Universidad de Huelva

Universidad de Huelva

Grado en Ingeniería Informática

Autómatas Finitos Deterministas y No Deterministas

Memoria de Prácticas

Autores:
Alberto Fernández Merchán
Juan Manuel Ruíz Pérez

Asignatura: Algorítmica y Modelos de Computación

Enero 2022

Índice

1. Introducción **2**
1.1. Definición . . . 2
1.1.1. Autómata Finito Determinista . . . 2
1.1.2. Autómata Finito No Determinista . . . 2
1.2. Objetivos de la práctica . . . 3

2. Implementación **3**
2.1. Autómata . . . 3

3. Utilización de la aplicación **5**

4. Problemas encontrados **6**
4.1. Elaboración de la salida gráfica . . . 6
4.2. Elaboración de una clase única de autómatas . . . 6

5. Ejemplos de autómatas **7**
5.1. Autómata Finito Determinista . . . 7
5.2. Autómata Finito No Determinista . . . 7

6. Webgrafía **8**

1. Introducción

1.1. Definición

Un autómata finito es una máquina de estados que está formada por cinco componentes:

1. Un conjunto finito de estados (E).
2. Un conjunto finito de símbolos. (S)
3. Un conjunto finito de transiciones. (f)
4. Un estado inicial.
5. Un conjunto de estados finales (F) de forma que $F \subseteq E$.

El comportamiento de un autómata finito es el siguiente:

Al comienzo de la ejecución, el autómata se encuentra en el **estado inicial**. Tras este instante, comienza a leer secuencialmente los símbolos de la cadena de entrada. Cada símbolo provocará una **transición** del autómata. Una vez termina la cadena, comprueba si el estado en el que ha quedado el autómata es un estado final (acepta) o no (rechaza).

A la hora de representar un autómata finito podemos hacerlo mediante una tabla o mediante un diagrama de estados.
Estos autómatas finitos se dividen en dos tipos: deterministas y no deterministas.

1.1.1. Autómata Finito Determinista

En un autómata finito determinista (AFD) solo existe un camino posible. Es decir, dado un estado y un símbolo, no puede existir más de una transición.
Además, para cada símbolo debe existir una transición en cada estado y para cada transición, el estado destino no puede ser un conjunto de más de un estado.

1.1.2. Autómata Finito No Determinista

Un autómata finito no determinista (AFND) puede tener transiciones con el mismo origen y símbolo que se dirija a un conjunto de uno o más estados. Además pueden tener λ-transiciones. Este tipo de transiciones no consumen ningún símbolo de la cadena de entrada, es decir, el autómata puede cambiar de estado sin leer ningún caracter.

Un AFND puede terminar en varios estados a la vez, de forma que una cadena será aceptada si y solo si existe un estado final en el conjunto de estados resultante al terminar de leer la cadena de entrada. Dicho de otra forma una cadena será aceptada si la intersección entre el conjunto de estados finales y el conjunto de estados resultantes no es vacío.
$M = \{x / F \cap f(q_0, x) \neq \emptyset\}$ Siendo x la cadena de entrada.

1.2. Objetivos de la práctica

Esta práctica consiste en realizar una simulación de estos dos tipos de **autómatas finitos** en lenguaje Java.

Para ello tendremos que implementar las clases AFD y AFND y hacer que ambas implementen la interfaz "Proceso"que nos proporciona el enunciado de la práctica.

Además, tendremos que realizar una salida gráfica por pantalla que nos permita crear a nuestro propio autómata, comprobar que una cadena de símbolos pueda ser aceptada o rechazada por el autómata y realizar el seguimiento de sus transiciones paso a paso.

2. Implementación

Primero hemos creado las clases "AFD"y "AFND"que contienen los métodos que usarán los autómatas finitos deterministas y los autómatas finitos no deterministas correspondientemente.
Además, hemos creado métodos para poder leer los ficheros que nos presentan en el enunciado de la práctica. De esta manera podemos crear un autómata de tres formas:

- Escribiendo manualmente los datos.
- Pasándole un fichero con extensión *.auto* con la estructura del fichero que hay en el enunciado.
- Utilizando el asistente de creación de autómatas.

En cuanto a la forma de representar el autómata, lo haremos de forma diagramática para que puedan observarse mejor las transiciones.

2.1. Autómata

Como un autómata finito determinista está englobado en un autómata finito no determinista, hemos decidido unificar ambas clases en una sola.

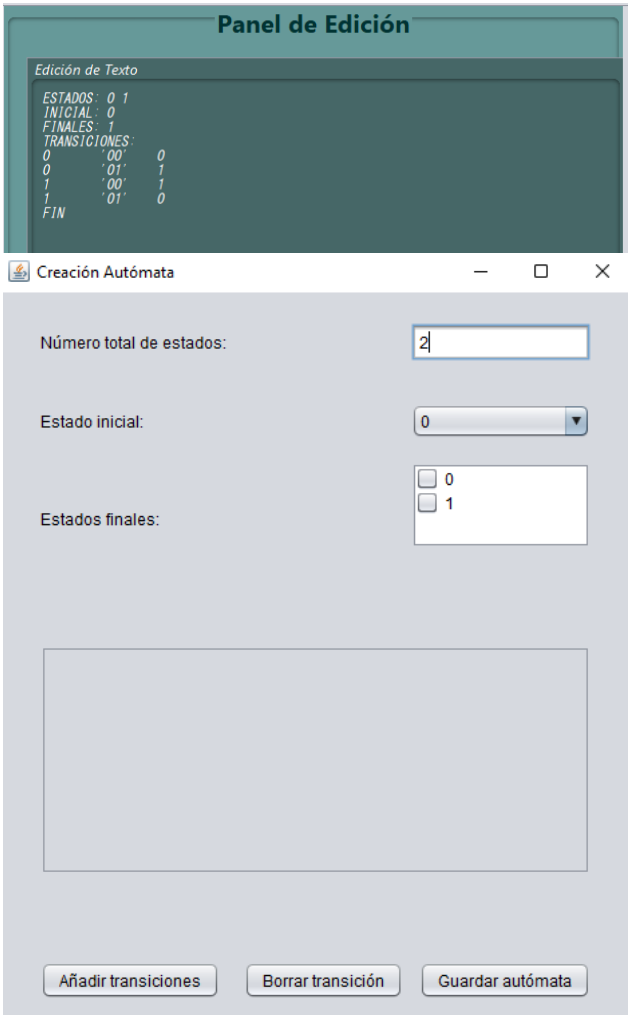
Para ello hemos implementado los distintos métodos que definen al autómata según el modelo MVC:

- **Modelo**:
  - **Automata**: Clase que representa el autómata que se define en el cuadro de texto de la ventana principal.
  - **Proceso**: Interfaz que debe implementar la clase "Autómata"según las especificaciones de la práctica.
  - **Transicion**: Clase que representa las transiciones normales del autómata. Consiste en 3 elementos: el estado origen, el símbolo que produce la transición y el conjunto de estados destino. En el caso de un AFD, el conjunto de estados destino estará formado por tan solo un elemento, mientras que para un AFND, el conjunto puede estar formado por más de un estado.
  - **TransicionL**: Clase que representa las transiciones lambda (*lambda*-transiciones) de un AFND. Por otro lado, un AFD no contiene ningún objeto del tipo "TransicionL".

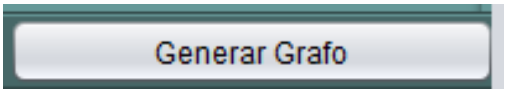
- **Vista**:
  - **Lienzo**: Clase que da forma al grafo que se dibuja en la parte izquierda de la pantalla.
  - **Mensaje**: Clase usada para lanzar mensajes de error y de información. Se usa, mayormente, al capturar excepciones.
  - **VentanaAutomata**: Clase que da forma a la ventana que se muestra al usar la opción de "Crear autómata". Contiene 1 cuadro de inserción de texto, una caja de selección de estado inicial, una lista de selección de estados finales, una tabla donde se muestran las transiciones, y 3 botones para guardar el autómata, crear transiciones o borrarlas.
  - **VentanaTransiciones**: Clase que da forma a la ventana que se abre al seleccionar la opción "Añadir transición"de la clase VentanaAutomata. Contiene 2 cajas de selección de estados y un campo de inserción de texto, además de un botón para guardar la transición.
  - **VentanaPrincipal**: Es la ventana que se muestra al principio de la ejecución del programa. Contiene el lienzo, además de un panel de edición de texto, y varios botones de creación de autómatas, carga y guardado de ficheros, generación de grafos y un menú superior que contiene botones para, nuevamente, trabajar con ficheros, y para abrir la ventana de ayuda y la de créditos.
  - **VentanaAyuda**: Clase que muestra una ventana en la que se proporciona información sobre el funcionamiento del programa.
- **Controlador**:
  - **Controlador Menu**: Es la clase que controla las acciones de la ventana de asistencia de creación de autómatas.
  - **Controlador Principal**: Es la clase que controla las acciones de la ventana principal.

3. Utilización de la aplicación

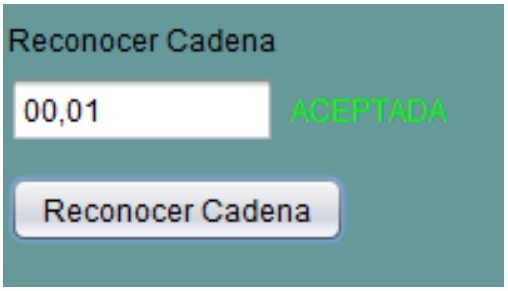
En este apartado describiremos como se utiliza la aplicación.
En primer lugar, deberemos escribir, generar o cargar los datos del fichero de descripción del autómata. Para ello podremos escribir la descripción directamente en el editor de texto, usar el botón de cargar fichero para utilizar un fichero con extensión .auto o utilizar el asistente de creación de autómatas.



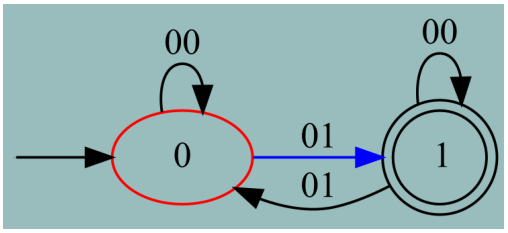
Una vez escrita la descripción del autómata, podremos generar el grafo que representa las transiciones de este. Para ello pulsaremos sobre el botón “Generar Grafo”. Tras un segundo, se generará una imagen a la izquierda de la ventana principal.



Una vez generado el autómata, podremos probar cadenas en él. Insertando una cadena formada por los símbolos aceptados por el autómata y separados por comas en la caja de texto “Reconocer Cadena”.
Una vez escrita la cadena, podremos darle al botón “Reconocer Cadena”para que nos diga si la cadena ha sido aceptada o, por el contrario, rechazada.



Si queremos ver las transiciones del autómata paso a paso, podemos darle al botón de "Siguiente"o "Anterior"para navegar por las transiciones de dicho autómata.
Los símbolos marcados en rojo representan los estados en los que se encuentra el autómata, mientras que los marcados en azul son las transiciones siguientes que leerá el autómata.



4. Problemas encontrados

Durante la elaboración de la práctica hemos afrontado diferentes problemas a la hora de implementar el código. A continuación se detallan dichos inconvenientes.

4.1. Elaboración de la salida gráfica

Para implementar la salida gráfica del diagrama de estados del autómata hemos decidido usar el motor gráfico de la biblioteca "*Graphviz*". Esta biblioteca permite representar grafos mediante un código de descripción de grafos.

Hemos tenido inconvenientes a la hora de aprender a utilizar los métodos de Graphviz para traducir un objeto autómata a un fichero *.dot* (extensión del fichero de descripción de grafos).

Hemos buscado en la documentación de la biblioteca y hemos encontrado ejemplos sobre como utilizar los métodos que incluye.

4.2. Elaboración de una clase única de autómatas

Viendo que la estructura de los Autómatas Finitos No Deterministas engloba a la de los Autómatas Finitos Deterministas, podemos realizar una única clase que agrupe a ambos tipos.

La diferencia fundamental entre ambos es que el No Determinista puede tener varios estados destino después de cada transición, mientras que el Determinista solo puede tener 1 estado destino por cada transición, además, el no determinista puede incluir lambda transiciones.

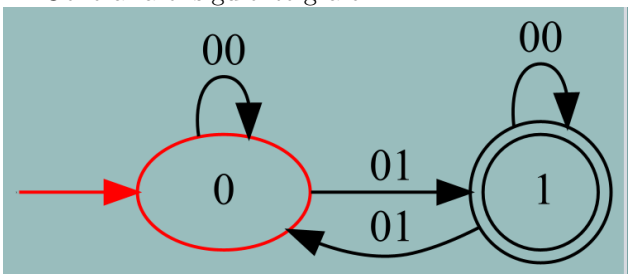
5. Ejemplos de autómatas

5.1. Autómata Finito Determinista

A continuación se muestra un código de ejemplo para describir un autómata finito determinista.

```
ESTADOS: 0 1
INICIAL: 0
FINALES: 1
TRANSICIONES:
0 '00' 0
0 '01' 1
1 '00' 1
1 '01' 0
FIN
```

Generaría el siguiente grafo:

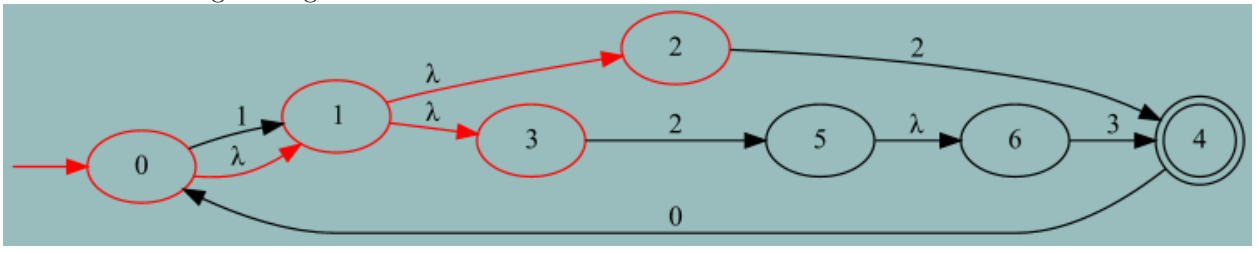


5.2. Autómata Finito No Determinista

A continuación se muestra un código de ejemplo para describir un autómata finito no determinista.

```
ESTADOS: 0 1 2 3 4 5 6
INICIAL: 0
FINALES: 4
TRANSICIONES:
0 '1' [1]
3 '2' [5]
2 '2' [4]
6 '3' [4]
4 '0' [0]
1 'lambda' [2, 3]
5 'lambda' [6]
0 'lambda' [1]
FIN
```

Generaría el siguiente grafo:



6. Webgrafía

[1] Documentación de Graphviz : `https://graphviz.org/doc/info/lang.html`

[2] GitHub de Graphviz : `https://github.com/nidi3/graphviz-java`