomsky

Una gramática libre de contexto $G = (V, T, P, S)$ se dice estar en *forma normal de Chomsky* si sus producciones son de cualquiera de las dos formas

- $X \to YZ$ con $X, Y, Z \in V$, o bien
- $X \to a$ con $X \in V$ y $a \in T$.

Proposición 6.3.1 *Toda gramática libre de contexto $G = (V, T, P, S)$ que no genere a la palabra vacía se puede transformar en una gramática libre de contexto $G' = (V', T, P', S')$ en forma normal de Chomsky.*

En efecto, dada una gramática G, apliquemos el último procedimiento de la sección anterior para transformar a G en una gramática G'' sin variables inútiles ni producciones vacías ni producciones unitarias equivalente a G.

A las producciones que quedasen de la forma $X \to a$ con $X \in V$ y $a \in T$ las dejamos sin cambio alguno.

A cada producción de la forma $X \to \xi_1\xi_2\cdots\xi_k$, con $\xi_1\xi_2\cdots\xi_k \in (V+T)^*$ y $k \geq 2$, la transformamos en una sucesión de producciones de la forma siguiente:

A cada símbolo terminal $a \in T$ que aparezca en la palabra $\xi_1\xi_2\cdots\xi_k$ le asociamos una variable nueva X_a e incorporamos la producción $X_a \to a$.

Así pues las producciones que no sean de la forma $X \to a$ con X variable y a terminal, han de ser de la forma $X \to X_1X_2\cdots X_k$, con $X, X_1X_2\cdots X_k$ todos variables. Para cada una de estas últimas producciones introducimos $k-2$ nuevas variables $Y_1, \ldots Y_{k-2}$ e incorporamos la sucesión de producciones

$$\begin{array}{lcl} X & \to & X_1Y_1 \\ Y_1 & \to & X_2Y_2 \\ \vdots & & \vdots \\ Y_i & \to & X_{i+1}Y_{i+1} \\ \vdots & & \vdots \\ Y_{k-3} & \to & X_{k-2}Y_{k-2} \\ Y_{k-2} & \to & X_{k-1}X_k \end{array}$$

Obtenemos así la forma normal equivalente a la gramática dada.

6.3.2 Forma normal de Greibach

Una gramática libre de contexto $G = (V, T, P, S)$ se dice estar en *forma normal de Greibach* si sus producciones son de la forma

$$X \to a\sigma \quad \text{con } X \in V, a \in T, \sigma \in V^*.$$

Veremos que la construcción de formas normales de Greibach equivalentes a gramáticas dadas es procedimental.

Para cualquier gramática libre de contexto G, definamos, para cada variable $X \in V$, a los conjuntos siguientes:

$$\begin{array}{rcl} P(X) & = & \{\text{producciones en } G \text{ cuyo antecedente es } X\} \\ Q(X) & = & \{\text{producciones en } P(X) \text{ cuyo consecuente comienza también con } X\} \\ & : & (X \to \gamma) \in Q(X) \Leftrightarrow \exists\gamma' : \gamma = X\gamma' \\ R(X) & = & P(X) - Q(X) \\ & : & (X \to \gamma) \in R(X) \Leftrightarrow \exists\xi \in (V+T) - \{X\}, \gamma' : \gamma = \xi\gamma' \end{array}$$

Primeramente observemos que podemos "componer producciones" de manera que tengamos siempre una gramática equivalente a la gramática dada.

Lema 6.3.1 (Composición de producciones) *Si $(X \to \alpha_1 Y \alpha_2)$ es una producción en G y las producciones en $P(Y)$ pueden escribirse como $Y \to \beta_1 | \cdots | \beta_m$ entonces al sustituir $(X \to \alpha_1 Y \alpha_2)$ por las producciones $(X \to \alpha_1\beta_1\alpha_2 | \cdots | \alpha_1\beta_m\alpha_2)$, obtenemos una gramática equivalente a G.*

En efecto, en toda derivación terminal que aplique en un momento la producción $(X \to \alpha_1 Y \alpha_2)$, necesariamente se ha de aplicar una producción en $P(Y)$ para suprimir el símbolo Y.

Lema 6.3.2 (Transformación de producciones "reflexivas") *Para cada variable $X \in V$ enumeremos $Q(X)$ y $R(X)$ como*

$$\begin{aligned} Q(X) &= (X \to X\gamma_1 | \cdots | X\gamma_q) \\ R(X) &= (X \to \beta_1 | \cdots | \beta_r) \end{aligned}$$

Sea Z una variable que no ocurra en V. Sea $G_X = (V \cup \{Z\}, T, P_X, S))$ la gramática que se obtiene al sustituir el conjunto de producciones $P(X)$ por las producciones

$$\begin{aligned} X &\to \beta_1 | \cdots | \beta_r \\ X &\to \beta_1 Z | \cdots | \beta_r Z \\ \\ Z &\to \gamma_1 | \cdots | \gamma_q \\ Z &\to \gamma_1 Z | \cdots | \gamma_q Z \end{aligned}$$

En efecto, toda derivación siniestra, en la gramática original, de una palabra en $L(G)$ ha de determinar una derivación diestra de la misma palabra en la gramática transformada.

La demostración de la afirmación anterior es directa. Como mera ilustración, consideremos tan solo un ejemplo:

La gramática con producciones $\{S \to SAb|b,\ A \to ASa|a\}$ genera al lenguaje consistente de las palabras de la forma $b(ab)^*$. De acuerdo con la construcción anterior, como $Q(S) = \{S \to SAb\}$ y $R(S) = \{S \to b\}$, obtenemos la gramática

$$\begin{aligned} S &\to b \\ S &\to bZ \\ Z &\to Ab \\ Z &\to AbZ \end{aligned}$$

Presentemos sendas derivaciones, siniestra en la gramática original y diestra en la transformada, para la palabra *bababab*:

$\underline{S}$	S
$\uparrow$	$\uparrow$
1.1	$1'$
$\underline{S}\ Ab$	$b\ \underline{Z}$
$\uparrow$	$\uparrow$
1.1	$4'$
$\underline{S}\ AbAb$	$bAb\ \underline{Z}$
$\uparrow$	$\uparrow$
1.1	$4'$
$\underline{S}\ AbAbAb$	$bAbAb\ \underline{Z}$
$\uparrow$	$\uparrow$
1.2	$3'$
$b\ \underline{A}\ bAbAb$	$bAbAb\ \underline{A}\ b$
$\uparrow$	$\uparrow$
2.2	2.2
$bab\ \underline{A}\ bAb$	$bAb\ \underline{A}\ bab$
$\uparrow$	$\uparrow$
2.2	2.2
$babab\ \underline{A}\ b$	$b\ \underline{A}\ babab$
$\uparrow$	$\uparrow$
2.2	2.2
$bababab$	$bababab$

Veamos ahora el resultado principal de esta sección:

Input: A context-free grammar $G' = (V', T, P', S')$.
Output: An equivalent grammar $G'' = (V'', T, P'', S'')$ such that, if $X_i \to X\sigma$ is a production in P'', then there is a $j > i$ such that $X = X_j$.

```
{  NonProcessedVariables := V' ; ProcessedVariables := nil ;
   while NonProcessedVariables ≠ nil do
   {  Pop[X, NonProcessedVariables] ; Push[X, ProcessedVariables] ;
      for each (X → Yσ) , with Y in ProcessedVariables, do
      {  let P(Y) = {Y → β_1 | ··· | β_{k_j}} ;
         replace (X → Yσ) by the productions X → β_1σ | ··· | β_{k_j}σ
      } ;
      if there is a production of the form (X → Xσ) then
      {  add a new variable Y into NonProcessedVariables ;
         for each (X → Xσ) do
            replace (X → Xσ) by the productions Y → σ | σY ;
         for each (X → σ) that is not of the above forms do
            add the productions X → σY
      }
   } ;
   V'' := ProcessedVariables ;
   P'' := current set of productions ;
   S'' := S'
}
```

Figura 6.3: Modificaciónde producciones de acuerdo con el orden de V'.

Proposición 6.3.2 *Toda gramática libre de contexto $G = (V, T, P, S)$ que no genere a la palabra vacía se puede transformar en una gramática libre de contexto $G^* = (V^*, T, P^*, S^*)$ en forma normal de Greibach.*

Sea pues $G = (V, T, P, S)$ una gramática libre de contexto que no genere a *nil*. La transformación a una forma normal de Greibach la hacemos gradualmente mediante los pasos siguientes:

1. Sea $G' = (V', T, P', S')$ la forma normal de Chomsky de G.

2. Modificaremos a las producciones en P' para tenerlas tales que toda producción, cuyo consecuente se inicie con una variable, ha de ser de la forma $X_i \to X_j\sigma$ con $j > i$, para un cierto orden en el conjunto de variables actuales, digamos $V'' = \{X_1, \ldots, X_m\}$. Para esto apliquemos el procedimiento cuyo seudocódigo se presenta en la figura 6.3.

   Por lo visto en el lema 6.3.2, la gramática G'' así obtenida es equivalente a G.

3. Ahora, hecha la transformación anterior, se tiene que

   - la última variable X_m sólo puede ser antecedente de producciones cuyos consecuentes se inician con símbolos terminales,
   - las producciones en $P(X_{m-1})$ cuyos consecuentes se inician con X_m pueden transformarse, siguiendo el lema 6.3.1 de "Composición de producciones", en producciones equivalentes cuyos consecuentes se inician con símbolos terminales,
   - de manera sucesiva para $i = m - 2$ hasta $i = 1$ las producciones en $P(X_i)$ cuyos consecuentes se inician con algún X_j, con $j > i$, pueden transformarse, siguiendo el lema 6.3.1 de "Composición de producciones", en producciones equivalentes cuyos consecuentes se inician con símbolos terminales.

Con todas estas transformaciones la gramática resultante $G^* = (V^*, T, P^*, S^*)$ es, en efecto, equivalente a G y está en forma normal de Greibach.

Ejemplo:

Consideremos la gramática con símbolos variables $V = \{X_1, X_2, X_3\}$ y producciones

1. $X_1 \to X_2X_3$
2. $X_2 \to X_3X_1|b$
3. $X_3 \to X_1X_2|a$

la cual ya está en forma normal de Chomsky.

Transformémosla de acuerdo con el procedimiento anterior.

Observemos que las dos primeras producciones ya tienen el tipo de las buscadas en el paso 2. del procedimiento anterior. La tercera tiene un consecuente que se inicia con un símbolo variable anterior al de su propio antecedente. Compongamos pues la producción 3. con la 1. Obtenemos

4. $X_3 \to X_2X_3X_2|a$

la cual también tiene un consecuente que se inicia con un símbolo variable anterior al de su propio antecedente. Compongamos pues la producción 4. con la 2. Obtenemos

5. $X_3 \to X_3X_1X_3X_2|bX_3X_2|a$

la cual es del tipo "reflexivo". Para transformarla, introduzcamos una nueva variable Y_3. Obtenemos

6. $X_3 \to bX_3X_2Y_3|aY_3|bX_3X_2|a$
7. $Y_3 \to X_1X_3X_2Y_3|X_1X_3X_2$

Con esto terminamos el paso 2. del procedimiento anterior. El conjunto actual de producciones consta de las producciones 1., 2., 6. y 7. Pasemos pues al paso 3. del procedimiento.

Sustituyendo 6. en 2. obtenemos

8. $X_2 \to bX_3X_2Y_3X_1|aY_3X_1|bX_3X_2X_1|aX_1|b$

Sustituyendo 8. en 1. obtenemos

9. $X_1 \to bX_3X_2Y_3X_1X_3|aY_3X_1X_3|bX_3X_2X_1X_3|aX_1X_3|bX_3$

Sustituyendo 9. en 7. obtenemos 10 nuevas producciones

7. $Y_3 \to$
$$\begin{array}{rr} bX_3X_2Y_3X_1X_3X_3X_2Y_3| & bX_3X_2Y_3X_1X_3X_3X_2| \\ aY_3X_1X_3X_3X_2Y_3| & aY_3X_1X_3X_3X_2| \\ bX_3X_2X_1X_3X_3X_2Y_3| & bX_3X_2X_1X_3X_3X_2| \\ aX_1X_3X_3X_2Y_3| & aX_1X_3X_3X_2| \\ bX_3X_3X_2Y_3 & bX_3X_3X_2 \end{array}$$

En resumen, la gramática equivalente, en forma normal de Greibach, tiene como conjunto de variables a $V = \{X_1, X_2, X_3, Y_3\}$ y sus producciones son la 6., 8., 9. y 10.:

$$\begin{array}{rcl} X_1 & \to & bX_3X_2Y_3X_1X_3|aY_3X_1X_3|bX_3X_2X_1X_3|aX_1X_3|bX_3 \\ X_2 & \to & bX_3X_2Y_3X_1|aY_3X_1|bX_3X_2X_1|aX_1|b \\ X_3 & \to & bX_3X_2Y_3|aY_3|bX_3X_2|a \\ Y_3 & \to & \begin{array}{rr} bX_3X_2Y_3X_1X_3X_3X_2Y_3| & bX_3X_2Y_3X_1X_3X_3X_2| \\ aY_3X_1X_3X_3X_2Y_3| & aY_3X_1X_3X_3X_2| \\ bX_3X_2X_1X_3X_3X_2Y_3| & bX_3X_2X_1X_3X_3X_2| \\ aX_1X_3X_3X_2Y_3| & aX_1X_3X_3X_2| \\ bX_3X_3X_2Y_3 & bX_3X_3X_2 \end{array} \end{array}$$

Ejemplo *"Cadenas equilibradas de paréntesis"*: Consideremos la gramática $S \to ()|SS|(S)$ que genera a las cadenas equilibradas de paréntesis (CEP).

La forma normal de Chomsky de la gramática dada es

1. $S \to SS|P_{Izq}P_{Der}|P_{Izq}A$
2. $P_{Izq} \to ($
3. $P_{Der} \to)$
4. $A \to SP_{Der}$

Consideremos el siguiente orden de las variables: S, A, P_{Izq}, P_{Der}.

La producción 1. es "reflexiva". Introduzcamos una nueva variable, X_1. Al hacer la transformación pertinente, obtenemos:

5. $S \rightarrow P_{Izq}P_{Der}X_1|P_{Izq}AX_1|P_{Izq}P_{Der}|P_{Izq}A$
6. $X_1 \rightarrow SX_1|S$

La producción 4. tiene un consecuente que se inicia con una variable anterior a su antecedente. Al componer 4. con 5. obtenemos

7. $A \rightarrow P_{Izq}P_{Der}X_1P_{Der}|P_{Izq}AX_1P_{Der}|P_{Izq}P_{Der}P_{Der}|P_{Izq}AP_{Der}$

La producción 6. tiene un consecuente que se inicia con una variable anterior a su antecedente. Al componer 6. con 5. obtenemos

8. $$\begin{array}{rcll} X_1 \rightarrow & P_{Izq}P_{Der}X_1X_1| & P_{Izq}P_{Der}X_1| \\ & P_{Izq}AX_1X_1| & P_{Izq}AX_1| \\ & P_{Izq}P_{Der}X_1| & P_{Izq}P_{Der}| \\ & P_{Izq}AX_1 & P_{Izq}A \end{array}$$

Al componer 8. con 2. obtenemos

9. $$\begin{array}{rcll} X_1 \rightarrow & (P_{Der}X_1X_1| & (P_{Der}X_1| \\ & (AX_1X_1| & (AX_1| \\ & (P_{Der}X_1| & (P_{Der}| \\ & (AX_1 & (A \end{array}$$

En este momento, en nuestro conjunto actual de producciones 2., 3., 5., 7. y 9., ningún consecuente se inicia con alguna variable que anteceda a la variable en el antecedente de la producción correspondiente.

En el paso 3. del procedimiento para obtener formas normales de Greibach, hemos de sustituir la variable P_{Izq} por el símbolo (, según la producción 2., toda vez que aparezca al inicio del consecuente de alguna producción. Obtenemos así la gramática:

$$\begin{array}{rcll} P_{Izq} & \rightarrow & (\\ P_{Der} & \rightarrow &) \\ S & \rightarrow & (P_{Der}X_1|(AX_1|(P_{Der}|(A \\ A & \rightarrow & (P_{Der}X_1P_{Der}|(AX_1P_{Der}|(P_{Der}P_{Der}|(AP_{Der} \\ X_1 & \rightarrow & (P_{Der}X_1X_1| & (P_{Der}X_1| \\ & & (AX_1X_1| & (AX_1| \\ & & (P_{Der}X_1| & (P_{Der}| \\ & & (AX_1 & (A \end{array}$$

Y esta gramática ya queda en forma normal de Greibach. Observemos que la variable P_{Izq} es irrelevante. De hecho si hacemos la sustitución del otro símbolo P_{Der} obtenemos la gramática equivalente

$$\begin{array}{rcll} S & \rightarrow & ()X_1|(AX_1|()|(A \\ A & \rightarrow & ()X_1)|(AX_1)|())|(A) \\ X_1 & \rightarrow & ()X_1X_1| & ()X_1| \\ & & (AX_1X_1| & (AX_1| \\ & & ()X_1| & ()| \\ & & (AX_1 & (A \end{array}$$

que también está en forma normal de Greibach.

6.4 Lenguajes libres de contexto y autómatas de pila

Veamos que todo lenguaje libre de contexto es reconocido por una autómata de pila.

Proposición 6.4.1 *Para cualquier lenguaje libre de contexto L existe un autómata de pila $AutP = (Q, Ent, AlfP, tran, q_0, X, \emptyset$ que reconoce al lenguaje, i.e.: $L = LPV(AutP)$.*

Sea L un lenguaje libre de contexto y sea G una gramática libre de contexto que lo genere. Supongamos por un momento que la palabra vacía no pertenezca al lenguaje L. Podemos pues suponer que la gramática

$G = (V, T, P, S)$ está en forma normal de Greibach. Sea $AutP = (Q, Ent, AlfP, tran, q_0, X, F)$ donde

$$\begin{array}{rcl} Q = \{q_0\} & : & \text{consta de un único estado,} \\ Ent & : & \text{es el alfabeto del lenguaje } L, \\ AlfP = (V + T) & : & \text{(el alfabeto de la pila) se forma de los símbolos en la gramática} \\ tran & : & Q \times (Ent \cup \{nil\}) \times AlfP \to \mathcal{P}\left(Q \times AlfP^*\right), \text{ es la función de } \textit{transición}, \text{ tal que} \\ & & (q_0, \sigma) \in tran(q_0, e, A) \;\Leftrightarrow\; (A \to a\sigma) \in P \\ q_0 \in Q & : & \text{es el estado } \textit{inicial}, \text{ y,} \\ S & : & \text{el símbolo "inicial" para la pila es el inicial de la gramática.} \end{array}$$

La función del autómata definido es la de simular derivaciones siniestras, en la gramática G, de palabras en el lenguaje L. De hecho, se puede demostrar que se cumple la equivalencia

$$G \vdash \left(S \overset{*}{\Rightarrow} \sigma\right) \;\Leftrightarrow\; AutP \vdash \left(q_0\sigma; X_0 \overset{*}{\Rightarrow} q_0 nil; nil\right)$$

(La implicación "$\Rightarrow$)" se demuestra mediante inducción en el número de producciones utilizadas en una derivación siniestra de σ. El recíproco "$\Leftarrow$)" se demuestra mediante inducción en el número de transiciones aplicadas para derivar la descripción final $q_0 nil; nil$ a partir de la inicial $q_0\sigma; X_0$ en $AutP$.)

Ahora bien, si L contuviera a la palabra vacía nil, entonces, anulemos a todas las producciones-nil, o anulables, en una gramática que genere a $G_1 = G - \{nil\}$ y apliquemos lo anterior para encontrar un autómata de pila $AutP_1$ que reconozca a G_1. Ampliemos, finalmente, a $AutP_1$ añadiendo la transición $(q_0, nil) \in tran(q_0, nil, S)$.

6.5 Series de potencias e inversión de Lagrange

6.5.1 Series en diccionarios

Sea $G = (V, T, P, s_0)$ una gramática libre de contexto. Para cada símbolo variable $v \in V$ sea $s_G = \{\sigma \in T^* | G \vdash s \overset{*}{\to} \sigma\}$ el conjunto de palabras derivadas por s en G. Así pues, el lenguaje de G es, $(s_0)_G$. De hecho, para cada palabra $\sigma \in (V \cup T)^*$ definamos

$$\sigma_G = \{\sigma_1 \in T^* | G \vdash \sigma \overset{*}{\to} \sigma_1\}.$$

Consideremos la transformación $(V \cup T)^* \to \mathcal{P}((V \cup T)^*)$, $\sigma \mapsto \sigma_G$, del diccionario $(V \cup T)^*$ a la clase de lenguajes $\mathcal{P}((V \cup T)^*)$.

La operación "suma" de $\mathcal{P}((V \cup T)^*)$ es la unión conjuntista y su producto es el obtenido "al concatenar lenguajes"

$$L_1 \cdot L_2 = \{\tau_1\tau_2 | \tau_1 \in L_1, \tau_2 \in L_2.$$

$(\mathcal{P}((V \cup T)^*), +, \cdot)$ tiene una estructura de *semianillo no-conmutativo*, es decir,

- $(\mathcal{P}((V \cup T)^*), +)$ es un monoide conmutativo con unidad el conjunto vacío, $0 = \emptyset$,
- $(\mathcal{P}((V \cup T)^*), \cdot)$ es un monoide con unidad la palabra vacía, $1 = nil$, y
- $\cdot$ se distribuye por ambos lados respecto a $+$:

$$\begin{array}{rcl} L_1(L_2 + L_3) & = & L_1L_2 + L_1L_3 \\ (L_2 + L_3)L_1 & = & L_2L_1 + L_3L_1 \end{array}$$

Ahora, para $(V \cup T)^*$ la "alternancia" puede ser vista como una suma, y por tanto $(\sigma_1 + \sigma_2)_G = (\sigma_1)_G + (\sigma_2)_G$, y, ya que la gramática G es libre de contexto, se tiene también la identidad $(\sigma_1\sigma_2)_G = (\sigma_1)_G(\sigma_2)_G$. Así pues la transformación $\sigma \mapsto \sigma_G$ es un homomorfismo de semianillos.

Consecuentemente, toda gramática libre de contexto puede ser presentada de manera algebraica: A cada producción de la forma $s \to \sigma_1|\cdots|\sigma_k$ se la escribe equivalentemente como $(s)_G = \sum_{\kappa=1}^{k}(\sigma_\kappa)_G$, o bien, dando la gramática como supuesta, simplemente como la *ecuación básica* $s = \sum_{\kappa=1}^{k}\sigma_\kappa$.

Ejemplos.

1. La gramática de cadenas equilibradas de paréntesis, $S \to ()|(S)|SS$ da la ecuación básica: $S = () + (S) + SS$.

2. La gramática de palíndromas $S \to nil|aSa|bSb$ da la ecuación básica: $S = 1 + aSa + bSb$.

3. La gramática con producciones

$$\begin{array}{lcl} S & \to & a \\ S & \to & aAS \\ A & \to & SbA \\ A & \to & SS \\ A & \to & ba \end{array}$$

da las ecuaciones básicas

$$\begin{array}{lcl} S & = & a(1 + AS) \\ A & = & ba + S(bA + S) \end{array}$$

Mediante sustituciones sucesivas se puede *generar* nuevas ecuaciones a partir de ecuaciones generadas previamente:

- toda ecuación básica es *generada*,
- si $s = F(\ldots, s_1, \ldots)$ es básica y $s_1 = G(V,T)$ ha sido generada entonces al sustituir s_1 por $G(V,T)$ en $s = F(\ldots, s_1, \ldots)$,
$$s = F(\ldots, s_1, \ldots)|_{s_1 = G(V,T)}$$
da una nueva ecuación generada.

La *serie generativa* del lenguaje $L(G)$ se obtiene al tomar al "supremo" de todas las ecuaciones generadas a partir de la ecuación básica que tiene como extremo izquierdo al símbolo inicial.

Como mero ejemplo, es fácil ver que para la ecuación básica $S = 1 + aSa + bSb$ la serie generativa quedará de la forma

$$S = 1 + \sum_{n=1}^{+\infty} \sum_{\sigma \in (a+b)^n} \sigma \cdot reverso(\sigma).$$

6.5.2 Series sobre los racionales

Sea $\mathbf{X} = (X_1, \ldots, X_n)$ una lista de n variables. Una *serie racional* sobre $\mathbb{Q}$ es de la forma $p(\mathbf{X}) = \sum_{\mathbf{i} \in \mathbb{N}^n} p_{\mathbf{i}} \mathbf{X}^{\mathbf{i}}$, con $p_{\mathbf{i}} \in \mathbb{Q}$, y $\mathbf{X}^{\mathbf{i}} = X_1^{i_1} \cdots X_n^{i_n}$. Denotamos por $\mathbb{Q}[[\mathbf{X}]]$ a la colección de todas las series racionales sobre $\mathbf{X}$.

La siguiente proposición es verdadera:

Proposición 6.5.1 *$\mathbb{Q}[[\mathbf{X}]]$ tiene una estructura de anillo conmutativo con la suma y el producto convencionales de series.*

Más aún, forma un dominio entero*:*

$$p(\mathbf{X})q(\mathbf{X}) = 0 \Rightarrow (p(\mathbf{X}) = 0) \text{ ó } (q(\mathbf{X}) = 0).$$

Además, los elementos invertibles en $\mathbb{Q}[[\mathbf{X}]]$ son aquellos cuyo coeficiente "independiente" no es nulo:

$$\exists q(\mathbf{X}) \in \mathbb{Q}[[\mathbf{X}]] : \left(\sum_{\mathbf{i} \in \mathbb{N}^n} p_{\mathbf{i}} \mathbf{X}^{\mathbf{i}} \right) q(\mathbf{X}) = 1 \Leftrightarrow p_{\mathbf{0}} \neq 0.$$

En el dominio de series $\mathbb{Q}[[\mathbf{X}]]$ es posible , bajo determinadas circunstancias, resolver ecuaciones de la forma $\mathbf{X} = F(\mathbf{X})$.

Proposición 6.5.2 *La ecuación* $X = \frac{1}{1-p(\mathbf{X})}$ *tiene como solución* $X = \sum_{m=0}^{+\infty} (p(\mathbf{X}))^m$.

En efecto, $\sum_{m=0}^{M} (p(\mathbf{X}))^m = \frac{1-(p(\mathbf{X}))^{M+1}}{1-p(\mathbf{X})}$. Así pues, en el conjunto de puntos $\mathbf{x} \in \mathbb{Q}^n$ tales que $p(\mathbf{x}) < 1$ se ha de tener que $\sum_{m=0}^{+\infty} (p(\mathbf{x}))^m = \frac{1}{1-p(\mathbf{X})}$. La igualdad de la serie con una función analítica dentro de un conjunto en $\mathbb{Q}^n$ con interior no vacío hace que la igualdad valga en el dominio $\mathbb{Q}[[\mathbf{X}]]$.

Proposición 6.5.3 *La ecuación* $X = \sum_{k=0}^{K} a_k(Y)X^k$, *donde cada coeficiente* a_k *es una forma polinomial en* Y, *se resuelve mediante un sistema de ecuaciones recurrentes.*

En efecto, escribamos $X = \sum_{m=0}^{+\infty} b_m Y^m$. De la ecuación, se tiene

$$\sum_{k=2}^{K} a_k(Y)X^k + (a_1(Y)-1)X + a_0(Y) = 0,$$

al sustituir X se tiene

$$\begin{aligned} 0 &= \sum_{k=2}^{K} a_k(Y)X^k + (a_1(Y)-1)X + a_0(Y) \\ &= \sum_{k=2}^{K} a_k(Y)\left(\sum_{m=0}^{+\infty} b_m Y^m\right)^k + (a_1(Y)-1)\left(\sum_{m=0}^{+\infty} b_m Y^m\right) + a_0(Y) \qquad (6.1) \\ &= \sum_{j=0}^{J} g_j(\mathbf{b})Y^j \end{aligned}$$

donde cada g_j se obtiene al expander las expresiones en 6.1.

Así pues se ha de tener que, $\forall j \leq J : g_j(\mathbf{b}) = 0$.

Ejemplo.

Resolver en $\mathbb{Q}[[X,Y]]$ la ecuación

$$X^2 + (Y-1)X + Y = 0 \qquad (6.2)$$

Escribamos $X = \sum_{m=0}^{+\infty} b_m Y^m$. Al tomar cuadrados $X^2 = \sum_{m=0}^{+\infty} c_m Y^m$, donde para cada $m \geq 0$, $c_m = \sum_{k=0}^{m} b_{m-k}b_k$. Es decir,

$$c_m = \begin{cases} 2\left(\sum_{k=0}^{\frac{m-1}{2}} b_{m-k}b_k + b_{\frac{m+1}{2}}b_{\frac{m-1}{2}}\right) & \text{si } m \equiv 1 \bmod 2, \\ 2\left(\sum_{k=0}^{\frac{m-1}{2}} b_{m-k}b_k\right) + b_{\frac{m}{2}}^2 & \text{si } m \equiv 0 \bmod 2. \end{cases}$$

Al sustituir en 6.2 tenemos

$$\begin{aligned} 0 &= \sum_{m=0}^{+\infty} c_m Y^m + (Y-1)\sum_{m=0}^{+\infty} b_m Y^m + Y \\ &= (c_0 - b_0) + (c_1 - b_1 + b_0 + 1)Y + \sum_{m=2}^{+\infty} (c_m - b_m + b_{m-1})Y^m \end{aligned}$$

y consecuentemente se tiene el sistema de ecuaciones recurrentes

$$\begin{aligned} 0 &= c_0 - b_0 \\ 0 &= c_1 - b_1 + b_0 + 1 \\ \forall m \geq 2 : \quad 0 &= c_m - b_m + b_{m-1} \end{aligned}$$

Recordando las expresiones para c_m obtenemos el sistema de recurrencias equivalente

$$\begin{array}{rrcll} & 0 & = & b_0^2 - b_0 & = & b_0(b_0+1) \\ & 0 & = & c_1 - b_1 + b_0 + 1 & = & b_1(2b_0-1) + b_0 + 1 \\ \forall m \geq 2: & 0 & = & \sum_{k=0}^{m} b_k b_{m-k} - b_m + b_{m-1} & = & b_m(2b_0-1) + \left(\sum_{k=1}^{m-1} b_k b_{m-k}\right) + b_{m-1} \end{array}$$

Así pues, para b_0 hay dos posibles valores, 0,1.
Fijo b_0 los demás coeficientes se calculan recurrentemente mediante las relaciones

$$\begin{array}{rrcl} & b_1 & = & \dfrac{1+b_0}{1-2b_0} \\ \forall m \geq 2: & b_m & = & \dfrac{1}{1-2b_0}\left(b_{m-1} + \left(\displaystyle\sum_{k=1}^{m-1} b_k b_{m-k}\right)\right) \end{array}$$

Las dos soluciones corresponden precisamente a que la ecuación 6.2 es de grado 2 y por tanto tiene dos raices.

Un antiguo método para resolver en $\mathbb{Q}[[Y]]$ ecuaciones de la forma $X = F(X,Y)$, donde F es una función afín de la forma $F(X,Y) = Yf(X) + c$, es debido a Lagrange.

Proposición 6.5.4 (Lagrange) *Para una ecuación de la forma $X = Yf(X) + c$, donde $c \in \mathbb{R}$,*

- *$f(X) = \sum_{m\geq 0} e_m X^m$ es una función analítica, y*
- *el término "independiente" no es nulo, $e_0 \neq 0$,*

y para una función derivable $g : \mathbb{R} \to \mathbb{R}$ se tiene que la expresión $g(X)$ satisface la ecuación

$$g(X) = g(c) + \sum_{m\geq 1} \frac{1}{m!}\left[\frac{d^{m-1}}{dx^{m-1}}\left(g'(x)f(x)^m\right)\Big|_{x=c}\right] Y^m.$$

En particular, para $g(x) = x$, o sea, para la función identidad, se tiene

$$X = c + \sum_{m\geq 1} \frac{1}{m!}\left[\frac{d^{m-1}}{dx^{m-1}}\left(f(x)^m\right)\Big|_{x=c}\right] Y^m. \tag{6.3}$$

En efecto, supongamos por un momento que $c = 0$.
Puesto que $X = Yf(X)$ y $f(0) = e_0 \neq 0$ se tiene que $Y = \frac{X}{f(X)}$ puede expresarse como una serie de potencias $Y = \sum_{n\geq 1} a_n X^n$.
Dada la función g, se busca expresar a $g(X)$ como una serie de potencias en Y, es decir, se busca coeficientes $(b_m)_{m\geq 0} \subset \mathbb{Q}$ tales que

$$g(Y) = \sum_{\nu\geq 0} b_\nu Y^\nu \tag{6.4}$$

Como para $X = 0$ se tiene $Y = 0$, a fortiori

$$b_0 = g(0) \tag{6.5}$$

Por otro lado, al derivar, respecto a X la ecuación 6.4 se tiene

$$\frac{dg}{dX} = \sum_{\nu\geq 0} \nu b_\nu Y^{\nu-1}\frac{dY}{dX},$$

al multiplicar por $f(X)^m$, con $m \geq 1$, resulta

$$g'(X)f(X)^m = \sum_{\nu\geq 0} \nu b_\nu Y^{\nu-1} f(X)^m \frac{dY}{dX},$$

y como $f(X) = \frac{X}{Y}$ se tiene

$$g'(X)f(X)^m = \sum_{\nu \geq 0} \nu b_\nu Y^{\nu-m-1} X^m \frac{dY}{dX} \tag{6.6}$$

Estimemos cada uno de los sumandos en la serie de la derecha. Consideremos dos casos:

- Para $\nu = m$ se tiene $b_\nu Y^{\nu-m-1} X^m \frac{dY}{dX} = m b_m X^m \frac{1}{Y} \frac{dY}{dX}$. Ahora bien,

$$\frac{1}{Y}\frac{dY}{dX} = \frac{\left(\sum_{n\geq 1} n a_n X^{n-1}\right)}{\left(\sum_{n\geq 1} a_n X^n\right)} = \frac{1}{X} \frac{\left(1 + \sum_{n\geq 2} n \frac{a_n}{a_1} X^{n-1}\right)}{\left(1 + \sum_{n\geq 2} \frac{a_n}{a_1} X^{n-1}\right)}$$

  y como este último cociente es anaítico (bajo ciertas condiciones), existen coeficientes c_m tales que

$$\frac{1}{Y}\frac{dY}{dX} = \frac{1}{X}\left(1 + \sum_{\mu \geq 1} c_\mu X^\mu\right).$$

  Por tanto $m b_m X^m \frac{1}{Y}\frac{dY}{dX} = m b_m X^{m-1}\left(1 + \sum_{\mu\geq 1} c_\mu X^\mu\right)$.

  Así pues este sumando determina, para la potencia X^{m-1}, el coeficiente $m b_m$.

- Para $\nu \neq m$ se tiene

$$\nu b_\nu Y^{\nu-m-1} X^m \frac{dY}{dX} = \nu b_\nu X^m \frac{1}{\nu - m} \frac{dY^{\nu-m}}{dX}.$$

  Al calcular la derivada $\frac{dY^{\nu-m}}{dX}$ se observa que el término $\frac{1}{X}$ no aparece y por tanto este sumando no afecta a la potencia X^{m-1}.

Ahora bien, al derivar la ecuación 6.6 $m-1$ veces respecto a X, en el punto $X = 0$, se obtiene el coeficiente de la potencia X^{m-1} multiplicado por el factorial $m!$. Es decir,

$$\frac{d^{m-1}}{dX^{m-1}} (f(X)^m)\Big|_{X=0} = (m-1)! m b_m,$$

y en consecuencia

$$b_m = \frac{1}{m!} \frac{d^{m-1}}{dX^{m-1}} (f(X)^m)\Big|_{X=0}.$$

Así pues, esto último junto con 6.5 y 6.4 da

$$g(X) = g(0) + \sum_{m\geq 1} \frac{1}{m!} \left[\frac{d^{m-1}}{dx^{m-1}} (g'(x)f(x)^m)\Big|_{x=0}\right] Y^m. \tag{6.7}$$

Finalmente, para $c \neq 0$, renombremos como Z a la variable en 6.7, y luego introduzcamos el cambio de variable $X = Z + c$. Se obtiene así la ecuación

$$g(X) = g(0) + \sum_{m\geq 1} \frac{1}{m!} \left[\frac{d^{m-1}}{dx^{m-1}} (g'(x)f(x)^m)\Big|_{x=c}\right] Y^m.$$

6.5.3 Conteo de árboles de derivación

Para una gramática libre de contexto $G = (V, T, P, s_0)$, su serie generativa puede hacérse corresponder con una serie en $\mathbb{Q}[[V \cup T]]$ de manera que las ecuaciones impuestas por las producciones en P se satisfagan también en $\mathbb{Q}[[V \cup T]]$.

En efecto, a cada símbolo s en $V \cup T$ asociémosle una incógnita X_s. Sea $\mathbf{X} = [X_s | s \in V \cup T]$. Tratemos ahora el producto de incógnitas como si fuera conmutativo, pues de hecho lo es en $\mathbb{Q}[[\mathbf{X}]]$. Con esto, la serie generativa p_L de $L(G)$ corresponde con una serie $\Phi(p_l)$ en $\mathbb{Q}[[\mathbf{X}]]$.

Observamos dos características:

- la serie generativa no involucra a los símbolos de V, pues esta serie se expande hasta la supresión de los símbolos variables, y
- todas las palabras que son permutaciones de un mismo "multiconjunto" se símbolos corresponden a un mismo monomio en $\mathbb{Q}[[\mathbf{X}]]$:

$$\begin{aligned} ababab, abbbaa, bababa, \ldots &\quad \text{corresponden a} \quad X_a^3 X_b^3 \\ abababb, abbbaba, babbaba, \ldots &\quad \text{corresponden a} \quad X_a^3 X_b^4 \end{aligned}$$

Consecuentemente, el coeficiente de un monomio en la serie $\Phi(p_l)$ es el número de posibles árboles de derivación derivables en G correspondientes a una palabra sobre el multiconjunto correspondiente al monomio.

Esta transformación da así un método para contar derivaciones. Si la gramática no es ambigua, entonces cada palabra del lenguaje corresponde a un único árbol y, consecuentemente, éste procedimiento cuenta palabras generadas.

Ejemplos.

1. Para la gramática que genera palíndromas pares ($S \to aSa|bSb|nil$) o sea $S = aSa + bSb + 1$ consideremos la correspondencia inducida por

$$\begin{aligned} S &\mapsto X \\ a &\mapsto Y \\ b &\mapsto Z \end{aligned}$$

La producción da la ecuación

$$X = YXY + ZYZ + 1 = X(Y^2 + Z^2) + 1$$

y por tanto $X = \frac{1}{1-(Y^2+Z^2)}$.

La serie $\Phi(p_L)$ correspondiente a la serie generativa ha de ser solución de esta última ecuación. Por tanto, según se vió en la proposición 6.5.2 de la anterior sección,

$$\begin{aligned} X &= \sum_{m \geq 0} \left(Y^2 + Z^2\right)^m \\ &= \sum_{m \geq 0} \sum_{k=0}^{m} \binom{m}{k} Y^{2(m-k)} Z^{2k} \end{aligned}$$

Así pues para cada $m \geq 0$ hay $\displaystyle\sum_{k=0}^{m} \binom{m}{k} = 2^m$ posibles árboles para palabras de longitud $2m$, y para cada $k \leq m$ hay $\binom{m}{k}$ árboles para palabras de longitud $2m$ con $2k$ apariciones del símbolo b.

Como la gramática no es ambigua, la cuenta de árboles correponde a la cuenta de palabras.

2. Para la gramática de cadenas equilibradas de paréntesis $[S \to ()|(S)|SS]$ o sea $S = () + (S) + SS$ consideremos la correspondencia inducida por

$$\begin{aligned} S &\mapsto X \\ ) &\mapsto Y \\ (&\mapsto Z \end{aligned}$$

La producción da la ecuación

$$X = ZY + ZXY + XX = X^2 + X \cdot ZY + ZY$$

y por tanto $X^2 + X \cdot (ZY - 1) + ZY = 0$.

Consideremos por un momento $Z = 1$. Entonces tenemos la ecuación 6.2 de la sección anterior. Consecuentemente, X tiene dos soluciones de la forma $X = \sum_{m \geq 0} b_m Y^m$ donde $b_0 = 0, 1$ y

$$\begin{aligned} b_1 &= \frac{1 + b_0}{1 - 2b_0} \\ \forall m \geq 2: \quad b_m &= \frac{1}{1 - 2b_0} \left(b_{m-1} + \left(\sum_{k=1}^{m-1} b_k b_{m-k} \right) \right) \end{aligned}$$

En nuestro caso no hay ninguna paabra equilibrada con 0 paréntesis derechos. Así pues, el término independiente de la serie generativa debe ser $b_0 = 0$. Con esta elección de b_0 se tiene determinados a los demás coeficientes b_m. Cada uno de ellos da el número de árboles de derivación de cadenas equilibradas de paréntesis para formar cadenas equilibradas con m paréntesis derechos.

Como la gramática es ambigua el número de árboles excede el de cadenas equilibradas.

3. Consideremos la gramática con producciones

1.	S	$\to$	SbS	*intercálese una "b" entre cualquier par de palabras generables,*
2.	S	$\to$	a	*termínese con una "a".*

El lenguaje generado por la gramática es

$$L(G) = \{(ab)^n a | n \geq 0\}.$$

La gramática se expresa por la ecuación $S = SbS + a$ y mediante la correspondencia

$$\begin{aligned} S &\mapsto X \\ b &\mapsto Y \\ a &\mapsto 1 \end{aligned}$$

se tiene la ecuación $X = YX^2 + 1$.

Al resolverla con la fórmula de Lagrange 6.3 se tiene

$$X = 1 + \sum_{m \geq 1} \frac{1}{m!} \left[\frac{d^{m-1}}{dx^{m-1}} x^{2m} \Big|_{x=1} \right] Y^m.$$

Para cada $m \geq 1$ se tiene

$$\begin{aligned} \frac{1}{m!} \left[\frac{d^{m-1}}{dx^{m-1}} x^{2m} \Big|_{x=1} \right] &= \frac{1}{m!} (2m)(2m-1) \cdots (2m - (m-2)) x^{2m-(m-1)} \\ &= \frac{1}{m!} (2m)(2m-1) \cdots (m+2) x^{m+1} \\ &= \frac{1}{m!} \frac{(2m)!}{(m+1)!} x^{m+1} \\ &= \frac{1}{m+1} \binom{2m}{m} x^{m+1} \end{aligned}$$

Al evaluar en el punto $x = 1$ obtenemos

$$X = 1 + \sum_{m \geq 1} \frac{1}{m+1} \binom{2m}{m} Y^m$$

lo cual significa que para cada m hay $\frac{1}{m+1}\binom{2m}{m}$ derivaciones de la única palabra $(ab)^m a$ generable en la gramática que contiene exactamente m b's.

6.6 Programas

1. *Simulación de un autómata de pila*: Escriba un programa que reciba un autómata de pila para que dada una palabra cualquiera, represente, mediante descripciones instantáneas, la aplicación del autómata sobre la palabra. (Cfr. HU)

2. *Equivalencia de reconocimientos*: Escriba un programa que reciba un autómata de pila con *reconocimiento por arribo a estados finales* y construya el equivalente autómata de pila con *reconocimiento por pila vacía*. (Cfr. HU)

3. *Cálculo de formas normales de Greibach*: Escriba un programa que reciba una gramática libre de contexto y calcule su Forma Normal de Greibach equivalente. (Cfr. HU)

4. *Cálculo de formas reducidas "de Greibach"*: Escriba un programa que reciba una gramática libre de contexto y calcule su Forma Reducida "de Greibach" equivalente. Una tal forma reducida es como la de Greibach, con la salvedad de que sus producciones sólo pueden contener a lo sumo 2 símbolos variables en las consecuencias.

Bibliografía

[1] Aho, Ullman: *Foundations of computer science*, W. H. Freeman & Co., 1992

[2] Anderson, Turing *et al*: *Mentes y máquinas*, en la colección *Problemas científicos y filosóficos*, Universidad Nacional Autónoma de México, 1970

[3] Arbib: *The algebraic theory of machines, languages and semigrups*, Academic Press, 1968

[4] Arbib, Kfoury, Moll: *A basis for theoretical computer science*, Springer-Verlag, 1981

[5] Brzozowski: Derivates of Regular Expression, *Journal of the ACM*, 11, 1964

[6] Bucher, Hagauer: It is decidable whether a regular language is pure context-free, *TCS: Theoretical Computer Science*, 26, 1983

[7] Conway: *Regular algebra and finite machines*, Chapman & Hall, 1971

[8] Crochemore, Rytter: *Text algorithms*, Oxford University Press, 1994

[9] Davis: *The undecidable*, Raven Press, 1965

[10] Denning, Dennis, Qualitz: *Machines, languages and computation*, Prentice-Hall, 1978

[11] Eilenberg: *Automata, languages, and machines*, Volume A, Academic Press, 1974

[12] Ginzburg: *Algebraic theory of automata*, Academic Press, 1968

[13] Harrison: *Introduction to switching and automata theory*, McGraw-Hill Book company, 1965

[14] Harrison: *Introduction to formal languages theory*, Addison Wesley, 1978

[15] Hofstadter: *Gödel, Escher, Bach: An eternal golden braid*, Vintage Books, 1980

[16] Hopcroft, Ullman: *Introduction to automata theory, languages and computation*, Addison-Wesley, 1979

[17] Iwama: ASPACE($o(\log\log n)$) is regular, *SIAM Journal on Computing*, 22, 1993

[18] Kalmár: An argument against the plaussibility of Church's thesis, in Heyting (ed.) *Constructivity in matematics: Proceedings of the colloquium held in Amsterdam, 1957* , North-Holland, 1959

[19] Kfoury, Moll, Arbib: *A programming approach to computability*, Springer-Verlag, 1982

[20] Kohavi: *Switching and finite automata theory*, McGraw-Hill, 1979

[21] Kozen: A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, **110** (2), 1994

[22] Kuich, Salomaa: *Semirings, automata, languages*, Springer-Verlag, 1985

[23] Lewis, Papadimitiou: *Elements of the theory of computation*, Prentice Hall, 1981

[24] Lindgren, Nilsson, Nordahl, Räde: Regular language inference using evolving neural networks, in Whitley, Schaffer (ed's), *Proceedings of COGANN-92 International Workshop on Combinations of Genetic Algorithms and Neural Networks*, IEEE Computer Society Press, 1992

[25] Minsky: *Computation finite and infinite machines*, Prentice Hall, Englewood Cliffs, 1967

[26] Penrose: *The emperor's new mind: Concerning computers, minds and the laws of physics*, Oxford University Press, 1989

[27] Ponty, Ziadi, Champarnaud: A new quadratic algorithm to convert a regular expression into an automaton, in Raymond, Wood, Yu (ed's) *Automata implementation*, Lecture Notes in Computer Science, 1260, 1997

[28] Pospielov: *Teoría de juegos y autómatas*, Siglo XXI, 1969

[29] Rogers: *Theory of recursive functions and effective computability*, McGraw-Hill, 1967

[30] Rosenblueth: *Mente y cerebro: Una filosofía de la ciencia*, Siglo XXI, 1970

[31] Ross: Animating finite state automata: Having fun with theory, *SIGACTN: SIGACT News* (ACM Special Interest Group on Automata and Computability Theory), 28, 1997

[32] Rytter: *100 execises in the theory of automata and formal languages*, Research report 99, Dept. Comp. Sci., University of Warwick, 1987

[33] Salomaa: *Automata theory*, Pergamon Press, 1969

[34] Salomaa: *Formal languages*, Academic Press, 1973

[35] Shanon, McCarthy (ed's): *Automata studies*, Princeton University Press, 1956

[36] Takada: A hierarchy of language families learnable by regular language learners, *Lecture Notes in Computer Science*, 862, 1994

[37] Trajtenbrot: *Introducción a la teoría matemática de las computadoras y de la programación*, Siglo XXI, 1967