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

Descripción del enunciado

En esta práctica implementaremos un algoritmo para convertir un autómata finito no determinista (AFND), en un autómata finito determinista (AFD). Para ello utilizaremos una librería ya existente, que incorpora además funciones para dibujar el autómata en formato DOT, de forma que podamos obtener una representación gráfica de los autómatas.

Conversión en determinista

Para convertir el autómata se seguirán los siguientes pasos:

1. Definir el autómata no determinista que se va a convertir usando las funciones de la librería AFND
2. Ejecutar el algoritmo, que partiendo del AFND anterior, genere el autómata finito 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 * AFNDTransforma(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 el 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 AFND original de los que está compuesto.
    i. Un AFND puede estar en varios estados simultáneamente. En el AFD simularemos esto creando un nuevo estado como la unión de los estados del autómata original en los que el AFND podía estar a la vez. Por ejemplo si el cierre transitivo del estado inicial es q0 y q1, el estado inicial del AFD será q0q1. En esta caso el criterio para nombrar los nuevos estados ha sido concatenarlos en orden alfabético. Se puede usar otro criterio para los nuevos nombres, siempre y cuando no genere dos nombres distintos para un mismos conjunto de estados del autómata original.
    ii. El algoritmo irá calculando todos los posibles estados desde cada estado nuevo, con todos los posibles símbolos. Hay que tener en cuenta que un estado en el nuevo autómata puede equivaler a varios estados del autómata original, y que por lo tanto hay que considerar todas las transiciones de cada uno de los estados originales que forman parte del nuevo estado. Al estado resultante, para cada símbolo, hay que añadir también los estados del original a los que se puede ir con una transición Lambda.
    iii. El algoritmo terminará cuando dejen de crearse nuevos estados.

iv. Para las transiciones, puesto que el nuevo autómata es determinista, será suficiente con guardar a qué estado transita, del AFD, con cada símbolo, desde cada nuevo estado.

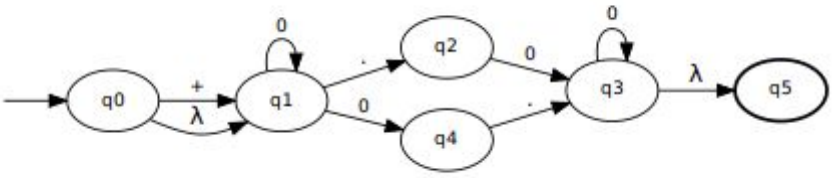
v. Se puede hacer tal como se indica en las transparencias, en dos pasadas, primero creando todos los estados y luego agregando las transiciones, o se pueden anotar las transiciones en la estructura intermedia.

b. Crear el AFD, usando el API de la librería AFND, con ayuda de la estructura creada en el paso anterior, y la información contenida en el AFND original. Para este paso crearemos un autómata nuevo, con el api de la librería de AFND. Marcaremos como estado inicial el primer estado, 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

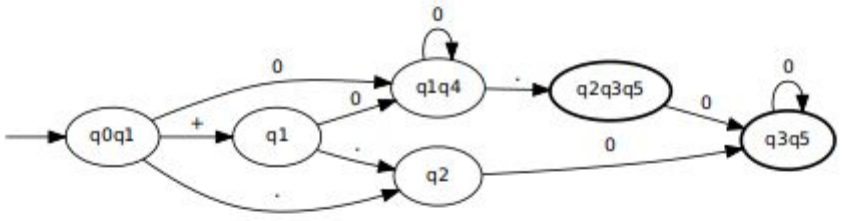
AFND original:




```
#include "afnd.h"
#include "transforma.h"

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

        AFND * p_afnd;
        AFND * afd;

        p_afnd= AFNDNuevo("af11", 6, 3);

        AFNDInsertaSimbolo(p_afnd,"+");
        AFNDInsertaSimbolo(p_afnd, "0");
        AFNDInsertaSimbolo(p_afnd,".");

        AFNDInsertaEstado(p_afnd, "q0",INICIAL);
        AFNDInsertaEstado(p_afnd, "q1", NORMAL);
        AFNDInsertaEstado(p_afnd, "q2", NORMAL);
        AFNDInsertaEstado(p_afnd, "q3", NORMAL);
        AFNDInsertaEstado(p_afnd, "q4", NORMAL);
        AFNDInsertaEstado(p_afnd, "q5", FINAL);
```

```
        AFNDInsertaTransicion(p_afnd, "q0", "+", "q1");
        AFNDInsertaTransicion(p_afnd, "q1", "0", "q1");
        AFNDInsertaTransicion(p_afnd, "q1", "0", "q4");
        AFNDInsertaTransicion(p_afnd, "q1", ".", "q2");
        AFNDInsertaTransicion(p_afnd, "q2", "0", "q3");
        AFNDInsertaTransicion(p_afnd, "q3", "0", "q3");
        AFNDInsertaTransicion(p_afnd, "q4", ".", "q3");

        AFNDInsertaLTransicion(p_afnd, "q0", "q1");
        AFNDInsertaLTransicion(p_afnd, "q3", "q5");
        AFNDCierraLTransicion(p_afnd);

        afd  = AFNDTransforma(p_afnd);
        AFNDImprime(stdout,afd);
        AFNDADot(afd);

        AFNDElimina(afd);
        AFNDElimina(p_afnd);

        return 0;
}
```

Autómata determinista transformado:



API de la librería AFND

A continuación se describe las cabeceras de las funciones de la librería.

Nota: Algunas de estas funciones sólo son necesarias para ejecutar el autómata. Aunque en esta práctica no es necesario la ejecución, puede servir para comprobar que el AFND y el determinista equivalente reconocen las mismas palabras.

```
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)

```
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.
- 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.
- Se realiza 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 (definidos en el fichero de cabecera)
    - INICIAL
    - FINAL
    - NORMAL
    - INICIAL_Y_FINAL

```
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
- 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 * AFNDInsertaLTransicion(
       AFND * p_afnd,
       char * nombre_estado_i,
       char * nombre_estado_f );
```

- Inserta en el AFND una transición lambda entre los dos estados cuyo nombre se proporciona.
- Se devuelve un puntero al AFND modificado (que se pasa como primer argumento)

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

- Inserta una letra nueva en la cadena de entrada que tiene que procesar el autómata.
- Realiza una copia en memoria nueva de la letra, si lo necesita para guardarla.
- Observe 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 tiene 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).

```
AFND * AFNDCierraLTransicion (AFND * p_afnd);
```

- Realiza el cierre transitivo de la relación de accesibilidad inducida por las transiciones lambda almacenada en el AFND que se proporciona como argumento.
- Se devuelve un puntero al AFND modificado.

```
AFND * AFNDInicializaCadenaActual (AFND * p_afnd );
```

- Esta función elimina la cadena actual dejándola vacía de forma que el autómata está preparado para especificar la siguiente cadena que se va a analizar mediante reiteradas llamadas a la función `AFNDInsertaLetra`

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

- Esta función guarda en un fichero predeterminado, la representación del autómata, de forma que se pueda generar fácilmente una imagen con la herramienta dot, o con el visor de archivos dot.
- El nombre del archivo será nombre_del_automata.dot
- Se debe utilizar un nombre fijo para el autómata determinista, “determinista”, de forma que el dot generado se llamará “determinista.dot”

Para poder analizar un AFND utilizaremos las siguientes funciones de la librería:

```
int AFNDNumSimbolos(AFND * p_afnd);
```

- Esta función devuelve el número de símbolos del alfabeto del autómata

```
int AFNDNumEstados(AFND * p_afnd);
```

- Esta función devuelve el número de estados del autómata

```
char * AFNDSimboloEn(AFND * p_afnd, int pos);
```

- Devuelve el símbolo que está en la posición pos del alfabeto. El primero es el 0.
- Importante: No es una copia, no se debe modificar ni eliminar

```
char * AFNDNombreEstadoEn(AFND * p_afnd, int pos);
```

- Devuelve el nombre del estado que está en la posición pos del autómata. El primero es el 0.
- Importante: No es una copia, no se debe modificar ni eliminar

```
int AFNDTipoEstadoEn(AFND * p_afnd, int pos);
```

- Devuelve el tipo del estado que está en la posición indicada.
- Los estados están definidos como
    - #define INICIAL 0
    - #define FINAL 1
    - #define INICIAL_Y_FINAL 2
    - #define NORMAL 3

```
int AFNDIndiceDeEstado(AFND * p_afnd, char * nombre);
```

- Devuelve la posición del estado pasado por nombre, -1 si no lo encuentra. El primero es el 0

```
int AFNDIndiceDeSimbolo(AFND * p_afnd, char * nombre);
```

- Devuelve la posición en el alfabeto del símbolo indicado, -1 si no lo encuentra. El primero es el 0

```
int AFNDTransicionIndicesEstadoiSimboloEstadof(AFND * p_afnd, int i_e1, int i_s,
int i_e2)
```

- Devuelve 1 si hay transición directa desde el estado de índice i_e1, con el símbolo de índice i_s, al estado de índice i_e2. Devuelve 0 si tal transición no existe.

```
int AFNDCierreLTransicionIJ(AFND * p_afnd, int i, int j);
```

- Devuelve 1 si se puede ir con transiciones lambda, desde el estado de índice i, hasta el estado de índice j. Devuelve 0 si tal transición no existe.
- Esta función requiere haber calculado el cierre transitivo de las transiciones lambda.

```
int AFNDIndiceEstadoInicial(AFND * p_afnd);
```

- Devuelve el índice del estado inicial

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 -ansi -pedantic), o no ejecute correctamente será considerada como no entregada.
    - Nota: El fichero que se entrega, afnd.c, está ofuscado, y se compila con -Wall -ansi, pero no con -pedantic, ya que no respeta la indentación de un fichero .c normal.
- Cualquier práctica que deje "lagunas de memoria" será penalizada de manera explícita en función de la gravedad de esas lagunas.