Pareja 11: Eva Lacaba Nicolás, Rodrigo Lardiés Guillén

Prácticas de Autómatas y Lenguajes. Curso 2020/21

Práctica 1: Transformación a determinista

En esta práctica se nos pedía implementar la función *transforma()* que se encarga de dado un autómata finito no determinista, devuelve su autómata determinista asociado. Para ello seguimos el algoritmo estudiado en clase de teoría.

Para poder realizar una implementación sencilla, creamos los ficheros *intermedia.c* e *intermedia.h*, en los que definiremos nuestra estructura intermedia. Llamamos a nuestra estructura intermedia **estadoIntermedio**, y está compuesta de los siguientes elementos:

```
typedef struct _EstadoIntermedio {
    char nombre[MAX_NOMBRE];
    int *config_estado;
    int tipo_estado;
    int transiciones_guardadas;
    Transicion **transiciones;

}EstadoIntermedio;
```

```
typedef struct _Transicion{
    char operador[MAX_NOMBRE];
    char destino[MAX_NOMBRE];
    int *config_estado_destino;

}Transicion;
```

En nuestro algoritmo, tendremos una lista de estados intermedios vacía inicialmente. Cada estado intermedio que creemos será un estado de nuestro autómata determinista final. Éste estará compuesto de:

- **nombre**: será el nombre de este estado en el autómata final. No es más que la concatenación del nombre de los estados del autómata inicial que lo forman.
- **config_estado**: es un array de enteros que nos indica que estados del autómata inicial están presentes en el estado intermedio marcando a 1 estos estados y a 0 el resto. Por ejemplo para un autómata de 6 estados, la configuración de “q2” será 0-0-1-0-0-0.
- **tipo_estado**: un entero que nos indica el tipo de estado que es. Éste puede ser INICIAL, FINAL, NORMAL o INICIAL Y FINAL.
- **transiciones_guardadas**: un entero que nos indica el numero de transiciones que se pueden hacer desde este estado.
- **Transiciones**: un array de transiciones que contiene las transiciones posibles para ese estado. Las transiciones están compuestas a su vez por:
  - *operador*: símbolo por el que se llega estado destino mediante esta transición.
  - *destino*: nombre del estado destino.
  - *config_estado_destino*: configuración del estado destino.

Una vez hemos definido las estructuras, el algoritmo es sencillo. Creamos el estado intermedio inicial. Para ello tomamos el estado inicial y buscamos sus transiciones landa. Si las tiene, concatenamos sus nombres al estado inicial y creamos nuestro primer estado intermedio con este nombre. A continuación añadiremos todas las transiciones para este estado. A su vez, añadiremos las transiciones landa de los estados destino y tendremos el primer estado intermedio completo.

Para acabar el algoritmo, iteramos sobre el total de estados intermedios, y los vamos explorando uno a uno. Obtenemos las transiciones del estado que exploramos y si el estado destino no es un estado intermedio todavía lo crearemos añadiendo sus transiciones correspondientes. El algoritmo finaliza cuando no quedan estados intermedios por explorar y construimos nuestro autómata finito determinista con los estados intermedios y sus transiciones.

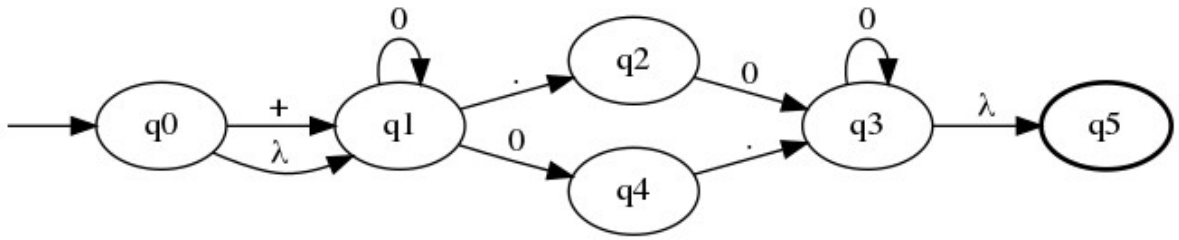
Pruebas

Para probar nuestro algoritmo, hemos realizado una serie de pruebas comprobando distintos aspectos en cada una.

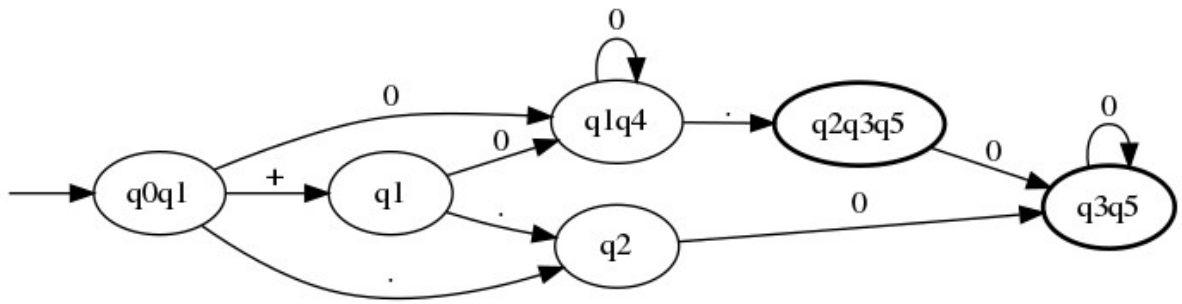
Main:

Este es el autómata que se nos daba de ejemplo:

- Autómata No Determinista



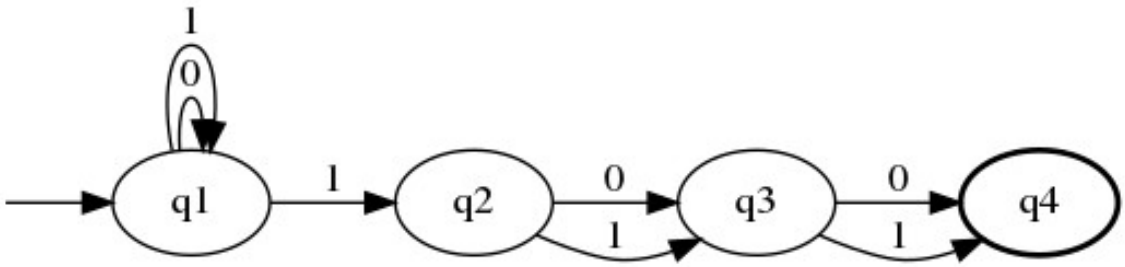
- Autómata Determinista



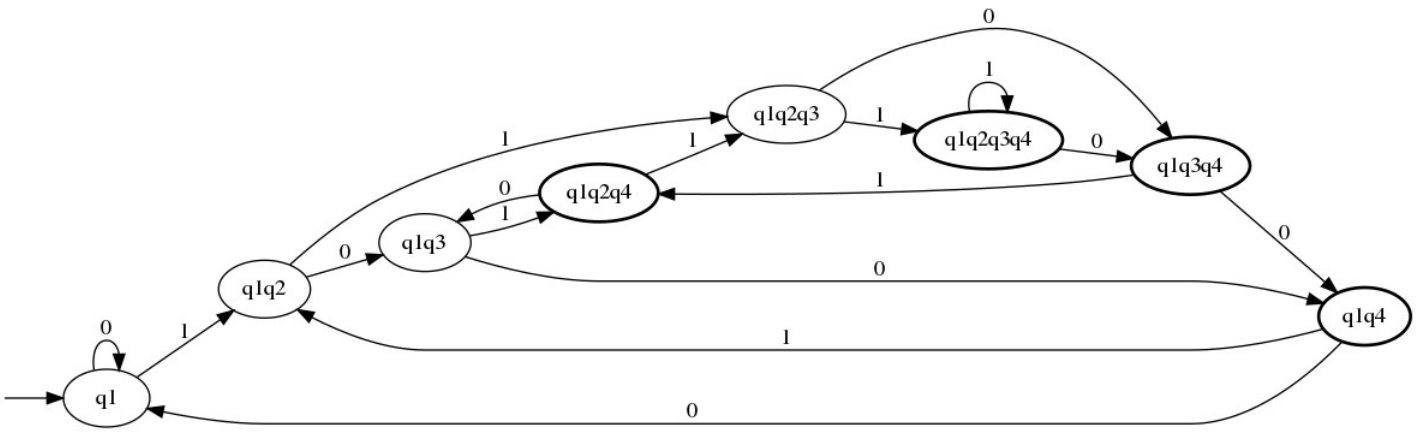
Ejericicio_prueba1:

En este ejercicio queremos probar que nuestro algoritmo funciona cuando no hay transiciones landa entre estados.

- Autómata No Determinista



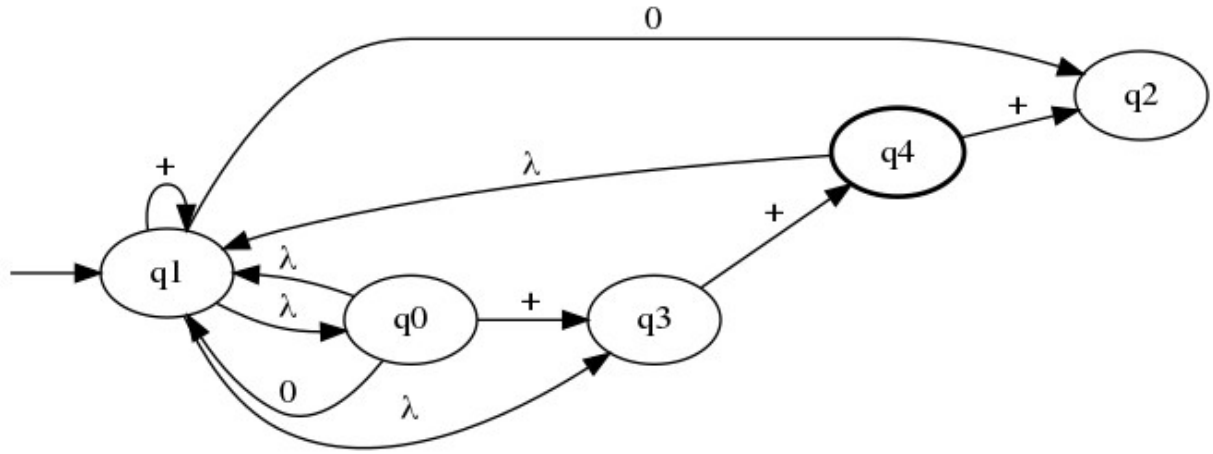
- Autómata Determinista



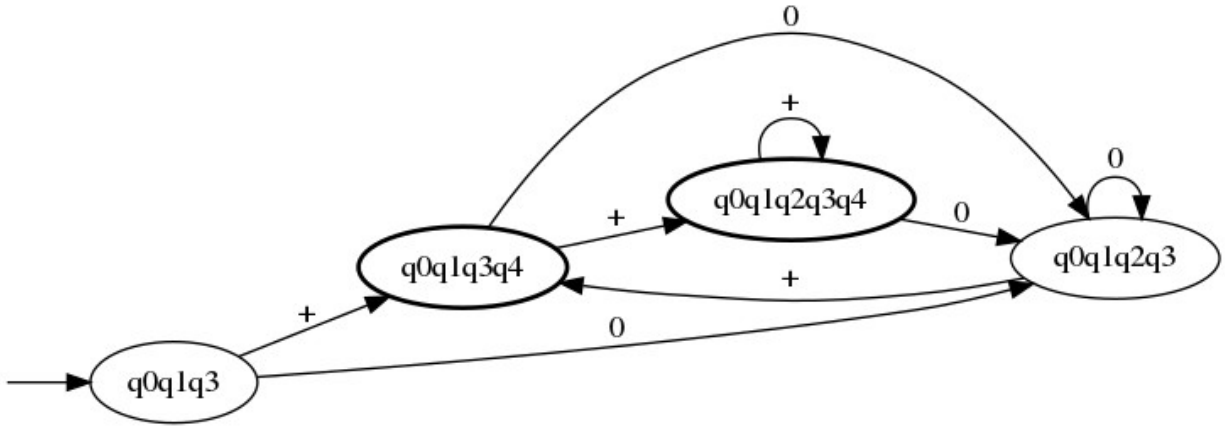
Ejercicio_prueba2:

En este ejercicio, el objetivo era probar nuestro algoritmo cuando el estado inicial es distinto de q0, y existen transiciones landa encadenadas, es decir, hay una transición de landa de q0 a q1, y a su vez una de q1 a q3.

- Autómata No Determinista



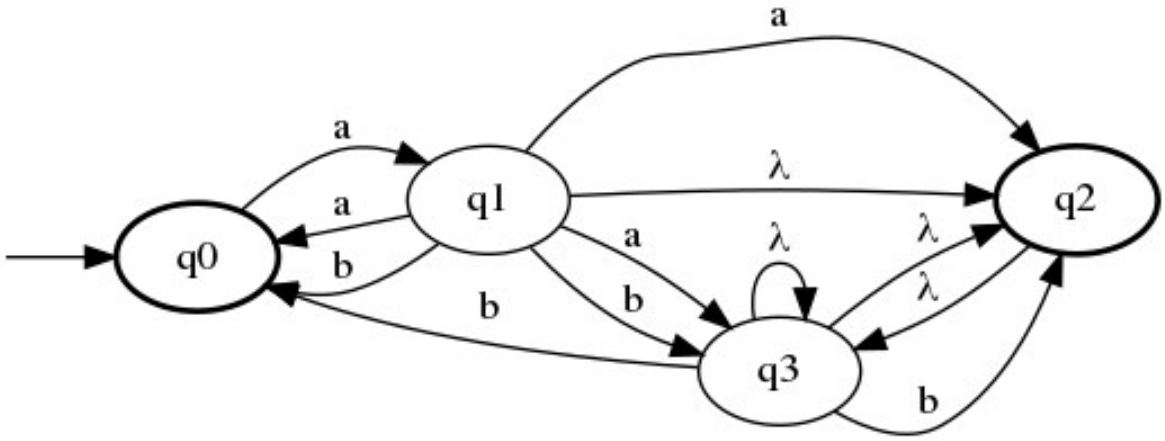
- Autómata Determinista



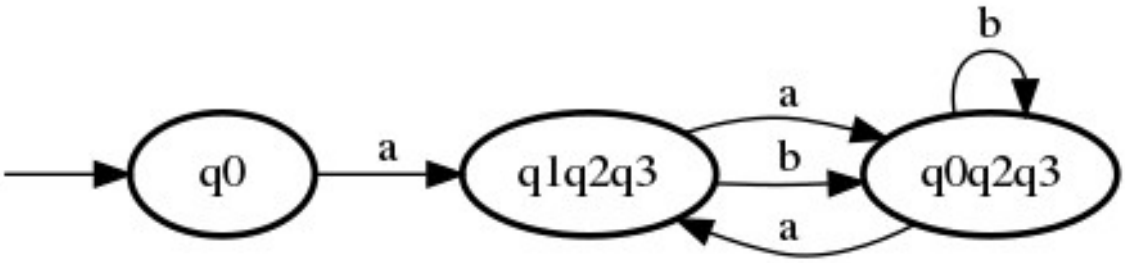
Ejercicio_prueba3:

En este ejercicio probamos el funcionamiento del algoritmo cuando hay estado INICIAL Y FINAL, las transiciones landa encadenadas, y cuando hay una transición landa de un estado sobre si mismo por si se produjera algún error de repetición en el nombre de los estados intermedios.

- Autómata No Determinista



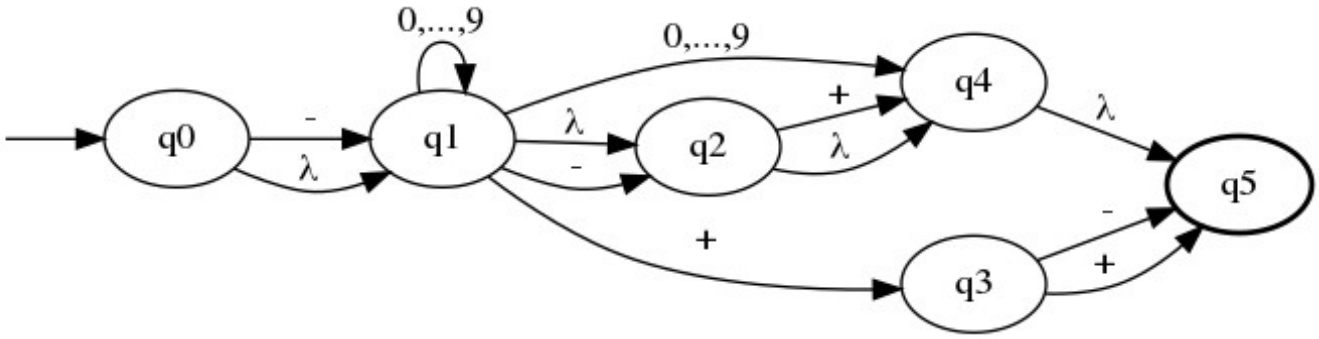
- Autómata Determinista



Ejercicio_prueba4:

Por último en este ejercicio, hemos propuesto un autómata un poco más complicado (6 estados), que usara como símbolo “0,…,9” , y que probara la mayoría de objetivos de las pruebas anteriores confirmando así finalmente el correcto funcionamiento de nuestro algoritmo.

- Autómata No Determinista:



- Autómata Determinista: