Eva Lacaba

Rodrigo Lardiés

**MEMORIA PRACTICA 2**

En esta práctica se implementa el algoritmo necesario para minimizar un autómata finito determinista. Se nos pedía para ello implementar la función *minimiza()* que nos devolviese el autómata de entrada minimizado.

Para realizar la práctica hemos utilizado la librería afnd, la misma de la práctica 1, así como los ficheros *intermedia.c* e *intermedia.h* que implementamos en la práctica anterior, y que necesitaremos para la creación de los estados de nuestro autómata minimizado.

Los ficheros *minimiza.c* y *minimiza.h* son los que contienen la función *AFNDMininiza,* que será la que implementa, con ayuda de llamadas a funciones auxiliares que hemos creado, la minimización del autómata.

Añadimos además una batería de ficheros de prueba con el fin de comprobar que nuestro algoritmo funciona correctamente.

**Decisiones de diseño**

Decidimos dividir en funciones las distintas tareas que necesitábamos llevar a cabo para la minimización de nuestro autómata. Es por ello que creamos una función *accesibles()* para descartar los estados inaccesibles del autómata y devolver sólo aquellos transitables; y una función *distinguibles()* que nos devuelva la matriz con las distintas clases que tendrá el nuevo autómata. Una vez tenemos esto ya estamos en disposición de empezar a crear los nuevos estados de nuestro autómata para su creación. Esta última parte la implementamos en la función *AFNDMinimiza()* y utilizamos las mismas estructura de estado intermedio y de transición que en la práctica anterior, del fichero *intermedia.*

**Función accesibles()**

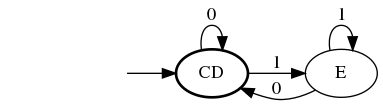
Para implementar el algoritmo de estados accesibles utilizamos dos listas: *accesibles* que marcará con 1 o 0 si el estado es accesible o no respectivamente; y *estados* del tamaño del número total de estados del autómata de entrada, inicializados a -1, y que irá guardando los estados a los que se vaya accediendo para recorrerlos. Iteraremos sobre esta última lista hasta que no haya más estados nuevos en ella que recorrer, y por cada uno miramos sus transiciones.

**Función distinguibles()**

Para implementar los estados distinguibles creamos una matriz y marcamos, de entre los estados accesibles, con 1’s (que identifican una clase diferente) todos aquellos estados finales o iniciales y finales que tenga nuestro autómata. Una vez hecho esto iteramos, comprobando para cada elemento de la matriz si sus transiciones llevan a estados de la misma clase o de distinta clase. Nuestra condición de parada será que no se produzca ningún cambio de clase en los elementos al recorrer la matriz.

**Función AFNDMinimiza()**

Por último explicamos la creación de nuestro nuevo autómata por medio del algoritmo de minimización.

**Banco de pruebas**

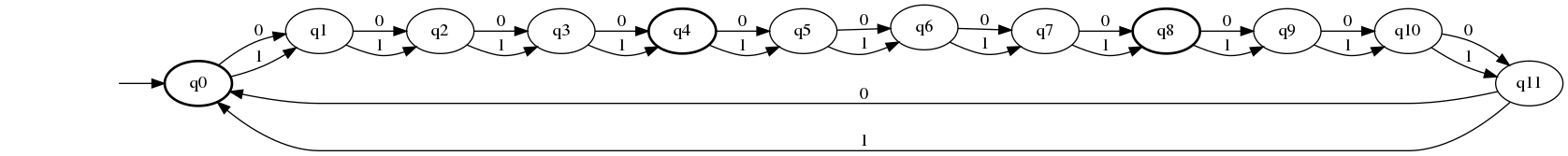
Para probar nuestro algoritmo hemos realizado diferentes pruebas para comprobar el comportamiento del autómata ante ejemplos diversos. Hicimos tres ficheros de prueba, además del que se nos proporcionaba en el enunciado, y creamos el script main.sh que hace el make y genera los autómatas de las pruebas y sus imágenes.

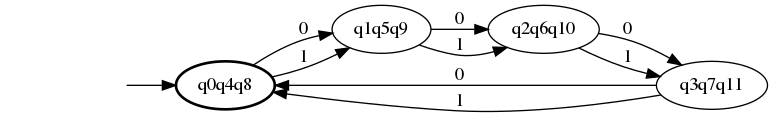
Ponemos a continuación la imagen del autómata de entrada y el minimizado.

**Main**

Este era el ejemplo de prueba que se nos proporcionaba para la realización de la práctica.

No minimizado

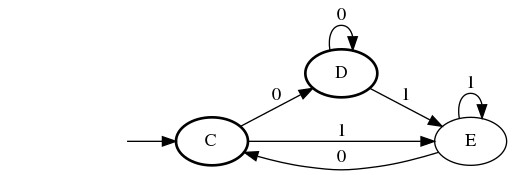


Minimizado

**Prueba 1**

Este autómata es muy sencillo, es el del powerpoint de explicación del algoritmo de minimización. Lo usamos para probar el funcionamiento del algoritmo cuando lo estábamos implementando para ver los errores más básicos.

No minimizado

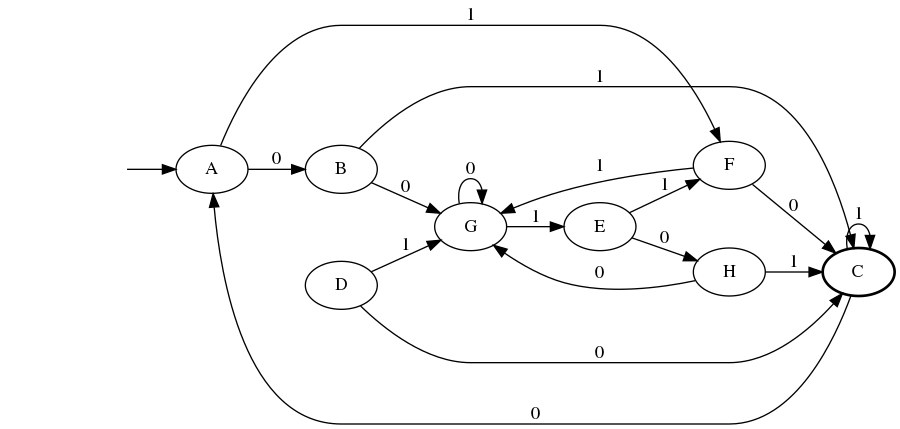


Minimizado

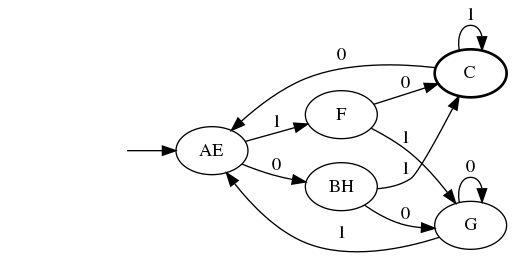
**Prueba 2**

Con este autómata probamos el correcto funcionamiento del autómata con más estados y uno de ellos inaccesible.

No minimizado



Minimizado



**Prueba 3**