Prácticas de Autómatas y Lenguajes. Curso 2018/19
Práctica 3: AFNDs equivalentes a ER

Descripción del enunciado

Esta práctica tiene dos objetivos principales:

1. Que añadas a tu librería de simulación de autómatas finitos un conjunto de funciones que te permita diseñarlos a partir de expresiones regulares.
2. Que añadas a tu librería la posibilidad de representar gráficamente la parte estática de los autómatas que estamos tratando conectándola con la herramienta linux de *Graphviz* llamada *dot*.

Diseño de autómatas finitos no deterministas a partir de expresiones regulares

Como fruto de este objetivo, tu librería permitirá, para poder reconocer el lenguaje representado por la expresión regular:

$$11(0+1)^*$$

realizar una secuencia de llamadas a las funciones similares a esta:

```
/* SE CREA UN AUTÓMATA FINITO PARA LA EXPRESION REGULAR "1" */
       p_afnd_l1 = AFND1ODeSimbolo("1");
/* SE CREA UN AUTÓMATA FINITO PARA LA EXPRESION REGULAR "0" */
       p_afnd_l0 = AFND1ODeSimbolo("0");
/* SE CREA UN AUTÓMATA FINITO PARA LA EXPRESION REGULAR "1"."1" */
       p_afnd_l2 = AFND1OConcatena(p_afnd_l1, p_afnd_l1);
/* SE CREA UN AUTÓMATA FINITO PARA LA EXPRESION REGULAR "0"+"1" */
       p_afnd_l4 = AFND1OUne(p_afnd_l0, p_afnd_l1);
/* SE CREA UN AUTÓMATA FINITO PARA LA EXPRESIÓN ( "0"+"1" )*  */
       p_afnd_l5 = AFND1OEstrella(p_afnd_l4);
/* SE CREA UN AUTÓMATA FINITO PARA LA EXPRESIÓN "1"."1".( "0"+"1" )*   */
       p_afnd_l3 = AFND1OConcatena(p_afnd_l2, p_afnd_l5);
```

Tu profesor te explicará en clase qué algoritmo se aplica para obtener cada uno de estos autómatas finitos no deterministas con transiciones λ pero, como puedes observar en el ejemplo, este algoritmo se ciñe a la estructura recursiva de las expresiones regulares que has visto en la parte de teoría de la siguiente manera:

- Primero se crean los autómatas finitos de las expresiones regulares básicas,
    - En nuestro ejemplo se crean los dos siguientes

```
/* SE CREA UN AUTÓMATA FINITO PARA LA EXPRESION REGULAR "1" */
       p_afnd_l1 = AFND1ODeSimbolo("1");
/* SE CREA UN AUTÓMATA FINITO PARA LA EXPRESION REGULAR "0" */
       p_afnd_l0 = AFND1ODeSimbolo("0");
```

    - Pero si recuerdas la teoría, hay más expresiones regulares básicas; en concreto las siguientes (para las que también tendrás que poder crear los autómatas finitos no deterministas equivalentes)

        - λ
        - ∅ (conjunto vacío)
- Y a partir de autómatas para expresiones regulares cualesquiera
    - Se puede generar el autómata para su unión, concatenación o cierre estrella, como en las siguientes sentencias de nuestro ejemplo

```
p_afnd_l4 = AFND1OUne(p_afnd_l0, p_afnd_l1);
p_afnd_l2 = AFND1OConcatena(p_afnd_l1, p_afnd_l1);
p_afnd_l5 = AFND1OEstrella(p_afnd_l4);
```

Observa que estas funciones nuevas tienen como prefijo la palabra AFND1O que hace referencia, como te explicará tu profesor, a las condiciones que tienen que cumplir los autómatas finitos para ser combinados correctamente por este algoritmo con el objetivo de conseguir autómatas finitos de expresiones regulares más complejas (1O significa que sólo tiene **1** estado de salida u **O***utput*).

Como tu profesor te habrá explicado, esta forma de construcción de autómatas finitos usando un estilo que podríamos considerar como de *combinación de bloques* podría utilizar como bloques elementales para sus operaciones (unión, concatenación, estrella) cualquier autómata finito que cumpliera esas condiciones que los harían ser AFND1O.

Para ello tu librería también incluirá una función para obtener el autómata *de tipo* 1O a partir de cualquier autómata finito no determinista con transiciones lambda.

Éste sería su prototipo:

```
AFND * AFNDAAFND1O(AFND * p_afnd);
```

Observa el siguiente ejemplo de programa principal en el que se define mediante tu librería los autómatas mencionados en estos ejemplos (las nuevas funciones están resaltadas) y se muestran ejemplos de uso más exhaustivos

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


#include "alfabeto.h"
#include "estado.h"
```

```
#include "afnd.h"
#include "palabra.h"


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

       AFND * p_afnd_l0;
       AFND * p_afnd_l1;
       AFND * p_afnd_l2;
       AFND * p_afnd_l3;
       AFND * p_afnd_l4;
       AFND * p_afnd_l5;
       AFND * p_afnd_l6;



/* SE CREA UN AUTÓMATA FINITO PARA LA EXPRESION REGULAR "1" */
       p_afnd_l1 = AFND1ODeSimbolo("1");
/* SE CREA UN AUTÓMATA FINITO PARA LA EXPRESION REGULAR "0" */
       p_afnd_l0 = AFND1ODeSimbolo("0");
/* SE CREA UN AUTÓMATA FINITO PARA LA EXPRESION REGULAR "1"."1" */
       p_afnd_l2 = AFND1OConcatena(p_afnd_l1, p_afnd_l1);
/* SE CREA UN AUTÓMATA FINITO PARA LA EXPRESION REGULAR "0"+"1" */
       p_afnd_l4 = AFND1OUne(p_afnd_l0, p_afnd_l1);
/* SE CREA UN AUTÓMATA FINITO PARA LA EXPRESIÓN ( "0"+"1" ) *  */
       p_afnd_l5 = AFND1OEstrella(p_afnd_l4);
/* SE CREA UN AUTÓMATA FINITO PARA LA EXPRESIÓN "1"."1".( "0"+"1" )*   */
       p_afnd_l3 = AFND1OConcatena(p_afnd_l2, p_afnd_l5);


/* SE CREA UN AUTÓMATA FINITO PARA LA EXPRESIÓN "1" * */
       p_afnd_l6 = AFND1OEstrella(p_afnd_l1);

/* SE CALCULA EL CIERRE REFLEXIVO-TRANSITIVO DE TODOS LOS AUTÓMATAS */
      p_afnd_l0 = AFNDCierraLTransicion(p_afnd_l0);
      p_afnd_l1 = AFNDCierraLTransicion(p_afnd_l1);
      p_afnd_l2 = AFNDCierraLTransicion(p_afnd_l2);
      p_afnd_l3 = AFNDCierraLTransicion(p_afnd_l3);
      p_afnd_l4 = AFNDCierraLTransicion(p_afnd_l4);
      p_afnd_l5 = AFNDCierraLTransicion(p_afnd_l5);
      p_afnd_l6 = AFNDCierraLTransicion(p_afnd_l6);

/*******************************************************/
       fprintf(stdout,"EJEMPLO DE AUTÓMATA DE UNA EXPRESIÓN CORRESPONDIENTE A UN
SÍMBOLO: \"1\"\n");
       AFNDImprime(stdout,p_afnd_l1);


       fprintf(stdout,"EJEMPLO DE AUTÓMATA DE UNA EXPRESIÓN CORRESPONDIENTE A LA
CONCATENACIÓN DE OTROS DOS, JUSTAMENTE CONCATENA EL ANTERIOR CONSIGO MISMO\n");
       AFNDImprime(stdout,p_afnd_l2);


       fprintf(stdout,"A CONTINUACIÓN SE VA A MOSTRAR LA UNIÓN DE DOS AUTÓMATAS
QUE VIENEN DE EXPRESIONES REGULARES, UNO DE ELLOS ES EL CORRESPONDIENTE A LA
EXPRESION \"1\" QUE YA SE MOSTRÓ ANTERIORMENTE, EL OTRO ES EL DE LA EXPRESIÓN
\"0\" QUE SE MUESTRA A CONTINUACIÓN\n");
       AFNDImprime(stdout,p_afnd_l0);


       fprintf(stdout, "Y ESTA ES SU UNIÓN\n");
       AFNDImprime(stdout,p_afnd_l4);

       fprintf(stdout,"SE  MUESTRA  EL  AUTÓMATA  FINITO  CORRESPONDIENTE  A  LA
```

```
EXPRESION \"1\" * A PARTIR DEL AUTÓMATA ASOCIADO CON \"1\" QUE YA SE MOSTRÓ
ANTERIORMENTE\n");
	AFNDImprime(stdout,p_afnd_l6);


/*******************************************************/
	fprintf(stdout,"Y, A CONTINUACIÓN, ALGUNOS EJEMPLOS DE PROCESADO DE
CADENAS DEL AUTÓMATA DE LA EXPRESIÓN 11(0+1)*\n");
	AFNDImprime(stdout,p_afnd_l3);
	fprintf(stdout,"\tLA CADENA 11 ES RECONOCIDA POR EL AUTOMATA DE 11(0+1)*
POR SU PRINCIPIO\n");
	AFNDInsertaLetra(p_afnd_l3,"1");
	AFNDInsertaLetra(p_afnd_l3,"1");
	AFNDImprimeCadenaActual(stdout,p_afnd_l3);
	AFNDInicializaEstado(p_afnd_l3);
	AFNDProcesaEntrada(stdout,p_afnd_l3);
	AFNDInicializaCadenaActual(p_afnd_l3);
	fprintf(stdout,"\tLA CADENA 110 TAMBIÉN ES RECONOCIDA POR EL AUTOMATA DE
11(0+1)* \n");
	AFNDInsertaLetra(p_afnd_l3,"1");
	AFNDInsertaLetra(p_afnd_l3,"1");
	AFNDInsertaLetra(p_afnd_l3,"0");
	AFNDImprimeCadenaActual(stdout,p_afnd_l3);
	AFNDInicializaEstado(p_afnd_l3);
	AFNDProcesaEntrada(stdout,p_afnd_l3);
	AFNDInicializaCadenaActual(p_afnd_l3);
	fprintf(stdout,"\tLA CADENA 111 TAMBIÉN ES RECONOCIDA POR EL AUTOMATA DE
11(0+1)* \n");
	AFNDInsertaLetra(p_afnd_l3,"1");
	AFNDInsertaLetra(p_afnd_l3,"1");
	AFNDInsertaLetra(p_afnd_l3,"1");
	AFNDImprimeCadenaActual(stdout,p_afnd_l3);
	AFNDInicializaEstado(p_afnd_l3);
	AFNDProcesaEntrada(stdout,p_afnd_l3);
	AFNDInicializaCadenaActual(p_afnd_l3);
	fprintf(stdout,"\tLA CADENA 1111001 TAMBIÉN ES RECONOCIDA POR EL AUTOMATA
DE 11(0+1)* \n");
	AFNDInsertaLetra(p_afnd_l3,"1");
	AFNDInsertaLetra(p_afnd_l3,"1");
	AFNDInsertaLetra(p_afnd_l3,"1");
	AFNDInsertaLetra(p_afnd_l3,"1");
	AFNDInsertaLetra(p_afnd_l3,"0");
	AFNDInsertaLetra(p_afnd_l3,"0");
	AFNDInsertaLetra(p_afnd_l3,"1");
	AFNDImprimeCadenaActual(stdout,p_afnd_l3);
	AFNDInicializaEstado(p_afnd_l3);
	AFNDProcesaEntrada(stdout,p_afnd_l3);
	AFNDInicializaCadenaActual(p_afnd_l3);
	fprintf(stdout,"\tLA CADENA VACIA SIN EMBARGO NO ES RECONOCIDA POR EL
AUTOMATA DE 11(0+1)* \n");
	AFNDImprimeCadenaActual(stdout,p_afnd_l3);
	AFNDInicializaEstado(p_afnd_l3);
	AFNDProcesaEntrada(stdout,p_afnd_l3);
	AFNDInicializaCadenaActual(p_afnd_l3);

/*******************************************************/
	AFNDElimina(p_afnd_l0);
	AFNDElimina(p_afnd_l1);
	AFNDElimina(p_afnd_l2);
	AFNDElimina(p_afnd_l3);
	AFNDElimina(p_afnd_l4);
	AFNDElimina(p_afnd_l5);
	AFNDElimina(p_afnd_l6);
/*************************************************************************
/
```

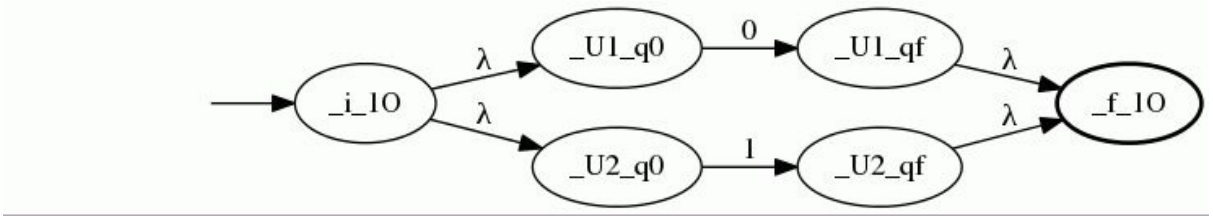
```
/*****************************************************************************
/
       return 0;

}
```

Como puedes observar, la primera secuencia de instrucciones es la que se ha descrito en el ejemplo inicial, pero también se crea un autómata para otras expresiones regulares y se muestra cómo procesan diferentes cadenas.

En Moodle puedes encontrar la salida que se espera para este programa. Su extensión recomienda que la consultes allí. En cualquier caso, podrás observar que el formato no cambia respecto a prácticas anteriores.

Representación gráfica compatible con dot (Linux)

Observa el siguiente diagrama



Es la imagen que se corresponde con el siguiente fuente compatible con la herramienta dot de linux:

```
digraph anfd1o_0_1_U_anfd1o_1_2  { rankdir=LR;
       _invisible [style="invis"];
       _U1_q0;
       _U1_qf;
       _U2_q0;
       _U2_qf;
       _i_1O;
       _f_1O [penwidth="2"];
       _invisible -> _i_1O ;
       _U1_q0 -> _U1_qf [label="0"];
       _U2_q0 -> _U2_qf [label="1"];
       _U1_qf -> _f_1O [label="&lambda;"];
       _U2_qf -> _f_1O [label="&lambda;"];
       _i_1O -> _U1_q0 [label="&lambda;"];
       _i_1O -> _U2_q0 [label="&lambda;"];
}
```

El objetivo de esta práctica no es que seas un experto diseñador de diagramas dot. Pero creemos que te será de ayuda comprobar gráficamente la corrección de tus operaciones.

Aunque puedes utilizar las características de dot que más te gusten siempre que los resultados resulten igual de claros que los que se te muestran, es suficiente con que sigas las siguientes líneas generales:

- `digraph` indica que es un grafo dirigido (es obligatorio)
- `anfd1o_0_1_U_anfd1o_1_2` es el nombre del grafo, puedes utilizar el que quieras
- Entre llaves debes describir el grafo del autómata
    - `rankdir=LR;` indica que se dibuje el diagrama de izquierda a derecha en lugar de arriba abajo.
    - Hay un truco para marcar el estado inicial:
        - Se crea un nodo invisible
          ```
          _invisible [style="invis"];
          ```
        - Y una transición desde él al nodo del estado inicial
          ```
          _invisible -> _i_1O ;
          ```
        - El estado final se muestra con su contorno más grueso
          ```
          _f_1O [penwidth="2"];
          ```
        - Las transiciones
            - muestran el estado origen y el destino mediante los nombres de sus nodos unidos por una flecha (->)
            - para indicar el símbolo mediante el que se transita de un estado a otro hay que asignar el nombre del símbolo como valor del atributo label de la transición.
              ```
              _U2_q0 -> _U2_qf [label="1"];
              ```
            - Cuando se quiere etiquetar una transición **λ** se puede usar la correspondiente etiqueta HTML
              ```
              _U2_qf -> _f_1O [label="&lambda;"];
              ```

En concreto esta imagen se ha obtenido a partir del fichero que se te muestra con el siguiente comando Linux

```
dot -Tgif anfd1o_0_1_U_anfd1o_1_2.dot > anfd1o_0_1_U_anfd1o_1_2.gif
```

Donde

- `anfd1o_0_1_U_anfd1o_1_2.dot` es el nombre del fichero que contiene la representación textual que se te ha mostrado inicialmente.
- es necesario volcar lo que el comando genera en el fichero que quieras que contenga la versión gif (`anfd1o_0_1_U_anfd1o_1_2.gif`)

En internet encontrarás muchas herramientas libres que te permiten visualizar estos diagramas.

Debes añadir a tu librería una función que genere un fichero .dot que represente gráficamente el diagrama del autómata en cuestión. Tienes libertad para elegir el nombre del fichero pero debes conservar la extensión .dot

```
void AFNDADot(AFND * p_afnd);
```

Descripción de la librería

A continuación resumimos las funciones que debes añadir obligatoriamente a tu librería:

Para la creación de los AFND1Os básicos

```
AFND * AFND1ODeSimbolo( char * simbolo);
```

Sigue las indicaciones de clase para construir el autómata finito para la expresión regular asociada con sólo un símbolo. Para ello:

- Usa un estado inicial
- Un estado final
- Una transición etiquetada con el símbolo entre ambos

```
AFND * AFND1ODeLambda();
```

Sigue las indicaciones de clase para construir el autómata finito para la expresión regular asociada con la cadena vacía. Para ello:

- Usa un estado inicial
- Un estado final
- Una transición lambda entre ambos

```
AFND * AFND1ODeVacio();
```

Casi con seguridad, en la práctica, no se usará esta función. Sólo para completar puedes crear este autómata, asociado con el lenguaje vacío, con

- Un estado inicial
- Un estado final
- Sin transiciones

```
AFND * AFNDAAFND1O(AFND * p_afnd);
```

Como verás en el laboratorio, para poder utilizar como bloque constructivo en este método un autómata finito, éste debe cumplir las siguientes condiciones:

- Tener un único estado inicial que no sea final.

- Tener un único estado final.

Si partes de un autómata que no cumpla alguna de estas condiciones, puedes conseguir crear un nuevo autómata finito equivalente al anterior y que sí las cumpla mediante los siguientes pasos que, aunque puedan contener en algunos casos algún elemento que podría simplificarse, te facilitará la codificación de la solución para cualquier caso general

- Añadir un nuevo estado inicial
- Añadir un nuevo estado final
- Añadir una nueva transición lambda desde el nuevo estado inicial al antiguo estado inicial
- Añadir nuevas transiciones lambda desde los anteriores estados finales al nuevo estado
- Quitar a los antiguos estados (inicial y finales) la esas características (de ser inicial o final).

Para la creación de los AFND1Os derivados de otros mediante operadores regulares

```
AFND * AFND1OUne(AFND * p_afnd1O_1, AFND * p_afnd1O_2);
```

Sigue las indicaciones de clase para crear un autómata con

- Un nuevo estado inicial (los dos de los antiguos dejarán de serlo)
- Un nuevo estado final (los dos de los antiguos autómatas dejarán de serlo)
- Nuevas transiciones lambda
    - Del nuevo estado inicial a los antiguos
    - De los antiguos estados finales al nuevo estado final.

```
AFND * AFND1OConcatena(AFND * p_afnd_origen1, AFND * p_afnd_origen2);
```

Sigue las indicaciones de clase para crear un autómata con

- Un nuevo estado inicial (los dos de los antiguos dejarán de serlo)
- Un nuevo estado final (los dos de los antiguos autómatas dejarán de serlo)
- Nuevas transiciones lambda
    - Del nuevo estado inicial al antiguo estado inicial del autómata origen 1
    - Del antiguo estado final del autómata origen 1 al antiguo estado inicial del autómata origen 2
    - Del antiguo estado final del autómata origen 2 al nuevo estado final.

```
AFND * AFND1OEstrella(AFND * p_afnd_origen);
```

Sigue las indicaciones de clase para crear un autómata con

- Un nuevo estado inicial (el del antiguo dejará de serlo)
- Un nuevo estado final (el del antiguo dejará de serlo)

- Nuevas transiciones lambda
    - Del nuevo estado inicial al antiguo
    - Del antiguo estado final al nuevo
    - Una para evitar el autómata antiguo, que vaya del nuevo estado inicial al nuevo final.
    - Otra para crear *el bucle* asociado con la operación * que vaya del nuevo estado final al nuevo estado inicial

Para la representación gráfica compatible con .dot de cualquier AFND

```
void AFNDADot(AFND * p_afnd);
```

Para ello

- Abre un fichero de texto
- Escribe la versión .dot de cada elemento siguiendo, por ejemplo, las indicaciones de este enunciado.
- Cierra el fichero

Observaciones

Es importante que comprendas y que tengas en cuenta las siguientes observaciones técnicas:

- Es especialmente importante ser cuidadoso con las reglas de formación de los diferentes componentes de los autómatas
    - *Alfabeto*
        - Puedes aplicar las siguientes reglas de formación
            - El alfabeto del autómata asociado con λ y con ∅ (conjunto vacío) está vacío.
            - El alfabeto de los autómatas asociados con un símbolo, contiene sólo ese símbolo.
            - El alfabeto de la unión o concatenación de otros dos es la unión de sus alfabetos.
            - El alfabeto del cierre estrella es el mismo que el del autómata de partida.
        - Esto es importante a la hora de saber qué tipo de cadenas puede aceptar cada autómata ya que tienen que estar formadas por los símbolos de sus alfabetos.

        - El nombre de los símbolos (la cadena de caracteres - char * - que los describe) no puede cambiar, tienes que mantener los mismos símbolos de partida.
    - *Estados*
        - Lo más importante es que decidas una política de renombrado de estados que asegure que no hay duplicidades, es decir, cuando combines más de un autómata, es posible que alguno de los nombres de estado se utilice en los dos autómatas, en este caso debes asegurarte de que en el autómata final los estados tienen un nombre único.
        - Tal vez lo más sencillo es que utilices un prefijo (idem sufijo) que añadas a cada estado en función de la operación que se realiza y de si es o no el primer operando para distinguirlos.
            - Por ejemplo, si haces la unión de dos autómatas y ambos llaman q0 a su estado inicial, el del primero podría acabar llamándose en el autómata de la unión U1_q0 y el del segundo U2_q0.
            - El convenio seguido en este enunciado representa respectivamente la unión, concatenación y cierre estrella con las letras U, K y X. Tú puedes utilizar el convenio que quieras.

En esta práctica se pide:

- Cada grupo debe realizar una entrega
- Tu profesor te indicará la tarea Moodle dónde y cuándo debes hacer la entrega.
- Cada grupo debe diseñar tantos módulos auxiliares (ficheros .c y .h) como necesite para completar la funcionalidad de la práctica, modificando para ello los que entregó al terminar la segunda práctica.
- Es imprescindible que mantengas los nombres especificados en este enunciado **sólo para las funciones que se piden de manera obligatoria**.
- Respecto a los módulos o funciones sugeridos tienes libertad de implementar estos u otros.