SSL 3

Algoritmos

(Operaciones con Autómatas Finitos y Máquina de Turing)

Jorge D. Muchnik, 2010

Para el ISBN

El libro “Sintaxis y Semántica de los Lenguajes” está formado por tres volúmenes:

Vol. 1 – Desde sus Usuarios (programadores y otros)

Vol.2 – Desde el Compilador

Vol.3 – Algoritmos

Este libro cubre todos los objetivos y los contenidos de la asignatura con el mismo nombre.

Agradezco a los profesores que integran la cátedra “Sintaxis y Semántica de los Lenguajes”, que siempre han apoyado mi gestión y han colaborado de distintas maneras para que este libro sea una realidad.

Por orden alfabético, los profesores de esta cátedra son: Adamoli, Adriana – Barca, Ricardo – Bruno, Oscar – Díaz Bott, Ana María – Ferrari, Marta – Ortega, Silvina – Sola, José Maria.

Jorge D. Muchnik, Titular
febrero 2010

ÍNDICE

En el Volumen 1 hemos hablado de Lenguajes Regulares y de Expresiones Regulares. En el Volumen 2 hemos hecho una introducción a los Autómatas Finitos, para luego aplicarlos a la construcción de Analizadores Léxicos.

En este Volumen veremos varias operaciones que facilitan y benefician el uso de los Autómatas Finitos en diversas aplicaciones. Y cerraremos el libro con una introducción al autómata más poderoso que, aunque no tiene utilidad a nivel de la Sintaxis y la Semántica de los Lenguajes de Programación, es muy importante conocerlo: la Máquina de Turing.

1 AFDs Y AFNs

Un Autómata Finito es un mecanismo que reconoce a un determinado Lenguaje Regular. "Reconocer a un lenguaje" significa aceptar cada cadena que pertenece al lenguaje (es decir, cada cadena que es una palabra) y rechazar toda cadena que no pertenece al lenguaje.

Los Autómatas Finitos que hemos descripto en el Volumen 2 pertenecen al grupo de autómatas denominado Autómatas Finitos Determinísticos (AFDs), cuya característica funcional es que para cualquier estado en que se encuentre el autómata en un momento dado, la lectura de un carácter determina, SIN AMBIGÜEDADES, cuál será el estado de llegada en la próxima transición.

Existe otro grupo de Autómatas Finitos, el de los *Autómatas Finitos No Determinísticos* (AFNs), que analizaremos hacia el final de este capítulo.

1.1 AUTÓMATAS FINITOS DETERMINÍSTICOS (AFDs)

Se ha especificado que un Autómata Finito es determinístico cuando, dado un carácter que es "leído" por el autómata, éste transita desde un estado de partida a un estado de llegada preciso, es decir: el estado de llegada está unívocamente determinado por el estado de partida y el carácter leído por el autómata.

Un AFD es una colección de tres conjuntos:

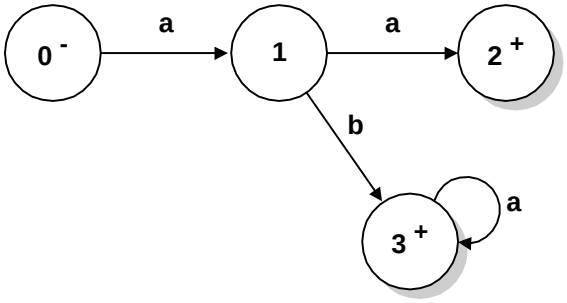
1) Un conjunto finito de estados, uno de los cuales se designa como el estado inicial (único) y otros (uno o más) son designados como estados finales o estados de aceptación;

2) Un alfabeto de caracteres con los que se forman las cadenas que serán procesadas por el AFD;

3) Un conjunto finito de transiciones que indican, para cada estado y para cada carácter del alfabeto, a qué estado debe "moverse" el AFD.

Ejemplo 1
A continuación se dibuja el Diagrama de Transiciones de un AFD formado por:

1) Conjunto de estados: 0, el estado inicial; 1, un estado intermedio; 2 y 3, estados finales.
2) Alfabeto: los caracteres a y b.
3) Conjunto de transiciones: 3.1) desde el estado 0 al estado 1 por el carácter a, 3.2) desde el estado 1 al estado 2 por el carácter a y al estado 3 por el carácter b, y 3.3) en el estado 3, un ciclo por el carácter a.



1.1.1 DEFINICIÓN FORMAL DE UN AFD

Formalmente, un AFD es una 5-upla (Q, Σ, T, q_0, F), donde:
– Q es un conjunto finito no vacío de estados,
– Σ es el alfabeto de caracteres reconocidos por el autómata,
– $q_0 \in Q$ es el estado inicial (único, no es un conjunto),
– $F \subseteq Q$ es el conjunto no vacío de estados finales, y
– T: $Q \times \Sigma \rightarrow Q$ es la función de transiciones, es decir: $T(q, x) = z$ significa que z es el estado al cual transita el autómata desde el estado q, al leer el carácter x, transición que también podemos simbolizar así: $q \Rightarrow x \Rightarrow z$. Generalmente, la función de transiciones es representada por medio de una Tabla de Transiciones (TT), tal como se muestra en el próximo ejemplo.

Ejemplo 2
El AFD del ejemplo anterior se define formalmente como: $M = (Q, \Sigma, T, q_0, F)$, donde:

$Q = \{0, 1, 2, 3\}$;
$\Sigma = \{a, b\}$;
$q_0 = 0$;
$F = \{2, 3\}$;
$T = \{0 \Rightarrow a \Rightarrow 1, 1 \Rightarrow a \Rightarrow 2, 1 \Rightarrow b \Rightarrow 3, 3 \Rightarrow a \Rightarrow 3\}$.

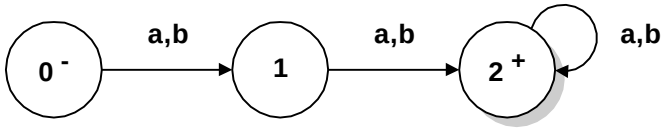
Si la función de transiciones T es representada mediante una TT, obtenemos el siguiente resultado:

T	a	b
0-	1	-
1	2	3
2+	-	-
3+	3	-

Esta tabla tiene una fila por cada estado del autómata y una columna por cada símbolo del alfabeto. Se lee así: desde el estado 0 (inicial), por a transita al estado 1 y por b no hay transición; desde el estado 1, por a transita al estado 2 y por b transita al estado 3; el estado 2 (final) no tiene

transiciones; y desde el estado 3 (final), por a tiene un ciclo y por b no hay transición, como se puede observar en el DT dibujado en el Ejemplo 1.

Ejemplo 3
Supongamos el lenguaje "Todas las palabras de aes y/o bes que tienen por lo menos dos letras". Las que siguen son las cuatro palabras de menor longitud: aa, bb, ab, ba. El diagrama de transiciones de un AFD que reconoce este lenguaje es:



En este diagrama, todas las transiciones tienen la curiosidad de estar etiquetadas con a,b. En realidad, cada etiqueta a,b significa dos transiciones diferentes: una por el carácter a y otra por el carácter b, ambas con el mismo estado de partida y el mismo estado de llegada. El uso de la etiqueta a,b simplifica el dibujo.

La definición formal de este AFD es $M = (Q, \Sigma, T, q_0, F)$, donde:
$Q = \{0, 1, 2\}$; $\Sigma = \{a, b\}$; $q_0 = 0$; $F = \{2\}$; T representada por la siguiente tabla de transiciones:

T	a	b
0-	1	1
1	2	2
2+	2	2

Una diferencia inmediata que se observa entre las tablas de los Ejemplos 2 y 3, es que la tabla del Ejemplo 2 tiene "huecos" (cuatro, para ser más precisos), que indican ausencia de transiciones, mientras que la tabla del Ejemplo 3 está completa.

* Ejercicio 1 *
Defina formalmente un AFD que reconozca el LR "Todas las palabras sobre el alfabeto {a,b,c} que tienen por lo menos tres letras".

1.1.2 AFD COMPLETO

➔ Un AFD es completo si cada estado tiene exactamente una transición por cada carácter del alfabeto.

Otra definición es:

➔ Un AFD es completo cuando su tabla de transiciones no tiene "huecos"; si los tiene, el AFD es *incompleto*.

La noción de AFD completo es muy importante en las aplicaciones computacionales de los AFDs, porque, como ya vimos en el capítulo 3 del Volumen 2 y como también veremos más adelante, un AFD se implementa mediante una matriz que representa a la TT y ésta, obviamente, no puede tener elementos sin información. En consecuencia, si el AFD constituye una herramienta que será implementada y no es completo, en primer lugar debe ser completado.

Completar un AFD es eliminar los "huecos" de su TT. Para ello:

1) Se agrega un nuevo estado, llamado estado de rechazo o estado de no aceptación;
2) Se reemplaza cada "hueco" por una transición a este nuevo estado; y
3) Se incorpora una nueva entrada en la tabla para el estado de rechazo en la que se representarán ciclos para todos los caracteres del alfabeto; con esto se informa que, una vez que el autómata se sitúa en el estado de rechazo, de él "no puede salir".

Ejemplo 4

Se desea completar el AFD del Ejemplo 2:

$M = (Q, \Sigma, T, q_0, F)$, donde: $Q = \{0, 1, 2, 3\}$; $\Sigma = \{a, b\}$; $q_0 = 0$; $F = \{2, 3\}$; y T representada por la TT:

T	a	b
0-	1	-
1	2	3
2+	-	-
3+	3	-

Para completar el AFD, agregamos un estado de rechazo (supongamos que este estado se llama 4). Entonces, el nuevo AFD es:

$M_2 = (Q, \Sigma, T, q_0, F)$, con $Q = \{0, 1, 2, 3, 4\}$; $\Sigma = \{a, b\}$; $q_0 = 0$; $F = \{2, 3\}$; y T representada por la tabla:

T	a	b
0-	1	4
1	2	3
2+	4	4
3+	3	4
4	4	4

Observe que la lectura de un carácter erróneo provoca que el AFD transite al estado de rechazo (el estado 4), del cual nunca podrá salir porque, como se distingue en la última fila, este estado tiene un ciclo a sí mismo por cualquier carácter del alfabeto.

Se ha obtenido, entonces, un nuevo AFD, con un nuevo conjunto de estados Q y una nueva función de transiciones T, que reconoce el mismo lenguaje que es aceptado por el Autómata Finito del cual se partió. Este hecho conduce a la siguiente definición:

➔ Dos AFDs son EQUIVALENTES si reconocen el mismo Lenguaje Regular.

Nota 1

Si bien, desde el punto de vista teórico, los AFDs no están obligados a ser completos, desde el punto de vista práctico es necesario trabajar con AFDs completos. Por lo tanto, de aquí en más adoptaremos la siguiente postura: cuando describimos un AFD formalmente mediante su TT, este autómata debe ser completo si ha sido diseñado dentro del ámbito de una aplicación computacional; en cambio, cuando dibujamos su DT, no agregamos el estado de rechazo para que el dibujo quede más "limpio" y así sea más sencilla su interpretación.

1.2 AUTÓMATA FINITO NO DETERMINÍSTICO

En la sección anterior se definió el tipo de autómata llamado Autómata Finito Determinístico (AFD), cuya característica esencial es que para cada estado del autómata existe cero (ninguna) o una transición por cada carácter del alfabeto. Y en el caso en que el AFD sea *completo*, entonces para cada estado existe exactamente una transición por cada símbolo del alfabeto.

En esta sección ofrecemos una modificación del modelo descripto en el párrafo anterior, permitiendo ahora que cualquier estado del autómata tenga cero (ninguna), una o más transiciones por el mismo carácter del alfabeto. Esta es la definición de un Autómata Finito No Determinístico (AFN), y se llama así porque habrá por lo menos un estado y un carácter para los que el autómata deberá elegir, entre dos o más transiciones, cuál es el camino a seguir.

Ejemplo 5
Sea el lenguaje "Todas las palabras de aes y bes que comienzan con a y terminan con b". Las tres palabras de menor longitud de este lenguaje son: ab, aab y abb. La Expresión Regular más simple que está asociada a este lenguaje es: a(a+b)*b.

Dibujemos el DT de un AF que reconozca a este lenguaje. Para ello, debemos tener en cuenta tres hechos:

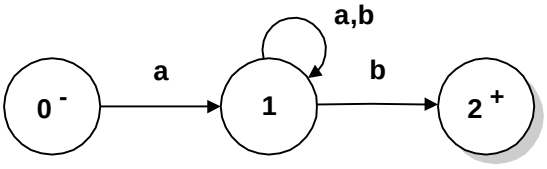
1) Como toda palabra del lenguaje comienza con a, debe existir una transición desde el estado inicial a un segundo estado etiquetada con el símbolo a;
2) Como toda palabra debe finalizar con b, entonces debe haber una transición desde un estado intermedio a un estado final por el carácter b; y
3) Después del carácter a inicial y antes del carácter b final, puede haber (no obligatoriamente) cualquier subcadena formada por aes y/o bes, que es lo que denota la expresión (a+b)* que forma parte de la ER de este lenguaje, y esto se representa por medio de un ciclo etiquetado con los caracteres a y b.

Resumiendo, el DT tendrá las siguientes transiciones:

0^- => a => 1, porque toda palabra debe comenzar con a
1 => a => 1 y 1 => b => 1, porque puede haber aes y/o bes entre la a inicial y la b final
1 => b => 2+, porque toda palabra debe finalizar con b.

Analizando las dos últimas líneas, observamos que existe una transición 1=>b=>1 y otra transición 1=>b=>2+. Son dos transiciones que parten del estado 1 y que, por lectura del carácter b, pueden "mover" al AFN a dos estados diferentes: al mismo estado 1 (un ciclo) o al estado 2. El autómata debe elegir una de estas dos transiciones para seguir trabajando, y esto caracteriza a un AFN.

El diagrama de transiciones tiene el siguiente digrafo:



Trabajemos con el AFN del ejemplo anterior. Supongamos que el AFN recibe la cadena abbb y debe determinar si es reconocida. Entonces, el AFN hará la siguiente tarea, intentando llegar al estado final lo antes posible:

0 => a => 1 => b => 2 (quedan dos caracteres sin procesar)
0 => a => 1 => b => 1 => b => 2 (queda un carácter sin procesar)
0 => a => 1 => b => 1 => b => 1 => b => 2 => RECONOCE

Segundo caso: el AFN recibe la cadena abba. Entonces, el AFN desarrolla el siguiente proceso:

0 => a => 1 => b => 2 (quedan dos caracteres sin procesar)
0 => a => 1 => b => 1 => b => 2 (queda un carácter sin procesar)
0 => a => 1 => b => 1 => b => 1 => a => ?? => RECHAZA

1.2.1 DEFINICIÓN FORMAL DE UN AFN

La definición formal de un AFN es similar a la definición formal de un AFD (una 5-upla con los mismos elementos), pero cambia la función de transiciones T.

En un AFD, esta función es del tipo T: Q x Σ –> Q, mientras que en un AFN, la función de transiciones es del tipo T: Q x Σ –> 2^Q, donde 2^Q es una notación que representa el conjunto de todos los conjuntos de estados, es decir: cada elemento de este "super conjunto" es un conjunto de estados.

Ejemplo 6
La definición formal del AFN cuyo DT fuera dibujado en el ejemplo anterior es:
M = (Q, Σ, T, q_0, F), donde Q = {0, 1, 2}; Σ = {a, b}; q_0 = 0; F = {2}; y la función T es representada por la TT:

T	a	b
0-	{1}	-
1	{1}	{1,2}
2+	-	-

Observe que ahora no podemos hablar de "estado de llegada" sino de "un conjunto de estados de llegada" (aunque el conjunto esté formado por un solo estado).

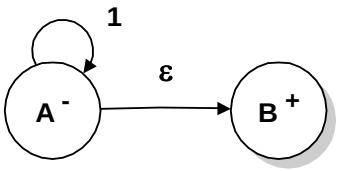
* Ejercicio 2 *
Defina formalmente un AFN que reconozca al LR representado por la ER (a+b)*a(a+b)*b.

1.2.2 AUTÓMATA FINITO CON TRANSICIONES-ε

El Autómata Finito con transiciones-ε es un segundo modelo de AFN y se caracteriza por la existencia de una o más transiciones que ocurren sin que el autómata lea el próximo carácter de la cadena que está analizando. Una transición-ε representa un cambio de estado "repentino", sin que intervenga ningún carácter del alfabeto.

La definición formal de un AF con transiciones-ε es igual a la de un AFN, como se la describió en la sección anterior, con la única diferencia que la función de transición es T: Q x (Σ ∪ {ε}) –> 2^Q.

Ejemplo 7
Sea el AFN con el siguiente Diagrama de Transiciones:



Este AFN tiene una transición-ε que "mueve" al autómata desde el estado A (inicial) al estado B (final), por lo que, sin leer un carácter, el autómata pasa del estado inicial al estado final. En consecuencia, este AFN reconoce a la palabra vacía.

Por otro lado, el ciclo A=>1=>A hace que este autómata reconozca 1*, que, concatenado con el ε de la transición que lo lleva al estado final, significa que este AFN reconoce al lenguaje representado por la ER: ε + 1*ε = ε + 1* = 1*.

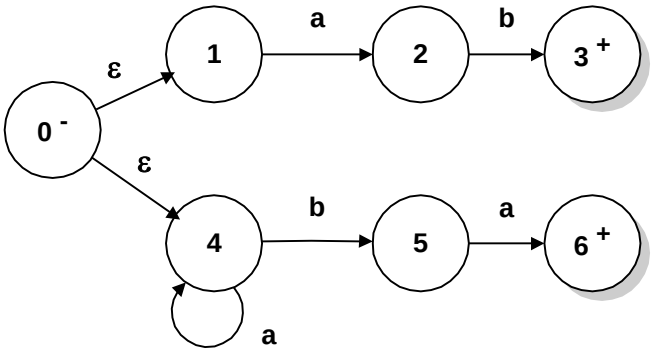
La definición formal de este AFN es:
M = (Q, Σ, T, q_0, F), donde Q = {A, B}; Σ = {1}; q_0 = A; F = {B}; y T representada por la tabla de transiciones:

T	1	ε
A-	{A}	{B}
B+	-	-

➔ La Tabla de Transiciones de un AFN con transiciones-ε debe tener una columna por cada carácter del alfabeto, más una columna por el símbolo ε.

Ejemplo 8
Sea el lenguaje representado por la siguiente ER: ab+a*ba, formada por la unión de dos ERs. El diseño de un AFD que reconozca a este lenguaje partiendo de la lectura de esta ER es complicado. Esto se debe a que la segunda expresión comienza con a* que, como ya hemos visto, significa "0 o más aes" y se representa mediante un ciclo, mientras que la primera expresión comienza, obligatoriamente, con exactamente una a. En cambio, no es dificultoso el diseño de dos AFDs, uno que reconozca a la ER ab y otro que acepte a la ER a*ba.

Si luego unimos estos dos AFs utilizando dos transiciones-ε, obtenemos un AFN con transiciones-ε que reconoce al LR dado. El Diagrama de Transiciones será:



La definición formal de este AFN es:
M = (Q, Σ, T, q_0, F), donde Q = {0, 1, 2, 3, 4, 5, 6}; Σ = {a, b}; q_0 = 0; F = {3, 6}; y T representada por la Tabla de Transiciones:

T	a	b	ε
0-	-	-	{1,4}
1	{2}	-	-
2	-	{3}	-
3+	-	-	-
4	{4}	{5}	-
5	{6}	-	-
6+	-	-	-

CONCLUSIÓN: se han presentado tres modelos de Autómatas Finitos, uno determinístico y dos no determinísticos. Aunque los AFNs son más flexibles que los AFDs, la capacidad de reconocimiento de LRs es la misma para los tres modelos. En un próximo capítulo se verá un algoritmo para convertir un AFN (con o sin transiciones-ε) en un AFD que reconoce el mismo lenguaje, con lo que se establece la equivalencia entre los diferentes modelos de AFs.

* Ejercicio 3 *
Defina formalmente un AFN con transiciones-ε que reconozca al LR representado por la siguiente ER: ab*+a*b+aa*.

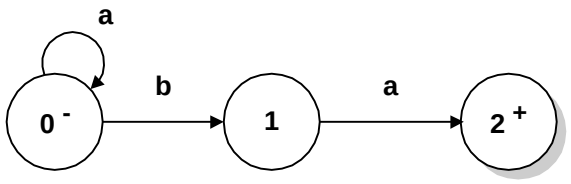
* Ejercicio 4 *
Defina formalmente un AFN con transiciones-**ε** que reconozca al LR representado por la siguiente ER: 0*11+001*+01*00.

* Ejercicio 5 *
a) Sea el lenguaje “Todos los números binarios que contienen una secuencia de exactamente dos ceros por lo menos una vez”. Dibuje el DT de un AFN o de un AFD que lo reconozca.
b) Escriba la descripción formal del AFN o del AFD diseñado.

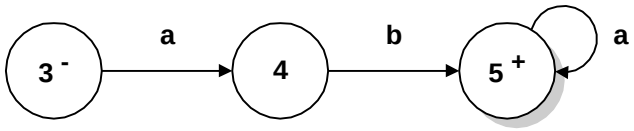
2 DE LA EXPRESIÓN REGULAR AL AUTÓMATA FINITO

En el capítulo anterior se han construido diversos AFDs y AFNs que reconocen a distintos LRs. Pero no siempre resulta tan sencillo ni tan seguro diseñar un Autómata Finito correcto; no existe un algoritmo para verificarlo.

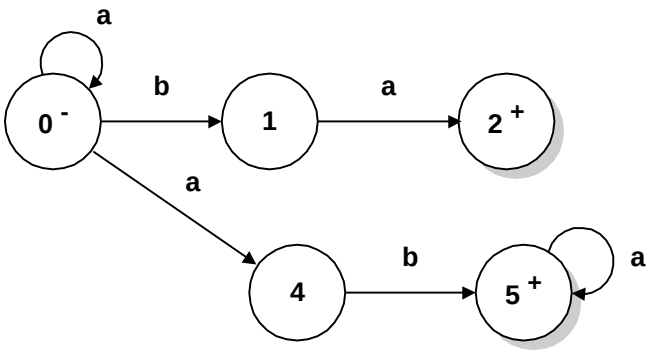
Ejemplo 1
Supongamos el lenguaje representado por la siguiente ER: a*ba+aba*. Comencemos a dibujar el DT de un AF que reconoce a este lenguaje. La ER dada está formada por la *unión* de dos expresiones, cada una de las cuales es reconocida por un AF cuya construcción individual es bastante simple.
La primera ER, a*ba, es reconocida por un AF que se puede construir, con cierta facilidad, siguiendo las consignas expuestas en el capítulo anterior:



La segunda expresión, aba*, también deriva en un AF de fácil construcción DT:



Ya tenemos a los dos AFs por separado; solo nos resta unirlos para obtener un AF que reconozca a la ER presentada inicialmente. Intentemos, entonces, la siguiente acción: consideremos al estado 0 como único estado inicial del AF final. Consecuentemente, eliminamos al estado 3 y dibujamos el siguiente DT:



Pero este AF no es correcto porque la parte del AF dibujada "hacia abajo" reconoce más de un carácter a inicial. Esto es adecuado para la ER de la izquierda (a*ba), pero no lo es para la ER de la derecha (aba*). En realidad, hemos construido un AF que reconoce al LR representado por a*(ba+aba*), ya que la expresión a*, reconocida por el estado 0 –el estado inicial–, es común a ambas "ramas" del digrafo. Observe, además, que este autómata es un AFN por las transiciones 0=>a=>0 y 0=>a=>4. La solución a este problema será lograda en las próximas secciones de este capítulo, mediante el uso del algoritmo que describiremos.

El proceso que conduce desde el conocimiento de un LR hasta la obtención del AFD más útil para trabajar con ese lenguaje, tiene varias etapas. En primer lugar, el LR es descripto, generalmente, mediante una frase en un lenguaje natural o mediante una ER. Si el LR es descripto mediante una frase, el camino que lleva a la construcción del AFD que lo reconoce puede ser muy recto pero también puede ser extremadamente sinuoso, dada la ambigüedad del lenguaje natural. No existe algoritmo alguno para realizar esta tarea, solo es cuestión de poseer buenos conocimientos previos y una adecuada dosis de inspiración para aplicar, en el momento oportuno, esos conocimientos; aunque nunca se podrá garantizar un resultado correcto.

En cambio, si el LR está representado por una ER, la situación se simplifica mucho porque existe un grupo de algoritmos que, aplicados ordenadamente, nos permitirán obtener el mejor AFD con absoluta seguridad. La sucesión de algoritmos es:

ER → Algoritmo de Thompson → AFN → Algoritmo de Clausura-ε → AFD → Algoritmo de Clases → AFD Mínimo

En este capítulo se describirá el ALGORITMO DE THOMPSON y también la utilización de sus propiedades para construir un AF en forma más rápida y menos compleja, pero no algorítmica. En los capítulos que siguen se cubren los otros dos algoritmos de esta trilogía.

2.1 ALGORITMO DE THOMPSON

La palabra *algoritmo* significa que el método que se describirá puede ser implementado mediante un programa de computadora, y es aquí donde radica su verdadera utilidad.

El Algoritmo de Thompson, desarrollado por Ken Thompson, consiste en:

1) Desmembrar la ER de partida en sus componentes básicos, es decir: caracteres, operadores y ε, si este símbolo forma parte de la expresión;

2) Generar un AF básico por cada carácter o símbolo ε que forme parte de la ER;

3) Componer estos autómatas básicos según los operadores existentes en la ER, hasta lograr el AF que reconoce a la ER dada. El autómata que se obtiene es un AFN con transiciones-ε. Cabe aclarar que el Algoritmo de Thompson trabaja ÚNICAMENTE con los tres operadores oficiales de las ERs (unión, concatenación y clausura de Kleene).

2.1.1 AUTÓMATA PARA CADA CARÁCTER Y PARA EL SÍMBOLO ε

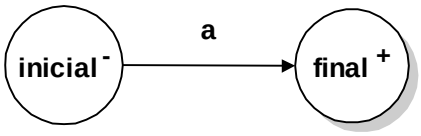
Como se describió anteriormente, en primer lugar el Algoritmo de Thompson construye tantos *autómatas básicos* como caracteres y símbolos ε haya en la Expresión Regular.

Ejemplo 2
a) Si la ER es a+b, el Algoritmo de Thompson comienza construyendo dos autómatas: uno que reconozca al carácter a y otro que reconozca al carácter b.

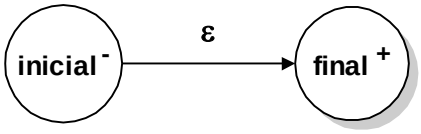
b) Si la ER es a+ε, el Algoritmo de Thompson comienza construyendo dos autómatas: uno que reconozca al carácter a y otro que reconozca al símbolo ε.

c) Si la ER es a+ba, el Algoritmo de Thompson comienza con la construcción de tres autómatas: uno que reconozca a la primera a, otro que reconozca al carácter b y un tercer autómata que reconozca a la segunda a.

El AF genérico que reconoce un símbolo del alfabeto (a, por caso) es:



El AF genérico que reconoce al símbolo ε es:



Una vez que están desarrollados los AFs que reconocen a los diversos caracteres y símbolos ε que forman la ER original, el Algoritmo de Thompson procede a COMPONER estos autómatas básicos, según las operaciones que están presentes en la ER. Surgen, así, autómatas más complejos, con mayor cantidad de estados, que corresponden a la aplicación de los distintos operadores que forman la ER, respetando, obviamente, las prioridades de estos operadores. A continuación se muestra cómo se crean los autómatas para los diferentes operadores "oficiales". Recuerde que el Algoritmo de Thompson solo trabaja con estos operadores.

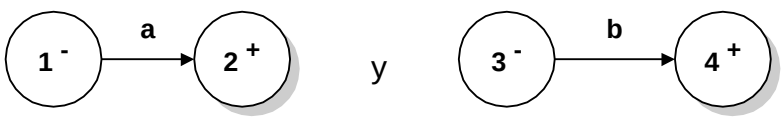
2.1.2 AUTÓMATA PARA LA UNIÓN

Se diseñará el AF que reconoce a la ER formada por la *unión* de dos caracteres o símbolo ε (como a+b o a+ε), de la siguiente forma:

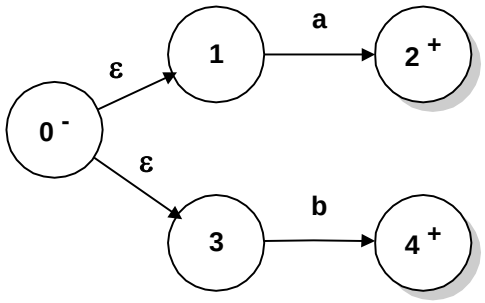
1) Se construyen los dos autómatas básicos;
2) Los estados iniciales de los autómatas básicos dejan de ser iniciales y los estados finales de estos mismos autómatas dejan de ser finales;
3) Se agrega un nuevo estado inicial;
4) Se incorporan dos transiciones-ε que relacionarán al nuevo estado inicial con los dos ex estados iniciales;
5) Se añade un nuevo estado final;
6) Se trazan dos transiciones-ε para unir a los dos ex estados finales con este nuevo estado final, y el autómata está construido.

Ejemplo 3
Sea la ER a+b. El proceso del Algoritmo de Thompson para obtener el AF reconocedor, siguiendo los seis pasos descriptos anteriormente, es el siguiente:

(1)

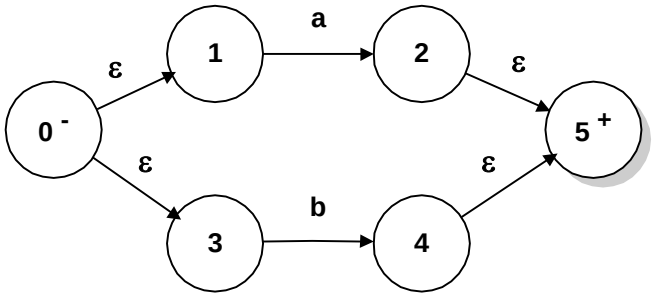


(2)



El AFN obtenido en este punto ya reconoce a la ER a+b pero no responde exactamente a las restricciones del Algoritmo de Thompson, porque este método exige que el autómata construido tenga un ÚNICO estado final. Por lo tanto, debemos seguir un paso más.

(3)



Este es el AFN que se obtiene aplicando el Algoritmo de Thompson; el autómata es un AFN por tener transiciones-ε.

* Ejercicio 1 *
Dibuje el DT del AF que se obtiene aplicando Thompson para reconocer la ER b+ε.

* Ejercicio 2 *
Dibuje el DT del AF que se obtiene aplicando Thompson para reconocer la ER b+c+a.
Ayuda: Trate a esta ER como si fuera: (b+c)+a.

2.1.3 AUTÓMATA PARA LA CONCATENACIÓN

Sea una ER formada por la concatenación de dos caracteres (como, por ejemplo, ab). El AFN "por Thompson" que reconoce a esta ER se obtiene de la siguiente manera:

1) Se construyen los autómatas básicos (denominemos *autómata izquierdo* al que reconoce el carácter "izquierdo" y *autómata derecho* al que acepta el carácter "derecho" de esta concatenación);

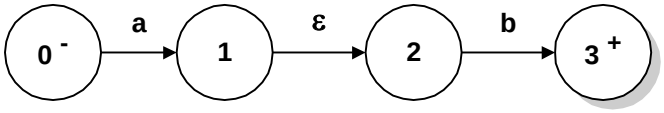
2) El estado final del *autómata izquierdo* deja de ser final y el estado inicial del *autómata derecho* deja de ser inicial (los otros estados no se alteran);

3) Se agrega una transición-ε que vincule al ex estado final del *autómata izquierdo* con el ex estado inicial del *autómata derecho*;

4) El AFN ya está terminado, siendo su estado inicial el estado inicial del *autómata izquierdo*, y siendo su estado final el estado final del *autómata derecho*.

Ejemplo 4
Sea la Expresión Regular ab. Para obtener el AFN "por Thompson" que reconoce a esta expresión, partimos de los dos autómatas básicos:

Siguiendo el algoritmo descripto, obtenemos el siguiente autómata:



* Ejercicio 3 *
Dibuje el DT del AF por Thompson que reconoce la ER bb.

* Ejercicio 4 *
Dibuje el DT del AF por Thompson que reconoce la ER baa.

2.1.4 AUTÓMATA PARA LA CLAUSURA DE KLEENE

Deseamos construir, mediante el Algoritmo de Thompson, un AFN que reconozca a una ER que solo tiene el operador "estrella" (como, por ejemplo, a*). Este es el tercero y último operador "oficial" que existe para la creación de ERs. Recuerde que los otros operadores que hemos agregado generan ERs que siempre se pueden escribir, aunque con mayor dificultad, utilizando solo los tres operadores "oficiales".

El autómata para la clausura de Kleene se obtiene así:

1) Se construye el autómata básico;

2) El estado inicial deja de ser inicial y el estado final deja de ser final;

3) Se incorporan un nuevo estado inicial y un nuevo estado final;

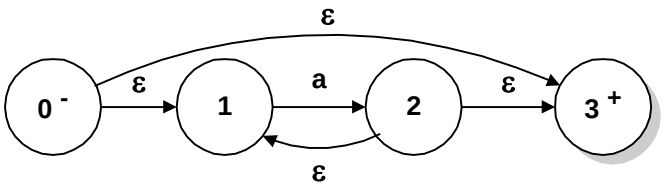
4) Se agrega una transición-ε desde el nuevo estado inicial hasta el ex estado inicial;

5) Se agrega una transición-ε desde el ex estado final al nuevo estado final;

6) Se incorpora una transición-ε desde el nuevo estado inicial al nuevo estado final (para que reconozca la palabra vacía);

7) Se agrega una transición-ε desde el ex estado final al ex estado inicial (para reconocer la repetición del carácter).

Ejemplo 5
Siguiendo los siete pasos descriptos arriba, y partiendo de un autómata básico con transición 1=>a=>2, el AFN "por Thompson" que reconoce a la Expresión Regular a* es:



* Ejercicio 5 *
Dibuje el DT del AF por Thompson que reconoce la ER b*.

* Ejercicio 6 *
Dibuje el DT del AF por Thompson que reconoce la ER (ba)*+a.

2.1.5 CARACTERÍSTICAS GENERALES DE UN AFN "POR THOMPSON"

El AFN obtenido mediante la aplicación del Algoritmo de Thompson es un AFN con transiciones-ε con las siguientes características fundamentales y taxativas:

1) Al estado inicial no llegan transiciones;

2) El estado final debe ser único;

3) Del estado final no parten transiciones;

4) De cualquier estado no final pueden partir: una sola transición etiquetada con un carácter del alfabeto, una sola transición-ε o dos transiciones-ε.

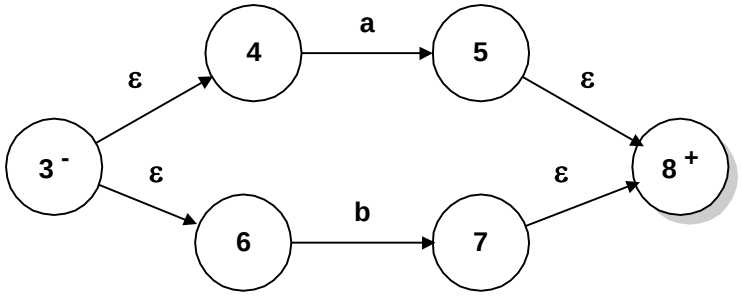
Ejemplo 6
Sea la ER a(a+b)*; esta es una expresión que posee los tres operadores oficiales. Construyamos, aplicando el Algoritmo de Thompson, un AF que reconozca a este LR.

En primer lugar, debemos tener en cuenta las prioridades de los operadores que actúan en esta expresión y la existencia de paréntesis. En consecuencia, el orden de evaluación será.

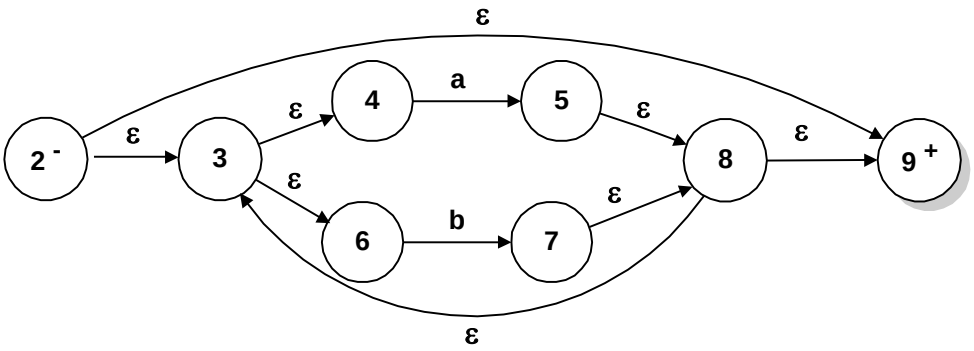
1) (a+b);
2) la clausura de Kleene de (a+b);
3) la concatenación de a con (a+b)*.

El orden a aplicar en la construcción del AF por Thompson debe coincidir con el orden de evaluación de la expresión. Por lo tanto:

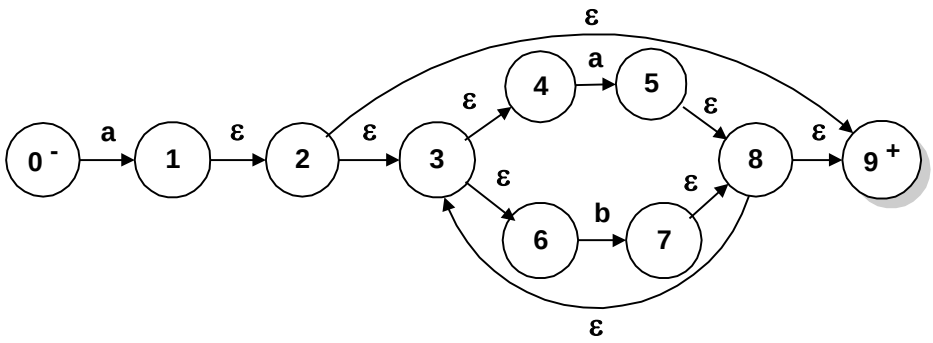
1) Se construyen los autómatas básicos para los caracteres a y b que intervienen en la expresión (a+b);
2) Se construye el autómata que reconoce (a+b):



3) Se construye el autómata que reconoce (a+b)*:



4) Se construye el autómata básico que reconoce el primer carácter a de la ER total;
5) Finalmente, se construye el autómata que reconoce a la Expresión Regular a(a+b)*:



* Ejercicio 7 *
Verifique que el AF recién construido cumple con todas las restricciones indicadas anteriormente (único estado final, etc.).

> *Nota 1*
> El *lex* reconoce LRs a través de su representación mediante ERs. Allen Holub, en su libro "Compiler Design in C" (1990), desarrolla su propia versión de LEX y muestra cómo lo implementa utilizando el Algoritmo de Thompson para reconocer las diferentes ERs. Luego lo transforma en un AFD, como veremos en el próximo capítulo.

A continuación incorporamos otra sección en la que se analizará un método, basado en algunas de las pautas que nos brinda el Algoritmo de Thompson, pero que resulta más simple para obtener un AF diseñado manualmente (no algorítmicamente).

2.2 UTILIZACIÓN DE THOMPSON EN RESOLUCIÓN NO ALGORÍTMICA

El Algoritmo de Thompson es muy útil por ser, justamente, un algoritmo. Esta circunstancia posibilita que se pueda diseñar un programa de computadora que implemente ese algoritmo de tal forma que dada una ER (como dato), obtenga la Tabla de Transiciones del AFN "por Thompson" (como resultado).

Como se ha visto en los ejemplos presentados, el AFN "por Thompson" tiene muchos estados y su construcción manual es lenta y muy tediosa. La computadora, en cambio, es muy rápida y no tiene

sentimientos como aburrimiento o fastidio; por lo tanto, dejemos que el Algoritmo de Thompson sea aplicado por la computadora.

No obstante, debemos conocer el Algoritmo de Thompson para saber, justamente, qué debemos tener en cuenta y qué podemos modificar en una aplicación manual, para que el AF obtenido sea correcto y más simple de procesar. En esta sección analizaremos un método que resulta más práctico para ser aplicado en forma manual (*papel y lápiz*), aunque debemos tener mucho más cuidado para obtener un AF que sea, efectivamente, correcto. El AF que se obtiene puede ser un AFN o un AFD.

Una característica fundamental de este método es que, por no ser un algoritmo implementable, el AF obtenido no debe respetar las restricciones que poseen los autómatas construidos mediante el Algoritmo de Thompson. En consecuencia:

1) al estado inicial pueden llegar transiciones;

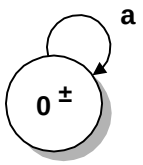
2) puede haber varios estados finales;

3) del estado final pueden salir transiciones;

4) desde cualquier estado puede partir un número y tipo de transiciones arbitrario.

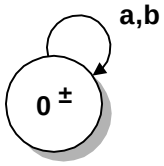
Si bien no se puede dar una "receta" aplicable a todos los casos, sí se pueden analizar algunas situaciones que se presentan a menudo o que forman una generalización de lo que realiza el Algoritmo de Thompson:

Caso 1: Expresión Regular a*. El AF que reconoce a esta expresión es inmediato:



Si construimos el AF que reconoce a este lenguaje utilizando el Algoritmo de Thompson, obtenemos un AFN con transiciones-ε y con cuatro estados. El autómata dibujado, en cambio, es un AFD y tiene un solo estado. Observe que este autómata no respeta el Algoritmo de Thompson por dos motivos: al estado inicial llega una transición y, además, del estado final sale una transición. Por otro lado, el Algoritmo de Thompson jamás puede producir un autómata en el que su estado inicial, también sea estado final.

Caso 2: Expresión Regular (a+b)*. El AF que reconoce a esta expresión es igual que el autómata anterior, con la única diferencia que ahora la transición está etiquetada con dos caracteres (en lugar de dibujar dos transiciones, cada una etiquetada con un carácter):

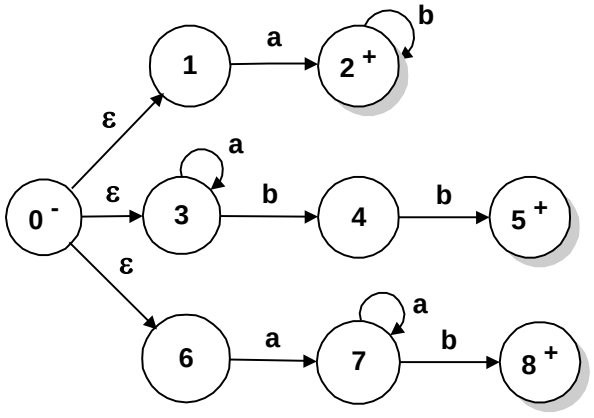


En este caso, la diferencia entre el AFN por Thompson y este AFD es aún mayor en cuanto a la cantidad de estados de cada autómata. El dibujado es un AFD y sigue teniendo un solo estado, mientras que el AFN por Thompson tiene ocho estados. Este AFD agrega una nueva violación a las

restricciones que define el Algoritmo de Thompson: desde el estado inicial parten dos transiciones por dos caracteres del alfabeto (aunque esté dibujado con un solo ciclo, para simplificar el gráfico).

* Ejercicio 8 *
Dibuje el DT del AF por Thompson que reconoce a esta misma ER. Compare las resoluciones.

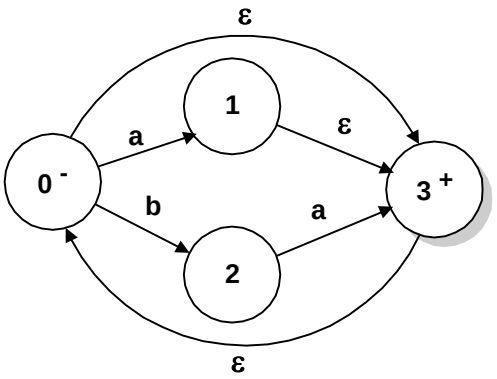
Caso 3: Expresión Regular $ab^*+a^*bb+a^+b$, constituida por la unión de tres expresiones, cada una de las cuales tiene concatenaciones. Generalizamos la implementación del operador unión del Algoritmo de Thompson, permitiendo que desde el estado inicial partan más de dos transiciones-ε. Además, ignoramos la necesidad de un único estado final y eliminamos la transición-ε que agrega Thompson para representar a cada concatenación. El DT resultante es:



Obviamente, esta generalización de "varias transiciones-ε" se puede aplicar a cualquier estado del autómata. Note, además, cómo reconoce a la "clausura positiva" ($a^+ = aa^*$, en los estados 6 y 7).

* Ejercicio 9 *
Dibuje el DT del AF por Thompson que reconoce a esta misma ER. Compare las dos resoluciones.

Caso 4: Expresión Regular $(a+ba)^*$. Seguramente nos conviene utilizar transiciones-ε para crear el ciclo que corresponde al operador clausura de Kleene, pero recuerde que no estamos obligados a agregar nuevos estados (como sí ocurre con el Algoritmo de Thompson). Por lo tanto, podemos construir el siguiente AFN reconocedor:



Como se observa, hay varias violaciones de las restricciones propias del Algoritmo de Thompson:

1) al estado inicial llega una transición;

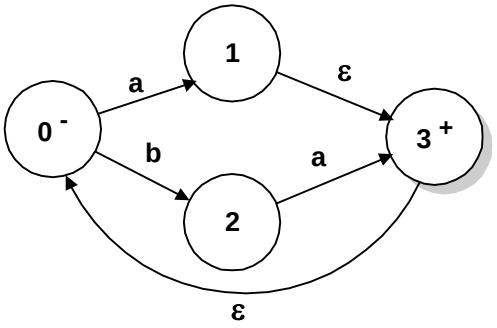
2) del estado inicial salen dos transiciones etiquetadas con caracteres del alfabeto, ya que no es necesario el uso de transiciones-ε para esta unión, más una transición etiquetada con ε;

3) del estado final sale una transición; y, finalmente,

4) no es necesario agregar una transición-ε para concatenar los caracteres b y a.

* Ejercicio 10 *
Dibuje el DT del AF por Thompson que reconoce a esta misma ER. Compare las dos resoluciones.

Caso 5: Expresión Regular $(a+ba)^+$. Este caso es similar al caso anterior: la única diferencia es que utiliza el operador clausura positiva en lugar del operador clausura de Kleene. Como sabemos, la clausura positiva no representa a la palabra vacía si ésta no pertenece al lenguaje base. En consecuencia, el AF resultante es igual al AF anterior, pero sin la transición-ε que transita del estado inicial al estado final para reconocer a la palabra vacía:



* Ejercicio 11 *
Dibuje el DT del AF por Thompson que reconoce a esta misma ER. Compare las dos resoluciones.

3 DEL AFN AL AFD

Un Autómata Finito, obtenido a partir de la descripción de un LR mediante una frase o desde una ER, puede ser un AFD o un AFN. Y si el Autómata Finito se construye utilizando el Algoritmo de Thompson, será un AFN con transiciones-ε.

Sabemos que un autómata es un AFN porque: (1) hay algún estado del que parten dos o más transiciones etiquetadas con un mismo carácter del alfabeto, o (2) tiene transiciones-ε, o (3) ambas cosas. Cualquiera sea la situación, habrá dificultades para implementar al AFN mediante un programa de computadora.

Afortunadamente, para todo AFN existe un AFD que reconoce al mismo lenguaje y que puede ser implementado en forma más sencilla. En otras palabras, para todo AFN existe un AFD equivalente. En este capítulo se describe un algoritmo que, dado un AFN, obtiene un AFD equivalente.

➔ Para todo AFN existe un AFD equivalente.

3.1 DEFINICIONES PRELIMINARES

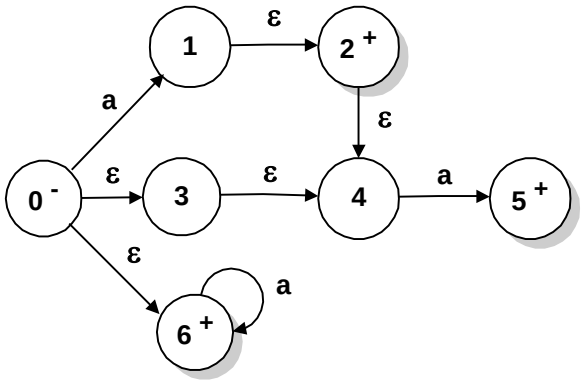
El algoritmo para construir un AFD a partir de un AFN se basa en el conocimiento de dos tipos de conjuntos: el conjunto Clausura-ε y el conjunto Hacia. Ambos tipos de conjuntos son definidos a continuación.

3.1.1 CLAUSURA-ε DE UN ESTADO

Sea q un estado de un AFN; entonces, la Clausura-ε (q) es el conjunto de estados formado por q y todos aquellos estados a los cuales se llega, desde q, utilizando solo transiciones-ε.

En consecuencia, el conjunto Clausura-ε de un estado nunca puede ser vacío porque contiene, como mínimo, a su propio estado, es decir: si un estado no tiene transiciones-ε, la clausura-ε de ese estado solo contiene al mismo estado.

Ejemplo 1
Sea el siguiente diagrama de transiciones:



La descripción formal de este autómata posee la siguiente Tabla de Transiciones:

TT	a	ε
0-	{1}	{3,6}
1	-	{2}
2+	-	{4}
3	-	{4}
4	{5}	-
5+	-	-
6+	{6}	-

Esta TT nos muestra, implícitamente: cuáles son los estados del autómata, cuál es el estado inicial (señalado con –), cuáles son los estados finales (señalados con +), qué caracteres integran el alfabeto (solo el carácter a en esta oportunidad, ya que el símbolo ε no pertenece a ningún alfabeto) y cuáles son las transiciones (cuyos estados de llegada son los elementos de esta matriz). En conclusión, esta TT tiene toda la información del Autómata Finito y es una forma más compacta de describirlo formalmente; por ello, la emplearemos a menudo.

Determinamos la clausura-ε de cada estado, utilizando la información que brinda la TT en su columna etiquetada con ε. Comenzamos con aquellos estados que no tienen transiciones-ε (caso más simple) y concluimos con el estado que tiene más transiciones-ε (caso más complejo). Los estados 4, 5 y 6 no tienen transiciones-ε, por lo que sus respectivas clausuras-ε serán:

clausura-ε (4) = {4}, clausura-ε (5) = {5}, clausura-ε (6) = {6}.

El estado 1 tiene una transición-ε al estado 2, quien a su vez tiene una transición-ε al estado 4. En consecuencia, desde el estado 1 se puede llegar a los estados 2 y 4 utilizando solo transiciones-ε, por lo que clausura-ε (1) = {1,2,4}.

El estado 2 fue analizado al determinar qué sucedía con el estado 1. En consecuencia, ya hemos visto que la clausura-ε (2) = {2,4}.

El estado 3 transita por ε al estado 4 y éste no tiene transiciones-ε. Luego, clausura-ε (3) = {3,4}.

Finalmente, el estado 0 transita por ε a los estados 3 y 6, que ya fueron analizados. En consecuencia, la clausura-ε del estado 0 es clausura-ε (0) = {0,3,4,6}.

* Ejercicio 1 *

Sea la siguiente Tabla de Transiciones de un AFN:

TT	a	ε
0-	{1}	{2,3,6}
1	-	{2}
2+	-	{3}
3	-	{6}
4	{5}	{5}
5+	-	-
6+	{6}	{1}

Determine la clausura-ε de cada estado.

* Ejercicio 2 *

¿Qué característica tienen las clausuras-ε de los estados de un AFD?

3.1.2 CLAUSURA-ε DE UN CONJUNTO DE ESTADOS

Sea R un conjunto de estados. Entonces, la clausura-ε(R) es la unión de las clausuras-ε de los estados que componen el conjunto R.

Ejemplo 2
Retomando el AFN del Ejemplo 1, supongamos un conjunto de estados R = {1,3,6}. Entonces, su clausura-ε será: clausura-ε(R) = clausura-ε(1) ∪ clausura-ε(3) ∪ clausura-ε(6) = {1,2,4} ∪ {3,4} ∪ {6} = {1,2,3,4,6}.

* Ejercicio 3 *
Sea la TT del Ejemplo 1 y sea el conjunto de estados M = {0,1}. Obtenga la clausura-ε(M).

3.1.3 EL CONJUNTO "HACIA"

Sea R un conjunto de estados y sea x un símbolo del alfabeto. Entonces, hacia(R,x) es el conjunto de estados a los cuales se transita por el símbolo x, desde los estados de R que tengan esa transición. En otras palabras, lo podemos definir como el conjunto de estados de llegada para (R,x).

Ejemplo 3
Supongamos que el conjunto R está formado por los estados 1, 2 y 3, y supongamos que existen las siguientes transiciones para estos estados: 1=>a=>4, 2=>a=>5 y 3=>b=>6. Entonces:

hacia(R,a) = {4, 5} y hacia(R,b) = {6}.

* Ejercicio 4 *
Sea el conjunto M formado por los estados 1, 2 y 3. Supongamos que estos estados tienen las siguientes transiciones: 1=>a=>1, 2=>a=>1, 3=>b=>2, 2=>b=>4. Obtenga:

hacia(M,a), hacia(M,b) y hacia(M,c).

3.2 EL ALGORITMO

La construcción de un AFD a partir de un AFN se realiza mediante un algoritmo que recibe el nombre de Algoritmo de Clausuras-ε o Construcción de Subconjuntos. Este último nombre se debe a que cada estado del AFD estará formado por un conjunto de estados del AFN, cuyos componentes deben reflejar, todos, la misma transición ante un símbolo dado del alfabeto. En otras palabras: el AFD simulará, "en paralelo", todas las transiciones que realiza el AFN del cual se parte, para cualquier cadena de entrada que el autómata deba procesar.

Para obtener el AFD, necesitamos determinar tres cosas:

1) el *estado inicial* del AFD;
2) sus *estados finales*;
3) su *Tabla de Transiciones*.

Este algoritmo comienza con la obtención del estado inicial del AFD. A partir de este estado, determinará los otros estados del AFD y, con ellos, construirá la TT del nuevo autómata y también resolverá cuáles estados son finales.

Por definición, el estado inicial del AFD es la clausura-ε del estado inicial del AFN dato, mientras que los estados finales del AFD son todos aquellos conjuntos de estados del AFN que contienen, por lo menos, un estado final.

Finalmente, la Tabla de Transiciones es construida de la siguiente manera:

1) Se obtiene el estado inicial del AFD, que, como ya se ha adelantado, es la clausura-ε del estado inicial del AFN;

2) Se agrega este estado a la primera columna de la tabla;

3) Para cada símbolo del alfabeto, se calcula el conjunto hacia del estado que se acaba de agregar a la primera columna de la tabla;

4) Se determinan nuevos estados del AFD por medio de la clausura-ε de cada conjunto hacia recién obtenido. Estos estados (conjuntos) se incorporan a la tabla, en las columnas que les corresponden;

5) Si un nuevo estado obtenido en el punto anterior no existe todavía en la primera columna de la Tabla de Transiciones, se lo agrega;

6) Se repiten los pasos (3) al (5) hasta que no surjan nuevos estados.

Ejemplo 4
Sea el lenguaje representado por la ER (a+b)*ab. Aplicando, primero, el Algoritmo de Thompson obtendremos un AFN que reconoce a este lenguaje. Luego, utilizaremos la Construcción de Subconjuntos para hallar un AFD equivalente. La Tabla de Transiciones del AFN hallado "por Thompson" es:

TT	a	b	ε
0-	-	-	{1,7}
1	-	-	{2,3}
2	{4}	-	-
3	-	{5}	-
4	-	-	{6}
5	-	-	{6}
6	-	-	{1,7}
7	-	-	{8}
8	{9}	-	-
9	-	-	{10}
10	-	{11}	-
11+	-	-	-

* Ejercicio 5 *
Verifique que esta TT es correcta.

A partir de esta TT del AFN, construimos la TT de un AFD equivalente por medio de la Construcción de Subconjuntos, de esta manera:

1) Calculamos el estado inicial del AFD, que es la clausura-ε del estado inicial del AFN, revisando la columna etiquetada con ε en la TT del AFN: clausura-ε(0) = {0,1,2,3,7,8}

2) Colocamos este estado-conjunto como primer estado de la TT del AFD y le agregamos el símbolo – para indicar que éste es el estado inicial del AFD que estamos construyendo:

estado	a	b
{0,1,2,3,7,8}-	?	?

(la columna ε desaparece porque estamos construyendo un AFD)

3) Calculamos el conjunto hacia de este estado inicial, para cada uno de los caracteres del alfabeto: a y b. Para el carácter a, en la tabla original existen las transiciones 2=>a=>4 y 8=>a=>9. Como los estados 2 y 8 forman parte del estado inicial del AFD, entonces:

hacia ({0,1,2,3,7,8}, a) = {4,9}.

De la misma forma, para el carácter b existen las transiciones 3=>b=>5 y 10=>b=>11. Pero solo el estado 3 pertenece al conjunto del estado inicial del AFD, por lo que:

hacia ({0,1,2,3,7,8}, b) = {5}.

4) Ahora calculamos la clausura-ε de cada conjunto *hacia* hallado en el punto anterior, y estos serán los estados de llegada, de las respectivas transiciones, desde el estado inicial del AFD:

- clausura-ε(hacia({0,1,2,3,7,8},a)) = clausura-ε({4,9}) = clausura-ε(4) ∪ clausura-ε(9) = {4,6,1,7,2,3,8} ∪ {9,10} = {1,2,3,4,6,7,8,9,10} = {1-4,6-10} y
- clausura-ε(hacia({0,1,2,3,7,8}, b)) = clausura-ε({5}) = {5,6,1,7,2,3,8} = {1,2,3,5,6,7,8} = {1-3,5-8}.

5) Completamos la fila del estado inicial en la TT, con los estados de llegada obtenidos en el paso anterior:

estado	a	b
{0-3,7,8}-	{1-4,6-10}	{1-3,5-8}

6) Han aparecido dos estados que no figuraban en la primera columna de la TT; por lo tanto, debemos agregarlos, formando dos nuevas filas:

estado	a	b
{0-3,7,8}-	{1-4,6-10}	{1-3,5-8}
{1-4,6-10}	?	?
{1-3,5-8}	?	?

Procedemos como hemos hecho en los puntos (3) y (4), de esta manera:

- hacia({1-4,6-10}, a) = {4,9}; clausura-ε({4}) ∪ clausura-ε({9}) = {4,6,1,7,2,3,8} ∪ {9,10} = {1-4,6-10}
- hacia({1-4,6-10}, b) = {5,11}; clausura-ε({5}) ∪ clausura-ε({11}) = {1-3,5-8} ∪ {11} = {1-3,5-8,11}

y

- hacia({1-3,5-8}, a) = {4,9}; clausura-ε({4}) ∪ clausura-ε({9}) = {4,6,1,7,2,3,8} ∪ {9,10} = {1-4,6-10}
- hacia({1-3,5-8}, b) = {5}; clausura-ε({5}) = {5,6,1,7,2,3,8} = {1-3,5-8}

Colocamos los estados resultantes en la TT para completar las tres primeras filas y obtenemos:

estado	a	b
{0-3,7,8}-	{1-4,6-10}	{1-3,5-8}
{1-4,6-10}	{1-4,6-10}	{1-3,5-8,11}
{1-3,5-8}	{1-4,6-10}	{1-3,5-8}

El único estado nuevo que ha surgido es el estado {1-3,5-8,11}. Este estado es final porque contiene al estado 11, estado final del AFN del cual partimos.

* Ejercicio 6 *
Complete el proceso para hallar la TT final del AFD.

Dado que la notación de los estados es difícil de leer, los renombramos de la siguiente manera:

nombre anterior	nombre nuevo
{0-3,7,8}-	0-
{1-4,6-10}	1
{1-3,5-8}	2
{1-3,5-8,11}+	3+

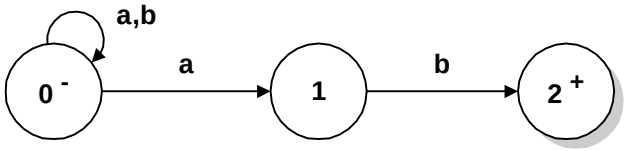
En consecuencia, la Tabla de Transiciones final del AFD que se halla mediante el algoritmo descripto es:

TT	a	b
0-	1	2
1	1	3
2	1	2
3+	1	2

* Ejercicio 7 *
Verifique que esta TT es correcta.

En el próximo ejemplo volvemos a analizar la conversión AFN -> AFD, pero esta vez partiendo de un AFN sin transiciones-ε.

Ejemplo 5
Sea, nuevamente, el lenguaje representado por la misma ER del ejemplo anterior: (a+b)*ab. Necesitamos hallar un AFD que reconozca este lenguaje, proceso que no es trivial a partir de esta expresión. Supongamos que estamos resolviendo el problema a partir de algunos conocimientos que ya tenemos incorporados para interpretar los operadores y los términos de la ER; por lo tanto, comenzamos el proceso con el siguiente AFN:



Este autómata es un AFN porque desde el estado 0 parten dos transiciones diferentes por el carácter a: una forma un ciclo, 0=>a=>0, y la otra transita al estado 1,0=>a=>1. La TT de este AFN es:

TT	a	b
0-	{0,1}	{0}
1	-	{2}
2+	-	-

Como no existen transiciones-ε, la clausura-ε de cada estado es el conjunto que solo contiene al mismo estado, por lo que solo se debe calcular, donde corresponda, el conjunto hacia.

En consecuencia, a partir de la TT del AFN obtenemos la TT del AFD equivalente aplicando el método de la Construcción de Subconjuntos, así:

1) El estado inicial del AFD es clausura-ε(0) = {0};

2) hacia({0}, a) = {0,1} y hacia({0}, b) = {0}.

TT	a	b
{0}-	{0,1}	{0}
{0,1}	?	?

3) hacia({0,1}, a) = {0,1} y hacia({0,1}, b) = {0,2}.

TT	a	b
{0}-	{0,1}	{0}
{0,1}	{0,1}	{0,2}
{0,2}+	?	?

4) hacia({0,2}, a) = {0,1} y hacia({0,2}, b) = {0}.

TT	a	b
{0}-	{0,1}	{0}
{0,1}	{0,1}	{0,2}
{0,2}+	{0,1}	{0}

(5) Renombrando los estados obtenemos, finalmente, la siguiente tabla de un AFD:

nombre anterior	nombre nuevo
0	0
{0,1}	1
{0,2}	2

y la tabla es:

TT	a	b
0-	1	0
1	1	2
2+	1	0

Analizando las TTs de los AFDs en los Ejemplos 4 y 5, observamos que son diferentes; por lo tanto, hemos encontrado dos AFDs diferentes que reconocen al mismo lenguaje. La equivalencia entre dos autómatas se demuestra en el próximo capítulo.

* Ejercicio 8 *

Diseñe un AFN que tenga 4 transiciones-ε, 2 ciclos y 3 estados finales. Luego, aplique la "Construcción de Subconjuntos" para hallar un AFD equivalente al AFN que diseñó.

* Ejercicio 9 *

Sea el AFN con la siguiente Tabla de Transiciones:

TT	a	b
0-	{1,3}	{1}
1	-	{1,2}
2	-	{2,3}
3+	{3}	{1,2,3}

Construya un AFD equivalente.

4 DOS OPERACIONES FUNDAMENTALES CON AUTÓMATAS FINITOS

"Un lenguaje es Regular si es reconocido por algún Autómata Finito". En este capítulo describiremos algoritmos que posibilitan la construcción de Autómatas Finitos capaces de reconocer LRs más complejos, gracias a la posibilidad de aplicar ciertas operaciones sobre AFDs.

> *Nota 1*
> La descripción de estas operaciones está basada en un texto originado en un curso dictado en 1981 en U.C.L.A. (*Universidad de California en Los Angeles*), EE.UU.

4.1 OPERACIONES CON AUTÓMATAS FINITOS DETERMINÍSTICOS

En esta sección operaremos con Autómatas Finitos. Si bien existen algoritmos especiales para obtener la unión y la concatenación de dos AFDs, podemos resolver estas operaciones con lo que conocemos del Algoritmo de Thompson. Lo mismo ocurre con la clausura de Kleene de un AFD y con la clausura positiva de un AFD.

En cambio, no podemos utilizar el Algoritmo de Thompson para dos operaciones muy importantes y muy útiles: el complemento de un AFD y la intersección de dos AFDs. Por ello, en este capítulo nos concentraremos solo en estas dos operaciones.

Nota 2
Los algoritmos que se describen no pueden ser aplicados a AFNs.

Nota 3
Para ejemplificar las dos operaciones que se desarrollarán, supondremos la existencia de estos dos LRs:

L_1: "Todas las palabras sobre el alfabeto {a,b} que comienzan y terminan con la misma letra", y

L_2: "Todas las palabras sobre el alfabeto {a,b} que comienzan con b y tienen longitud mayor o igual que tres".

El lenguaje L_1 es reconocido por un AFD cuya definición formal es:

$$M_1 = (\{0,1,2,3,4\}, \{a,b\}, T_1, 0, \{2,4\}),$$

con la función de transiciones T_1 representada mediante la TT:

TT	a	b
0-	1	3
1	2	1
2+	2	1
3	3	4
4+	3	4

* Ejercicio 1 *
Verifique que el AFD construido reconoce a L_1 y escriba la hipótesis que se asume.

* Ejercicio 2 *
¿La cadena a pertenece a L_1? ¿La cadena b pertenece a L_1? Justifique su respuesta.

El lenguaje L_2, a su vez, es reconocido por un AFD cuya definición formal es:
$M_2 = (\{5,6,7,8\}, \{a,b\}, T_2, 5, \{8\})$, con la función T_2 representada por la TT:

TT	a	b
5-	-	6
6	7	7
7	8	8
8+	8	8

* Ejercicio 3 *
Verifique que el AFD construido reconoce a L_2.

4.1.1 COMPLEMENTO DE UN AFD

El complemento de un AFD ya es un AFD que se obtiene invirtiendo los estados finales y no finales, es decir:

> Todo estado no final del AFD dato será un estado final del AFD complemento, y todo estado final del AFD dato será un estado no final del AFD complemento. Por esta situación, es muy importante que el AFD dato esté completo, ya que su estado de rechazo se convierte en un estado final.

Formalmente, sea un AFD $M_1 = (Q_1, \Sigma, T_1, q_1, F_1)$. Entonces, el autómata $M = M_1^c$ se define formalmente así: $M = (Q_1, \Sigma, T_1, q_1, Q_1 - F_1)$.

Observe que la única diferencia entre el autómata dato y el autómata resultado radica en el conjunto de estados finales: en el autómata resultado, el conjunto $Q_1 - F_1$ es el complemento, con respecto al conjunto total de estados, del conjunto de estados finales del autómata original, F_1.

Ejemplo 1
Sea el autómata M_2 que reconoce al lenguaje L_2 descripto en la Nota 3. Este autómata es un AFD pero está incompleto porque el estado 5 no tiene transición por el símbolo a.

Para hallar el AFD complemento, primero debemos completar el autómata dato con el agregado del estado de rechazo; si llamamos 9 a este estado, la TT completa de M_2 queda así:

TT	a	b
5-	9	6
6	7	7
7	8	8
8+	8	8
9	9	9

El AFD complemento es $M_2^c = (\{5,6,7,8,9\}, \{a,b\}, T, 5, \{5,6,7,9\})$, con la siguiente TT:

TT	a	b
5-+	9	6
6+	7	7
7+	8	8
8	8	8
9+	9	9

Observe que el estado 8 se ha transformado en un estado de rechazo, por lo que desde él no se puede llegar a un estado final. En consecuencia, su presencia es solo para completar el AFD obtenido y puede eliminarse si no es imprescindible que la TT esté completa.

El LR que reconoce el autómata dato, M_2, es: "Todas las palabras sobre el alfabeto {a,b} que comienzan con b y tienen longitud mayor o igual que 3". Por lo tanto, el LR reconocido por el AFD complemento será: "Todas las palabras sobre el alfabeto {a,b} que no comienzan con b o que comienzan con b y tienen longitud menor que 3".

* Ejercicio 4 *
¿La palabra vacía pertenece al LR complemento hallado? Justifique su respuesta.

Ejemplo 2
Supongamos un AFD con la siguiente Tabla de Transiciones:

TT	a	b
0-+	1	2
1+	1	0
2+	0	1

* Ejercicio 5 *
a) Obtenga el AFD complemento del AFD descripto mediante su TT en el Ejemplo 2;
b) Lo obtenido, ¿es un AFD? Justifique su respuesta.

Una aplicación muy útil del AFD complemento se presenta cuando se debe hallar un AFD que reconozca un LR cuyas palabras no tienen cierto patrón de caracteres.

Ejemplo 3
Supongamos que se desea hallar un AFD que reconozca al lenguaje: "Todas las palabras sobre {a,b} que no contienen la secuencia bab".

La solución más sencilla consiste en resolver este problema en dos pasos:

1) Hallar un AFD que reconozca al lenguaje: "Todas las palabras sobre {a,b} que contienen la secuencia bab".
2) Hallar el AFD complemento del anterior.

* Ejercicio 6 *
Resolver el problema planteado en el Ejemplo 3. Describa la Definición Formal del AFD.

* Ejercicio 7 *
Hallar un AFD que reconozca al lenguaje: "Todas las palabras sobre {a,b,c} que no comienzan con la secuencia caab". Describa la Definición Formal del AFD.

* Ejercicio 8 *
Hallar un AFD que reconozca al lenguaje: "Todas las palabras sobre {a,b,c} que no terminan con la secuencia abc". Describa la Definición Formal del AFD.

4.1.2 INTERSECCIÓN DE DOS AFDs

El autómata intersección de dos AFDs también es un AFD, y reconoce las palabras comunes a los LRs reconocidos por los dos AFDs datos. Los estados del AFD intersección son pares ordenados de estados, uno de cada AFD dato.

Formalmente: Sea un AFD $M_1 = (Q_1, \Sigma, T_1, q_1, F_1)$ y sea un AFD $M_2 = (Q_2, \Sigma, T_2, q_2, F_2)$. Entonces, el autómata $M = M_1 \cap M_2 = (Q, \Sigma, T, q_0, F)$ tiene los siguientes componentes:

$Q = Q_1 \times Q_2$
$q_0 = (q_1,q_2)$
$F = F_1 \times F_2$ y
T, la función de transiciones, es: $T((p,q), x) = (T_1(p,x), T_2(q,x)) \ \forall\ p \in Q_1 \wedge \forall\ q \in Q_2$

En la construcción del AFD intersección debemos tener en cuenta las siguientes consideraciones:
− Todos los estados son pares ordenados. Si uno de estos estados es un ESTADO ERRÓNEO (un estado inalcanzable desde el estado inicial o un estado desde el cual no se puede llegar a un estado final), entonces el par ordenado representará un estado erróneo.
− Normalmente, muchos "posibles" estados del AFD intersección serán estados erróneos y, por ello, no deberíamos considerarlos en la TT final.

Consecuentemente, una buena técnica para hallar el AFD intersección es:

1) Obtener el estado inicial y colocarlo en la primera fila de la TT;

2) Determinar las transiciones desde el estado inicial y agregar nuevas entradas en la tabla a medida que surjan nuevos estados, siempre que éstos no sean estados erróneos;

3) Repetir este proceso para cada nuevo estado, hasta que no se puedan crear más estados;

4) Tener en cuenta que los únicos estados finales son aquellos en los que ambos estados del par ordenado son finales.

Ejemplo 4
Sean los dos autómatas definidos en la Nota 3:

$M_1 = (\{0,1,2,3,4\}, \{a,b\}, T_1, 0, \{2,4\})$, con la función T_1 representada por la TT:

TT	a	b
0−	1	3
1	2	1
2+	2	1
3	3	4
4+	3	4

y $M_2 = (\{5,6,7,8\}, \{a,b\}, T_2, 5, \{8\})$, con la función T_2 representada por la TT:

TT	a	b
5−	−	6
6	7	7
7	8	8
8+	8	8

Para hallar el AFD intersección entre estos dos AFDs, construimos su TT de acuerdo a lo descripto anteriormente. Comenzamos por su estado inicial, que es (0,5), y las transiciones útiles que partan de él. Obtenemos, así, esta primera presentación de la TT:

TT	a	b
(0,5)-	-	(3,6)
(3,6)	?	?

Estado por estado, completamos la TT de esta manera:

TT	a	b
(0,5)-	-	(3,6)
(3,6)	(3,7)	(4,7)
(3,7)	(3,8)	(4,8)
(4,7)	(3,8)	(4,8)
(3,8)	(3,8)	(4,8)
(4,8)+	(3,8)	(4,8)

Nota 4
Observamos que hay tres estados que tienen el mismo COMPORTAMIENTO.

> Dos o más estados tienen el mismo comportamiento si todos son estados no finales o todos son estados finales, y, además, sus filas son idénticas.

En este ejemplo, los estados (3,7), (4,7) y (3,8) tienen el mismo comportamiento. Observe que aunque el estado (4,8) tiene una fila idéntica a los tres estados mencionados anteriormente, no tiene el mismo comportamiento porque es un estado final y los otros tres no lo son.

Que dos o más estados tengan el mismo comportamiento significa que son equivalentes. Por lo tanto, es suficiente con que exista uno de ellos. En consecuencia, podemos simplificar la TT de esta manera:

TT	a	b
(0,5)-	-	(3,6)
(3,6)	(3,7)	(3,7)
(3,7)	(3,7)	(4,8)
(4,8)+	(3,7)	(4,8)

y, renombrando los estados, resulta la siguiente TT:

TT	a	b
0-	-	1
1	2	2
2	2	3
3+	2	3

El proceso que hemos aplicado para simplificar el AFD forma parte de un algoritmo para "hallar el AFD mínimo". Lo veremos, más detalladamente, en el capítulo 6 de este volumen.

* Ejercicio 9 *
Describa qué debe ocurrir para que el AFD intersección reconozca la palabra vacía.

* Ejercicio 10 *
Dadas las ERs (a+b)*aa y b(a+b)*, obtenga la TT del AFD intersección.

* Ejercicio 11 *
Dadas las ERs (ab*)* y b(a+b)*, obtenga la TT del AFD intersección. ¿Es un AFD? Justifique su respuesta.

5 DEL AUTÓMATA FINITO A LA EXPRESIÓN REGULAR

Algunos programas de computadora que resuelven problemas sobre LRs requieren que el dato sea presentado como un AFD que reconoce a un LR. En los capítulos anteriores se ha visto cómo se puede diseñar un AF a partir de un LR (ya sea que esté expresado mediante una frase o a través de una ER). La conclusión de este proceso se describe en el capítulo 6 de este volumen, que muestra cómo obtener el mejor AFD: el AFD MÍNIMO.

Otros programas de computadora, como el *lex*, requieren que el dato sea una ER; pero no siempre resulta sencillo encontrar una ER correcta a partir de la frase en Lenguaje Natural que describe a cierto LR. Piensen, por ejemplo, qué sucede cuando en una frase aparecen "Y" (resuelto con la intersección de AFDs) y "NO" (resuelto con el complemento de un AFD), dos operaciones que no tienen su equivalencia en las ERs.

Por eso, muchas veces es más fácil y más seguro dibujar el DT de un AF que reconozca al lenguaje, hallar su TT y luego aplicar un método para obtener una ER a partir de este autómata, que intentar encontrar una ER en forma directa.

En este capítulo se presenta un método tal que, dado un AFD o un AFN que reconoce a cierto LR, nos permite obtener una ER que representa al LR dado.

> *Nota 1*
> La descripción de este método está basado en un texto originado en un curso dictado en 1981 en U.C.L.A. (*Universidad de California en Los Angeles*), EE.UU.

5.1 DEPURACIÓN DEL AUTÓMATA FINITO

Hay ciertas operaciones que producen autómatas con dos tipos de ESTADOS ERRÓNEOS:
a) estados a los que no se puede llegar desde el estado inicial (llamados estados inalcanzables);
b) estados que no conducen a un estado final (como los estados de *rechazo*).

La DEPURACIÓN del AF es el primer paso del algoritmo que describimos en este capítulo, y consiste en detectar y eliminar todos los estados erróneos.

Ejemplo 1
Sea la siguiente TT de un AFD completo:

TT	a	b
0-	1	2
1	1	4
2	4	5
3	4	4
4+	5	4
5	5	5

Se observa que desde el estado 0, estado inicial del autómata, podemos llegar a los estados 1, 2, 4 y 5. Sin embargo, desde el estado inicial no podemos llegar al estado 3, por lo que éste estado es un *estado inalcanzable*.

Por otro lado, el estado 5 es utilizado solo para completar el autómata; desde este estado no se puede llegar a un estado final. Todos los otros estados tienen caminos que conducen al estado final 4.

En consecuencia, la depuración del autómata exige que se eliminen los dos estados erróneos que hemos encontrado, 3 y 5, como así también todas las referencias que haya a estos estados. Por ello, la tabla depurada es la siguiente:

TT	a	b
0-	1	2
1	1	4
2	4	-
4+	-	4

* Ejercicio 1 *
Sea el AFD representado por la siguiente TT:

TT	a	b
0-	1	6
1	1	4
2	-	-
4+	-	4
5	2	2
6+	-	5

Obtenga la TT depurada.

* Ejercicio 2 *
Sea el AFN representado por la siguiente TT:

TT	a	b
0-	{1,2}	{0,6}
1	{1}	{4}
2	-	-
4+	-	{4,5}
5	{2}	{2}
6+	-	{5}

Obtenga la TT depurada.

Una vez depurado el AF, el segundo paso de este algoritmo consiste en establecer un SISTEMA DE ECUACIONES que tendrá tantas ecuaciones como estados haya en el AF ya depurado. Cada ecuación describe el COMPORTAMIENTO de un estado del AF. Cuando este sistema de ecuaciones sea resuelto, su resultado será la ER buscada.

5.2 RESOLUCIÓN DEL SISTEMA DE ECUACIONES

Partimos de la TT depurada y habrá tantas ecuaciones como estados tenga el AF. En términos generales, cada ecuación tiene:

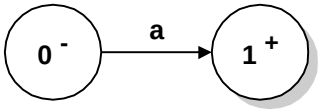
a) En su lado izquierdo, el estado cuyo comportamiento es descripto por una ecuación;

b) En su lado derecho, la unión de términos que representan las transiciones que parten del estado mencionado en el lado izquierdo. Cada uno de estos términos se forma, en general, mediante el carácter que etiqueta a la transición concatenado con el estado de llegada. La única excepción se produce cuando se desarrolla la ecuación de un estado final; en este caso, se agrega un término ε

para indicar que este estado, por ser final, acepta una cadena que finalice cuando el AF llega a este estado.

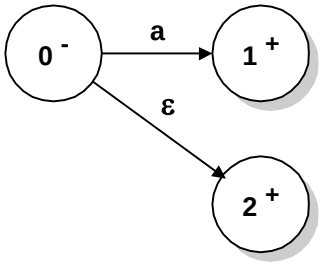
Ejemplo 2
Algunos casos que se pueden presentar son los siguientes:

(1)



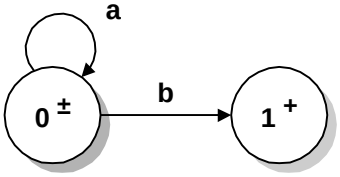
El sistema tendrá dos ecuaciones:
0 = a1 (estado 0 = carácter a concatenado con estado 1)
1 = ε (porque el estado 1 es final)

(2)



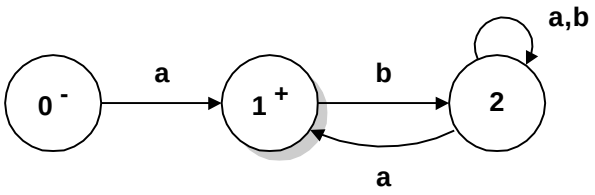
El sistema tendrá tres ecuaciones:
0 = a1 + ε2 = a1 + 2 (porque ε es la identidad para la concatenación)
1 = ε
2 = ε

(3)



El sistema tendrá dos ecuaciones:
0 = a0 + b1 + ε (porque el estado 0 también es final)
1 = ε

(4)



El sistema tendrá tres ecuaciones (y no importa que sea un AFN):
0 = a1
1 = b2 + ε
2 = a2 + b2 + a1 = (a+b)2 + a1

Los casos (3) y (4) del ejemplo anterior tienen la característica de tener *ciclos* o *bucles*. Entonces, en la resolución de la correspondiente ecuación se debe realizar un proceso de REDUCCIÓN que se describe a continuación.

5.3 REDUCCIONES

El proceso de reducción se debe realizar en una ecuación que tiene uno o más *ciclos*. En estas ecuaciones se puede apreciar que el estado que figura en el lado izquierdo de la ecuación también se encuentra en el lado derecho. Por ello, una ecuación con estas características se las denomina ecuación recursiva.

El formato general de una ecuación recursiva es:

$$e = \alpha e + \beta$$

donde:

- e representa un estado,
- α es una ER, y
- β es una expresión que puede estar formada por caracteres del alfabeto, el símbolo ε y estados.

Ejemplo 3

El caso (3) del Ejemplo 2 muestra la ecuación recursiva: $0 = a0 + b1 + \varepsilon$. Entonces, e es el estado 0, α es la ER a y β es $(b1 + \varepsilon)$.

En el caso (4), la ecuación recursiva es: $2 = (a+b)2 + a1$. Entonces, e es el estado 2, α es la ER $(a+b)$ y β es a1.

Cuando existe una ecuación recursiva, ésta se debe reducir así:

$e = \alpha e + \beta$ se convierte en $e = \alpha^*\beta$

Observe que el operador de *unión* que aparece en la ecuación original se transforma en el operador de *concatenación* en la ecuación reducida. Este hecho es muy importante porque la unión es conmutativa pero la concatenación no lo es:

$e = \alpha e + \beta$ es equivalente a $e = \beta + \alpha e$

pero $e = \alpha^*\beta$ no es equivalente a $e = \beta\alpha^*$

Entonces, para aplicar esta reducción es necesario que, en la ecuación original, el término αe siempre figure primero, a la izquierda del término β.

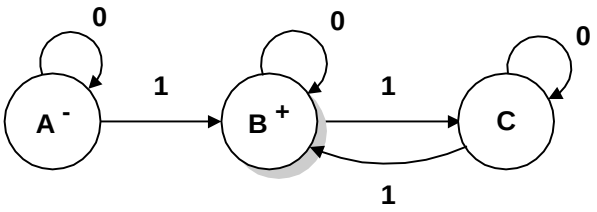
Ejemplo 4

La ecuación recursiva $0 = a0 + b1 + \varepsilon = a0 + (b1 + \varepsilon)$ se reduce a $0 = a^* (b1 + \varepsilon)$.
La ecuación recursiva $2 = (a+b)2 + a1$ se reduce a $2 = (a+b)^* a1$.

Finalmente, la resolución del sistema de ecuaciones planteado puede realizarse en cualquier orden, con una sola excepción: la ecuación que corresponde al estado inicial debe ser la última en resolverse porque en su lado derecho quedará la ER buscada.

Ejemplo 5

Sea el lenguaje "Todos los números binarios que tienen una cantidad impar de 1s". Supongamos que necesitamos hallar una ER que represente a este LR y no resulta sencillo. Sin embargo, podemos diseñar con facilidad un AF que reconozca a este lenguaje, con el siguiente Diagrama de Transiciones:



La Tabla de Transiciones de este autómata es:

TT	0	1
A-	A	B
B+	B	C
C	C	B

El primer paso que debemos realizar para hallar una ER a partir de este AF es determinar si hay estados erróneos y, en caso afirmativo, depurar el autómata. En esta TT se comprueba que no hay estados erróneos, por lo que seguimos con el segundo paso del método.

A partir de la TT de este autómata obtenemos el siguiente sistema de tres ecuaciones:

$A = 0A + 1B$

$B = 0B + 1C + \varepsilon$ (por ser un estado final)

$C = 0C + 1B$

Para resolver este sistema de ecuaciones, supongamos que comenzamos con la tercera ecuación que, como se observa, es una ecuación recursiva que está escrita en el orden que corresponde. En consecuencia:

$C = 0C + 1B$ se resuelve como $C = 0^*1B$.

En la ecuación del estado B, que también es recursiva, reemplazamos C por lo que acabamos de obtener y resolvemos:

$$B = 0B + 1C + \varepsilon = 0B + 1(0^*1B) + \varepsilon = (0 + 10^*1)B + \varepsilon = (0 + 10^*1)^*\varepsilon = (0 + 10^*1)^*$$

Tratamos, finalmente, la ecuación para el estado inicial:

$$A = 0A + 1B = 0A + 1(0+10^*1)^* = 0^*1(0+10^*1)^*$$

Por lo tanto, una ER que denota al lenguaje "Todos los números binarios con una cantidad impar de 1s" es:

$0^*1(0+10^*1)^*$

Ejemplo 6

Supongamos resolvemos el mismo sistema de ecuaciones del Ejemplo 5, pero comenzando por otra ecuación. El sistema original es:

$A = 0A + 1B$

$B = 0B + 1C + \varepsilon$

$C = 0C + 1B$

Ahora resolvemos, primero, la ecuación que corresponde al estado B:

$$B = 0B + 1C + \varepsilon = 0B + (1C + \varepsilon) = 0^*(1C+\varepsilon)$$

Reemplazamos en la ecuación del estado C y resolvemos:

$$C = 0C+1B = 0C+1(0^*(1C+\varepsilon)) = 0C+10^*1C+10^* = (0+10^*1)C+10^* = (0+10^*1)^*10^*$$

Dado que en la ecuación de A, el estado inicial, se ve claramente que A está en función de B, debemos completar, primero, la resolución del estado B que, a su vez, está en función del estado C.

Por lo tanto:

$$B = 0^* (1C + \varepsilon) = 0^*1C + 0^* = 0^*1(0+10^*1)^*10^*+0^*$$

Luego:

$$A = 0A+1B = 0A+1(0^*1(0+10^*1)^*10^*+0^*) = 0^*1(0^*1(0+10^*1)^*10^*+0^*) = 0^*10^*1(0+10^*1)^*10^*+0^*10^*$$

Hemos hallado una segunda ER que define al lenguaje "Todos los números binarios con una cantidad impar de 1s". Esta ER, $0^*10^*1(0+10^*1)^*10^*+0^*10^*$, es, obviamente, equivalente a la ER obtenida en el ejemplo anterior (representa el mismo LR) aunque, visualmente, es más compleja.

* Ejercicio 3 *
Obtenga una ERs para el AF del Ejemplo 1.

* Ejercicio 4 *
Si se parte de un AFN, ¿el método sufre alguna modificación? Justifique su respuesta.

* Ejercicio 5 *
Sea el lenguaje: "Todas las palabras sobre {a,b} que contienen, por lo menos, tres aes consecutivas y que no terminan con b". Obtenga dos ERs que representen este LR, partiendo de un autómata que lo reconozca. La ambigüedad del lenguaje natural posibilita que la frase que describe al lenguaje pueda ser interpretada de varias formas; si es así, escriba las hipótesis de trabajo que utiliza. Para construir el AF del cual partirá, debe utilizar "complemento de un AFD" e "intersección de dos AFDs".

* Ejercicio 6 *
Obtenga una ER para el AFN del Ejercicio 2.

6 OBTENCIÓN DEL AFD MÍNIMO

Hay LRs que pueden ser reconocidos por muchos AFDs y AFNs. El AFD mínimo es el AFD con la mínima cantidad de estados que reconoce a un LR. Por ello, se lo considera como el único autómata óptimo asociado a la aceptación de determinado LR.

En este capítulo se presenta un algoritmo tal que, dado un AFD que reconoce a cierto LR, permite obtener el AFD mínimo. La importancia de obtener el AFD mínimo tiene tres aplicaciones inmediatas:

1) Determinar si dos o más AFDs son equivalentes;

2) Probar la equivalencia de dos o más Expresiones Regulares;

3) El AFD mínimo es el que tiene la TT más reducida, con menor cantidad de filas, hecho que beneficia la implementación del AFD mediante un programa de computadora; si la TT es muy grande, la obtención del AFD mínimo puede significar la diferencia entre poder o no poder implementar la TT como una matriz en memoria.

➔ Dos o más AFDs son equivalentes si el AFD mínimo que se obtiene a partir de ellos es el mismo (solo pueden diferir los nombres de los estados).

➔ Dos o más Expresiones Regulares son equivalentes si son reconocidas por el mismo AFD mínimo.

6.1 EL ALGORITMO

En un ejemplo de un capítulo anterior se presentó una ER que es relativamente compleja: a*ba+aba*. Si el carácter a significa "cualquier dígito decimal" y el carácter b significa "punto decimal", esta ER puede representar el lenguaje de los números reales sin signo y en punto fijo en algún Lenguaje de Programación. Partiremos de este LR para desarrollar el primer ejemplo referido a la obtención del AFD mínimo.

> * Ejercicio 1 *
> Escriba dos palabras del LR representado por la ER de la frase anterior, que muestren dos tipos de números reales en punto fijo distintos.

Ejemplo 1
Sea, entonces, la ER a*ba+aba*. Si, en base a los conocimientos adquiridos, construimos un AFN que reconozca a este lenguaje y luego aplicamos la "Construcción de Subconjuntos", podemos obtener un AFD con la siguiente TT:

TT	a	b
0-	1	2
1	3	4
2	7	-
3	3	2
4+	5	-
5+	6	-
6+	6	-
7+	-	-

* Ejercicio 2 *

Verifique que el AFD representado por esta TT reconoce al LR que corresponde.

Continuamos con el Ejemplo 1. Cada guión ("–") en la TT indica una transición al estado de rechazo. En la implementación manual del algoritmo que desarrollaremos no es necesario completar este AFD porque, como ya se verá más adelante, el estado de rechazo quedará, de todas maneras, separado. No obstante, y para mayor claridad en este primer contacto con el algoritmo, trabajaremos sobre el AFD completo. Agregamos, entonces, el estado de rechazo (sea su nombre 8) y obtenemos la siguiente TTs:

TT	a	b	
0–	1	2	
1	3	4	
2	7	8	
3	3	2	
4+	5	8	
5+	6	8	
6+	6	8	
7+	8	8	
8	8	8	(estado de rechazo)

El primer paso del algoritmo consiste en particionar el conjunto de estados del AFD en dos clases: la clase de los estados no finales (el estado de rechazo siempre es no final) y la clase de los estados finales (el estado inicial, por ejemplo, puede ser final, aunque no en este autómata):

TT	a	b		
0–	1	2		
1	3	4		
2	7	8	clase C0	(estados no finales)
3	3	2		
8	8	8		

4+	5	8		
5+	6	8	clase C1	(estados finales)
6+	6	8		
7+	8	8		

➔ El COMPORTAMIENTO de un estado es la sucesión de estados de llegada de las transiciones que parten de ese estado, para un ordenamiento establecido de los caracteres del alfabeto. En otras palabras: es la n-upla formada por los estados de su fila en la TT.

En términos de la TT, el comportamiento de un estado se obtiene mediante la lectura de la fila que corresponde a ese estado. Por ejemplo, el comportamiento del estado 0 es (1,2) y el comportamiento del estado 4 es (5,8).

➔ Dos estados son equivalentes si: (1) pertenecen a la misma clase y (2) tienen el mismo comportamiento.

El segundo paso del algoritmo consiste en detectar estados equivalentes. En la clase C0 no hay estados equivalentes porque los cuatro estados que la componen tienen comportamientos diferentes.

En cambio, en la clase C1 detectamos una equivalencia, porque el estado 5 tiene el mismo comportamiento que el estado 6: (6,8).

Analicemos la situación del conjunto de estados equivalentes {5,6} (esto se generaliza a cualquier conjunto de estados equivalentes). Si dos o más estados son equivalentes, significa que solo uno de ellos es necesario porque tienen el mismo comportamiento. En consecuencia, el tercer paso del algoritmo consiste en reducir el AFD, el cual quedará con uno solo de los estados equivalentes como representante del conjunto de estados equivalentes.

Los restantes estados del conjunto de estados equivalentes son eliminados de la TT, teniendo la precaución de reemplazar, primero, cada referencia a un nombre de estado que será eliminado por el nombre del estado que permanecerá en la tabla como representante del conjunto. Por convención, se elige el estado de nombre menor (5, en este caso), por lo que cada referencia al estado 6 debe reemplazarse por una referencia al estado 5 antes de eliminar la fila del estado 6.

En este caso hay una sola referencia al estado 6 (5=>a=>6), por lo que la reemplazamos por el estado 5, quedando la tabla de esta manera:

TT	a	b	
0-	1	2	
1	3	4	
2	7	8	clase C0
3	3	2	
8	8	8	
4+	5	8	
5+	5	8	clase C1
7+	8	8	

Al hacer la reducción, observamos que en la clase C1 aparece otro conjunto de estados equivalentes, compuesto por los estados 4 y 5. Se aplica nuevamente el mismo procedimiento, eligiendo al estado 4 como representante, y la TT queda así:

TT	a	b	
0-	1	2	
1	3	4	
2	7	8	clase C0
3	3	2	
8	8	8	
4+	4	8	
7+	8	8	clase C1

Los estados equivalentes que se han descubierto en este paso del algoritmo son estados de equivalencia inmediata, es decir: es una situación de equivalencia que se detecta inmediatamente, por simple inspección ocular de la TT. Pero hay otros estados que pueden ser equivalentes aunque no se lo determine en forma inmediata, por lo que se debe desarrollar un nuevo paso en este algoritmo, como veremos a continuación.

Una vez finalizada la reducción de la TT mediante la detección de equivalencias inmediatas (si las hay), el cuarto paso del algoritmo consiste en construir una tabla auxiliar que denominamos Tabla de Transiciones por Clases (TTC). Esta tabla se forma a partir de la última TT obtenida y las

transiciones indicadas en la tabla original para los estados que han quedado en esta última tabla, reemplazando cada estado de llegada por la clase a la que pertenece ese estado. En este caso, obtenemos la siguiente TTC:

TTC	a	b	
0-	C0	C0	
1	C0	C1	
2	C1	C0	clase C0
3	C0	C0	
8	C0	C0	
4+	C1	C0	
7+	C0	C0	clase C1

➔ El COMPORTAMIENTO POR CLASES de un estado es la sucesión de clases de llegada de las transiciones que parten de ese estado, para un ordenamiento establecido de los caracteres del alfabeto, tal como se representa en la TT por Clases.

El quinto paso del algoritmo consiste en aplicar el proceso de búsqueda de estados equivalentes por clase, que son aquellos estados que están en la misma clase y que tienen el mismo comportamiento por clases. Ejemplo: en la clase C0, los estados 0, 3 y 8 son *equivalentes por clase* porque los tres están en la misma clase y tienen el mismo comportamiento por clases: (C0, C0).

A diferencia de lo que sucede cuando encontramos estados de equivalencia inmediata, la detección de estados equivalentes por clase no conduce a la elección inmediata de un representante del conjunto de estados y a la eliminación de los restantes estados, sino que provoca la partición de la clase en subclases que se caracterizan por contener estados que son equivalentes por clase. Y este proceso continúa hasta que se llega a la situación en que cada clase está formada por un solo estado, con lo que se deduce que el AFD no se puede minimizar más, o bien algunas clases están constituidas por más de un estado equivalente por clase, pero ya no se pueden particionar. En este último caso, se elige un representante de la clase como se ha hecho anteriormente.

Continuamos el proceso. La última TTC que hemos obtenido es:

TTC	a	b	
0-	C0	C0	
1	C0	C1	
2	C1	C0	clase C0
3	C0	C0	
8	C0	C0	
4+	C1	C0	
7+	C0	C0	clase C1

Analizando esta TTC, detectamos un solo conjunto de estados equivalentes por clase: el formado por los estados 0, 3 y 8. En los restantes casos, cada subclase estará integrada por un solo estado. Reordenando los estados y separándolos en subclases, obtenemos la siguiente TTC:

TTC	a	b	
0-	C0	C0	
3	C0	C0	clase C0
8	C0	C0	
1	C0	C1	clase C2
2	C1	C0	clase C3
4+	C1	C0	clase C1
7+	C0	C0	clase C4

Nota 1
Cuando una clase está formada por un solo estado, significa que ese estado pertenecerá al AFD mínimo.

Ahora debemos *actualizar* esta TTC porque, al haber particionado las clases originales, la TTC ha cambiado. Por ejemplo, leyendo la *TT original*, vemos que el estado 0 tiene comportamiento (1,2); y se observa que, con esta última partición, el estado 1 pertenece a la clase C2, mientras que el estado 2 pertenece a la clase C3. Por lo tanto, la fila del estado 0 en la TTC ya no es (C0,C0) sino que ahora es (C2,C3). Además, no hace falta establecer el comportamiento de aquellos estados que son únicos en sus respectivas clases. En consecuencia, la nueva clase TTC es la siguiente:

TTC	a	b	
0-	C2	C3	
3	C0	C3	clase C0
8	C0	C0	
1			clase C2
2			clase C3
4+			clase C1
7+			clase C4

Las clases C2, C3, C1 y C4 ya son mínimas porque cada una está constituida por un solo estado. En cuanto a la clase C0, se distingue que está formada por tres estados que ahora tienen diferentes comportamientos por clases, por lo que se deben separar, colocando a cada uno en una nueva subclase. Finalmente, todas las clases quedan con un único estado y, en consecuencia, el AFD no se puede minimizar más de lo que se ha logrado en los primeros pasos de este algoritmo.

En definitiva, y volviendo a la TT original, el AFD mínimo sin completar (es decir, sin el estado de rechazo) está representado por la siguiente tabla:

TT	a	b
0-	1	2
1	3	4
2	7	-
3	3	2
4+	4	-
7+	-	-

Ejemplo 2
Supongamos un AFD completo con la siguiente TT:

TT	a	b	
0-	1	5	
1	5	2	
2+	2	3	
3+	4	2	
4+	3	4	
5	5	5	(estado de rechazo)

Para hallar el AFD mínimo, primero particionamos la TT en dos clases: la clase de estados no finales y la clase de estados finales:

TT	a	b	
0-	1	5	
1	5	2	clase C0
5	5	5	
2+	2	3	
3+	4	2	clase C1
4+	3	4	

En ninguna de las dos clases detectamos estados de equivalencia inmediata. Entonces, continuamos con el proceso y construimos la TTC:

TTC	a	b	
0-	C0	C0	
1	C0	C1	clase C0
5	C0	C0	
2+	C1	C1	
3+	C1	C1	clase C1
4+	C1	C1	

En la clase C0, los estados 0 y 5 son equivalentes por clase, y en la clase C1 sus tres estados tienen el mismo comportamiento por clase. Entonces, aplicando el mismo criterio que en el ejemplo anterior, reagrupamos los estados equivalentes por clase en una misma subclase y creamos una nueva clase para el estado que no tiene equivalencias por clase. La nueva TTC es:

TTC	a	b	
0-	C0	C0	clase C0
5	C0	C0	
1			clase C2
2+	C1	C1	
3+	C1	C1	clase C1
4+	C1	C1	

Al surgir la nueva clase C2, tenemos que actualizar la TTC, siempre a partir de la TT original:

TTC	a	b	
0-	C2	C0	clase C0
5	C0	C0	
1			clase C2
2+	C1	C1	
3+	C1	C1	clase C1
4+	C1	C1	

Al hacerlo, vemos que, por ahora, solo se modifica la clase C0:

TTC	a	b	
0-	C2	C0	clase C0
5	C0	C0	clase C3
1			clase C2
2+	C1	C1	
3+	C1	C1	clase C1
4+	C1	C1	

Al surgir la nueva clase C3, debemos determinar si no aparecen otras nuevas clases, siempre a partir de la TT original.

Al analizar la TT original, observamos que no se ha creado ninguna subclase nueva, y dado que la clase C1 no se puede particionar, se elige a un estado como representante (sea el estado 2) y se reemplaza toda referencia a los estados que serán eliminados (3 y 4) por una referencia al estado 2 en la TT original.

Finalmente, la TT del AFD mínimo sin estado de rechazo es:

TT	a	b
0-	1	-
1	-	2
2+	2	2

Ejemplo 3. "El estado inicial también es estado final"
Sea la ER a^*+b^+a. Supongamos que, a partir de esta ER, obtenemos un AFD reconocedor con la siguiente TT, que tiene la particularidad que el estado inicial también es estado final:

TT	a	b	
0-+	1	2	
1+	1	4	
2	3	2	
3+	4	4	
4	4	4	(estado de rechazo)

* Ejercicio 3 *
Verifique que el AFD obtenido y su TT son correctos.

Continuamos con el Ejemplo 3. Para hallar el AFD mínimo, primero debemos particionar la TT en la clase de estados no finales y la clase de estados finales:

TT	a	b	
2	3	2	clase C0
4	4	4	
0-+	1	2	
1+	1	4	clase C1
3+	4	4	

➔ Nótese que el estado 0 forma parte de la clase de estados finales, sin perder su condición de ser el estado inicial.

En ninguna de las dos clases detectamos estados de equivalencia inmediata. Continuamos con el proceso y construimos, entonces, la TTC:

TTC	a	b	
2	C1	C0	clase C0
4	C0	C0	
0-+	C1	C0	
1+	C1	C0	clase C1
3+	C0	C0	

En la clase C0, los estados 2 y 4 no son equivalentes por clase; y en la clase C1, los estados 0 y 1 tienen el mismo comportamiento por clases, mientras que el estado 3 no lo tiene. Entonces, aplicando el mismo criterio que en los ejemplos anteriores, reagrupamos los estados equivalentes en una misma subclase y creamos dos nuevas clases para los estados que no tienen equivalencias por clase. La nueva TTC es:

TTC	a	b	
2			clase C0
4			clase C2
0-+	C1	C0	
1+	C1	C0	clase C1
3+			clase C3

Al surgir las clases C2 y C3, tenemos que actualizar la TTC, siempre a partir de las transiciones de la TT original:

TTC	a	b	
2			clase C0
4			clase C2
0-+	C1	C0	
1+	C1	C2	clase C1
3+			clase C3

Analizando la última TTC, vemos que hay tres clases con un solo estado cada una; obviamente, ninguna de ella se podrá particionar. En cuanto a la clase C1, queda con dos estados que tienen diferentes comportamientos por clase; por lo tanto, debemos separarlos. Finalmente, obtendremos una TTC en la que cada clase tiene un único estado; por lo tanto, el AFD original ya es mínimo.

* Ejercicio 4 *
Obtenga la TT del AFD mínimo que reconoce al LR representado por la ER a*ba*+a*b*aa.

* Ejercicio 5 *
Verifique que las ERs 0*1(0+10*1)* y 0*10*1(0+10*1)*10*+0*10*, de los ejemplos 5 y 6 del capítulo anterior, son equivalentes.

* Ejercicio 6 *
Determine si los siguientes AFDs son equivalentes:

TT1	a	b
0-	1	0
1+	2	3
2+	1	4
3	4	5
4	3	5
5	1	5

TT2	a	b
0-+	1	5
1	2	4
2	3	4
3	2	4
4	5	3
5+	5	1

* Ejercicio 7 *
Sea el AFN con la siguiente TT:

TT	a	b	c	ε
0-	{1}	{1}	-	-
1	{2,3}	-	{3}	{4,5}
2	{2}	-	{4}	{5}
3+	-	-	{1,2}	{5}
4	{4,5}	{0,2}	-	-
5+	-	{3,4}	-	-

Obtenga el AFD mínimo.

7 MÁQUINA DE TURING

Una Máquina de Turing (MT) es un autómata determinístico con la capacidad de reconocer cualquier lenguaje formal. Una MT está formada por los siguientes elementos:

(1) un alfabeto A de símbolos o caracteres del lenguaje a reconocer;

(2) una cinta infinita dividida en una secuencia de celdas, cada una de las cuales contiene un carácter o un blanco. Aquí se coloca la cadena a analizar. El resto de las celdas contiene blancos;

(3) una cabeza de cinta que puede, en un solo paso: leer el contenido de una celda de la cinta, reemplazarlo con otro carácter o dejar el mismo, y reposicionarse para apuntar a la celda contigua, ya sea la de la derecha (avanzando) o la de la izquierda (retrocediendo). La cabeza de cinta nunca se puede mover a la izquierda de la celda en la que se encuentra el primer carácter de la cadena a analizar;

(4) un alfabeto A' de símbolos o caracteres que pueden ser escritos en la cinta por la cabeza de cinta;

(5) un conjunto finito de estados que incluye un estado inicial, donde comienza la ejecución, y un conjunto (posiblemente vacío) de estados finales que producen la terminación de la ejecución cuando se llega a alguno de ellos;

(6) un programa: conjunto de reglas que nos dicen, en función del estado en que se encuentra la MT y del carácter leído por la cabeza de cinta, qué carácter escribir en la cinta en la misma posición, en qué dirección se debe mover la cabeza de cinta y a qué estado debe realizar la transición.

Ejemplo 1

Sea $L_6 = \{a^n b^n c^n / n \geq 1\}$. Este lenguaje no es Regular (por lo tanto, no puede ser reconocido por un AFD) y tampoco es un LIC (por lo tanto, no puede ser reconocido por un AFP). En cambio, L_6 sí puede ser reconocido por una MT.

Construiremos una MT con 9 estados, con un alfabeto A = {a,b,c} y con un alfabeto A' = {X,Y,Z}. Además, simbolizaremos con R al movimiento de la cabeza de cinta hacia la derecha y con L al movimiento de la cabeza de cinta hacia la izquierda.

Téngase en cuenta que la cinta tendrá un blanco (□) después del último carácter de la cadena a analizar (en realidad tendrá infinitos blancos). Este blanco actuará como un centinela, como el '\0' para las cadenas en ANSI C.

Completamos la MT que reconoce al lenguaje L_6 con un programa en el que cada regla tiene la siguiente notación:

estado actual – carácter leído, carácter escrito, dirección – estado de llegada

```
e0 – a,X,R – e1  /* e0 es el estado inicial */
e0 – Y,Y,R – e6  /* ya marcó todas las aes */
e1 – a,a,R – e1  /* lee todas las aes que hay en la cinta */
e1 – b,Y,R – e2  /* marca la primera b */
e1 – Y,Y,R – e4  /* lee una b marcada y cambia de estado */
e2 – b,b,R – e2  /* lee todas las bes que hay en la cinta */
e2 – c,Z,L – e3  /* marca una c */
e2 – Z,Z,R – e5  /* lee una c marcada y cambia de estado */
e3 – Z,Z,L – e3  /* lee las ces marcadas y retrocede */
e3 – b,b,L – e3  /* lee las bes y retrocede */
e3 – Y,Y,L – e3  /* lee las bes marcadas y retrocede */
e3 – a,a,L – e3  /* lee las aes y retrocede (buscando una X) */
```

```
e3 – X,X,R – e0  /* encuentra una X y avanza */
e4 – Y,Y,R – e4  /* lee bes marcadas y avanza */
e4 – b,Y,R – e2  /* lee una b y la marca */
e5 – Z,Z,R – e5  /* lee ces marcadas y avanza */
e5 – c,Z,L – e3  /* lee una c y la marca */
e6 – Y,Y,R – e6  /* encuentra todas las bes marcadas */
e6 – Z,Z,R – e7  /* encuentra la 1° c marcada */
e7 – Z,Z,R – e7  /* verifica que todas las ces están marcadas */
e7 - □,□,L – e8+ /* reconoce la cadena y transita al estado final */
```

* Ejercicio 1 *
Verifique si la MT construida reconoce abc y aaabbbccc.

* Ejercicio 2 *
Verifique si la MT construida no reconoce abcc.

* Ejercicio 3 *
Construya una MT que reconozca al lenguaje $L_7 = \{a^n b^n c^{n+1} / n \geq 1\}$.

Cabe destacar que la MT no solo reconoce cualquier lenguaje formal de la Jerarquía de Chomsky, sino que también puede computar (puede realizar lo que hace una computadora).

Ejemplo 2
Dado un número binario, construimos una MT que obtiene su complemento. Sean A = A' = {0,1}. Entonces, la MT puede tener el siguiente programa:

```
e0 – 0,1,R – e0  /* e0 es el estado inicial */
e0 – 1,0,R – e0
e0 – □,□,L – e1 /* transita al estado final */
```

* Ejercicio 4 *
Construya una MT que duplique una cadena de aes y bes en su cinta. Ejemplo: si la MT comienza con abbaa□ en su cinta, luego de procesar su programa debe terminar con abbaa□abbaa.

8 BIBLIOGRAFÍA

Alfonseca Cubero, Enrique y otros (2007): "*Teoría de Autómatas y Lenguajes Formales*; McGraw-Hill/Interamericana, España.

Cohen, Daniel (1986): "*Introduction to Computer Theory*"; John Wiley & Sons, EE. UU.

Gottlieb, Peter (1981): "*Theoretical Models in Computer Science*", U.C.L.A., EE.UU.

Hopcroft, John E. y otro (1979): "*Introduction to Automata Theory, Languages, and Computation*"; Addison-Wesley, EE. UU.

Muchnik, Jorge D. (2005): "*Autómatas Finitos y Expresiones Regulares*"; Editorial CEIT, Cdad. de Bs. As.

9 EJERCICIOS RESUELTOS

Agradezco la colaboración de la Prof. Ana Díaz Bott

CAPÍTULO 1

* Ejercicio 1 *
AFD = ({0,1,2,3}, {a,b,c}, T, 0, {3})

TT	a	b	c
0-	1	1	1
1	2	2	2
2	3	3	3
3+	3	3	3

* Ejercicio 2 *
AFN = ({0,1,2}, {a,b}, T, 0, {2})

TT	a	b
0-	{0,1}	{0}
1	{1}	{1,2}
2+	-	-

CAPÍTULO 2

* Ejercicio 8 *
Esta es la TT del AFN por Thompson:

TT	a	b	ε
0-	-	-	{1,7}
1	-	-	{2,3}
2	{4}	-	-
3	-	{5}	-
4	-	-	{6}
5	-	-	{6}
6	-	-	{1,7}
7+	-	-	-

CAPÍTULO 3

* Ejercicio 1 *
Clausura-ε (0) = {0, 1, 2, 3, 6}
Clausura-ε (1) = {1, 2, 3, 6}
Clausura-ε (2) = {1, 2, 3, 6}
Clausura-ε (3) = {1, 2, 3, 6}
Clausura-ε (4) = {4, 5}
Clausura-ε (5) = {5}
Clausura-ε (6) = {1, 2, 3, 6}

* Ejercicio 3 *
Clausura-ε (M) = {0, 1, 2, 3, 4, 6}

* Ejercicio 4 *
Hacia(M,a) = {1}

* Ejercicio 9 *

TT	a	b
{0}-	{1,3}	{1}
{1,3}+	{3}	{1,2,3}
{1}	-	{1,2}
{3}+	{3}	{1,2,3}
{1,2,3}+	{3}	{1,2,3}
{1,2}	-	{1,2,3}

TT	a	b
0-	1	2
1+	3	4
2	-	5
3+	3	4
4+	3	4
5	-	4

CAPÍTULO 4

* Ejercicio 5 *

TT	a	b
0-	1	2
1	1	0
2	0	1

No es un AFD. ¿Por qué?

* Ejercicio 10 *

TT	a	b
(0,3)-	-	(0,4)
(0,4)	(1,4)	(0,4)
(1,4)	(2,4)	(0,4)
(2.4)+	(2,4)	(0,4)

CAPÍTULO 5

* Ejercicio 1 *

TT	a	b
0-	1	6
1	1	4
4+	-	4
6+	-	-

* Ejercicio 3 *

0 = a1+b2
1 = a1+b4
2 = a4
4 = b4+ε
=> ER = aa*bb*+bab*

CAPÍTULO 6

* Ejercicio 4 *

TT	a	b
0-	1	2
1	3	2
2+	4	5
3+	3	2
4+	4	-
5	6	5
6	7	-
7+	-	-

Ya es un AFD mínimo.

CAPÍTULO 7

* Ejercicio 1 *

Reconocimiento de abc:

```
e0 - a,X,R - e1
e1 - b,Y,R - e2
e2 - c,Z,L - e3
e3 - Y,Y,L - e3
e3 - X,X,R - e0
e0 - Y,Y,R - e6
e6 - Z,Z,R - e7
e7 - □,□,L - e8+
```

* Ejercicio 2 *

No reconocimiento de abcc:

```
e0 - a,X,R - e1
e1 - b,Y,R - e2
e2 - c,Z,L - e3
e3 - Y,Y,L - e3
e3 - X,X,R - e0
e0 - Y,Y,R - e6
e6 - Z,Z,R - e7
e7 - c,?,? - ?
```