INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

GARZÓN DOMÍNGUEZ GERARDO ISMAEL

2CV13

TEORÍA COMPUTACIONAL

PRÁCTICA 3:
AUTÓMATA FINITO DETERMINISTA

INTRODUCCIÓN

Un autómata finito determinista está definido como una 5-tupla: $M = (Q, \Sigma, \delta, q_0, F)$. [1]

Dónde:

Q : El conjunto finito de estados internos del autómata.
Σ : El alfabeto de entrada del autómata, toda cadena de entrada debe estar construida sobre este. Para que pueda ser procesada.
δ : La función de transición, una regla de asociación tal que: $\delta : Q \times \Sigma \to Q$. Una función que mapea un par (q_i, a), $q_i \in Q, a \in \Sigma$ a un elemento $q_j \in Q$.
$q_0 \in Q$: Un elemento perteneciente al conjunto de estados del autómata, se considera como el estado inicial del autómata, es decir desde este estado comenzaría su funcionamiento al procesar una cadena.
$F \subseteq Q$: Subconjunto del conjunto de estados internos del autómata. Este subconjunto representa los estados finales válidos del autómata, es decir que las cadenas que lleven al autómata a este estado como paso final, serán aceptadas. [1]

El modo de operación, de un autómata finito determinista es el siguiente:

El sistema se encuentra en el estado q_0 al que llamamos estado inicial, después, dada una cadena formada sobre el alfabeto de entrada Σ, se consume un símbolo de la cadena de manera secuencial, es decir desde el primer símbolo hasta el último de la secuencia, por cada símbolo consumido, el autómata tiene una transición a alguno de sus estados internos, estas transiciones están gobernadas por la función de transición δ. Si el autómata llega a un estado de aceptación $q_f \in F$ al consumir todos los símbolos de la cadena, entonces se acepta la cadena, en caso contrario, se rechaza. [1]

Una forma de notación de autómatas son los grafos de transición. Como sabemos, un grafo está definido como un par ordenado $G = (V, E)$, tal que V es el conjunto de vértices del grafo, y E es el conjunto de aristas que unen a los vértices, las aristas pueden ser representadas como pares no ordenados.

Para este caso en particular, si $M = (Q, \Sigma, \delta, q_0, F)$, un autómata finito determinista. Entonces su grafo de transición asociado G_M será un grafo dirigido que tendrá exactamente $|Q|$ vértices, cada uno etiquetado con un elemento $q_i \in Q$. Cada regla de transición del autómata $\delta(q_i, a) = q_j$, representa un vértice (q_i, q_j) del grafo, el cual será etiquetado como a, donde claramente $a \in \Sigma$. [1]
El vértice asociado con el estado q_0, se considera como el estado inicial, el cual es comúnmente representado con una arista entrante hacia sí mismo, la cual no está conectada a ningún otro vértice y no está etiquetada. [1]
Los vértices asociados con los elementos $q_f \in F$, (estados de aceptación), son los vértices finales, y generalmente se denotan con un doble círculo.[1] Como se mencionó anteriormente, si el autómata llega a un estado $q_f \in F$ tras consumir todos los símbolos de la cadena, entonces la cadena se acepta y es considerada válida.

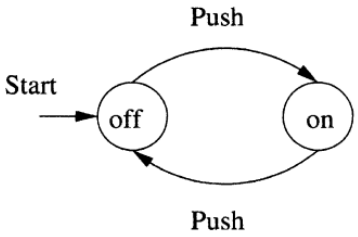
Nótese que un diagrama de transiciones se puede generar a partir de la tabulación de la función δ, como ya se menciono dado un par (q_i, a), $q_i \in Q, a \in \Sigma$ debe corresponder a solo un elemento (Para que este se considere autómata finito determinista) $q_j \in Q$. Así que dicha función δ, se puede tabular de la siguiente manera:

Estados / Símbolo	a
q_i	q_j

Figura 1.0 Forma tabular de una función de transición δ

La tabla se puede expandir para $|Q|$ renglones y $|\Sigma|$ columnas.

A continuación se muestra un ejemplo sencillo de un autómata finito determinista (AFD).



Observamos que este autómata modela el comportamiento de botón de encendido/apagado. En este ejemplo en particular, se podría considerar el estado “on”, como un estado de aceptación, pues en tal estado, el sistema podría entrar en funcionamiento [2], aunque tal cosa depende de la aplicación que se le de, este ejemplo es algo general en sí.

Figura 1.1 Diagrama de transición de un AFD que modela el comportamiento de un botón de encendido/apagado [2]

Como se puede observar, un autómata finito es bastante versátil para representar los posibles estados de algún sistema (siempre y cuando los estados del sistema sean finitos y bien definidos), es por ello que, los autómatas tienen un rango de aplicaciones amplio. Algunos ejemplos son:

- Software para diseñar y simular el comportamiento de circuitos digitales
- Analizadores léxicos para compiladores
- Software que escanea grandes cantidades de texto para analizar ocurrencias y patrones dentro del texto (expresiones regulares)
- Software que verifique sistemas con numero finito de estados, como protocolos de comunicación.

[2]

DESARROLLO E IMPLEMENTACIÓN

Para esta práctica fue necesario implementar 2 AFD's. Pero antes de llegar a esa parte, se mostrará cual es la idea detrás de la implementación del tipo abstracto de dato que representa un autómata.

Analizando la teoría detrás del AFD, se puede notar que hay bastantes maneras de implementarlo, por ejemplo, mediante grafos, los cuales podrían ser representados mediante listas multi-enlazadas, (para generalizar los autómatas finitos no deterministas), o representar la función de transición mediante un arreglo o tabla hash cuyos elementos están compuestos por una dupla (q_i, a), y una lista de elementos $q_j \in Q$.

Otra manera de representar un AFD es mediante funciones, donde cada función representa un vértice del grafo, en otras palabras, un estado, así, cada función recibe como mínimo dos argumentos; la cadena, y el índice actual de la cadena, y en base a estos argumentos, se llamará a otra función, la cual estará definida de la misma manera, en este caso, se tendría que llamar a la función de entrada primero. Notamos que algunas de las ventajas de implementar un AFD de esta manera es que, no usa espacio extra en memoria, y dependiendo de la complejidad del autómata, puede resultar mejor dividir la función δ, en múltiples funciones. Sin embargo algunas de las desventajas pueden ser que: No existe un tipo abstracto de dato que generalice el comportamiento de un autómata, y la funcionalidad del autómata debe ser distribuida a través de todas las funciones que componen a la función δ. Es por ello que en esta práctica, un autómata será implementado como un tipo abstracto de dato que generaliza el comportamiento de un AFD.

El tipo abstracto autómata, está definido como una estructura de 5 elementos, (para apegarse más a su definición matemática), sin embargo, la implementación es bastante mínima, ya que por ahora, algunos de los elementos que componen la estructura, no tienen uso dentro del código, como es el caso del elemento que representa el conjunto de estados internos del autómata. A continuación se muestra el fragmento de código que corresponde a la definición del autómata dentro del lenguaje C:

```
struct automata_t
{
    strset_t* alphabet;
    ty
    *states,
    *(*Delta)(ty* dom_range, const char* dom),
    *initial_state,
    *final_state;
};
```

Figura 2.0: La definición del autómata como una estructura en C.

"ty" es un macro que expande a "void", esto se hizo con el propósito de separar el autómata como tipo, de su funcionalidad, ya que, como se mencionó previamente, un autómata puede tener muchas aplicaciones, y es por esta razón que, los estados del autómata son considerados como algo abstracto y fuera del alcance de la implementación del tipo abstracto autómata, y es algo que se le deja a la implementación del programador.

Como se nota, el alfabeto de entrada del autómata es de tipo conjunto de cadenas, que en este caso se usa, pues es un tipo desarrollado dentro del curso. La función δ, es representada como un apuntador a función, que recibe como argumentos, un estado y un apuntador a char, tal que al de-referenciar, se obtiene el símbolo actual de la cadena, (asumiendo símbolos de 1 byte).

```
ty* determine_state(automata_t* automata, str_t* expr)
{
    ty* curr_state = automata->initial_state;
    const char* curr_symbol = NULL;

    for(size_t i=0, n=str_len(expr); i<n; i++)
    {
        curr_symbol = str_arr(expr)+i;
        curr_state = automata->Delta(curr_state, curr_symbol);
    }
    automata->Delta(curr_state, curr_symbol);

    return curr_state;
}
```

Figura 2.1: Modo de operación general de un autómata, regresará el estado al que llega con cadena.

La función de la figura 2.1 generaliza el modo de operación de un AFD, por cada símbolo de una cadena dada, llamará a la función delta, con los argumentos, estado actual y símbolo actual, se espera que la función delta, mapee el par a un nuevo estado, para repetir en la siguiente iteración. La primera iteración enviará como argumento el estado inicial del autómata, (proveído por el programador).

La función 2.1 puede ser envuelta en otra función que cheque si el estado al que llega es el estado final proveído por el programador. (Por ahora se asumió en la implementación que el autómata solo tiene un estado final, probablemente cambie después).

Antes de probar la implementación con los ejemplos de las lecturas de clase, describirá brevemente como se implementa las función delta. En sí, solo es una serie de condiciones if, else-if, para saber que estado se recibió como argumento, cuando se detecta el argumento en alguna condición, esta pasa a decidir que estado regresar en base a un switch sobre símbolo recibido, comparado con los miembros del alfabeto binario.

Al ejecutar el programa, es necesario introducir algún código, los códigos son los siguientes:
“1” para probar el autómata 1, se esperará que se introduzca una cadena que será validada para saber si se procede a probarla, lo mismo ocurre con el código “2”. Para salir se debe introducir la letra “q”.

Los autómatas implementados son los siguientes (según su función de transición):

Estados/símbolos	0	1
$\rightarrow * q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Figura 2.2: Tabulación de la función de transición del autómata 1 de las lecturas de clase.

Estados/símbolos	0	1
$\rightarrow q_0$	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	q_4
$*q_4$	q_1	q_2

Figura 2.3: Tabulación de la función de transición del autómata 2 de las lecturas de clase.

A continuación se muestran dos casos para cada autómata, uno donde la cadena se acepta, pues se llega a un estado de aceptación, y otro donde no ocurre esto.

```
1: automata1, 2: automata2, q: quitar
>> 1
Introduce cadena valida sobre: {"0", "1", }
... 1100
>>(estado 0)>>(estado 1)>>(estado 0)>>(estado 2)>>(estado 0)
Se llega a estado de aceptacion
>>
```

Figura 2.4: Prueba sobre el autómata 1 con una cadena válida.

```
>> 1
Introduce cadena valida sobre: {"0", "1", }
... 101
>>(estado 0)>>(estado 1)>>(estado 3)>>(estado 2)
No se llega a estado de aceptacion
```

Figura 2.5: Prueba sobre el autómata 1 con una cadena inválida.

```
>> 2
Introduce cadena valida sobre: {"0", "1", }
... 011
>>(estado 0)>>(estado 1)>>(estado 3)>>(estado 4)
Se llega a estado de aceptacion
```

Figura 2.6: Prueba sobre el autómata 2 con una cadena válida.

```
>> 2
Introduce cadena valida sobre: {"0", "1", }
... 0010
>>(estado 0)>>(estado 1)>>(estado 1)>>(estado 3)>>(estado 1)
No se llega a estado de aceptacion
```

Figura 2.7: Prueba sobre el autómata 2 con una cadena inválida.

Como se puede observar, el autómata imprime cada estado al que pasa, de esta manera también es comprobable el funcionamiento del programa.

CONCLUSIÓN

Al aprender el concepto de autómata finito es posible darse cuenta de la cantidad de aplicaciones que puede tener, ya que este permite modelar sistemas cuyos posibles estados están bien definidos y son finitos. Por lo aprendido, y según mi perspectiva, el poder de un autómata finito determinista, radica en la abstracción de que es lo que representa cada estado interno del mismo, sin contar tal vez la abstracción del lenguaje, ya que un lenguaje podría incluso se definido de manera distinta a simples caracteres, es por esa razón que el autómata fue implementado de tal manera; para que un estado represente la funcionalidad de la aplicación que se le de al autómata y de cierta manera separar el modo de operación de un autómata de la funcionalidad que este tiene.

Algunos ejemplos que se me ocurren sobre la aplicaciones de un AFD son, en el desarrollo de videojuegos, para determinar si una secuencia de pulsaciones de teclas es una acción válida o no. Otro caso podría ser para comprobar la validez de una secuencia de palabras en una gramática, aunque eso corresponde a un curso superior.

Esta práctica no fue de mayor problema respecto a la implementación, sin contar el hecho de que esta vez opté por una implementación bastante mínima, sobre el tema estudiado.

En general el aprendizaje que obtuve sobre esta práctica, fue más teórico y sobre como los autómatas son una parte fundamental del software y la computación, debido a sus múltiples aplicaciones.

REFERENCIAS

[1] P. Linz, An Introduction to Formal Languages and Automata, 6th ed. Davis, Davis, California: Jones & Bartlett, 2017, pp. 39-40.

[2] J. E. Hopcroft, Introduction to Automata Theory, Languages, and Computation, 3rd ed. Ithaca NY and Stanford CA: Prentice Hall (2006), pp. 2-4.