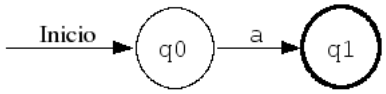
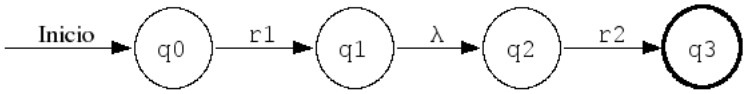
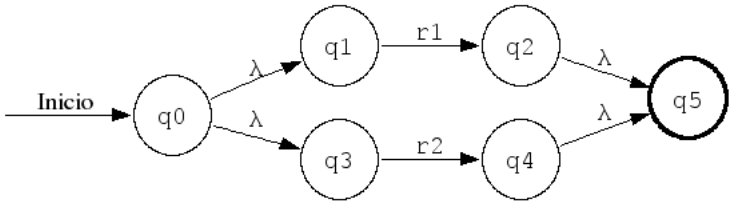
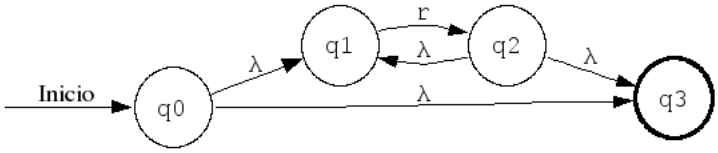
CI3725 - Traductores e Interpretadores
Abril-Julio 2006
Prof. Ernesto Hernández-Novich

Laboratorio Semana III

Conversión de Expresiones Regulares a AFND

El algoritmo para convertir expresiones regulares a AFNDs opera construyendo recursivamente el AFND en base a las expresiones regulares particulares. Los AFND a construir son los siguientes

Expresión Regular	Autómata Finito No Determinístico Generado
$r = \lambda$	
$r = a$	
$r = r_1 r_2$	
$r = r_1 + r_2$	
$r = r_1^*$	

Nótese que en los constructores recursivos, el algoritmo asume que los conjuntos de estados de los AFND a combinar son disjuntos. Al aplicar el algoritmos constructivamente esto puede garantizarse simplemente renombrando los estados de manera que nunca se repitan.

A continuación, algunas expresiones regulares y el AFND generado aplicando el algoritmo.

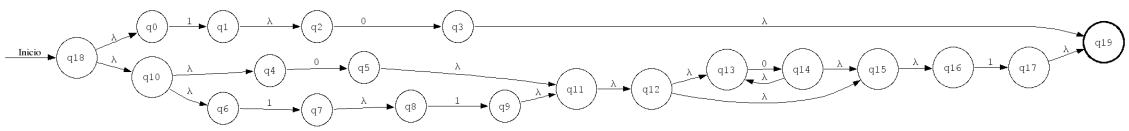


Figura 1: AFND generado a partir de la expresión regular$10 + (0 + 11)0^*1$

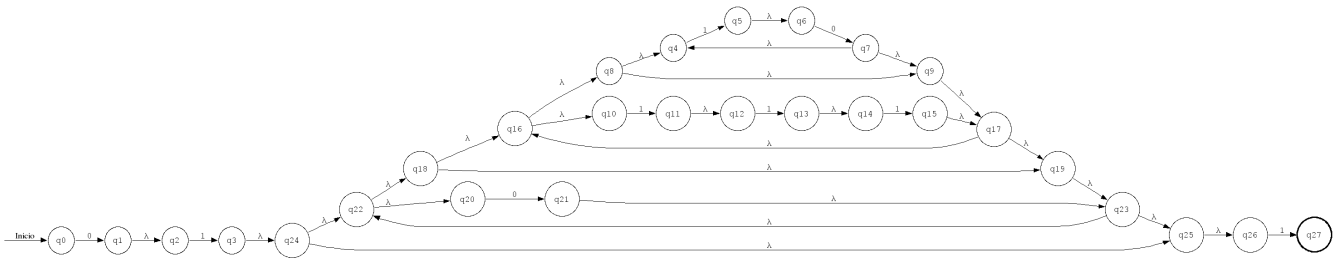


Figura 2: AFN generado a partir de la expresión regular $01(((10)^* + 111)^* + 0)^*1$

Conversión de AFND a AFD

Sea $M =< Q, \Sigma, \delta, q_0, F >$ un AFND con λ-transiciones, se construye un AFD equivalente $M' =< Q', \Sigma, \delta', q_0', F' >$ tal que

- $Q' = 2^Q$, esto es los estados del AFD son los conjuntos del conjunto de partes de Q. Cada uno de los estados de M' representa los posibles estados en los que se encuentra M en un momento particular de su ejecución.
- $F' = \{s | s \in Q' \wedge \exists f \in F \wedge f \in s\}$, esto es, cualquier estado en Q' que contenga un estado final de M será estado final en M'.
- $q_0' = [q_0]$, esto es, el estado inicial de M' es el estado que representa al conjunto $\{q_0\}$ donde $q_0$es el estado inicial de M. La notación $[q]$ es equivalente a decir "el estado que representa al conjunto $\{q\}$" y es encontrada en la literatura con frecuencia.
- $\delta'([q_1, q_2, \ldots, q_i], a) = [p_1, p_2, \ldots, p_j] \iff \delta(\{q_1, q_2, \ldots, q_i\}, a) = \{p_1, p_2, \ldots, p_j\}$, esto es, la nueva función de transición se construye tomando la unión de los estados destino resultantes de aplicar δ en M a los estados representados en el estado q de M'. Si el AFND tiene λ-transiciones, entonces debe calcular la $\lambda - CLAUSURA$ del conjunto.

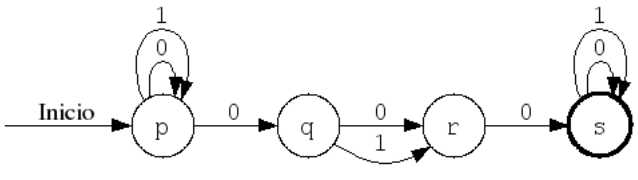
Nótese que el algoritmo producirá muchos estados inalcanzables, sin embargo, si se construye la función δ' comenzando desde el estado inicial y continuando solamente con los estados que aparezcan como resultado de la construcción (buscando en profundidad como en DFS), se obtienen sólo los estados alcanzables desde el inicial. A continuación, algunos AFND convertidos a AFD usando el algoritmo:

1. Sea el AFND $M =< \{p, q, r, s\}, \{0, 1\}, \delta, p, \{s\} >$ con δ definida como

	0	1
p	p, q	p
q	r	r
r	s	-
s	s	s

.

Una representación gráfica del AFND sería



Primero calculamos la nueva δ' partiendo del estado $[p]$ que es el inicial de M'. Así nos queda

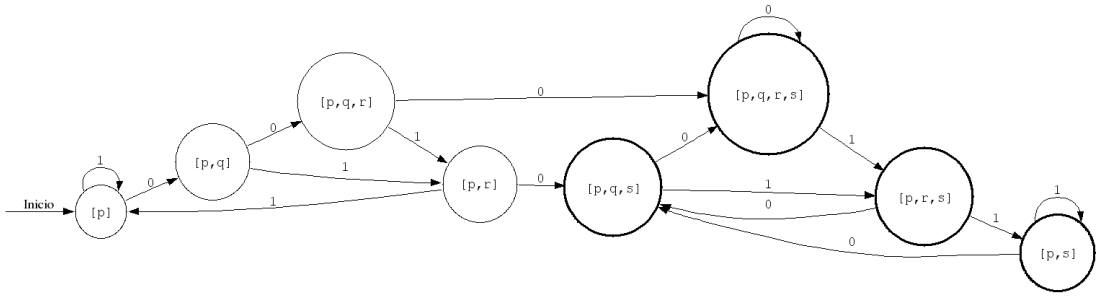
$$\begin{aligned} \delta'([p], 0) &= [p, q] \\ \delta'([p], 1) &= [p] \\ \delta'([p, q], 0) &= [p, q, r] \\ \delta'([p, q], 1) &= [p, r] \end{aligned}$$

$$
\begin{aligned}
\delta'([p,q,r],0) &= [p,q,r,s] \\
\delta'([p,q,r],1) &= [p,r] \\
\delta'([p,r],0) &= [p,q,s] \\
\delta'([p,r],1) &= [p] \\
\delta'([p,q,r,s],0) &= [p,q,r,s] \\
\delta'([p,q,r,s],1) &= [p,r,s] \\
\delta'([p,q,s],0) &= [p,q,r,s] \\
\delta'([p,q,s],1) &= [p,r,s] \\
\delta'([p,r,s],0) &= [p,q,s] \\
\delta'([p,r,s],1) &= [p,s] \\
\delta'([p,s],0) &= [p,q,s] \\
\delta'([p,s],1) &= [p,s]
\end{aligned}
$$

Entonces, podemos construir el AFD M' definitivo como

$M' =< \{[p],[p,q],[p,q,r],[p,r],[p,q,r,s],[p,q,s],[p,r,s],[p,s]\},\{0,1\},\delta',[p],\{[p,q,r,s],[p,q,s],[p,r,s],[p,s]\} >$

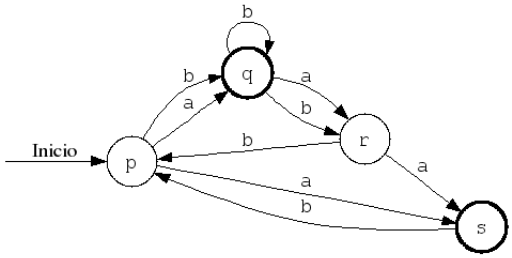
cuya representación gráfica sería



2. Sea el AFND $M =< \{p,q,r,s\},\{a,b\},\delta,p,\{q,s\} >$ con δ definida como

	a	b
p	q,s	q
q	r	q,r
r	s	p
s	-	p

Una representación gráfica del AFND sería



Primero calculamos la nueva δ' partiendo del estado $[p]$ que es el inicial de M'. Así nos queda

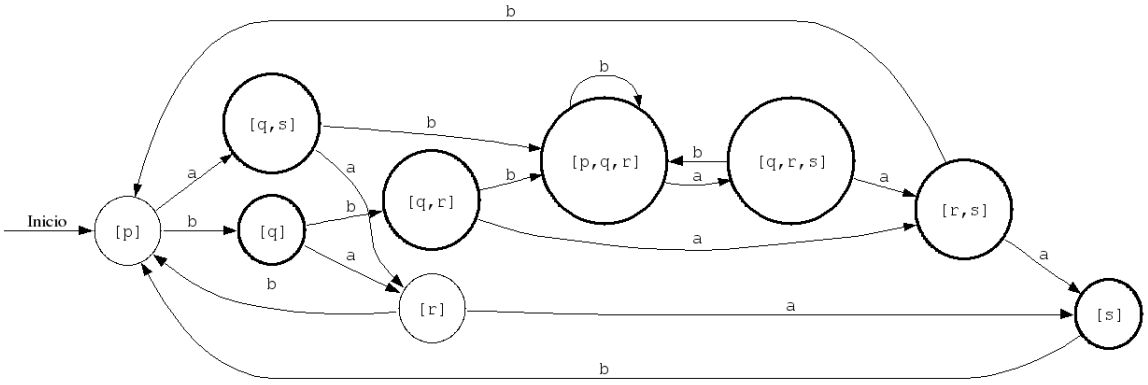
$$
\begin{aligned}
\delta([p],a) &= [q,s] \\
\delta([p],b) &= [q] \\
\delta([q,s],a) &= [r] \\
\delta([q,s],b) &= [p,q,r]
\end{aligned}
$$

$$
\begin{aligned}
\delta([q],a) &= [r] \\
\delta([q],b) &= [q,r] \\
\delta([r],a) &= [s] \\
\delta([r],b) &= [p] \\
\delta([p,q,r],a) &= [q,r,s] \\
\delta([p,q,r],b) &= [p,q,r] \\
\delta([q,r],a) &= [r,s] \\
\delta([q,r],b) &= [p,q,r] \\
\delta([s],a) &= \emptyset \\
\delta([s],b) &= [p] \\
\delta([q,r,s],a) &= [r,s] \\
\delta([q,r,s],b) &= [p,q,r] \\
\delta([r,s],a) &= [s] \\
\delta([r,s],b) &= [p]
\end{aligned}
$$

Entonces, podemos construir el AFD M' definitivo como

$M' =< \{[p],[q,s],[q],[r],[p,q,r],[q,r],[s],[q,r,s],[r,s]\},\{a,b\},\delta',[p],\{[q,s],[q],[p,q,r],[q,r],[s],[q,r,s],[r,s]\} >$

cuya representación gráfica sería

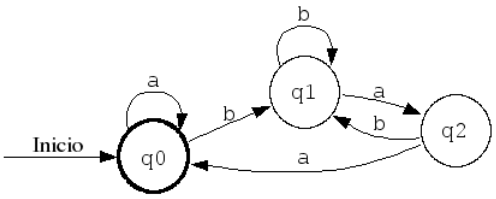
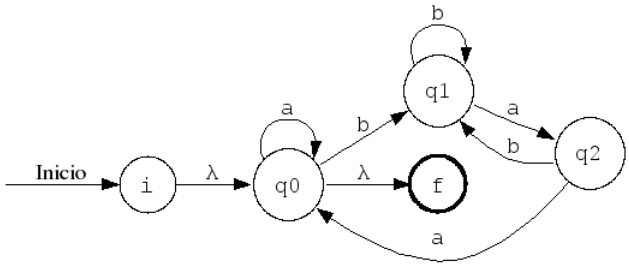
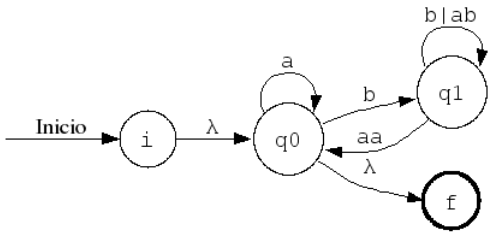
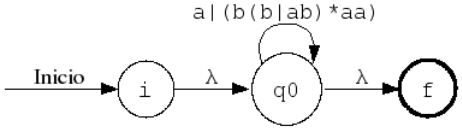
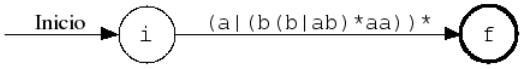


Conversión de AFD a Expresión Regular

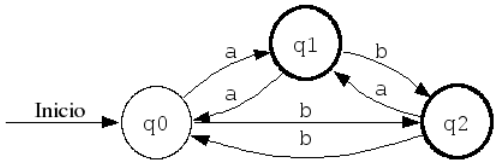
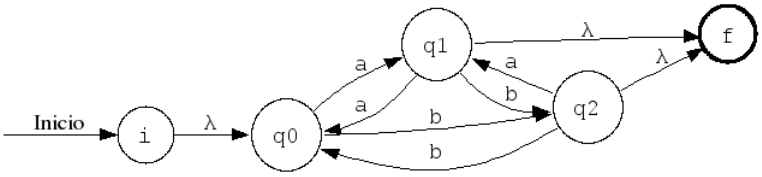
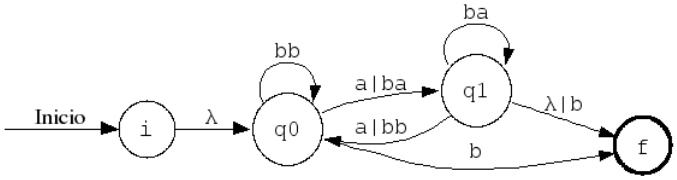
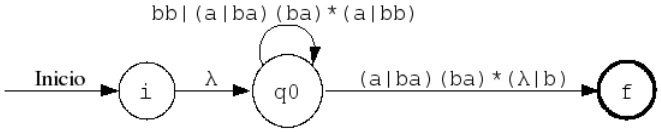
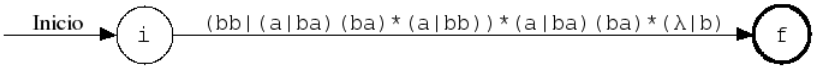
Si se tiene un AFD $M =< Q,\Sigma,\delta,q,F >$ y se desea saber la Expresión Regular que este acepta, puede aplicarse un algoritmo de construcción muy sencillo descrito informalmente como sigue:

1. Agregar un nuevo estado inicial i conectado con una λ-transición a q.
2. Agregar un nuevo estado final f y agregar λ-transiciones desde todos los $p \in F$ hasta f.
3. Seleccionar cualquier estado q del AFD extendido, salvo i o f, y eliminarlo del autómata. Al eliminarlo deben preservarse los caminos anteriores, sólo que las transiciones serán consideradas expresiones regulares y serán repuestas de la siguiente forma:
   a) Supongamos que existía una transición "entrante" desde algún $p \in Q$ hasta q y que tal transición tenía la expresión e_1.
   b) Supongamos que existía una transición "saliente" desde q hasta algún $r \in Q$ tal que tenía la expresión e_2.
   c) Supongamos que q tiene un bucle con la expresión e_3.
   d) Entonces, para eliminar el estado q, deben sustituirse la transición entrante, la transición saliente y el bucle, por una sola transición desde p hasta r que utilice la expresión $e_1e_3^*e_2$. Si ya existía una transición desde p hasta r con la expresión $e_4$entonces la nueva transición será $e_1e_3^*e_2 + e_4$.
4. Repetir el paso anterior hasta que solamente quede una transición entre i y f. La expresión asociada a esa transición es la Expresión Regular reconocida por el AFD.

Ejemplo 1:

Operación	Autómata Resultante
Original	
Extender el Autómata	
Eliminar q_2	
Eliminar q_1	
Eliminar q_0	
Expresión Regular	$(a\|(b(b\|ab)^*aa))^*$

Ejemplo 2:

Operación	Autómata Resultante
Original	
Extender el Autómata	
Eliminar q_2	
Eliminar q_1	
Eliminar q_0	
Expresión Regular	$(bb\|(a\|ba)(ba)^*(a\|bb))^*(a\|ba)(ba)^*(\lambda\|b)$

Construcción de Autómatas "Ad Hoc"

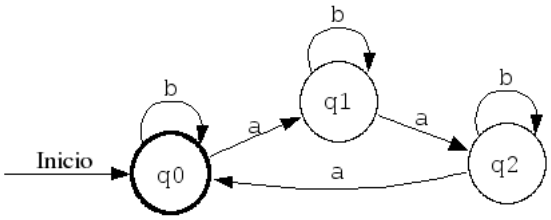
A veces es difícil encontrar una expresión regular para un conjunto que nos interesa reconocer. En esos casos suele ser más práctico construir un autómata "manualmente" y luego aplicar alguno de los algoritmos anteriores para encontrar la expresión regular específica.

1. Construir una expresión regular que reconozca cadenas sobre el alfabeto $\{a, b\}$ solamente si la cantidad de ocurrencias de a es múltiplo de tres.

   Comenzamos por observar que los múltiplos de tres son $0, 3, 6, 9, \ldots$ de modo que el autómata tendría que aceptar la palabra vacía. La manera de llegar al autómata final consiste en combinar las siguientes ideas:

   - El estado inicial q_0 tiene que ser final, puesto que debe aceptar la palabra vacía. Si el autómata está en el estado inicial es porque ha consumido una cantidad de a que es múltiplo de tres, y la cantidad de b es irrelevante. Esto quiere decir que habría un ciclo de q_0 a q_0 consumiendo b.
   - Si un número no es múltiplo de tres, entonces es congruente a 1 o a 2 módulo tres. Utilizaremos dos estados adicionales q_1y q_2 para modelar ambos casos. Si el autómata se encuentra en el estado q_0 y se consume una a, entonces debe pasar al estado q_1 pues de ser congruente a 0 ahora es congruente a 1. Un razonamiento análogo nos lleva desde q_1 hasta q_2 y desde q_2 hasta q_0 nuevamente. En q_1y q_2 cualquier cantidad de b pueden ser consumidas, así que al igual que en q_0 habría ciclos.

Así, nos quedaría construir el autómata como

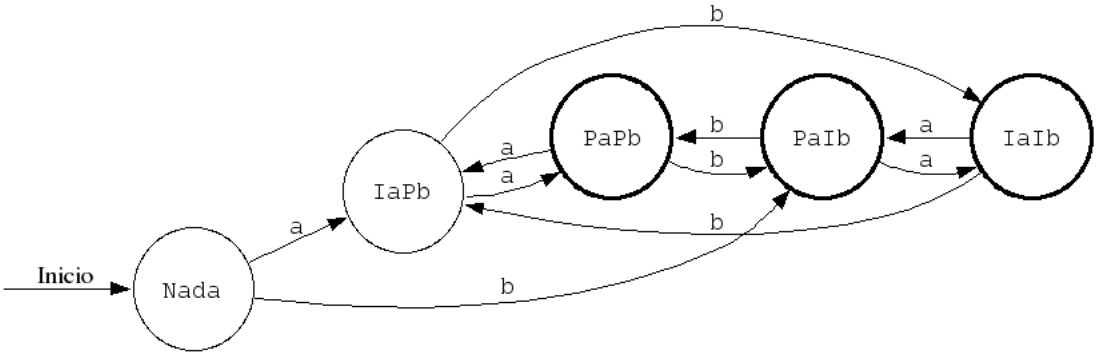


Y nos resta obtener la expresión regular asociada. Los pasos para hacerlo quedan como ejercicio para el lector, siendo el resultado $(b|ab^*ab^*a)^*$.

2. Construir una expresión regular que reconozca cadenas no vacías en $\{a, b\}$ si la cantidad de ocurrencias de a es par o la cantidad de ocurrencias de b es impar.

   - Comenzamos por observar que la palabra vacía no forma parte del lenguaje de manera explícita, así que el estado inicial no es final.
   - Observamos que en cualquier configuración del autómata hay cuatro posibilidades:
     a) Se ha recibido una cantidad par de a y una cantidad par de b. Llamaremos $PaPb$ a este estado. Más aún, ese estado es final pues tiene una cantidad par de a.
     b) Se ha recibido una cantidad par de a y una cantidad impar de b. Llamaremos $PaIb$ a este estado. Más aún, ese estado es final pues tiene una cantidad par de a o una cantidad impar de b.
     c) Se ha recibido una cantidad impar de a y una cantidad par de b. Llamaremos $IaPb$ a este estado. Este estado no es final pues no cumple con ninguna de las dos condiciones de aceptación.
     d) Se ha recibido una cantidad impar de a y una cantidad impar de b. Llamaremos $IaIb$ a este estado. Este estado es final pues tiene una cantidad impar de b.
   - Establecidas éstas configuraciones, es trivial deducir las transiciones entre esos estados dependiendo de la entrada, e.g. estando en $PaPb$ si se recibe una a la cantidad se vuelve impar, por tanto el estado pasa a ser $IaPb$; análogamente, estando en $PaPb$ si se recibe una b la cantidad se vuelve impar, por tnato el estado pasa a ser $PaIb$. El resto de las transiciones puede deducirlas el lector.
   - Cuando se está en el estado inicial, se ha recibido la palabra vacía. La palabra vacía tiene un número par de a y b (cero, en ambos casos) pero no corresponde al estado $PaPb$ sino al estado inicial. En ese estado inicial, si se recibe una a, debe pasarse a $IaPb$ pues tendríamos la primera a (una en total, y uno es impar) y ninguna b (cero, que es par); análogamente, si se recibe una b, debe pasarse a $PaIb$ por un razonamiento similar.

Así, nos quedaría construir el autómata como



Y nos resta obtener la expresión regular asociada. Los pasos para hacerlo quedan como ejercicio para el lector. **Nota:** se sugiere minimizar el autómata con el algoritmo que se estudiará en la próxima clase antes de calcular la expresión regular equivalente.