Teoría de la Computación para Ingeniería de Sistemas: un enfoque práctico

Prof. Hilda Contreras

25 de abril de 2012

Índice general

Capítulo 1

Expresiones regulares

Los Autómatas de estados finitos (AFD, AFND y AFND-λ) realizan una descripción procedural de los lenguajes regulares. En este tema se muestra un lenguaje denominado *Expresiones Regulares (ER)*, que permite definir un Lenguaje Regular de forma declarativa. Ambos modelos (ER y AF de forma equivalente) se aplican a los lenguajes tipo 3 de la jerarquía de Chomsky.

1.0.1. Definición de las expresiones regulares

Una expresión regular E describe el lenguaje L que representa, y se denota como L(E).

Lenguaje y sus operaciones

Para poder definir una expresión regular es necesario recordar las operaciones sobre lenguajes (conjuntos de cadenas): Sea Σ un alfabeto y L_1, L_2 y L_3 conjuntos de cadenas en Σ^*, las operaciones sobre ellos son:

1. **Unión**
   Sean L_1 y L_2 lenguajes, entonces $L_1 \bigcup L_2$ se define:
   $L_1 \bigcup L_2 = \{$ x | x en L_1 o x en L_2 $\}$

2. **Concatenación**
   Sean L_1 y L_2 lenguajes, entonces $L_1.L_2$ se define:
   $L_1.L_2 = \{$ x.y | x en L_1 y y en L_2 $\}$
   La cadena vacía λ es la identidad para la concatenación (λ.L = L.λ = L).

3. **Clausura** (Operador estrella o Clausura de Kleene)
   Sea L un lenguaje, la clausura de Kleene L^* se define:
   $L^* = \bigcup_{i=0}^{\infty} L^i$ (la unión de los L^i con i desde cero hasta el infinito)
   Donde se define L^i de la siguiente manera:

$$L^i = \begin{cases} L^0 = \lambda & \text{si i} = 0 \\ L^i = L.L^{i-1} & \text{i} \geq 1 \end{cases}$$

En otras palabras, L^* es el conjunto de todas las cadenas que se pueden formar tomando cualquier número de cadenas de L (con repeticiones) y concatenándolas entre si.
También, la *Clausura Positiva* se define como $L^+ = \bigcup_{i=1}^{\infty} L^i$ (la unión de los L^i con i desde

uno hasta el infinito), L^+ incluye a λ sólo si L incluye a λ
Ejemplo:
Dado los lenguajes L_1 y L_2 sobre el alfabeto Σ = { 0, 1 }:
L_1 = { 10, 1 } y L_2 = { 011, 11 }. Se realizan las siguientes operaciones sobre estos lenguajes:
$L_1 \bigcup L_2$ = { 10, 1, 011, 11 }
$L_1.L_2$ = { 10011, 1011, 111 }
L_1^* = {λ, 10, 1 , 1010, 101, 110, 11, 101010, 10101, 10110, 11010, ... }
Siendo:
L_1^0 = {λ}
L_1^1 = {10, 1}
L_1^2 = {1010, 101, 110, 11 }
L_1^3 = {101010, 10101, 10110, 1011, 11010, 1101, 1110, 111 }
Así sucesivamente por la definición $L.L^{i-1}$, se contruyen las cadenas de tamaño i a partir del lenguaje L y el operador de concatenación. Existen algunos casos particulares del operador clausura sobre lenguajes:

- ϕ^* = { λ }, ϕ^0 = { λ }
- La cadena vacía es la identidad en la concatenación de lenguajes. $L_1 = \lambda.L_1 = L_1.\lambda$
- El conjunto vacio y el conjunto de la cadena vacía, son los únicos lenguajes cuyas clausuras no son infinitas.

Definición formal de una expresión regular

Para la construcción de una Expresión Regular es necesario tener constantes y variables para la representación de los lenguajes, además de los operadores de Unión, Concatenación y Clausura sobre lenguajes ($\bigcup$, ., *). A partir de estos dos componentes su definición se basa en la recursividad según el número de operadores. Una Expresión Regular (ER) E describe el lenguaje L, L(E) = L, donde L $\subseteq \Sigma^*$ y se define recursivamente así:

- **Caso base** (sin operadores):

  1. Las Constantes λ y ϕ son ER y representan los Lenguajes {λ} y {} (ϕ) respectivamete. Por tanto L(λ) = {λ} y L(ϕ) = ϕ.
  2. Si a es un símbolo de Σ entonces a es una ER que denota el lenguaje {a}. Por tanto L(a) = {a}.
  3. Una variable (mayúscula) representa cualquier lenguaje. Por ejemplo E es una expresión regular que denota al conjunto de cadenas L(E).

- **Caso inductivo** (incluyendo los operadores +,.,* y el paréntesis):

  1. Operador Unión +: Si E y F son ER, entonces $E + F$ es una ER.
     Representa la unión de L(E) y de L(F), es decir L($E + F$) = L(E) $\bigcup$ L(F)
  2. Operador Concatenación .: Si E y F son ER, entonces $E.F$ es una ER.
     Representa la concatenación de L(E) y de L(F), es decir L($E.F$) = L(E).L(F)

3. Operador Clausura de Klenee (estrella o asterisco) *: Si E es una ER, entonces E^* es una ER.
   Representa la clausura de L(E), es decir L(E^*) = (L(E))*
4. Paréntesis (): Si E es una ER, entonces (E) (es una ER entre paréntesis) también es una ER.
   Denota el mismo lenguaje que E, es decir L((E)) = L(E)

El orden de precedencia de los operadores básicos de las ER es *, . y + (de mayor a menor). Se debe considerar lo siguiente:

1. . y + son asociativos por tanto no importa el orden de aplicación, por convención se comienza por la izquierda.
2. * se aplica a la secuencia de símbolos mas pequeña a su izquierda (es decir la ER bien formada a su izquierda).
3. Si se quiere cambiar el orden se debe usar paréntesis.

Ejemplos:

- La ER 00 denota el lenguaje {00}.
  L(00) = L(0).L(0) = {0}.{0} = {00}.
- La ER $0 + 1$ denota el lenguaje {0,1}.
  L($0 + 1$) = L(0) $\bigcup$ L(1) = {0} $\bigcup$ {1} = {0,1}.
- La ER $(0 + 1)^*$ denota todas las cadenas que se pueden formar con 0's y 1's.
  L((0+1)*) = (L((0+1)))* = (L(0) $\bigcup$ L(1))* = ({0} $\bigcup$ {1})* = {0, 1}* = {λ, 0, 1, 00, 01, 10, 11, 000, 001,...}. Otra forma de denotar Σ^* cuando Σ = {0,1}.

Algebra de las expresiones regulares

Dado que r,s,t son ER que denotan L(r), L(s) y L(t) respectivamente, se cumplen las siguientes propiedades:

- rλ = λr = r, el λ es el elemento neutro de la concatenación
- rϕ = ϕr = ϕ, el ϕ es el elemento nulo de la concatenación
- rr* = r*r = r^+
- r + s = s + r, conmutatividad de la unión
- (r + s) + t = r + (s + t), unión distributiva
- (r.s).t = r.(s.t), asociatividad de la concatenación
- r.(s + t) = rs + rt
- (r + s).t = rt + st

- $\phi^* = \lambda$
- $(r^*)^* = r^*$
- $(\lambda + r)^* = (r + \lambda)^* = r^*$
- $r^* + s^* = s^* + r^*$
- $(rs + r)^* r = r(sr + r)^*$

Ejemplos:
Para el alfabeto $\Sigma = \{0, 1\}$, se denotan los siguientes lenguajes con expresiones regulares.

1. $(1 + 10)^*$, denota las cadenas binarias, incluyendo la cadena vacia, que comienzan con 1 y no hay ceros seguidos
2. $(1^* + 0^*)$, denota las cadenas binarias, incluyendo la cadena vacía λ, que contienen solo ceros o bien solo unos.
3. $(01 + 1)^*$, denota las cadenas binarias que terminan con 1 y que no contiene la subcadena 00.
4. $(0 + 1)^*(00 + 11)(0 + 1)^*$, denota las cadenas binarias con al menos 2 ceros seguidos ó 2 unos seguidos

1.0.2. Autómatas Finitos y Expresiones Regulares

Sea A un Autómata Finito y E una Expresión Regular que respectivamente reconocen y denotan el mismo lenguaje L entonces L(A) = L(E). Ambos modelos representan el mismo conjunto de los *Lenguajes Regulares.*

Teorema 1. *Sea R una ER entonces existe un AFND M con λ-transiciones tal que L(M) = L(R).*

Demostración: por inducción sobre el número de operadores de R. Se asume que existe un AFND-λ [1] que acepta L(R) y que tiene solo un estado final sin transiciones de salida a partir de él.

- **CASO BASE**: Sin operadores.
  1. R = λ, por tanto $\{\lambda\}$ = L($\{\lambda\}$) y el AFND-λ se muestra en 1. de la figura 1.1.
  2. R = ϕ, por tanto ϕ = L(ϕ) y el AFND-λ se muestra en 2. de la figura 1.1.
  3. R = a, {a} = L(a) y el AFND-λ se muestra en 3. de la figura 1.1.
- **PASO INDUCTIVO**: Sea cierto para menos de i operadores y sea R con i o mas operadores.
  Por hipótesis inductiva existe $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\})$ y $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, \{f_2\})$ tal que L(M_1)=L(R_1) y L(M_2)=L(R_2), donde R_1 y R_2 son expresiones regulares que denotan a los lenguajes regulares respectivos.
  Asumiendo que $Q_1 \bigcap Q_2 = \phi$, se presentan los siguientes casos por cada operador:

[1] El AFND-λ obtenido en cada caso tiene un solo estado final que no tiene transiciones de salida (condición de la demostración y definición de este teorema).

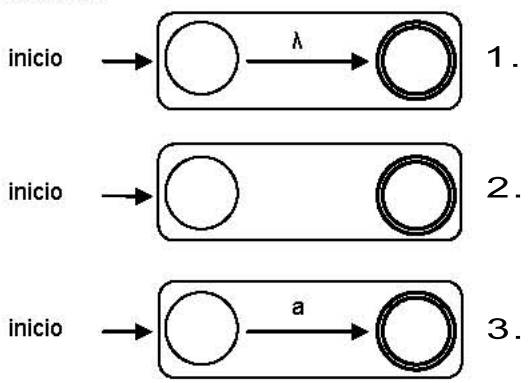

Figura 1.1: Definión del caso base de un AFND-λ a partir de una ER

1. Caso 1: Operador de Unión
   R = R_1 + R_2 con menos de i operadores.

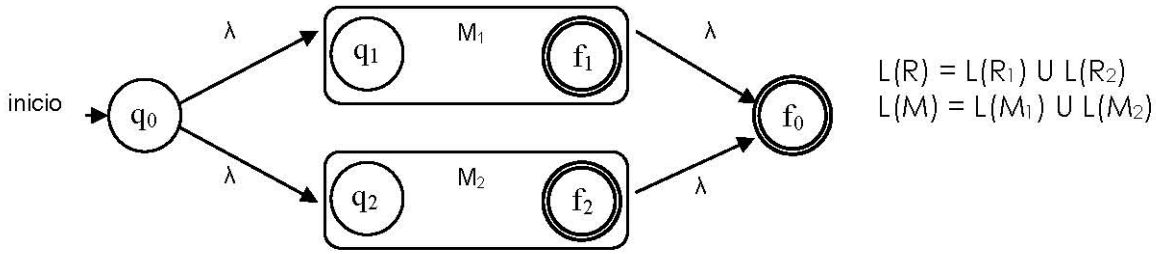

Figura 1.2: Definición del operador unión del AFND-λ a partir de Expresiones Regulares

Sean q_0 y f_0, dos nuevos estados, se construye M:
M = ($Q_1 \bigcup Q_2 \bigcup \{q_0, f_0\}$, $\Sigma_1 \bigcup \Sigma_2$, δ, q_0, $\{f_0\}$))
Donde δ:

a) δ(q,a) = δ_1(q,a), Para todo q en Q_1 - $\{f_1\}$ y a en $\Sigma_1 \bigcup \{\lambda\}$
b) δ(q,a) = δ_2(q,a), Para todo q en Q_2 - $\{f_2\}$ y a en $\Sigma_2 \bigcup \{\lambda\}$
c) $\delta(q_0,\lambda) = \{q_1, q_2\}$ (q_1, q_2 son estados iniciales de M_1 y M_2 respectivamente)
d) $\delta(f_1,\lambda) = \delta(f_2,\lambda) = \{f_0\}$

Ver figura 1.2. Se debe cumplir que L(R) = L(R_1) $\bigcup$ L(R_2) y L(M) = L(M_1) $\bigcup$ L(M_2). Si hay intersección entre 2 lenguajes L(M_1) y L(M_2) entonces por cualquiera de los dos autómatas se puede llegar al estado final.

2. Caso 2: Operador de Concatenación
   R = $R_1.R_2$ con R_1 y R_2 con menos de i operadores. En función de la hipótesis inductiva se construye M de la siguiente forma:
   M = ($Q_1 \bigcup Q_2$, $\Sigma_1 \bigcup \Sigma_2$, δ, q_1, $\{f_2\}$))
   Donde δ:

   a) δ(q,a) = δ_1(q,a), Para todo q en $Q_1 - \{f_1\}$ y a en $\Sigma_1 \bigcup \{\lambda\}$

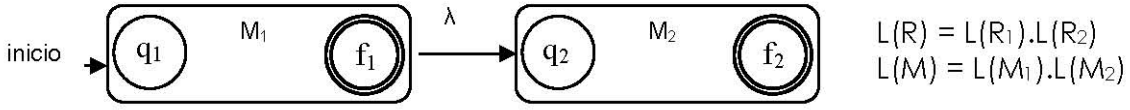

Figura 1.3: Definición del operador concatenación del AFND-λ a partir de Expresiones Regulares

b) δ(q,a) = δ_2(q,a), Para todo q en $Q_2 - \{f_2\} yaen \Sigma_2 \bigcup \{\lambda\}$

c) $\delta(f_1,\lambda) = \{q_2\}$

Ver figura 1.3. Se debe cumplir que L(R) = L(R_1).L(R_2) y L(M) = L(M_1).L(M_2). Lo único que acepta M son cadenas de M_1 seguidas por cadenas de M_2.

3. Caso 3: Operador Estrella o Clausura de Kleene
   R = R_1^* y R_1 con i-1 operadores. Por la hipótesis inductiva, existe una $M_1 = (Q_1$,

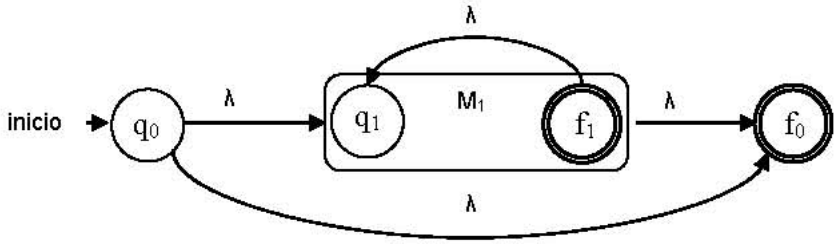

Figura 1.4: Definición del operador clausura de Kleene del AFND-λ a partir de Expresiones Regulares

Σ_1, δ_1, q_1, $\{f_1\}$) tal que L(M_1) = L(R_1), sea q_0 y f_0 son 2 nuevos estados entonces se construye M así:
M = ($Q_1 \bigcup \{q_0, f_0\}$, Σ_1, δ, q_0, $\{f_0\}$))
Donde δ:

a) δ(q,a) = δ_1(q,a), Para todo q en $Q_1 - \{f_1\}$ y a en $\Sigma_1 \bigcup \{\lambda\}$

b) $\delta(f_1,\lambda) = \{q_1, f_0\}$

c) $\delta(q_0,\lambda) = \{q_1, f_0\}$

Ver figura 1.4. Se debe cumplir que L(R) = $L(R_1)^*$ y L(M) = $(L(M_1))^*$. El autómata M acepta la cadena vacia, las cadenas de M_1 y cualquier cantidad de concatenación de las cadenas de L(M_1) seguidas por cadenas de L(M_1).

Ejemplos: Dado un lenguaje y/o su expresión regular obtenga el AF que lo reconozca.

1. R_1= 010, R= R_1^*, M= ($\{q_0, q_1, q_2, q_3, q_4, f_0\}$, {0, 1}, δ, q_0, $\{f_0\}$), L(M) = L(R)
   La figura 1.5 muestra el AFND-λ equivalente.

2. R= 01*+1 y L(R) = {1, 0, 01, 011, 01111, ...} R = $R_1 + R_2$, $R_2 = 1$, $R_1 = R_3R_4$, $R_3 = 0$, $R_4 = R_5^*$, $R_5 = 1$, entonces R = $R_3R_4 + R_2$
   La figura 1.6 muestra el AFND-λ equivalente.

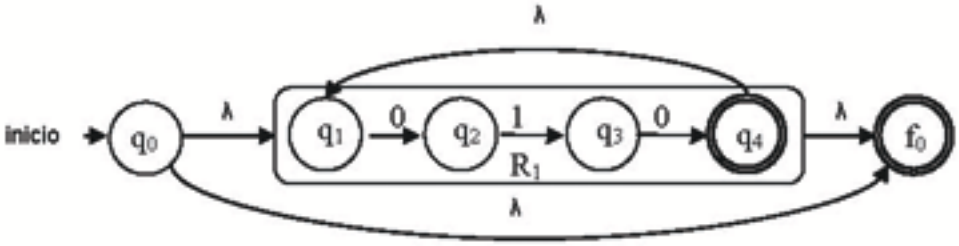

Figura 1.5: AFND-λ generado a partir de la Expresión Regular $(010)^*$

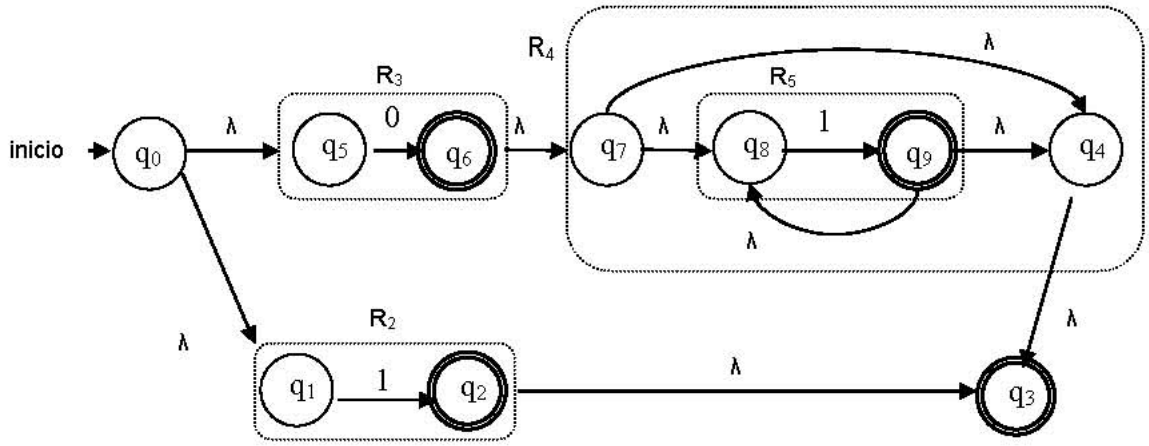

Figura 1.6: AFND-λ generado a partir de la Expresión Regular $01^* + 1$

El proceso recursivo que describe la demostración del teorema anterior ofrece una forma automática de realizar la transformación de una expresión regular en un AFND-λ. Sin embargo, debe notarse que dicho proceso genera un autómata con muchos estados y transiciones, además de una forma no determinística que es costosa en su ejecución y cálculo. Debe recordarse que es posible llevar un AFND-λ a un eficiente AFD a través de la construcción de subconjunto (tema anterior). Claro está, este proceso (de ER a AF) puede programarse o implementarse para realizar dicha transformación automaticamente, pero siempre se tiene la alternativa de realizarla entendiendo el lenguaje descrito por la expresión regular y tratando de obtener un autómata reconocedor del lenguaje. Esto se muestra en la figura 1.7, donde el AFD equivalente que se obtiene con sólo 4 estados fue realizado por un humano sin usar dicho procedimiento debido a que el lenguaje es trivial, en caso contrario un lengauje más complejo puede requerir el uso del procedimiento.

Esta sección muestra la equivalencia de ambos modelos (AF y ER), por tanto también debe ser posible dado un Autómata Finito obtener una Expresión Regular equivalente, tal como lo muestra el siguiente teorema.

Teorema 2. *Si L es aceptado por un AFD A, entonces L puede ser denotado por una ER E. Es decir, que L(A) = L(E).*

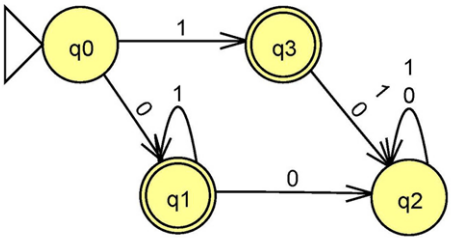

Figura 1.7: AFD equivalente al AFND-λ de la figura 1.6, generado a partir de la interpretación de la Expresión Regular $01^* + 1$

Demostración:
Los estados del AFD son etiquetados secuencialmente {1, 2, ..., n}. El autómata A tiene n estados. Se define R_{ij}^k a la expresión Regular cuyo lenguaje es L(R_{ij}^k) el conjunto de cadenas w tal que w es la etiqueta de un camino que va del estado i al j en A, sin pasar por nodos intermedios etiquetados con un número mayor que k ($\delta(q_i$,x) = q_j y si $\delta(q_i$,y) = q_l). Para cualquier prefijo propio y de x ($x \neq \lambda$) entonces $l \leq k$, como solo hay n estados R_{ij}^n son todas cadenas que llevan de q_i a q_j. Podemos definir R_{ij}^k recursivamente como sigue por inducción sobre k:

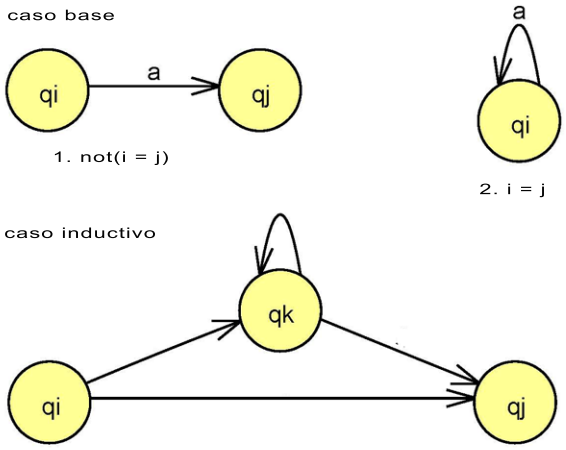

Figura 1.8: Caso base e inductivo de la transformación de un AFD a una Expresión regular

- Caso Base, para k = 0, determinar R_{ij}^0

  1. i $\neq$ j (arcos de i a j). Ver figura 1.8 caso base 1
     L(R_{ij}^0) = { a | $\delta(q_i$,a) = q_j }
     a) No existen arcos entre q_i y q_j, $R_{ij}^0 = \phi$
     b) Existen m símbolos $a_1, a_2, ..., a_m$ que etiquetan los arcos entre q_i y q_j, entonces

$L(R^0_{ij})$ = { $a_1, a_2, ..., a_m$ } y por tanto la expresión regular es $R^0_{ij} = a_1 + a_2 + ... + a_m$

2. i = j (ciclos de i a i, bucle). Ver figura 1.8 caso base 2
   $L(R^0_{ij})$ = { a | $\delta(q_i$,a) = q_j } $\bigcup$ { λ }
   a) Si existe el arco etiquetado a entre q_i y q_i, $L(R^0_{ij})$ = { a, λ } y por tanto la ER es $R^0_{ij} = a + \lambda$ [2]
   b) Si no existen arcos de q_i a q_i, entonces $L(R^0_{ij})$ = { λ}, y por tanto la expresión regular es $R^0_{ij} = \lambda$

- Caso Inductivo, sea cierto por hipótesis inductiva para k-1, determinar R^k_{ij}. Ver figura 1.8 caso inductivo

$$R^k_{ij} = R^{k-1}_{ik}(R^{k-1}_{kk})^* R^{k-1}_{kj} + R^{k-1}_{ij}$$

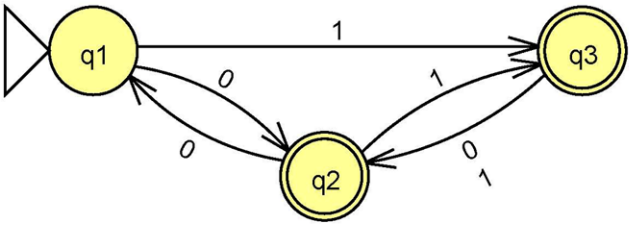

Figura 1.9: Ejemplo de un AFD para obtener la Expresión regular

La expresión regular para el lenguaje aceptado será la suma (unión) de las expresiones regulares R^k_{ij}, tal que i es el estado inicial, j sea un estado de aceptación y k la cardinalidad del conjunto de estados Q (k = n = |Q|).
Las cadenas aceptadas por el AFD A = (Q, Σ, δ, q_1, F), |Q| = n son L(A) = $\bigcup_{q_j enF} R^n_{1j}$. Es decir, la unión de todas las expresiones regulares desde el estado inicial q_1 hasta cada uno de los estados finales. De esta forma, L(A) estaría denotado por $R^n_{1j_1} + R^n_{1j_2} + ... + R^n_{1j_p}$, ya que F = $\{q_{1j_1}, q_{1j_2}, .. q_{1j_p}\}$. Por ejemplo en la figura 1.9 se muestra un AFD A = ($\{q_1, q_2, q_3\}$, Σ, δ, q_1, $\{q_2, q_3\}$) y la expresión regular a calcular sería R = $R^3_{12} + R^3_{13}$ ya que |Q| = 3. Dicha expresión regular se interpreta: R^3_{12} como las cadenas denotadas que van desde el estado inicial q_1 hasta el estado final q_2 pasando por todos los estados (los 3 estados), unido con R^3_{13} de las cadenas denotadas que van desde el estado inicial q_1 hasta el estado final q_3 pasando por todos los estados (los 3 estados).

Ejemplo:
Dado el AFD A de la figura 1.10, halle la expresión regular R que denota el lenguaje que reconoce el autómata (L(A) = L(R)).
A = ($\{q_1, q_2\}$, {0, 1}, δ, q_1, $\{q_2\}$), donde |Q| = 2

[2] Recuerdese que para todo AF se cumple que $\widehat{\delta}$(q,λ) = q, para todo q en Q, es decir la cadena vacía no cambia de estado, por tanto debe agregarse a la expresión regular del ciclo.

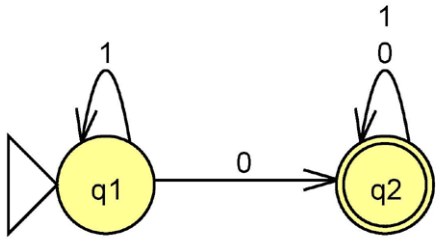

Figura 1.10: Ejemplo de un AFD para obtener la Expresión regular 1*0(0 + 1)*

Cuadro 1.1: Caso base de la transformación del AFD A de la figura 1.10 a una expresión regular R_{ij}^0

R_{11}^0	$1 + \lambda$
R_{12}^0	0
R_{21}^0	ϕ
R_{22}^0	$0 + 1 + \lambda$

R = $R_{12}^2 = R_{12}^1(R_{22}^1)^* R_{22}^1 + R_{12}^1$ (por la definición de la hipótesis inductiva). Se aplica la misma definición por recursividad a R_{12}^1 y R_{22}^1 hasta llegar al caso base (las expresiones del caso base se muestran en el cuadro 1.1).
$R_{12}^1 = R_{11}^0(R_{11}^0)^* R_{12}^0 + R_{12}^0 = (1+\lambda)(1+\lambda)^*0 + 0 = (1+\lambda)^*0 + 0 = 1^*\mathbf{0} + \mathbf{0}$
$R_{22}^1 = R_{21}^0(R_{11}^0)^* R_{12}^0 + R_{22}^0 = \phi(1+\lambda)^*0 + (0 + 1 + \lambda) = \phi + \mathbf{0} + \mathbf{1} + \lambda$
Realizando las sustituciones sobre R_{12}^2 y las simplificaciones convenientes resulta:
$R_{12}^2 = R_{12}^1(R_{22}^1)^* R_{22}^1 + R_{12}^1 = (1^*0 + 0)(0 + 1 + \lambda)^*(0 + 1 + \lambda) + 1^*0 + 0 =$
$= (1^*0 + 0)(0 + 1 + \lambda)^+ + 1^*0 + 0 = (1^*0 + 0)(0 + 1)^* + (1^*0 + 0) =$
$= (1^*0 + 0)((0 + 1)^* + \lambda)$, aplicando factor común $(1^*0 + 0)$
$= (1^*0 + 0)(0 + 1)^* = \mathbf{1^*0(0 + 1)^* = R}$

1.0.3. Autómatas con salidas

En las sesiones anteriores se realizó una revisión teórica de los autómatas finitos. Se presentaron los diferentes tipos de autómatas: determinista, no determinista y con transiciones nula. Se recuerda que los AFD tienen una salida binaria *si acepto* o *no acepto* al problema de decisión sobre una cadena w en Σ^* que está en el lenguaje L. En esta sección se presentan los autómatas finitos con salida, los cuales son capaces de generar una salida adicional.
Estos tipos de autómatas son muy utilizados en la vida real en problemas de cálculo matemático, transformación, traducción, contador, definición de protocolos, etc. Ellos proveen un valor agregado de mayor utilidad práctica que los autómatas finitos. Sin embargo, todas estas máquinas (AFD, AFND, AFND-λ, y cualquier AF con salida) son equivalentes, en el sentido de que reconocen el mismo tipo de lenguajes, pues poseer este valor agregado (todos reconocen el tipo 3 correspondientes a los lenguajes regulares según la jerarquía de Chomsky).
La definición de un Autómata con Salida AFS es la siguiente:

$$\text{AFS} = (\text{Q}, \Sigma, \Gamma, q_0, \delta, \omega)$$

- Q es el conjunto finito de estados
- Σ es el alfabeto (conjunto finito de símbolos) del lenguaje (entrada)
- Γ es el alfabeto (conjunto de símbolos) de salida
- q_0 es el estado inicial
- δ es la función de transición
- ω es la función de salida (estado o transición)

No existe el conjunto de estados finales F, porque ahora la respuesta no es *acepta* o *no acepta*, sino que la salida es una cadena formada por los símbolos del alfabeto de salida Γ a través de una función de transformación de salida ω. Existen 2 tipos de autómatas finitos con salida, según la función de transición de salida que se aplique:

1. La Máquina de Moore: una salida asociada a cada estado
2. La Máquina de Mealy: una salida asociada a cada transición

Máquina de Moore

La función de transición de salida de la Máquina de Moore[3], esta asociada al estado actual, es decir ω: Q $\rightarrow \Gamma$
En la Máquina de Moore, dada una entrada de longitud n ≥ 0 $a_1a_2...a_n$, la salida de M es de longitud n+1, tal que $\omega(q_0)\omega(q_1)...\omega(q_n)$ es la secuencia de estados de $q_0, q_1, ..., q_n$ tal que $\delta(q_{i-1}, a_i) = q_i$ para $1 \leq$ i $\leq$ n. Para el estado inicial se define que la $\omega(q_0) = \lambda$.
Entonces, un AFD puede ser visto como un caso particular de la Máquina de Moore donde el alfabeto de salida es {0, 1} y el estado q es de aceptación si y solo si ω(q) = 1.
Ejemplo:
Realizar un autómata para determinar o calcular el residuo de la división entera de 3 (modulo 3) de un número decimal expresado en binario.
Si m se divide entre 3 y su resultado es x y su residuo es p, entonces es claro que x * 3 + p = m. Si p = 0 entonces x * 3 es múltiplo de 3, y p llamado el módulo de 3 y es un valor entero, tal que p pertenece a los posible valores {0, 1, 2}. Moore resolvió este problema en los años 50 y determinó que para tres residuos posibles necesitaba 3 estados, tal como se muestra en la figura 1.11 y el cuadro 1.2 contiene los valores de entrada y salida para los primeros ocho (8) números decimales.
El autómata con salida definido por AFS = (Q, Σ, Γ, q_0, δ, ω), donde los conjuntos Q = $\{q_0, q_1, q_2\}$, Σ = {0, 1} y Γ = {0, 1, 2}. La función de transición de salida esta definida por la formula $\omega(q_i)$ = i, es decir:

- $\omega(q_0) = 0$
- $\omega(q_1) = 1$

[3]El nombre *Máquina de Moore* viene de su promotor: Edward F. Moore, pionero en el estudio de Autómatas, 1956.

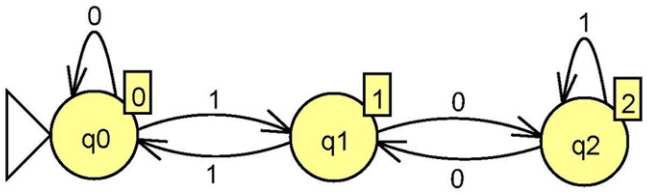

Figura 1.11: Máquina de Moore para calcular el módulo 3 de un número decimal

Cuadro 1.2: Relación de entrada y salida para calcular el modulo 3 de un número (Moore)

Decimal de m	Binario de m	Decimal de p
0	0	0
1	1	1
2	10	2
3	11	0
4	100	1
5	101	2
6	110	0
7	111	1

- $\omega(q_2) = 2$

Para el número decimal 11, expresado en binario como 1011 el autómata con salida de Moore anterior (ver figura 1.11) se ejecuta como un AFD salvo que en cada paso de transición se ejecuta la función de salida también, comportándose de la siguiente forma:

$\widehat{\delta}(q_0,1011) = \delta(\widehat{\delta}(q_0,101),1) = ... = \delta(\delta(\delta(\widehat{\delta}(q_0,\lambda),1),0),1),1)$, dado que $\widehat{\delta}(q_0,\lambda) = q_0$

- $= \delta(\delta(\delta(\delta(q_0,1),0),1),1), \omega(q_0) = 0$
- $= \delta(\delta(\delta(q_1,0),1),1), \omega(q_1) = 1$
- $= \delta(\delta(q_2,1),1), \omega(q_2) = 2$
- $= \delta(q_2,1), \omega(q_2) = 2$
- $= q_2, \omega(q_2) = 2$

La salida que es el resultado del autómata anterior corresponde a $\omega(q_2) = 2$ (q_2 es el estado que alcanza el autómata al terminar de procesar la cadena de entrada), es decir 2 es el residuo de la división entera de 11 entre 3. Sin embargo, el autómata muestra toda la secuencia de salida: 01222 en este caso, el último símbolo del alfabeto de salida es el que soluciona el problema.

Cuadro 1.3: Máquina de Mealy para traducir cadenas binarias

δ	0	1
q_0	$q_0/0$	$q_1/1$
q_1	$q_0/1$	$q_1/0$

Máquina de Mealy

La función de transición de salida de la Máquina de Mealy [4], esta asociada con la transición, es decir ω: Q x $\Sigma \rightarrow \Gamma$.
En la Máquina de Mealy para una entrada de longitud n ≥ 0 $a_1a_2...a_n$, la salida de M es de longitud n, tal que $\omega(q_0, a_1)$, $\omega(q_1, a_2)$ ... $\omega(q_{n-1}, a_n)$ donde $q_0, q_1, ..., q_n$ es la secuencia de estados tal que $\delta(q_{i-1}, a_i) = q_i$ para $1 \leq i \leq n$. La entrada de la cadena vacía en cualquier estado es $\omega(q, \lambda) = \lambda$.
Ejemplo:
Dado el alfabeto de entrada es {0, 1} y el de salida {0, 1}, se muestra un Autómata con salida que aplique una traducción que viene dada por las siguientes reglas:

- Si el primer símbolo es $0 \rightarrow 0$ y $1 \rightarrow 1$
- Para los siguientes símbolos:
  - Si el anterior es un 0 entonces: $0 \rightarrow 0$ y $1 \rightarrow 1$ (no varía)
  - Si el anterior es un 1: $0 \rightarrow 1$ y $1 \rightarrow 0$ (se aplica el contrario o complemento)

Esta reglas permiten que dado un número binario, por ejemplo 0110 su salida traducida por dichas reglas sería 0101. El cuadro 1.3 muestra la tabla de transición de la función de δ y ω y el diagram de transición se muestra en la figura 1.12. Para la cadena 0110 el autómata con

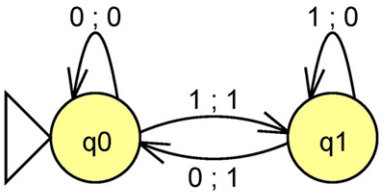

Figura 1.12: Máquina de Mealy para realizar una traducción

salida de Mealy de la figura 1.12 se ejecuta como un AFD salvo que en cada paso de transición se ejecuta la función de salida también, comportándose de la siguiente forma: $\widehat{\delta}(q_0,0110) = \delta(\widehat{\delta}(q_0,011),0) = ... = \delta(\delta(\delta(\delta(\widehat{\delta}(q_0,\lambda),0),1),1),0)$, dado que $\widehat{\delta}(q_0,\lambda) = q_0$

- $= \delta(\delta(\delta(\delta(q_0, 0), 1), 1), 0), \omega(q_0, 0) = 0$
- $= \delta(\delta(\delta(q_0, 1), 1), 0), \omega(q_0, 1) = 1$

[4] El nombre "Máquina de Mealy"viene dado por G. H. Mealy, un pionero de las máquinas de estados, quien escribió Un Método para sintetizar Circuitos Secuenciales, 1955

- $= \delta(\delta(q_1, 1), 0), \omega(q_1, 1) = 0$
- $= \delta(q_1, 0), \omega(q_0, 0) = 1$

Dependiendo de la aplicación del autómata de salida, la salida resultado puede ser una parte específica, o bien toda la cadena generada de salida. En el caso de este ejemplo debe considerarse la traducción completa de w, pues representa la traducción según las reglas descritas sobre toda la cadena de entrada [5].

Equivalencia entre la Máquina de Moore y Mealy

Existe un procedimiento para obtener una Máquina de Moore a partir de una Máquina de Mealy. Consiste en el siguiente algoritmo que permite obtener una máquina de Moore M_2 a partir de una máquina de Mealy M_1:
Dado el autómata M_1:

$$M_1 = (\text{Q}, \Sigma, \Gamma, q_0, \delta, \omega) \text{ un autómata de Mealy}$$

El Autómata M_2 se genera con el siguiente procedimiento:

$$M_2 = (\text{Q x } \Gamma, \Sigma, \Gamma, [q_0,b_0], \delta', \omega') \text{ un autómata de Moore}$$

Los estados de M_2 son pares Q x Γ, por ejemplo [q,b] con q en Q y b en Γ:

- δ'([q,b],a) = [δ(q,a), ω(q,a)]
- ω(q,b) = b

Ejemplo:
Dado el autómata de Mealy anterior, figura 1.12, se transforma en una máquina de Moore, figura 1.4 utilizando la definición anterior:

- Q = $\{q_0, q_1\}$, $\Sigma = \{0, 1\}$ y $\Gamma = \{0, 1\}$
- Q x Γ = {[q_0,0], [q_0,1], [q_1,0], [q_1,1]}
- q_0 en M_2 es uno de los pares (q_0, A) con A en Γ
- Función de transición δ' está definida en la tabla de transición del cuadro 1.4 y se muestra el diagrama de transición en la figura 1.13.

[5] La salida se interpreta dependiendo del problema de aplicación, en algunos casos es parte de la salida, en otros es la salida completa, todo al mismo tiempo para diferentes aplicaciones, etc.

Cuadro 1.4: Máquina de Moore (traducción) obtenida a partir de una Máquina de Mealy

	δ'	0	1
q_0	$[q_0,0]$	$[q_0,0]$	$[q_1,1]$
q_1	$[q_0,1]$	$[q_0,0]$	$[q_1,1]$
q_2	$[q_1,0]$	$[q_0,1]$	$[q_1,0]$
q_3	$[q_1,1]$	$[q_0,1]$	$[q_1,0]$

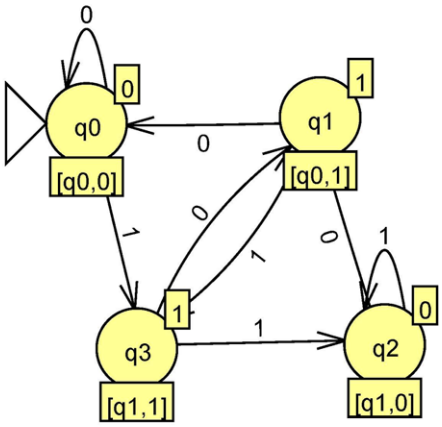

Figura 1.13: Diagrama de transición Máquina de Moore (traducción) obtenida a partir de una Máquina de Mealy

1.0.4. Herramientas, implementación y uso de Lenguajes regulares

Expresiones regulares en lenguajes de programación

[6] Las aplicaciones basadas en expresiones regulares, usan una definición más comoda, reducida y funcional que su definición formal. En el cuadro 1.5 se compara los operadores básicos de la definición formal con su equivalente en POSIX. El POSIX Portable Operating System Interface (X viene de UNIX), surge de un proyecto de normalización del API de Unix. Actualmente tiene un amplio uso en muchos lenguajes de programación, sistemas y aplicaciones, además de poseer su versión abierta GNU Regular Expression Extensions (The Open Group en Single Unix Specification).

La ventaja principal de esta notación de las expresiones regulares POSIX es que se refiere al conjunto de los caracteres ASCII, como la mayor parte de las aplicaciones reales. Las clases, secuencias e intervalos de caracteres (las notaciones especiales para ciertas clases de caracteres) se denotan de forma mas sucinta a pesar de contar con 128 caracteres. Las expresiones regulares UNIX puede utilizar varios operadores además de los básicos, sin que esto signifique que se amplíe el tipo de lenguaje a reprsentar (se trata solo de facilitar la expresión). Un resumen de los operadores adicionales POSIX se muestra en el cuadro 1.6 y de la extensión de POSIX se muestra en el cuadro 1.7. La predecendia de los operadores de la ER en UNIX es la misma

[6] Ver presentación sobre Taller de POSIX.

Cuadro 1.5: Comparación de los operadores básicos ER formal y POSIX

Operador básico	ER-formal	ER-POSIX
unión	+ pe. (a+b)	| pe. (a|b)
concatenación	. pe. (a.b) ó (ab)	[7] pe. (ab) ó ab
clausura (repite cero o más)	* pe. a*	* pe. a* [8]

Cuadro 1.6: POSIX

POSIX	Descripción
.	Cualquier carácter excepto fin de línea
∧	Comienzo de la cadena ó negación de conjuntos
[]	Lista o clases de símbolos. Pe. [abc] Conjunto {a,b,c}, [∧abc] Conjunto Σ - {a,b,c}, [0-9] Rangos del 0 al 9, todos los dígitos
∧	Comienzo de la cadena ó negación de conjuntos
$	Final de la cadena
╲	Escape (no considera la expresión siguiente)
r+	1 o más ocurrencias de r
r?	0 o 1 ocurrencia de r
r{n}	n ocurrencias de r
r{n,}	n o mas ocurrencias de r
r{,m}	0 o a lo sumo m ocurrencias de r
(r)	r evaluada con prioridad ó caracteres de salida con variable $n
r	no interpretar los símbolos de r

Cuadro 1.7: ER POSIX extendido

ER-POSIX	Equivalentes	Descripción
\d	[:digit:]	los dígitos del 0 al 9, lo mismo que [0-9]
\w	[:alpha:]	Caracter alfabético, lo mismo que [A-Za-z]
\s	[:almun:]	* Caracteres alfanuméricos (dígitos y letras) lo mismo que [A-Za-z0-9]
\r ó \n		Salto de linea
\t		Tabulación

que la definición formal, además de considerarse que ?, + y {n} tienen la misma precedencia que *.
Considerando el siguiente texto:

```
Mérida 20-09-2012. Para conocer objetivos, ejes temáticos, tarifas y
procedimiento de inscripción, visite nuestra página web: www.seguridadsf.com
<http://www.seguridadsf.com/> o comuníquese con nosotros a través de los
números: +58 274 251.26.99 / +58 274 511.21.12 / 0414 723.39.32 / 0416 775.92.53
/ 0416 117.99.81 o el correo: info@seguridadsf.com
```

Las siguientes expresiones regulares denotan salidas especificas:

- Identifica el formato fecha dd-mm-aaaa y retorna por separado cada componente de la fecha:
  ER entrada

  ```
  ^.*([0-9]{1,2})-([0-9]{1,2})-([0-9]{4}).*$
  ```

  ER salida $3/$2/$1
  Salida del ejemplo: 2012/09/20

- Reconoce una secuencia de caracteres con formato email y retorna el nombre de usuario:
  ER entrada

  ```
  ^.* ([a-z0-9._\%-]+)@[a-z0-9.-]+\.[a-z]{2,4}.*$
  ```

  ER salida $1
  Salida del ejemplo: info

La lista completa de los operadores y abreviaturas disponibles en la notación UNIX para las expresiones regulares se encuentra en el manual de los comandos del sistema operativo (se presentan diferencias entre las versiones de cada sistema). La aplicación más útil es el grep (Global Regular Expression and Print) o egrep de UNIX (ver man grep).

Apesar de que existe un estandar, POSIX fue definido antes del surgimiento del lenguaje de programación Perl (famoso por sus expresiones regulares) y desde entonces se han ido incorporando muchas características. Los lenguajes de programación y aplicaciones están adoptando las expresiones regulares de Perl en vez de las expresiones regulares POSIX, incluyendo PHP, Java, Ruby y el Servidor Web de Apache. Por tanto se recomienda revisar que expresiones emplea el lenguaje de programación o la aplicación a utilizar pues probablemente tenga variantes a las mostradas en este documento.

1.0.5. Preguntas y respuestas, ejercicios resueltos y propuestos

Preguntas y respuestas

1. Qué significa que la expresión regular es una descripción declarativa del lenguaje regular
2. Las expresiones regulares POSIX parecen más poderosas que la definición formal, por qué no lo son?
3. Cuál es la utilidad de los autómatas con salida?

Ejercicios propuestos

1. Construir los autómatas para reconocer los lenguajes denotados por las siguientes expresiones regulares:
   a) (0+1)*011
   b) (1+10)*
   c) 0*1*0*
   d) (0+λ)(0+10)*
   e) (111+100)*0
   f) (010+00)*(10)*
2. Halle la expresión regular ER de los siguientes lenguajes cuando $\Sigma = \{0,1\}$
   a) Cadenas binarias de longitud par.
   b) Cadenas binarias con un número de 1's impar.
   c) Cadenas binarias que terminan con 1 y no contienen la subcadena 00.
   d) Cadenas binarias con el 0 como penúltimo símbolo.
   e) Cadenas binarias que contienen 01 o 110.
   f) Cadenas binarias que contiene por cada 0 un 11 seguido.
3. En cada caso, encuentre la cadena de longitud mínima (sin considerar la cadena vacía λ) que NO esté en el lenguaje denotado por las siguientes expresiones regulares:
   a) 10* + 1*0 + 01*
   b) (1*)(0*+1*)(0*+1*)

c) ((0*1*))* + (0*1)*

d) (00 + λ)*(01+00+10)*

e) (a + b + ab + λ + bba)*

f) (1*)(0*+11*)(0*+1*)

4. Cuántas y cuáles cadenas de tamaño estrictamente menor que 3 tiene el lenguaje denotado por la siguiente expresión regular ((0*1*)*0)*. Dé el autómata finito que reconoce dicho lenguaje.

5. Construir autómatas con salida para solucionar los siguientes problemas:

   a) Máquina expendedora de boletos (por un monto de 1,75 Bs) que muestre como salida el mondo introducido y el resto. Asuma que la máquina reconoce solo las monedas de 1, 0.5 y 0.25

   b) Dado un número en binario (a partir de los dígitos menos significativos) calcule la función incremento o sucesor en binario. Puede traducirse este autómata a un circuito lógico

   c) Dado un número en binario (a partir de los dígitos menos significativos) multiplique por 2 en binario (considere en que consiste multiplicar un número por su base)

   d) Dado una cadena en el alfabeto binario 0,1 muestre su traducción o codificación sobre el alfabeto a,b, tal que una a es equivalente a 2 unos seguidos y una b es equivalente a dos ceros seguidos (Ejemplo sencillo de la codificación de caracteres, por ejemplo el ASCII)

Bibliografía

[1] Hopcroft, Rajeev Motwani, Jeffrey D. Ullman. Introducción a la Teoría de Autómatas y Lenguajes Formales. PrenticeHall, 2002.

[2] Jan Goyvaerts, Steven Levithan. Regular Expressions Cookbook, Detailed Solutions in Eight Programming Languages. O'Reilly Media, May 2009

[3] Jeffrey E.F. Friedl. Mastering Regular Expressions, 3rd Edition, Understand Your Data and Be More Productive. O'Reilly Media. August 2006.

`http://cdn.oreilly.com/oreilly/booksamplers/9780596528126_sampler.pdf`