Universidad de La Laguna

Grado en Ingeniería Informática

Computabilidad y Algoritmia

PRÁCTICA 5

Autómatas finitos en JFLAP

Factor de ponderación: 6

1. Objetivos

El objetivo de esta práctica consiste en introducir los fundamentos básicos de los *autómatas finitos* [1]. Se comprobará y verificará el funcionamiento de algunos ejemplos y se diseñarán autómatas finitos que reconozcan ciertos lenguajes especificados en el enunciado. Para simular el comportamiento de los autómatas diseñados utilizaremos la herramienta JFLAP [2]. JFLAP es un paquete de gran interés en el ámbito de la enseñanza de lenguajes formales pues constituye una herramienta interactiva para visualización y simulación de autómatas finitos y máquinas de Turing, entre otros. Puesto que en esta práctica utilizaremos JFLAP para diseñar y simular autómatas finitos, será necesario descargar e instalar la herramienta siguiendo estos sencillos pasos [3].

Para esta práctica será necesario realizar los ejercicios propuestos en este enunciado y llevarlos resueltos a la clase práctica de laboratorio. Las propuestas de solución de los ejercicios se entregarán a modo de informe. Durante la sesión presencial se les podrá proponer la resolución de nuevos ejercicios.

Tal y como venimos insistiendo a lo largo de las prácticas de la asignatura, 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.

2. Autómatas finitos

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:

- Un alfabeto de entrada Σ
- Una colección finita de estados Q
- Un estado inicial s
- Una colección F de estados finales o de aceptación
- Una función $\delta : Q \times \Sigma \to Q$ que determina el único estado siguiente para el par (q_i, σ) correspondiente al estado actual y la entrada.

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.

Para hacer el diseño de dicho diagrama de transiciones podríamos utilizar cualquier herramienta de diseño o más específicamente alguna herramienta para definición o visualización de grafos [5]. Sin embargo, si queremos ir más allá y utilizar herramientas que nos permitan realizar una simulación de un DFA tendremos que buscar alternativas específicas para autómatas finitos.

En este sentido, JFLAP [2] es una aplicación diseñada para experimentar con tópicos de la asignatura: autómatas finitos deterministas, autómatas finitos no deterministas, máquinas de Turing, etc. En esta práctica proponemos la utilización de la herramienta JFLAP para diseñar y, posteriormente simular el comportamiento de autómatas finitos.

En primer lugar, deberá descargar e instalar JFLAP en su ordenador. Para ello, siga los siguientes pasos [3]:

1. Descargar el fichero JFLAP.jar:
   https://campusingenieriaytecnologia2223.ull.es/pluginfile.php/8439/mod_page/content/7/JFLAP.jar
2. Para ejecutarlo: `java -jar JFLAP.jar`

A modo de ejemplo vamos a diseñar un DFA para reconocer *cadenas binarias que comiencen por 1 o que contengan dos ceros consecutivos (que contengan la subcadena 00)*. Al realizar el diseño, debemos pensar que si la cadena tiene como primer símbolo un 1, entonces directamente podremos pasar a un estado de aceptación porque a partir de ese momento, sea cual sea el resto de la cadena, dicha cadena nunca dejará de cumplir la condición de "comenzar por un 1". Si por el contrario, la cadena comienza por 0, entonces nos tendremos que centrar en detectar si en algún momento nos encontramos con dos ceros consecutivos. Para ello, necesitaremos dos estados adicionales: uno para cuando nos encontremos el primer 0 y otro para cuando el siguiente símbolo no sea otro 0 sino un 1. De esta forma sabremos que tendremos que seguir esperando a que vuelva otro 0 y con él la posibilidad de cumplir con la condición de tener los dos ceros consecutivos. Por su parte, desde que encontremos los primeros dos ceros consecutivos ya podremos pasar al estado de aceptación: a partir de ahí siempre aceptaremos la cadena, independientemente del resto de símbolos que contenga.

Este DFA puede definirse formalmente como $M = (\Sigma, Q, q_0, F, \delta)$ donde:

- El alfabeto de entrada es $\Sigma = \{0, 1\}$
- El conjunto de estados del autómata será $Q = \{q_0, q_1, q_2, q_3\}$
- El estado q_0 es el estado inicial o de arranque
- El estado q_1 es el único estado de aceptación $F = \{q_1\}$
- La función de transición para este autómata sería la siguiente:

δ	0	1
q_0	q_2	q_1
q_1	q_1	q_1
q_2	q_1	q_3
q_3	q_2	q_3

Para diseñar un DFA utilizando JFLAP, tendremos que ejecutar la aplicación y escoger del menú principal la opción `Finite Automaton`:

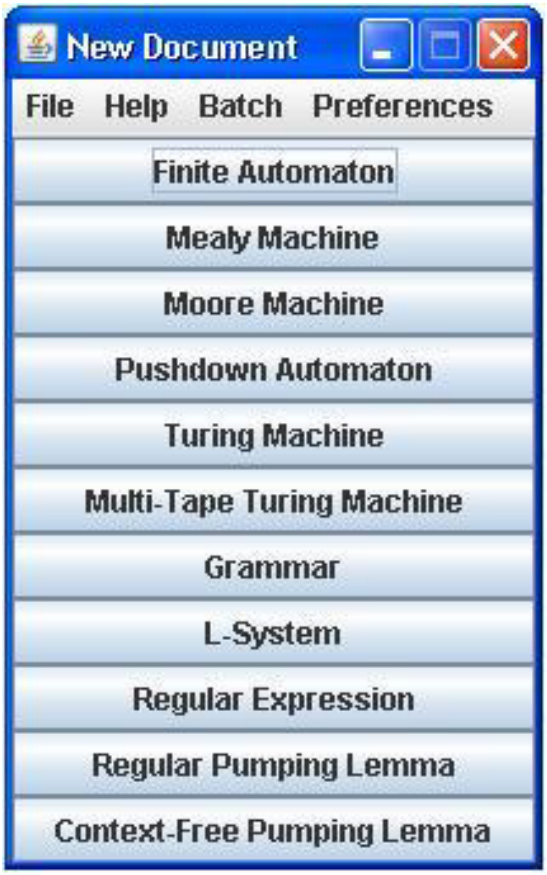

Figura 1: Menú principal de la herramienta JFLAP

A continuación encontrará un videotutorial [6] en el que se describe paso a paso cómo diseñar este DFA mediante la utilización de la herramienta JFLAP:

https://youtu.be/EzWxFMfqeFs

Tras seguir los pasos descritos en el tutorial, en la ventana de diseño de JFLAP deberíamos tener el DFA que se muestra en la Figura 2.

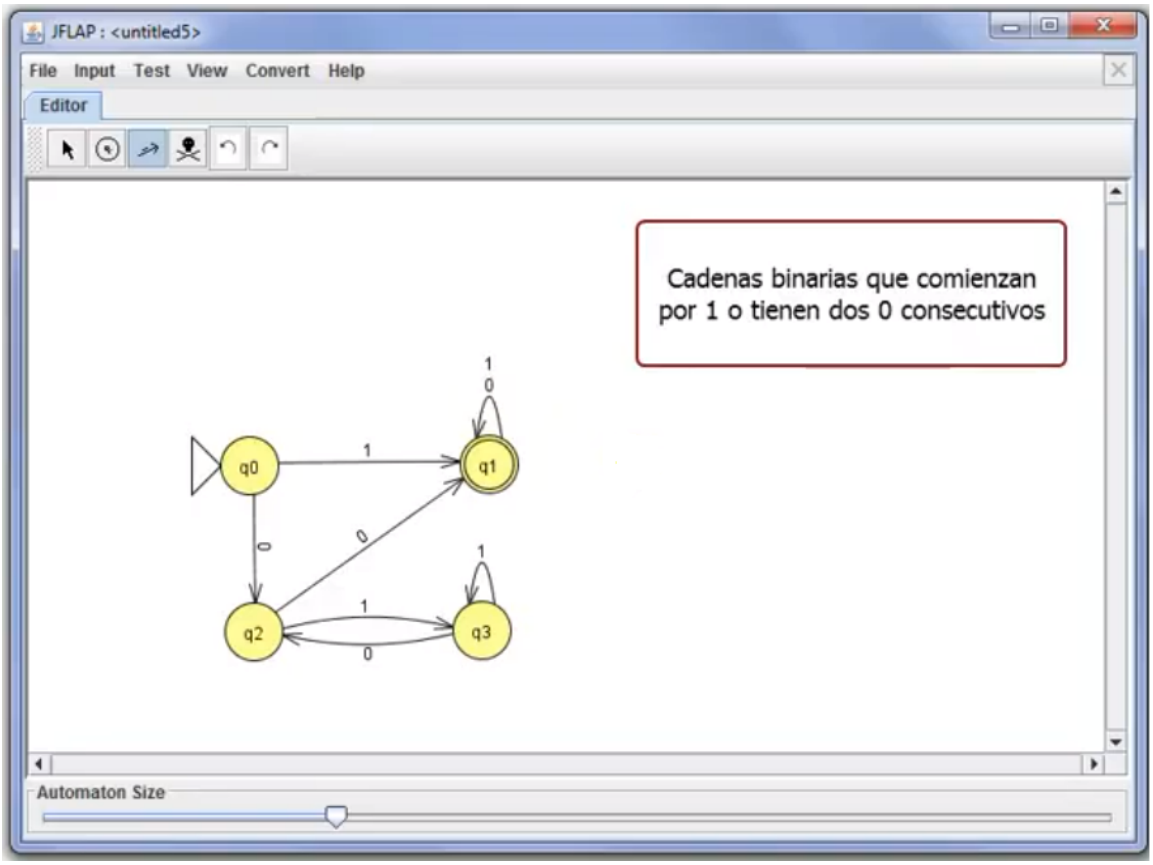

Figura 2: DFA que reconoce $L(r)$ donde $r = 1(0|1)* \ | \ (0|1)*00 \ (0|1)*$

A partir del autómata diseñado podremos realizar simulaciones "paso a paso" para ver el comportamiento del DFA para una determinada cadena, o bien, llevar a cabo una simulación múltiple para ver cuáles de una serie de cadenas de entrada son las que se aceptan o rechazan por el autómata en cuestión.

Para simular el comportamiento del autómata paso a paso, desde la ventana de diseño utilizaremos la opción `Step` del menú `Input`. Esto nos permitirá simular el comportamiento del autómata, paso a paso, con una determinada cadena de entrada. Por ejemplo, si fijamos como entrada la cadena `01100` podremos ir viendo cómo se realiza la transición correspondiente en función del estado actual y del símbolo que tenemos en la entrada. Se ejecutará un paso por cada símbolo de la cadena de entrada hasta que finalmente lleguemos al último. En ese momento, la cadena se aceptará o no en función de si hemos terminado el proceso en un estado de aceptación o un estado que no es de aceptación, tal y como puede apreciarse en la Figura 3. Esta simulación nos permite visualizar de una forma clara e interactiva el comportamiento del autómata en cada caso, ayudando a entender el funcionamiento interno de dichos modelos de cómputo.

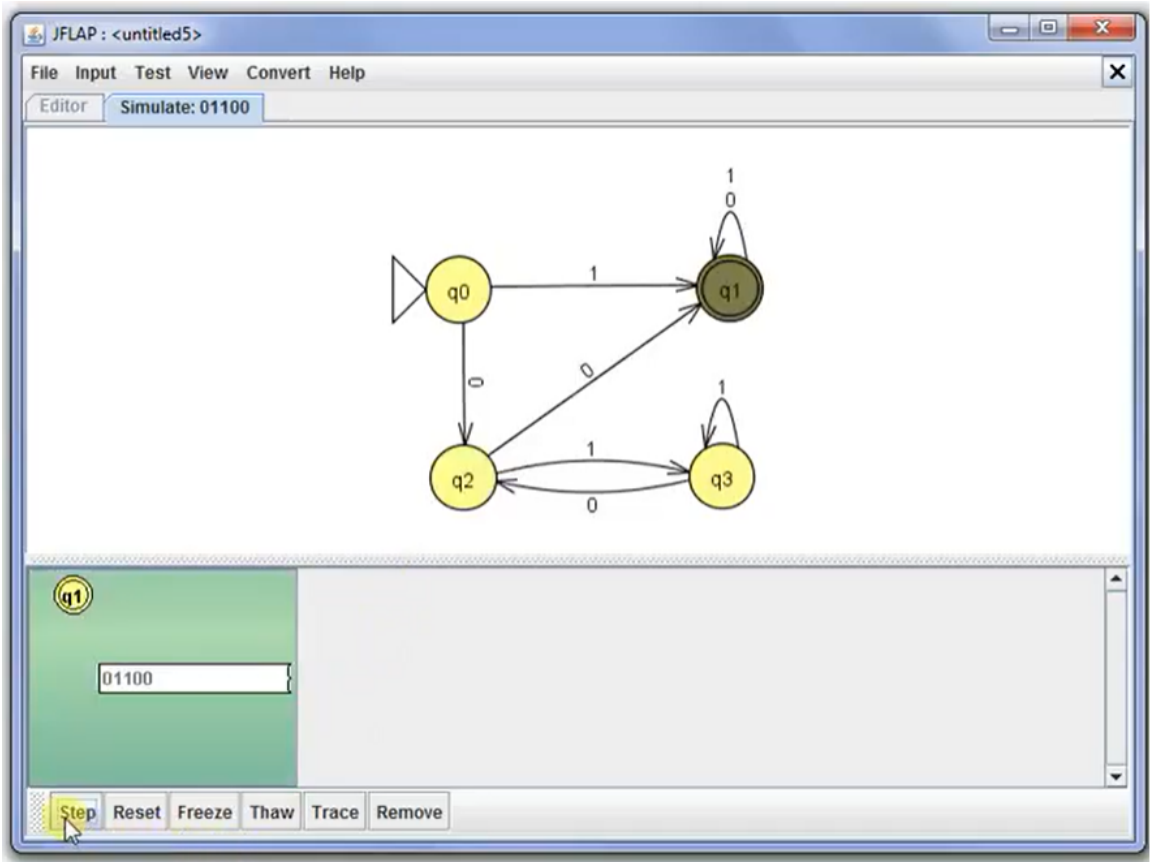

Figura 3: Simulación del DFA para la cadena 01100

En el videotutorial [6] también encontrará un ejemplo de cómo simular el autómata con múltiples entradas. Para ello podríamos utilizar la opción `Multiple Run` disponible también en el menú `Input`. Con esta opción de simulación, JFLAP ofrece la posibilidad de obtener de una forma rápida y sencilla la respuesta a qué cadenas de entrada serían aceptadas por el autómata y cuáles rechazadas.

La herramienta JFLAP puede ser utilizada de una forma similar para el diseño y la simulación de autómatas finitos no deterministas, AFN (o NFA de su denominación en inglés *Non-Deterministic Finite Automaton*). Estos autómatas pueden reconocer exactamente los mismos lenguajes que son capaces de reconocer los autómatas finitos deterministas pero, en algunos casos nos ofrecen una alternativa de diseño bastante más sencilla que la que obtendríamos al usar su equivalente determinista. La principal diferencia con respecto a un DFA es que un NFA permite que desde un estado se realicen cero, una o más transiciones mediante el mismo símbolo de entrada. Esto decir, desde un estado actual y ante un determinado símbolo de entrada, podemos no tener un estado siguiente, o bien, podemos tener varios estados siguientes. Este comportamiento *no determinista* es el que precisamente le confiere a los NFAs su rango distintivo frente a los DFAs. Téngase en cuenta que cualquier DFA es un caso particular de NFA en el que hay uno y solo un estado siguiente desde cada estado y para cada símbolo de entrada.

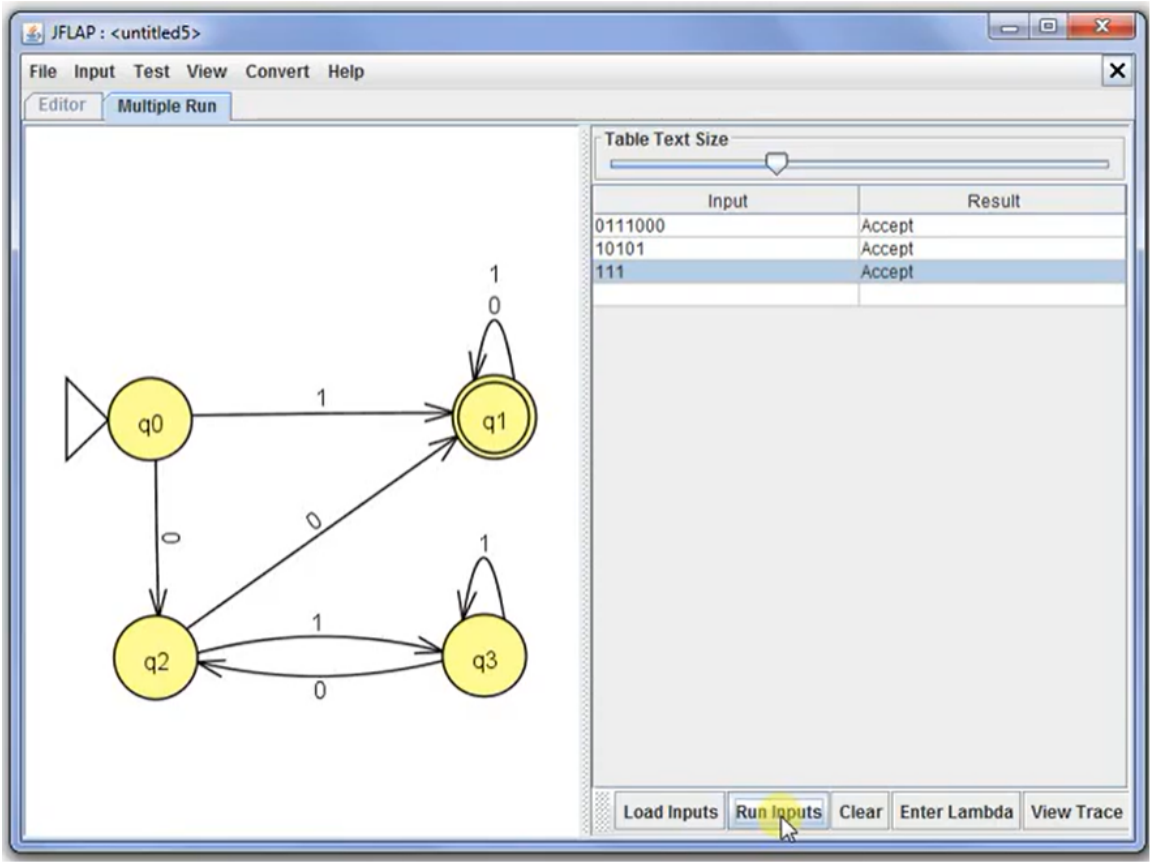

Figura 4: Simulación del DFA para múltiples cadenas

Formalmente, un *autómata finito no determinista* M es una colección de cinco elementos $M = (\Sigma, Q, q_0, F, \delta)$:

- Un alfabeto de entrada Σ
- Una colección finita de estados Q
- Un estado inicial s
- Un conjunto F de estados finales o de aceptación
- Una función de transición $\delta : (Q \times \Sigma) \to 2^Q$. Téngase en cuenta que $\emptyset \in 2^Q$ y también $Q \in 2^Q$.

Atendiendo a la función de transición anterior, dado un estado y un símbolo, el NFA transita a un **conjunto de estados** en lugar de a un único estado. No existe nada en el modelo que determine la elección: el comportamiento es no-determinista.

A modo de ejemplo, vamos a diseñar un NFA que reconozca aquellas cadenas sobre el alfabeto $\Sigma = \{0, 1\}$ que tengan un 1 en la antepenúltima posición. Como sabemos el diseño de un DFA que reconozca este lenguaje puede no resultar trivial. En la Figura 5 se puede

apreciar que es necesario controlar todas las posiciones terminaciones de la cadena para poder discernir entre aquellos unos que pueden llegar a convertirse en símbolos válidos para la antepenúltima posición.

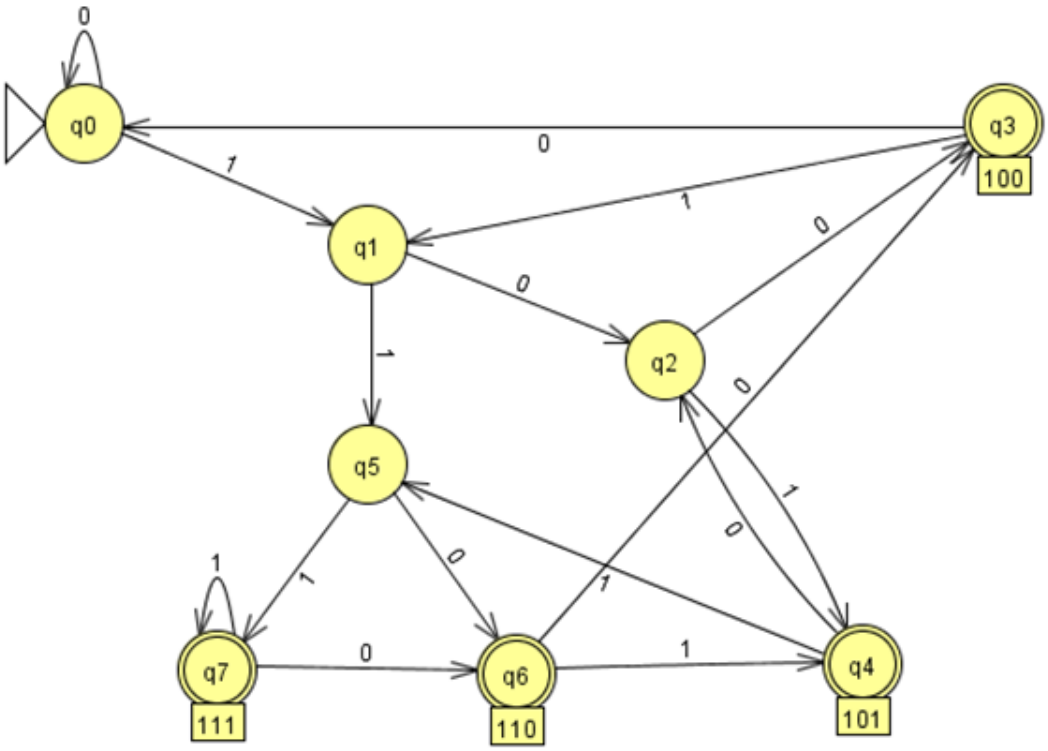

Figura 5: DFA que reconoce $L(r)$ donde $r = (0|1)*1\ (0|1)(0|1)$

Sin embargo, diseñar un NFA que reconozca este mismo lenguaje puede resultar mucho más sencillo e intuitivo, tal y como se muestra en el siguiente videotutorial [7]:

https://youtu.be/audjcmz7ons

Tras seguir los pasos descritos en el tutorial, en la ventana de diseño de JFLAP deberíamos tener el NFA que se muestra en la Figura 6. Además, es interesante cómo en dicho videotutorial [7] se describen los pasos para convertir un NFA en un DFA equivalente (mediante la aplicación del algoritmo de construccioón de subconjuntos). También se muestra la existencia de una opción en la herramienta para determinar si dos automátas finitos dados como entrada son equivalentes (reconocen el mismo lenguaje) o no. Estas funcionalidades pueden ser de interés para verificar el diseño de autómatas así como la correcta aplicación de algoritmos como la construcción de Thompson, la construcción de subconjuntos o la minimización de estados.

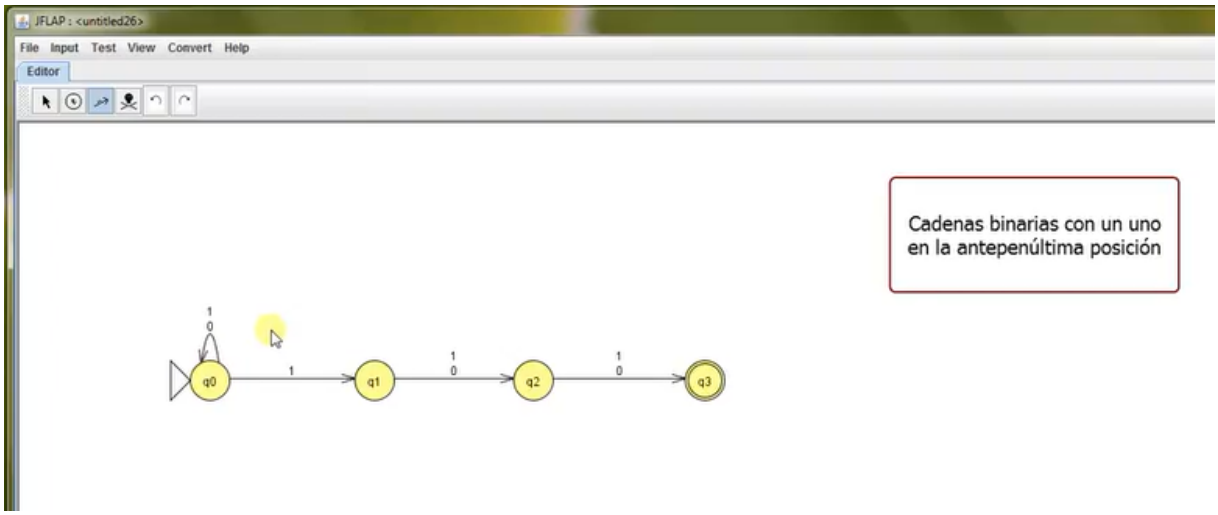

Figura 6: NFA que reconoce $L(r)$ donde $r = (0|1)* 1\ (0|1)(0|1)$

3. Ejercicios: diseño de DFAs

Con el objetivo de profundizar en el diseño y la simulación de autómatas finitos se proponen los siguientes ejercicios. La idea es que se realice, para cada ejercicio, un diseño de autómata finito determinista en JFLAP y se realicen las simulaciones oportunas para comprobar su correcto funcionamiento. Para verificar el comportamiento de los autómatas finitos diseñados, se debería comprobar el comportamiento de los mismos para al menos tres cadenas que sean aceptadas y para otras tres que no sean aceptadas, aunque lo ideal sería utilizar en cada caso una relación más exhaustiva de cadenas de prueba.

1. Diseñar un autómata finito determinista que reconozca cadenas binarias que contengan un número impar de unos y un número impar de ceros.
2. Diseñar un autómata finito determinista que reconozca cadenas binarias de longitud par.
3. Diseñar un autómata finito determinista que reconozca cadenas sobre el alfabeto $\Sigma = \{a, b, c\}$ que no contengan la subcadena `abc`.
4. Diseñar un autómata finito determinista que acepte números reales. El alfabeto que usa el autómata se define como $\Sigma = \{+, -, ., E, e, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ y las cadenas a aceptar se definen de la siguiente forma:
   - La cadena comienza opcionalmente por el símbolo "+" o "-"
   - A continuación la cadena contiene uno o varios símbolos en el rango $[0 - 9]$
   - Posteriormente, y de forma opcional, aparece en la cadena el símbolo ".". Si aparece este símbolo, la cadena debe continuar con uno o más símbolos entre el rango $[0 - 9]$.
   - Opcionalmente la cadena puede ir seguida del símbolo "E" o "e" para indicar un número en notación científica. En este caso, tendrá que ir seguido de un símbolo "+" o "-" (opcional) y una cadena de uno o más símbolos entre el rango [0-9] que representan el exponente.
   - Algunas cadenas aceptadas por este autómata serían: 009, -78, -78.7, +78.7E-5, -7.876e+56, -78.87E56, etc.
   - Algunas cadenas no aceptadas por este autómata serían: .90, +78., +78.90E, etc.

4. Ejercicios: diseño de NFAs

Con el objetivo de profundizar en el diseño y la simulación de autómatas finitos no deterministas se proponen los siguientes ejercicios. La idea es que se realice, para cada ejercicio, un diseño de NFA en JFLAP y se realicen las simulaciones oportunas para comprobar su correcto funcionamiento, tal y como se hizo para el caso de los DFAs. Una vez diseñado el NFA correspondiente, se propone aplicar manualmente (con lápiz y papel) el algoritmo de construcción de subconjuntos y el algoritmo de minimización de estados para obtener el DFA mínimo equivalente. A continuación se utilizará JFLAP para obtener también el DFA mínimo equivalente y así poder verificar que tanto el proporcionado por JFLAP como el que el estudiante haya obtenido de forma manual sean equivalentes.

- Diseñar un autómata finito no determinista que reconozca cadenas sobre el alfabeto $\Sigma = \{a, b\}$ que tengan un número de a's múltiplo de tres o longitud par. A partir del NFA diseñado, obtenga un DFA mínimo equivalente.
- Diseñar un autómata finito no determinista que reconozca cadenas sobre el alfabeto $\Sigma = \{0, 1\}$ tales que contengan la subcadena `0110`. A partir del NFA diseñado, obtenga un DFA mínimo equivalente.

Referencias

[1] Transparencias del Tema 2 de la asignatura: Autómatas finitos y lenguajes regulares, https://campusingenieriaytecnologia2223.ull.es/mod/resource/view.php?id=5919

[2] JFLAP: Java Formal Language and Automata Package, https://www.jflap.org/

[3] Instalación de JFLAP, https://campusingenieriaytecnologia2223.ull.es/mod/page/view.php?id=5928

[4] Susan H. Rodger and Thomas W. Finley: *An Interactive Formal Languages and Automata Package*. Jones & Bartlett Publishers, Sudbury, MA, 2006. ISBN 0763738344. https://www.jflap.org/jflapbook/jflapbook2006.pdf

[5] Graphviz: open source graph visualization software, https://graphviz.org

[6] Ejemplo de DFA en JFLAP, https://www.youtube.com/watch?v=EzWxFMfqeFs

[7] Ejemplo de NFA en JFLAP, https://www.youtube.com/watch?v=audjcmz7ons