ESIT – INFORMÁTICA
Departamento de Ingeniería Informática y de Sistemas

Computabilidad y Algoritmia
Curso 2019-2020

PRACTICA 7: La construcción de subconjuntos

7.1. Objetivos

- Consolidar los conocimientos adquiridos sobre Autómatas Finitos.
- Estudiar y practicar el algoritmo de Construcción de Subconjuntos.
- Implementar en C++ una clase para representar NFAs.
- Profundizar en las capacidades de diseñar y desarrollar programas orientados a objetos en C++.

7.2. Introducción

Se pueden conceptualizar los autómatas finitos no deterministas o en inglés *Nondeterministic Finite Automaton*, NFAs como una modificación de los autómatas finitos deterministas en los que son posibles cero, una o varias transiciones desde un estado con un símbolo del alfabeto de entrada. La Figura 7.1 muestra un ejemplo de NFA. Se observa en él que el estado `0`, ante el símbolo `a` puede transitar sobre sí mismo o bien al estado `1`. Análogamente, el estado `1` puede transitar con entrada `a` al estado `2` o bien al estado `3`.

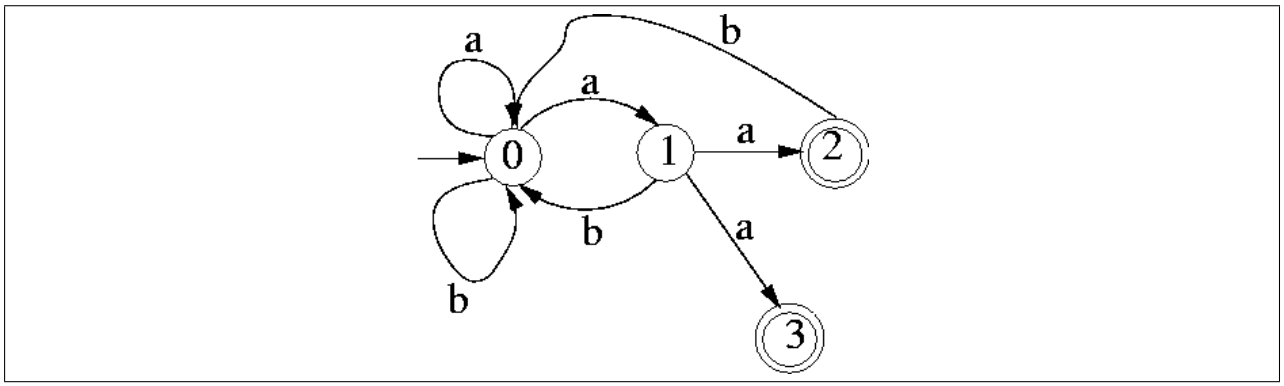

Figura 7.1: Ejemplo de NFA

Se define formalmente un NFA como una tupla $(\Sigma, Q, q_0, F, \delta)$ donde todas las componentes tienen el mismo significado que en un DFA, salvo la función de transición, δ que en este caso es una función:

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^Q$$
$$(q, \sigma) \rightarrow \{q_1, q_2, \ldots, q_N\}$$

es decir, asigna a un par (q, σ) un elemento de 2^Q (partes de Q, el conjunto de todos los subconjuntos de Q) donde $q \in Q, \sigma \in (\Sigma \cup \{\epsilon\})$. $\delta(q, \sigma)$ es el conjunto de estados p tales que hay una transición etiquetada σ que va desde q hasta p y donde σ es ϵ o bien un símbolo del alfabeto Σ.

En el NFA de la Figura 7.1:
$\delta(2, a) = \emptyset$
$\delta(3, a) = \delta(3, b) = \emptyset$
$\delta(0, a) = \{0, 1\}$

La Tabla 7.1 muestra la función de transición del NFA de la Figura 7.1.

δ	a	b
0	$\{0, 1\}$	$\{0\}$
1	$\{2, 3\}$	$\{0\}$
2	$\emptyset$	$\{0\}$
3	$\emptyset$	$\emptyset$

Tabla 7.1: Función de transición δ para el NFA de la Figura 7.1

Los autómatas en los que σ es ϵ o bien un símbolo del alfabeto Σ se denominan NFAs con transiciones vacías (ϵ-transiciones) porque el autómata puede transitar sin necesidad de entrada (con una entrada vacía, ϵ). Por su parte, el no determinismo viene dado porque el autómata puede transitar con una misma entrada hacia diferentes estados. En particular, puede transitar a un conjunto de estados vacío (lo cual significaría que el autómata es incapaz de realizar una transición).

De forma análoga a como se hizo para el caso de un DFA, se puede extender la función de transición δ de un NFA para que acepte como entrada un estado y una cadena de símbolos del alfabeto:

$$\hat{\delta} : Q \times \Sigma^* \rightarrow 2^Q$$
$$(q, w) \rightarrow \{q_1, q_2, ..., q_N\}$$

$\hat{\delta}(q, w)$ serán todos los estados p tales que se puede ir desde el estado q hasta p recorriendo un camino etiquetado con los símbolos de $w \in \Sigma^*$, incluyendo potencialmente arcos etiquetados con ϵ.

Para construir $\hat{\delta}$ será importante calcular el conjunto de estados alcanzables a partir de un determinado estado q utilizando solamente ϵ-transiciones. Este cálculo es equivalente al de calcular qué vértices son alcanzables partiendo de uno dado en un grafo dirigido. El vértice origen será el vértice correspondiente al estado q en el diagrama

de transición de estados y el grafo dirigido será el formado por los arcos etiquetados ϵ. Se utilizará la notación $\epsilon - clausura(q)$ para denotar al conjunto de todos los vértices p tales que hay un camino desde q hasta p etiquetado ϵ.

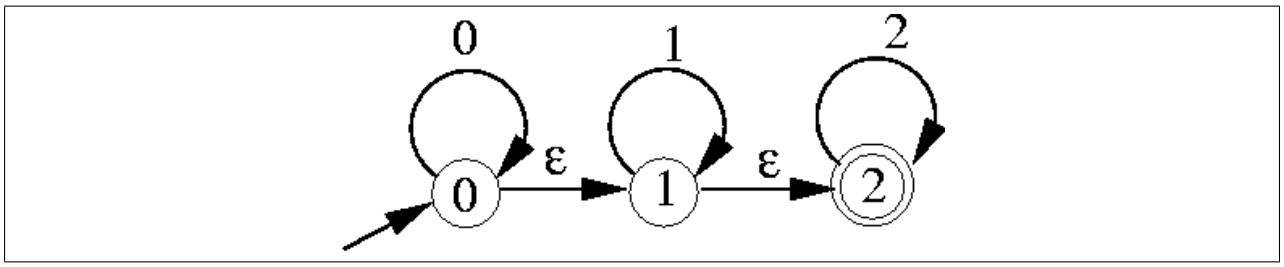

Figura 7.2: Un NFA que reconoce el lenguaje denotado por `0*1*2*`

En el NFA de la Figura 7.2, $\epsilon - clausura(0) = \{0, 1, 2\}$ puesto que hay un camino desde el estado 0 hasta él mismo etiquetado con ϵ (no hay arcos en este camino), el camino $0 - 1$ etiquetado ϵ indica que el estado 1 está en la $\epsilon - clausura(0)$ y el camino $0 - 1 - 2$ etiquetado ϵ indica que $2 \in \epsilon - clausura(0)$. De forma natural, se define $\epsilon - clausura(R)$ siendo R un conjunto de estados como $\bigcup_{q \in R} \epsilon - clausura(q)$.

El Listado 7.1 presenta el pseudocódigo de un algoritmo para calcular la $\epsilon - clausura$ de un conjunto de estados T. El cálculo de la $\epsilon - clausura$ se puede interpretar como el cálculo del conjunto de nodos alcanzables partiendo de un nodo dado en un grafo dirigido. En el caso de la $\epsilon - clausura$, el conjunto de nodos del grafo son los estados del conjunto T y el grafo serían las transiciones etiquetadas con ϵ en el diagrama de transición de estados del NFA. El algoritmo utiliza una pila y un conjunto de estados. El conjunto de estados se inicializa con todos los estados $q \in T$ y al final de la ejecución almacenará todos los estados de la $\epsilon - clausura(T)$.

```
For (cada estado q de T) do
  push(q)
epsilon-clausura(T) := T;
while (not pila vacia) do
  begin
  p := pop();
  for (cada estado u con una arista de p a u etiquetada epsilon) do
    if not (u in epsilon-clausura(T)) then
      begin
      epsilon-clausura(T) := epsilon-clausura(T) + {u}
      push(u);
      end;
  end;
```

Listado 7.1: Pseudocódigo del algoritmo para el cálculo de la $\epsilon - clausura$ de un conjunto de estados, T

Con los conceptos introducidos hasta este punto se puede extender la función de transición:

- $\hat{\delta}(q, \epsilon) = \epsilon - clausura(q)$
- Para $w \in \Sigma^*$ y $a \in \Sigma$, $\hat{\delta}(q, wa) = \epsilon - clausura(P)$ siendo $P = \{p \mid$ para algún r en $\hat{\delta}(q, w)$, p está en $\delta(r, a)\}$

También es conveniente extender δ y $\hat{\delta}$ a conjuntos de estados R:

- $\delta(R, a) = \bigcup_{q \in R} \delta(q, a)$
- $\hat{\delta}(R, w) = \bigcup_{q \in R} \hat{\delta}(q, w)$

Nótese que en el caso de un NFA $\hat{\delta}(q, a)$ no coincide necesariamente con $\delta(q, a)$ puesto que $\hat{\delta}(q, a)$ incluye todos los estados alcanzables a partir de q utilizando caminos etiquetados con a (incluyendo caminos con arcos etiquetados con ϵ) mientras que $\delta(q, a)$ incluye sólo aquellos estados alcanzables desde q a través de arcos etiquetados a.

Se define $L(M)$, el lenguaje aceptado por el NFA $M \equiv (\Sigma, Q, q_0, F, \delta)$, como el conjunto de cadenas $\{w \mid \hat{\delta}(q_0, w)$ contiene algún estado de F$\}$.

El *Algoritmo de Construcción de Subconjuntos* [1, 4] permite convertir un NFA en un DFA que reconoce el mismo lenguaje, y por tanto equivalente. El algoritmo es importante en teoría porque establece que los NFAs, a pesar de su flexibilidad adicional, son incapaces de reconocer cualquier lenguaje que no pueda ser reconocido por algún DFA. También es importante a efectos prácticos porque permite convertir NFAs, que habitualmente son más fáciles de obtener, en DFAs que se pueden simular de forma más eficiente.

Mientras que en la tabla de transiciones de un NFA cada posición de la tabla ($\delta(q, a)$) es un conjunto de estados, en la de un DFA cada posición de la tabla contiene un único estado. La idea que se utiliza para convertir un NFA en un DFA consiste en asociar cada estado del DFA con un conjunto de estados del NFA. El DFA utilizará un estado para "representar" todos los posibles estados en los que se puede encontrar el NFA después de leer cada símbolo de la entrada. Dicho de otro modo, después de leer la entrada $x_1, x_2, \ldots, x_m$ el DFA se encontrará en un estado que representa al subconjunto M de los estados del NFA alcanzables desde el estado de arranque del NFA siguiendo algún camino etiquetado con los símbolos de la cadena de entrada. El número de estados del DFA puede crecer de forma exponencial con respecto al número de estados del NFA equivalente, pero en la práctica este caso peor no es frecuente.

El algoritmo de construcción de subconjuntos se basa en construir la tabla de transiciones `tranDFA` del DFA. Cada estado del DFA será un conjunto de estados del NFA y se construye `tranDFA` de modo que el DFA simulará "en paralelo" todos los posibles movimientos que el NFA pueda realizar con una determinada cadena de entrada. El algoritmo utiliza las operaciones $\epsilon - clausura()$ y $move()$. Dado un conjunto de estados $S \subseteq Q$ del NFA y un símbolo $a \in \Sigma$, se define $move(S, a)$, como el conjunto de todos los estados que pueden alcanzarse desde un estado de S mediante una transición con

el símbolo de entrada, a. El código para implementar la función $move()$ es muy similar al del cómputo de la $\epsilon - clausura$ que se muestra en el Listado 7.1.

Antes de leer el primer símbolo de la cadena de entrada, el NFA puede estar en cualquiera de los estados de $\epsilon - clausura(s_0)$ siendo s_0 el estado de arranque del NFA. Si se supone que los estados alcanzables partiendo de s_0 con una determinada secuencia de símbolos de entrada son los del conjunto `T` y suponemos también que `a` es el siguiente símbolo de la entrada, al leer `a` el NFA puede cambiar su estado a cualquiera de los estados del conjunto $move(T, a)$. Si además se tiene en cuenta que puede haber ϵ-transiciones, el NFA puede transitar a cualquiera de los estados de $\epsilon - clausura(move(T, a))$.

El algoritmo construye `estadosDFA`, el conjunto de estados del DFA y `tranDFA`, su tabla de transiciones del siguiente modo: cada estado del DFA corresponde a un conjunto de estados del NFA en los que éste podría estar después de leer alguna secuencia de símbolos de entrada, incluidas todas las posibles ϵ-transiciones anteriores o posteriores a la lectura de símbolos. El estado inicial del DFA será $\epsilon - clausura(s_0)$. Se añadirán estados y transiciones al DFA utilizando el algoritmo (construcción de subconjuntos) que se presenta en la Figura 7.3.

```
1  q0 := ε − clausura(s0);
2  desmarcar(q0);
3  estadosDFA := {q0};
4  while (∃T ∈ estadosDFA && not marcado(T)) do
5     begin
6     marcar(T);
7     for (cada símbolo de entrada, a) do
8        begin
9        H := ε − clausura(move(T, a));
10       if H ∉ estadosDFA then
11          begin
12          desmarcar(H);
13          estadosDFA := estadosDFA ∪ H
14          end
15       tranDFA[T, a] := H;
16       end
17    end
```

Figura 7.3: El algoritmo de construcción de subconjuntos

Un estado del DFA será de aceptación si se corresponde con un conjunto de estados del NFA que contenga al menos un estado de aceptación del DFA. Veamos a continuación una traza del funcionamiento del algoritmo de la construcción de subconjuntos. Considérese el NFA de la 7.4 que acepta el lenguaje representado por la expresión regular `(a|b)*abb`.

- El estado inicial del DFA equivalente es $\epsilon - clausura(0) = A = \{0, 1, 2, 4, 7\}$ puesto

que los estados de `A` son los alcanzables con ϵ-transiciones partiendo del estado `0`.

- Siguiendo el algoritmo de la construcción de subconjuntos, se marca el conjunto `A` y se calcula

  $\epsilon - clausura(move(A, a)) = \epsilon - clausura(\{3, 8\}) = \{1, 2, 3, 4, 6, 7, 8\} = B$

  y la correspondiente entrada en la tabla de transiciones del DFA será `tranDFA[A, a] = B`.

- Siguiendo con el bucle de la línea 7 del algoritmo, se utiliza ahora el siguiente símbolo del alfabeto, `b` para calcular

  $\epsilon - clausura(move(A, b)) = \epsilon - clausura(\{5\}) = \{1, 2, 4, 5, 6, 7\} = C$

  y se obtiene también que `tranDFA[A, b] = C`

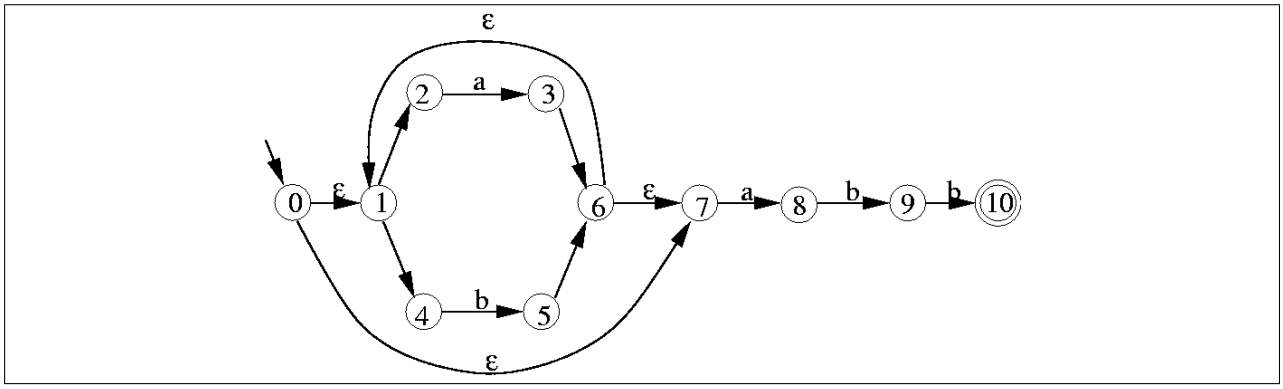

Figura 7.4: NFA que reconoce el lenguaje representado por la expresión regular `(a|b)*abb`

- Se continúa el proceso con el conjunto `B` que está sin marcar, y que se marca en este momento:

  $\epsilon - clausura(move(B, a)) = \epsilon - clausura(\{3, 8\}) = B$
  `tranDFA[B, a] = B`

- Se utiliza el siguiente símbolo del alfabeto con el conjunto `B`:

  $\epsilon - clausura(move(B, b)) = \epsilon - clausura(\{5, 9\}) = \{1, 2, 4, 5, 6, 7, 9\} = D$
  `tranDFA[B, b] = D`

- y se pasa al siguiente conjunto sin marcar, `C`, marcándolo:

  $\epsilon - clausura(move(C, a)) = \epsilon - clausura(\{3, 8\}) = B$
  `tranDFA[C, a] = B`
  $\epsilon - clausura(move(C, b)) = \epsilon - clausura(\{5\}) = C$
  `tranDFA[C, b] = C`

- Ahora se considera el estado `D` que no está marcado:

  $\epsilon - clausura(move(D, a)) = \epsilon - clausura(\{3, 8\}) = B$
  `tranDFA[D, a] = B`
  $\epsilon - clausura(move(D, b)) = \epsilon - clausura(\{5, 10\}) = \{1, 2, 4, 5, 6, 7, 10\} = E$
  `tranDFA[D, b] = E`

- Como el estado `E` no está marcado, se calcula

  $\epsilon - clausura(move(E, a)) = \epsilon - clausura(\{3, 8\}) = B$
  `tranDFA[E, a] = B`
  $\epsilon - clausura(move(E, b)) = \epsilon - clausura(\{5\}) = C$
  `tranDFA[E, b] = C`

- y aquí se detiene el algoritmo porque no se cumple la condición del bucle de la línea 4, ya que todos los estados del DFA están ahora marcados.

Los conjuntos `A`, `B`, `C`, `D`, y `E` son los estados del DFA que se muestra en la Figura 7.5, aunque en esa figura aparecen etiquetados respectivamente como 0, 1, 2, 3 y 4.

En las referencias [2, 3] pueden verse ejemplos adicionales con trazas de la ejecución del algoritmo.

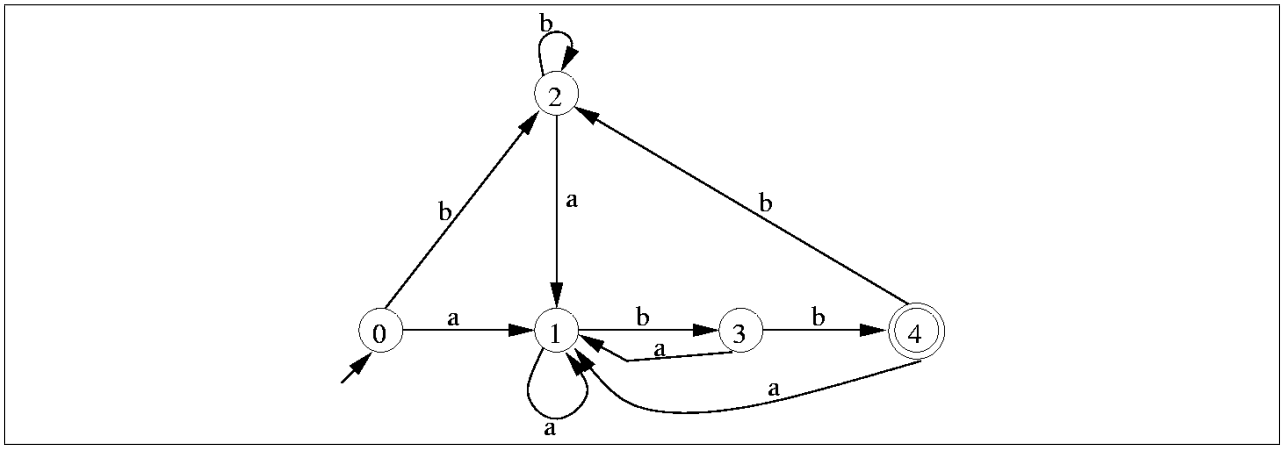

Figura 7.5: DFA equivalente al NFA de la Figura 7.4

7.3. Ejercicio práctico

Desarrollar un programa `NFA2DFA.cpp` que lea un fichero de texto en el que figura la especificación de un NFA y genere un fichero `.dfa` con la especificación de un DFA equivalente, construido mediante el Algoritmo de Construcción de Subconjuntos. El comportamiento del programa al ejecutarse en línea de comandos debiera ser:

```
$ ./NFA2DFA
Modo de empleo: ./NFA2DFA input.nfa output.dfa
Pruebe 'NFA2DFA --help' para más información.
```

Donde `input.dfa` es el fichero conteniendo la descripción del NFA y `output.dfa` es el fichero de especificación del DFA equivalente. La opción `--help` en línea de comandos ha de producir que se imprima en pantalla un breve texto explicativo del funcionamiento del programa.

Los ficheros de entrada que especifican DFAs tienen la estructura que ya se definió en una práctica anterior. Los ficheros de especificación de NFAs tienen una estructura similar, y definen en este orden el alfabeto, los estados, el estado inicial, los estados finales y las transiciones. El formato de cada uno de estos elementos en el fichero es el siguiente:

1. Alfabeto: una línea que contiene N, el número de símbolos en el alfabeto seguida de N+1 líneas. Una ϵ-transición se representará mediante el carácter ~ (código ASCII 126). De las N+1 líneas, la primera de ellas contiene el símbolo correspondiente a ϵ (caracter ~) y las N siguientes se corresponden con cada uno de los símbolos del alfabeto. Cada símbolo del alfabeto debe ser una cadena de caracteres imprimibles, sin incluir espacios en blanco.

2. Estados: una línea que contiene M, el número de estados, seguida de M líneas, cada una de las cuales contiene el identificador (etiqueta) de un estado. Cada identificador de estado debe ser una cadena alfanumérica sin espacios.

3. Estado inicial: una línea que contiene el identificador del estado inicial. El estado inicial ha de ser uno de los estados relacionados anteriormente.

4. Estados de aceptación: una línea que contiene F, el número de estados finales, seguido por F líneas, cada una de las cuales contiene el identificador de un estado final. Cada estado final ha de ser uno de los relacionados anteriormente.

5. Transiciones: una línea que contiene T, el número de transiciones en la función δ, seguida de T líneas, cada una de las cuales contiene tres cadenas *qi*, *sim*, *qf* separadas por espacios. Cada una de estas líneas indica una transición del estado *qi* al *qf* con símbolo *sim*, es decir, $\delta(qi, sim) = qf$. *qi* y *qf* han de estar listados en la lista de estados y *sim* debe estar en la lista de símbolos del alfabeto o ser el carácter ~ para el caso de una ϵ-transición.

En los ficheros que representan NFAs, cualquier línea que comience con los caracteres `//` será ignorada, puesto que representará un comentario. Los comentarios se utilizarán para anotar en los ficheros características significativas de los autómatas representados.

El programa `NFA2DFA.cpp` ha de contener al menos dos clases, `Dfa` y `Nfa` para representar a ambos tipos de autómatas finitos. La clase `Dfa` ha de disponer de un método *drawDFA()* que posibilita la generación de un fichero gráfico en formato DOT que representa el DFA en cuestión.

7.4. Rúbrica de evaluación del ejercicio

Se señalan a continuación los aspectos más relevantes (la lista no es exhaustiva) que el profesorado tendrá en cuenta a la hora de evaluar el trabajo que el alumnado presentará en la sesión de evaluación de la práctica:

- Capacidad del programador(a) de introducir cambios en el programa desarrollado.
- El comportamiento del programa debe ajustarse a lo solicitado en el enunciado.
- El programa diseñado ha de seguir el paradigma OOP.
- Se valorará la coherencia e idoneidad del diseño realizado en las clases diseñadas para representar autómatas finitos.
- Modularidad: el programa ha de escribirse de modo que las diferentes funcionalidades que se precisen sean encapsuladas en métodos cuya extensión textual se mantenga acotada.
- El programa ha de ceñirse al formato de escritura de programas adoptado en las prácticas de la asignatura.
- En la sesión de evaluación de este trabajo, todos los ficheros con códido fuente se han de alojar en un único directorio junto con el fichero Makefile de compilación.
- Todos los atributos de las clases definidas en el proyecto han de tener un comentario descriptivo de la finalidad del atributo en cuestión.
- Se valorará que los comentarios del código fuente sigan el formato de los comentarios de Doxygen.

Si el alumnado tiene dudas respecto a cualquiera de estos aspectos, debiera acudir al foro de discusiones de la asignatura para plantearlas allí. Se espera que, a través de ese foro, el alumnado intercambie experiencias y conocimientos, ayudándose mutuamente a resolver dichas dudas. También el profesorado de la asignatura intervendrá en las discusiones que pudieran suscitarse, si fuera necesario.

Bibliografía

[1] Powerset construction https://en.wikipedia.org/wiki/Powerset_construction

[2] Finite Automata: Subset Algorithm http://www.cs.utsa.edu/~wagner/CS3723/fa/sa.html

[3] NFA to DFA conversion algorithm with solved example https://er.yuvayana.org/nfa-to-dfa-conversion-algorithm-with-solved-example/

[4] Aho, A., Sethi, R., y Ullman, J. Compiladores. Principios, Técnicas y Herramientas. Addison-Wesley Iberoamericana, 1990. ISBN: 0-201-62903-8