Universidad de Huelva

ALGORITMIA Y MODELOS DE COMPUTACIÓN

Práctica 1

Borja López Pineda y
Miguel Á. Sánchez

INDICE

Introducción:

En esta práctica desarrollaremos un simulador de autómatas finitos deterministas (AFD) y, autómatas finitos no deterministas (AFND). Explicaremos varios conceptos que creemos que son necesarios y que se requieren en la práctica, y veremos el uso de la aplicación, como introducir datos y como visualizar transiciones paso a paso quedando así de una forma más vistosa y clara para entender cualquiera de los autómatas mencionado anteriormente.

Definición formal del lenguaje de definición de autómatas

Características generales

La serialización de los datos del autómata puede ser guardada en ficheros de texto plano. Se utilizan las extensiones “afd” y “afnd” para autómatas finitos deterministas y no deterministas correspondientemente. Estas son las reglas de codificación en orden de aplicación.

- Las tabulaciones serán remplazadas por espacios.
- Todas las agrupaciones de dos o más espacios serán sustituidas por un solo espacio.
- Los espacios a principio o final de línea serán ignorados.
- Si una línea comienza por el carácter # será ignorada.
- Los estados pueden estar formados por múltiples caracteres y no pueden contener espacios.
- Los símbolos solo pueden ser un carácter y están acotados por comillas simples.
- Las listas de estados se separan por espacios.

Requisitos del AFD

Se deberán especificar una lista de estados, un estado inicial, una lista de estados finales y una lista de transiciones. Los estados, estado inicial y estados finales se declararán en una sola línea. Todos los estados en la lista de estados finales, las transiciones y el estado inicial deben aparecer en la lista de estados. Por ejemplo:

```
ESTADOS: A B C
INICIAL: A
FINALES: A B
```

```
ESTADOS: q1 q2
INICIAL: q2
FINALES: q2
```

Las transiciones se declararán en múltiples líneas. La primera línea identifica el comienzo de la lista y la última marca su fin. En una línea solo puede haber una transición. Las transiciones están formadas por el estado del que se parte, separado por un espacio el símbolo que provoca esa transformación y separado por un espacio el estado al que se llega. Por ejemplo:

```
TRANSICIONES:
        A '0' A
        A '1' B
        B '0' B
        B '1' A
FIN
```

```
TRANSICIONES:
        q1 '0' q1
        q1 '2' q2
        q2 '1' q1

FIN
```

Todos estos campos son obligatorios para que el AFD sea válido y pueda ejecutarse.

Requisitos del AFND

Adicionalmente a la lista de estados, estado inicial, lista de estados finales y transiciones; los AFND requieren una lista de transiciones lambda. Estas transformaciones se definen de forma muy similar a las transiciones que consumen símbolos salvo que carecen de símbolo, solo es necesario el estado de partida y una lista estados a los que se llega. Por ejemplo:

```
TRANSICIONES LAMBDA:
        A B C
        B D
FIN
```

```
TRANSICIONES LAMBDA:
        q1 q2

FIN
```

El estado inicial es una lista de estados iniciales en los AFND y en las transiciones que consumen símbolo el estado de destino también es una lista. Por ejemplo:

```
INICIAL: A   B
```

```
TRANSICIONES:
        A '0' A B
FIN
```

Las transiciones lambda no son un campo obligatorio para que el AFND sea correcto y las nuevas listas comentadas anteriormente pueden estar formada por un solo elemento. Es por esto por lo que todo AFD es un AFND válido que puede ejecutarse con las reglas del AFND y llegaría al mismo resultado que si se ejecutara en un AFD. Lo mismo no sucede al revés.

Implementación de los autómatas

Implementación del AFD

Los estados se representan con la clase String y los símbolos con el primitivo char.

Las listas de estados y estados finales se almacenan en un HashSet ya que el orden no es importante y no se permiten duplicados. Las transiciones se almacenan en un HashMap donde la clave se forma con el estado de partida y el símbolo, el valor es el estado al que se llega.

Antes de comenzar la ejecución se deben validar los datos del autómata. El conjunto de estados no puede estar vacío, el estado inicial debe pertenecer al conjunto de estados. El conjunto de estados finales no puede estar vacío y todos sus elementos deben pertenecer al conjunto de estados. En las transiciones los estados de partida y llegada deben pertenecer al conjunto de estados. También se realiza el seguimiento de los símbolos para poder diferenciar si una transición no existe o si el símbolo de la cadena es inválido.

Se realiza un proceso iterativo para determinar si la cadena es reconocida o no. El estado actual comienza siendo el estado actual. Por cada símbolo de la cadena se comprueba si el símbolo es válido, si no lo es se produce una excepción. Se busca una transición correspondiente al estado actual y al símbolo que se procesa. Si no es encontrada se produce una excepción. Si se encuentra la transición, el estado de llegada remplaza al estado actual.

Al terminar de procesar todos los símbolos se comprueba si el estado actual está contenido en el conjunto de estados finales. Si es así la cadena es reconocida.

Implementación del AFND

Los estados finales pasan a ser un conjunto. El valor del HashMap de transiciones es un array de String para representar los múltiples estados a los que se puede transitar. El valor del HashMap de transiciones en el caso de las Lambda Transiciones es solo el estado de partida. Cuando se deba descomponer en partes la transición el símbolo en una Lambda Transición tendrá el valor null.

El proceso de validación no sufre demasiadas variaciones con respecto al AFD salvo que el es conjunto de estados finales debe estar incluido en su totalidad en el de estados e igualmente con los estados a los que se llega en la transición.

El macroestado actual es un conjunto de estados igualado al estado inicial al principio. Se utilizará un conjunto de estados secundario para realizar las transiciones para evitar el problema de la modificación concurrente.

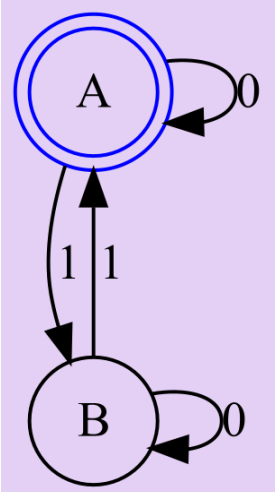
Por cada símbolo se calculará la Lambda Clausura y se aplicarán las transiciones asociadas al símbolo y a los estados del macroestado.

Para calcular la lambda clausura se agregan todos los estados a los que se llega sin consumir símbolo estando en cada uno de los estados del macroestado. Al introducir un estado nuevo en la lista de nuevos estados se examinan las lambda transiciones de ese nuevo estado de manera recursiva. Solo le agregan los estados que no estuvieran ya incluidos en el conjunto de nuevos estados para evitar bucles infinitos.

Una vez calculada la lambda clausura se calculan las transiciones que consumen símbolos. Se buscan todas las transiciones posibles para los estados del macroestado con el símbolo y se agregan los estados de llegada al conjunto de nuevos estados. Cuando se termina de buscar transiciones se vacía el macroestado y se agregan todos los nuevos estados. Por cada transición se debería haber eliminado el estado de partida, pero se produciría una modificación concurrente, es mejor no tocar el macroestado y limpiarlo después ya que si un estado no hace transición se elimina igualmente.

Ejemplos

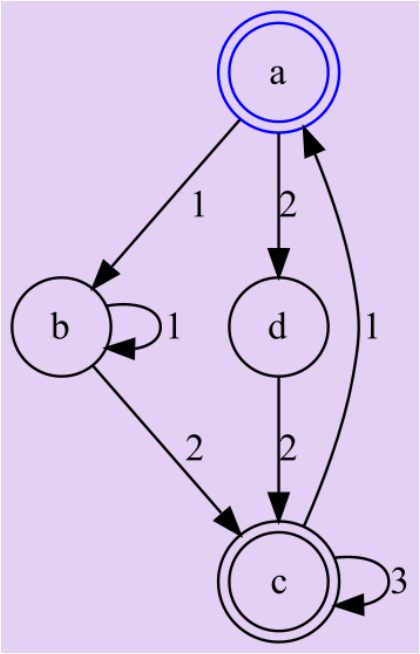
Ejemplos AFD



Test 1: El grafo más simple posible

ESTADOS: A B
INICIAL: A
FINALES: A
TRANSICIONES:
A '0' A
A '1' B
B '0' B
B '1' A
FIN

Cadena: 01101 -> No aceptada
Cadena 0110 -> Aceptada

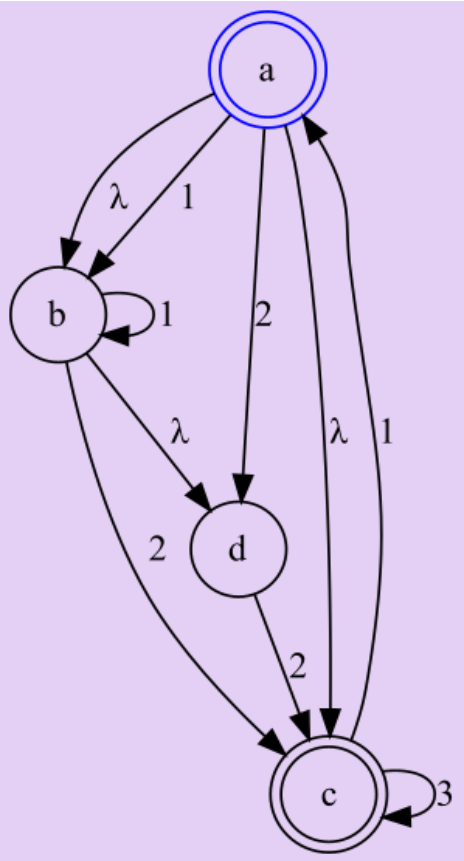


Test 2: El ejemplo visto en clase de practicas

ESTADOS: a b c d
INICIAL: a
FINALES: a c
TRANSICIONES:
a '1' b
a '2' d
b '1' b
b '2' c
c '1' a
c '3' c
d '2' c
FIN

Cadena: 112312211 -> No reconocida
Cadena 1123122 -> Reconocida
Cadena 122312 -> Error al ejecutar 122

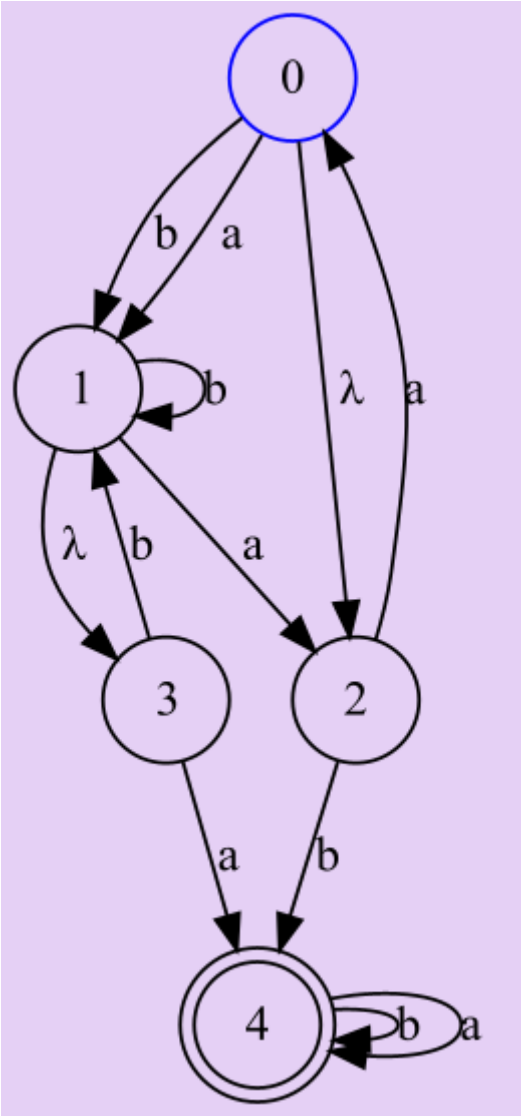
Ejemplos AFND



Test 3: El ejemplo visto en clase de practica modificado

ESTADOS: a b c d
INICIAL: a
FINALES: a c
TRANSICIONES:
a '1' b
a '2' d
b '1' b
b '2' c
c '1' a
c '3' c
d '2' c
FIN
TRANSICIONES LAMBDA:
a b c
b d
FIN

Cadena: 311 -> Reconocida
Cadena 1123 -> Reconocida
Cadena 112321 -> Error al ejecutar 11232



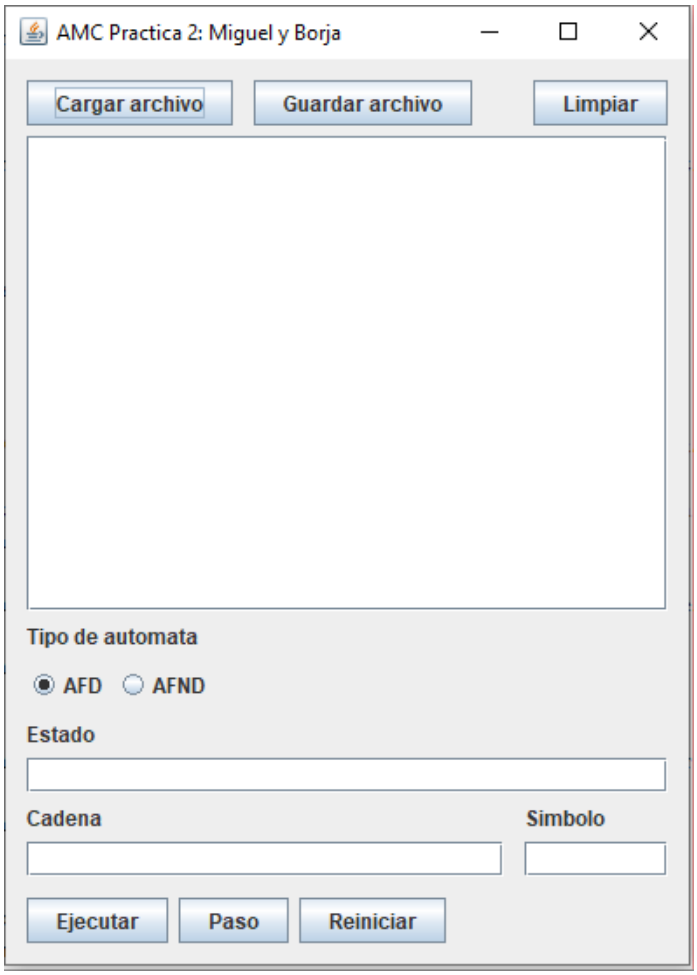
Test 4: Ejemplo de teoría

ESTADOS: 0 1 2 3 4
INICIAL: 0
FINALES: 4
TRANSICIONES:
0 'a' 1
0 'b' 1
1 'a' 2
1 'b' 1
2 'a' 0
2 'b' 4
3 'a' 4
3 'b' 1
4 'a' 4
4 'b' 4
FIN
TRANSICIONES LAMBDA:
0 2
1 3
FIN

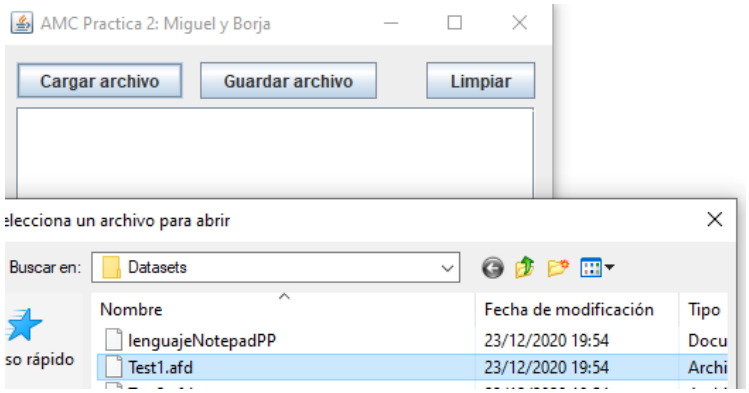
Cadena: aabbaba -> Reconocida
Cadena a -> No reconocida

Descripción y funcionamiento de la interfaz gráfica

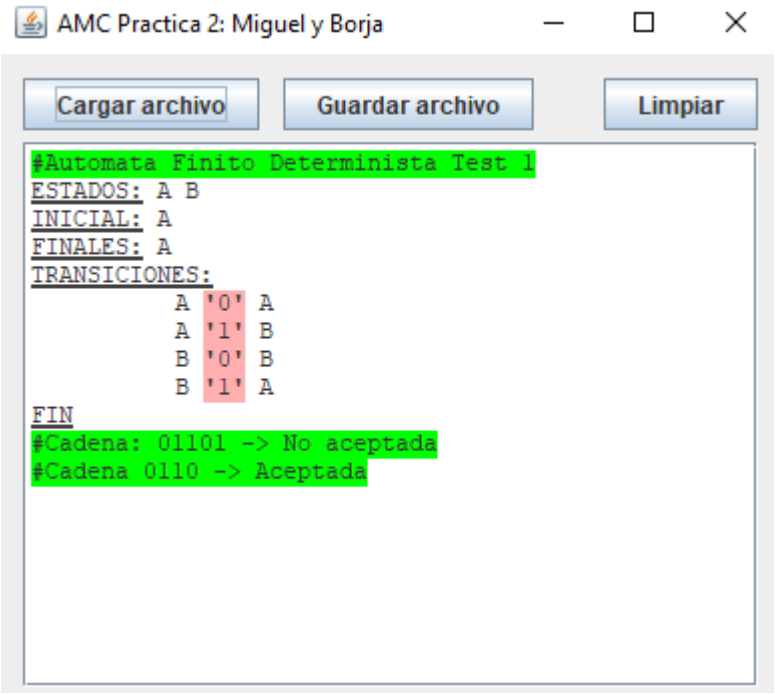
Al ejecutar la práctica nos encontramos con una ventana que tiene el siguiente aspecto:



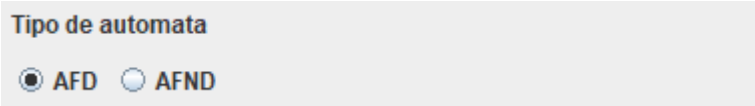
Como podemos observar, tenemos diferentes opciones. Las más sencillas en este caso son: cargar archivo, la cual nos permitirá simular un autómata desde un archivo tal y como vemos a continuación:



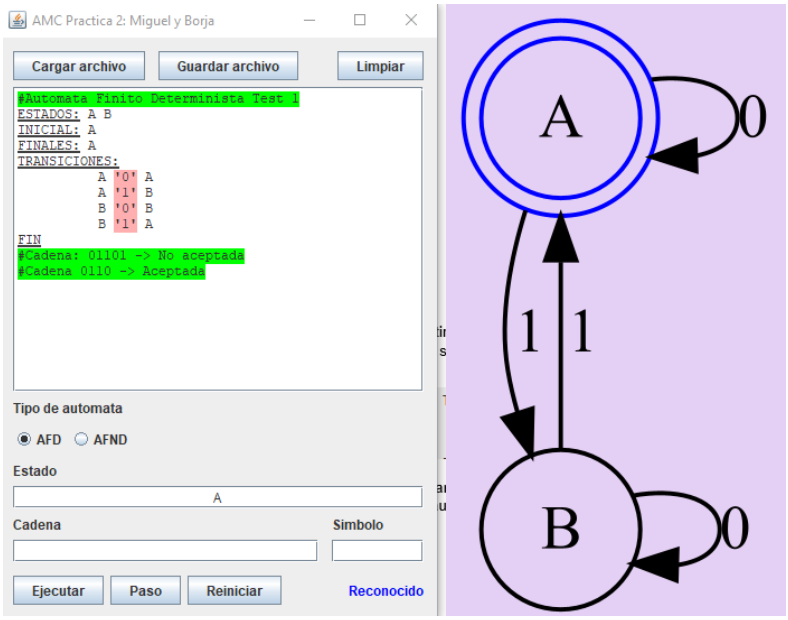
Al seleccionar el archivo deseado, nos aparecerán los estados del autómata, los estados finales, iniciales y las transiciones que se corresponden, junto con las cadenas que no son aceptadas.



A continuación, se seleccionará automáticamente el tipo de autómata en función de la extensión del archivo (En este caso se trata de un AFD):



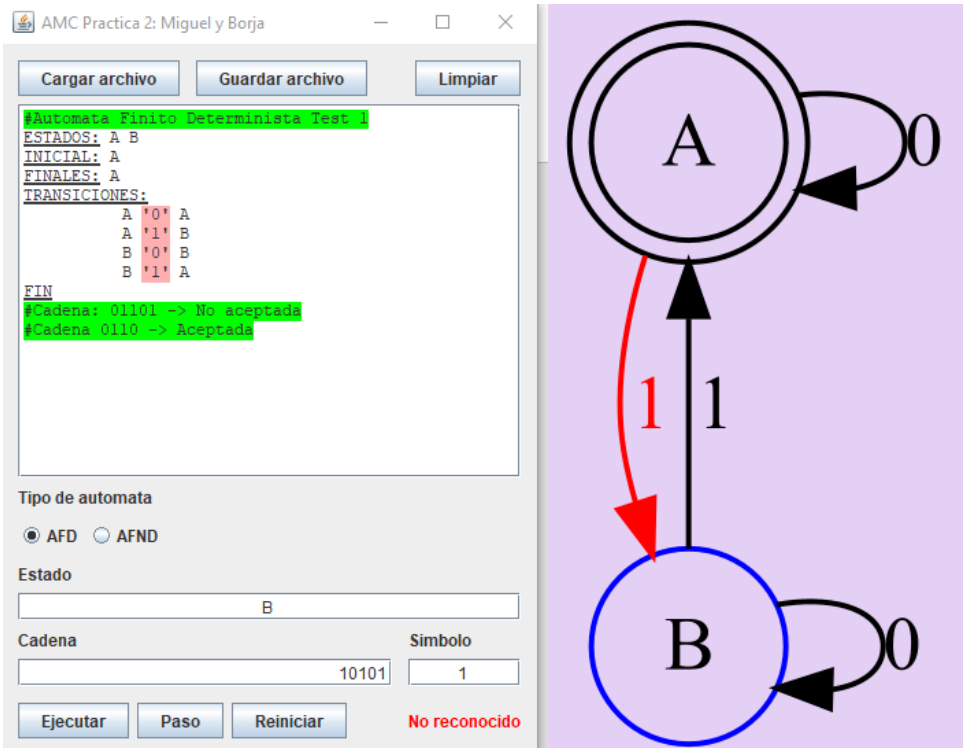
Y pasamos a darle a ejecutar, lo cual el programa nos dibujará el esquema que pertenece al autómata definido en el archivo:



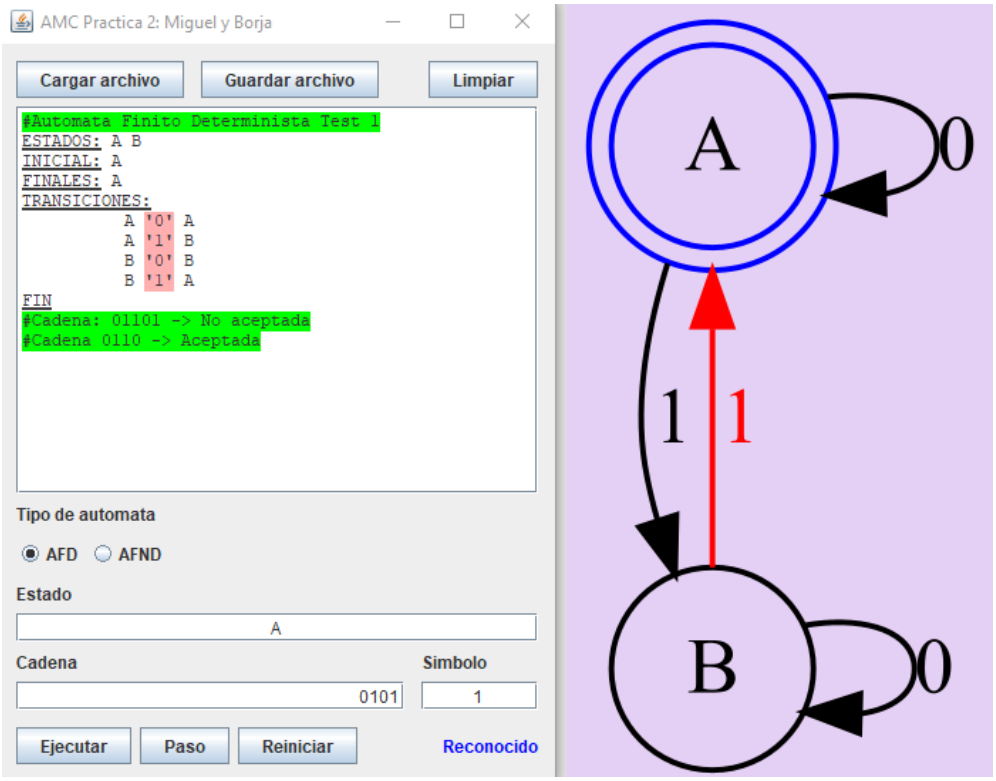
Cuando el estado tiene dos círculos, significa que es un estado final, y, el color azul se corresponde con el estado actual.

Pasemos ahora a introducir una cadena para ver las transiciones en función de la cadena que vamos a introducir a continuación, que será la cadena “110101”, por tanto, solo hay que introducir la cadena deseada en su correspondiente apartado:

Si le damos a ejecutar, no veremos las transiciones, pero si veremos cuál es el estado en el que termina esa cadena que hemos introducido, por tanto, vamos a darle a PASO, que es el botón correspondiente de ejecutar la cadena paso por paso:

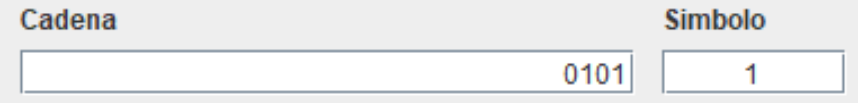


Vemos como se consume en el apartado cadena un carácter y a su vez se muestra en rojo la transicion que se ha producido, quedando la cadena restante aun visible e indicando en la esquina inferior derecha que el estado al que hemos procedido a ir no es un estado final reconocido, tal y como aparece en el autómata. Si seguimos dandole a paso, seguira llevandonos al estado que pertenece en funcion de la transición definida en el archivo:



Ahora en este caso, vemos como al representar el siguiente estado, en la esquina inferior derecha, si que nos aparece que el estado actual en el que estamos si es un estado reconocido, por lo que esto quiere decir que, ese estado, es un estado final.

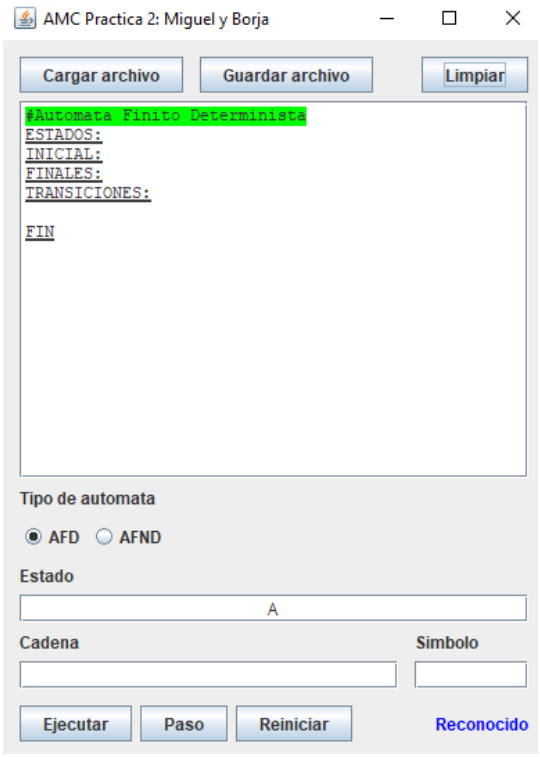
También podemos observar en la interfaz, que, al darle a paso, en el apartado Símbolo nos indica el carácter de la cadena que se ha consumido y que a su vez se dibuja en rojo en la imagen:



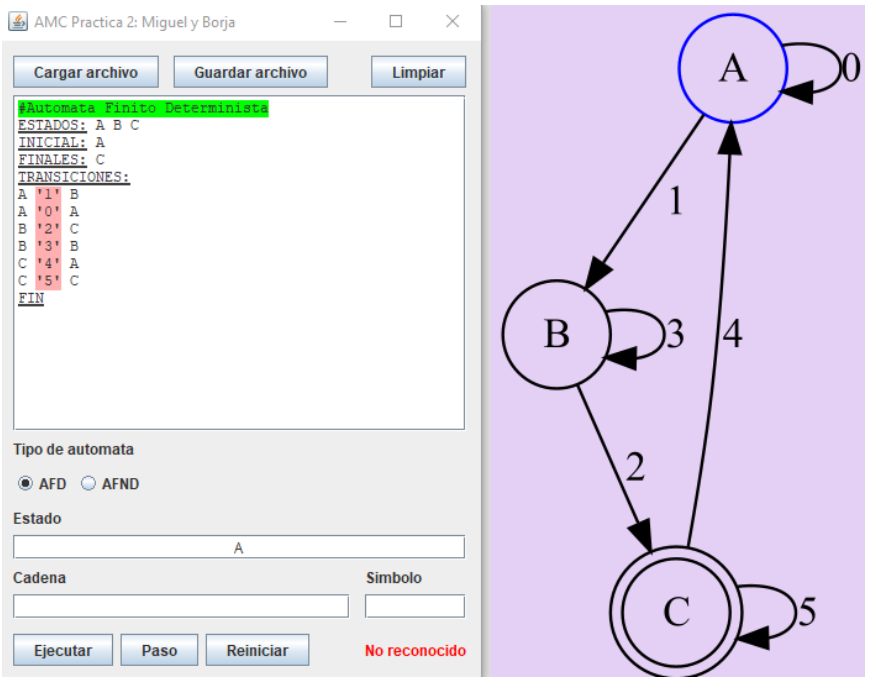
Finalmente, obtendremos el resultado de la cadena y se quedara en el ultimo estado que hayamos introducido en la cadena dibujado en color azul.

Si deseamos probar otra cadena con el mismo autómata, solo tendremos que pulsar el botón Reiniciar, y, pasaremos a introducir la nueva cadena deseada.

Ahora vamos a crear un autómata propio, desde cero, por tanto, si pulsamos el botón limpiar, nos escribirá automáticamente el formato necesario para poder ejecutar este:



Pondremos un AFD con tres estados (A, B y C), el estado A como inicial y el estado C como final y sus transiciones correspondientes quedando así el autómata:



Por tanto, una vez definido, el funcionamiento es exactamente el mismo que el explicado anteriormente, con la única diferencia de que ahora, podemos guardar el archivo con la estructura correspondiente para así, en otro momento volver a cargarlo y simularlo.

Guía de instalación

El Proyecto requiere la librería graphviz-java y todas sus dependencias. Aunque es posible instalarla mediante Maven, recomendamos incluir los archivos jar manualmente.

1. Descargar el archivo comprimido con todas las librerías.

2. Descomprimir la carpeta graphviz entera. Es aconsejable guardarla junto a Netbeans. EJEMPLO: C:\Program Files\NetBeans-11.1\Libraries\graphviz

3. Agregar la librería a Netbeans.
   3.1. Tools > Libraries > New Library
   3.2. Debe llamarse graphviz
   3.3. Add JAR/Folder
   3.4. Seleccionar todos los ficheros de la carpeta graphviz descomprimida
   3.5. Ok

4. Importar el proyecto
   4.1. File > Import Proyect > From Zip
   4.2. Seleccionar el fichero zip con el proyecto entregado
   4.3. Import
   4.4. Es posible que Netbeans alerte de que no se encuentra el JDK. El proyecto está configurado con OpenJDK 11 pero funcionará con cualquier JDK igual o superior al 11. El editor permite solucionar este problema al momento de importar el proyecto.

NOTA: Aunque es posible instalarlo en Eclipse, se recomienda encarecidamente usar el IDE de Netbeans.