Victoria Pelayo Alvaredo
José Benjumeda Rubio

Prácticas de Autómatas y Lenguajes. Curso 2019/20
Práctica 1: Conversión a determinista

En esta práctica hemos implementado un algoritmo para convertir un autómata finito no determinista (AFND), en un autómata finito determinista (AFD). Para ello, hemos utilizado la librería proporcionada, aunque hemos implementado dos TAD's, pack y transición, y algunas funciones.

El TAD pack representa un estado del autómata, y es un struct que almacena el numero total de estados individuales, el nombre que tendrá el conjunto de estados, un array de 0's y/o 1's que indicará por qué estados individuales está compuesto, un array de transiciones, un indice que nos sirve para saber cuantas transiciones tiene, y un tipo que nos dice si estamos en un estado inicial, inicial y final, normal o final.

El TAD transición contiene un símbolo y un estado destino. El estado origen no es necesario ya que será el que contenga a la transición, con lo cual una vez llegamos a la transición, ya conocemos el estado origen de ésta.

El algoritmo utilizado es el siguiente:

La función AFNDTransforma recibe el autómata no determinista. La estrategia será encontrar todos los nuevos estados y transiciones y una vez tengamos todo eso, crear el nuevo autómata en un solo paso.

Primero creamos el estado inicial. Aquí, la única manera de que esté formado por varios estados es que haya transiciones lambda, así que es lo único que comprobamos. Lo hacemos recorriendo todos los estados, y viendo si entre el inicial y este hay alguna transición lambda, mediante la función AFNDCierreLTransicionIJ. También comprobamos si tenemos que cambiar el tipo del estado por ejemplo en el caso de que nuestro estado q0 sea de tipo INICIAL pero haya una transición lambda al final. Aquí cambiaríamos el tipo de nuestro estado INICIAL a INICIAL_FINAL.

Creamos el pack del estado inicial, y lo añadimos a nuestro array de packs. A continuación entramos en el siguiente bucle, que para cada pack de array_packs, y para cada símbolo, vemos a qué estados podemos llegar desde el estado que cada pack representa, y cuando los tenemos todos, creamos un pack nuevo y una transición, siempre y cuando este nuevo estado no exista ya, en cuyo caso crearíamos solo la transición.

Esto se hace de la siguiente manera:
Creamos una copia del pack que vamos a estudiar, y comprobamos si está en array_packs_cerrados. Si no está, lo añadimos y seguimos estudiándolo, y si sí está, nos saltamos este pack, que ya está estudiado, y seguimos con el siguiente.

El estudio del pack consiste en ver a qué estados se puede llegar para cada letra, y para esto tenemos una función auxiliar que se llama nuevo_pack. Una vez conseguido el pack, que se creará siempre que haya al menos un estado al que transitar, si ya teníamos uno igual añadimos solo la

transición al pack que estábamos estudiando y del que ha surgido el nuevo pack, y si no, creamos también el pack nuevo y lo añadimos a nuestro array de estados nuevos, que es array_packs.

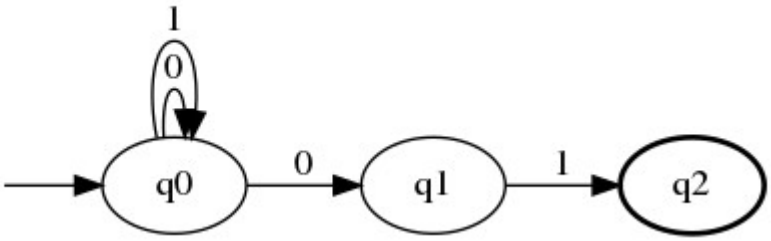
La función nuevo_pack funciona de la siguiente manera. Para el pack que recibe, vamos comprobando, para cada estado individual que lo forma, y para el símbolo recibido, a qué estados puede llegar, y ponemos un 1 en la posición correspondiente del array “estados”, y, después de recorrer todos todos los estados para ver si transita a ellos con el símbolo, los volvemos a recorrer comprobando si hay transiciones con lambda, utilizando las funciones de la librería afnd. Una vez conseguidos todos los estados construimos el nombre concatenando los nombres de todos los estados en orden. Finalmente creamos el pack y lo devolvemos.

Por último, lo que queda hacer en el bucle del que hablaba antes, es crear el autómata, pues ya tenemos todos los estados y transiciones.

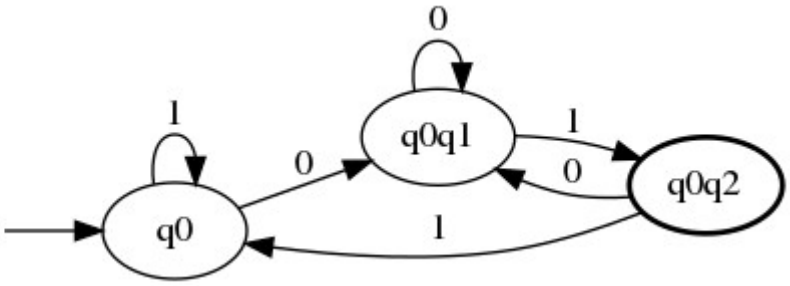
Ejemplos y pruebas: (Hacer make)

- **prueba:** para ejecutar hacer ./prueba y luego “dot -Tpng afd.dot > afd.png” y “dot -Tpng af12.dot > af12.png”. Ya se pueden abrir las imágenes y ver los resultados (“eog af12.png” y “eog afd.png”). Resultado de la transformación:

No determinista:



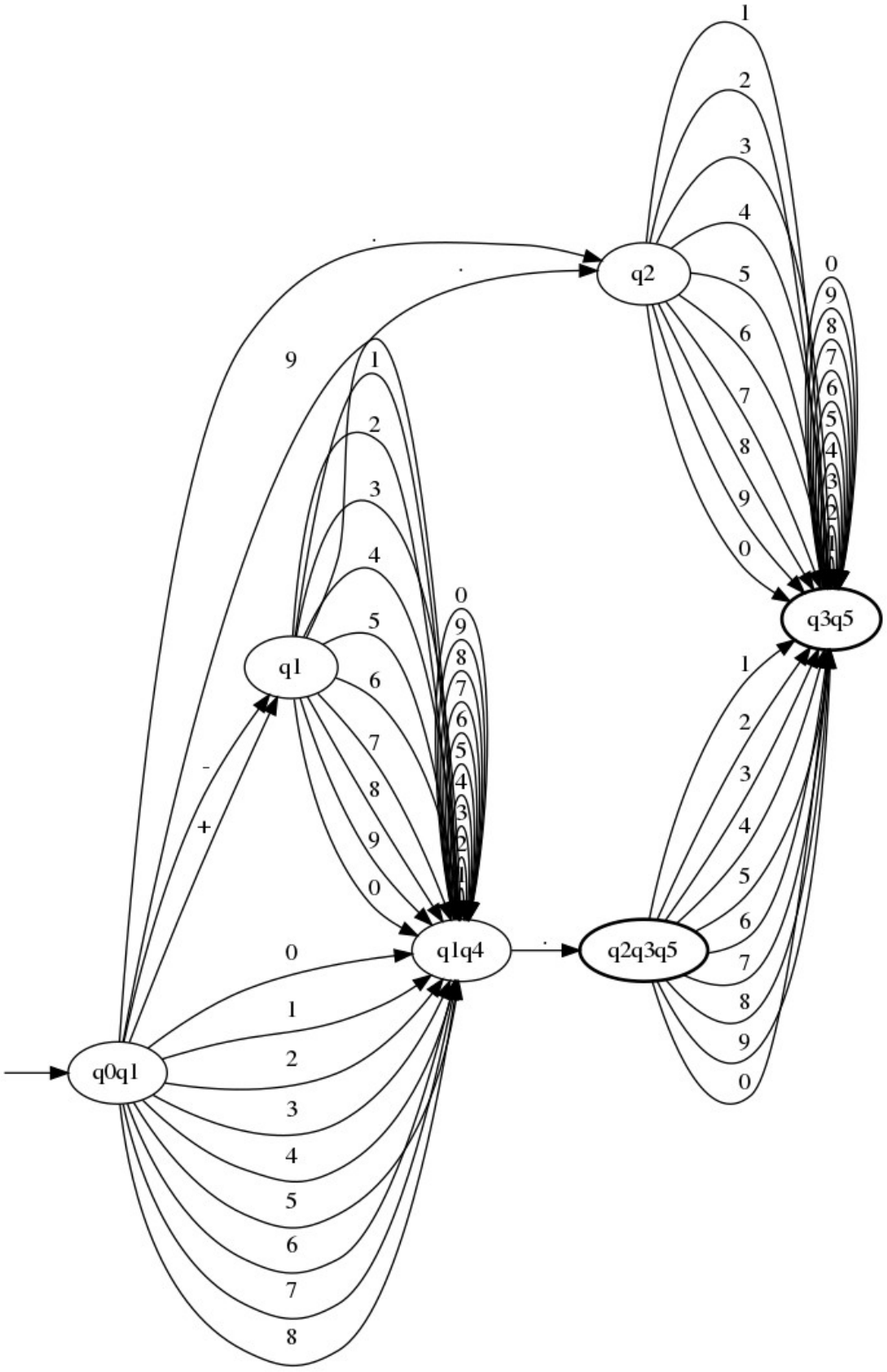
Determinista:



Victoria Pelayo Alvaredo
José Benjumeda Rubio

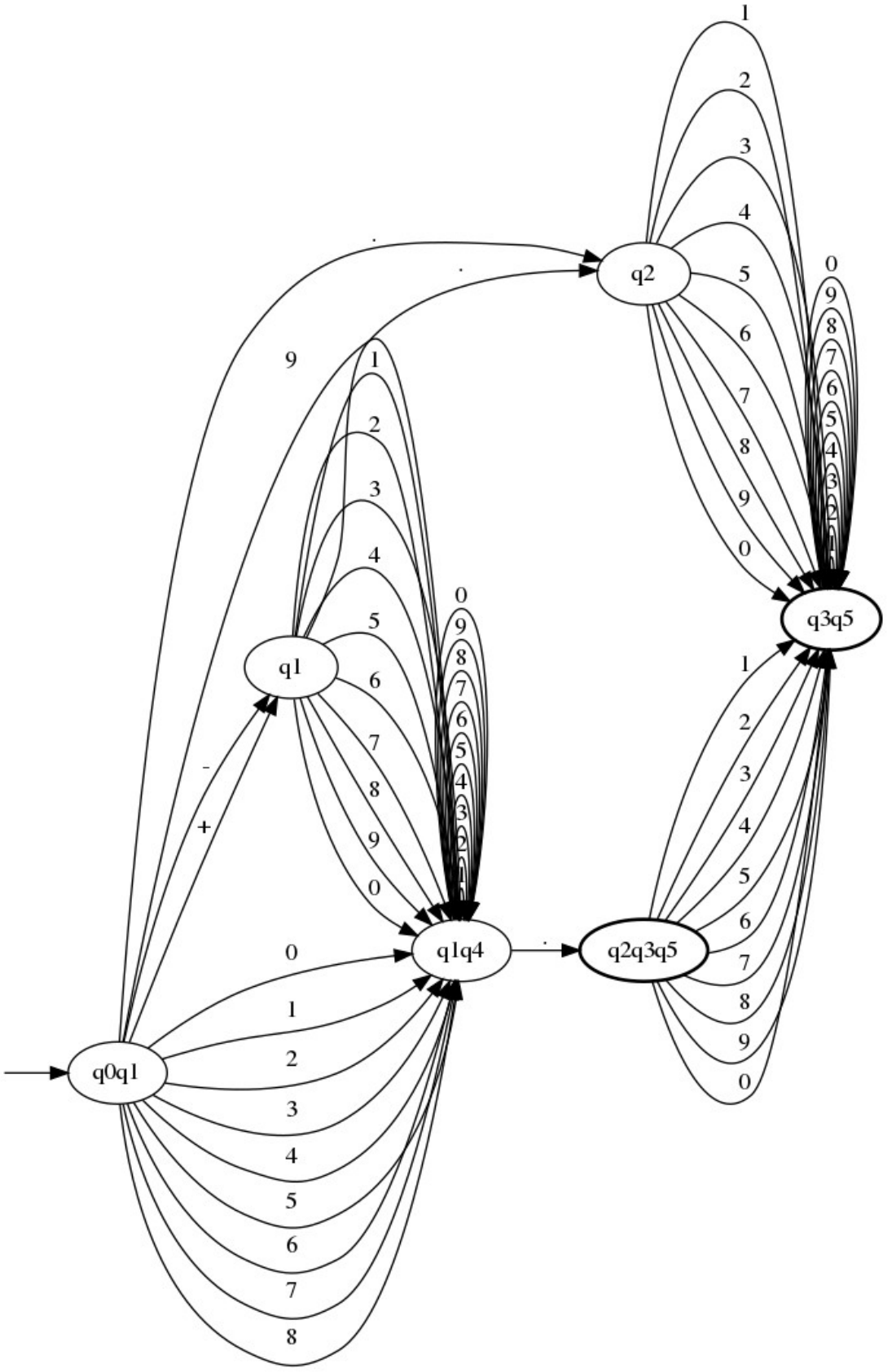
- **prueba3:** para ejecutar hacer ./prueba3 y luego “dot -Tpng afd.dot > afd.png” y “dot -Tpng af12.dot > af12.png”. Ya se pueden abrir las imágenes y ver los resultados (“eog af12.png” y “eog afd.png”). Resultado de la transformación:

Victoria Pelayo Alvaredo
José Benjumeda Rubio



Victoria Pelayo Alvaredo
José Benjumeda Rubio

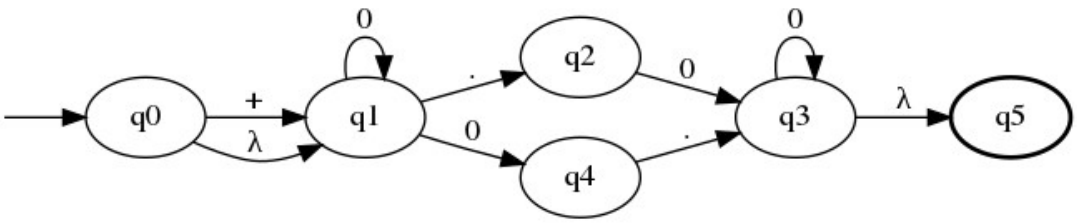
Victoria Pelayo Alvaredo
José Benjumeda Rubio



Victoria Pelayo Alvaredo
José Benjumeda Rubio

- **prueba_afnd**: para ejecutar hacer ./prueba_afnd y luego “dot -Tpng afd.dot > afd.png” y “dot -Tpng af11.dot > af11.png”. Ya se pueden abrir las imágenes y ver los resultados (“eog af11.png” y “eog afd.png”). Este fichero lo hemos modificado para que también imprima el automata no determinista resultado de la transformación:

No determinista:



Determinista: