Prácticas de Autómatas y Lenguajes. Curso 2020/21
Práctica 2: Minimización de autómatas finitos deterministas

Descripción del enunciado

En esta práctica implementaremos un algoritmo para minimizar un autómata finito determinista (AFD) cualquiera. Para ello utilizaremos la librería ya existente que ya conoces de la práctica anterior

Minimización

Para minimizar el autómata se seguirán los siguientes pasos:

1. Tratar estados no accesibles y no productivos (como determine en clase su profesor)
2. Ejecutar el algoritmo que, partiendo del resultado del paso anterior, obtenga un autómata mínimo equivalente. El nuevo autómata se guardará en una estructura de tipo AFND, aunque en este caso será determinista. **Este algoritmo se describe en más detalle en las transparencias de Moodle.**

```
AFND * AFNDMinimiza(AFND * afnd);
```

En cuanto a la implementación del algoritmo descrito en las transparencias, hay que tener en cuenta que será necesario crear una estructura intermedia, ya que la API no permite añadir más estados que los indicados al construir la estructura. Por lo tanto el proceso consistirá de dos fases:

a. Creación de una estructura intermedia con los nuevos estados del AFD, y las transiciones. Para cada nuevo estado se debe guardar los estados del AFD original de los que está compuesto. Para realizar esta práctica tienes que decidir cómo implementar el algoritmo. Ten en cuenta que la explicación teórica no cierra todos los flecos del algoritmo. Tienes al menos dos aproximaciones posibles. Cualquiera que implemente correctamente el algoritmo vale.

i. Puedes intentar seguir paso por paso lo visto en las diapositivas de Moodle
   - Construcción de la matriz de la relación de distinguibilidad/indistinguibilidad
   - Deducción de la clase de equivalencia final.
   - Deducción de las transiciones

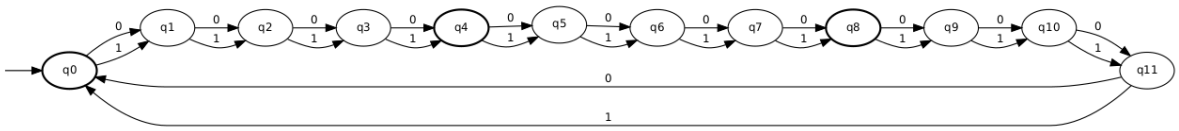
ii. Manipulación del conjunto de estados inicial para calcular la clase de equivalencia sin necesidad de la matriz de la relación de distinguibilidad/indistinguibilidad

b. Crear el AFD, usando la API de la librería AFND, con ayuda de la estructura creada en el paso anterior, y la información contenida en el AFD original. Para este paso crearemos un autómata nuevo, con la API de la librería de AFND. Marcaremos como estado inicial el que contenga al estado inicial del AFD original, y como finales cualquier estado compuesto por algún estado final del AFND original.

3. Producir el fichero en formato DOT, para poder visualizar el autómata gráficamente, usando la función de la librería. La función de conversión a DOT ya existe en la librería AFND, por lo que no será necesario implementarla.

Main de ejemplo

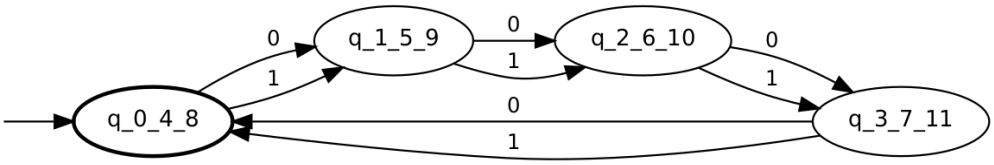
AFD original: es un autómata con demasiados estados redundantes para reconocer el lenguaje
{w{0,1}*:|w|=4n , n0}




```
#include <stdio.h>
#include "afnd.h"
#include "minimiza.h"


int main(int argc, char ** argv)
{

	AFND * p_afnd;
	AFND * p_afnd_min;

	p_afnd = AFNDNuevo("af1",12,2);



	AFNDInsertaSimbolo(p_afnd,"0");
	AFNDInsertaSimbolo(p_afnd,"1");

	AFNDInsertaEstado(p_afnd,"q0",INICIAL_Y_FINAL);
	AFNDInsertaEstado(p_afnd,"q1",NORMAL);
	AFNDInsertaEstado(p_afnd,"q2",NORMAL);
	AFNDInsertaEstado(p_afnd,"q3",NORMAL);
	AFNDInsertaEstado(p_afnd,"q4",FINAL);
	AFNDInsertaEstado(p_afnd,"q5",NORMAL);
	AFNDInsertaEstado(p_afnd,"q6",NORMAL);
	AFNDInsertaEstado(p_afnd,"q7",NORMAL);
	AFNDInsertaEstado(p_afnd,"q8",FINAL);
	AFNDInsertaEstado(p_afnd,"q9",NORMAL);
	AFNDInsertaEstado(p_afnd,"q10",NORMAL);
	AFNDInsertaEstado(p_afnd,"q11",NORMAL);



	AFNDInsertaTransicion(p_afnd, "q0", "0", "q1");
	AFNDInsertaTransicion(p_afnd, "q0", "1", "q1");
	AFNDInsertaTransicion(p_afnd, "q1", "0", "q2");
	AFNDInsertaTransicion(p_afnd, "q1", "1", "q2");
	AFNDInsertaTransicion(p_afnd, "q2", "0", "q3");
	AFNDInsertaTransicion(p_afnd, "q2", "1", "q3");
	AFNDInsertaTransicion(p_afnd, "q3", "0", "q4");
	AFNDInsertaTransicion(p_afnd, "q3", "1", "q4");
	AFNDInsertaTransicion(p_afnd, "q4", "0", "q5");
	AFNDInsertaTransicion(p_afnd, "q4", "1", "q5");
	AFNDInsertaTransicion(p_afnd, "q5", "0", "q6");
	AFNDInsertaTransicion(p_afnd, "q5", "1", "q6");
	AFNDInsertaTransicion(p_afnd, "q6", "0", "q7");
	AFNDInsertaTransicion(p_afnd, "q6", "1", "q7");
	AFNDInsertaTransicion(p_afnd, "q7", "0", "q8");
	AFNDInsertaTransicion(p_afnd, "q7", "1", "q8");
	AFNDInsertaTransicion(p_afnd, "q8", "0", "q9");
	AFNDInsertaTransicion(p_afnd, "q8", "1", "q9");
	AFNDInsertaTransicion(p_afnd, "q9", "0", "q10");
	AFNDInsertaTransicion(p_afnd, "q9", "1", "q10");
	AFNDInsertaTransicion(p_afnd, "q10", "0", "q11");
	AFNDInsertaTransicion(p_afnd, "q10", "1", "q11");
	AFNDInsertaTransicion(p_afnd, "q11", "0", "q0");
	AFNDInsertaTransicion(p_afnd, "q11", "1", "q0");
```

```
        p_afnd_min = AFNDMinimiza(p_afnd);

        AFNDADot(p_afnd_min);
        AFNDImprime(stdout,p_afnd_min);

        AFNDElimina(p_afnd);
        AFNDElimina(p_afnd_min);

        return 0;

}
```

Autómata determinista equivalente:



En esta práctica se pide:

- Cada grupo debe realizar una entrega
- Tu profesor te indicará la tarea Moodle donde debes hacer la entrega.
- Es imprescindible que mantengas los nombres especificados en este enunciado **sólo para las funciones que se piden de manera obligatoria**.
- Cualquier práctica que no compile (con los flags de compilación -Wall) , o no ejecute correctamente será considerada como no entregada.
    - Cualquier práctica que deje “lagunas de memoria” será penalizada de manera explícita en función de la gravedad de esas lagunas.