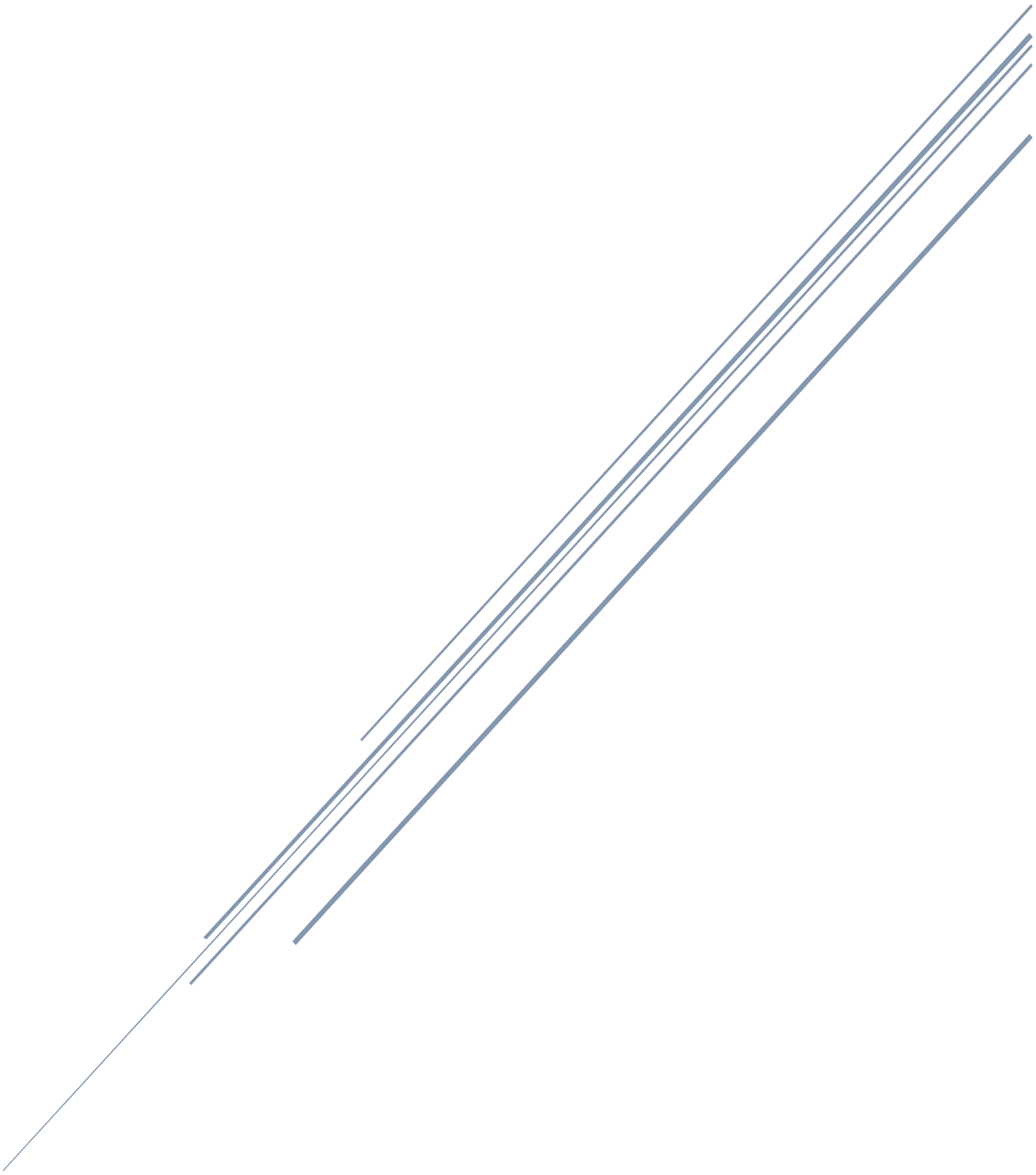
MEMORIA DESCRIPTIVA

De decisiones de diseño e implementación – P1



AUTÓMATAS Y LENGUAJES

Carlos Molinero Alvarado y Rafael Hidalgo Alejo

INTRODUCCIÓN.

Para la implementación de este proyecto se ha optado por el uso del lenguaje de programación C, con las flags de compilación -Wall, -Ansi, -pedantic.

Se ha utilizado el sistema operativo Ubuntu, así como el software necesario para la edición de textos y la web para realizar consultas.

DECISIONES DE DISEÑO.

Estructura intermedia.

Se ha optado por la implementación de una estructura intermedia "Autómata intermedio" (auti) como Tipo Abstracto de Dato (TAD). Esta estructura está compuesta a su vez por dos subtipos Abstractos de Datos – nuevoestado y transición - que, como sus propios nombres indican, representan los nuevos estados formados tras la ejecución del algoritmo y las transiciones entre ellos, respectivamente.

Así pues, un nuevoestado está formado por:

- Un nombre, formado por la concatenación ordenada de forma ascendente de los estados por los que está formado. Así pues, si un nuevoestado está formado por los estados antiguos qx y qy, si x < y se tendrá que el nombre será qxqy, siendo x e y enteros positivos mayores o iguales que cero.
  Nótese que, ante varios estados antiguos ordenados de distinta forma, el algoritmo devuelve siempre el mismo nombre. Para esto se utiliza la función de ordenación básica BubbleSort.
- Los estados antiguos por los que está formado.
- De forma auxiliar y para ayudarnos en el manejo de los datos, incluimos la variable entera nestados, que guarda el número de estados que posee el nuevoestado.

Por su parte, una transición estará formada por:

- Estado origen y estado final del que parte y llega la transición, respectivamente.
- Símbolo mediante el cual se transita.

La estructura intermedia está compuesta por los siguientes elementos:

- Array de nuevoestados.
- Array de transiciones.
- Alfabeto: los símbolos que reconoce el autómata.
- De forma auxiliar, las siguientes variables enteras: nestados, ntransiciones y nsimbolos, que representan la cantidad de nuevoestado, transiciones y símbolos, respectivamente.

Algoritmo:

Para realizar el algoritmo no nos hemos apoyado en la API que se nos facilitaba, sino que nos hemos basado en el código realizado en la P0 y en el formato .txt que utilizábamos para expresar un AFND con lambda. El procedimiento que hemos seguido es el siguiente:

Lo que utilizamos a lo largo de todo el código son transiciones. Partimos de una sola transición actual que solo está compuesta por un estado destino que es el estado inicial. A continuación, comprobamos para todas las transiciones actuales (inicialmente solo una), si podemos expandirla a través de una transición lambda. En este caso, modificamos el estado destino creando uno nuevo que será la unión de todos los que hemos transitado con lambda partiendo de la inicial.

A continuación es cuando podemos almacenar tanto las transiciones actuales de esta iteración, como sus estados destino en nuestro autómata estructura intermedia, que almacenará todas las transiciones y estados del AFD final.

Una vez que ya hemos almacenado, hemos de comprobar si, con estas transiciones actuales, realmente hemos descubierto un nuevo estado, ya que si no es así, hemos acabado de crear el AFD y por tanto nuestro algoritmo ha de finalizar.

Si por el contrario en nuestras transiciones hemos descubierto nuevos estados, lo que hacemos es ver a partir de estos estados destinos a qué nuevos estados podemos llegar, creando, para cada uno de ellos, nuevas transiciones, y comprobando que desde cada estado si tenemos varias transiciones para un mismo símbolo las unimos todas en una nueva transición hacia un único estado que los une.

Una vez que tenemos listas todas nuestras transiciones próximas, volvemos al principio del algoritmo copiándolas en las actuales, y, por lo tanto mirando si hay lambda, almacenándolas...

Transforma estructura:

Se ha implementado en el fichero transforma.c una rutina cuya función es la de transformar un autómata guardado en nuestra estructura intermedia en un autómata de la API. La función se llama *transforma_estructura.*

LECTURA Y EJECUCIÓN.

Para la lectura del autómata que queremos convertir, se ha optado por un fichero de texto, automata.txt, que debe encontrarse en la carpeta raíz donde se encuentran los demás archivos de nuestro proyecto. Este fichero contiene toda la información necesaria para que nuestro programa funcione correctamente.

La estructura de este archivo de texto debe ser la siguiente:

donde:

1) El primer dígito se corresponde con el número de estados que posee el autómata inicial.
2) El segundo dígito se corresponde con el estado que es inicial.
3) La siguiente cadena codifica los estados finales. Usamos un 0 si el estado no es final, y un 1 si sí lo es. Nótese que están ordenados, esto es, si el primer dígito de la cadena es un cero, significa que q0 no es final.
4) El próximo dígito representa el número de símbolos que posee el alfabeto.
5) La cadena siguiente será al propio alfabeto, incluyendo la lambda. (Lambda está codificada como "|").
6) Todo lo demás representa la tabla de transiciones entre los estados. La codificación seguida es de la forma:

	q0	q1	q2	q3	q4	q5
q0	-	+, |	-	-	-	-
q1	-	0	.	-	0	-
q2	-	-	-	0	-	-
q3	-	-	-	0	-	|
q4	-	-	-	.	-	-
q5	-	-	-	-	-	-

La tabla de transiciones del autómata, escrita de izquierda a derecha y de arriba abajo, donde un "-" significa que no se transita con ningún símbolo, y "|" significa lambda. Además, se utiliza un "*" como separador entre celdas, ya que se puede transitar a une estado con varios símbolos.

Para la compilación y ejecución se deben escribir los siguientes comandos en la terminal, dentro de la carpeta del proyecto:

make -> compilación.

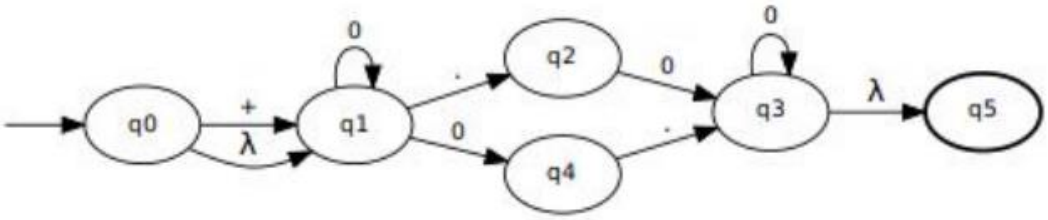
./transforma nombre_fichero_automata.txt -> ejecución, donde nombre_fichero_automata es el nombre del archivo de texto donde se encuentra el autómata, descrito como se ha comentado previamente.

Esto generará el autómata en un archivo .dot. **Para poder visualizarlo, se ha habilitado un comando, make png, que convierte el archivo .dot a uno del tipo .png.**

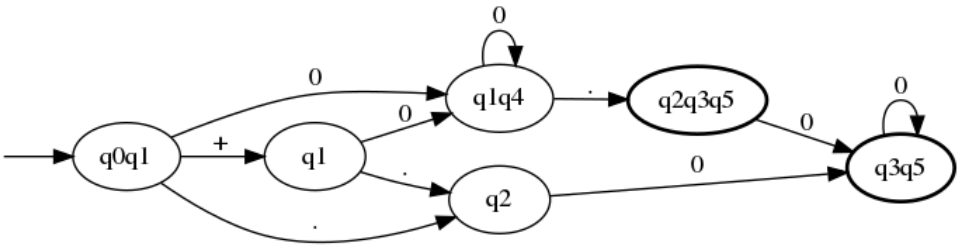
make clean -> borra los archivos generados por la compilación y ejecución anterior.

BANCO DE PRUEBAS.

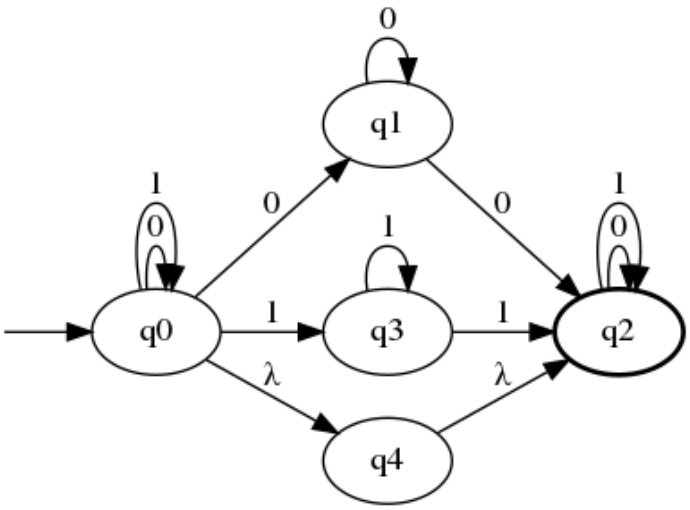
AUTÓMATA 1:



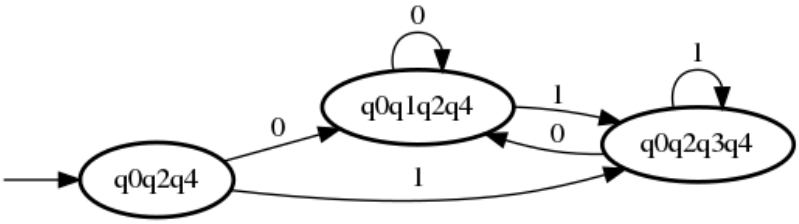
se convierte a:



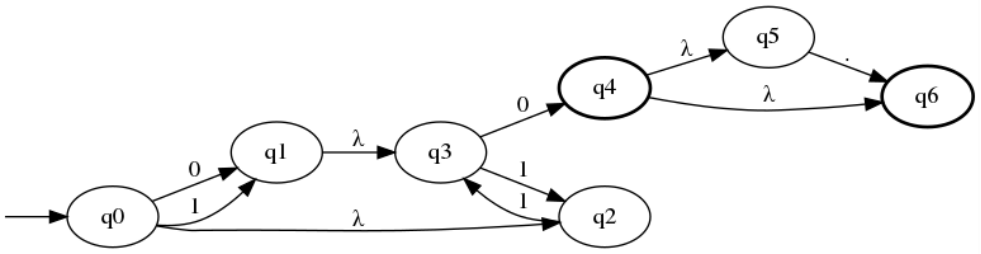
AUTÓMATA 2:



se convierte a:



AUTÓMATA 3:



se convierte a: