



Ejercicios descritos en la semana 12 del libro Engineering a Compiler (2nd Ed 2012) de Cooper

1.

a)  $\Sigma = \{a, b\}$

$Q = \{S_0, S_1, S_2, F\}$

b)

$\Sigma = \{0, 1\}$

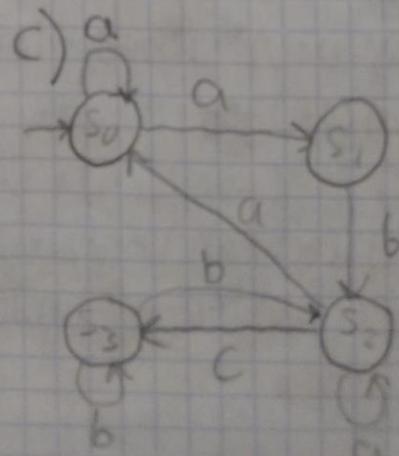
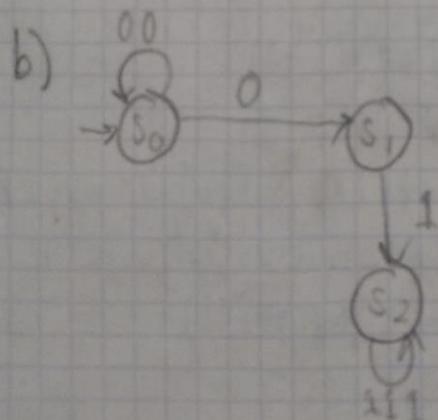
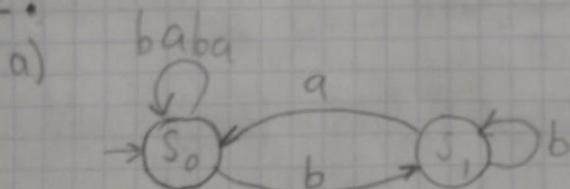
$Q = \{S_0, S_1, S_2, S_3, S_4, F\}$

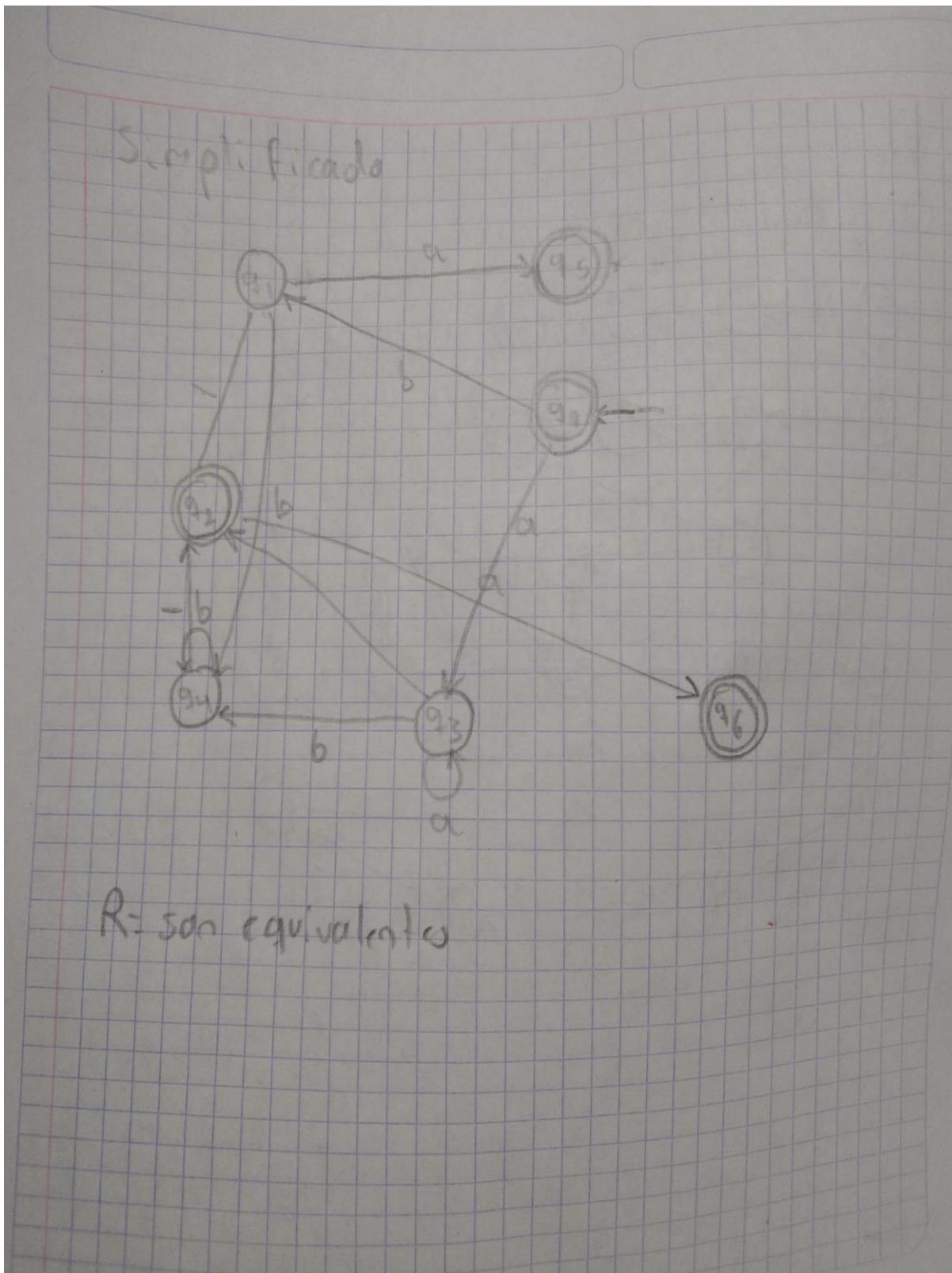
c)

$\Sigma = \{a, b\}$

$Q = \{S_0, S_1, S_2, S_3, S_4, S_5, S_6, F\}$

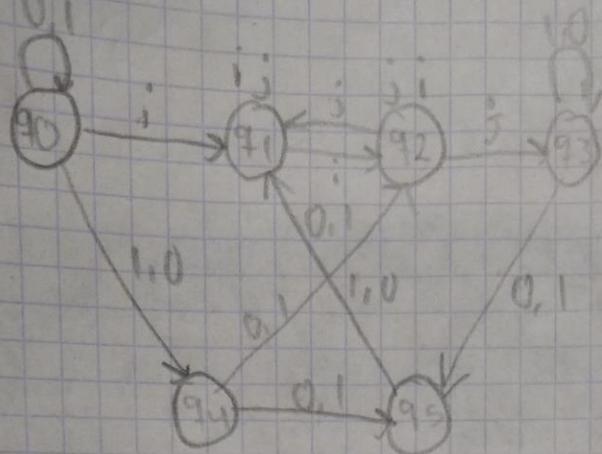
2.







3.



4.

a)  $E.R = (0 - q^*) \setminus d^* + (0 - q) \setminus H^*$

b)  $E.R = (0 - q) \setminus W^*$

c)  $E.R = (0 - q) \setminus \cdot d^* + (0 - q) \setminus (0 \cdot d)^* + (0 - q) \setminus (0 \cdot 0d)^*$

5.

a)  $(0+0)^* 1 (0+0) + (1+1)^* 1 (1+1) (1+1)$

b)  $(0+1)^* 1 (0+1)^* + (0+1)^* 1^* (0+1) (0+1)$

c)  $L = \{abcde\mid f, g\} + \{hi\mid klmn\} + \{opqrstu\} + \{vwxyz\}$

d)  $(xyzwyz)^* + (x) \cdot (y)^* + (xy) + (z) + (wy)^* + (wxyz)^*$

e)  $w = (0 - q)^* + (1, 3) \setminus d \cup (+ - ^* 1) + (( )) + ^{11}$

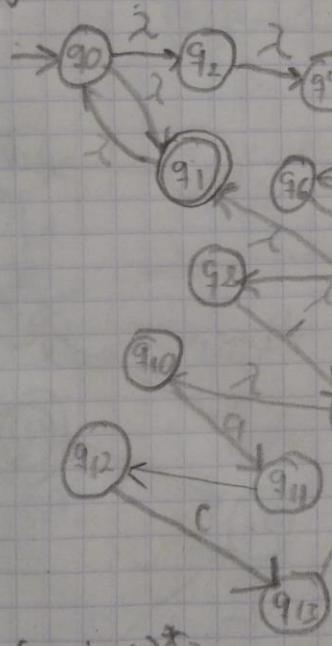


6.

- a)  $(11 \wedge [ ])^*$
- b)  $(/*) \rightarrow (2)$
- c)  $L = \Sigma A, \Sigma \bar{B} + \Sigma \bar{A}, \Sigma \bar{B}$
- d)  $(0 \cdot q) \setminus d^*$

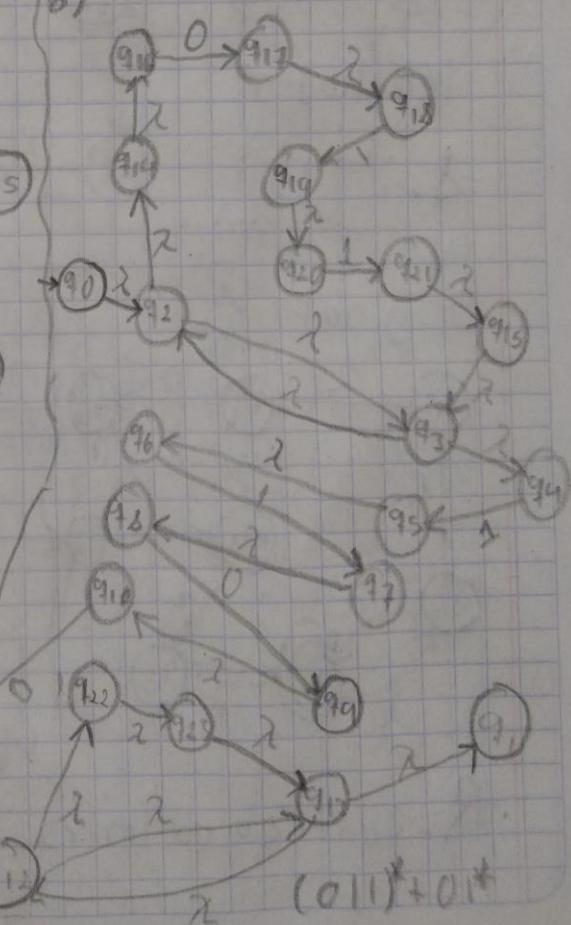
7.

a)

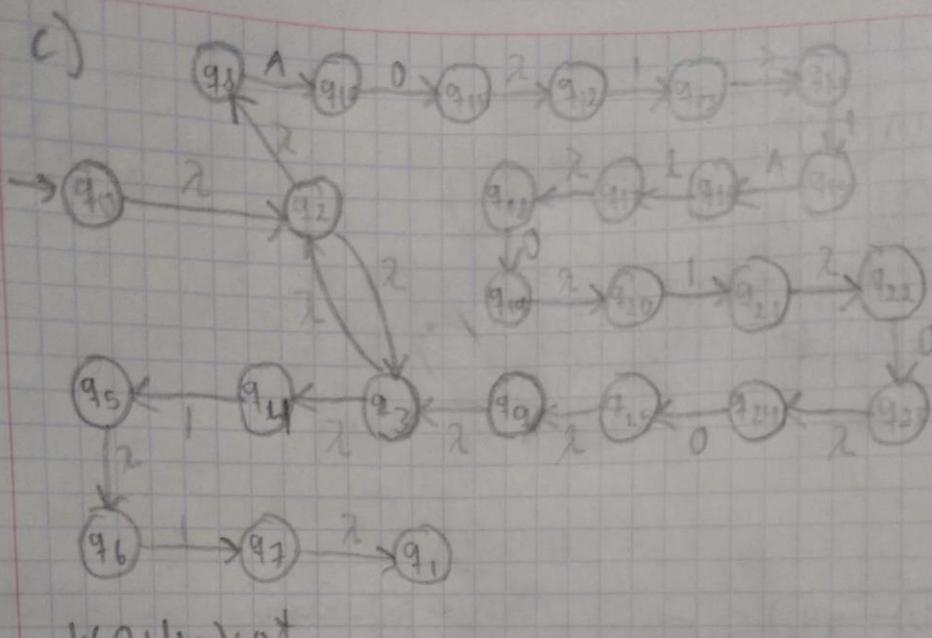


$$(a, b, c)^* \lambda$$

b)



$$(011)^* + 01^*$$

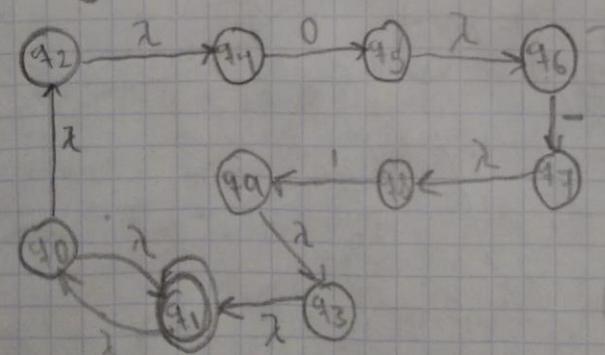


$$1(01110) + 0^*$$

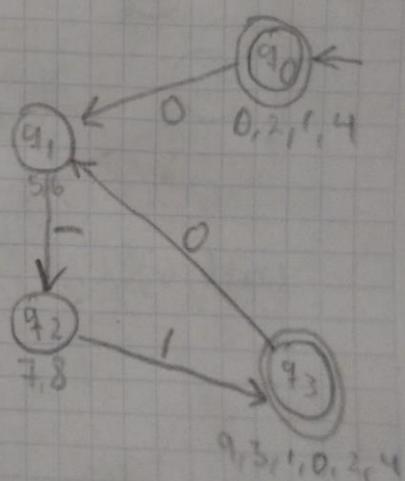
d.

a)

Original



Minimizado

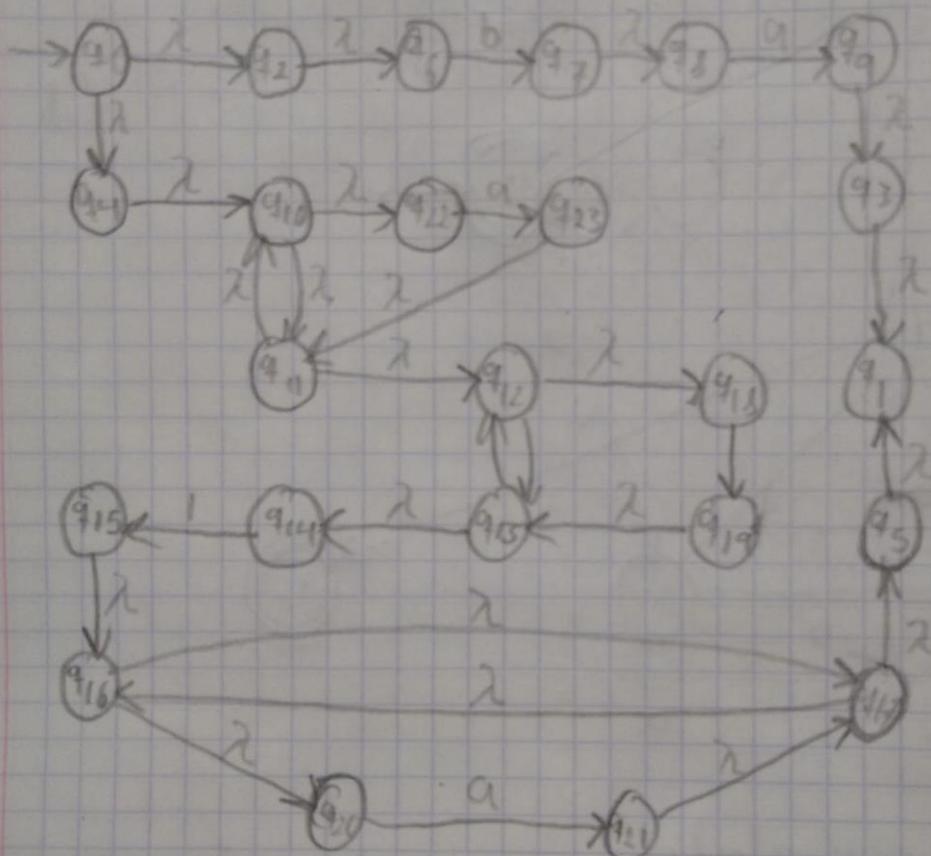


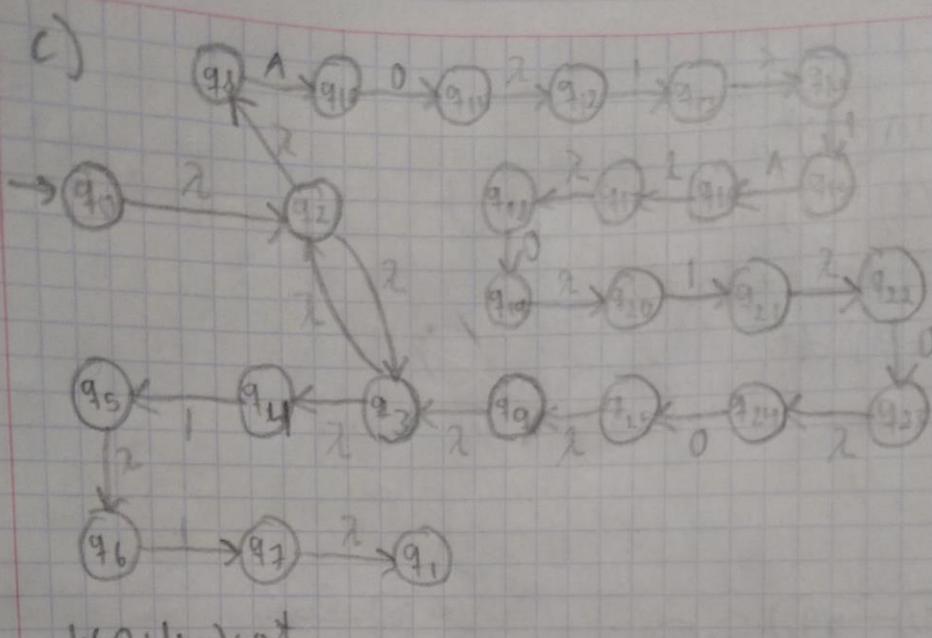
R = Son equivalentes



b)

Original



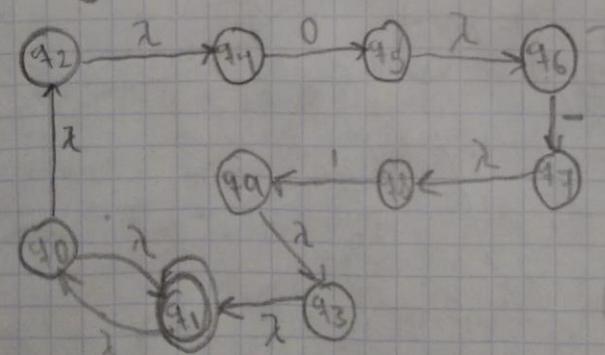


$$1(01110) + 0^*$$

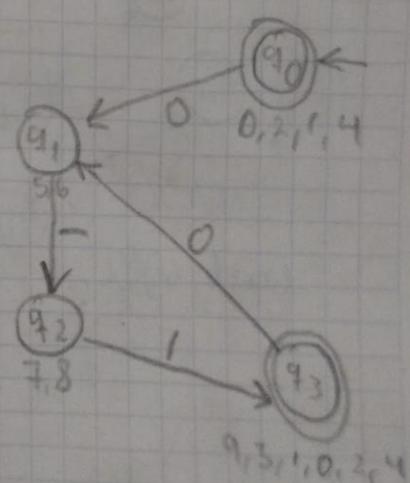
d.

a)

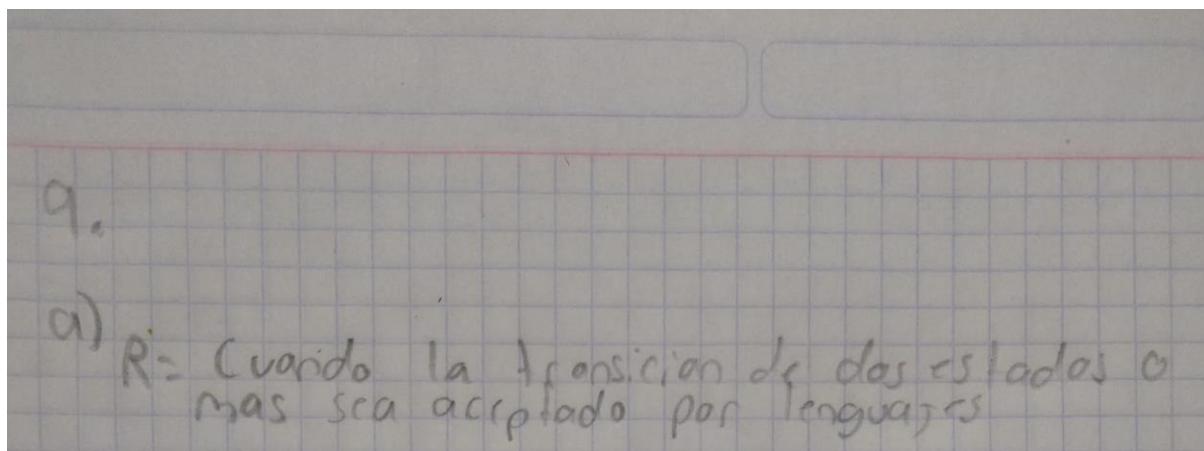
Original



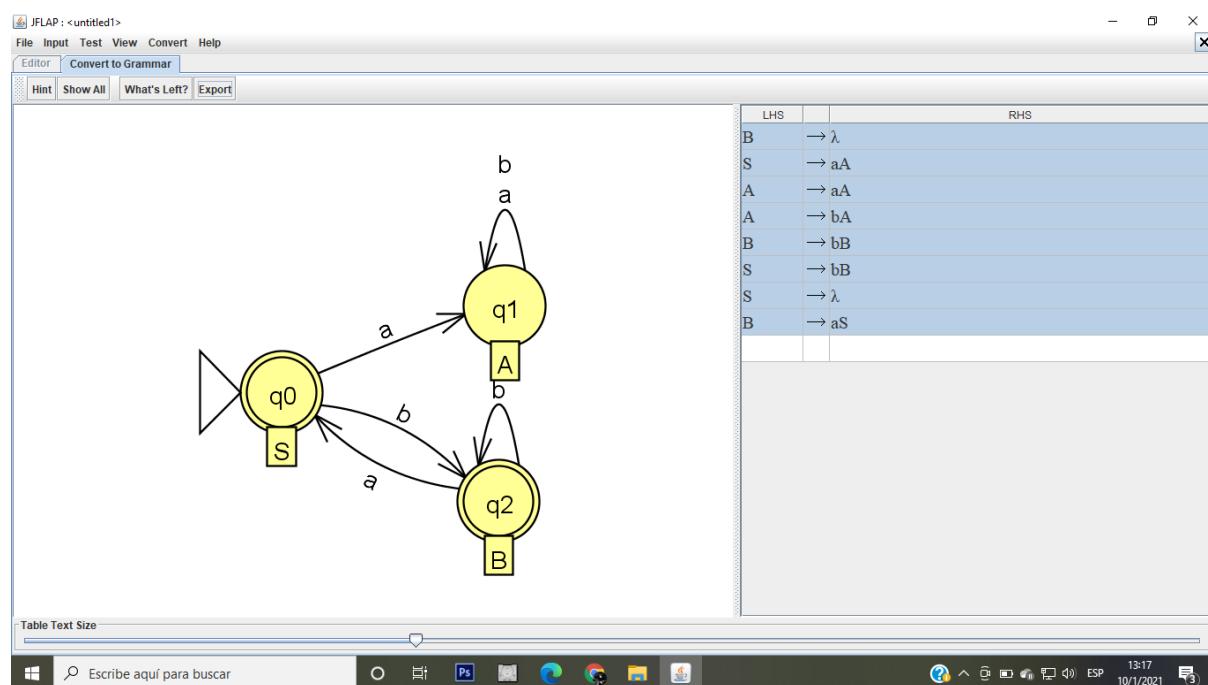
Minimizado



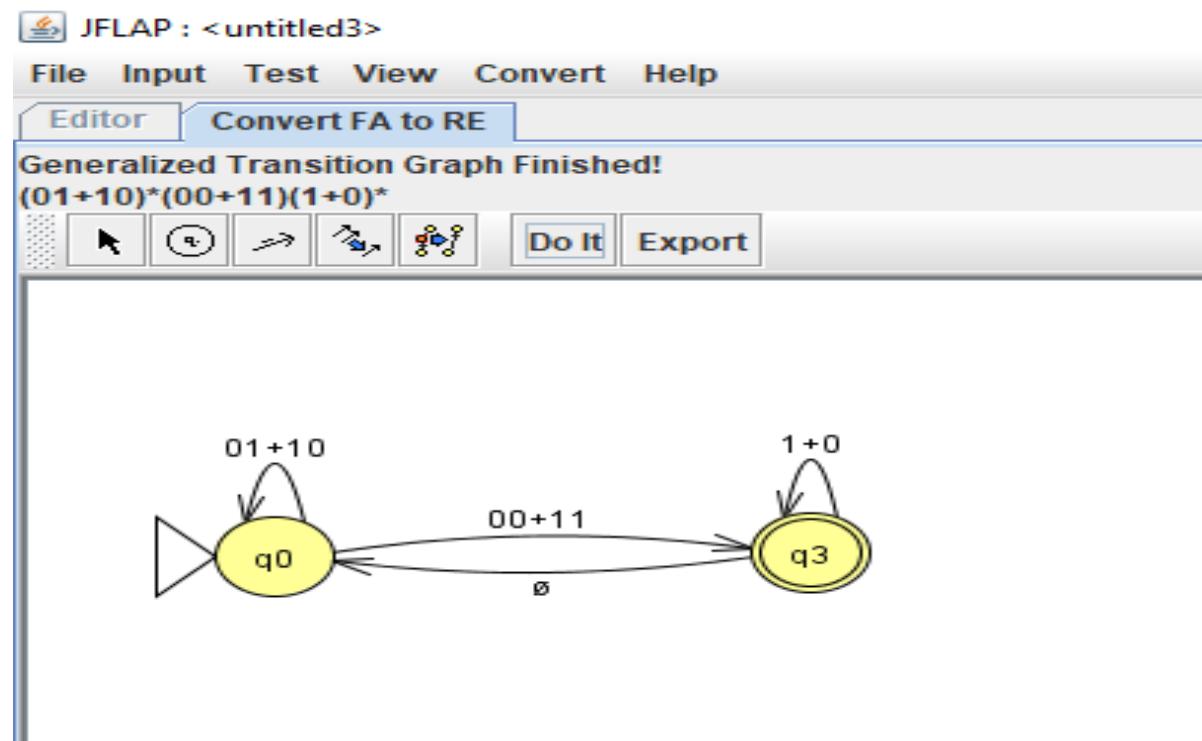
R = Son equivalentes



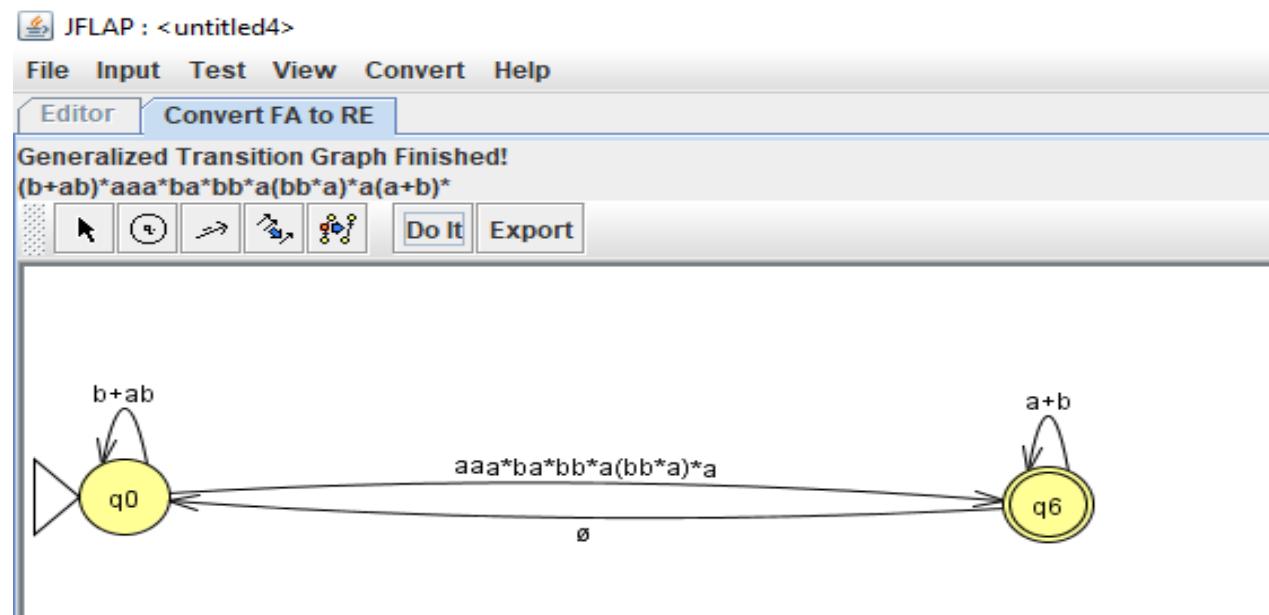
a)



b)



c)





1.

Solo ocupamos un subconjunto de la sintaxis de la E.R.

Quantity quantifiers: \*, +, ?

Choose : |

Group : ( )

$$P \rightarrow P | P$$

$$| PP$$

$$| (P) Q$$

$$| A$$

$$B \rightarrow B \text{ char}$$

$$| B [S]$$

$$| \text{char}$$

$$| [S]$$

$$A \rightarrow A B Q$$

$$\rightarrow B$$

$$S \rightarrow S \text{ char}$$

$$\rightarrow \text{char}$$

$$Q \rightarrow *$$

$$| ?$$

$$| +$$

$$| E$$



2.

$S \rightarrow <\text{word}> := P$

$P \rightarrow P \sqcup A$

$IA$

$A \rightarrow A "word"$

$IA <\text{word}>$

$I "word"$

$I <\text{word}>$

3. "Una gramática en la que cada oración puede tener un árbol gramatical único por inferencia del extremo izquierdo".

"A grammar in which each sentence can get a unique grammar tree by Far-left inference".



4. Primero convertiremos la recursividad izquierda

$R \rightarrow Rbc$  en recursión por la derecha. La sintaxis convertida es la siguiente

$$L \rightarrow R.a$$

$$I \ Q.ba$$

$$R \rightarrow abaR'$$

$$I \ cababR'$$

$$R' \rightarrow b.c.R'$$

$$I \epsilon$$

$$L \rightarrow R.a$$

$$I \ Q.ba$$

$$R \rightarrow abaR'$$

$$I cababR'$$

$$R' \rightarrow b.c.R'$$

$$I \epsilon$$

$$Q \rightarrow b.b.c \quad \rightarrow \quad Q \rightarrow b.Q'$$

$$I \ b.c$$

$$Q' \rightarrow bc$$

$$I c$$

$$(D e f i n i c i ó n)$$

scribble

scribble

Scribe



Extraemos el factor común b y nos resulta.

$$Q \rightarrow Qa$$

$$B \rightarrow d abD$$

$$I c Bt a c E$$

$$D \rightarrow a c E$$

$$IE$$

$$E \rightarrow b F$$

$$F \rightarrow a c E$$

$$IE$$



5.- La gramática tiene recursión por la izquierda de manera indirecta.

La derivación provocará una recursividad directa a la izquierda

$$A \rightarrow Ba$$

$$B \rightarrow dab$$

$$I C b$$

$$C \rightarrow cBC'$$

$$I dabac'C'$$

$$C' \rightarrow bacC$$

$$I e$$

C en B  $\rightarrow Cb$

$$A \rightarrow Ba$$

$$B \rightarrow dab$$

$$I cBC'b$$

$$I aabac'C'b$$

$$C' \rightarrow bacC$$

$$I e$$

Extraemos dab

$$A \rightarrow Ba$$

$$B \rightarrow dabD$$

$$I cBbac'C'$$

$$D \rightarrow accC'b$$

$$C' \rightarrow bacC$$

$$I e$$

Encontramos que FIRST<sub>+1</sub>(C'  $\rightarrow bacC'$ )

$\cap$  FIRST<sub>+1</sub>(C'  $\rightarrow e$ ) = {b}, no es un conjunto vacío, entonces hay retroceso.

Agregamos E  $\rightarrow C'b$  y eliminamos C'

$$A \rightarrow Ba$$

$$B \rightarrow dabD$$

$$I cBbac'E$$

$$D \rightarrow accE$$

$$I e$$

$$E \rightarrow bacE$$

$$I b$$



6

FIRSTK Definición de la colección: Para la sintaxis  $\alpha$ , si  $\alpha$  la declaración derivada (que contiene solo el terminal) es menor que  $k$ , agregue la declaración completa a FIRSTK ( $\alpha$ ), si la longitud de la declaración  $\alpha$  es mayor o igual que  $k$ , agregue un prefijo con una longitud de  $k$  a FIRSTK ( $\alpha$ ), el conjunto de todas las declaraciones que se procesan es FIRSTK ( $\alpha$ )

FOLLOWK (A) Definición de la colección: Para el no terminal A, FOLLOWK (A) es un prefijo de  $k$  para la longitud de todas las cadenas de símbolos posibles inmediatamente después de A.

La expresión es de la siguiente manera:

FOLLOWK (A) = {FIRSTK ( $x$ ) |  $x$  Cualquier cadena de símbolos en forma de  $wAx$  derivada de S}

Para facilitar las descripciones de las condiciones que cumplen con la sintaxis LL( $k$ ), defina un operador:  $\oplus k$

Definimos dos colecciones, S1 y S2 y cre las cadenas en S1 y cadenas en S2, y la colección de prefijos de  $k$  para todas las cadenas posibles es S1  $\oplus k$  S2.



Finalmente se dan las condiciones para cumplir con la sintaxis

LL(k): para cualquier no terminador  $A, A$ , que coincide con varias generaciones,  $\Rightarrow \text{beta}_1 \beta_2 \dots \beta_n$

$$[\text{FIRST}_k(b_i) \cap \text{Follow}_k(A)] \cap [\text{FIRST}_k(b_j) \cap \text{Follow}_k(A)] = \emptyset,$$

Y si,  $i, j \leq n, i \neq j$

$A \Rightarrow \text{beta}_1 \beta_2 \dots \beta_i \beta_{i+1} \dots \beta_n$  y  $A \Rightarrow \text{beta}_1 \beta_2 \dots \beta_j \beta_{j+1} \dots \beta_n$

$\beta_i$  coincide con  $\beta_j$  en su  $k$ -ésima generación.

$A \Rightarrow \text{beta}_1 \beta_2 \dots \beta_i \beta_{i+1} \dots \beta_n$  y  $A \Rightarrow \text{beta}_1 \beta_2 \dots \beta_j \beta_{j+1} \dots \beta_n$

$\beta_i$  coincide con  $\beta_j$  en su  $k$ -ésima generación.



7 L.F+List → L.F+List

| E

L.F+ → ↑ L.F+'

L.F+ → L.F+↓

| ↓

La gramática no puede derivar de la secuencia  $\overline{W} \downarrow \overline{W}$ , y la secuencia cumple con los requisitos de la pregunta, por lo que no requiere que la gramática pueda derivar todas las secuencias posibles.

Si deseamos derivar  $\overline{W} \downarrow \overline{W}$ , podemos transformar la gramática

L.F+List → L.F+ L.F+List

| ↑ L.F+ List

| E

L.F+ → ↑ L.F+'

L.F+ → L.F+↓

| ↓



8. Tomamos la sintaxis de la expresión de suma, resta, multiplicación y división y declaración  $b + c * d + a$  como ejemplo:

$$E \rightarrow E + T$$

$$| E - T$$

$$| T$$

$$T \rightarrow T \times F$$

$$| T \div F$$

$$| F$$

$$F \rightarrow \text{name}$$

El analizador de sintaxis top-down (LL1) construye un árbol de sintaxis para la declaración  $b + c * d + a$  de la siguiente forma

