Universidad de La Laguna

ESIT – INFORMÁTICA
Departamento de Ingeniería Informática y de Sistemas

Computabilidad y Algoritmia
Curso 2019-2020

PRACTICA 7: Autómatas Finitos Deterministas

7.1. Objetivos

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

7.2. 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:

- El alumnado ha de acreditar conocimientos para trabajar con la shell de GNU/Linux en su VM.
- El alumnado ha de acreditar capacidad para editar ficheros remotos en la VM de la asignautra usando VSC.
- El código ha de estar escrito de acuerdo al estándar de la guía de Estilo de Google para C++
- El programa desarrollado deberá compilarse utilizando la herramienta make y un fichero Makefile.
- El comportamiento del programa debe ajustarse a lo solicitado en este documento.
- Ha de acreditarse capacidad para introducir cambios en el programa desarrollado.
- Modularidad: el programa ha de escribirse de modo que las diferentes funcionalidades que se precisen sean encapsuladas en funciones y/o métodos cuya extensión textual se mantenga acotada.

- El programa ha de ser fiel al paradigma de programación orientada a objetos.
- Se requiere 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 que todos los atributos de las clases definidas en el proyecto tengan un comentario descriptivo de la finalidad del atributo en cuestión.
- Se requiere que los comentarios del código fuente sigan el formato especificado por Doxygen [1].
- Utilice asimismo esta [2] referencia 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.

7.3. 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. $\delta : Q \times \Sigma \rightarrow Q$ 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 7.1.

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 7.1: La función de transición δ

En la Tabla 7.1, 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.

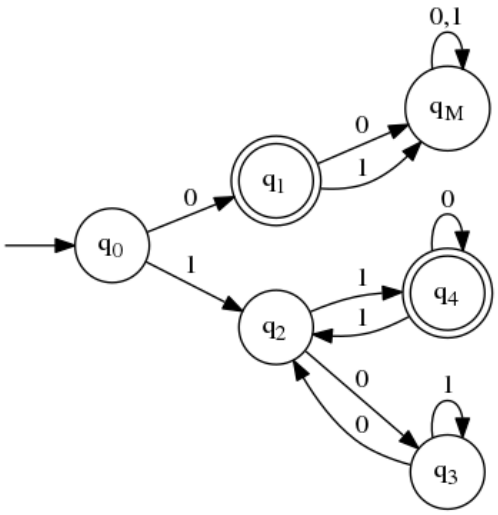

Figura 7.1: Diagrama de transiciones del DFA definido en la Tabla 7.1

El diagrama de transición correspondiente al DFA de la Tabla 7.1 es el que se muestra en la Figura 7.1. 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 7.2.

7.4. Ejercicio práctico

Desarrollar un programa `dfa_simulation.cc` que evalúe el reconocimiento de una serie de cadenas por parte del DFA.

El programa debe ejecutarse como:

```
$ ./dfa_simulation input.dfa input.txt output.txt
```

Donde los tres parámetros pasados en la línea de comandos corresponden en este orden con:

- Un fichero de texto en el que figura la especificación de un DFA

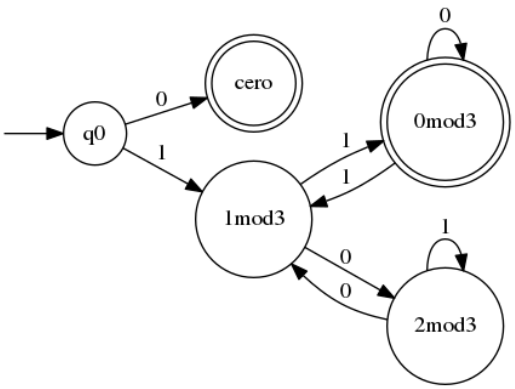

Figura 7.2: DFA que reconoce cadenas binarias que representan números naturales múltiplos de 3

- Un fichero de texto con extensión `.txt` en el que figura una serie de cadenas (una cadena por línea) sobre el alfabeto del DFA anterior.
- Un fichero de texto de salida con extensión `.txt` en el que el programa ha de escribir las mismas cadenas del fichero de entrada seguidas de un texto `-- Sí / No` indicativo de la aceptación (o no) de la cadena en cuestión.

El comportamiento del programa 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 sin parámetros el comando `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 del programa a desarrollar:

```
$ ./dfa_simulation
Modo de empleo: ./dfa_simulation input.dfa input.txt output.txt
Pruebe 'dfa_simulation --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.

Se explica a continuación el contenido de los ficheros de entrada que especifican los DFAs. El Listado 7.1 muestra el fichero de especificación del DFA de la Figura 7.2.

```
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
7  // enteros divisibles por 3
8  // ER: (1(01*0)*1|0)+
9  ////////////////////////////////////
10 2
11 0
12 1
13 5
14 q0
15 cero
16 0mod3
17 1mod3
18 2mod3
19 q0
20 2
21 cero
22 0mod3
23 8
24 q0 0 cero
25 q0 1 1mod3
26 1mod3 0 2mod3
27 1mod3 1 0mod3
28 2mod3 0 1mod3
29 2mod3 1 2mod3
30 0mod3 0 0mod3
31 0mod3 1 1mod3
```

Listado 7.1: Fichero de especificación del DFA de la Figura 7.2

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 7.1) seguida de N líneas, cada una de las cuales contiene un símbolo del alfabeto. Cada símbolo del alfabeto debe ser un caracter imprimible.

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 q_i, sim, q_f separadas por espacios. Cada una de estas líneas indica una transición del estado q_i al q_f con símbolo *sim*, es decir, $\delta(q_i, sim) = q_f$. q_i y q_f 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.

7.5. Consideraciones de implementación

La idea central de la Programación Orientada a Objetos, OOP (*Object Oriented Programming*) es dividir los programas en piezas más pequeñas y hacer que cada pieza sea responsable de gestionar su propio estado. De este modo, el conocimiento sobre la forma en que funciona una pieza del programa puede mantenerse local a esa pieza. Alguien que trabaje en el resto del programa no tiene que recordar o incluso ser consciente de ese conocimiento. Siempre que estos detalles locales cambien, sólo el código directamente relacionado con esos detalles precisa ser actualizado.

Las diferentes piezas de un programa de este tipo interactúan entre sí a través de lo que se llama interfaces: conjuntos limitados de funciones que proporcionan una funcionalidad útil a un nivel más abstracto, ocultando su implementación precisa. Tales piezas que constituyen los programas se modelan usando objetos. Su interfaz consiste en un conjunto específico de métodos y atributos. Los atributos que forman parte de la interfaz se dicen públicos. Los que no deben ser visibles al código externo, se denominan privados. Separar la interfaz de la implementación es una buena idea. Se suele denominar encapsulamiento.

C++ es un lenguaje orientado a objetos. Si se hace programación orientada a objetos, los programas forzosamente han de contemplar objetos, y por tanto clases. Cualquier programa que se haga ha de modelar el problema que se aborda mediante la definición de clases y los correspondientes objetos. Los objetos pueden componerse: un objeto "coche" está constituido por objetos "ruedas", "carrocería" o "motor". La herencia es otro potente mecanismo que permite hacer que las clases (objetos) herederas de una determinada clase posean todas las funcionalidades (métodos) y datos (atributos) de la clase de la que descienden. De forma inicial es más simple la composición de objetos

que la herencia, pero ambos conceptos son de enorme utilidad a la hora de desarrollar aplicaciones complejas. Cuanto más compleja es la aplicación que se desarrolla mayor es el número de clases involucradas. En los programas que desarrollará en esta asignatura será frecuente la necesidad de componer objetos, mientras que la herencia tal vez tenga menos oportunidades de ser necesaria.

Tenga en cuenta las siguientes consideraciones:

- No traslade a su programa la notación que se utiliza en este documento, ni en la teoría de Autómatas Finitos. Por ejemplo, el cardinal del alfabeto de su autómata no debiera almacenarse en una variable cuyo identificador sea `N`. Al menos por dos razones: porque no sigue lo especificado en la Guía de Estilo respecto a la elección de identificadores y más importante aún, porque no es significativo. No utilice identificadores de un único carácter, salvo para situaciones muy concretas.
- Favorezca el uso de las clases de la STL, particularmente `std::array`, `std::vector` o `std::string` frente al uso de estructuras dinámicas de memoria gestionadas a través de punteros.
- Construya su programa de forma incremental y depure las diferentes funcionalidades que vaya introduciendo.
- En el programa parece ineludible la necesidad de desarrollar una clase `Dfa`. Estudie las componentes que definen a un DFA y vea cómo trasladar esas componentes a su clase `Dfa`.
- Valore análogamente qué otras clases identifica Ud. en el marco del problema que se considera en este ejercicio. Estudie esta referencia [3] para practicar la identificación de clases y objetos en su programa.

Bibliografía

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

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

[3] Cómo identificar clases y objetos http://www.comscigate.com/uml/DeitelUML/Deitel01/Deitel02/ch03.htm