UNIVERSIDAD NACIONAL DE SAN AGUSTÍN

FACULTAD DE PRODUCCION Y SERVICIOS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

LABORATORIO TEORIA DE LA COMPUTACIÓN

Estudiante: Apaza Calsin Michael Secarlos

CUI: 20180564

Arequipa - Perú

2020

I OBJETIVOS

- Comprender las Expresiones Regulares (ER)
- Convertir Autómatas finitos (AF) a Expresiones Regulares (ER) y viceversa.

II EJERCICIOS PROPUESTOS

1. Ejercicio 1: (2pt)
   Crear un programa que defina una ER y también realizar su respectivo AF, que reconozca números enteros con signo.
2. Ejercicio 2: (3pt)
   Crear un programa que defina una ER y también realizar su respectivo AF, que reconozca una dirección IP.
3. Ejercicio 3: (3pt)
   Crear un programa que defina una ER y también realizar su respectivo AF, que dado un texto de entrada reconozca una url cuyo dominio termine en .com
4. Ejercicio 4: (3pt)
   Dado la siguiente ER: X(X|YZ*)*X, realizar su respectivo AF.
5. Ejercicio 4: (3pt)
   Dado la siguiente ER: letra(_+digito)+letra, realizar su respectivo AF.
6. Ejercicio 5: (3pt)
   Dado el siguiente AF, reconocer la expresión regular y hacer su codificación en Flex. Realizar la reducción de la expresión regular.

III SOLUCIÓN

Ejercicio 1:

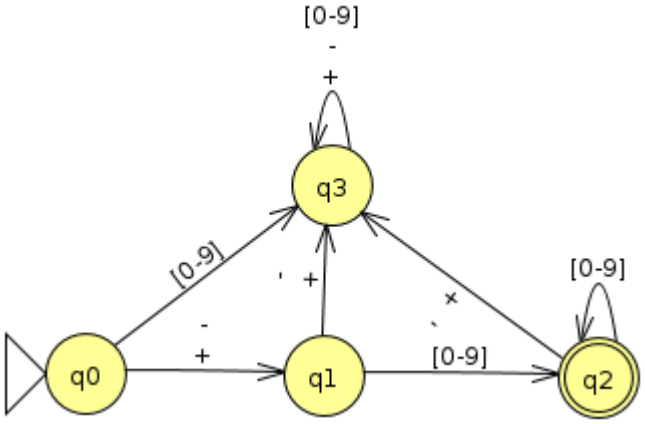
Al definir un número entero, necesitamos que este tenga al menos un dígito, y obligatoriamente un signo “+” o un “-”, entonces la expresión regular sería la siguiente:

```
("+"|"-")[0-9]+
```

Usando la misma para implementar el programa .l:

```
%{
%}
%%
("+"|"-")[0-9]+ {printf("Ha ingresado un numero entero con signo\n");}
%%
int yywrap(){};
int main()
{
    yywrap();
    printf("Ingrese un numero entero especificando su signo \n");
    yylex();
    return 0;
}
```

Y la implementación del autómata finito que lo representa sería el siguiente:



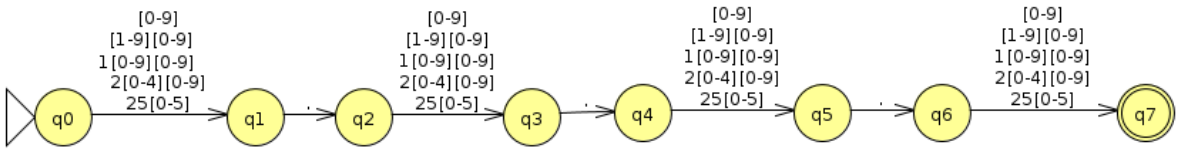
Que consta de:
Σ = {+, -, [0-9]} #Donde [0-9] alberga dígitos entre 0 y 9
Q = {q0, q1, q2, q3}
s = {q0}
F = {q2}

Se implementó un AFD para el ejercicio, siendo las transiciones hacia q3 entradas no válidas, de forma que cada entrada debe empezar si o si con "+" o "-", seguido de al menos un dígito para ser una cadena aceptable.

Ejercicio 2:
Las direcciones ips constan de 4 números desde 0 a 255, separados por puntos, entonces la expresión regular que la define es:
`^((25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9])(\.)){3}(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[0-9][0-9]|[0-9])$`
Que básicamente solo separa números desde 250 a 255, luego de 200 a 249, de 100 a 199, 10 a 99, y finalmente de 0 a 9, seguidas de un punto cada una. Para luego repetir esta 3 veces a través de **{3}** y finalmente solo esperar el mismo número definido con la misma expresión regular para los números de 0 a 255, con esto se implementó el siguiente programa flex:

```
%{
%}
%%
^((25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[0-9][0-9]|[0-9])(\.)){3}(25[0
-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[0-9])$ {printf("Ha
ingresado una direccion ip\n");}
%%
int yywrap(){};
int main()
{
    yywrap();
    printf("Ingrese una direccion ip valida.\n");
    yylex();
    return 0;
}
```

Y el autómata que define la expresión regular es:



Que consta de:

Σ = {., 25[0-5], 2[0-4][0-9], 1[0-9][0-9], [1-9][0-9], [0-9]} #Donde [x-y] son dígitos entre **x** y **y**

Q = {q0, q1, q2, q3, q5, q6, q7}

s = {q0}

F = {q7}

Al tener cantidades limitadas de repeticiones, crear un bucle no sería lo óptimo, por lo tanto se instancian varios estados, teniendo en cuenta que cada transición de un número seguido de un punto se repita solo 3 veces, y para el final termine con solo un número más.

Ejercicio 3:

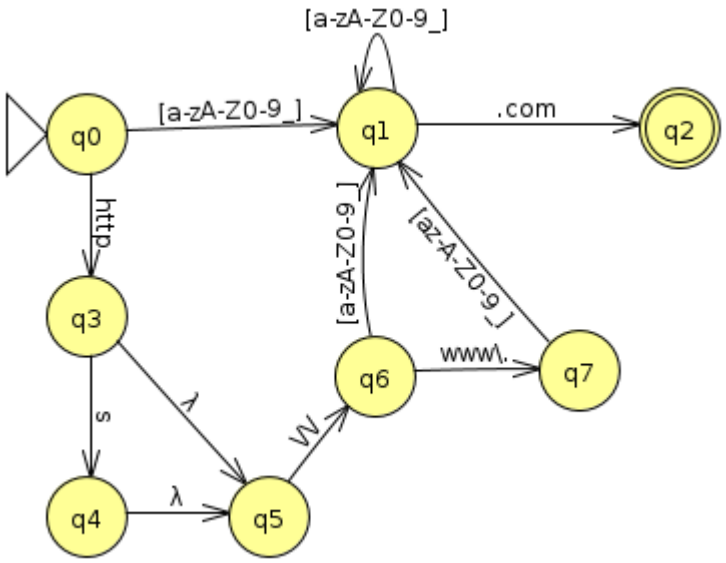
Las urls vienen de distintas formas, por ello se trato de abarcar cierta cantidad de ellas, a través de la siguiente expresión regular:

`[https?:\/\/[www\.]?]?[a-zA-Z0-9_]+\.com`

Con la cual se implementó el siguiente programa en flex:

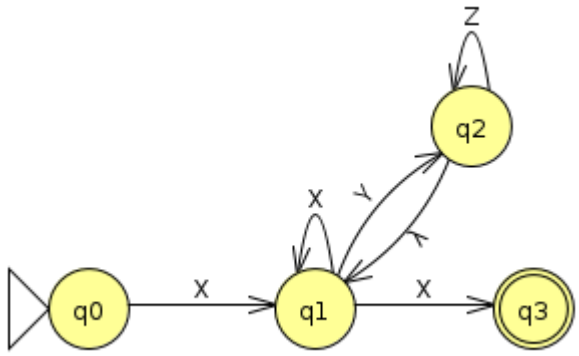
```
%{
%}
%%
[https?:\/\/[www\.]?]?[a-zA-Z0-9_]+\.com   {printf("Se ha
detectado una url que termina en .com\n");}
%%
int yywrap(){};
int main()
{
    printf("Ingrese un texto con una url que termine en .com\n");
    yylex();
    return 0;
}
```

Y definimos la expresión regular a través del siguiente autómata:



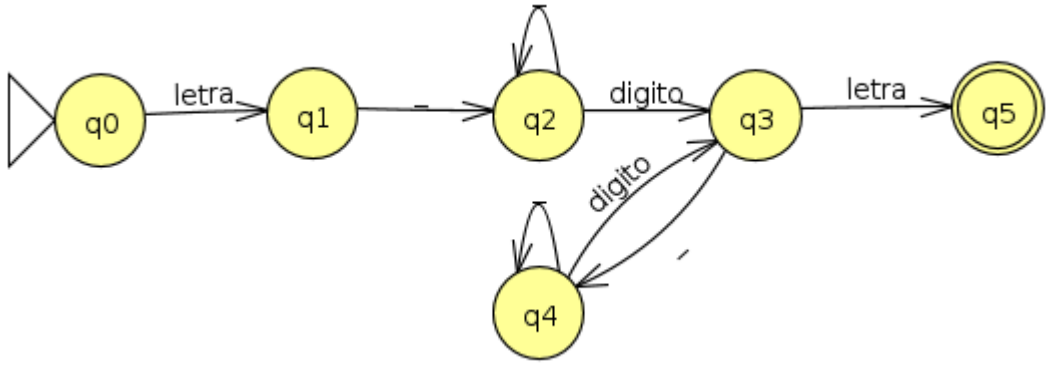
Que consta de:
Σ = {[a-zA-Z0-9], .com, http, s, //, www.} #Donde [x-y] son dígitos/letras entre **x** y **y**
Q = {q0, q1, q2, q3, q5, q6, q7}
s = {q0}
F = {q2}
Al añadir transiciones que pueden o no estar en la cadena gracias al símbolo **?**, de la expresión regular, se añadieron varios nodos al autómata, aunque aun así, este funciona tan solo con entradas como: algo.com.

Ejercicio 4:
Con la siguiente expresión regular:
ER: X(X|YZ*)*X,
Podemos definir el autómata como:



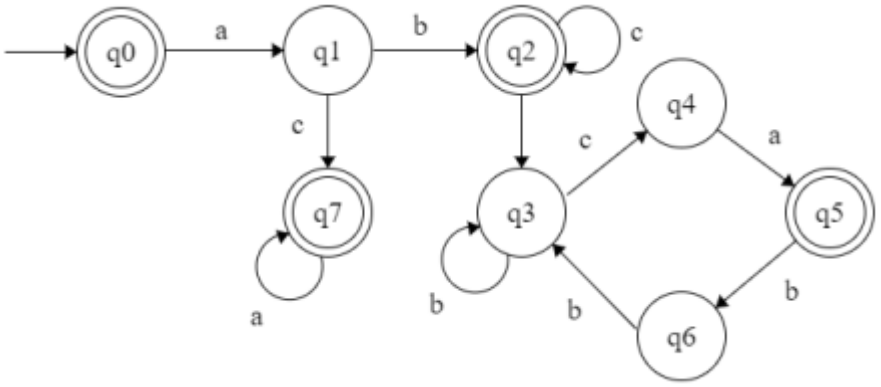
Σ = {X,Y,Z,λ} #Donde [x-y] son dígitos/letras entre **x** y **y**
Q = {q0, q1, q2, q3}
s = {q0}
F = {q3}
La expresión regular indica que se inicia con X, luego puede o no contener X o Y con varias o ninguna Z, es por este último caso(ninguna Z), que se retorna desde q2 a q1, con una transición de cadena vacía, para finalmente desde q1 ir al estado q3 con el caracter X.

Ejercicio 5:
Podemos representar la siguiente expresión regular:
ER: letra(_+digito)+letra
A través del siguiente autómata:



Empezamos con una letra, seguida de un subguión o más, luego un dígito y se puede o no repetir desde el primer subguión, para finalmente llegar a la letra.

Ejercicio 6:



Partida y final	Estados que atraviesa	Expresión regular
q0 - q2	q0, q1, q2	abc*

Partida y final	Estados que atraviesa	Expresión regular
q0 - q5	q0, q1, q2,q3,q4,q5,q6	abc*b*ca(bbb*ca)*

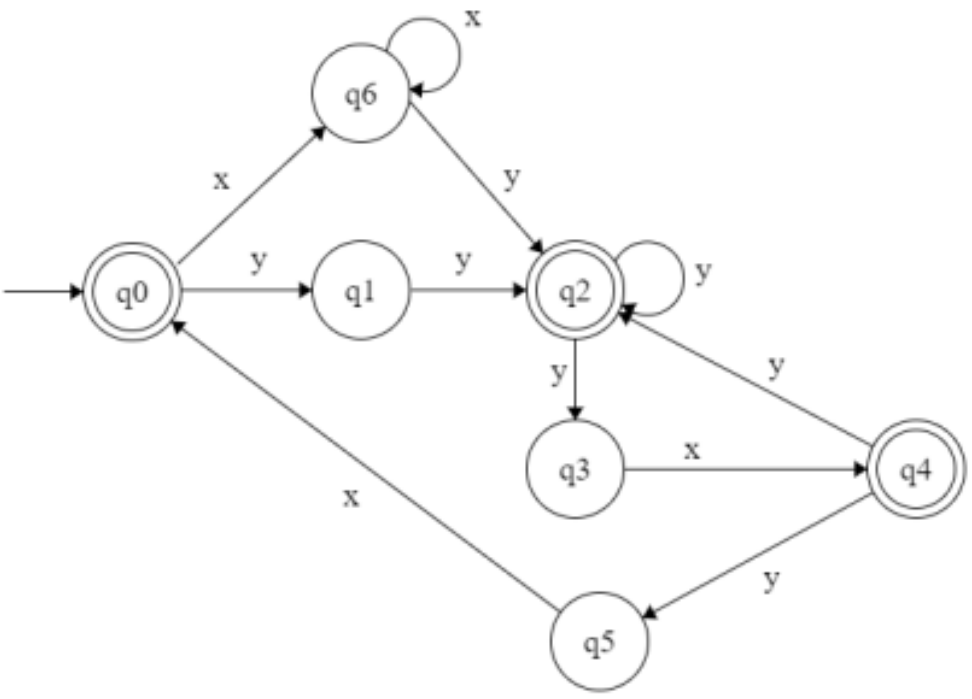
Partida y final	Estados que atraviesa	Expresión regular
q0 - q7	q0, q1, q7	ac*a

Entonces podemos definir el autómata como la unión de las expresiones regulares:
ER: abc* + abc*b*ca(bbb*ca)* + ac*a

Reduciendo esta expresión podemos obtener:
a(bc* + bc*b*ca(bbb*ca)* + c*a)
Y la el programa en flex se veria como:

```
%{
%}
%%
abc*                 {printf("Primer caso");}
abc*b*ca(bbb*ca)*    {printf("Segundo caso");}
ac*a                 {printf("Tercer caso");}
%%
int yywrap(){};
int main()
{
    printf("Ingrese una cadena\n");
    yylex();
    return 0;
}
```

Ejercicio 7:



Partida y final	Estados que atraviesa	Expresión regular
q0 - q2	q0, q1, q2	yyy*
q0 - q2	q0, q1, q2, q3, q4	yyy*yxyy*
q0 - q2	q0, q1, q2, q3, q4, q5	yyy*yx(yy*yx)yxyyy*
q0 - q2	q0, q1, q2, q3, q4, q5, q6	yyy*yx(yy*yx)yxxx*yy*

Partida y final	Estados que atraviesa	Expresión regular
q0 - q4	q0, q1, q2, q3, q4	yyy*yx
q0 - q4	q0, q1, q2, q3, q4, q5	yyy*yx(yy*yx)*yx
q0 - q4	q0, q1, q2, q3, q4, q6	yyy*yx(yy*yx)yxxx*yy*yx(yy*yx)*
q0 - q4	q0, q1, q2, q3, q4, q5, q6	yyy*yx(yy*yx)yxxx*yy*

En el programa flex tenemos:

```
%{
%}
%%
yyy*                              {printf("Primer caso\n")}
yyy*yxyy*                         {printf("Segundo caso\n")}
yyy*yx(yy*yx)yxyyy*               {printf("Tercer caso\n")}
yyy*yx(yy*yx)yxxx*yy*             {printf("Cuarto caso\n")}
yyy*yx                            {printf("Quinto caso\n")}
yyy*yx(yy*yx)*yx                  {printf("Sexto caso\n")}
yyy*yx(yy*yx)yxxx*yy*yx(yy*yx)*   {printf("Septimo caso\n")}
yyy*yx(yy*yx)yxxx*yy*             {printf("Octavo caso\n")}
%%
int yywrap(){};
int main()
{
    printf("Ingrese una cadena\n");
    yylex();
    return 0;
}
```

Entonces podemos expresar la solución como:
yyy* + yyy*yxyy* + yyy*yx(yy*yx)yxyyy* + yyy*yx(yy*yx)yxxx*yy* + yyy*yx + yyy*yx(yy*yx)*yx + yyy*yx(yy*yx)yxxx*yy*yx(yy*yx)* + yyy*yx(yy*yx)yxxx*yy*
Reduciendo:
yy(y* + y*yxyy* + y*yx(yy*yx)yxyyy* + y*yx(yy*yx)yxxx*yy* + y*yx + y*yx(yy*yx)*yx + y*yx(yy*yx)yxxx*yy*yx(yy*yx)* + y*yx(yy*yx)yxxx*yy*)