**UNIVERSIDAD DE ANTIOQUIA**

**Departamento de Ingeniería de Sistemas**

**Laboratorio de Teoría de Lenguajes (ISI-471)**

Consultar

1.     Que es una expresión regular, que la componen y como se representa.

2.     Como construir un autómata finito a partir de una expresión regular.

3.     Entregar en forma escrita varios ejemplos de expresiones regulares donde se muestra el proceso para construir el autómata finito a partir de la ER,  mostrar este mediante una matriz de transiciones o mediante un gráfico.  Se debe entregar la ER y autómata resultante.

**Observaciones Generales:**

1.   Hacer la consulta,  no entregar las mismas notas de clase.

2.   Las prácticas serán presentadas en grupos conformados por dos ó tres personas y se debe exponer la consulta.

3.   El porcentaje de la primera práctica será del 30% de la evaluación del curso.

4.   La fecha de entrega para la primera práctica será: Junio 26.

**Desarrollo**

* Que es una expresión regular?

Una expresión regular es una forma de representar a los lenguajes regulares (finitos o infinitos) y se construye utilizando caracteres del alfabeto sobre el cual se define el lenguaje.

Las expresiones regulares son una poderosa herramienta para manipular textos y datos, con el uso de expresiones regulares usted puede ahorrar tiempo y problemas a manipular documentos, mensajes de correo electrónico (e-mail), archivos de registro, cualquier tipo de información que contenga texto o datos, por ejemplo, las expresiones regulares juegan un papel importante en la construcción de programas que corren sobre Internet (cgi-bin), ya que pueden implicar textos y datos de todo tipo.

Las expresiones regulares no son un programa de computadora pero son incluidas como parte de un programa mas grande, el clásico ejemplo es grep2, una potente utilidad comúnmente utilizada en Unix; Las expresiones regulares pueden ser encontradas en:

Lenguajes de programación (perl, tcl, awk, python, java, ...) Editores (Emacs, vi, Nisus Writer, ... ) Entornos de desarrollo (Delphi, Visual C++, ...)

Por el Libro:**Construcción de Compiladores:**

<< Las expresiones regulares representan patrones de cadenas de caracteres. Una expresión regular ***r*** se encuentra completamente definida mediante el conjunto de cadenas con las que concuerda. Este conjunto se denomina **Lenguaje generado por la expresión regular** y se escribe como L(*r*). Aquí la palabra lenguaje se utiliza solo para definir "conjunto de cadenas" y no tiene una relación específica con el lenguaje de programación en concreto. Este lenguaje depende, en primer lugar, del conjunto de caracteres que se encuentra disponible. El conjunto de símbolos legales que compone un Lenguaje se le conoce como **alfabeto** y por lo general se representa mediante el símbolo griego sigma (∑).

Una expresión regular *r* puede contener caracteres que tengan significados especiales. Éste tipo de caracteres se llaman **metacaracteres** o **metasímbolos**, y por lo general no puede ser caracteres legales en el alfabeto, por que no podríamos distinguir su uso como metacaracteres de su uso como miembros del afabeto.

Consideremos todas las piezas de la definición de una expresión Regular

Una Expresión Regular es una de las siguientes:

1. Una esxpresión regulas **básica** constituida por un sólo caracter **a**, donde a probiene de un alfabeto ∑ de caracteres legales; el metacarácter &#x1d73a;; o el metacarácter &#x1d753;. En el primer caso, L(a)={a}; en el segundo, L(&#x1d73a;)={&#x1d73a;}; en el tercero, L()={&#x1d753;}.
2. Una expresión de la forma ***r*|s**, donde *r* y *s* son expresiones regulares. en este caso, L(*r*|s)=L(*r*)∪L(s).
3. Una expresión de la forma **rs**, donde *r* y *s* son expresiones regulares. En este caso L(rs)=L(*r*)L(s).
4. Una expresión de la forma ***r*\***, donde *r* es una expresión regular. En este caso, L(*r*\*)=L(*r*)\*.
5. Una expresion de la forma **(*r*)**, donde *r* es una expresión regular. En este caso L( (*r*) )=L(*r*). De este modo, los paréntesis no cambian el lenguaje, sólo se utilizan para ajustar la precedencia de las operaciones.

* Que compone una expresión regular?

 Las expresiones regulares consisten en las constantes y los operadores que denotan sistemas de secuencias y de operaciones sobre estos sistemas, respectivamente. Dado un alfabeto finito Σ las constantes siguientes se definen:

* (*vacié el sistema*) φ denotar el sistema {φ}
* (*secuencia vacía*) ε que denota el sistema {ε}
* (*carácter literal*) *a* en Σ que denota el sistema {*a*}

Se definen las operaciones siguientes:

* (*encadenamiento*) *RS* denotar el sistema {αβ | α adentro *R* y β adentro *S* }. Por ejemplo {“ab”, “c”} {“d”, “ef”} = {“abd”, “abef”, “Cd”, “cef”}.
* (*alternación*) *R|S* denotar la unión del sistema de *R* y *S*. Muchos libros de textos utilizan los símbolos , +, o   para la alternación en vez de la barra vertical. Por ejemplo {“ab”, “c”} {“d”, “ef”} = {“ab”, “c”, “d”, “ef”}
* (*Estrella de Kleene*) *R*\* denotar el más pequeño sobreconjunto de *R* eso contiene el ε y es cerrado bajo encadenamiento de la secuencia. Éste es el sistema de todas las secuencias que puedan ser hechas concatenando cero o más secuencia adentro *R*. Por ejemplo, {“ab”, “c”} \* = {ε, “ab”, “c”, “abab”, “ABC”, “taxi”, “cc”, “ababab”, “abcab”,…}.

Las constantes y los operadores antedichos forman a Álgebra de Kleene.

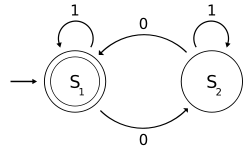
Para evitar los soportes se asume que la estrella de Kleene tiene la prioridad más alta, después el encadenamiento y la unión entonces fijada. Si no hay ambigüedad entonces los soportes pueden ser omitidos. Por ejemplo, (ab) c puede ser escrito como ABC, y a|(b (c\*)) puede ser escrito como a|bc\*.

**Ejemplos:**

* a|b\* denota {ε, *a*, *b*, *bb*, *bbb*, ...}
* (a|b) \* denota el sistema de todas las secuencias sin símbolos con excepción de *a* y *b*, incluyendo la secuencia vacía
* ab\* (c|ε) denota el sistema de secuencias comenzando con *a*, entonces ponga a cero o más *b*s y finalmente opcionalmente a *c*.
* Cómo se representa una expresión regular?

Una expresión regular es representada como una cadena de caracteres, además puede ser representada mediante un autómata finito y este a su vez se puede representar de las siguientes formas:

**Representación como diagramas de estados**

[](http://es.wikipedia.org/wiki/Archivo:DFAexample.svg)

Los autómatas finitos se pueden representar mediante [grafos](http://es.wikipedia.org/wiki/Grafo) particulares, también llamados *diagramas de estados finitos*, de la siguiente manera:

* Los estados se representan como [vértices](http://es.wikipedia.org/wiki/V%C3%A9rtice_(teor%C3%ADa_de_grafos)), etiquetados con su nombre en el interior.
* Una transición desde un estado a otro, dependiente de un símbolo del alfabeto, se representa mediante una [arista](http://es.wikipedia.org/wiki/Arista_(teor%C3%ADa_de_grafos)) [dirigida](http://es.wikipedia.org/wiki/Grafo_dirigido) que une a estos vértices, y que está etiquetada con dicho símbolo.
* El estado inicial se caracteriza por tener una arista que llega a él, proveniente de ningún otro vértice.
* El o los estados finales se representan mediante vértices que están encerrados a su vez por otra circunferencia.

**Representación como tabla de transiciones**

Otra manera de describir el funcionamiento de un autómata finito es mediante el uso de **tablas de transición de estados**.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |

|  |  |  |
| --- | --- | --- |
|  | **0** | **1** |
| ***s*1** | *s*2 | *s*1 |
| ***s*2** | *s*1 | *s*2 |

En esta representación los valores de las columnas corresponden a los símbolos de entrada o caracteres posibles de la expresión regular, y los elementos de la fila corresponden al estado del autómata finito en el que estará dependiendo del símbolo de entrada.

* Cómo construir un autómata finito a partir de una expresión regular?

El algoritmo más simple para traducir una expresión regular en un AF pasa por una construcción intermedia, en la cual se deriva un AFND de la expresión regular y posteriormente se emplea para construir un AF equivalente.

Para pasar de una expresión regular a un AFND se emplean las **construcciones de Thompson.** La cual utiliza transiciones ε para “pegar” las maquinas de cada segmento de una expresión regular con el fin de formar una maquina que corresponda a la expresión regular completa. Se tienen unas construcciones básicas que en base a ellas nos permitirán llegar al AFND.

Ahora para pasar de un AFND a un AF o AFD (Autómata finito determinístico∪), necesitaremos algún método para eliminar tanto las transiciones ε como las transiciones múltiples de un estado en un carácter de entrada simple.

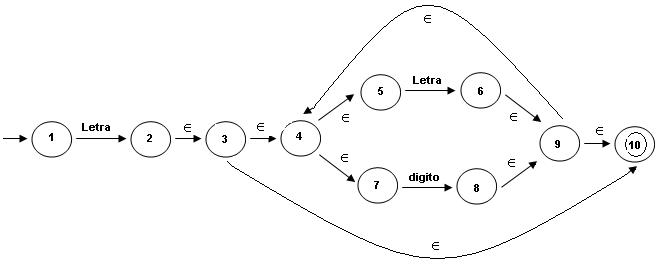
La eliminación de las transiciones ε implica construir cerraduras ε (las cuales son el conjunto de todos los estados que pueden alcanzar las transiciones ε desde un estado o estados.)

La eliminación de las transiciones múltiples en un carácter de entrada simple implica mantenerse al tanto del conjunto de estados que son alcanzables al igualar un carácter simple. Lo cual se denomina **construcción de subconjuntos**.

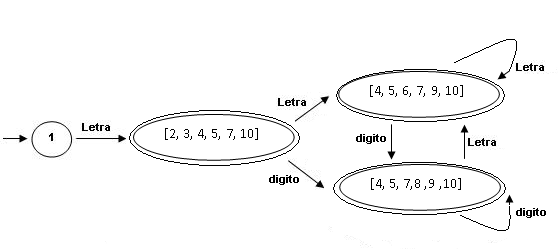
* Ejemplos de ER donde se muestre el proceso para construir el autómata finito mediante la ER.

Considere el AFND de la siguiente figura que corresponde a la construcción de Thompson para la ER: **Letra (Letra | digito)\***

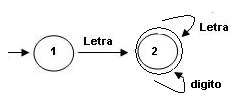
(La definición de construcción de Thompson para esta clase de ER está definida por la forma de la ilustración).



La construcción del subconjunto continúa como se explica a continuación. El estado de inicio es {1} =1.Existe una transición en **Letra** para {2}= [2, 3, 4, 5, 7, 10]. Desde este estado existe una transición en **Letra** para {6}= [4, 5, 6, 7, 9, 10] y una transición en **digito** para {8}= [4, 5, 7,8 ,9 ,10]. Finalmente, cada uno de estos estados también tiene transiciones en **Letra** y **digito**, ya sea hacia si mismos o hacia el otro, los estados que contienen el numero 10 se convierten en estados de aceptación. El AFD completo quedaría entonces así:



El cual se puede simplificar de la siguiente forma:



Construcción del autómata finito con transiciones ε

Si *r* es una expresión regular con n operadores y sin variables como operandos atómicos, existe un autómata finito no determinístico M con transiciones ε (AFND-ε) que acepta solamente aquellas cadenas que están en L(*r*). M tiene un estado final, no entran arcos al estado inicial y no salen arcos del estado final.

*r* puede ser una expresión sin operadores (φ, εo un símbolo) o con operadores (|, \*, o concatenación).

Si *r* no tiene operadores, entonces:



Para *r* = φel AFND-εes



Para *r* = εel AFND-εes



Para *r* = a (a ∈A) el AFND-εes

Si *r* tiene operadores, se dan tres casos dependiendo de la forma de *r*:

1) *r* = r1 | r2

Sean M1 = <E1, A, δ1, e01, {ef1}> y M2 = <E2, A, δ2, e02, {ef2}>, los autómatas correspondientes a r1 y r2.

Se construye un nuevo autómata M que une a estos dos autómatas M1 y M2 agregando un estado inicial e0 y un estado final ef0; M = < E1 ∪ E2 ∪{e0, ef0}, A, δ e0, {ef0}>. El estado inicial de M tiene transiciones εa los estados iniciales de M1 y M2; los estados finales de estos autómatas tienen transiciones al estado final del autómata M. 

2) *r* = r1r2

Sean M1 = <E1, A, δ1, e01, {ef1}> y M2 = <E2, A, δ2, e02, {ef2}>, los autómatas correspondientes a r1 y r2.

Se construye un nuevo autómata M, M = < E1 ∪E2, A, δ, e01, {ef2}> que tiene como estado inicial al estado inicial de M1 y como estado final al estado final de M2; tiene además un arco rotulado ε desde el estado final de M1 al estado inicial de M2.



3) *r* = r1\*

Sea M1 = <E1, A, δ1, e01, {ef1}> el autómata correspondiente a r1. Se construye un nuevo autómata M, M = < E1 ∪{e0, ef0}, A, δ e0, {ef0}>; y se agregan arcos rotulados εdesde e0 al estado inicial de M1 y al estado final de M, y desde el estado final de M1 al estado inicial de M1 y a ef0.



Ejemplo:

Construir el AFND-correspondiente a la siguiente expresión regular *r* = 00 | 0\*1

*r* es de la forma r1 | r2, donde r1 = 00 y r2 = 0\*1

r1 se puede expresar como r3r4, donde

r3 = 0



r4 = 0

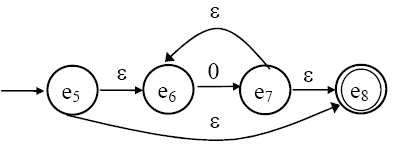


El autómata para r1 es

r2 se puede expresar como r5r6, donde



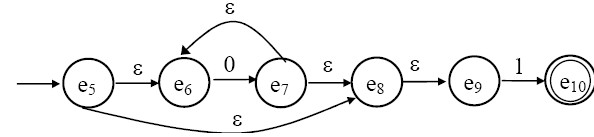
r5 = r7\* siendo r7 = 0



El autómata para r5 es



r6 = 1

El autómata para r2 es

El autómata correspondiente a *r* es

AFND-ε=<{e0, e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11}, {0, 1}, δ, e0, {e11}>, con δdefinida por el siguiente diagrama de transición de estados



Construcción del AFND correspondiente al AFND-del ejemplo anterior.

El conjunto de estados está formado por e0 (estado inicial) y por los estados e2, e4, e7 y e10 (estados importantes).

Los estados finales son e4 y e10 (existe un camino en el AFND-εcon transiciones a un estado final del AFND-ε).

El estado inicial es e0.

Transiciones para el estado e0: desde el estado e0 se pueden alcanzar con transiciones los estados e0, e1, e5, e6, e8 y e9. En el AFND-existe una transición de e1 a e2 con 0, una transición de e6 a e7 con 0 y una transición de e9 a e10 con 1. Entonces en el nuevo autómata hay una transición de e0 a e2 con 0, una transición de e0 a e7 con 0 y una transición de e0 a e10 con 1.

Transiciones para el estado e2: desde el estado e2 se pueden alcanzar con transiciones los estados e2 y e3. En el AFND-εexiste una transición de e3 a e4 con 0. Entonces en el nuevo autómata, hay una transición del estado e2 al estado e4 con 0.

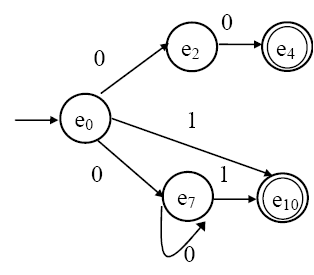
Transiciones para el estado e7: desde el estado e7 se pueden alcanzar con transiciones los estados e6, e7, e8 y e9. En el AFND-existe una transición de e6 a e7 con 0 y una transición de e9 a e10 con 1.

Entonces en el nuevo autómata, hay una transición del estado e7 al estado e7 con 0 y una transición del estado e7 al estado e10 con 1.

Transiciones para el estado e4: desde el estado e4 se pueden alcanzar con transiciones los estados e4 y e11. Como en el AFND-no existen transiciones sobre símbolos reales desde el estado e11, entonces no se agregan transiciones desde el estado e4 en el nuevo autómata.

Transiciones para el estado e10: desde el estado e10 se pueden alcanzar con transiciones los estados e10 y e11. Como en el AFND-no existen transiciones sobre símbolos reales desde el estado e11, entonces no se agregan transiciones desde el estado e10 en el nuevo autómata.

El autómata correspondiente sin transiciones se define AFND = <{e0, e2, e4, e7, e10}, {0, 1}, , e0, {e4, e10}>, con definida por el siguiente diagrama de transición de estados



Se puede observar que este autómata es no determinístico. Como ya se ha visto, es posible construir a partir del mismo el autómata finito determinístico equivalente que reconoce el mismo lenguaje.

Ejemplo 2:

Construir el AFND-correspondiente a la siguiente expresión regular *r* = (0|0\*)1

*r* es de la forma r1r2, donde r1 = 0|0\* y r2 = 1

r1 se puede expresar como r3|r4, donde

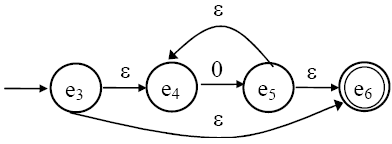


r3 = 0



r4 = r5\*  siendo r5 = 0

El autómata para r4 es



El autómata para r1 es



El autómata para r2 es

El autómata correspondiente a *r* es

AFND-=<{e0, e1, e2, e3, e4, e5, e6, e7, e8, e9}, {0, 1}, δ, e0, {e9}>, con δdefinida por el siguiente diagrama de transición de estados



Construcción del AFND correspondiente al AFND-anterior

El conjunto de estados está formado por e0 (estado inicial) y por los estados e2, e5 y e9 (estados importantes).

El único estado final es e9

El estado inicial es e0.

El autómata correspondiente sin transiciones se define AFND = <{e0, e2, e5, e9}, {0, 1}, δ, e0, {e9}>, con δdefinida por el siguiente diagrama de transición de estados



Se puede observar que este autómata es no determinístico. Como ya se ha visto, es posible construir a partir del mismo el autómata finito determinístico equivalente que reconoce el mismo lenguaje, que podría ser luego minimizado.

Otra posibilidad es simplificar la expresión regular primero para obtener directamente el AFD mínimo.

*r* = (0 | 0\*)1

(0 | 0\*)1 ≡( 0| 00\* | )1

≡(0(| 0\* ) | )1

≡ (0(| 00\* | ) | )1

≡(0(00\* | ) | )1

≡(00\* | )1

≡0\*1

Referencias

<http://html.rincondelvago.com/compiladores_automatas-finitos.html>

<http://trevinca.ei.uvigo.es/~formella/doc/talf05/talf/node30.html>