Universidad de La Laguna

ESIT – INFORMÁTICA
Departamento de Ingeniería Informática y de Sistemas

Computabilidad y Algoritmia
Curso 2019-2020

PRACTICA 6: Autómatas Finitos Deterministas

6.1. Objetivos

- Consolidar los conocimientos adquiridos sobre Autómatas Finitos Deterministas (DFAs).
- Conocer Graphviz, un software para la visualización de grafos.
- Implementar en C++ una clase para representar DFAs.
- Profundizar en las capacidades de diseñar y desarrollar programas orientados a objetos en C++.

6.2. Introducción

Un autómata finito determinista, AFD (o DFA de su denominación en inglés *Deterministic Finite Automaton*) es una máquina de estados finitos que acepta o rechaza una cadena de símbolos determinada, realizando una secuencia de transiciones entre sus estados, determinada únicamente por la cadena de símbolos. El determinismo alude a la unicidad de la ejecución del cálculo. Fueron Warren McCulloch y Walter Pitts en 1943 quienes, en busca de modelos simples para representar máquinas de estado finito, introdujeron el concepto de autómata finito.

Si bien un DFA es un concepto matemático abstracto, con frecuencia se implementa en hardware y software para resolver diferentes problemas. Por ejemplo, un DFA puede modelar un software que decida si las entradas de un usuario en respuesta a una solicitud de dirección de correo electrónico son válidas o no.

Los DFAs reconocen exactamente el conjunto de lenguajes regulares, que son útiles entre otras finalidades, para hacer analizadores léxicos y detectores de patrones textuales.

Un DFA se define con un conjunto de estados y un conjunto de transiciones entre estados que se producen a la entrada de símbolos de entrada pertenecientes a un alfabeto Σ. Para cada símbolo de entrada hay exactamente una transición para cada estado. Hay un estado especial, denominado estado inicial o de arranque que es el

estado en el que el autómata se encuentra inicialmente. Por otra parte, algunos estados se denominan finales o de aceptación. Así pues, un Autómata Finito Determinista se caracteriza formalmente por una quíntupla $(\Sigma, Q, q_0, F, \delta)$ donde cada uno de estos elementos tiene el siguiente significado:

- Σ es el alfabeto de entrada del autómata. Se trata del conjunto de símbolos que el autómata acepta como entradas.
- Q es el conjunto finito de los estados del autómata. El autómata siempre se encontrará en uno de los estados de este conjunto.
- q_0 es el estado inicial o de arranque del autómata ($q_0 \in Q$). Se trata de un estado distinguido. El autómata se encuentra en este estado al comienzo de la ejecución.
- F es el conjunto de estados finales o de aceptación del autómata ($F \subseteq Q$). Al final de una ejecución, si el estado en que se encuentra el autómata es un estado final, se dirá que el autómata ha aceptado la cadena de símbolos de entrada.
- δ es la función de transición. $\delta : Q \times \Sigma \to Q$ que determina el único estado siguiente para un par (q_i, σ) correspondiente al estado actual y la entrada.

El que sigue es un ejemplo de DFA definiendo los cinco elementos que lo componen:

1. $\Sigma = \{0, 1\}$
2. $Q = \{q_0, q_1, q_2, q_3, q_4, q_M\}$
3. q_0
4. $F = \{q_1, q_4\}$
5. La función de transición, δ se define en la Tabla 6.1.

δ	0	1
q_0	q_1	q_2
q_1	q_M	q_M
q_M	q_M	q_M
q_2	q_3	q_4
q_3	q_2	q_3
q_4	q_4	q_2

Tabla 6.1: La función de transición δ

En la Tabla 6.1, la primera columna contiene los estados posibles del autómata, y la primera fila todos los símbolos del alfabeto. Dado un estado actual q_i y un símbolo de entrada σ, la tabla define el estado al que transita el autómata cuando estando en el estado q_i, recibe la entrada σ: $\delta(q_i, \sigma) \in Q$

La característica esencial de un DFA es que δ es una función, por lo que debe estar definida para todos los pares del producto cartesiano $Q \times \Sigma$. Por ello, sea cual fuere el símbolo actual de la entrada y el estado en que se encuentre el autómata, siempre hay un estado siguiente y además este estado se determina de forma unívoca. Dicho de otro modo, la función de transición siempre determina unívocamente el estado al que ha de transitar el autómata dados el estado en que se encuentra y el símbolo que se tiene en la entrada.

Con un DFA siempre se puede asociar un grafo dirigido que se denomina diagrama de transición. Su construcción es como sigue: los vértices del grafo se corresponden con los estados del DFA. Si hay una transición desde el estado q hacia el estado p con la entrada i, entonces deberá haber un arco desde el nodo q hacia el nodo p etiquetado i. Los estados de aceptación se suelen representar mediante un círculo de doble trazo, y el estado de arranque se suele señalar mediante una flecha dirigida hacia ese nodo.

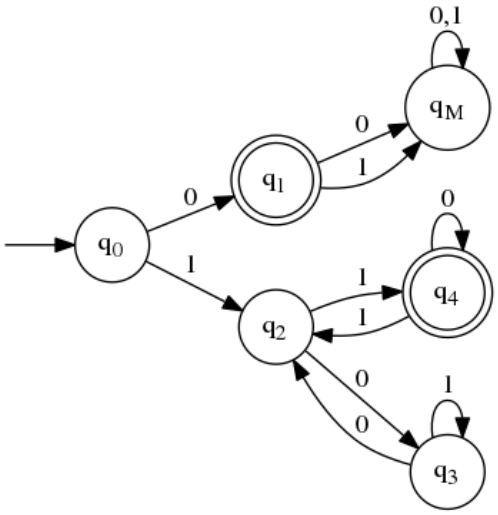

Figura 6.1: Diagrama de transiciones del DFA definido en la Tabla 6.1

El diagrama de transición correspondiente al DFA de la Tabla 6.1 es el que se muestra en la Figura 6.1. Haciendo un cambio de nombres en las etiquetas de los estados y habiendo eliminado el estado de muerte, ese mismo DFA es el que se muestra en la Figura 6.2.

Graphviz [1] es un conjunto de aplicaciones de código abierto que se utilizan para la visualización de gráficos. De las utilidades de Graphviz, centraremos nuestro interés en DOT [2] que es un lenguaje de descripción de gráficos. La sintaxis de DOT es muy simple y puede consultarse en su documentación oficial [3] que define la gramática del lenguaje. Los gráficos DOT son ficheros con la extensión `.gv` que pueden ser manipulados por diversos programas.

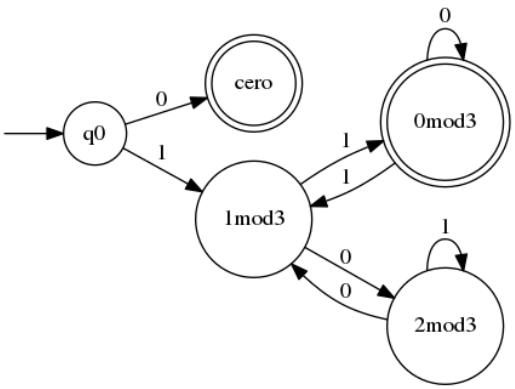

Figura 6.2: DFA que reconoce cadenas binarias que representan números naturales múltiplos de 3

```
1  /* Universidad de La Laguna
2     Grado en Ingenieria Informatica
3     Computabilidad y Algoritmia
4
5     Fichero DOT de representacion de un DFA
6     Lenguaje reconocido: Cadenas binarias que representan numeros
         enteros divisibles por 3 sin ceros por la izquierda
7     ER: 0|(1(01*0)*10*)+
8  */
9  digraph DFA {
10   rankdir=LR;
11   size = "10, 4";
12   d2tstyleonly = true;
13   node [shape = none]; "";
14   node [shape = doublecircle]; "cero" "0mod3";
15   node [shape = circle];
16   "" -> "q0"
17   "1mod3" -> "2mod3" [ label="0" ];
18   "1mod3" -> "0mod3" [ label="1" ];
19   "0mod3" -> "0mod3" [ label="0" ];
20   "0mod3" -> "1mod3" [ label="1" ];
21   "2mod3" -> "1mod3" [ label="0" ];
22   "2mod3" -> "2mod3" [ label="1" ];
23   "q0"    -> "cero"  [ label="0" ];
24   "q0"    -> "1mod3" [ label="1" ];
25 }
```

Listado 6.1: fichero DOT correspondiente al diagrama de transiciones del DFA de la Figura 6.2

El fichero DOT correspondiente al diagrama de transiciones del DFA de la Figura 6.2 es el que se muestra en el Listado 6.1. En este fichero, las líneas 1–8 corresponden con comentarios. Debería escribirse con tildes en ese texto. No se hace aquí por limitaciones del software que se utiliza para generar este documento. Las líneas 13–15 especifican el tipo de trazo que ha de usarse en el gráfico para cada uno de los estados, dependiendo de si son o no de aceptación. Téngase en cuenta que se contempla un estado "ficticio", que no tiene trazo (línea 13 del fichero) del que partiría el arco que marca el estado de arranque. Por último las líneas (16) 17–24 especifican las transiciones entre estados.

Una vez instalado Graphviz, se puede generar un fichero gráfico en formato SVG ejecutando:

```
$ dot -Tsvg < DFA.gv > DFA.svg
```

Graphviz soporta otros formatos gráficos (jp, PostScript, xfig, gif, ...) [4]

6.3. Ejercicio práctico

Desarrollar un programa `DFA2dot.cpp` que lea un fichero de texto en el que figura la especificación de un DFA y genere un fichero DOT con la especificación de ese DFA a partir del cual pueda generarse un fichero gráfico con el diagrama de transiciones del DFA. El programa debiera ejecutarse como:

```
$ ./DFA2dot input.dfa output.gv
```

Donde `input.dfa` es el fichero conteniendo la descripción del DFA y `output.gv` es el correspondiente fichero en formato DOT.

El comportamiento del programa `DFA2dot` al ejecutarse en línea de comandos debiera ser similar al de los comandos de Unix. Este es un requisito que solicitaremos para todos los programas de prácticas de la asignatura a partir de ésta. Así por ejemplo, si se ejecuta `grep` en una terminal Unix (se recomienda estudiar este programa) se obtiene:

```
$ grep
Modo de empleo: grep [OPCIÓN]... PATRÓN [FICHERO]...
Pruebe 'grep --help' para más información.
```

En el caso de `DFA2dot`:

```
$ ./DFA2dot
Modo de empleo: ./DFA2dot input.dfa output.gv
Pruebe 'DFA2dot --help' para más información.
```

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 los DFAs son ficheros de texto con extensión `.dfa`. El Listado 6.2 muestra el fichero de especificación del DFA de la Figura 6.2.

```
1  // Universidad de La Laguna
2  // Grado en Ingenieria Informatica
3  // Computabilidad y Algoritmia
4  //
5  // Fichero de representacion de un DFA
6  // Lenguaje reconocido: Cadenas binarias que representan numeros
       enteros divisibles por 3
7  // ER: (1(01*0)*1|0)+
8  ////////////////////////////////////
9  2
10 0
11 1
12 5
13 q0
14 cero
15 0mod3
16 1mod3
17 2mod3
18 q0
19 2
20 cero
21 0mod3
22 8
23 q0 0 cero
24 q0 1 1mod3
25 1mod3 0 2mod3
26 1mod3 1 0mod3
27 2mod3 0 1mod3
28 2mod3 1 2mod3
29 0mod3 0 0mod3
30 0mod3 1 1mod3
```

Listado 6.2: Fichero de especificación del DFA de la Figura 6.2

Los ficheros de especificación de DFAs 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 (primera línea después de los comentarios, en el Listado 6.2) seguida de N líneas, cada una de las cuales contiene un símbolo 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.

Todas las líneas de un fichero `.dfa` que comiencen con los caracteres `//` corresponden a comentarios, y deben ser ignorados por el programa a la hora de procesar el fichero.

6.4. 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.
- 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.
- Se requiere en esta práctica que, en la sesión de evaluación de la misma, todos los ficheros con códido fuente se alojen en un único directorio junto con el fichero Makefile de compilación.
- Se requiere en esta práctica que todos los atributos de las clases definidas en el proyecto tengan 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 [5].

- Utilice asimismo estas [6] [7] referencias para mejorar la calidad de la documentación de su código fuente.

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] Graphviz. http://www.graphviz.org/

[2] DOT (graph description language) https://en.wikipedia.org/wiki/DOT_(graph_description_language)

[3] The DOT Language. https://www.graphviz.org/doc/info/lang.html

[4] Graphviz Outuput Formats https://www.graphviz.org/doc/info/output.html

[5] Doxygen http://www.doxygen.nl/index.html

[6] Diez consejos para mejorar tus comentarios de código fuente https://www.genbeta.com/desarrollo/diez-consejos-para-mejorar-tus-comentarios-de-codigo-fuente

[7] Tips para la escritura de documentación interna http://www.galeon.com/neoprogramadores/howdocod.htm