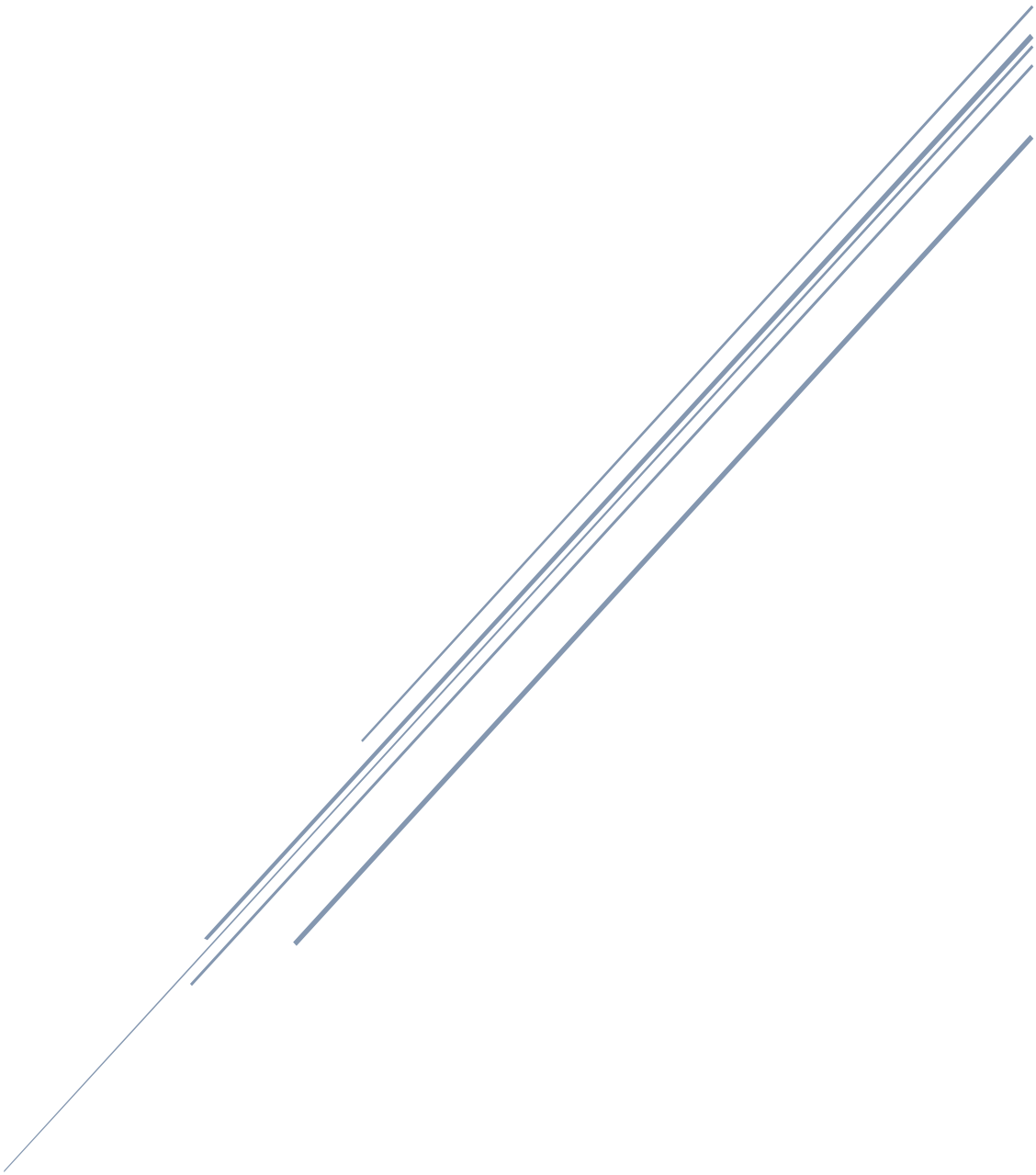
MEMORIA DESCRIPTIVA

De decisiones de diseño e implementación – P2



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, ESTRUCTURACIÓN Y EXPLICACIÓN DEL CÓDIGO.

Estructura intermedia, transformafin, leerFichero y transforma_estructura.

A la hora de implementar esta práctica, hemos decidido reutilizar código de la anterior. Para tenerlo de una forma más clara, y no confundirlo con el código nuevo de esta práctica hemos decidido modularizarlo en las siguientes funciones que se encuentran dentro de minimiza.c. Para poder utilizarlos importamos tanto afnd como transforma.h de la práctica anterior:

- leerFichero: En esta función tan solo recibimos el puntero al fichero que tenemos que leer y almacenamos todos los valores necesarios(nestados, finales.e iniciales, transiciones de uno a otro)… en variables globales para que se puedan acceder a ellos desde las distintas funciones.
- transformafin: En esta función, reutilizando el código de transforma, lo que hacemos es dado un AFD leído por fichero, su array de accesibles y su matriz de distinguibles generamos la estructura intermedia del autómata mínimo. Más adelante explicamos cómo hemos implementado el algoritmo. Seguimos dando uso de la estructura intermedia que implementamos en la práctica anterior para la creación de AFD minimizado.
- transforma_estructura: Esta función es idéntica a la de la práctica anterior. Dada la estructura intermedia del AFD minimizado, lo pasamos con las funciones de la API y generamos su .dot.

Distinguibles y accesibilidad:

Para controlar en el autómata los dos requisitos que se nos exigían, hemos creado dos funciones que, a partir de los datos almacenados en las variables globales, nos devuelven los accesibles y distinguibles de un AFD leído por fichero. Estas dos funciones son distinguibles y eliminarInaccesibles.

Algoritmo:

Para el algoritmo de distinguibilidad, hemos creado una pequeña estructura para definir una lista recursiva, y así poder tener una matriz de listas recursivas en las que para cada par posible, almacenamos todos los pares que recursivamente dependen de ellos, es decir, que han de ser marcados si marcamos al par en cuestión.

Una vez definida, la estructura, hemos seguido el algoritmo de distinguibilidad de las transparencias para poder realizarlo.

Para, una vez obtenido los accesibles y distinguibles, generar el AFD, lo que hemos hecho es utilizar directamente el código de la práctica anterior cambiando únicamente una parte. En la parte en la que comprobábamos antes de almacenar estado y transición en nuestra estructura intermedia a medida que recorríamos el autómata, ahora comprobamos accesibilidad, (ya que si no es accesible no lo almacenamos), y también distinguibilidad, ya que, al igual que con lambda, si llegas a un estado, también llegas a

todos sus indistinguibles, tan solo teniendo que mirar en la matriz de distinguibilidad diagonal quien tenía un 0 en la columna-fila del estado en cuestión.

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, aunque en estos autómatas al ser AFD no la usaremos. 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:

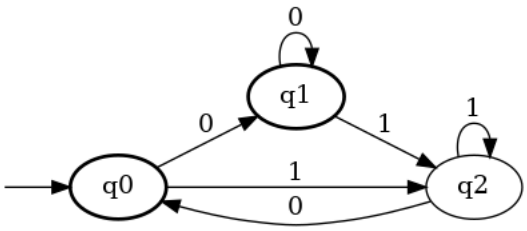
make -> compilación.

./minimiza 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.

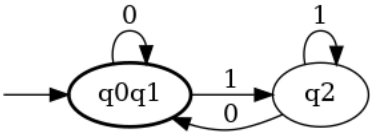
Esto generará el autómata en un archivo .dot. **Para poder visualizarlo, se ha habilitado un comando, make png, que convierte el archivo .dot a uno del tipo .png.** make clean -> borra los archivos generados por la compilación y ejecución anterior.

BANCO DE PRUEBAS.

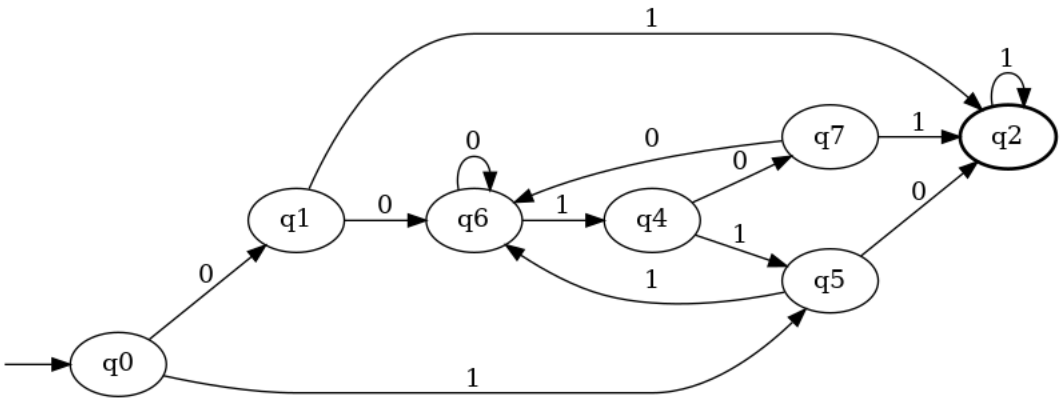
AUTÓMATA 1: (se encuentra en el archivo aut1.txt)



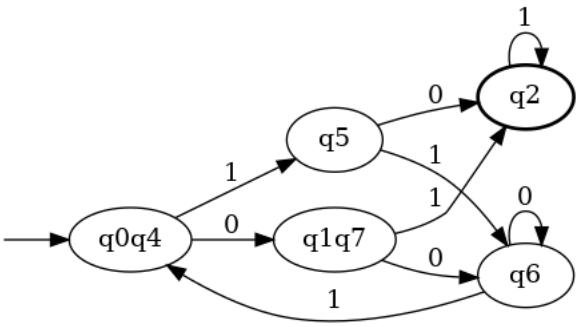
se convierte a:



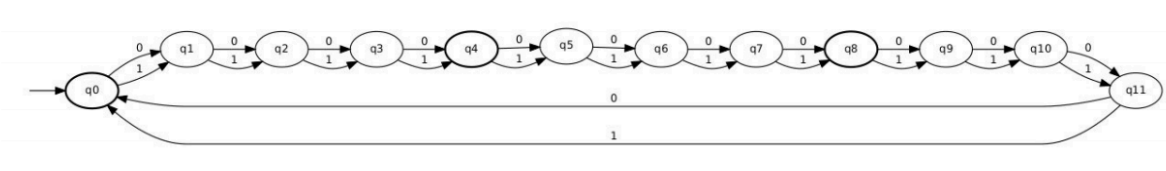
AUTÓMATA 2: (se encuentra en el archivo aut2.txt)



se convierte a:



AUTÓMATA 3: (se encuentra en el archivo aut3.txt)



se convierte a: