

Práctica 2 AMC

Autómatas Finitos Deterministas y

Autómatas Finitos No Deterministas



Universidad
de Huelva

Estudiante: Juan Diego Díaz Domínguez

Fecha: 15/01/2022

Índice

Introducción.....	3
Autómata Determinista.....	3
Autómata No Determinista.....	4
Implementación del Autómata Determinista.....	5
Implementación del Autómata No Determinista.....	6
Ejemplos de Autómatas Deterministas.....	6
Ejemplos de Autómatas No Deterministas.....	7

Introducción

En esta práctica vamos a desarrollar un autómata finito determinista y un autómata finito no determinista. Para ello primero qué es cada uno y la diferencia que podemos encontrar entre ambos.

Un autómata finito está definido por un conjunto de estados y unas transiciones que irá realizando debido a unas entradas externas. Se clasifican de la siguiente forma:

-Deterministas: El autómata solo puede encontrarse en un estado en un instante dado.

-No deterministas: El autómata es capaz de estar en varios estados a la vez.

Aunque ambos definen a los lenguajes regulares, los no deterministas son capaces de describir ciertos problemas de forma más eficiente.

Autómata determinista

Este autómata solo es capaz de moverse a un estado determinado cuando lee un símbolo de la cadena que se le pasa como parámetro. La cadena será aceptada si el estado en el que finaliza el autómata es final.

Si en la aplicación decidimos crear un autómata desde teclado, primero debemos ir añadiendo las distintas transiciones. El principal problema que he encontrado en esta parte es que no pueden existir dos transiciones iguales y además no debe existir una transición de un estado q_1 a un estado q_2 con dos símbolos distintos. Este problema se soluciona añadiendo un esquema de búsqueda para comprobar si no existe o una transición igual o una transición con el mismo estado 1 y estado 2 en el conjunto de transiciones del autómata, si no se encuentra se añade esta nueva transición y si no muestra un mensaje de error.

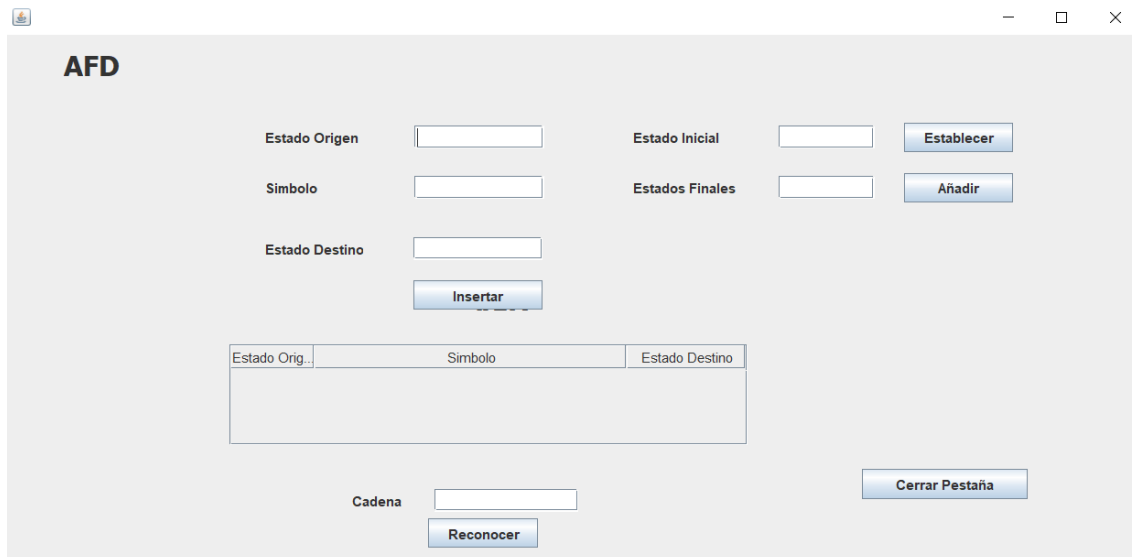
A medida que añadimos transiciones, automáticamente se van añadiendo los estados de las transiciones al conjunto de estados del autómata. El único problema que ha ocurrido aquí es que no se debe añadir dos veces el mismo estado ya que estando una vez funciona. Para arreglar esto se pone un if para comprobar que el conjunto de

estado no contiene ya a ese estado(El conjunto de estados se ha implementado mediante una List<String>).

Una vez declaradas todas las transiciones, se debe rellenar el campo estado inicial con un estado válido ya que no se puede escribir un estado que no existe(Esto lo comprueba la aplicación).

Por último, para tener por completo el autómata, se debe añadir uno por uno los estados finales del autómata, que se irán añadiendo al conjunto de estados finales y a su vez irá comprobando si estos estados existen.

Para comprobar el correcto funcionamiento del autómata, se rellena el campo cadena y se pulsa el botón Reconocer, esto devolverá Aceptado o Rechazado dependiendo de si la cadena es válida o no.



Autómata no determinista

Este autómata es capaz de estar en varios estados a la vez en un momento dado (es lo que se denomina macroestado). Además de avanzar hacia otro macroestado con caracteres normales, también es capaz de avanzar con un carácter vacío(es lo que se conoce como λ). Para este tipo de autómatas, una cadena será aceptada si cuando finaliza, alguno de los estados pertenecientes al macroestado en el que se encuentra es final. La última diferencia con el autómata determinista es que el determinista puede tener varias transiciones con el mismo símbolo a distintos estados.

En la aplicación debemos rellenar las distintas transiciones del autómata y automáticamente se irán añadiendo los estados al conjunto de estados del autómata.

A su vez, si queremos que un estado tenga transición lambda a algún estado, debemos escribirlo en el campo Transiciones lambda.

Los problemas encontrados en esta parte han sido parecidos a los descritos anteriormente y se han solucionado de una forma parecida.

Para reconocer una cadena con este autómata, debemos tener en cuenta la lambda-clausura. Esto es una función que devuelve el conjunto de estados a los que se puede mover el autómata con el símbolo λ , y además, añade al conjunto los estados a los que son capaces de moverse los estados a los que se mueve el estado inicial, por tanto, debe ser una función recursiva, ya que tiene que realizar esta secuencia hasta comprobar que todos los estados que debe mirar no sean capaces de hacer una transición λ .

AFND

Transiciones lambda (Separar por comas) Estado Inicial Establecer

Estado Origen Estados Finales Añadir

Simbolo

Estados Destino (Separar por comas)

Insertar

Estado...	Simbolo	Estado Dest.	λ

Cerrar Pestaña

Cadena Reconocer

Implementación del Autómata determinista

Tanto el conjunto de estados como el conjunto de estados finales se ha implementado con un `List<String>`. Debido a esto, debemos tener cuidado para que no haya duplicados en ellos.

El estado inicial es de tipo `String`, esta decisión se ha tomado para poder llamar a los estados de varias formas y no solo de forma numérica.

El conjunto de transiciones es un `List<TransicionAFD>` y ocurre el mismo problema que con los estados, debemos procurar que no existan duplicados. `TransicionAFD` es una clase que define cada transición del autómata, está compuesta por un estado 1(`String`), un símbolo(`char`) y un estado 2(`String`).

Para reconocer una cadena se realiza un bucle `for` que itera tantas veces como el tamaño de la cadena que le pasemos como parámetro. En cada iteración se realizará una transición a un estado dependiendo del símbolo que lea y cuando acabe, la función devolverá `true` si ese estado es final o `false` en caso contrario.

Implementación del Autómata no determinista

El conjunto de estados y estados finales siguen siendo un `List<String>`. El conjunto de transiciones con símbolo es un `List<TransicionAFND>`, `TransicionAFND` es una clase que define cada transición con un estado de origen, un macroestado de destino y un símbolo.

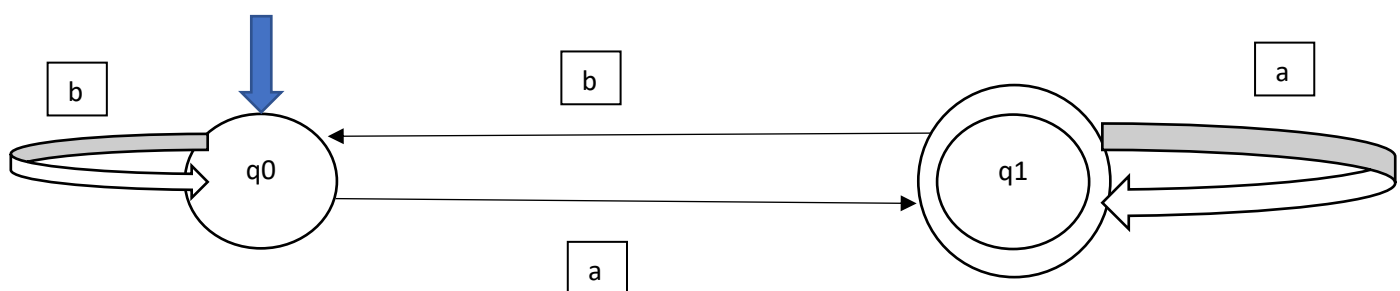
Este autómata también tiene un conjunto de transiciones lambda, que en este caso es un `List<Transicionlambda>`, una transición lambda está definida por el estado de origen y el macroestado de destino.

El estado inicial del autómata es de tipo `String`.

Para reconocer una cadena con un AFND primero debemos implementar la lambda-clausura. Para realizar esto debemos crear una función que al pasarle dos parámetros (un conjunto viejo y uno nuevo al que se le irá añadiendo los resultados) vaya comprobando a qué estados puedo llegar desde un estado con el símbolo λ . Además, hay que realizar esa misma comprobación en los estados a los que puede llegar, de esta forma se observa que seguramente deba implementarse de forma recursiva.

Una vez hecha la lambda-clausura, el autómata es capaz de reconocer las cadenas. Para conseguir esto, se debe iterar tantas veces como número de caracteres tenga la cadena y en cada iteración se recoge a qué macroestado se ha desplazado el autómata dado un símbolo, se realiza la lambda-clausura del macroestado y una vez finalizadas todas las iteraciones se comprueba si algunos de los estados pertenecientes al macroestado que hemos llegado es final o no.

Ejemplos de Autómatas deterministas



Estados: q0 q1

Símbolos: a b

Estados Finales: q1

Estado Inicial: q0

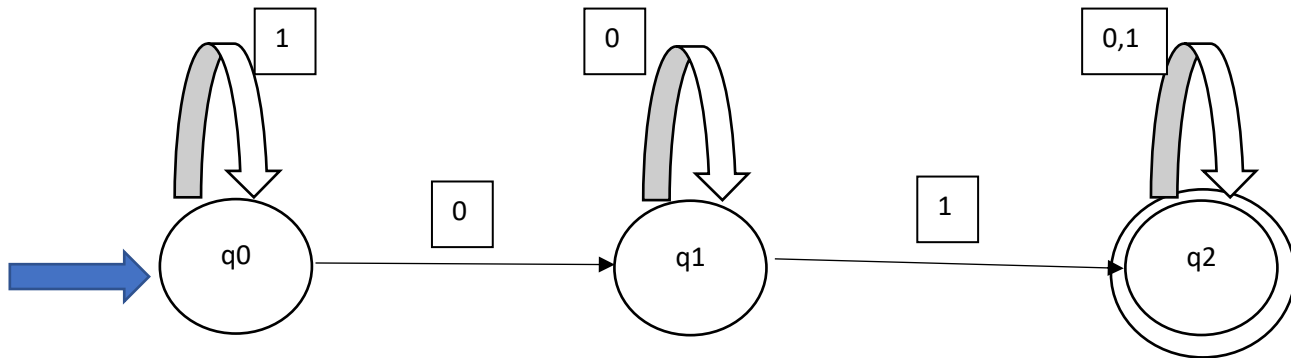
aaaaa->Reconocida

aaaaab->No reconocida

ab->No reconocida

aabbbbb->No reconocida

aaba->Reconocida



Estados: q0 q1 q2

Símbolos 0 1

Estados finales: q2

Estado inicial: q0

01->Reconocida

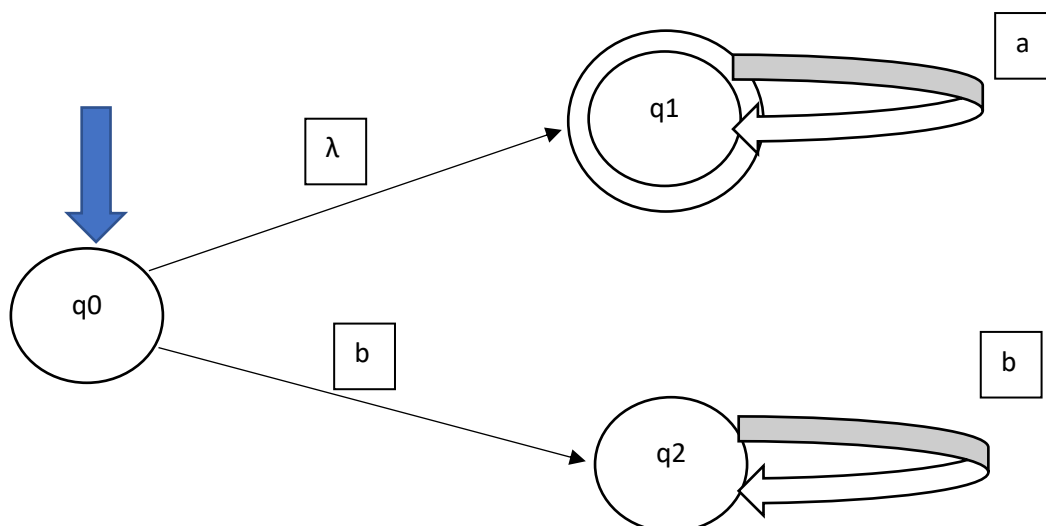
010->Reconocida

011->Reconocida

00->Rechazada

10->Rechazada

Ejemplos de Autómatas No deterministas



Estados: q0 q1 q2

Símbolos: a b λ

Estados finales: q1

Estado inicial: q0

a->Reconocida

aaaaa->Reconocida

ba->Reconocida

bbb->Rechazada

-El siguiente ejemplo corresponde al que reconoce los binarios cuyo tercer último dígito es un 0 mostrado en la práctica:

Estados: A B C D

Símbolos: 0 1

Estado final: D

Estado Inicial: A

000->Reconocida

011->Reconocida

111111->Rechazada

