Primer Conjunto de Ejercicios - Compiladores

David Ricardo Pedraza Silva

November 25, 2021

Ejercicio 3.3.1

Lenguaje de Programación C

- Caracteres: ascii, UFT-8 en algunos compiladores.
- Enteros: números comunes como se les entiende usualmente. Hay soporte para números octales (empiezan pro 0) y números hexadecimales (empiezan por 0x) que incluyen como dígitos a las letras de la A (o a) a la F (o f). Si termina en l o L, pasa a ser de tipo long.

  Flotantes: constan de un entero, un punto, y otro entero. En ocasiones pueden contar con exponentes. Esto se escribe como un E seguida de un signo y un entero.
- Identificadores: no pueden empezar por un número, incluyen caracteres alfanuméricos y guiones bajos. Son sensibles a la diferencia entre minúsculas y mayúsculas.

Lenguaje de Programación C++

- Caracteres: igual que en C.
- Numeros: igual que en C.
- Identificadores: igual que en C.

Lenguaje C#

- Caracteres: UTF-16 (Por Microsoft), Unicode.
- Igual que en C, pero pueden terminar en f, o m. estos indican flotantes o decimales.
- Identificadores: Igual que en C.

Lenguaje Fortran

- Alfabeto inglés con todos sus signos de puntuación, números comunes y corrientes, así como símbolos adicionales ($,&,<,>,+,=,_,/).
- Digitos con $+ -$ al principio de forma opcional en el caso de los enteros. En el caso de los flotantes e ssimilar a C: una parte entera, otra fraccionaria, y opcionalmente un exponente.
- No mas de 31 caracteres. No pueden comenzar por _ o un número. Por lo demás pueden usar tanto números, como guiones bajos, como letras.

Java

- Igual que en $C\#$
- Igual que en $C++$
- Igual que en C

Lisp

- Unicode
- Secuecnias de dígitos como se esperarían normalmente. Números arabigos en el caso de enteros, caso contrario un entero seguido de punto y un fraccionario (una cadena arbitraria de números arábigos)
- Una secuencia de numeros y letas.

SQL

- Unicode, UFT-8
- enteros de no más de 128 dígitos y reales como en C
- Como en C, de no más de 128 caracteres.

Ejercicio 3.3.2

d. Lenguajes en el alfabeto $\{a, b\}$ con exactamente tres b's.

e. El lenguaje sobre $\{a, b\}$ con un número par de a's y b's.

Ejercicio 3.3.4

$$select \rightarrow (S|s)(E|e)(L|l)(E|e)(C|c)(T|t)$$

Ejercicio 3.3.5

A.

$$\begin{aligned} &S \to \Sigma - \{a, e, i, o, u\} \\ &T \to S^* a(S|a)^* e(S|e)^* i(S|i)^* o(S|o)^* u(S|u)^* \end{aligned} \tag{1}$$

B.

$$S \to a^* \cdots z^*$$

C.

$$\begin{aligned} &S \to [a - zA - Z] \\ &T \to /^*(S|"^*/")^* S(S)^* * / \end{aligned} \tag{2}$$

D.

$$S \to [\wedge([0-9]^*0[0-9]^*0[0-9]^*| \cdots [0-9]^*9[0-9]^*9[0-9]^*)^*]$$

E.

$$S \to ([0-9]^*0[0-9]^*0[0-9]^*| \cdots [0-9]^*9[0-9]^*9[0-9]^*)^*$$

F.

$$S \to (aa|bb)^*((ab|ba)(aa|bb)^*(ab|ba)(aa|bb)^*)^*$$

G.

$$S \to [kqrbnp0-8]?[-x][kqrbnp0-8]?$$

H.

$$S \to b^*(ab?)^*$$

I.

$$S \to b^*a^*b?a^*$$

Ejercicio 3.3.7

\" \\

Ejercicio 3.3.8 Suponga el lenguaje regular $L \subset \Sigma$. Existe un autómata cone estados Q y estados de aceptación F que acepte a L. Para aceptar a su complemento basta con tomar el mismo autómata, pero tomando como estados de aceptación a $Q - F$. Luego $\Sigma - L$ es un lenguaje regular, por lo que existe una expresión regular (de las cláicas, sin ningún $\wedge$) representa a este lenguaje. Como la concatenación de lenguajes regulares es regular, es claro que cualquier expresión regular con $\wedge$ en sus entradas tiene un equivalente sin $\wedge$.

3.3.10

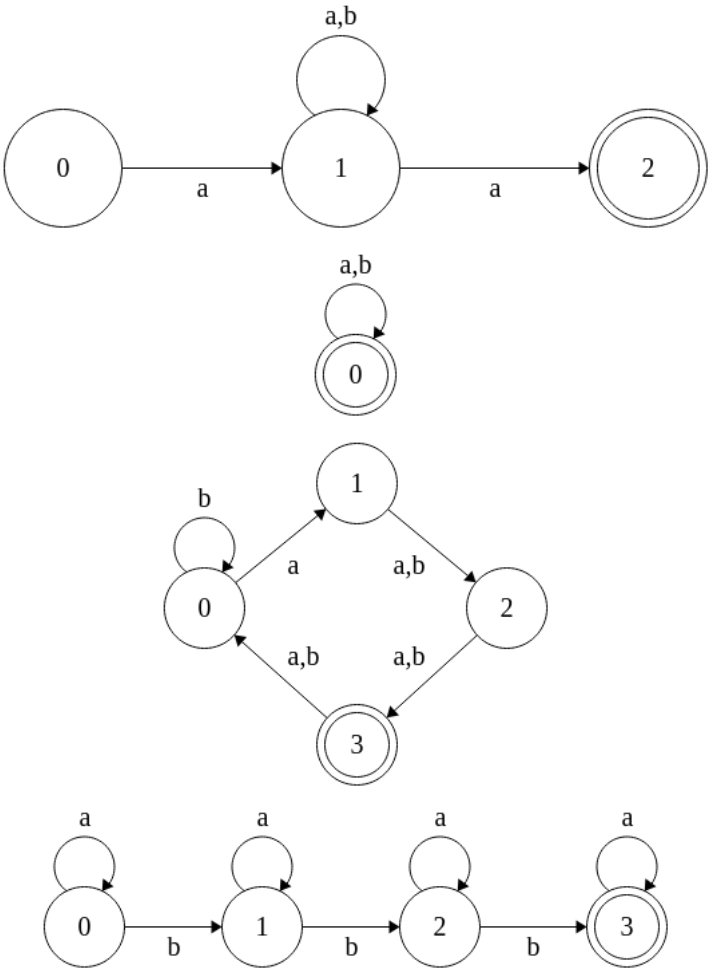
A.

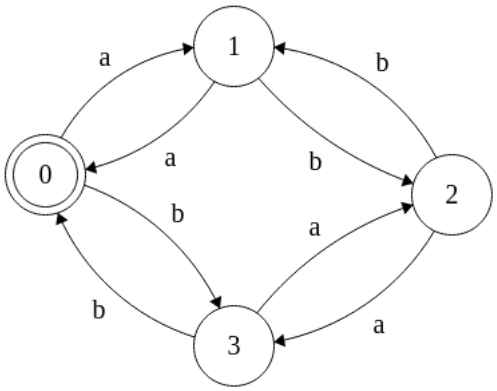
Si $\wedge$ está ala derecha de un paréntesis cuadrado, es complemento de algo. Caso contrario es inicio de cadena.

3.3.11 EL primero y segundo caso se quedan igual. * se reemplaza por .*, ? se reemplaza por '.' y $[s]$ por $[s]$ o, en el caso clásico, si $s = s_1 \cdots s_n$, entonces $(s_1 \cdots s_n)$

3.3.12 _ se reemplaza por '.', % se reemplaza por '.*' y y se puede reemplazar e por $.n$.

3.4.1





3.4.3

A.

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

B.

x	1	2	3	4	5	6
f(x)	0	1	2	3	4	5

C.

x	1	2	3	4	5	6	7
f(x)	0	0	0	1	1	2	3

3.4.4 Es claro que $f(1) = 0$, por lo que se cumple para este caso particular. Suponga que se cumple para n. Es decir: $f(n)$ es el índice más grande para el que $b_1 \cdots b_{f(n)}$ es tanto prefijo como subfijo propio. Esto es: $b_1 \cdots b_{f(n)} = b_{n-f(n)} \cdots b_{f(n)}$. t empieza como $t = f(n)$. Si $b_{t+1} = b_{f(n)+1}$ entonces $f(n+1) = f(n)+1$. De lo contrario hay que encontrar un prefijo que sea subfijo tan grande como sea posible: así $t = f(f(n))$. Claramente este procedimiento debe llegar a cero y se detiene cuando $b_{t+1} = b_{n+1}$, en cuyo caso $t + 1$ será el índice del prefijo más grande: es decir $f(n + 1) = t + 1$. Por lo que el algoritmo sirve para todo natural.

3.4.5 Cuando se asigna $t = f(t)$, t disminuye así sea en 1 unidad. EL número de veces que peude disminuir de esta forma es $s - 1$. En la siguiente operación (para s+1) t sólo podrá aumentar, y el bucle se ejecuta 0 veces, si tomamos el peor caso para s. Para $s + 2$ sólo podrá disminuir en 1 o aumentar 1, y así sucesivamente. POr este motivo la cantidad total de iteraciones para $t = f(t)$ es $s + (n - s)$, a los sumo n veces.

3.4.6 Para ababaab verdadero, para abababbaa es falso.

3.4.9

A.

El numero de fibonacci f_n

B. ver archivo colab

C. ver archivo colab

D.