

Universidad del Valle de Guatemala
Diseño de Lenguajes de Programación

Pablo Díaz
13203

Pre-examen

1. La finalidad de un compilador es traducir el lenguaje de alto nivel a bajo nivel para que la computadora lo entienda. Aunque no necesariamente tiene que ser así, en general un compilador puede traducir cualquier lenguaje. La diferencia entre un intérprete y un compilador es que el intérprete ejecuta instrucción por instrucción lo cual da lugar a una traducción intermedia en lugar de traducir todo el programa a código ejecutable, como hace un compilador.

2. La primera fase llamada análisis léxico o scanning se encarga de identificar los elementos de un programa fuente de acuerdo a la gramática del lenguaje analizado. Esta fase guarda información en la tabla de símbolos y genera los tokens, que representan un identificador dentro de la tabla. La fase llamada análisis sintáctico o parsing es la encargada de imponer la estructura gramatical al programa fuente, utiliza los tokens para producir el árbol sintáctico.

la fase 3 es el análisis semántico, el cual verifica la
también semántica de las instrucciones en el programa fuente. En
esta fase se revisan los tipos de las variables y su ámbito.
La fase 4 consiste en generar código intermedio entre del
código de máquina, sirve para aumentar la portabilidad. La fase 5
se encarga de generar el código objeto deseado. En esta parte
se utilizan los registros de la máquina.

③ Una gramática libre de contexto posee:

- un conjunto de símbolos terminales
- un conjunto de símbolos no terminales
- un conjunto de reglas de producción
- un símbolo no terminal como el símbolo de inicio

Existen gramáticas libres de contexto ambiguas, esto quiere decir
que se pueden producir diferentes árboles sintácticos a partir
de las mismas reglas de producción.

④ Una definición regular define símbolos que no pertenecen al alfabeto de una expresión regular de la siguiente forma.

$$d_1 \rightarrow r_1$$

"

$$d_n \rightarrow r_n$$

donde cada símbolo d_i no pertenece al alfabeto y la expresión regular que lo define está conformada por símbolos del alfabeto.

De cierta forma una definición regular es una notación para representar expresiones regulares.

⑤ La diferencia entre un token y un lexema es que un token es una secuencia de caracteres tratados como unidad en la gramática que consisten de un nombre y un atributo. A cambio un lexema es el resultado de leer un lexema, las cuales son secuencias de caracteres de un programa.

7

⑦ Una forma de demostrar que 2 lenguajes regulares

son iguales es probando que las expresiones regulares son

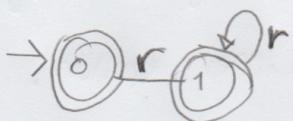
equivalentes. Se puede hacer por medio de construir un AFD

y minimizarlo. En caso del $L(r^*) = L(r^{**})$ se cumple que

que ambos

automatas

quedan así:



⑧ a. $r = a^+ b^+ c^*$

b. $r = b^* (ab^* ab^*)$

c. $r = 1^+ 00$

⑨ 1. No se pueden generar strings con mas 'a's que 'b's porque cada simbolo se genera al mismo tiempo.

2. No se pueden generar palíndromos porque el alfabeto consiste de 2 simbolas, para generar palíndromos se necesitan al menos 3 simbolas.

$$(aabb)^* = (ab)^* = (ba)^*$$

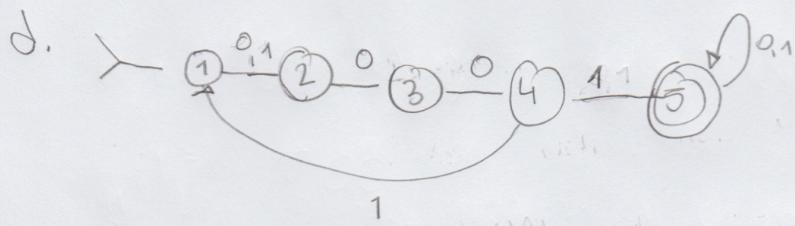
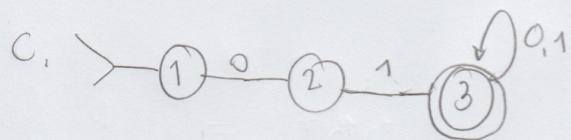
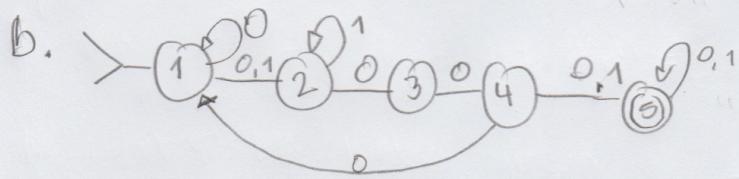
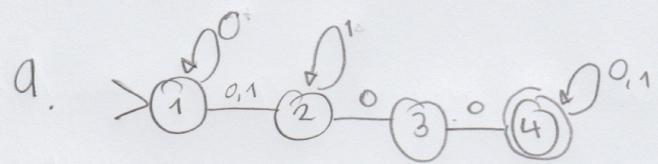
⑩ Un AFN es un autómata finito no determinista que consiste de un alfabeto, reglas de transición, estado inicial y conjunto de estados finales. La diferencia que tiene con un AFD es que un AFN no tiene transiciones epsilon y los diagramas de transición tampoco tienen transiciones epsilon.

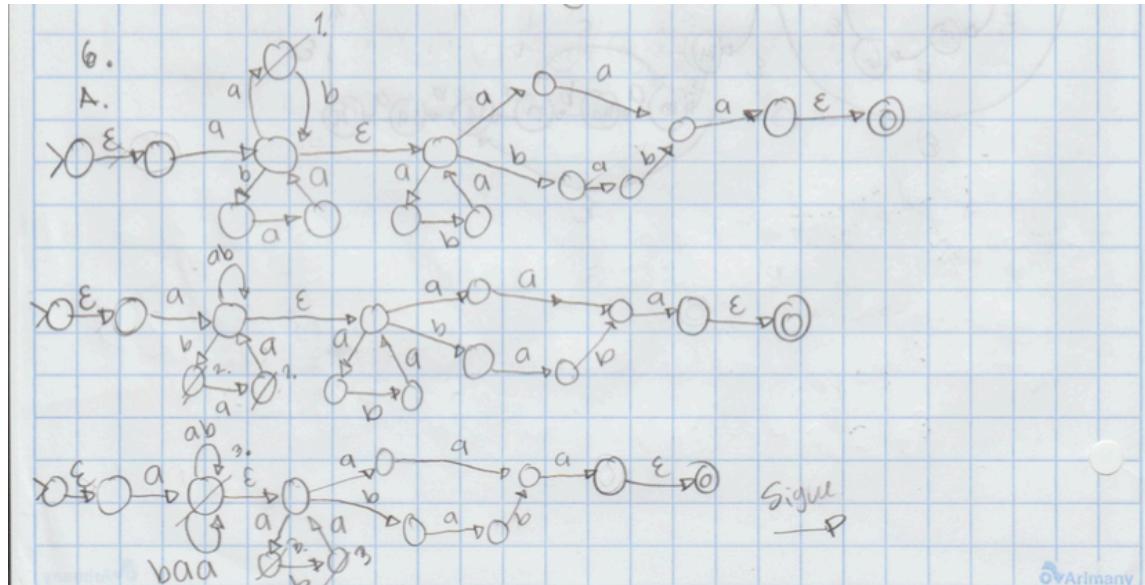
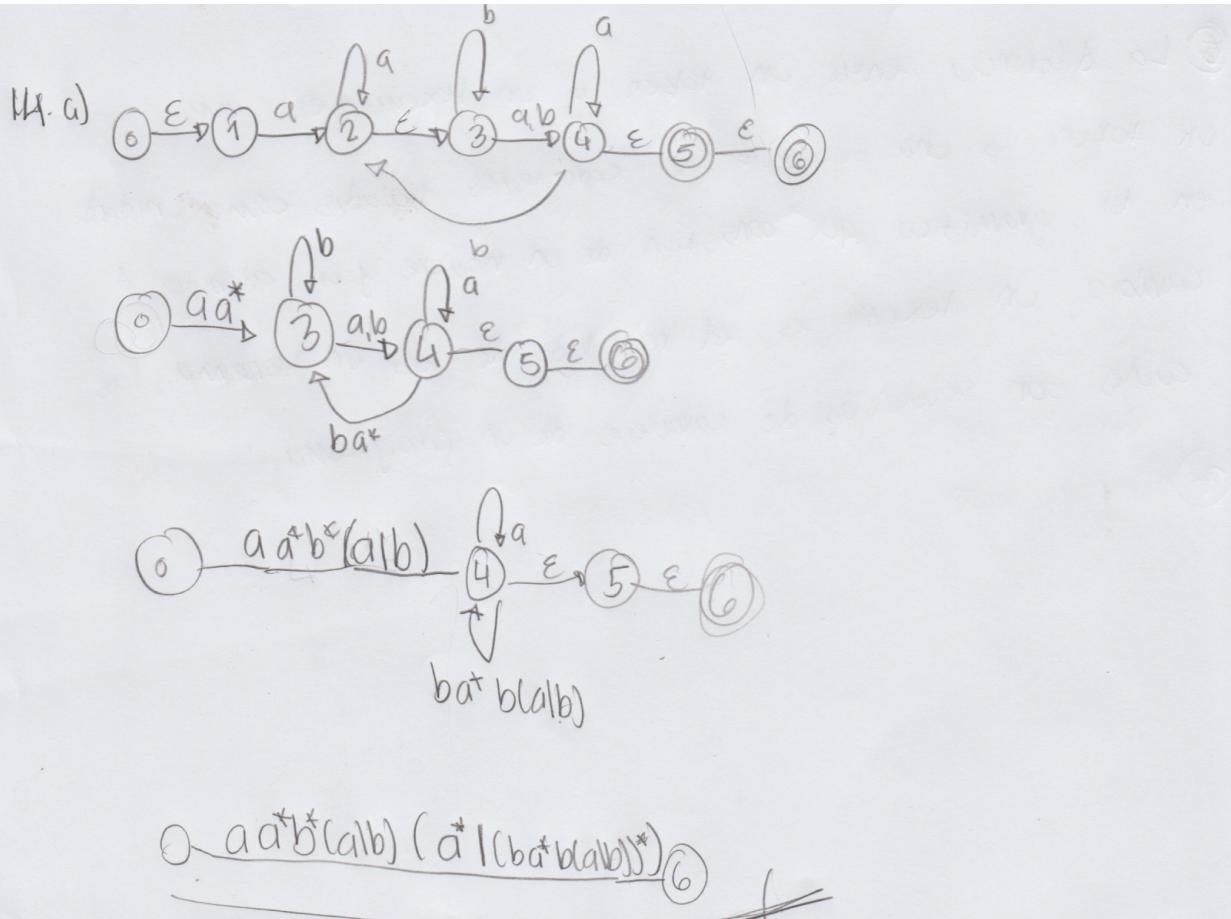
⑪ Los estados importantes de un AFN son los que no tienen transiciones con epsilon y se relaciona con la construcción directa mediante la operación nullable, la cual deuelve true si la subexpresión puede generar transiciones epsilon, como en un AFN.

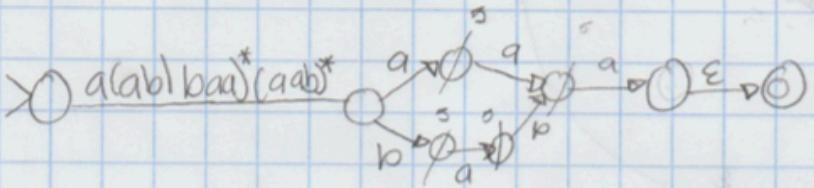
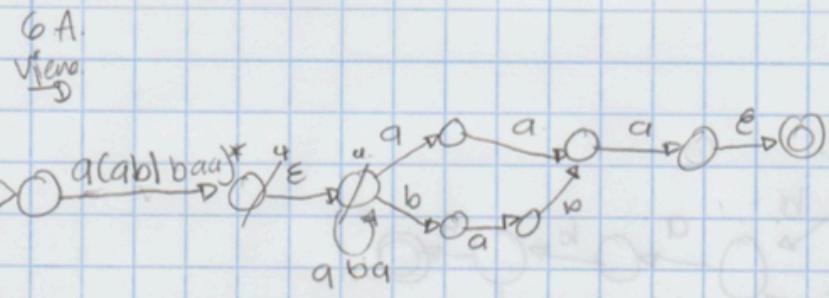
⑫ Para demostrar que $S(q, xy) = S(S(q, x), y)$ es necesario definir una \hat{S} recursiva de la siguiente forma Sea $S(q, a) \rightarrow q_1 \Rightarrow \hat{S} : K \times \Sigma^* \rightarrow K$. Por lo tanto se obtiene una función recursiva de S extendida

$$\hat{S}(q, w) = \hat{S}(q, xa) = \hat{S}(\hat{S}(q, x), a)$$

(13)

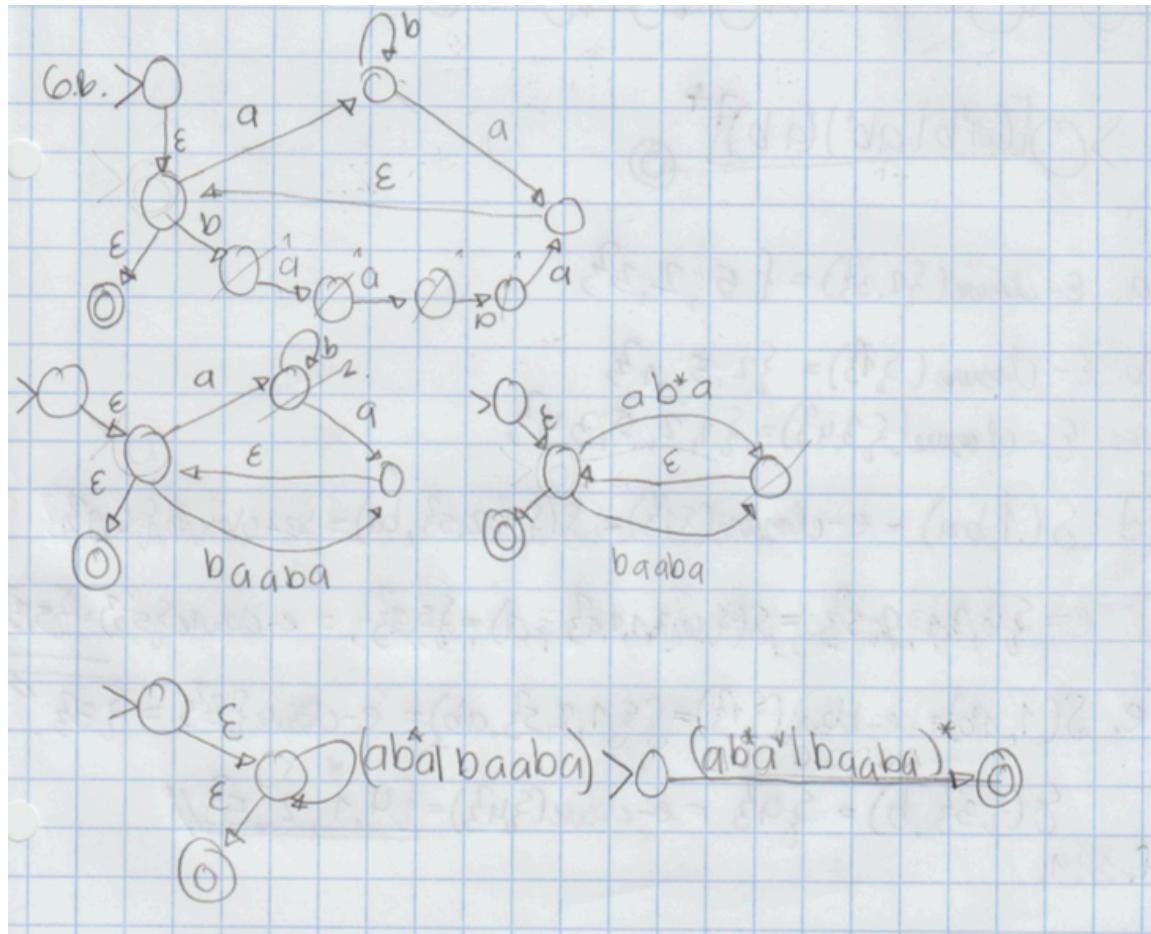


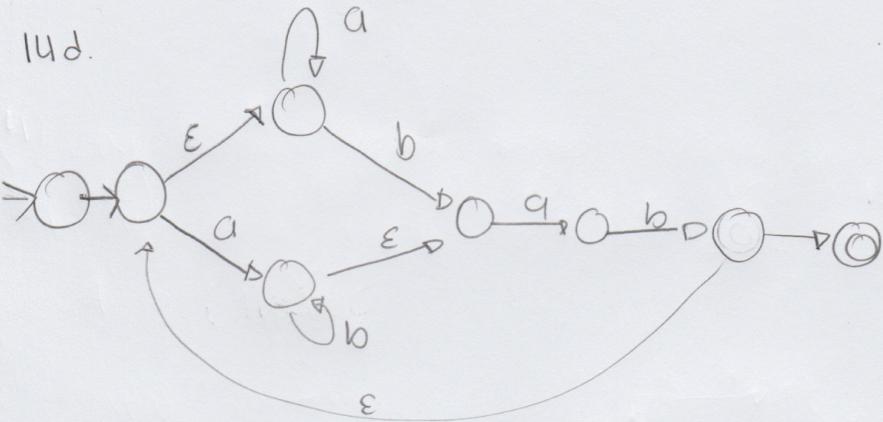




> O a(ab|baa)* (aba)* (aa|b|ab)(a) O

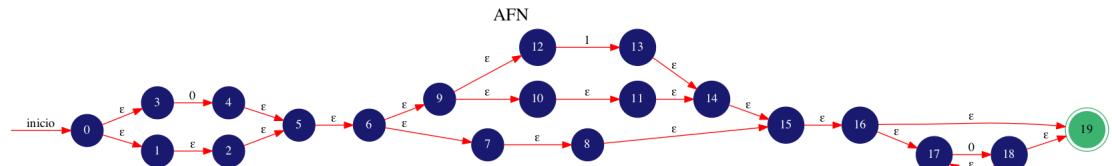
14c.



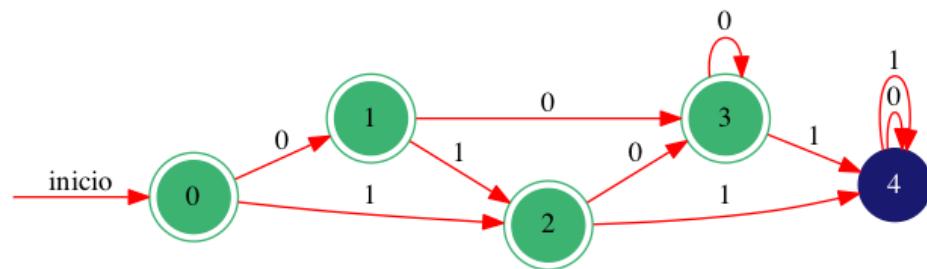


$\rightarrow \text{O } (\overbrace{a^* b \mid ab^*}) \text{ O } a \rightarrow \text{O } b \rightarrow \text{O}$

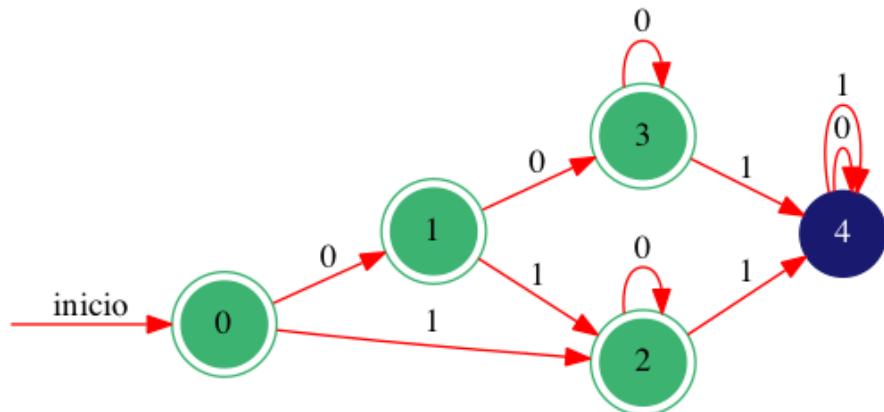
$\rightarrow \text{O } (\overbrace{a^* b a b^*}) a b \text{ O}$

a. $0?(1|\epsilon)?0^*$ 

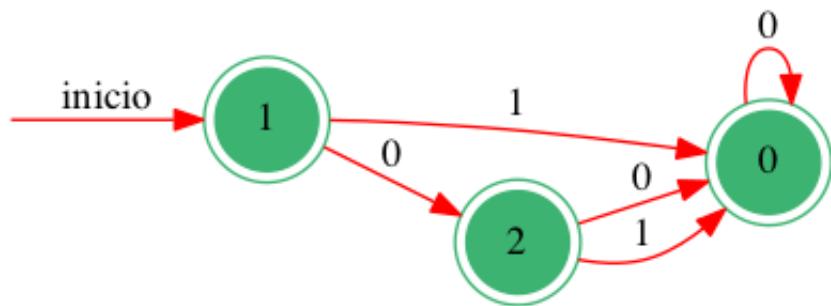
AFD Subconjuntos



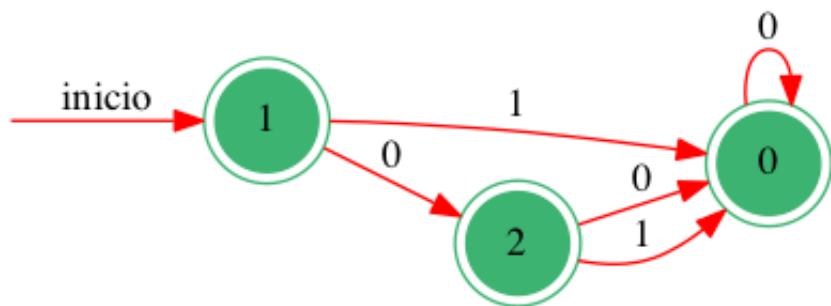
AFD Directo



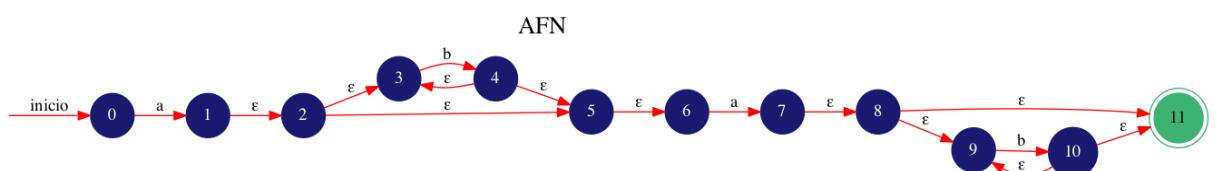
AFD Min Directo



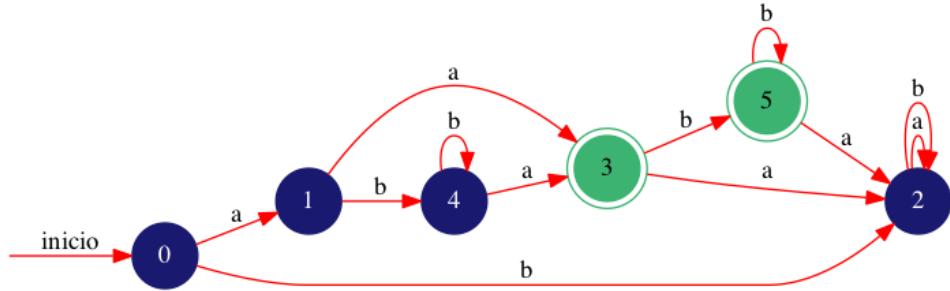
AFD Min Subconjuntos



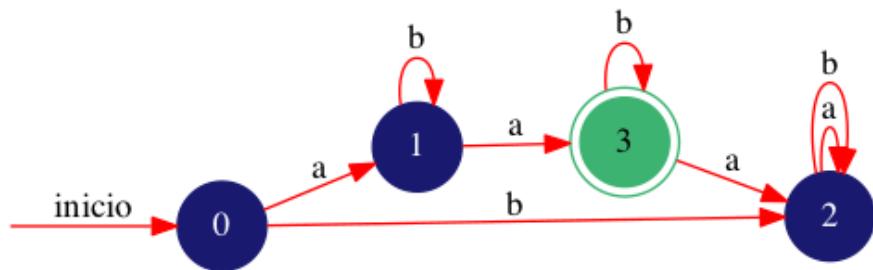
b. ab^*ab^*



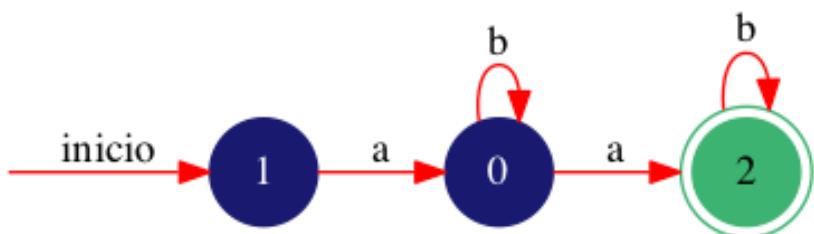
AFD Subconjuntos



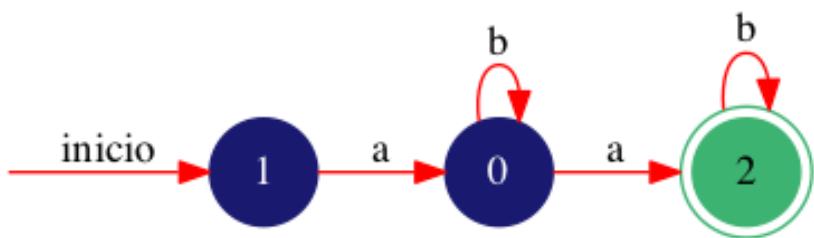
AFD Directo

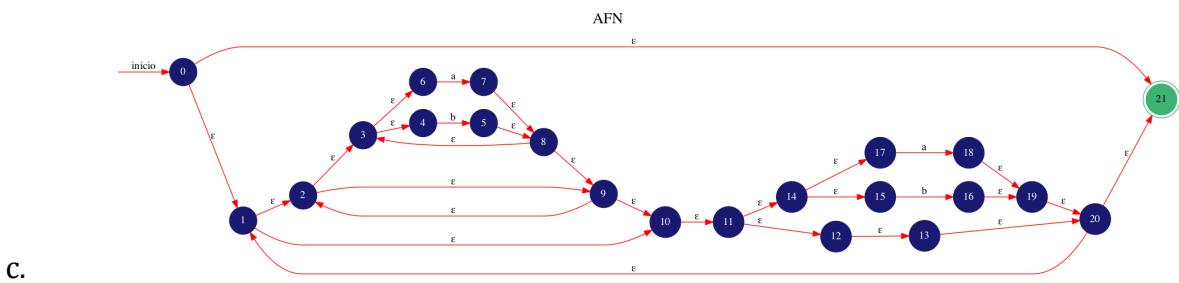


AFD Min Directo



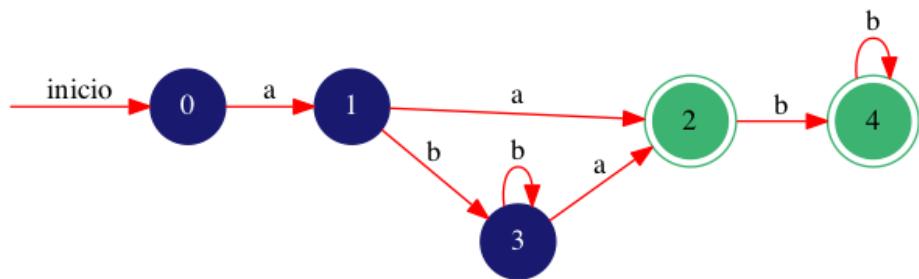
AFD Min Subconjuntos



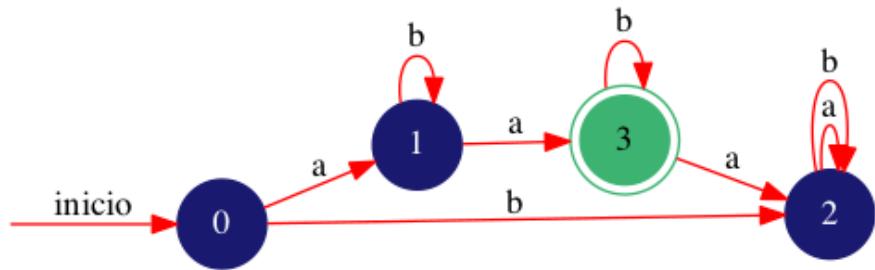


c.

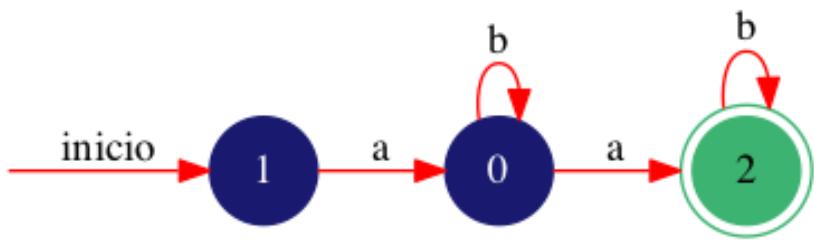
AFD SubConjuntos Sin Estados Trampa



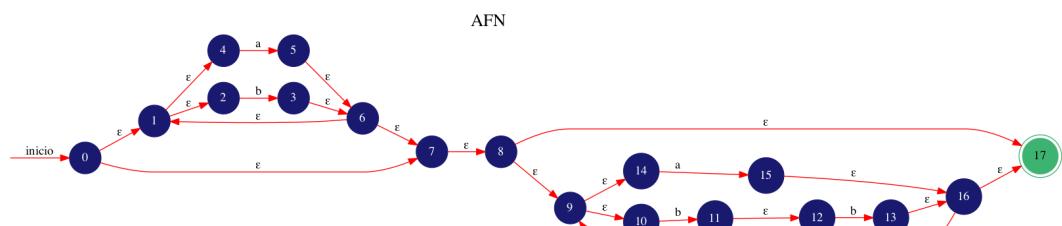
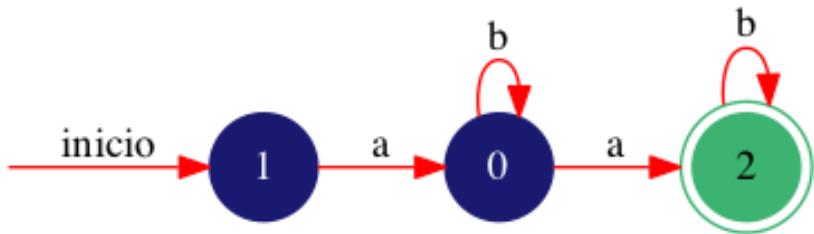
AFD Directo



AFD Min Directo

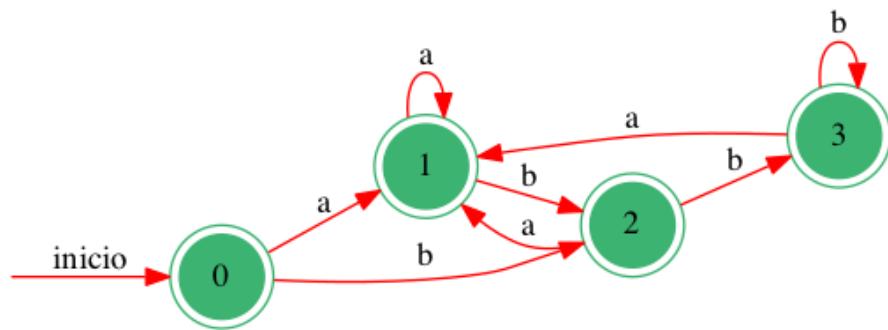


AFD Min Subconjuntos

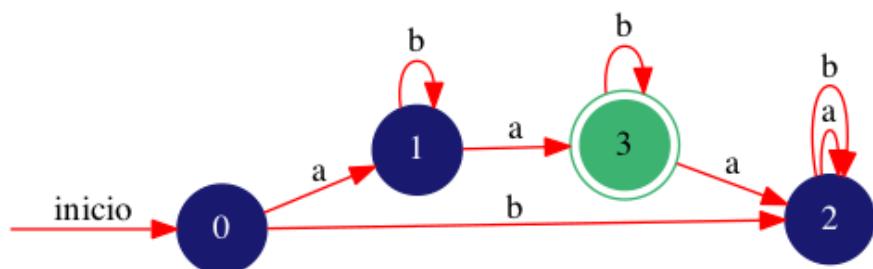


d.

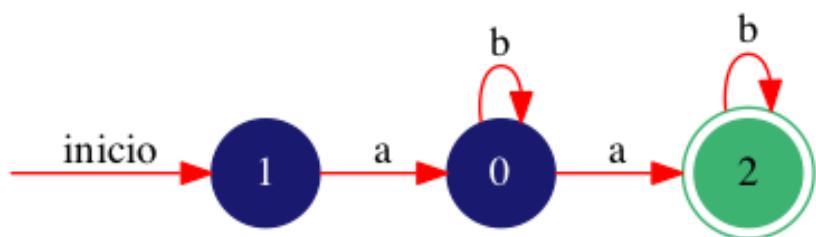
AFD Subconjuntos



AFD Directo



AFD Min Directo



AFD Min Subconjuntos

