Prácticas de Autómatas y Lenguajes. Curso 2018/19

Práctica 1: Simulación de autómatas finitos no deterministas

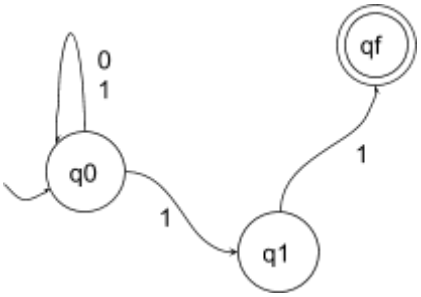
Descripción del enunciado

El objetivo de esta práctica es que diseñes una librería C que permita simular la definición y funcionamiento de cualquier autómata finito no determinista que no tenga transiciones λ.

Como introducción, se trata de que tu librería permita escribir programas C como el que se muestra a continuación y que implementaría y simularía el autómata finito no determinista para el lenguaje de las cadenas binarias que terminen en 1 1

$$L=\{x11,,x\in\{0,1\}^*\}$$

Y cuyo diagrama de estados se muestra a continuación




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


#include "afnd.h"

/* ESTOS INCLUDES DEPENDEN DE LA IMPLEMENTACIÓN, TAL VEZ TÚ DISEÑES OTROS MÓDULOS
*/
#include "alfabeto.h"
#include "estado.h"
#include "palabra.h"


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

/* DECLARACIÓN DE UN PUNTERO A UN AFND */
    AFND * p_afnd;


/* INICIALIZACIÓN DE UN NUEVO AFND DE NOMBRE af1 Y CON 3 ESTADOS Y 2 SÍMBOLOS EN
SU ALFABETO */
    p_afnd = AFNDNuevo("af1",3,2);


/* DEFINICIÓN DEL ALFABETO DEL AFND */
    AFNDInsertaSimbolo(p_afnd,"0");
    AFNDInsertaSimbolo(p_afnd,"1");
```

```
/* DEFINICIÓN DEL CONJUNTO DE ESTADOS */
    AFNDInsertaEstado(p_afnd,"q0",INICIAL);
    AFNDInsertaEstado(p_afnd,"q1",NORMAL);
    AFNDInsertaEstado(p_afnd,"qf",FINAL);

/* DEFINICIÓN DE LAS TRANSICIONES NO LAMBDA */
    AFNDInsertaTransicion(p_afnd, "q0", "0", "q0");
    AFNDInsertaTransicion(p_afnd, "q0", "1", "q0");
    AFNDInsertaTransicion(p_afnd, "q0", "1", "q1");
    AFNDInsertaTransicion(p_afnd, "q1", "1", "qf");

/* SE MUESTRA EL AFND DEFINIDO */
    fprintf(stdout,"\n****************** AFND ********************\n");
    AFNDImprime(stdout,p_afnd);
    fprintf(stdout,"\n********************************************\n");

/* DEFINICIÓN DE LA CADENA DE ENTRADA [ 0 1 0 1 1 ] */
    p_afnd= AFNDInsertaLetra(p_afnd,"0");
    p_afnd= AFNDInsertaLetra(p_afnd,"1");
    p_afnd= AFNDInsertaLetra(p_afnd,"0");
    p_afnd= AFNDInsertaLetra(p_afnd,"1");
    p_afnd= AFNDInsertaLetra(p_afnd,"1");

/* SE ESTABLECE COMO ESTADO ACTUAL DEL AUTÓMATA EL INICIAL */

    AFNDInicializaEstado (p_afnd);

/* SE MUESTRA LA CADENA ACTUAL */

    fprintf(stdout,"\n**************** PROCESA CADENA *************\n");
    AFNDImprimeCadenaActual(stdout,p_afnd);
    fprintf(stdout,"\n*********************************************\n");

/* SE PROCESA LA CADENA DE ENTRADA ACTUAL MOSTRANDO UNA TRAZA DEL FUNCIONAMIENTO
DEL AUTOMATA: EN CADA PASO DE ANÁLISIS SE MUESTRA LA CADENA ACTUAL Y EL CONJUNTO
DE ESTADOS EN LOS QUE SE ENCUENTRA EL AUTÓMATA */

    AFNDProcesaEntrada(stdout,p_afnd);


/* DEFINICIÓN DE LA CADENA DE ENTRADA [ 0 1 1 0 0 ] */
    p_afnd= AFNDInsertaLetra(p_afnd,"0");
    p_afnd= AFNDInsertaLetra(p_afnd,"1");
    p_afnd= AFNDInsertaLetra(p_afnd,"1");
    p_afnd= AFNDInsertaLetra(p_afnd,"0");
    p_afnd= AFNDInsertaLetra(p_afnd,"0");

/* SE ESTABLECE COMO ESTADO ACTUAL DEL AUTÓMATA EL INICIAL */

    AFNDInicializaEstado (p_afnd);

/* SE MUESTRA LA CADENA ACTUAL */

    fprintf(stdout,"\n**************** PROCESA CADENA *************\n");
    AFNDImprimeCadenaActual(stdout,p_afnd);
    fprintf(stdout,"\n*********************************************\n");


/* SE PROCESA LA CADENA DE ENTRADA ACTUAL MOSTRANDO UNA TRAZA DEL FUNCIONAMIENTO
DEL AUTOMATA: EN CADA PASO DE ANÁLISIS SE MUESTRA LA CADENA ACTUAL Y EL CONJUNTO
DE ESTADOS EN LOS QUE SE ENCUENTRA EL AUTÓMATA */

    AFNDProcesaEntrada(stdout,p_afnd);
```

```
/* SE LIBERAN TODOS LOS RECURSOS ASOCIADOS CON EL AFND */
    AFNDElimina(p_afnd);

    return 0;

}
```

Al ejecutar este programa la salida que se muestra es la siguiente. Fíjate en la información y el formato utilizados ya que tienes que conseguir que tus funciones generen una salida lo más parecida, en la medida de lo posible, a ésta. Para posibles situaciones que no estén incluidas en estos ejemplos utiliza un formato de salida similar.

```
****************** AFND ********************
af1={
   num_simbolos = 2

   A={ 0 1 }

   num_estados = 3

   Q={->q0  q1  qf*  }

   Funcion de Transición = {

          f(q0,0)={ q0 }
          f(q0,1)={ q0 q1 }
          f(q1,0)={ }
          f(q1,1)={ qf }
          f(qf,0)={ }
          f(qf,1)={ }
   }
}

*****************************************

*************** PROCESA CADENA *************
[(5) 0 1 0 1 1]

*****************************************

ACTUALMENTE EN {->q0 }
[(5) 0 1 0 1 1]

ACTUALMENTE EN {->q0 }
[(4) 1 0 1 1]

ACTUALMENTE EN {->q0 q1 }
[(3) 0 1 1]
```

```
ACTUALMENTE EN {->q0 }
[(2) 1 1]

ACTUALMENTE EN {->q0 q1 }
[(1) 1]

ACTUALMENTE EN {->q0 q1 qf* }
[(0)]

*************** PROCESA CADENA *************
[(5) 0 1 1 0 0]

*****************************************

ACTUALMENTE EN {->q0 }
[(5) 0 1 1 0 0]

ACTUALMENTE EN {->q0 }
[(4) 1 1 0 0]

ACTUALMENTE EN {->q0 q1 }
[(3) 1 0 0]

ACTUALMENTE EN {->q0 q1 qf* }
[(2) 0 0]

ACTUALMENTE EN {->q0 }
[(1) 0]

ACTUALMENTE EN {->q0 }
[(0)]
```

A continuación se describe con más detalle las funciones de la librería

```
AFND * AFNDNuevo(char * nombre, int num_estados, int num_simbolos);
```

- Genera un nuevo AFND (estructura de ese tipo)
    - Reserva memoria para todos los componentes necesarios.
    - Guarda como su nombre identificador el pasado como argumento.
    - Dimensiona alfabeto, estado y funciones de transición a partir de los valores de los argumentos num_estados y num_simbolos
    - Inicializa convenientemente todas las componentes

- Devuelve un puntero a esa estructura nueva (NULL si no es posible)

- Información necesaria en el AFND
  - Se recomienda que se guarde de forma explícita
    - El nombre
    - El alfabeto de entrada (con posibilidad de acceder fácilmente a su tamaño)
    - El conjunto de estados (con posibilidad de acceder fácilmente a su tamaño)
    - La función de transición (que excluye transiciones lambda)
      - Esto significa que la función de transición asigna a cada pareja (estado, símbolo de entrada) un conjunto de estados.
      - No se recoge en esta práctica nada relacionado con la posibilidad de que haya transiciones lambda.
    - La cadena que se está procesando
    - El conjunto de estados en el momento actual.
  - Estas dos últimas informaciones facilitarán el proceso de análisis de cualquier cadena que, como se ve en el programa principal que sirve de ejemplo inicial, consistiría en
    - Inicialmente los autómatas están en un conjunto de estados que sólo contiene al inicial.
    - Cuando se vaya a procesar una cadena de entrada:
      - Hay que insertar letra a letra la cadena que se debe analizar en el autómata.
      - Hay que indicarle que la procese con la llamada a la correspondiente función.
      - Cada símbolo de entrada será procesado de forma que se actualice tanto la cadena actual como el conjunto de estados actuales.
      - Esto se puede hacer fácilmente si se guardan esas dos informaciones.

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

- Libera adecuadamente todos los recursos asociados con un autómata finito no determinista.

```
void AFNDImprime(FILE * fd, AFND* p_afnd);
```

- Imprime en el FILE * proporcionado como argumento el autómata finito proporcionado como argumento.
- Debe seguirse el formato del ejemplo. En este sentido observa que para los estados se está utilizando la siguiente representación

- Si el estado es1 es inicial se representa como “->es1”
- Si el estado es3 es final se representa como “es3*”
- Si el estado s es normal se representa como “s”
- Si el estado s0_f es inicial y final se representa como “->s0_f*”

```
AFND * AFNDInsertaSimbolo(AFND * p_afnd, char * simbolo);
```

- Inserta en el alfabeto de entrada del autómata proporcionado como primer argumento un nuevo símbolo de entrada cuyo nombre se proporciona como segundo argumento.
- Debe hacerse una copia en memoria nueva para ser guardada en el autómata.

```
AFND * AFNDInsertaEstado(AFND * p_afnd, char * nombre, int tipo);
```

- Inserta en el conjunto de estados del autómata proporcionado como primer argumento un nuevo estado cuyo nombre se proporciona como segundo argumento y del tipo proporcionado como tercer parámetro.
- Se están utilizando los siguientes tipos (que deben ser definidos respetando estos nombres)
  - INICIAL
  - FINAL
  - NORMAL
  - INICIAL_Y_FINAL
- Debe hacerse una copia en memoria nueva para ser guardada en el autómata.

```
AFND * AFNDInsertaTransicion(AFND * p_afnd,
                             char * nombre_estado_i,
                             char * nombre_simbolo_entrada,
                             char * nombre_estado_f );
```

- Inserta en la función de transición guardada en el autómata proporcionado como primer argumento una nueva transición (una flecha con un símbolo de entrada del diagrama)
- La añade a la transición desde el estado de nombre nombre_estado_i y con el símbolo de entrada de nombre nombre_simbolo_entrada

```
AFND * AFNDInsertaLetra(AFND * p_afnd, char * letra);
```

- Inserta una letra nueva en la cadena de entrada que tiene que procesar el autómata.
- Debe hacer una copia en memoria nueva de la letra si lo necesita para guardarla.

- Observe en el ejemplo inicial del enunciado que los símbolos se introducen en el mismo orden en el que aparecen en la cadena. Es decir, si se desea que el autómata procese la palabra

  a1 a3 a0 a1 a1

- Se tienen que realizar 5 llamadas a esta función en el siguiente orden:
  a. a1
  b. a3
  c. a0
  d. a1
  e. a1

```
void AFNDImprimeConjuntoEstadosActual(FILE * fd, AFND * p_afnd);
```

- Imprime el conjunto de estados actual con el formato que puedes ver en el ejemplo

```
void AFNDImprimeCadenaActual(FILE *fd, AFND * p_afnd);
```

- Imprime la cadena que se está procesando en el instante actual (la pendiente de procesar) con el formato que puedes ver en el ejemplo

```
AFND * AFNDInicializaEstado (AFND * p_afnd);
```

- Establece como estado inicial del autómata el que se haya declarado como estado inicial.

```
void AFNDProcesaEntrada(FILE * fd, AFND * p_afnd);
```

- Desencadena el proceso completo de análisis de la cadena guardada como cadena actual.
- A su finalización
  - Se habrá mostrado en cada paso de proceso la información correspondiente al conjunto de estados actuales y a la cadena pendiente de procesar.
  - Por lo tanto la cadena pendiente de procesar debería estar vacía en un caso normal.

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

- Esta función debe realizar sólo un paso de proceso de la cadena actual (el correspondiente al siguiente símbolo, es decir, al primero de la cadena).

Se pide

- En tu clase tu profesor habrá formado los grupos de prácticas correspondientes a tu turno.
- Cada grupo debe realizar una entrega
- El tiempo del que dispones para hacer la práctica es el correspondiente a las semanas de las próximas tres sesiones, incluida esta en la que se te presenta el enunciado.
- Tu profesor te indicará la tarea Moodle donde debes hacer la entrega.
- Cada grupo debes diseñar tantos módulos auxiliares (ficheros .c y .h) como necesite para completar la funcionalidad de la práctica. De ellos, uno debe obligatoriamente ser el módulo afnd.c afnd.h que contiene al menos las funciones descritas en este enunciado.
- Es imprescindible que mantengas los nombres especificados en este enunciado.