ALGORÍTMICA Y MODELOS DE COMPUTACIÓN

Práctica 2: AFD y AFND

Alfonso J. Redondo Díaz

Código Fuente

Clase AFD: Describe un autómata finito determinista genérico mediante un estado inicial, una lista de transiciones y un conjunto de estados finales. Los estados se representan mediante números enteros, los símbolos mediante caracteres y las transiciones mediante tuplas de 3 elementos: (estado_origen, símbolo, estado_destino).

En ella, incorporo los siguientes métodos:

- Public AFD(): Constructor por defecto, que inicializa el autómata sin reglas ni estados.
- Public AFD(AFD): Constructor de copia. Crea un autómata a partir de otro.
- Public void agregarTransición(String, char, String): Agrega una transición al automata. Cuenta con 3 variables que recibe por parámetro:
    1. estadoOrigen, Estado inical del autómata en la transición.
    2. Símbolo, Simbolo de la transición.
    3. estadoDestino, Estado final del autómata tras la transición.

- Public String transicion(String,char): Busca la transición del autómata para el estado y el símbolo dados. Cuenta con 2 variables que recibe por parámetro:
    1. Estado, Estado actual del autómata.
    2. Símbolo, Símbolo de transición.

  Y devuelve el estado del autómata tras la transición o -1 si no existe la misma

- Public boolean esFinal(String): Indica si el autómata tiene más transiciones para ejecutarse.

- Public boolean reconocer(String): Devuelve true si la cadena introducida concuerda con las transiciones del autómata.

- Protected Object clone(): Devuelve una copia del autómata

- Public String toString(): Transforma el automata en String con formato:

  transiciones:
  (origen1, simbolo1) -> destino1
  (origen2, simbolo2) -> destino2
  ...
  estadosFinales:
  estado1
  estado2

  y lo devuelve.

- Public static AFD contenido(String): Crea un objeto AFD a partir de la información almacenada en un archivo con el formato que he establecido.

Clase TransiciónAFD: Aquí he agrupado todas las transiciones posibles del autómata (transición normal/transición lambda).

En ella, incorporo los siguientes métodos:

- Public TransicionAFD(String,char,String): Constructor que crea una transición con los atributos pasados por parámetro.

- Public TransicionAFD(TransicionAFD): Constructor que crea una transición copiando las del autómata pasado por parámetro. Ambos reciben los siguientes valores:

  1. estadoOrigen Estado inicial de la transición
  2. simbolo Simbolo de la transición
  3. estadoDestino Estado final de la transición

- Public String getDestino(): Devuelve el estado final de la transición.

- Public String getSimbolo(): Devuelve el símbolo de la transición.

- Public boolean equals(Object): Compara si dos objetos son iguales, devolviendo true si lo son y false en caso contrario.

- Public Object clone(): Devuelve una copia de la transición del autómata.

- Public String toStringLong(): Devuelve la transición en formato origen: origen, simbolo: simbolo, destino: destino y devuelve un String con el contenido del objeto.

- Public String toString (): Devuelve la transición en formato (origen, simbolo) -> destino y devuelve un String con el contenido del objeto.

Clase AFND: Describe un autómata finito no determinista genérico mediante un estado inicial, una lista de transiciones y transiciones-λ y un conjunto de estados finales.

Los estados se representan mediante números enteros, los símbolos mediante caracteres, las transiciones mediante tuplas de 3 elementos:

(estado_origen, símbolo, estados_destinos[] y las λ transiciones mediante tuplas de 2 elementos: (estado_origen, estados_destinos[]).

En ella, incorporo los siguientes métodos:

- Public AFND(): Constructor por defecto, crea un autómata vacío.
- Public AFND(AFND): Constructor de copia, agrega al autómata una transición:
    1. estadoOrigen: Estado origen de la transición.
    2. Símbolo: Símbolo de la transición
    3. estadosFinales Estados finales a los que puede ir el autómata desde el estado origen utilizando el símbolo de esta transición.
- Public agregarTransicion(String,char,ArrayList<>): Realiza una transición (si existe) para un estdo y un simbolo dados.
- Public ArrayList<TransicionAFND> getTransiciones(): Devuelve el número de transiciones que realiza el autómata.

- Public ArrayList<TransicionLambda> getTransicionesL(): Devuelve el número de transiciones lambda que realiza el autómata.

- Public ArrayList<String> getEstadosFinales(): Devuelve el número de estados finales que tiene el autómata.

- Public void agregarTransicionLambda(String,ArrayList<>): Agrega una nueva transición lambda al autómata.

- Private ArrayList<String> transicion(String,char): Realiza todas las transiones posibles a partir de un macroestado y un simbolo.

- Private ArrayList<String> transicionLambda(String,char): Realiza todas las transiones posibles a partir de un estado y devuelve el conjunto de estados a los que el autómata puede ir desde el estado inicial mediante únicamente transiciones lambda.

- Public boolean esFinal(String): Comprueba si algun estado de un macroestado es final, y por tanto lo es el macroestado. Devuelve true si hay al menos un estado final en el macroestado, si no, devuelve false.

- Public ArrayList<String>lambda_clausura(String): Comprueba si algun estado de un macroestado es final, y por tanto lo es el macroestado.

- Public ArrayList<String> lambda_clausura(ArrayList<String>): Calcula la lambda-clausura de un macroestado.

- Public ArrayList<String> lambda_clausura(ArrayList<String>): Método auxiliar para facilitar el cálculo de una lambda-clausura.

- Public boolean reconocer(String): Devuelve true si la cadena introducida concuerda con las transiciones del autómata.

- Protected Object clone(): Devuelve una copia del autómata

- Public static AFD contenido(String): Crea un objeto AFND a partir de la inforación almacenada en un archivo con formato:

  origen;simbolo;destino1,destino2,destino3,...
  origen;simbolo;destino1,destino2,destino3,...
  ...
  #!
  origen;destino1,destino2,destino3,...
  #!!
  estadoFinal

- Public String toString(): Transforma el automata en String con formato:

  transiciones:
  (origen1, simbolo1) -> destino1
  (origen2, simbolo2) -> destino2
  ...

  transicionesLambda:

  (origen1, λ) -> destino1 destino2
  (origen1, λ) -> destino1 destino2

  estadosFinales:
  estado1
  estado2

Clase TransiciónAFND:

Aquí he agrupado las transiciones normales del autómata.

En ella, incorporo los siguientes métodos:

- Public TransicionAFND(String,char,String): Constructor que crea una transición con los atributos pasados por parámetro.

- Public TransicionAFND(TransicionAFND): Constructor que crea una transición copiando las del autómata pasado por parámetro. Ambos reciben los siguientes valores:

    1. estadoOrigen Estado inicial de la transición
    2. simbolo Simbolo de la transición
    3. estadoDestino Conjunto de estados destino a los que el autómata puede ir desde el estado inicial con el mismo simbolo.

- Public String getEstadoOrigen(): Devuelve el estado origen de la transición.

- Public String getSimbolo(): Devuelve el símbolo de la transición.

- Public ArrayList<String> getEstadosDestino(): Devuelve el conjunto de estados destino de la transición.

- Public boolean equals(Object): Compara dos objetos {@this} atendiendo a si representan el conjunto de transiciones para un mismo estado inicial y simbolo. Si estadoInicial y simbolo coinciden, devolverá true aunque el conjuntos de estados destino sea diferente.

- Public Object clone(): Devuelve una copia de la transición del autómata.

- Public String toString (): Devuelve la transición en formato (origen, simbolo) -> destino y devuelve un String con el contenido del objeto.

Clase TransiciónLambda: Aquí he agrupado todas las transiciones lambda posibles del autómata.

En ella, incorporo los siguientes métodos:

- Public TransicionLambda(String,char,String): Constructor que crea una transición con los atributos pasados por parámetro.

- Public TransicionLambda (TransicionLambda): Constructor que crea una transición copiando las del autómata pasado por parámetro. Ambos reciben los siguientes valores:

    1. estadoOrigen Estado inicial de la transición
    2. estadoDestino Conjunto de estados destino a los que el autómata puede ir desde el estado inicial mediante una transición lambda.

- Public String getEstadoOrigen(): Devuelve el estado origen de la transición.

- Public ArrayList<String> getEstadosDestino(): Devuelve el conjunto de estados destino de la transición.

- Public boolean equals(Object): Compara dos objetos {@this} atendiendo a si representan el conjunto de transiciones para un mismo estado inicial y simbolo. Si estadoInicial y simbolo coinciden, devolverá true aunque el conjuntos de estados destino sea diferente.

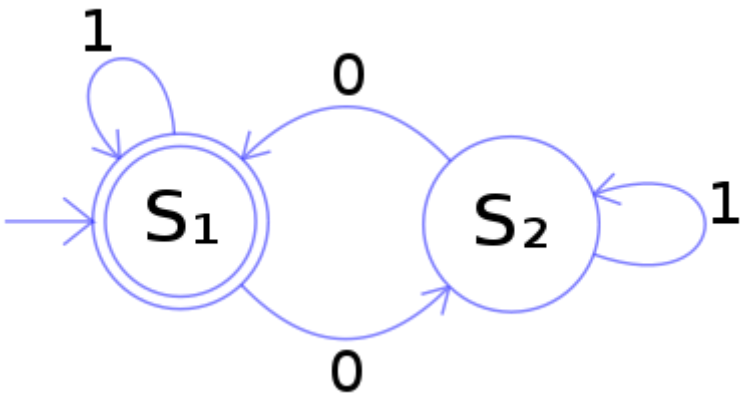
- Public Object clone(): Devuelve una copia de la transición del autómata.

- Public String toString (): Devuelve la transición en formato (origen, simbolo) -> destino y devuelve un String con el contenido del objeto.

Realización de la práctica.

Planteamiento

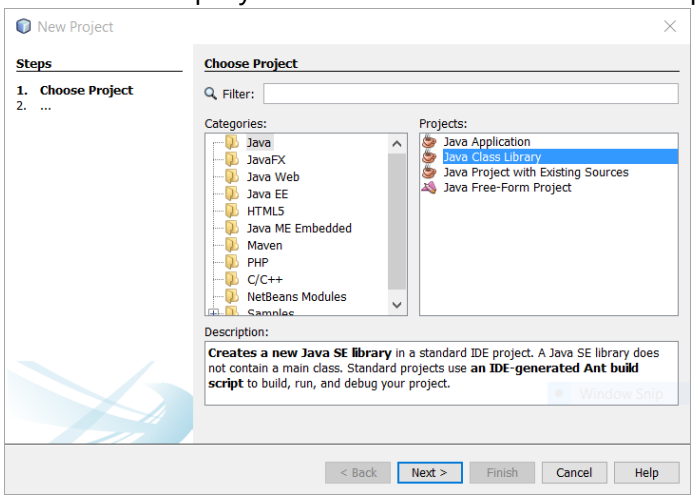
Al ser una materia que no había impartido antes, la primera fase de esta práctica consistió en reunir información acerca de los autómatas y su funcionamiento, a través de los temas proporcionados en moodle, apuntes cogidos en clase, explicaciones de vídeos en distintas plataformas, etc.



Creación del proyecto

Una vez realizado lo anterior, me dispuse a crear el proyecto de la práctica con la herramienta que hará posible su implementacion (Netbeans).

Dividiendo el proyecto en cada una de las fases que vienen explicadas en la práctica, y organizándolas de manera fácil e intuitiva para que a la hora de empezar a programar esté todo lo más completo y entendible posible.



Realización de la práctica

Lógicamente la parte más complicada del proceso, en ella, tengo que volver a recurrir a información adicional para poder entender bien la complejidad del programa que voy a abordar.

Me surgen problemas como no saber usar bien las matrices hash, ya que después de buscar la información, parecían un procedimiento bastante útil para realizar toda la práctica. En su lugar, debido a que los días pasaban y su funcionamiento no es nada intuitivo, decidí usar en vez de esa estructura de datos, un ArrayList<> que recogiese toda la información que necesitaba.

Para implementar el método de lectura desde un archivo también tuve que buscar bastante información, pero gracias a un vídeo y un compañero pude hacerlo correctamente y la lectura se hace bien. No tuve tanta suerte a la hora de implantar el formato para que leyese desde el archivo(bloc de notas), ya que inicialmente el formato pedido es:

```
ESTADOS: q0 q1 q2
INICIAL: q0
FINALES: q2
TRANSICIONES:
 q0  '0'  q1
 q0  '1'  q2
 q2  '0'  q1
FIN
```

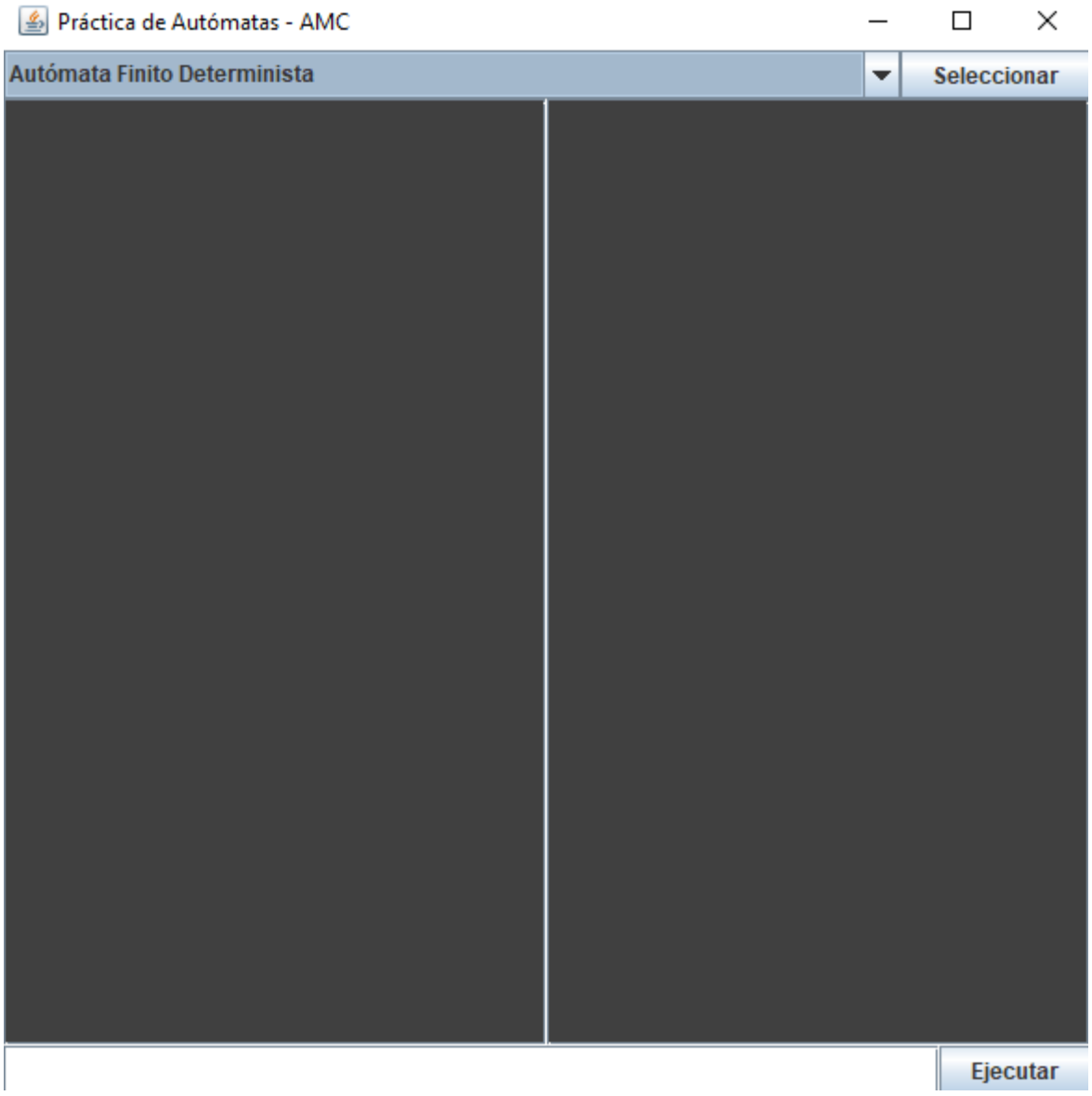
Tuve que adaptarlo a otro diferente para que las transiciones pudiesen leerse para al menos poder continuar con la ejecución.

Podría decir que esos han sido los dos grandes problemas que se me han planteado en la práctica, más allá de la ausencia de conocimientos necesarios inicialmente y la necesidad de estar varias semanas leyendo información e intentando transmitirla en el código.

Prueba de funcionamiento

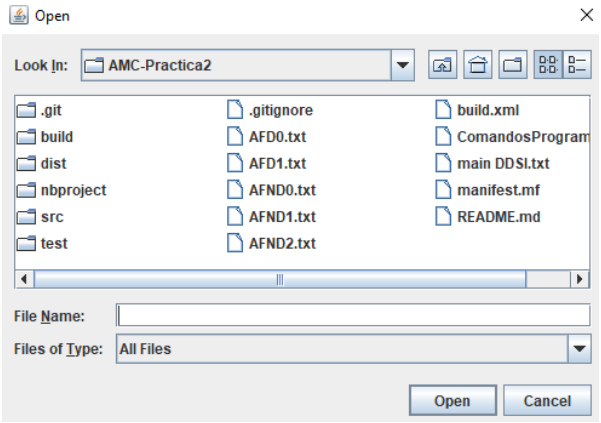
A continuación, procedo a mostrar los ejemplos en casos favorables y no, para los dos tipos de autómatas:

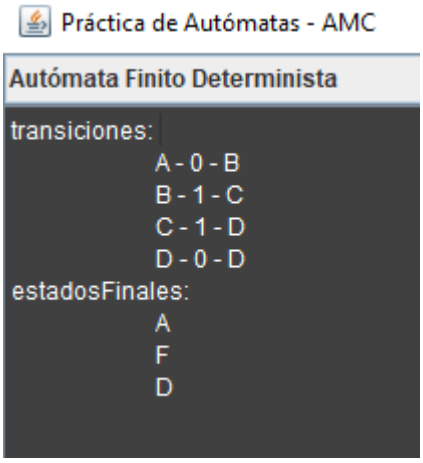
Interfaz Principal:



Ejemplificación para AFD:

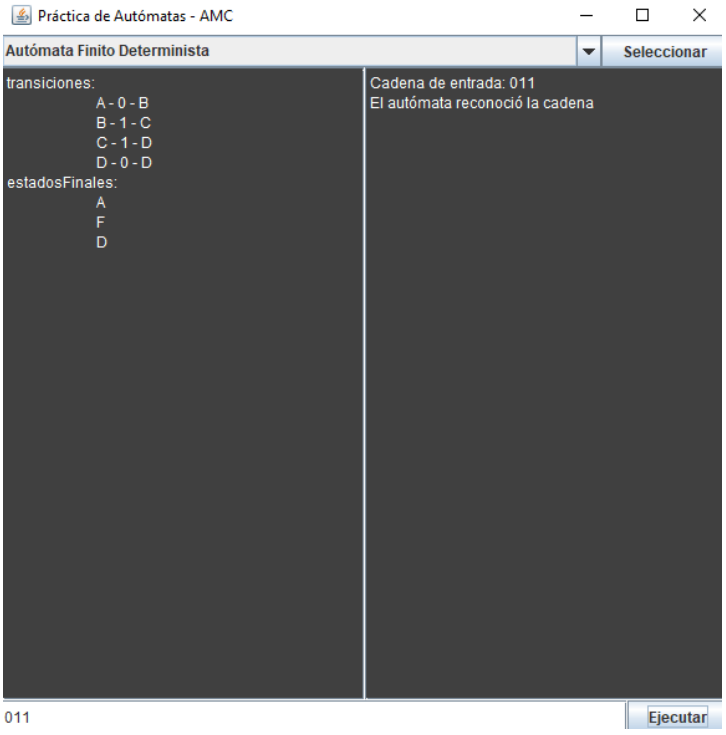
Elección del archivo que vamos a tomar como ejemplo (AFD0.txt)





Observamos las transiciones:

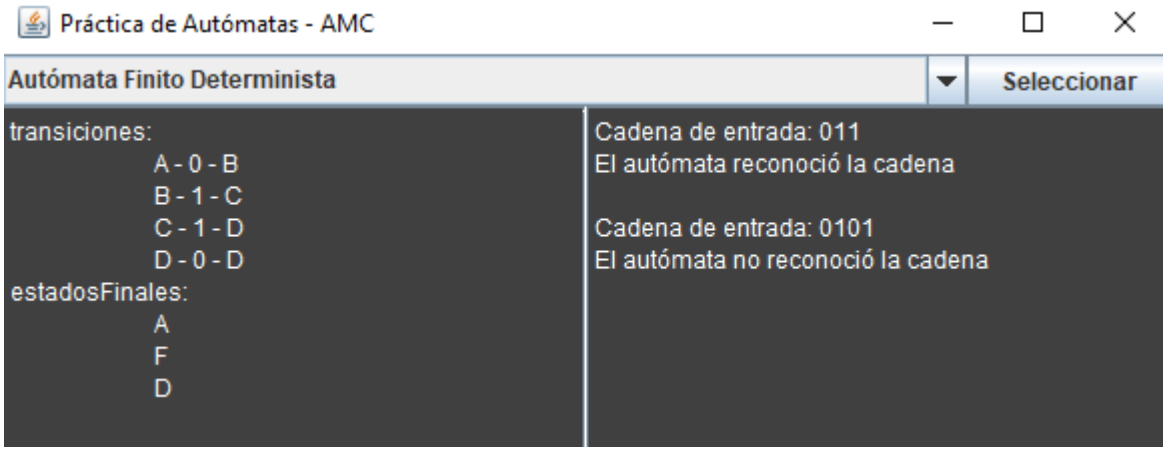
Observamos el estado inicial (A) y los estados finales (A,F,D). Por lo tanto, vamos a comprobar ahora si intruduciéndole una cadena que desemboque en uno de esos dos estados finales, el autómata los detecta o no.



Introduciendo la cadena: 011

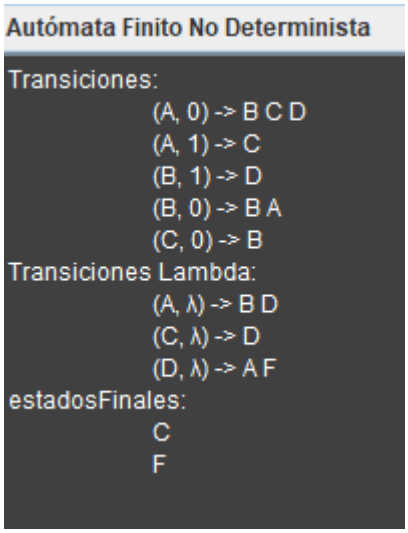
Como podemos observar, el aútomata reconoce correctamente la secuencia, ya que si la analizamos con detenimiento, el estado inicial (A), si recibe un 0 va hacia B, si esta recibe un 1, se trasladaría hacia C y finalmente con un 1 acabaríamos en el estado final D.

En caso de introducir una combinación que no sea válida, el sistema devolverá este mensaje:



Ejemplificación para AFND:

Elección del archivo que vamos a tomar como ejemplo (AFND2.txt)



Observamos las transiciones:

Observamos el estado inicial (A) y los estados finales (C,F). Por lo tanto, vamos a comprobar ahora si intruduciéndole una cadena que desemboque en uno de esos dos estados finales, el autómata los detecta o no.

Introduciendo la cadena: 001

Como podemos observar, el aútomata reconoce correctamente la secuencia, ya que si la analizamos con detenimiento, el estado inicial (A), si recibe un 0 va hacia B, si esta recibe un 1, se trasladaría hacia C y finalmente con un 1 acabaríamos en el estado final D.