No determinismo y autómatas

Como ya fue anticipado, en los autómatas finitos, la primera forma de no determinismo se manifiesta con la presencia de múltiples opciones posibles en la función de transición. Esto es, estando el AF en un cierto estado y ante una cierta señal de entrada, el desempeño del autómata admite múltiples posibilidades. Al igual que en el caso de los autómatas finitos deterministas, los *no deterministas* quedan definidos como una quíntupla:

$$AFND = (\Sigma_E, Q, q_0, A, f)$$

y su característica distintiva está en la definición de las transiciones, que aquí no es más una función de estado a estado, sino una relación de transición, tomando la forma:

$$f: Q \times \Sigma_E \rightarrow \boldsymbol{P}(Q)$$

donde $\boldsymbol{P(Q)}$ es el conjunto de todos los subconjuntos que se pueden formar con elementos del conjunto **Q** (*conjunto potencia de **Q***).

Debe notarse que esta indeterminación no significa, en modo alguno, incertidumbre o aleatoriedad en el desempeño final del autómata. Más bien significa incertidumbre en cuanto al esfuerzo requerido al evaluar una cierta cadena. En efecto, la cantidad de intervalos de tiempo que le insume a un AFD aceptar o rechazar una cadena será como máximo igual a la longitud de la misma. Por el contrario, en el caso de un Autómata Finito No Determinista (AFND) es difícil acotar anticipadamente este esfuerzo. Esto es así porque la operación del autómata implica una búsqueda exhaustiva en un árbol de posibilidades, que será más frondoso según sea mayor la cantidad de próximos estados posibles en cada transición, llamada factor de ramificación $\mathbf{F_R}$, donde $\mathbf{F_R = |}f\mathbf{(q,e)|}$, $\mathbf{q \in Q}$, $\mathbf{e \in \Sigma_E}$. Se debe tener presente que el tiempo demandado para resolver este tipo de problemas es exponencial, lo que implica un esfuerzo muy considerable aún para casos muy simples; y para casos complejos estos problemas no tienen solución práctica, por lo que son denominados problemas intratables.

Surge naturalmente un primer interrogante: ¿con qué finalidad se proponen autómatas finitos no deterministas? La respuesta es simple: la flexibilidad que ofrece el no determinismo es de gran ayuda en el diseño de autómatas destinados a reconocer lenguajes complejos, lo que fue advertido por primera vez por Rabin y Scott en 1959. Esto significa que se facilita el diseño de ciertos autómatas pagando el costo de un mayor esfuerzo de operación: se obtienen diseños más simples a costa de algoritmos menos eficientes.

Autómatas Finitos No Deterministas (AFND)

Según estas afirmaciones el no determinismo es un recurso conveniente para definir máquinas abstractas. Para comprobarlo, se propone el Ejemplo 4.1, en el que un AFND reconoce el mismo tipo de sentencias que el AFD del Ejemplo 3.3 del capítulo anterior.

Ejemplo 4.1: Proponer y analizar un AFND destinado a reconocer cadenas que responden a la forma general α **= (0 + 1)*1000**.

Con algunas pruebas y una mínima experiencia, se llega a que el AFND de la Figura 4.1 es capaz de reconocer toda cadena que responda a la forma general indicada: un estado inicial (**p**) que está destinado a admitir cualquier prefijo sobre el alfabeto de entrada, que es seguido por una secuencia de cuatro estados (**q**-**r**-**s**-**t**) destinados a asegurar la presencia del sufijo deseado en la cadena que es validada. Nótese la condición no determinista en el estado **p** ante la entrada del símbolo **1**.

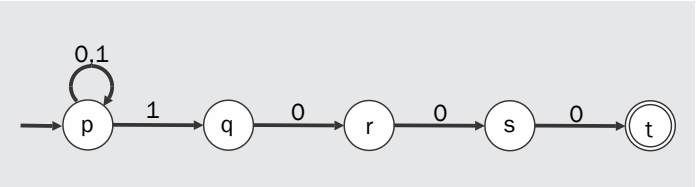


Figura 4.1: Grafo del AFND que reconoce cadenas α **= (0+1)*1000**.

Su representación formal toma la forma habitual:

AFND = (Σ_E, Q, q_o, A, *f*)

Σ_E = {0, 1}

Q = {p, q, r, s, t}

q_o = p

A = {t}

f:	0	1
→ p	{p}	{p, q}
q	{r}	
r	{s}	
s	{t}	
* t		

Tabla 4.1: Definición formal del AFND del Ejemplo 4.1.

En estos casos, tanto el *árbol de descripciones instantáneas* como el *plano estados-entradas*, que fueron utilizados en el estudio de autómatas finitos deterministas en el capítulo anterior, son representaciones muy apropiadas para visualizar el proceso de aceptación de una cadena por parte del AFND. En las Figuras 4.2 y 4.3, se muestran ambas representaciones cuando el AFND propuesto evalúa la cadena **β = 1011000**.

Puede comprobarse que finalmente el AFND acepta la cadena al arribar al estado **t** pero, para ello, fue necesario explorar numerosas opciones que quedan representadas tanto en el árbol de descripciones instantáneas como en el plano estados-entradas mostrados en las figuras antes indicadas. El proceso exploratorio es inevitable ya que el AFND no puede establecer anticipadamente si existe un camino hacia la configuración final de aceptación y, en caso afirmativo, no puede saber cuál es. Como ya fue anticipado, la relativa facilidad con la que se definió el autómata requerido tiene como contrapartida un esfuerzo exploratorio en cada proceso de validación de cadenas.

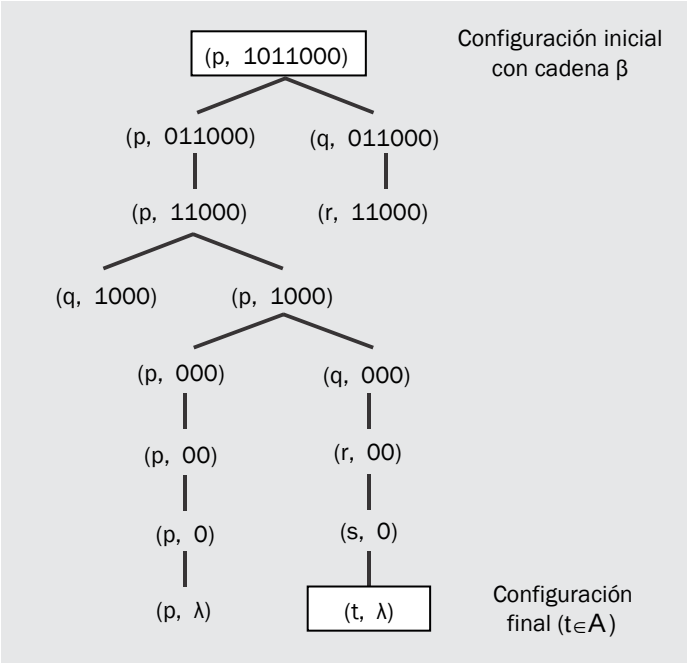


Figura 4.2: Árbol de descripciones instantáneas.

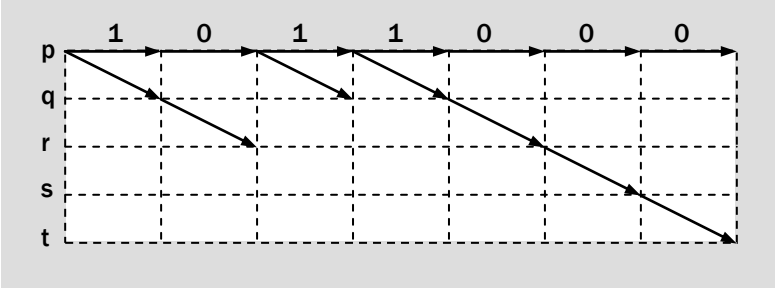


Figura 4.3: Representación del plano estados-entradas.

Nótese que en dos oportunidades el AFND se vio imposibilitado de continuar operando con la cadena de entrada estudiada. Esto es debido a la inexistencia de transiciones que lo permitan

cuando habiendo alcanzado los estados **q** o **r** el autómata lee un **1**. También debe observarse que en otras dos oportunidades el autómata lee la cadena completa, pero en un solo caso arriba al estado final de aceptación. Se concluye que el AFND acepta esta cadena, pero con un esfuerzo que no fue posible anticipar.

Ejemplo 4.2

Se propone una variante al ejercicio anterior, de manera que el AFND reconozca cadenas de la forma general α**=(0 + 1)*1(0 + 1)00**. Es decir que los sufijos de largo cuatro comiencen con el símbolo **1**, no tengan restricciones en el segundo símbolo y terminen con **00**.

Nuevamente el AFND se obtiene a partir de algunas pruebas, y como se comprueba en el grafo de la Figura 4.4, presenta una leve variante con respecto al AFND del ejercicio anterior.

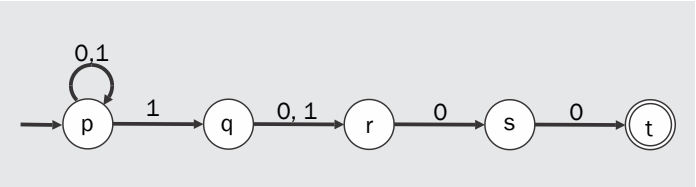


Figura 4.4: Grafo del AFND que reconoce cadenas α = (0+1)*1(0+1)00.

Su representación formal es:

AFND = (Σ_E, Q, q_o, A, *f*)

Σ_E = {0, 1}

Q = {p, q, r, s, t}

q_o = p

A = {t}

***f*:**	0	1
→ p	{p}	{p, q}
q	{r}	{r}
r	{s}	
s	{t}	
* t		

Tabla 4.2: Definición formal del AFND del Ejemplo 4.2.

El árbol de descripciones instantáneas y el plano estados-entradas que corresponden al análisis de la misma cadena **β = 1011000** por parte del AFND son mostrados en las Figuras 4.5 y 4.6.

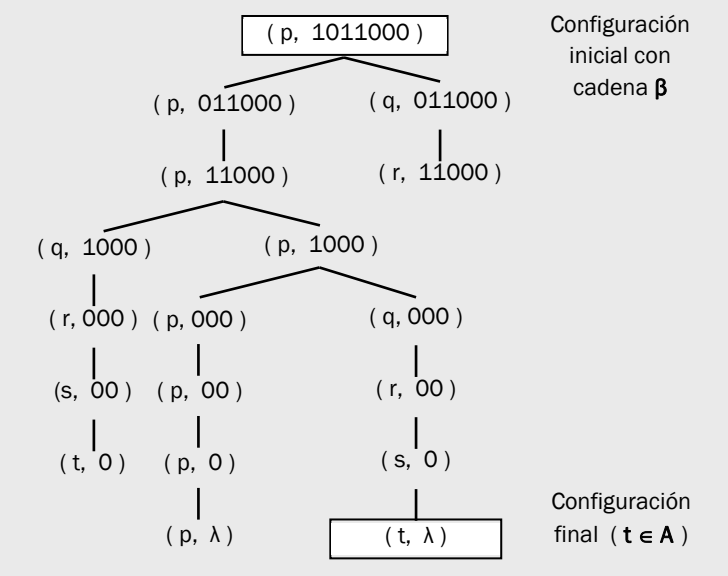


Figura 4.5: Árbol de descripciones instantáneas.

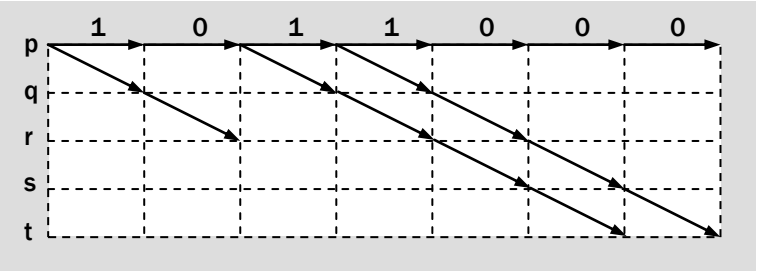


Figura 4.6: Representación del plano estados-entradas.

Como puede comprobarse, este lenguaje es menos restrictivo que el anterior y el nuevo AFND necesita un esfuerzo adicional para comprobar que la cadena **β** puede ser aceptada. Esto se debe a que la nueva transición incorporada en el estado **q** permitió un movimiento adicional hasta el estado de aceptación, que fue alcanzado antes de completarse la lectura de la cadena.

Transiciones Lambda

La definición de los AFND puede ampliarse para incluir transiciones de un estado a otro que no dependan de ninguna entrada, las que son denominadas ***transiciones λ***. Con el símbolo **λ**, se representa la cadena vacía (**| λ | = 0**) y estos autómatas se denominarán AFND-λ.

En cualquier descripción instantánea de la máquina, la opción de una de estas transiciones **λ** se considera de la misma forma que si se tratara de una transición convencional ante una cierta entrada $\mathbf{e} \in \mathbf{\Sigma}_E$. Es así que para contemplarlas se incorpora una columna adicional a la relación ***f*** a fin de considerar los pares **(q, λ)** y se asume además que cada estado tiene su propia transición **λ** que cicla sobre sí mismo. Por lo tanto, las transiciones **λ** son consistentes con la matriz no determinista que representa la conducta del AFND ya estudiado antes. Luego la expresión de ***f*** debe ser reescrita de manera de adicionar **λ** a los símbolos del alfabeto de entrada:

$$f\colon Q \times (\Sigma_E \cup \{\lambda\}) \rightarrow \boldsymbol{P}(Q)$$

A todos los pares de estados que están vinculados por una transición λ es conveniente reunirlos en un conjunto **T** denominado *relación de transición* **λ**. Por lo ya anticipado, en esta relación, se incluyen además las transiciones de los estados con ellos mismos, reconociéndose el carácter reflexivo de **T**. Se tiene entonces:

$$\mathbf{T} = \{(p, q) / q \in f(p, \lambda)\} \cup \{(q, q) / q \in Q\}$$

es decir que, utilizando notación de relaciones:

$(p, q) \in \mathbf{T}$, si $q \in f(p, \lambda)$ y se representa pTq

$(q, q) \in \mathbf{T}$, ya que $\forall q \in Q: q \in f(q, \lambda)$ y se representa qTq

Luego se incorporan al conjunto **T** nuevos pares ordenados, que se originan en la aplicación de la propiedad transitiva a los pares anteriores. Se obtiene así el cierre transitivo de la relación, que toma la forma:

$$\mathbf{T}^* = \mathbf{T} \cup \{(p, q) / pTr \wedge rTq\}$$

Extensión al tratamiento de palabras

La relación de transición ***f*** presentada en el punto anterior define los estados alcanzables a partir de cada uno de los estados y para cada uno de los símbolos del alfabeto de entrada. Esta relación puede ser extendida para describir lo que ocurre a partir de cada estado si en lugar de recibir símbolos aislados se recibe una secuencia concatenada de los mismos. La noción de relación de transición extendida $\boldsymbol{f}^{\mathbf{e}}$ para reconocer cadenas o palabras puede inducirse a partir de un razonamiento similar al hecho cuando se definió la función de transición extendida para el AFD en el capítulo anterior, solo que considerando además la eventualidad de las transiciones **λ**. Se obtiene una relación que queda definida como:

$$f^e\colon Q \times (\Sigma_E \cup \{\lambda\})^* \rightarrow \boldsymbol{P}(Q)$$

Esto significa que al considerarse la lectura de una cadena genérica **β** deben contemplarse las posibles transiciones **λ**; luego si **β** es definida como $\boldsymbol{\beta = a_1 a_2 a_3 a_4 \ldots a_n}$, debe preverse por parte

del AFND la lectura de cadenas tales como $\boldsymbol{\alpha=\lambda^*a_1\lambda^*a_2\lambda^*a_3\lambda^*a_4 \ldots \lambda^*a_n\lambda^*}$. Esta cadena puede, a su vez, ser sucesivamente redefinida como la concatenación de un símbolo y un sufijo tal como se hizo anteriormente. En forma intuitiva:

$$f^e(q,\beta)=f^e(q,\lambda^*\chi)=f^e(f(q,\lambda^*),\chi)= \ldots =f(f(f(f(q,\lambda^*),a_1),\lambda^*),\ldots,a_n),\lambda^*)$$

La cadena original **β** es entonces aceptada por el autómata si

$$f^e(q_o, \alpha) \in A \quad \text{para} \quad \alpha \in (\Sigma_E \cup \{\lambda\})^+$$

Equivalencia con autómatas finitos deterministas

Al tratarse la equivalencia entre autómatas finitos, se destacan los dos teoremas que se enuncian a continuación (sin ser demostrados).

Teorema 1

Todo AFND -λ puede ser redefinido como un AFND equivalente y, para ello, se deben eliminar sus transiciones **λ**.

Enunciado

Sea un AFND -λ que es definido como M = (Σ_E, Q, q_o, A, *f*) donde

$$f\colon Q \times (\Sigma_E \cup \{\lambda\}) \rightarrow \boldsymbol{P}(Q)$$

Este autómata puede siempre ser redefinido como un AFND en el que M' = (Σ_E, Q, q_o, A', ***f'***) donde $\boldsymbol{f'}: Q \times \Sigma_E \rightarrow \boldsymbol{P}(Q)$.

Procedimiento de aplicación

La relación ***f'*** se obtiene a partir de ***f*** con las siguientes operaciones:

a. Para todo par de estados distintos relacionados por transiciones λ (**pT*q**, **p≠q**), y para cada transición **r=*f*(q,e)** donde **e≠λ**, es decir **r=*f*(*f*(p,λ*), e)**, se define una nueva transición **r=*f*(p,e)**. Se incorporan las nuevas transiciones en ***f***.
b. Para todo par de estados distintos relacionados por transiciones λ (**qT*r**, **q≠r**), y para cada **q=*f*(p,e)** donde **e≠λ,** es decir, **r=*f*(*f*(p,e),λ*)**, se define la nueva transición **r=*f*(p,e)**. Se incorporan las nuevas transiciones en ***f***.
c. En caso de existir una transición lambda desde el estado inicial a un estado de aceptación (**q_0T*r**, **r∈A**), hacer **q_0∈A**.
d. Eliminar todas las transiciones **λ**, lo que implica eliminar la columna **λ** de la tabla de ***f***.

Nótese que A' = A ya que las transiciones definidas en el punto (b) mantienen el mismo conjunto de estados de aceptación, siempre que no se presente el caso definido en el punto (c).

Teorema 2

Por cada AFND, siempre existe un AFD que acepta el mismo lenguaje y estos dos autómatas se dice que son equivalentes.

Enunciado

Sea un AFND definido como M = (Σ_E, Q, q_o, A, *f*) donde

$$f\colon Q \times \Sigma_E \rightarrow \boldsymbol{P}(Q).$$

Siempre existirá un AFD M' = (Σ_E, Q', c_o , A', *f*') con $f'\colon Q' \times \Sigma_E \rightarrow Q'$, que es equivalente.

Procedimiento de aplicación – Caso 1

Se considera el caso de un AFND en el que ya han sido eliminadas sus transiciones **λ**.

En un cierto estado, **q** y ante una entrada **e**, el AFND tiene la posibilidad a pasar a cualquiera de los estados pertenecientes al subconjunto ***f*(q, e)** y actúa como si se encontrara en cualquiera de ellos. Por esta razón, todo el subconjunto se comporta como un único estado y así se cumplen los sucesivos movimientos hasta completarse la lectura de la cadena de entrada. Ésta se conside-

rará aceptada si el subconjunto de estados alcanzado incluye entre sus miembros a algunos de los estados que pertenecen a conjunto de estados de aceptación.

Resulta así que los estados de **Q'** se corresponden con los subconjuntos de **Q** que se obtienen de explorar todas las transiciones sobre $\Sigma_E \times Q$ y se reconoce que pertenecen a **A'** aquellos subconjuntos que incluyen a alguno de los estados definidos en **A**.

Considerando que $q_i \in Q$ y $c_i \in Q'$ donde $c_i=\{q_j,\ldots,q_n\}$, el procedimiento es el siguiente:

a. Reconocer el estado inicial $c_o = \{q_o\}$.

b. Definir los nuevos estados $c_i = f'(c_k, e)$ para todo $e \in \Sigma_E$ y $k < i$.

c. Para todos los casos en que $c_i = \{\ldots, q_n, \ldots\}$ y $q_n \in A$ se reconoce que $c_i \in A'$.

Procedimiento de aplicación – Caso 2

Se considera el caso de un AFND-λ, es decir que no han sido previamente eliminadas las transiciones **λ** y el procedimiento es el siguiente:

a. Reconocer el estado inicial $c_o = f(q_o, \lambda^*) = \{p / q_o T^* p\}$.

b. Definir nuevos estados $c_i = f'(c_k, e)$ para todo $e \in (\Sigma_E \cup \{\lambda\})$ y $k < i$.

c. Para todos los casos en que $c_i = \{\ldots, q_n, \ldots\}$ y $q_n \in A$ se reconoce que $c_i \in A'$.

Debe notarse que, cualquiera sea el procedimiento empleado, es muy probable que el AFD equivalente no sea conexo e incluya estados indistinguibles, por lo que deba ser minimizado.

Ejemplo 4.3

Encontrar un AFD equivalente al AFND del Ejemplo 4.1 que estaba destinado a reconocer cadenas de la forma general $\alpha = (0+1)^*1000$.

Solución # 1 (aplicación de Teorema 2, Caso 1)

Por tratarse de un AFND, es decir un autómata sin transiciones **λ**, se aplica el caso 1 - Teorema 2:

$c_o = \{p\}$ (estado inicial del AFD equivalente)

$f'(c_o, 0) = \{p\} = c_o$ $\quad$ $f'(c_o, 1) = \{p, q\} = c_1$

$f'(c_1, 0) = \{p, r\} = c_2$ $\quad$ $f'(c_1, 1) = \{p, q\} = c_1$

$f'(c_2, 0) = \{p, s\} = c_3$ $\quad$ $f'(c_2, 1) = \{p, q\} = c_1$

$f'(c_3, 0) = \{p, t\} = c_4$ $\quad$ $f'(c_3, 1) = \{p, q\} = c_1$

$f'(c_4, 0) = \{p\} = c_0$ $\quad$ $f'(c_4, 1) = \{p, q\} = c_1$

$AFD = (\Sigma_E, Q', q_o', A', f')$

$\Sigma_E = \{0, 1\}$

$Q' = \{c_o, c_1, c_2, c_3, c_4\}$

$q_o' = c_o$

$A' = \{c_4\}$

f':	0	1
→ c_o	c_o	c_1
c_1	c_2	c_1
c_2	c_3	c_1
c_3	c_4	c_1
* c_4	c_o	c_1

Tabla 4.3: Definición formal del AFD del Ejemplo 4.3.

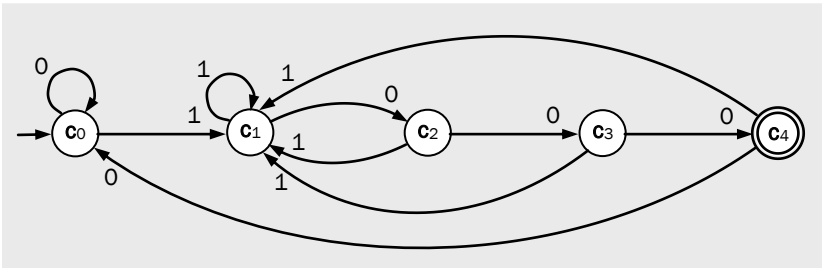


Figura 4.7: Grafo del AFD del Ejemplo 4.3.

Puede comprobarse que, como era de esperar, el AFD equivalente recientemente encontrado es el mismo ya estudiado en el Ejemplo 3.3 del capítulo anterior.

Solución # 2 (procedimiento práctico sobre la tabla de transición)

En la solución anterior (#1), se obtuvieron los elementos de Q' trabajando con la propia función de transición.

Esto mismo puede hacerse sobre la tabla de la función de transición, extendiéndola a medida que se presentan nuevos estados a partir del agrupamiento de estados del AFND, como se muestra en la Tabla 4.4. Lo expuesto ocurre en el caso del estado **pq** (1), para el cual se abre una nueva entrada **pq** (2) en la tabla y se completa la fila considerando las transiciones posibles para todas las entradas. Así se obtiene el nuevo estado **pr** (3) y vuelve a ser invocado el estado **pq** (4).

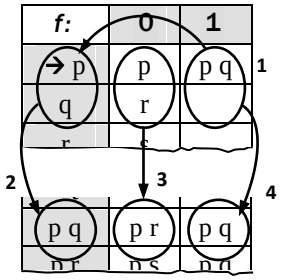


Tabla 4.4: Construcción de la función ***f*** del AFD.

Se prosigue de esta manera incorporando entradas a la tabla por cada nuevo estado que aparezca para el AFD equivalente a partir de la combinación de estados del AFND, como es el caso de los **pr**, **ps** y **pt**.

Nótese que serán estados de aceptación del AFD todos aquellos que incluyan al estado final **t** del AFND original.

Surge así un AFD no conexo, cuya función de transición se muestra en la Tabla 4.5. Nótese que es necesario eliminar los estados que no son accesibles desde el estado inicial, que están representados por las filas rayadas en la tabla.

Como último paso, restaría designar los estados como c_0, c_1, c_2, c_3 y c_4 , con lo que la Tabla 4.5 toma la forma de la Tabla 4.3.

f':	0	1
→ p	p	p q
q	r	
r	s	
s	t	
* t		
p q	p r	p q
p r	p s	p q
p s	p t	p q
* p t	p	p q

Tabla 4.5: Función ***f'*** del AFD equivalente no conexo.

Ejemplo 4.4

Encontrar un AFD equivalente al AFND del Ejemplo 4.2, que estaba destinado a reconocer cadenas de la forma general α **= (0+1)*1(0+1)00**.

f:	0	1
→ p	{p}	{p, q}
q	{r}	{r}
r	{s}	
s	{t}	
* t		

f':	0	1
→ p	p	p q
q	r	r
r	s	
s	t	
*t		
p q	p r	p q r
p r	p s	p q
p s	p t	p q
*p t	p	p q
p q r	p r s	p q r
p r s	p s t	p q
*p s t	p t	p q

Tabla 4.6: Funciones de transición del AFND y AFD equivalente (no conexo).

Se sigue el mismo procedimiento usado en el ejercicio anterior, en el que se desarrolla la función ***f'*** del AFD equivalente a partir de la tabla de la función ***f*** del AFND. Nótese que la tabla de ***f'*** representa al AFD no conexo y que los estados finales son ahora dos (**pt** y **pst**).

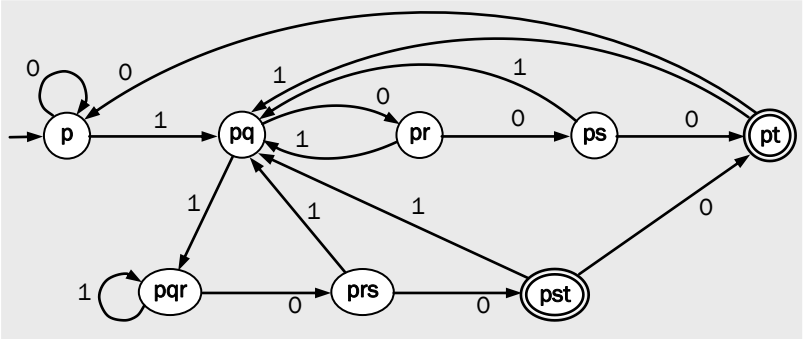


Figura 4.8: Gráfico del AFD equivalente del Ejemplo 4.4.

La definición formal del autómata equivalente es:

$$AFD = (\Sigma_E, Q, q_o, A, f')$$

donde: $\Sigma_E = \{0, 1\}$

$Q = \{p, pq, pr, ps, pt, pqr, prs, pst\}$

$q_o = p$

$A = \{pt, pst\}$

Ejemplo 4.5

En los Ejemplos 4.1 y 4.2, se propusieron AFND destinados a reconocer cadenas binarias que se distinguen por la composición de sus sufijos de largo cuatro. Luego, en los Ejemplos 4.3 y 4.4, se aplicaron los procedimientos propuestos para determinar los AFD equivalentes en cada uno de los casos.

En esta oportunidad, se propone una nueva variante, donde la cadena a ser reconocida responde a la forma general $\alpha = (0+1)^*(11)^+00$, es decir, que el sufijo de las cadenas a ser reconocidas se caracteriza a tener cualquier potencia (mayor que cero) de la subcadena **11,** que es seguida por la subcadena **00.**

En este caso, se requiere:

a) Definir un AFND para resolver el problema

b) Convertirlo a un AFD equivalente.

c) Representar el comportamiento de ambos autómatas ante una cadena **β = 0111100**.

a) Definición del AFND

Sin mayor dificultad, se puede comprobar que el grafo del AFND buscado es el representado en la Figura 4.9, que luego es definido formalmente y su función de transición es representada en la Tabla 4.7.

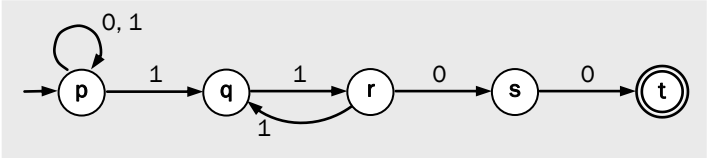


Figura 4.9: Grafo del AFND del Ejemplo 4.5.a.

La definición formal del AFND es:

$AFND = (\Sigma_E, Q, q_0, A, f)$

$\Sigma_E = \{0, 1\}$

$Q = \{p, q, r, s, t\}$

$q_0 = p$

$A = \{t\}$

f:	0	1
→ p	{p}	{p, q}
q		{r}
r	{s}	{q}
s	{t}	
* t		

Tabla 4.7: Función *f* del Ejemplo 4.5.a.

b) Conversión a AFD *equivalente*

Se desarrolla la función ***f'*** del AFD equivalente a partir de la tabla de la función ***f*** del AFND. Para ello, se incorporan a la tabla en forma progresiva los nuevos estados que corresponden a grupos de estados del autómata no determinista. Esta última función es representada en la Tabla 4.8.

f':	0	1
→ p	p	pq
q		r
r	s	q
s	t	
*t		
pq	p	pqr
pqr	ps	pqr
ps	pt	pq
*pt	p	pq

Tabla 4.8: Función de transición del AFD equivalente no conexo del Ejemplo 4.5.

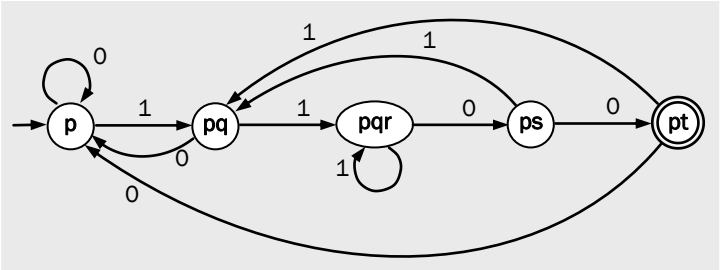


Figura 4.10: Grafo del AFD equivalente del Ejemplo 4.5.

c) Comportamiento de ambos autómatas ante la entrada β = 0111100

Para estudiar el comportamiento de ambos autómatas con la entrada **β = 0111100**, se presentan los árboles de descripciones instantáneas mostrados en la Figura 4.11.

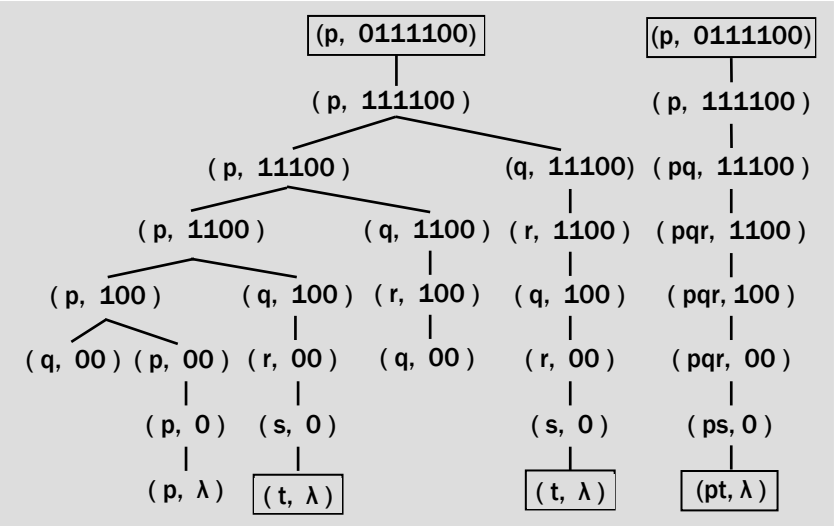


Figura 4.11: Árboles de descripciones instantáneas del AFND y AFD.

La segunda opción es representar la respuesta de los autómatas ante la cadena β en el plano estados-entradas, que para el AFND se muestra en la Figura 4.12. Por haber dos secuencias de movimientos que conducen a la configuración de aceptación y para mayor claridad, se utilizan dos tipos diferentes de líneas.

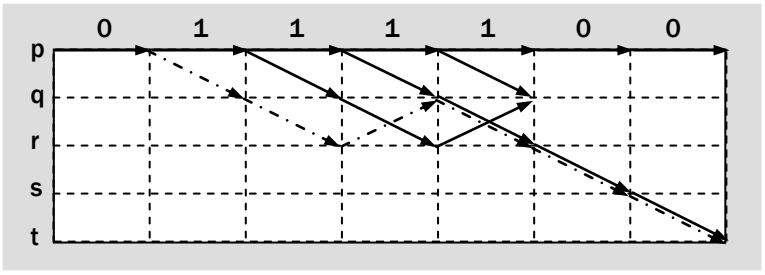


Figura 4.12: Plano estados-entradas del AFND del Ejemplo 4.5.

Como puede observarse, tanto el árbol de descripciones instantáneas como el plano estados-entradas ponen claramente en evidencia que el trabajo de aceptación de una cadena por el AFND involucra un proceso exploratorio. Se sugiere al lector analizar estas representaciones en detalle y justificar las condiciones que se presentan en cada caso.

La representación con el plano estados-entradas del mismo proceso por el *AFD* equivalente se muestra en la Figura 4.13. Nótese que, en este caso particular, el nuevo autómata tiene los mismos cinco estados que el AFND, de los cuales uno de ellos representa la condición de aceptación.

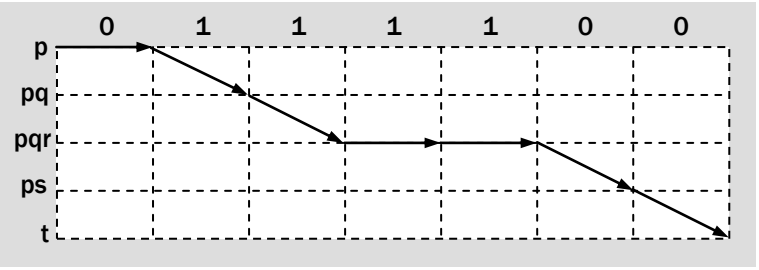


Figura 4.13: Plano estados-entradas del AFD del Ejemplo 4.5.

Es muy interesante analizar el aumento de la complejidad de los AFD equivalentes con respecto a los AFND originales en los Ejemplos 4.4 y 4.5. En algunos casos, como en el Ejemplo 4.4, a pesar de que el AFND es muy simple, no solo se comprueba un importante aumento en la cantidad de estados sino también la presencia de múltiples estados de aceptación. Además, y a diferencia de los AFND, el comportamiento de estos autómatas equivalentes ante diferentes cadenas de entrada no es obvio y requiere de una verificación cuidadosa. En otros casos, como en el Ejem-

plo 4.5, y a pesar de tratarse de un AFND muy similar al anterior, el AFD equivalente mantiene la misma cantidad de estados y un único estado de aceptación. Como se comprueba, no es fácil anticipar cuál será el esfuerzo requerido por la definición del AFD equivalente. Sin embargo, cualquiera sea el caso el beneficio es obvio, ya que el proceso de aceptación o rechazo de cadenas es directo, tal como se comprueba en el plano estados-entradas de la Figura 4.13.

Es necesario advertir que cuando los AFD equivalentes son complejos suele ser conveniente mantener la denominación de los estados tal cual se la obtuvo a partir de los estados originales del AFND y no asignarles nuevos nombres, ya que muy probablemente dificultarán su interpretación.

Ejemplo 4.6

Se presenta a continuación un AFND-λ destinado a reconocer el lenguaje $\mathbf{L=\{0^m1^n2^k \ / \ m,n,k \geq 0\}}$, cuyo grafo se representa en la Figura 4.14. Se pide: a) presentar la definición formal del autómata, b) encontrar un AFND equivalente sin transiciones **λ**, c) encontrar el AFD equivalente a partir de la función de transición, d) encontrar el mismo AFD equivalente trabajando sobre la tabla de la función de transición y e) representar el comportamiento del AFND y del AFD equivalente al leerse la cadena **β = 001122**.

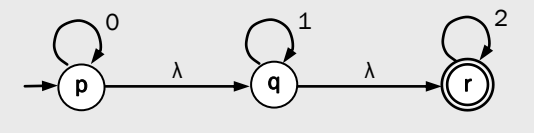


Figura 4.14: AFND-λ del Ejemplo 4.6.

a) Definición formal:

AFND-λ = (Σ_E, Q, q_o, A, *f*)

Σ_E = {0, 1, 2}

Q = {p, q, r}

q_o = p

A = {r}

***f*:**	0	1	2	λ
→ p	{p}			{q}
q		{q}		{r}
*r			{r}	

Tabla 4.9: Función de transición del AFND -λ.

b) Determinación del AFND sin transiciones λ:

Se define la relación de transición **T** y la clausura $\mathbf{T^*}$ del autómata:

T = {(p, p), (p, q), (q, q), (q, r), (r, r)}

T^* = {(p, p), (p, q), (q, q), (q, r), (r, r), (p, r)}

Luego se aplica el Teorema 1 que permite definir transiciones nuevas a partir de la función ***f*** y la clausura $\mathbf{T^*}$:

p = *f*(p, 0) y pT^*q → q = *f*(p, 0)

p = *f*(p, 0) y pT^*r → r = *f*(p, 0)

q = *f*(q, 1) y qT^*r → r = *f*(q, 1)

p debe ser ahora de aceptación

pT^*q y q = *f*(q, 1) → q = *f*(p, 1)

pT^*q y r = *f*(q, 1) → r = *f*(p, 1)

pT^*r y r = *f*(r, 2) → r = *f*(p, 2)

qT^*r y r = *f*(r, 2) → r = *f*(q, 2)

***f*:**	0	1	2
→*p	{p, q, r}	{q, r}	{r}
q		{q, r}	{r}
*r			{r}

AFND = (Σ_E, Q, q_o, {p, r}, *f*)

f: Q × Σ_E → ***P***(Q)

{{p,q,r},{q,r},{r}} ⊆ *P*(Q)

En la función de transición original de la Tabla 4.9, se elimina la columna **λ** y se incorporan las nuevas transiciones recientemente determinadas, obteniéndose la nueva función representada en la Tabla 4.10. El grafo de este AFND sin transiciones **λ** es representado en la Figura 4.15.

Tabla 4.10: Función de transición del AFND (sin λ).

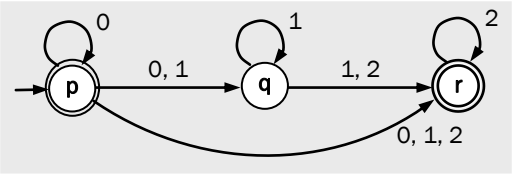


Figura 4.15: AFND sin λ del Ejemplo 4.6.

c) Determinación del AFD equivalente mediante la función de transición:

$c_0 = \{ p \}$

$f'(c_0, 0) = f(p, 0) = \{ p, q, r \} = c_1$

$f'(c_0, 1) = f(p, 1) = \{ q, r \} = c_2$

$f'(c_0, 2) = f(p, 2) = \{ r \} = c_3$

$f'(c_1, 0) = \{ p, q, r \} = c_1$

$f'(c_1, 1) = \{ q, r \} = c_2$

$f'(c_1, 2) = \{ r \} = c_3$

$f'(c_2, 0) = \varnothing$

$f'(c_2, 1) = \{ q, r \} = c_2$

$f'(c_2, 2) = \{ r \} = c_3$

$f'(c_3, 0) = \varnothing$

$f'(c_3, 1) = \varnothing$

$f'(c_3, 2) = \{ r \} = c_3$

La nueva función obtenida es representada en la Tabla 4.11 y el grafo de este AFD equivalente es representado en la Figura 4.16.

f:	0	1	2
→*c_0	c_1	c_2	c_3
*c_1	c_1	c_2	c_3
*c_2		c_2	c_3
*c_3			c_3

Tabla 4.11: Función de transición del AFD equivalente.

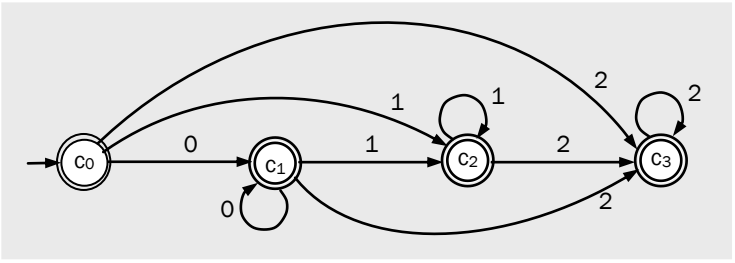


Figura 4.16: Grafo del AFD equivalente.

d) Determinación del AFD equivalente sobre la tabla de transición:

Siguiendo el procedimiento ya explicado con anterioridad, se puede trabajar sobre la tabla de la función de transición del AFND (sin λ) para obtener la tabla de la función de transición del AFD equivalente, tal como se muestra en la Tabla 4.12. La zona rayada corresponde a un estado inaccesible que debe ser eliminado para obtener un AFD conexo. Solo faltaría renombrar sus estados a **c_0**, **c_1**, **c_2** y **c_3** en reemplazo de las designaciones **p**, **pqr**, **qr** y **r**.

f:	0	1	2
→ p	{p, q, r}	{q, r}	{r}
q		{q, r}	{r}
*r			{r}

f':	0	1	2
→ p	pqr	qr	r
q		qr	r
*r			r
*pqr	pqr	qr	r
*qr		qr	r

Tabla 4.12: Función de transición del AFND y AFD equivalente.

e) Verificación de la aceptación de la cadena **β = 001122**:

Usando el árbol de descripciones instantáneas se representan, en la Figura 4.17, el comportamiento de los AFD (sin **λ**) y AFD equivalente al leerse la cadena indicada. Luego, en la Figura 4.18, se representa los planos estados-entradas de ambos autómatas.

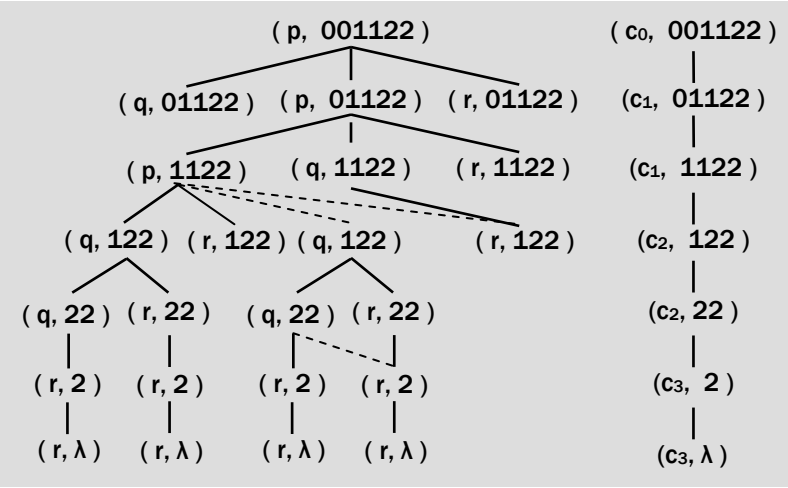


Figura 4.17: Árboles de descripciones instantáneas del AFND y AFD.

Tal como ocurrió en el Ejemplo 4.5, se presenta una circunstancia a destacar que es la presencia de descripciones instantáneas de convergencia, en las que vuelven a reunirse diferentes caminos en el proceso del reconocimiento de cadenas. Ésta es una evidencia de que, en algunos casos, el proceso de aceptación de cadenas admite más de un camino para una misma configuración final y que, los hasta aquí llamados *árboles* de configuraciones podrían dibujarse en realidad como grafos (ver las líneas punteadas en la Figura 4.17).

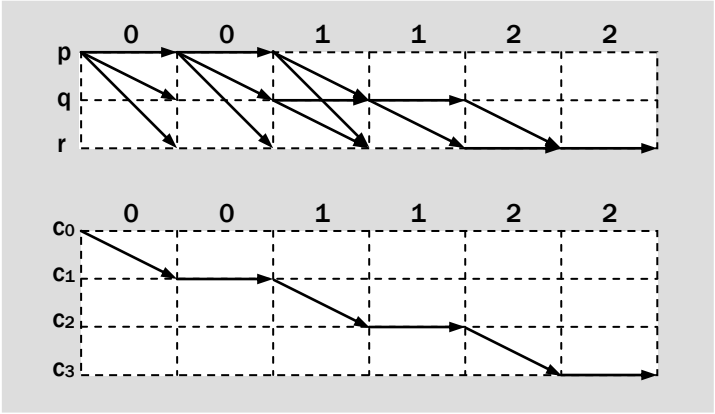


Figura 4.18: Representación del plano estados-entradas del AFND y AFD.

Gramáticas regulares y autómatas finitos

Una vez que se ha visto que todo AFND puede ser llevado a la forma de AFD equivalente, el siguiente paso es estudiar el vínculo entre los AFD y las gramáticas regulares. En el Capítulo 1, ya se anticipó que Chomsky y Miller (1958) establecieron esta correspondencia y que es reconocida como un isomorfismo entre ambos.

Hasta ahora los autómatas finitos reconocedores fueron construidos en forma intuitiva a partir de las formas generales de las cadenas de los lenguajes a ser tratados, lo que implica un proceso de prueba y error. Se propone en su reemplazo un procedimiento sistemático que facilite esta tarea, que permita definir los AF a partir de las reglas de producción de las gramáticas, que tendrá enorme importancia en correctores de texto y otras muchas aplicaciones. En sentido inverso, también interesa definir las reglas de producción capaces de generar lenguajes a ser reconocidos por ciertos AFD. Ambos procedimientos son presentados a continuación.

Definición de gramáticas regulares a partir de autómatas

Se comienza por el caso en el que se quiere definir el conjunto de reglas de producción de una gramática regular que generará lenguajes que están destinados a ser reconocidos por un cierto AFD. Para el procedimiento propuesto, el AF debe ser determinista y la gramática a ser obtenida estará bien formada y será lineal por derecha.

Dado un **AFD=($\sum$, Q, q_0, A, *f*)**, la gramática que genera el mismo lenguaje que acepta el autómata, será **G=(Σ, Q, q_0, P)**, donde el conjunto de reglas de reescritura queda definido como:

$$P = \{X := aY \,/\, f(X, a) = Y\} \cup \{X := a \,/\, f(X, a) = Y,\ Y \in A\}$$

y se agrega $q_0 := \lambda$ si $q_0 \in A$; $X, Y, q_0 \in Q$, $a \in \Sigma$, $A \subseteq Q$.

Estas gramáticas, donde sus reglas de producción se corresponden directamente con las transiciones de un AFD, son siempre no ambiguas. Cabe el interrogante: ¿son todas las gramáticas regulares no ambiguas?

Definición de autómatas a partir de gramáticas regulares

El caso inverso es aquel en que se conoce cierta gramática y se desea determinar el AFD que será necesario para reconocer o validar las sentencias del lenguaje generado. Como en el caso anterior, el procedimiento propuesto se refiere a gramáticas bien formadas y cuyas reglas de producción sean lineales por derecha.

Dada la gramática regular **$G=(\Sigma_T, \Sigma_N, S, P)$**, el autómata finito que acepta el mismo lenguaje que genera la gramática será:

$$AF = (\Sigma_T, \Sigma_N \cup \{A\}, S, \{A\}, f)$$

donde **A** es un nuevo símbolo que no estaba en Σ_N y ***f*(X, a)=Y** si **X:=aY** está en **P** y ***f*(X, a)=A** si **X:=a** está en **P**, con **$X,Y,S \in \Sigma_N$**, **$a \in \Sigma_T$**.

La regla de producción **$S:=\lambda$** incorpora la transición ***f*(S,λ)=A**.

Deben considerarse que: *i*) la existencia en la gramática de una regla de producción **$S:=\lambda$** convierte al autómata asociado en AFND-λ y *ii*) para el caso de que la gramática sea lineal por izquierda será necesario convertirla previamente a lineal por derecha, aplicando el procedimiento que es definido a continuación.

Conversión de gramática lineal por Izquierda a lineal por derecha

Para cada gramática lineal por derecha, existe una gramática lineal por izquierda equivalente y viceversa; como se recordará, dos gramáticas son equivalentes si generan el mismo lenguaje. En este caso, interesa convertir gramáticas lineales por izquierda en lineales por derecha y el procedimiento para convertir una en la otra tiene cuatro pasos:

Paso 1: Convertir la gramática dada a otra equivalente que no sea recursiva en el axioma. Para ello, se debe incorporar un nuevo símbolo no terminal **B** y modificar las reglas de producción según se describe:

- Por cada regla **S:=α**, se crea una nueva regla **B:=α**.
- Cada regla de la forma **A:=Sa** debe ser reemplazada por **A:=Ba**, donde **S**, **A**, **B** $\in \Sigma_N$, **a** $\in \Sigma_T$ y **α** es una cadena válida.

Paso 2: Se debe construir un grafo con las siguientes instrucciones:

- Los nodos del grafo se identifican con los símbolos no terminales de la gramática y se incluye un nodo adicional identificado con λ.
- Por cada regla de la forma **A:=Ba**, se dibuja un arco desde el nodo **A** al nodo **B** que es identificado con la etiqueta **a**.
- Por cada regla de la forma **A:=a**, se incluye un arco desde el nodo **A** al nodo λ identificado con **a**.
- Para la regla **S:=**λ, se incorpora un arco sin etiqueta desde el nodo **S** al nodo λ.

Paso 3*:* Se construye un nuevo grafo a partir del grafo anterior según se indica:

- Se intercambian los identificadores de los nodos **S** y λ.
- Se cambia el sentido de todos los arcos.

Paso 4*:* A partir del nuevo grafo, se obtiene la gramática lineal por derecha siguiendo los siguientes pasos:

- Los alfabetos de símbolos terminales y no terminales son los mismos de la gramática original lineal por izquierda.
- Por cada arco dirigido etiquetado con **a** desde el nodo **B** al nodo **A**, se incorpora a la nueva gramática la regla de producción **B:=aA**.
- Por cada arco dirigido etiquetado con **a** desde el nodo **B** al nodo λ, se incorpora la regla de producción **B:=a**.
- En caso de un arco sin etiqueta desde el nodo **S** al nodo λ, se incorpora a la gramática la regla **S:=**λ.

A efectos de comprobar la efectividad de los procedimientos descriptos, se propondrá un AFD y se determinarán las reglas de producción de la gramática regular correspondiente, que será lineal por derecha y bien formada. Luego, se derivarán algunas sentencias del lenguaje. En una segunda etapa, se operará en sentido inverso, es decir, se definirá la función de transición del AFD que es necesario para reconocer el lenguaje que es generado por la gramática que contiene las reglas de producción obtenidas. Por último, se convertirá una gramática lineal por izquierda en lineal por derecha, todo lo cual es el motivo de los siguientes ejemplos.

Ejemplo 4.7

Determinar la gramática bien formada capaz de generar las sentencias del lenguaje reconocido por el AFD del Ejemplo 3.3 del Capítulo 3 y cuya definición formal se presenta en la Tabla 4.13.

$AF=(\Sigma_E, Q, q_o, A. f)$

Σ_E = {0, 1}

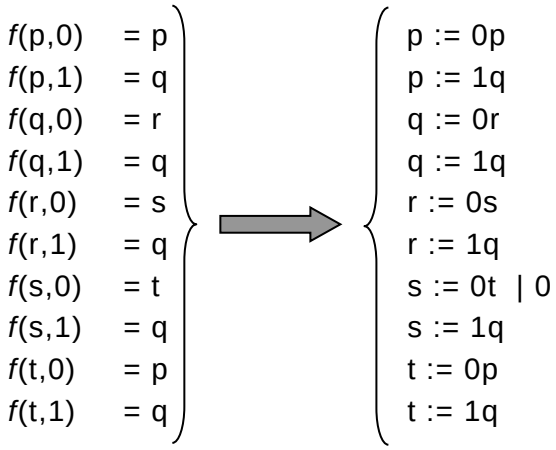
Q = {p, q, r, s, t}

q_o = p

***f*:**	0	1
→ p	p	q
q	r	q
r	s	q
s	t	q
* t	p	q

Tabla 4.13: Función ***f*** del AFD del Ejemplo 4.7.

Las transiciones son convertidas en reglas de producción según el procedimiento que fue descripto con anterioridad, tal como se lo detalla a continuación:



Nótese que hay una regla de producción de tipo **X:=a** a raíz de que el estado **t** es un estado de aceptación (**t∈A**). Luego, el conjunto completo de reglas de producción de la gramática es:

P = {p := 0p | 1q,
q := 0r | 1q,
r := 0s | 1q,
s := 0t | 1q | 0,
t := 0p | 1q }

Para completar el estudio, se comprueba que con estas reglas de producción pueden derivarse algunas cadenas tales como las siguientes:

$p \rightarrow 0p \rightarrow 01q \rightarrow 010r \rightarrow 0100s \rightarrow 01000$

$p \rightarrow 0p \rightarrow 01q \rightarrow 011q \rightarrow 0111q \rightarrow 01110r \rightarrow 011100s \rightarrow 0111000$

$p \rightarrow 1q \rightarrow 10r \rightarrow 101q \rightarrow 1011q \rightarrow 10110r \rightarrow 101100s \rightarrow 1011000$

y que todas ellas responden a la forma general **$(0+1)^*1000$** que son las reconocidas por el AFD original.

Ejemplo 4.8

Se desea definir un AFD destinado a reconocer las sentencias que puedan ser derivadas con la siguiente gramática lineal por derecha:

G = ({a, b}, {p, q, r, s, t}, p, P)

donde P = {p := 0p | 1q,
q := 0r | 1q,
r := 0s | 1q,
s := 0t | 1q | 0,
t := 0p | 1q }

Según el procedimiento indicado para las gramáticas bien formadas, a cada producción **X:=aY** le corresponde una transición ***f*(X, a) = Y**. Luego, a partir del conjunto de reglas de producción de la gramática anterior, se obtienen las siguientes transiciones:

f(p,0) = p *f*(p,1) = q
f(q,0) = r *f*(q,1) = q
f(r,0) = s *f*(r,1) = q
f(s,0) = t *f*(s,1) = q
f(t,0) = p *f*(t,1) = q

Además, resulta **p** ser el estado inicial (**q_0 = p**), por ser **p** el axioma de la gramática. El conjunto de estados de aceptación estará compuesto por un único estado **A** y la regla de producción **s:=0** indica que debe agregarse la transición ***f*(s, 0)=A**. Esto hace al autómata no determinista

(desde **s** con **0** puede transitar a **t** y **A**). Los estados **t** y **A** se unifican al transformar el AF en determinista.

Con la información obtenida, que incluye las transiciones, el estado inicial y el estado de aceptación, puede ahora elaborarse el grafo del AFD. Este grafo es mostrado en la Figura 4.19 y corresponde a la definición formal del AFD presentada en el Ejemplo 4.7.

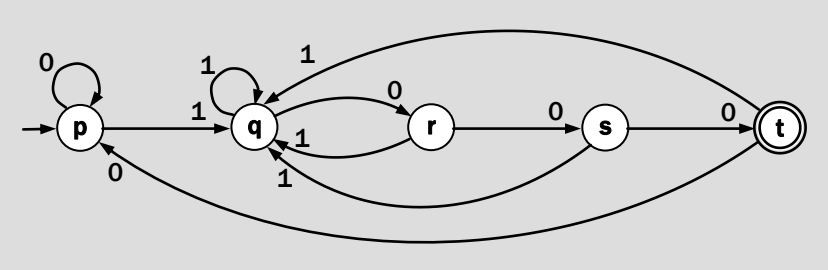


Figura 4.19: Grafo del AFD del Ejemplo 4.8

Con estos dos ejemplos, se ilustró la correspondencia entre las gramáticas regulares (lineales) y los autómatas finitos deterministas.

Ejemplo 4.9

Dada la gramática lineal por izquierda $\mathbf{G_1}$ que se presenta a continuación se necesita convertirla a lineal por derecha con el fin de poder definir posteriormente el AFD capaz de reconocer su lenguaje. Con ambas gramáticas, mostrar la derivación de una misma sentencia.

$G_1 = (\{0, 1, 2\}, \{X, Y, Z\}, X, P)$

P = {X := Z0 | Y0 | 0 | 1, Y := Y0 | 1, Z := Z2 | 2}

Según el procedimiento descripto para convertir una gramática lineal por izquierda a lineal por derecha, se debe comenzar por verificar que la gramática no sea recursiva en el axioma, lo que claramente no ocurre en este caso. El siguiente paso es representar las reglas de producción en un grafo, que se muestra en la Figura 4.20.

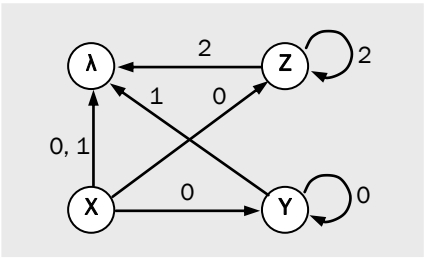


Figura 4.20: Grafo de la gramática lineal por izquierda $\mathbf{G_1}$.

A continuación, se trabaja sobre el grafo (paso 3), intercambiando los nodos del axioma y de **λ** y cambiando el sentido de todos los arcos. Se obtiene el nuevo grafo que corresponde a una gramática lineal por derecha equivalente a la anterior, que es mostrado en la Figura 4.21. Es muy importante advertir que estos grafos no representan autómatas.

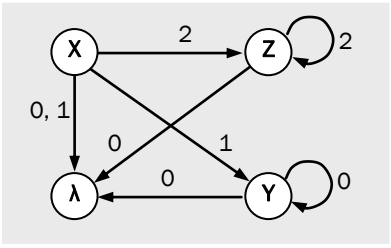


Figura 4.21: Grafo de la gramática lineal por derecha $\mathbf{G_2}$

El último paso consiste en reconocer las nuevas reglas de producción a partir de la interpretación del nuevo grafo de la Figura 4.21. Se obtiene así la gramática lineal por derecha **G_2** que es presentada a continuación:

$G_2 = (\{0, 1, 2\}, \{X, Y, Z\}, X, P)$

$P = \{X := 2Z \mid 1Y \mid 0 \mid 1,\ Y := 0Y \mid 0,\ Z := 2Z \mid 0\}$

Los ejemplos de una misma sentencia derivada con ambas gramáticas son los siguientes:

G_1: $X \rightarrow Y0 \rightarrow Y00 \rightarrow Y000 \rightarrow 1000$
G_2: $X \rightarrow 1Y \rightarrow 10Y \rightarrow 100Y \rightarrow 1000$

Ejemplo 4.10

Dada una máquina abstracta que es representada, en la Figura 4.22, se pide: a) por observación identificar y describir algebraicamente el lenguaje que es reconocido por la máquina, b) mediante un árbol de descripciones instantáneas (o árbol de configuraciones) mostrar el reconocimiento de una de las palabras del lenguaje de largo cinco, c) identificar de qué tipo de máquina se trata, justificarlo y describirla formalmente, d) en caso de ser necesario, presentar la máquina determinista equivalente, incluyendo la representación de su grafo y descripción formal, e) verificar si la máquina determinista equivalente puede ser minimizada y f) encontrar la gramática asociada al autómata estudiado y comprobar que se corresponde con el lenguaje identificado en el punto a).

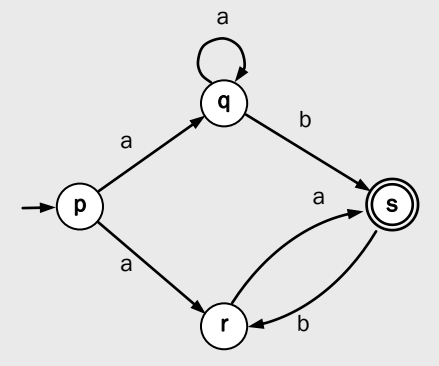


Figura 4.22: Grafo de la máquina abstracta estudiada.

a. Lenguaje reconocido por la máquina

A partir de la observación de la máquina, se comprueba que puede reconocer dos tipos diferentes de sentencias, de manera que el lenguaje puede ser descripto algebraicamente como:

$$L = \{ a^m b(ba)^n \mid m > 0, n \geq 0 \} \cup \{ a^2(ba)^n \mid n \geq 0 \}$$

b. Reconocimiento de una palabra

Se comprueba la aceptación de una sentencia del lenguaje de largo cinco, tal como **α = aabba** del formato general **$a^m b(ba)^n$**:

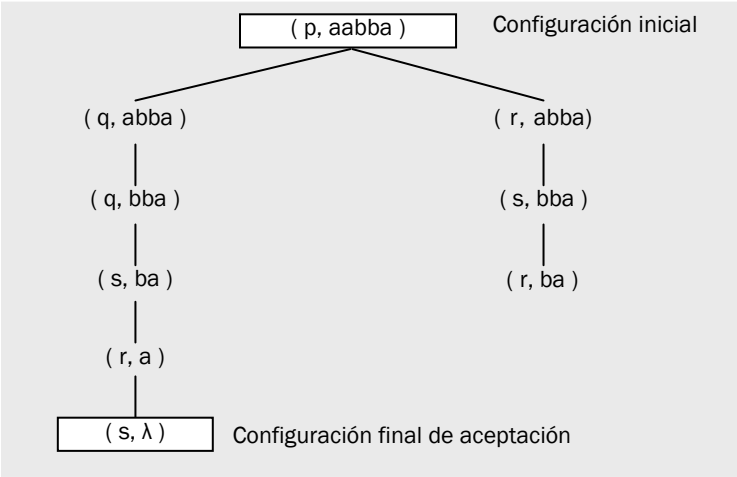


Figura 4.23: Árbol de descripciones instantáneas para la cadena **α**

c. Identificación del tipo de máquina

Se trata de un *autómata finito* por disponer de un estado inicial **p** y un estado de aceptación **s**. Además, es un *autómata finito no determinista* debido a las dos transiciones que tiene desde el estado **p** para el mismo símbolo de entrada **a**, lo que conduce a que el árbol de descripciones instantáneas de la Figura 4.23 presente una ramificación.

Descripción formal:

$$AFND = (\{ a, b \}, \{ p, q, r, s \}, p, \{ s \}, f)$$

donde ***f*** está representada en la siguiente tabla:

f:	a	b
→ p	{q, r}	
q	{q}	{s}
r	{s}	
*s		{r}

Tabla 4.14: Función de transición de la máquina estudiada.

d. Máquina determinista equivalente

Por tratarse de un AFND, se debe determinar el autómata finito determinista equivalente por alguno de los procedimientos ya estudiados, el cual queda definido como:

$$AFD = (\{ a, b \}, \{ p, q, r, s, qr, qs, rs \}, p, \{ s, qs, rs \}, f)$$

Por razones de claridad, resulta conveniente redesignar los nombres de los estados y, para facilitar su identificación, se los enumera en el mismo orden en el que están en la definición anterior, por ejemplo:

$$AFD = (\{ a, b \}, \{ c_0, c_1, c_2, c_3, c_4, c_5, c_6 \}, c_0, \{ c_3, c_5, c_6 \}, f)$$

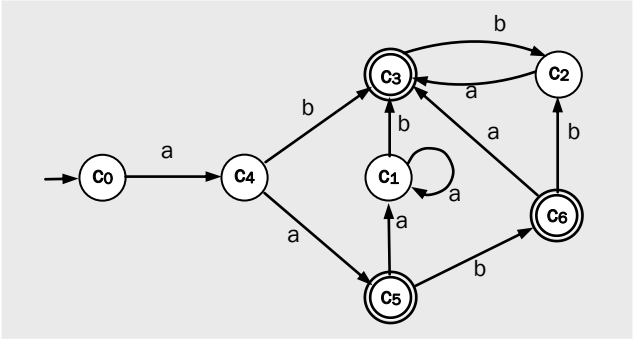


Figura 4.24: Grafo del AFD equivalente a la máquina estudiada.

La función de transición del AFD se muestra en la Tabla 4.15.

e. Minimización

Es importante minimizar el AFD para verificar la eventual presencia de estados equivalentes o indistinguibles. Siguiendo el procedimiento ya estudiado en el Capítulo 3, se comprueba que no hay posibilidad de encontrar estados equivalentes, lo que indica que la máquina ya es mínima:

$$Q/E = \{\{ c_0 \}, \{ c_1 \}, \{ c_2 \}, \{ c_3 \}, \{ c_4 \}, \{ c_5 \}, \{ c_6 \}\}$$

f:	a	b
→ c_0	c_4	
c_1	c_1	c_3
c_2	c_3	
* c_3		c_2
c_4	c_5	c_3
* c_5	c_1	c_6
* c_6	c_3	c_2

Tabla 4.15: Función de transición de la máquina estudiada.

f. Gramática asociada a la máquina abstracta

$$G = (\{ a, b \}, \{ c_0, c_1, c_2, c_3, c_4, c_5, c_6 \}, c_0, P)$$

Se determina las reglas de producción de la gramática a partir de las transiciones del *AFD*, obteniéndose:

$P = \{ c_0 := a\ c_4,\ \ c_1 := a\ c_1 \mid b\ c_3 \mid b,\ \ c_2 := a\ c_3 \mid a,\ \ \ c_3 := b\ c_2,$

$c_4 := a\ c_5 \mid a \mid b\ c_3 \mid b,\ \ c_5 := a\ c_1 \mid b\ c_6 \mid b,\ \ c_6 := a\ c_3 \mid a \mid b\ c_2 \}$

Algunas derivaciones de sentencias para confirmar que la gramática anterior genera el lenguaje descripto en el punto uno son las siguientes:

Sentencias de la forma general $\mathbf{a^m b (ba)^n}$

$c_0 \rightarrow a\ c_4 \rightarrow aa\ c_5 \rightarrow aaa\ c_1 \rightarrow aaab\ c_3 \rightarrow aaabb\ c_2 \rightarrow aaabba$

$c_0 \rightarrow a\ c_4 \rightarrow aa\ c_5 \rightarrow aaa\ c_1 \rightarrow aaab\ c_3 \rightarrow aaabb\ c_2 \rightarrow aaabba\ c_3 \rightarrow$

$\rightarrow aaabbab\ c_2 \rightarrow aaabbaba$

Sentencias de la forma general $\mathbf{a^2 (ba)^n}$

$c_0 \rightarrow a\ c_4 \rightarrow aa\ c_5 \rightarrow aab\ c_6 \rightarrow aaba$

$c_0 \rightarrow a\ c_4 \rightarrow aa\ c_5 \rightarrow aab\ c_6 \rightarrow aaba\ c_3 \rightarrow aabab\ c_2 \rightarrow aababa$

Ejemplo 4.11

En el Ejemplo 3.9 del Capítulo 3, se desarrolló un analizador léxico como una de las principales aplicaciones de los autómatas finitos. En el caso de este ejemplo propuesto con anterioridad, el objetivo del AFD fue identificar secuencias de caracteres que corresponden a valores numéricos, como los que se presentan en los códigos de los programas, en ese caso Java, o en los documentos elaborados con editores de texto. El AFD fue desarrollado por inspección de las secuencias de caracteres a ser identificadas. Llegado a este punto se recomienda al lector remitirse al Apéndice A e interiorizarse sobre la estructura de un compilador y en particular la importante función denominada análisis léxico.

Ahora, habiéndose visto en este Capítulo 4, la definición de autómatas finitos a partir de gramáticas regulares, se propone definir una gramática capaz de expresar valores numéricos y deducir a partir de ella la máquina de estados necesaria para su reconocimiento. Se considerarán valores positivos y negativos, enteros y reales, y que estos últimas puedan estar expresadas en notación científica.

La gramática regular que cumple con lo especificado es obtenida sin mayores dificultades a partir del análisis de las distintas cadenas a ser generadas. Se tiene así:

Σ_T = {0, n, m, +, -, **.** , E, b} donde **E** define el exponente en la notación científica, **n** números enteros en el intervalo 1 a 9, **m** números enteros en el intervalo 0 a 9 y **b** los símbolos terminales y separadores (<~~b~~> y ;).

Σ_N = {S, T, U, V, W, X, Y} donde **S** es el axioma de la gramática.

P = { S := 0X | +W | -W | nU,
W := nU | 0X,
X := **.**Y | b,
U := mU | .Y | b,
Y := mY | ET,
T := +V | -V,
V := mV | b}

Con facilidad, se comprueba que con esta gramática pueden derivarse los números buscados, tales como:

S → -W → -2U → -27U → -277U → -277b
S → 3U → 3.Y → 3.1Y → 3.14Y → 3.141Y → 3.1416Y → 3.1416b
S → 0X → 0.Y → 0.8Y → 0.87Y → 0.87ET → 0.87E-V → 0.87E-5V → 0.87E-5b

Siguiendo el procedimiento ya descripto con anterioridad, se convierten las reglas de producción en transiciones. Para ello, se incorpora un nuevo estado final Z y se tiene:

S := 0X	*f*(S, 0) := X
S := +W	*f*(S,+) := W
S := -W	*f*(S, -) := W
S := nU	*f*(S, n) := U
W := nU	*f*(W, n) := U
W := 0X	*f*(W,0) := X
X := **.**Y	*f*(X, .) := Y
X := b	*f*(X, b) := Z
U := mU	*f*(U, m) := U
U := .Y	*f*(U, .) := Y
U := b,	*f*(U, b) := Z
Y := mY	*f*(Y, m) := Y
Y := ET	*f*(Y, E) := T
T := +V	*f*(T,+) := V
T := -V	*f*(T, -) := V
V := mV	*f*(V, m) := V
V := b	*f*(V, b):= Z

Una vez identificadas las transiciones, el siguiente paso es definir formalmente el autómata finito y representar su grafo, que se muestran a continuación.

a. Definición formal del analizador léxico

AF = (Σ_E, Q, q_o, A, *f*)

donde:

Σ_E = {0, n, m, +, -, **.** , E, b}

Q = {S, T, U, V, W, X, Y, Z}

q_o = S

A = { Z }

f	0	n	m	+	-	.	E	b
→**S**	X	U		W	W			
T				V	V			
U			U			Y		Z
V			V					Z
W	X	U						
X						Y		Z
Y			Y				T	
***Z**								

Tabla 4.16: Función *f* del analizador léxico.

b. Grafo del analizador léxico

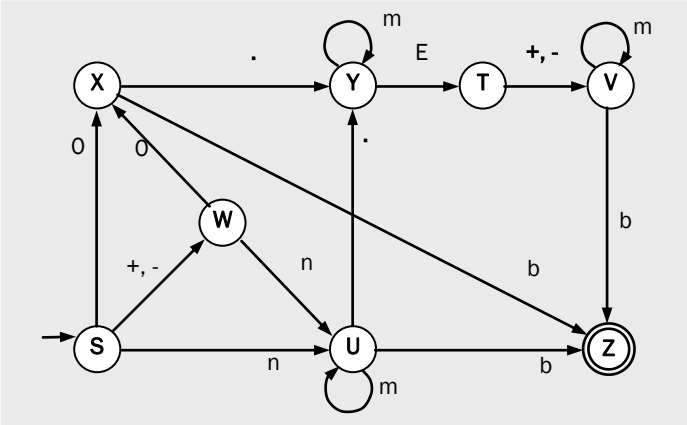


Figura 4.25: Grafo del analizador léxico del Ejemplo 4.11.

Se obtuvo así un analizador léxico capaz de reconocer cadenas que representan números enteros y reales, que es similar al del Ejemplo 3.9, solo que aquél fue obtenido por inspección de las cadenas a reconocer y éste en forma sistemática a partir de la gramática que genera el lenguaje de estos números. Nótese que, tal como está planteado, el analizador reconoce los números correctos pero no distingue enteros de reales. Además, como puede observarse, este analizador solo reconoce números de base decimal, mientras que el propuesto, en el Capítulo 3, también reconocía números octales y hexadecimales. Se deja al lector la tarea de completar la gramática y luego definir un autómata que incorpore la capacidad de reconocimiento de estos números.

Expresiones regulares y autómatas finitos

Al estudiar lenguajes regulares, en el Capítulo 2, se presentó una notación para especificar lenguajes regulares de uso usual en las rutinas de búsqueda de patrones, tanto en los utilitarios de los sistemas operativos (*ed*, *vi*, *lex*, *grep* entre otros) como en funciones de librería de los lenguajes de programación modernos (*regexp*): las *Expresiones Regulares* (ER).

Stephen Kleene, lógico-matemático norteamericano, estudió con profundidad en la década del cincuenta los conjuntos regulares y sus propiedades; dos resultados de su trabajo son los llamados *teorema de análisis* y *de síntesis*, con los cuales se establece y demuestra que un lenguaje es regular si y solo si es aceptado por un autómata finito. Para ello, Kleene desarrolló la operación estrella ya estudiada y la notación que llamamos expresiones regulares.

La comprobación de estos teoremas requiere conceptos que exceden el objetivo de este texto (ecuaciones características de un AF, derivadas de una ER y otros). Sin embargo, por su practicidad y sencillez, se avanzará aquí sobre un algoritmo debido a Ken Thompson y Denis Ritchie que permite construir (automáticamente) para cada expresión regular, un autómata finito no determinista con transiciones λ que reconozca el mismo lenguaje descripto por ella.

Método o Algoritmo de Thompson

El método que se presenta, traduce cada parte de la definición de una expresión regular en un autómata que acepta el mismo lenguaje denotado por ella, de tal forma que el autómata finito quede definido al analizar la expresión parte por parte.

Recordemos que la definición de una expresión regular tiene una parte básica ($\varnothing$, λ, **a**, siendo **a** cualquier símbolo del alfabeto) y una parte recursiva (**E+F**, **E**$\circ$**F**, **E***, **(E)**, siendo **E** y **F** expresiones regulares).

En los siguientes puntos, se trabajará siempre suponiendo que las expresiones regulares están describiendo un lenguaje sobre algún alfabeto Σ y, en cada caso, se construirá un autómata finito AF = (Σ, Q, q_0, A, *f*) mostrando su grafo dirigido.

a) Para la expresión regular $\varnothing$ que denota al lenguaje vacío, se construye un autómata finito que reconozca este lenguaje. Esto puede hacerse de dos formas equivalentes: construyendo un autómata que tenga el conjunto **A** de estados de aceptación vacío, o que tenga un único estado de aceptación aislado; se procede de la segunda forma:

Figura 4.26.a: AF que reconoce $\varnothing$.

b) La expresión regular λ denota al lenguaje cuya única cadena es la vacía. Nuevamente, tenemos dos modos de construir el AF: haciendo que su estado inicial sea a la vez de aceptación, o equivalentemente con una transición λ como único acceso a un estado de aceptación desde su estado inicial:

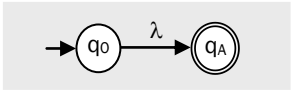


Figura 4.26.b: AF que reconoce {λ}.

c) Para cada símbolo $\mathbf{a}\in\Sigma$, la expresión regular **a** denota al lenguaje **{a}**, y el autómata finito correspondiente es el mostrado en la siguiente figura:

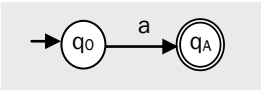


Figura 4.26.c: AF que reconoce {a}.

Sean ahora **E** y **F** dos expresiones regulares, para las cuales ya se cuenta con sendos autómatas finitos $\mathbf{AF_E}$ y $\mathbf{AF_F}$, que reconocen respectivamente a los lenguajes **L(E)** y **L(F)** denotados por **E** y por **F**. Entonces:

d) El lenguaje unión **L(E)** y **L(F)** denotado por la expresión regular **E+F**, será reconocido por el autómata construido de la siguiente forma:

- Quitar la característica de inicial de los estados iniciales de $\mathbf{AF_E}$ y $\mathbf{AF_F}$.
- Crear un nuevo estado inicial $\mathbf{q_0}$ y relacionarlo mediante transiciones λ con los antiguos estados iniciales de $\mathbf{AF_E}$ y $\mathbf{AF_F}$.
- Quitar la característica de aceptación, de todos los estados de aceptación de $\mathbf{AF_E}$ y $\mathbf{AF_F}$.
- Crear un nuevo estado de aceptación $\mathbf{q_A}$ y agregar desde cada antiguo estado de aceptación de $\mathbf{AF_E}$ y $\mathbf{AF_F}$, una transición λ hacia él.

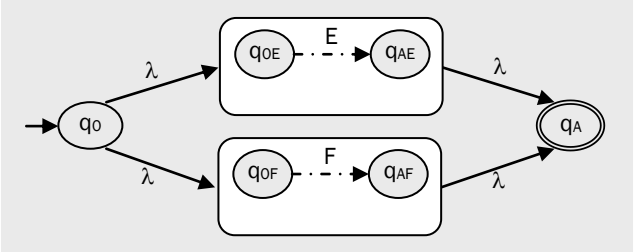


Figura 4.26.d: AF que reconoce L(E+F).

El autómata finito no determinista construido de esta forma, *conecta en paralelo* los autómatas ya existentes para **E** y **F** utilizando transiciones λ, logrando aceptar todas las cadenas que ambos aceptaban, es decir, la unión de sus lenguajes.

e) La expresión regular **E◦F** (o sencillamente **EF**), que denota al lenguaje concatenación de **L(E)** y **L(F)**, se reconocerá por el autómata construido de la siguiente forma:

- Quitar la característica de inicial de los estados iniciales de $\mathbf{AF_E}$ y $\mathbf{AF_F}$.
- Quitar la característica de aceptación, de todos los estados de aceptación de ambos autómatas, $\mathbf{AF_E}$ y $\mathbf{AF_F}$.
- Crear un nuevo estado inicial $\mathbf{q_0}$ y relacionarlo mediante una transición λ con el antiguo estado inicial de $\mathbf{AF_E}$.
- Agregar una transición λ desde cada antiguo estado de aceptación de $\mathbf{AF_E}$ hacia el antiguo estado inicial de $\mathbf{AF_F}$.
- Crear un nuevo estado de aceptación $\mathbf{q_A}$ y agregar desde cada antiguo estado de aceptación de $\mathbf{AF_F}$, una transición λ hacia él.

  El autómata finito no determinista construido, *conecta en serie* los autómatas que se tenían para **E** y **F,** utilizando transiciones λ.

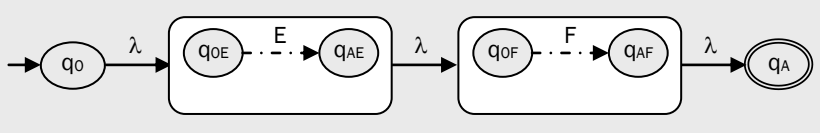


Figura 4.26.e: AF que reconoce L(EF).

Cualquier cadena que tenga un prefijo descripto por **E**, hará que el nuevo AF transite desde su estado inicial hasta alguno de los antiguos estados finales de $\mathbf{AF_E}$ y desde allí espontáneamente hasta el antiguo estado inicial de $\mathbf{AF_F}$ (por la transición λ); si la cadena tiene luego un sufijo que sigue el patrón **F**, entonces hará que el autómata arribe a alguno de los antiguos estados de aceptación de $\mathbf{AF_F}$ y desde allí pueda pasar sin leer entradas hasta el nuevo estado de aceptación $\mathbf{q_A}$. Esto acepta entonces, cadenas que sigan el patrón **EF**.

f) Finalmente, para la estrella de Kleene de una expresión regular $\mathbf{E^*}$, que representa al lenguaje $\mathbf{[L(E)]^*}$, se construye el siguiente autómata finito:

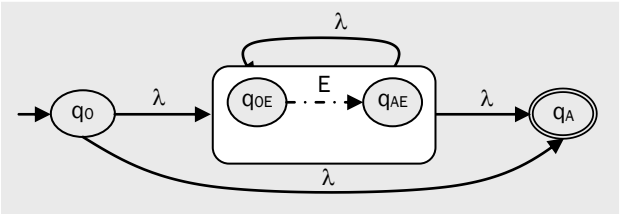


Figura 4.26.f: AF que reconoce L(E*).

- Quitar la característica de inicial del estado inicial y la denominación de aceptación, de cada estado de aceptación, del autómata que reconoce cadenas representadas por la expresión regular **E**.
- Crear un nuevo estado inicial $\mathbf{q_0}$ y agregar una transición λ desde el mismo hacia el antiguo estado inicial de $\mathbf{AF_E}$.
- Crear un nuevo estado de aceptación $\mathbf{q_A}$ y agregar transiciones λ hacia él desde el estado inicial $\mathbf{q_0}$ (para aceptar la cadena vacía) y desde cada antiguo estado de aceptación del autómata $\mathbf{AF_E}$ correspondiente a la expresión regular **E**.
- Terminar el proceso creando una transición λ desde cada antiguo estado de aceptación de $\mathbf{AF_E}$ hacia el antiguo estado inicial del mismo, lo cual sirve para reconocer tan-

tas concatenaciones consigo misma de cadenas con el patrón **E** como se quiera, logrando aceptar el lenguaje deseado.

Para la expresión regular **(E)**, que denota al mismo lenguaje descripto por **E**, no se requiere construir un nuevo autómata ya que es el mismo $\mathbf{AF_E}$.

El método de Thompson termina construyendo un AFND -λ con muchos estados, por lo cual debe ser luego transformado en determinista (y posiblemente minimizado) para su implementación eficiente en programación, pero su importancia radica como se indicó, en que es un procedimiento automático que puede ser programado (es lo que hace el generador de analizadores léxicos *lex* de UNIX).

Ejemplo 4.12

Dada la expresión regular $\mathbf{(0+1)^*1000}$ ya utilizada en los Ejemplos 4.1 y 4.3, que denota al conjunto de cadenas binarias con sufijo **1000**, aplicaremos el método de Thompson para determinar un autómata finito que reconozca el lenguaje.

La precedencia natural de los operadores involucrados, estipula que primero debe efectuarse la unión indicada por **+** (por los paréntesis), luego determinar la estrella de Kleene del conjunto resultante y concatenar este resultado sucesivamente con **1**, **0**, **0** y **0**; esta concatenación podrá hacerse en cualquier orden ya que la concatenación de palabras es asociativa.

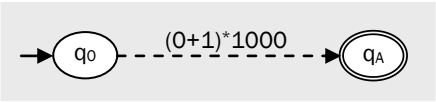


Figura 4.27.a: Esquema de AF que reconoce cadenas con el patrón $(0+1)^*1000$.

Iniciamos entonces el análisis de la expresión entre paréntesis **0+1**. Vemos que se tienen que generar autómatas básicos para reconocer la cadena **0** y la cadena **1**, para luego conectar en paralelo estos autómatas logrando reconocer ambas cadenas.

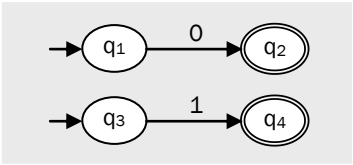


Figura 4.27.b: AF que reconocen L(0)={0} y L(1)={1}.

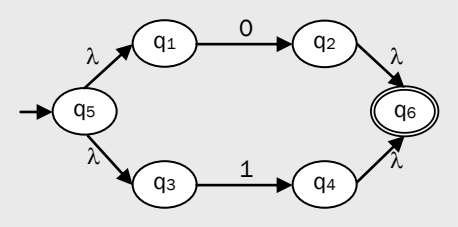


Figura 4.27.c: AF que reconoce L(0+1)={0, 1}.

Ahora se debe construir el autómata para la estrella de Kleene de {0, 1}. Para ello, se deben crear un nuevo estado inicial $\mathbf{q_7}$, un nuevo estado de aceptación $\mathbf{q_8}$ y agregar las transiciones λ correspondientes, que los conecten y que representen la aceptación de la cadena vacía (transición λ inferior en la Figura 4.27.d) y de las concatenaciones posibles de las cadenas consigo mismas (transición λ superior en la Figura 4.27.d).

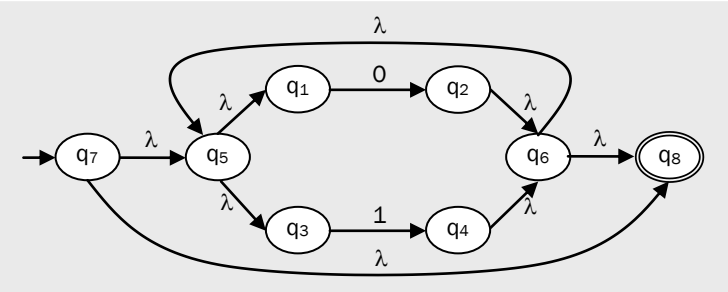


Figura 4.27.d: AF que reconoce L((0+1)*)={0, 1}*.

Para completar el autómata, que hasta ahora reconoce los prefijos de las cadenas buscadas, se debe concatenar con uno que reconozca los sufijos **1000**. Si bien no son necesarias las transiciones λ indicadas por el algoritmo de Thompson para esta tarea, se incluirán todas para respetar el procedimiento descripto, y sus correspondientes nuevos estados, en los siguientes cuatro pasos.

Se usarán en la Figura 4.27.e esquemas de AF (con expresiones regulares como etiquetas de algún arco dirigido) por razones de espacio, para los autómatas de expresiones ya construidos en pasos anteriores.

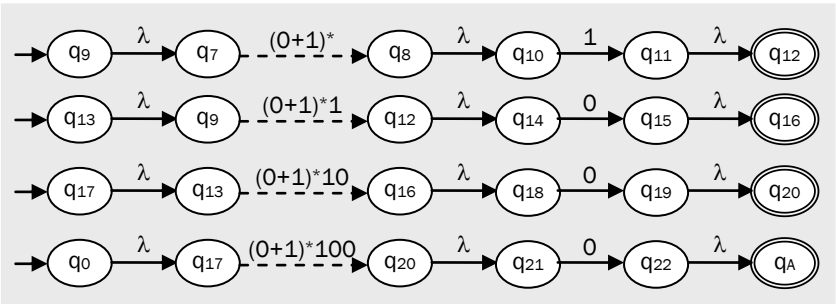


Figura 4.27.e: Esquemas de AF que reconocen sucesivamente {0, 1}*1, {0, 1}*10, {0,1}*100 y {0,1}*1000.

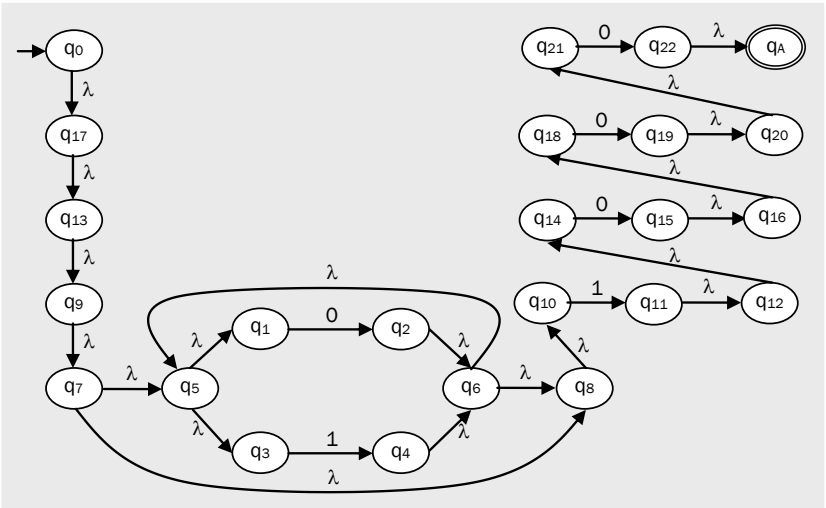


Figura 4.28: AF que reconoce el lenguaje L((0+1)*1000).

Tabla 4.17: Función de transición del Ejemplo 4.12, con las transiciones λ reflexivas y transitivas incorporadas.

f:	0	1	λ
→q_0			{q_0, q_1, q_3, q_5, q_7, q_8, q_9, q_{10}, q_{13}, q_{17}}
q_1	{q_2}		{q_1}
q_2			{q_1, q_2, q_3, q_5, q_6, q_8, q_{10}}
q_3		{q_4}	{q_3}
q_4			{q_1, q_3, q_4, q_5, q_6, q_8, q_{10}}
q_5			{q_1, q_3, q_5}
q_6			{q_1, q_3, q_5, q_6, q_8, q_{10}}
q_7			{q_1, q_3, q_5, q_7, q_8, q_{10}}
q_8			{q_8, q_{10}}
q_9			{q_1, q_3, q_5, q_7, q_8, q_9, q_{10}}
q_{10}		{q_{11}}	{q_{10}}
q_{11}			{q_{11}, q_{12}, q_{14}}
q_{12}			{q_{12}, q_{14}}
q_{13}			{q_1, q_3, q_5, q_7, q_8, q_9, q_{10}, q_{13}}
q_{14}	{q_{15}}		{q_{14}}
q_{15}			{q_{15}, q_{16}, q_{18}}
q_{16}			{q_{16}, q_{18}}
q_{17}			{q_1, q_3, q_5, q_7, q_8, q_9, q_{10}, q_{13}, q_{17}}
q_{18}	{q_{19}}		{q_{18}}
q_{19}			{q_{19}, q_{20}, q_{21}}
q_{20}			{q_{20}, q_{21}}
q_{21}	{q_{22}}		{q_{21}}
q_{22}			{q_{22}, q_A}
*q_A			{q_A}

El autómata finito no determinista con transiciones λ final, contiene entonces 24 estados y reconoce cualquier patrón denotado por la expresión regular **(0+1)*1000**. Para poder hacer alguna comparación con el autómata diseñado en el Ejercicio 4.3, procederemos a obtener un autómata finito determinista.

Se define primero formalmente el autómata no determinista como:

$$\text{AFND-}\lambda = (\{0, 1\}, \{q_0, q_1, q_2, \ldots, q_{21}, q_{22}, q_A\}, q_0, \{q_A\}, f)$$

La función de transición ***f*** estará representada por el grafo del último esquema de la Figura 4.27.e. El grafo completo se muestra en la Figura 4.28 y la tabla, incluyendo las transiciones λ explícitas, reflexivas y transitivas (de T y T*) en la Tabla 4.17.

A partir de la Tabla 4.17, puede determinarse el AFD equivalente, mediante la aplicación del Teorema 2 (se usará el procedimiento del Caso 2):

a) Estado Inicial: $c_0 = f(q_0, \lambda) = \{q_0, q_1, q_3, q_5, q_7, q_8, q_9, q_{10}, q_{13}, q_{17}\}$.

b) Se determinan los siguientes estados:

$$f'(c_0,0) = \bigcup_{q_k \in c_0} f(q_k,0) = \{q_2\} \xrightarrow{\lambda} \{q_1,q_2,q_3,q_5,q_6,q_8,q_{10}\} = c_1$$

$$f'(c_0,1) = \bigcup_{q_k \in c_0} f(q_k,1) = \{q_4,q_{11}\} \xrightarrow{\lambda} \{q_1,q_3,q_4,q_5,q_6,q_8,q_{10},q_{11},q_{12},q_{14}\} = c_2$$

$$f'(c_1,0) = \bigcup_{q_k \in c_1} f(q_k,0) = \{q_2\} \xrightarrow{\lambda} \{q_1,q_2,q_3,q_5,q_6,q_8,q_{10}\} = c_1$$

$$f'(c_1,1)=\bigcup_{q_k\in c_1} f(q_k,1)=\{q_4,q_{11}\}\xrightarrow{\lambda}\{q_1,q_3,q_4,q_5,q_6,q_8,q_{10},q_{11},q_{12},q_{14}\}=c_2$$

$$f'(c_2,0)=\bigcup_{q_k\in c_2} f(q_k,0)=\{q_2,q_{15}\}\xrightarrow{\lambda}\{q_1,q_2,q_3,q_5,q_6,q_8,q_{10},q_{15},q_{16},q_{18}\}=c_3$$

$$f'(c_2,1)=\bigcup_{q_k\in c_2} f(q_k,1)=\{q_4,q_{11}\}\xrightarrow{\lambda}\{q_1,q_3,q_4,q_5,q_6,q_8,q_{10},q_{11},q_{12},q_{14}\}=c_2$$

$$f'(c_3,0)=\bigcup_{q_k\in c_3} f(q_k,0)=\{q_2,q_{19}\}\xrightarrow{\lambda}\{q_1,q_2,q_3,q_5,q_6,q_8,q_{10},q_{19},q_{20},q_{21}\}=c_4$$

$$f'(c_3,1)=\bigcup_{q_k\in c_3} f(q_k,1)=\{q_4,q_{11}\}\xrightarrow{\lambda}\{q_1,q_3,q_4,q_5,q_6,q_8,q_{10},q_{11},q_{12},q_{14}\}=c_2$$

$$f'(c_4,0)=\bigcup_{q_k\in c_4} f(q_k,0)=\{q_2,q_{22}\}\xrightarrow{\lambda}\{q_1,q_2,q_3,q_5,q_6,q_8,q_{10},q_{22},q_A\}=c_5$$

$$f'(c_4,1)=\bigcup_{q_k\in c_4} f(q_k,1)=\{q_4,q_{11}\}\xrightarrow{\lambda}\{q_1,q_3,q_4,q_5,q_6,q_8,q_{10},q_{11},q_{12},q_{14}\}=c_2$$

$$f'(c_5,0)=\bigcup_{q_k\in c_5} f(q_k,0)=\{q_2\}\xrightarrow{\lambda}\{q_1,q_2,q_3,q_5,q_6,q_8,q_{10}\}=c_1$$

$$f'(c_5,1)=\bigcup_{q_k\in c_5} f(q_k,1)=\{q_4,q_{11}\}\xrightarrow{\lambda}\{q_1,q_3,q_4,q_5,q_6,q_8,q_{10},q_{11},q_{12},q_{14}\}=c_2$$

c) El único estado de aceptación es $\mathbf{c_5}$ ya que contiene al estado de aceptación $\mathbf{q_A}$ del autómata no determinista antes determinado.

La definición formal del autómata obtenido puede verse en la Tabla 4.17 y su grafo en la Figura 4.29. Debe notarse que este autómata difiere del AFD obtenido en el Ejemplo 4.3, debido a éste que no es aún mínimo.

AFD = (Σ_E, Q', q_o', A', *f'*)

Σ_E = {0, 1}

Q' = {c_o, c_1, c_2, c_3, c_4, c_5}

q_o' = c_o

A' = {c_5}

***f'*:**	0	1
→ c_o	c_1	c_2
c_1	c_1	c_2
c_2	c_3	c_2
c_3	c_4	c_2
c_4	c_5	c_2
* c_5	c_1	c_2

Tabla 4.18: Definición formal del AFD del Ejemplo 4.12.

Se deja como ejercicio minimizarlo, determinando que los estados $\mathbf{c_0}$ y $\mathbf{c_1}$ resultan equivalentes, con lo cual el autómata mínimo será precisamente el mostrado en el Ejemplo 4.3.

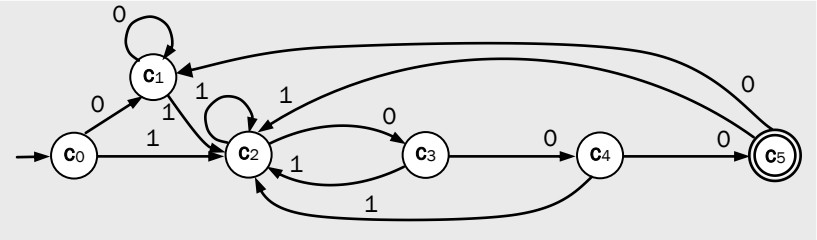


Figura 4.29: Grafo del AFD del Ejemplo 4.12.

Nuevamente, se debe recalcar el hecho que el procedimiento de Thompson puede parecer laborioso, como lo muestra el anterior ejemplo, pero al ser automático tanto el proceso de construcción del AFND-λ, como el de conversión a AFD equivalente y el de minimización, todo el proce-

so puede ser empaquetado en un programa que se constituirá en el analizador léxico de la expresión regular.

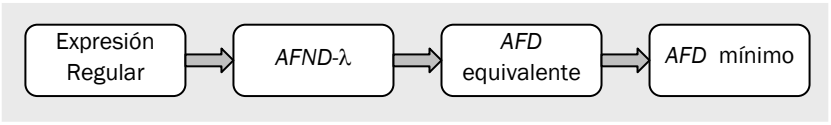


Figura 4.30: Proceso de construcción de un reconocedor de patrones.

Un comentario final sobre el análisis léxico de un lenguaje de programación real. Al diseñar los componentes léxicos del lenguaje, no se tendrá en general una única expresión regular; seguramente se describirán gran cantidad de *tokens* (palabras clave, constantes numéricas enteras en formato decimal, octal y hexadecimal, constantes de punto flotante en formato decimal y exponencial, constantes alfabéticas, operadores, identificadores, etcétera) utilizando expresiones regulares distintas.

A partir de cada una de ellas, se obtendrá por el método de Thompson un AFND-λ y todos ellos podrán ser unificados definiendo un nuevo estado inicial conectado con transiciones **λ** a cada uno de los estados iniciales de los mismos, con lo cual el proceso de la Figura 4.30 en realidad se verá como en la Figura 4.31.

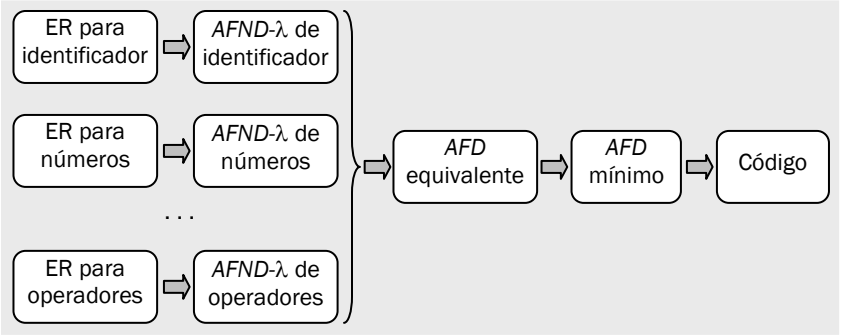


Figura 4.31: Proceso de construcción de un analizador léxico.

//