Autómatas y Lenguajes Práctica 1

Grupo 10 – Andrés Calvente & Tomás Higuera

Descripción de la práctica

Esta práctica consiste en crear un **Autómata Finito No Determinista (AFND).** La característica más relevante de este autómata es que podemos estar en más de un estado en cualquier momento mientras procesamos una entrada. Para realizar la práctica hemos tenido que diseñar varios módulos los cuales explicaremos en el siguiente apartado.

Estructuras diseñadas

- ### Alfabeto

  Módulo que nos permite almacenar los símbolos que nuestro autómata permitirá procesar.

***char** simbolos*	Array a caracteres que contiene los símbolos permitidos.
int tamanio	Tamaño total de alfabeto, la cantidad de símbolos que contiene.
int last_elemento	Índice del ultimo símbolo insertado.
ORDEN ordenado	Enum que nos permite saber si el array está o no ordenado.

- ### Estado

  Módulo que contendrá los elementos propios de un estado como los vistos en teoría.

char* nombre	Nombre del propio estado
ESTADO tipoEstado	Enum que nos permite saber si el estado es de tipo INICIAL (->Q_n), NORMAL (Q_n) o final (Q_n*)

- ### Palabra

  Módulo que contiene la propia palabra a procesar por el AFND.

int countSimbolosActuales	Contador de los símbolos de todas las palabras generadas
char** simbolosActuales	Array que contiene los símbolos de la palabra actual
int posicionActual	Índice actual en el array de palabra

NOTA: Al insertar nuevos símbolos en una palabra, no borramos la anterior, si no que añadimos los símbolos insertados después de la primera generada.

Funcionamiento del algoritmo principal

```c
void AFNDProcesaEntrada(FILE* pf, AFND* afnd){

    int i, j, k;
    char* simboloActual = NULL;
    int indiceElemento;
    BOOL* nuevosActuales = NULL;
    int flag;

    if (pf == NULL || afnd == NULL || afnd->actuales == NULL)
        return;


    for (i = getIndiceActual(afnd->palabra); i < getCountSimbolosPalabraAct(afnd->palabra); i++){


        fprintf(pf, "ACTUALMENTE EN { ");                   /* Primera linea */

        for (j = 0; j < afnd->maxEstados; j++){         /* Imprimimos los nombres de los estados diferenciando si son FINAL o NO */
            if(afnd->actuales[j] == TRUE){
                imprimeEstado(pf, afnd->estados[j]);
                fprintf(pf, " ");
            }
        }

        fprintf(pf, "}\n");

        AFNDImprimeCadenaActual(pf, afnd);                      /* Imprimimos la cadena */
```

Primero inicializamos todas las variables que vayamos a usar en la función y realizamos un control de errores.

El primer bucle que tenemos y el que abarca prácticamente toda la función irá desde el índice donde nos hayamos quedado de nuestra palabra (recordamos que lo hemos planteado tal que, si insertamos los símbolos *1010,* los procesamos y después metemos *0101* para procesarlos nuestra cadena se guardará como *10100101*), hasta los símbolos totales de la palabra.

El segundo bucle de *j* pasará por todos los estados actuales e imprimirá todos aquellos que sean *TRUE.* Después imprimiremos la cadena de símbolos que nos quedan por procesar.

```c
        simboloActual = getSimboloActual(afnd->palabra);        /* Conseguimos el simbolo siguiente a ser procesar */

        nuevosActuales = (BOOL*) malloc(sizeof(BOOL) * afnd->maxEstados);
        for (k = 0; k < afnd->maxEstados; k++)
            nuevosActuales[k] = FALSE;

        for (j = 0; j < afnd->maxEstados; j++){ _____/* Vamos a transitar todos los estados actuales */ /* countEstadosActuales es nuevo */
            if(afnd->actuales[j] == TRUE){

                indiceElemento = buscarElemento(afnd->simbolos, simboloActual);
                flag = 0;

             /* Hacemos una OR logica con lo que tenemos y la siguiente transicion */

                for (k = 0; k < afnd->maxEstados; k++){
                    nuevosActuales[k] = nuevosActuales[k] || afnd->transiciones[j][indiceElemento][k];
                    if(nuevosActuales[k] == TRUE)
                        flag++;
                }
            }
        }
        free(afnd->actuales);
        afnd->actuales = nuevosActuales; /* Igualamos el array antiguo al nuevo */

        if(flag != 0){
            procesarSimbolo(afnd->palabra);
        }
        else{
            printf(">> LA CADENA INSERTADA NO SE PUEDE PROCESAR! (NO HAY POSIBLE ESTADO SIGUIENTE)\n");
            setIndiceActual(afnd->palabra, getCountSimbolosPalabraAct(afnd->palabra));
            return;
        }
```

Ahora tendremos que pasar de un estado a los demás posibles. Para ello deberemos de crear un nuevo array de tipo BOOL, inicializarlo entero a FALSE y por cada transición posible dentro de los estados en los que estamos, hacemos una OR de nuestro nuevo array con la transición posible. Después, liberamos el puntero de estado actual y lo asignamos al nuevo. Si *flag* es distinto de 0, procesamos el siguiente símbolo, si no, es que no hay ninguna transición posible y por tanto cortaremos el procesamiento de la cadena.

Este es el ultimo bucle de impresión, donde sacamos por pantalla el paso final del autómata. También controlamos que el autómata llegue a un estado final, imprimiendo por pantalla lo que corresponda.

Carpeta donde están las pruebas y cómo ejecutarlas

Al hacer *make all* se crearán dos directorios:

- *automatas/*:
  Autómatas que son los descritos más adelante. Para ejecutarlos:
  autómatas/./<ejecutable>
- *tests/:*
  Todos los tests que hemos hecho para probar la funcionalidad de cada módulo. Para ejecutarlos:
  make runtests

Autómatas y palabras utilizados

- Test del autómata proporcionado por el profesor.
  - [01011]
  - [01100]
- Test del autómata subido por el grupo 11.
  - [02011211210101]
  - [111111]
- Test del autómata hecho por nosotros.
  - [bbAddCddb]
- Test de fallos para autómatas hecho por nosotros.
  - [01a11]