Ejercicios Jorge Camacho

gercicio 3.3.1

b) C++

ii) las constantes numericas se dividen en 3 casos, con las siguientes reglas:

- constantes, enteros decimales

las constantes octales son una secuencia de digitos que comienza con cero

las hexadecimales son secuencia de digitos precedida por 0x o 0X.

iii) un identificador es una secuencia larga de tamaño arbitrario de letras y digitos, el primer caracter es una letra. y se distingue de mayúsculas y minúsculas

i) conjunto de caracteres es una combinacion de letras numeros y algunos simbolos como +, -, *, >, <, &&, == etc.

f) LISP

ii) Soporta dos tipos de datos numéricos enteros y números de punto flotante
Se definen de la misma forma que en C++ y además se pueden expresar en notación exponencial con E al final del número, donde representa 10^2.

iii) En LISP los identificadores se toman como los propios símbolos.

iv) El alfabeto Unicode

g) SQL permite nombres que consisten en un solo identificador o multiples identificadores. Los componentes de un nombre de varias partes deben estar separados por puntos (".").
Las partes iniciales de un nombre de multiples partes actúan como calificadores que afectan el contexto dentro del cual se interpreta el identificador final

ii) La forma lexica numerica es igual a los demas lenguajes

i) Unicode

a) C

i) Utiliza el mismo que C++

ii) es igual que en C++

iii) es igual que C++

c) C#

i) consiste en los caracteres numericos y especiales

iii) Deben empezar por una letra @, -, @, y luego ir por una secuencia de letras o digitos donde "_" o "@" es una letra

ii) es igual a C++

d) fortran

i) Consiste de 26 letras minusculas 26 mayúsculas, los digitos de 0 a 9 y espacios y los secuencias como \n o \t

iii) los identificadores no pueden exceder una longitud de 31, el primer caracter debe ser una letra y el resto puede ser una secuencia de letras, digitos o espacios.

ii) es igual a C++

e) Java

i) Unicode

iii) son de longitud ilimitada (empiezan de letras y números donde el primero debe ser una letra

ii) es igual al de LISP

Ejercicio 3.2.2

d) cadena de a y b que solo tiene b's

e) cadena de a y b que tiene un número par de a y b

Ejercicio 3.3.4

. (s|S)(c|E)(L|l) (e|E)
(c|C) (t|T)

Ejercicio 3.3.5

a) otro → [b,c,d,f,g,h,j,k,l,m,n,ñ,p,q,r,s,t,v,w,x,y,z]

Expresión → otro* a (otro|a)* c
(otro|e)* i (otro|i)*
o (otro|o)* u (otro|u)*

b) a* b* ... z*

c) /*(.|("*/"))**/

Donde . es cualquier caracter excepto *

d) Solución → 0(A?0?1(A0?1|01)*
*?0?1A0?

A → 0?2(02)* esto para {1,2,3,4}

Ejercicio 3.3.7

- \" \\ "

Ejercicio 3.3.8

Usando la unión de todos los caracteres distintos de x siendo [^x] el complemento de x.

Ejercicio 3.3.10

a) la expresión [^s] identifica el complemento de s y la expresión ^s significa el inicio de una linea.

b) podemos cambiar el ^ y $ expresion \n la cual se puede usar para hacer un salto de linea

Ejercicio 3.3.11

Sea Σ el alfabeto de la terminal unix s\n.

a) S:= " concatenacion de caracteres $a_1 a_2 a_3 \dots a_n$ "

podemos reemplazar "S" por la concatenacion de todos los elementos de S $a_1 a_2 a_3 \dots a_n$.

b) \c, se puede reemplazar por la expresion regular C.

c) *: se puede reemplazar por la expresion [A-z a-z]*

d) ?: se puede reemplazar por la expresion [A-z a-z]

e) [S]: se puede remplazar por la unión de los caracteres de S.

Ejercicio 3.3.12

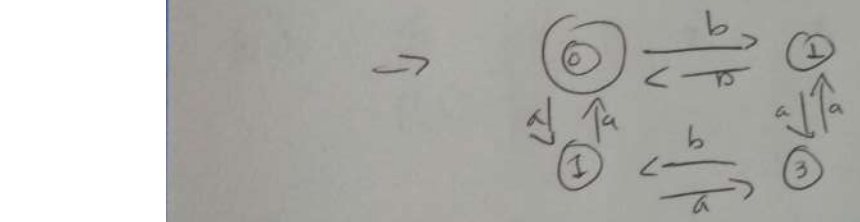
$- : [A-z\ a-z\ 0-9]$

$\% : [A-z\ a-z\ 0-9]^*$

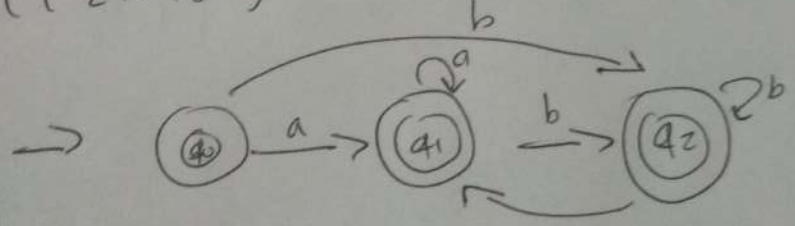
$(\varepsilon \mid \% \mid -)^*$

Ejercicio 3.4.1

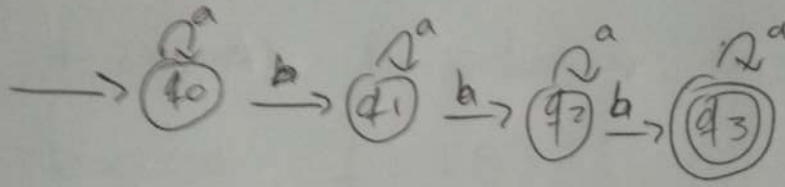
1. $(aa|bb)^*((ab|ba)(ab|bb)^*(ab|ba)(aa|bb)^*)^*$



2. $((\varepsilon|a)b^*)^*$



3. $a^* ba^* ba^* ba^*$



Ejercicio 3.4.3

a)

S	1	2	3	4	5	6	7	8	9
f(s)	6	0	1	2	3	1	5	1	0

b)

S	1	2	3	4	5
f(s)	0	2	3	4	5

c)

S	1	2	3	4	5	6	7
f(s)	0	0	0	1	1	2	3

Ejercicio 3.4.4

Hipótesis de inducción: Al inicio de la iteración $t = f(s)$

Caso base: Antes de la primera iteración se realiza la asignación $t=0$ y $f(0)=0$, lo cual se cumple por definición de la función de fallo

Caso inductivo: Suponemos que se cumple $t=f(k)$ antes de la iteración (k). Ahora que dentro del for, primero se verifica $b_1 b_2 \dots b_{f(k)} \dots b_{f(k)+1}$ el cual es el prefijo mas largo de $b_1 b_2 \dots b_k b_{k+1}$ que podría ser sufijo, esto se realiza en el while si no lo es, busca el siguiente prefijo mas largo que podría ser sufijo de $b_1 b_2 \dots b_{k+1}$, lo cual es $b_1 b_2 \dots b_{f(f(k))} b_{f(f(k))+1}$

luego cuando sale del while, habrá encontrado el prefijo más grande de la cadena hasta k+1 que también es sufijo también) Se asignará dentro del if F(k+1), si no es posible ningún el prefijo y sufijo hasta k+1, por lo tanto F(k+1)=0

así concluimos que la función de fallo se realiza correctamente ☐

Ejercicio 3.4.5

El while se ejecuta a lo mucho n veces, ya que t<s<n y el t se va reduciendo luego las líneas 1 y 2 se ejecutan 1 vez y las líneas 5 y 6 y 8 del for se ejecutan n veces. Siendo los costos lineales el algoritmo es $\Theta(n)$ en particular Será $O(n)$

Ejercicio 3.4.9

a) $L_{on}(s_n) = L_n(s_{n-1}) + L_n(s_{n-2})$

b) $S_6 = S_5 S_4 = abaababa$

Factor de Fallo

S	1	2	3	4	5	6	7	8
$F(S)$	0	0	1	1	2	3	2	3

c) $S_7 = S_6 S_5 = abaababaabaab$

Factor de Fallo

S	1	2	3	4	5	6	7	8	9	10	11	12	13
$F(S)$	0	0	1	1	2	3	2	3	4	5	6	4	5

d) la función de fallo para F(1) por definición es cero, y como $S_1 \neq S_2$ entonces $f(2) = 0 = f(1)$

luego para $2 < j \leq |S_n|$

vemos por qué se cumple $f(j) = j - |S_{k-1}|$ con $|S_k| \leq j+1$

Sea S_{m-j} una subcadena de S_m con $m \leq n$

con $|S_{m-1}| < |S_{m-j}| \leq |S_m|$.

luego como S_k es el fibonacci que antes se acerca S_{m-j+1} o es igual

como S_{m-j+1} está entre $|S_{m-1}| < |S_{m-j+1}| < |S_{m+1}|$

si $S_k = S_{m-j+1}$ quiere decir que S_{m-j+1} es S_m o $|S_{m-j+1}| > |S_m|$