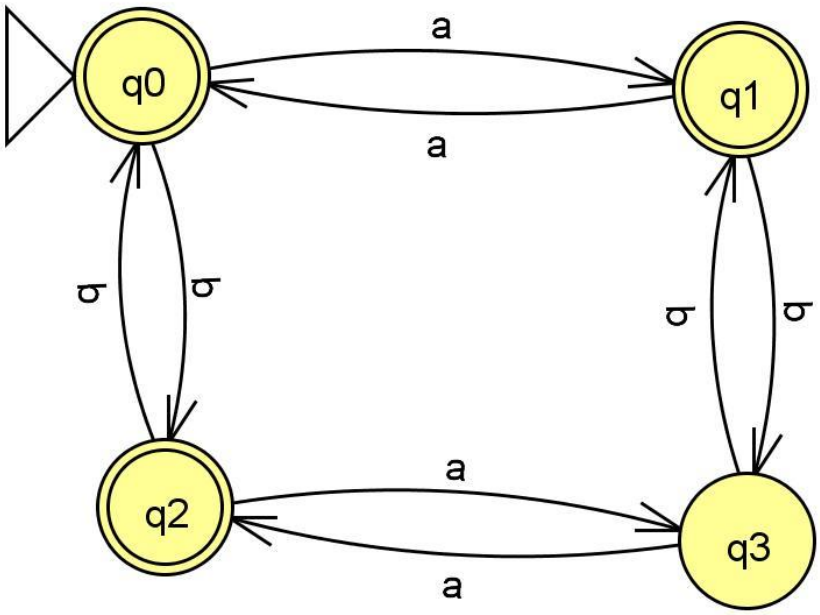
Universidad de La Laguna
Escuela Superior de Ingeniería y Tecnología. Sección de Ingeniería Informática

# INFORME DE LA PRÁCTICA 7

Rafael Herrero Álvarez. Grupo L9.
Computabilidad y Algoritmia. Turno de tarde

## Nº 1.-

Nuestro DFA quedaría así:




a) Para obtener la gramática desde el DFA, asignamos a cada estado un símbolo no terminal. Aquellos que sean de aceptación, tendrán $\varepsilon$ para poder terminar la cadena. De cada estado, analizamos a cuál puede ir y con qué símbolo, y de esa manera lo reflejamos en la gramática:
   - *S -> aA|bB|ε*
   - *A -> aS|bC|ε*
   - *B -> aC|bS|ε*
   - *C -> aB|bA*

La gramática obtenida es una gramática regular lineal por la derecha, ya que todas sus producciones son regulares por la derecha. Es una gramática de tipo 0.

b) Procedemos a comprobar las cadenas *bababb*, *ababa* y *bbaab*.
   - *bababb*: S => bB => baC => babA => babaS => bababB => bababbS => bababbε.
   - *ababa*: S => aA => abC => abaB => ababS => ababaA => ababaε.
   - *bbaab*: S => bB => bbS => bbaA => bbaaS => bbaabB => bbaabε.

   Como hemos podido comprobar, todas las cadenas han sido aceptadas.

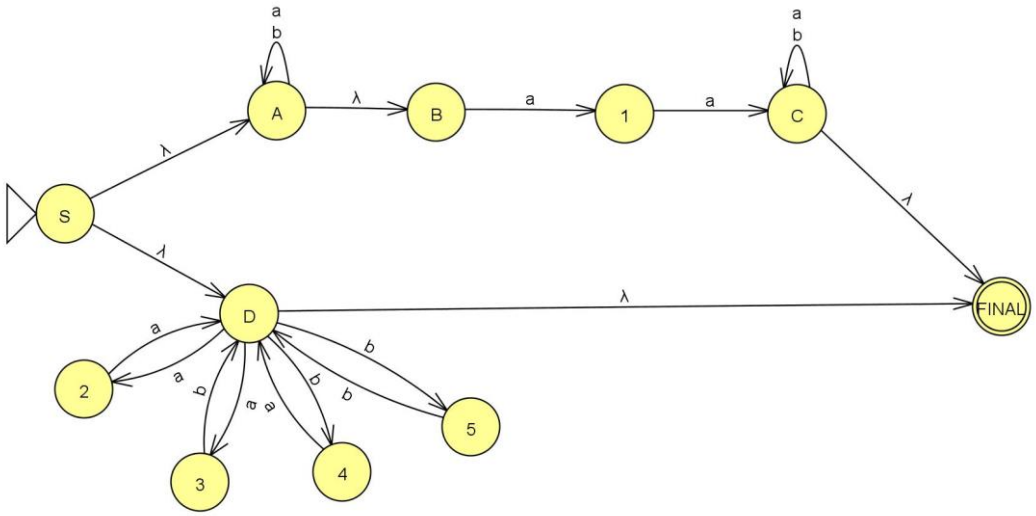
c) Con el JFLAP podemos obtener la gramática mediante la opción ‘Convert to Grammar’ en el menú ‘Convert’. Esta herramienta, con el DFA cargado, nos abre una nueva ventana, donde nos será tan sencillo obtener la gramática como pulsar el botón ‘Show All’. A la izquierda nos aparecerán todos los símbolos no terminales con sus correspondientes producciones.

## Nº 2.-

a) El lenguaje generado por esta gramática comprende a aquellas cadenas de longitud par sobre el alfabeto$\sum = \{a, b\}$, es decir se acepta ($aa|ab|ba|bb$), además de que en cuanto aparezcan dos aes consecutivas se acepta cualquier cadena. También la cadena vacía.

b) La expresión regular sería: ($bb|ab|aa(a|b)*|ba|baa(a|b)*)*$

c) Para obtener un NFA para dicho lenguaje, seguimos los pasos según nuestra gramática, uniendo un estado con otro como si de un símbolo no terminal se tratase, usando los símbolos terminales como transiciones. Quedaría un NFA así:



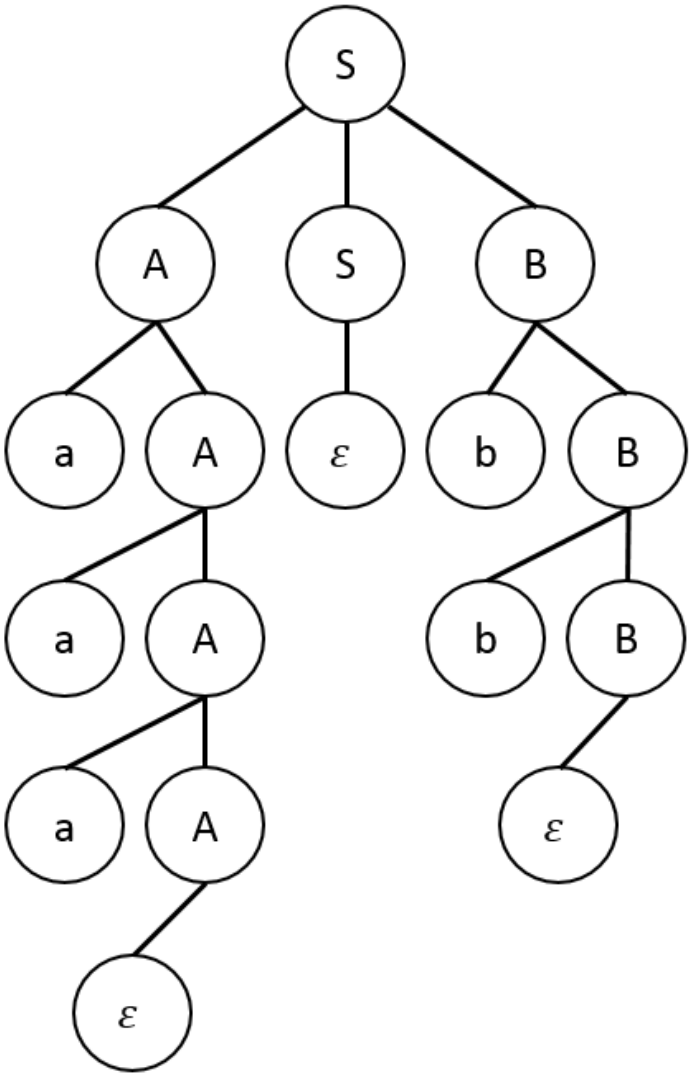

d) Podríamos introducir toda la gramática con la opción ‘Grammar’, después usar ‘Convert Right-Linear Grammar to FA’, de manera que nos sacará un NFA por pantalla. Ahora podemos comparar ambos gracias a ‘Compare equivalence’. También, podemos usar ‘Brute Force Parse’ con la gramática, de manera que probamos distintas cadenas para ver si son aceptadas, y lo mismo con la opción ‘Multiple Run en el NFA’.

## Nº 3.-

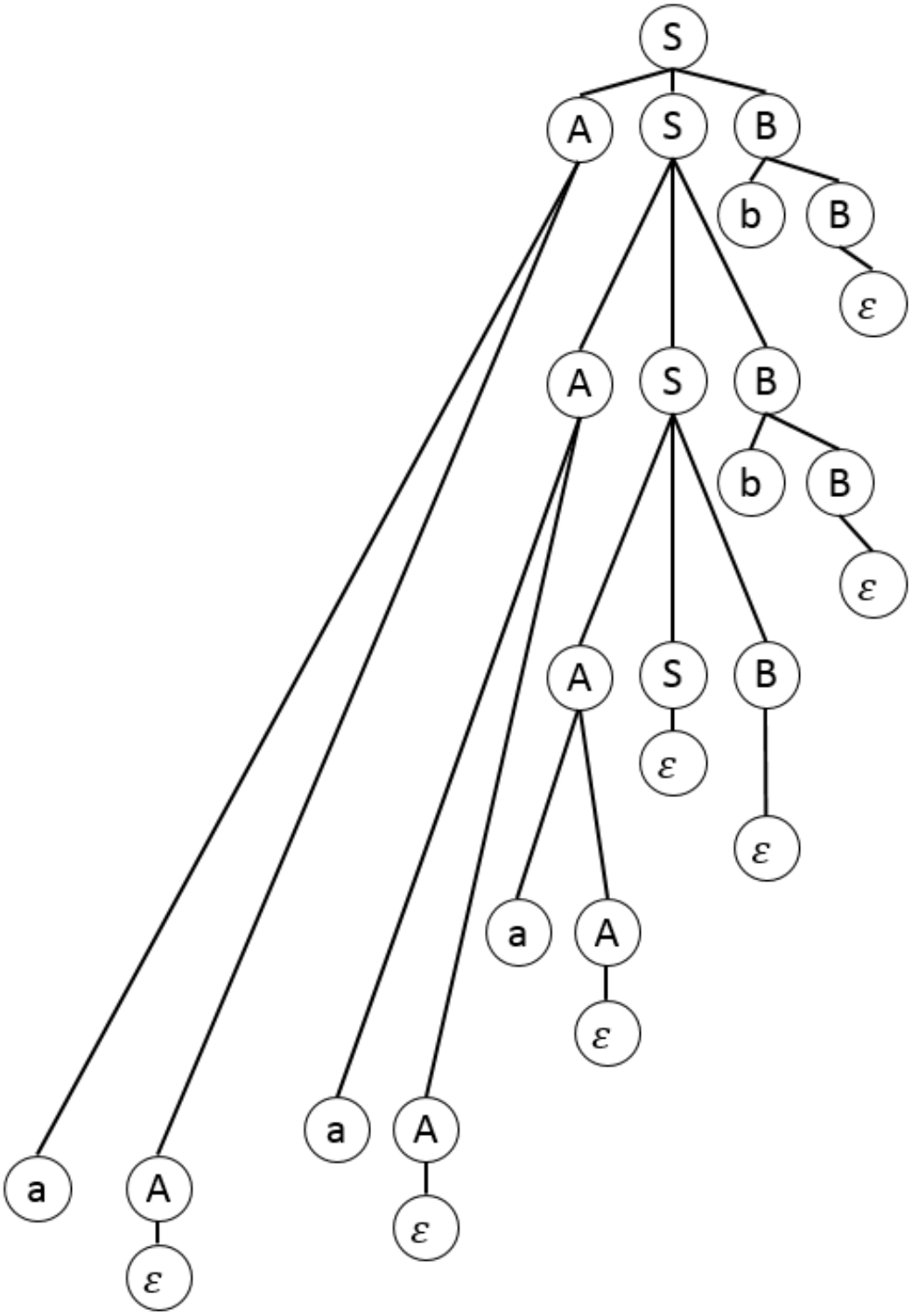
a) Derivación a la derecha:
   - S => ASB => aASB => aaASB => aaaASB => aaaSB => aaaB => aaabB => aaabbB => aaabb

a) Derivación a la izquierda:
   - S => ASB => ASbB => AASBbB => AASbBbB => AAASBbBbB => AAaASBbBbB => AaAaASBbBbB => aAaAaASBbBbB => aAaAaASBbBb => aAaAaASBbb => aAaAaASbb => aAaAaAbb => aAaAabb => aAaabb => aaabb

b) Árbol de la derivación a la derecha:



b) Árbol de la derivación a la izquierda:



c) Esta gramática es una gramática ambigua, ya que la derivación a la derecha se podría hacer de varias maneras, esta, por ejemplo, que es diferente a la anterior y genera un árbol sintáctico distinto. Por ejemplo, aquí en S se coge ASB dos veces, mientras que en el anterior es solo una vez:
   - S => ASB => aASB => aAASBB => aaAASBB => aaAaASBB => aaAaASbBB => aaAaASbBbB => aaaASbBbB => aaaSbBbB => aaabBbB => aaabbB => aaabb

d) Una gramática no ambigua sería:
   - *S -> A*
   - *A -> aA|B*
   - *B -> bB|*$\varepsilon$

e) El lenguaje generado por la gramática son todas las cadenas con el alfabeto $\sum = \{a, b\}$, que pueden empezar por a o por b, pero desde que aparezca una b, no puede volver a aparecer otra a, incluyendo a la cadena vacía. Su expresión regular sería (*a*)*(*b*)*. Es un lenguaje regular, ya que cumple las normas para que lo sea. La cadena vacía es regular, el cierre de Kleene es regular, y cualquier cadena se puede crear concatenando unas con otras.

f) **MODIFICACIÓN:**
   - G* =
     i. S -> AB
     ii. A -> aA|$\varepsilon$
     iii. B -> bB|$\varepsilon$
   - Como ambas tienen la misma expresión regular, (*a*)*(*b*)*, ambas representan el mismo lenguaje. Anteriormente comprobamos que era un lenguaje regular. Ahora, demostramos que ya no es ambigua con la cadena *ab*, solo podemos obtener este árbol: