Universidad de Huelva

ALGORITMICA Y MODELOS DE COMPUTACION

AFD Y AFND

ESTUDIANTE: SAUL RODRIGUEZ NARANJO

PROFESOR: ANTONIO ANGEL MARQUEZ

ETSI
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
uhu.es

AUTOMATA AFD

Para implementar el Autómata AFD hemos considerado tres parámetros principales, un array de String en representación de los Estados Finales, un String que hara de Estado Inicial y un ArrayList de TransicionAFD que contendrá todas las transiciones posibles de nuestro autómata.

```
public class AFD implements Cloneable, Proceso {

    private String[] estadosFinales;
    private String estadoInicial;
    private ArrayList<TransicionAFD> transiciones;
```

Mas allá de los Getters y los Setters hemos implementado varios métodos clave como son leeFichero(), reconocer() y transicion()

```
public String transicion(String estado, char simbolo) {
    boolean encontrado = false;
    int i = 0;
    while (i < this.transiciones.size() && !encontrado) {
        if (this.transiciones.get(i).getEstadoOrigen().equals(estado)
                && this.transiciones.get(i).getSimbolo() == simbolo) {
            encontrado = true;
        } else {
            i++;
        }
    }
    if (encontrado) {
        return this.transiciones.get(i).getEstadoFinal();
    } else {
        return "M";
    }

}
```

```java
public boolean reconocer(String cadena) {
    char[] simbolo = cadena.toCharArray();
    String estado = this.estadoInicial;
    for (int i = 0; i < simbolo.length; i++) {
        estado = transicion(estado, simbolo[i]);
        System.out.println("Estado: " + estado);
    }
    return esFinal(estado);
}
```

```java
public static void leeFichero(String ruta, AFD automata) {
    FileReader entrada = null;
    try {
        File archivo = new File(ruta);
        entrada = new FileReader(archivo);

        BufferedReader buffer = new BufferedReader(entrada);
        String linea = "";
        String split_linea[] = null;

        //Leemos las dos primeras lineas, "ESTADOS:" e "INICIAL:"
        //que no nos interesan
        for (int i = 0; i < 2; i++) {
            linea = buffer.readLine();
            if (i == 1) {
                String[] split = linea.split(" ");
                automata.setEstadoInicial(split[1]);
            }
        }
```

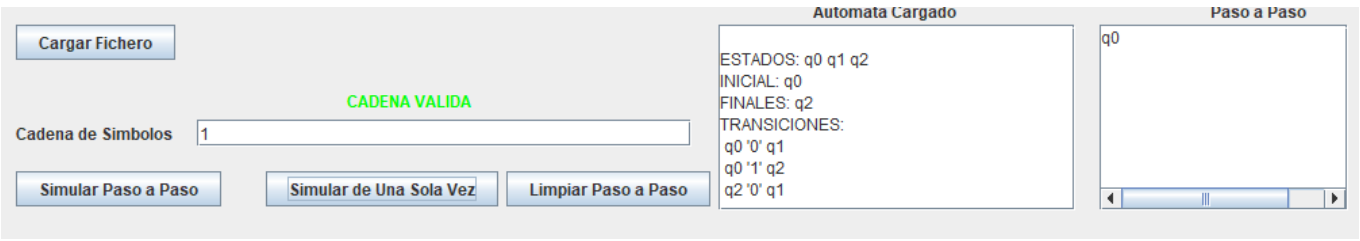
```
//Leemos los estados finales y los añadimos a nuestro automata
linea = buffer.readLine();
split_linea = linea.split(" ");
String[] estadosFinales = new String[split_linea.length - 1];
for (int i = 0; i < estadosFinales.length; i++) {
    estadosFinales[i] = split_linea[i + 1];
}
automata.setEstadosFinales(estadosFinales);
//SE LEERA LA CADENA "TRANSICIONES:" y se desecha
linea = buffer.readLine();

//Comenzamos a leer las transiciones e insertarlas en el automata
while (!linea.equals("FIN")) {
    linea = buffer.readLine();
    if (!linea.equals("FIN")) {
        split_linea = linea.split(" ");

        String simbolo_array[] = split_linea[2].split("'");
        char simbolo = simbolo_array[1].toCharArray()[0];

        automata.agregarTransicion(split_linea[1], simbolo, split_linea[3]);
    }

}

entrada.close();
```

```
    } catch (FileNotFoundException ex) {
        Logger.getLogger(AFD.class.getName()).log(Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(AFD.class.getName()).log(Level.SEVERE, null, ex);
    } finally {
        try {
            entrada.close();
        } catch (IOException ex) {
            Logger.getLogger(AFD.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

leeFichero() acepta Autómatas AFD con el siguiente formato:

```
ESTADOS: q0 q1 q2 q3
INICIAL: q0
FINALES: q2 q3
TRANSICIONES:
 q0 '0' q1
 q0 '1' q3
 q1 '0' q0
 q1 '1' q2
 q3 '0' q2
 q3 '1' q1
FIN
```

Y es capaz de insertarlo al programa adaptando los simbolos y estados. Como símbolos, nuestro Automata AFD aceptara cualquier tipo de char (en este caso 0 o 1) y se constará con 1 o varios estados finales, a los cuales se llegará mediante una cadena de símbolos a partir del estado de origen.



El funcionamiento es el siguiente, por cada símbolo se avanzará un estado a partir de otro (primero el de origen) hasta que se terminen los símbolos de la cadena, tras comprobar la ultima transición se indicara si la cadena es valida o no, en función de si el último estado es un Estado Final.

La implementación de la interfaz ha sido bastante sencilla, se ha optado por JFrame al que se le han aderido 2 JPanels, uno para el Autómata AFD y otro para el AFND.

AUTOMATA AFND

La implementación del Autómata AFND es prácticamente idéntica a la del AFD, solo que en esta ocasión contamos con la peculiaridad de las transiciones lambda y los macroestados.

```
public class AFND implements Proceso, Cloneable {

    private String[] estadosFinales;
    private ArrayList<TransicionAFND> transiciones;
    private ArrayList<TransicionLambda> transicionesLambda;
    private String estadoInicial;
```

La funcion para las transiciones completas de un Autómata AFND es la siguiente:

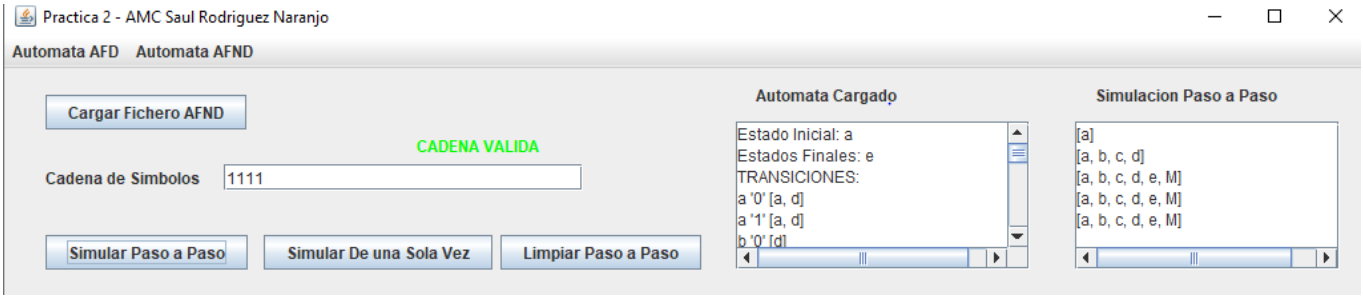
```java
/**
 * Devuelve una transicion completa del Automata AFND
 *
 * @param estado Estado de Origen
 * @param simbolo Simbolo para la transicion
 * @return La transicion completa
 */
public ArrayList<String> transicion(ArrayList<String> estado, char simbolo) {

    ArrayList<String> AFNDTransitions = this.transicionAFND(estado, simbolo);

    return this.lambdaClausura(AFNDTransitions);

}
```

Partimos de un conjunto de estados iniciales, a los cuales les hacemos las transiciones AFND normales, y posteriormente se realiza la lambda clausura de ese resultado, que no es mas que buscar todos los estados a los que se puede llegar con lambda a partir de un determinado estado. Todo ello se combina en un ArrayList final y se devuelve como la transicion completa de este Autómata.

```java
/**
 * Devuelve el conjunto de transiciones lambda de un macroestado
 *
 * @param estado Macroestado
 * @return El conjunto de transiciones lambda
 */
public ArrayList<String> transicionLambda(ArrayList<String> estados) {
    ArrayList<String> resultado = new ArrayList<>();

    for (int i = 0; i < estados.size(); i++) {
        if (!resultado.contains(estados.get(i))) {
            resultado.add(estados.get(i));
            this.lambdaRecursivo(resultado, estados.get(i));
        }
    }

    return resultado;
}
```

```java
/**
 * De forma recursiva busca todas las pòsibles transiciones lambda a partir
 * de un estado y las deposita en un array
 *
 * @param lambda Array de estados de destino lambda
 * @param estadoActual Estado lambda actual
 */
public void lambdaRecursivo(ArrayList<String> lambda, String estadoActual) {
    ArrayList<String> aux = this.transicionLambdaIndividual(estadoActual);
    //System.out.println("length: " + aux.size());
    if (aux.size() == 2) {
        if (!lambda.contains(aux.get(1))) {
            lambda.add(aux.get(1));
            this.lambdaRecursivo(lambda, aux.get(1));
        }
    }

}
```

En cuanto a la interfaz de usuario es practicamente idéntica a la de los Autómatas AFD:



El formato de los ficheros es muy similar, con la peculiaridad de que los estados lambda se representan con una 'L' y además esta vez de detalla el estado muerto M:

```
ESTADOS: a b c d e M
INICIAL: a
FINALES: e
TRANSICIONES:
 a '0' a d
 a '1' a d
 a 'L' a b
 b '0' d
 b '1' b
 b 'L' b c
 c '0' M
 c '1' e
 c 'L' c d
 d '0' e
 d '1' M
 d 'L' d
 e '0' M
 e '1' M
 e 'L' e
 M '0' M
 M '1' M
 M 'L' M
FIN
```

TRANSICIONES AFD/AFND Y LAMBDA

Para los distintos tipos de autómatas se han detallado sus respectivas transiciones:

```java
public class TransicionAFD {

    private String estadoOrigen;
    private char simbolo;
    private String estadoFinal;

    /**
     * Constructor de TransicionAFD, dados un Estado de Origen, un Simbolo y un
     * Estado Final construye la transicion.
     * @param estadoOrigen Estado de Origen
     * @param simbolo Simbolo para la transicion
     * @param estadoFinal Estado de Destino
     */
    public TransicionAFD(String estadoOrigen, char simbolo, String estadoFinal) {
        this.estadoOrigen = estadoOrigen;
        this.simbolo = simbolo;
        this.estadoFinal = estadoFinal;
    }
```

```java
public class TransicionAFND {
    private String estadoOrigen;
    private char simbolo;
    private ArrayList<String> estadosFinales;

    /**
     * Constructor de TransicionAFND, dados un Estado de Origen, un Simbolo y un
     * ArrayList de Estados Finales construye la transicion.
     * @param estadoOrigen Estado de Origen
     * @param simbolo Simbolo para la transicion
     * @param estadosFinales Conjunto de Estados Finales
     */
    public TransicionAFND(String estadoOrigen, char simbolo, ArrayList<String> estadosFinales) {
        this.estadoOrigen = estadoOrigen;
        this.simbolo = simbolo;
        this.estadosFinales = estadosFinales;
    }
```

```java
public class TransicionLambda {
    private String estadoOrigen;
    private ArrayList<String> estadosFinales;

    /**
     * Constructor de la clase TransicionLambda
     * @param estadoOrigen Estado de Origen de la Transicion Lambda
     * @param estadosFinales Estados Finales de la Transicion Lambda
     */
    public TransicionLambda(String estadoOrigen, ArrayList<String> estadosFinales) {
        this.estadoOrigen = estadoOrigen;
        this.estadosFinales = estadosFinales;
    }
```