REPORTE DE PRÁCTICA NO. 0

BIBLIOTECA DIGITAL

ALUMNO: HERNANDEZ CARLOS MAYTE ERIDANI

Dr. Eduardo Cornejo-Velázquez

Lenguajes Formales

- **ALFABETO:** Un alfabeto es un conjunto finito de símbolos utilizados para formar palabras. Por ejemplo, el alfabeto binario es 0,1, mientras que el alfabeto de letras mayúsculas en inglés es A, B, C, ..., Z.
- **PALABRAS O CADENAS:** Una palabra o cadena es una secuencia finita y ordenada de símbolos de un alfabeto. Por ejemplo, en el alfabeto binario 0,1, la cadena "1011" es una palabra válida. En el alfabeto de letras, "CHAT" es una palabra conformada por el alfabeto inglés.
- **CADENA VACÍA:** La cadena vacía es una cadena de longitud 0 que no contiene ningún símbolo. Se representa comúnmente como "". Por ejemplo, en el alfabeto binario 0,1, la cadena vacía es "". En el alfabeto de letras A, B, C, ..., Z, la cadena vacía sigue siendo "".
- **LONGITUD:** La longitud de una palabra es el número de símbolos que contiene. Por ejemplo, en la palabra "1011" su longitud es 4, y en la palabra "GATO", la longitud es 4 también.
- **APARICIONES:** Se refiere al número de veces que un símbolo aparece en una palabra. Por ejemplo, en la palabra "BANANA", la letra "A" aparece 3 veces. En la cadena binaria "1100101", el símbolo "1" aparece 4 veces.
- **ORDENACIÓN:** Se basa en la longitud de las palabras para establecer un orden, como ordenar "casa", "sol" y "elefante" de menor a mayor longitud: "sol" (3), "casa" (4) y "elefante" (8); o en números binarios: "1", "10", "110".
- **COMBINACIONES:** Cantidad de símbolos de nuestro alfabeto elevado a la cantidad de caracteres que se usarán para hacer cada combinación; por ejemplo, en el alfabeto binario 0,1 con longitud 3 se tienen 2 elevado a la potencia $3 = 8$ combinaciones ("000", "001", "010", etc.), y en el alfabeto decimal 0,1,2,3,4,5,6,7,8,9 con longitud 2 se tienen 10 elevado a la potencia $2 = 100$ combinaciones. CLAUSULA DE KLEENE: Puede incluir todos los caracteres del alfabeto y la cadena vacía.
- **CLAUSULA DE KLEENE:** Puede incluir todos los caracteres del alfabeto y la cadena vacía, como en el caso del alfabeto a, b, donde el cierre de Kleene genera cadenas como "", "a", "b", "aa", "ab", "ba", "bb", "aaa", y así sucesivamente, o en el alfabeto 0,1, donde se generan cadenas como "", "0", "1", "00", "01", "10", "11", etc.

Operaciones con Palabras, Lenguajes Formales II

CONCATENACIÓN: Resultado de juntar dos palabras. Por ejemplo, si tenemos `x1 = "hola"` y `x2 = "mundo"`, su concatenación es `"holamundo"`.
POTENCIA: La potencia n-ésima de una palabra es el resultado de concatenar esta palabra consigo misma n veces.
PROPIEDADES

- **PREFIJOS:** Se obtienen al considerar los caracteres y sus posiciones, empezando por la cadena vacía, de izquierda a derecha y tomando un carácter más en cada pasada.
- **SUFIJOS:** Se obtienen al considerar los caracteres y sus posiciones, empezando por la cadena vacía, de derecha a izquierda y tomando un carácter más en cada pasada.
- **SEGMENTOS:** Se analizan los prefijos desde cada posición y sin repetir (se considera la cadena vacía). Por ejemplo, si tenemos la palabra "1001", sus segmentos son: {`A, 1, 10, 100, 1001, 0, 00, 001, 01`}.

Definiciones adicionales

REVERSO: Misma palabra leída de derecha a izquierda.
PALABRAS CAPICÚA: Todas aquellas palabras cuyo reverso es igual a la palabra en sí.
LENGUAJE: Subconjunto de la cláusula de Kleene sobre un alfabeto, que puede ser finito o infinito.

Operaciones con Lenguajes y Aplicaciones Lenguajes Formales III

PRODUCTO: Concatenación de cada una de las palabras del primer lenguaje con cada una de las palabras del segundo lenguaje.
POTENCIA SOBRE LENGUAJE: Se concatenan cada una de las palabras del lenguaje con cada una de las palabras del propio lenguaje.
CIERRE: Unión de todas las potencias de un lenguaje desde L^0.
COCIENTE POR LA IZQUIERDA: Todas aquellas palabras obtenidas tras eliminar el símbolo o palabra de la cual aplicamos el cociente y están al inicio de cada palabra. L={recortar, revivir, comer} $(re)^{-1}$ L={cortar, vivir}
HOMOMORFISMO: Consiste en la asignación entre dos alfabetos en las cuales traducimos/definimos un carácter del primer alfabeto, el cual será equivalente a otro carácter del segundo lenguaje.

Descubre los Autómatas: El Corazón de la Computación

AUTÓMATA: Máquina abstracta que se utiliza para modelar y describir procesos de cálculo, está compuesta por estados y transiciones.
AUTÓMATA FINITO: Número finito de estados y procesan las entradas secuencialmente.
AUTÓMATA DE PILA: Utilizan una pila para procesar la información.
MÁQUINA DE TURING: Poseen una cinta infinita.

¿Qué es un Autómata Finito Determinista (AFD)?

AUTÓMATA FINITO DETERMINISTA (AFD): Un autómata finito se describe como una máquina de estados que procesa entradas, específicamente cadenas o palabras, secuencialmente símbolo por símbolo. Un DFA consta de cinco elementos:

- Q : Conjunto de estados.
- E : Alfabeto utilizado por el autómata.
- A : Función de transición que define posibles transiciones entre estados según los símbolos de entrada.
- q_0 : Estado inicial desde el que comienza el procesamiento.
- F : Conjunto de estados finales donde se produce la aceptación.

Características de los DFA

- El término "determinista" indica que cada estado tiene como máximo una transición para cada símbolo, lo que elimina la ambigüedad en las rutas de procesamiento.
- Si existen múltiples transiciones para el mismo símbolo desde un solo estado, deja de ser determinista.

Representando DFAs

Representación de la Tabla de Transición: Los DFA se pueden representar mediante tablas de transición, donde las filas corresponden a los estados y las columnas representan símbolos.
Grafos: Representa los estados mediante círculos y las transiciones mediante flechas dirigidas.

¿Qué es un Autómata Finito No Determinista (AFND)?

AUTÓMATA FINITO NO DETERMINISTA (NFA): Se define como una tupla que consta de cinco elementos: Q, E, δ, q_0 y F.

- Q representa el conjunto de estados en el autómata; estas son las diversas situaciones encontradas al procesar una cadena de entrada.
- E denota el alfabeto que contiene símbolos que el autómata aceptará como entradas.
- q_0 es el estado inicial desde el cual comienza cualquier procesamiento de cadena de entrada, indicado por una pequeña flecha.
- F significa el conjunto de estados finales; si el procesamiento termina en uno de estos estados, se acepta la cadena de entrada.

Diferencias entre las funciones de transición

En los DFA, cada estado tiene como máximo una transición para cada símbolo; sin embargo, los NFA pueden tener múltiples transiciones para un solo símbolo desde cualquier estado dado.

¿Qué es un Autómata con Transiciones Épsilon?

AUTÓMATA FINITO LAMBDA: Representa el siguiente nivel de abstracción donde cada AFND puede interpretarse como un AF . Lambda (λ) denota la cadena vacía o la ausencia de símbolos. Se representa como una cadena con longitud cero; no es equivalente a un conjunto vacío.

- **Ejemplo en programación:** En algunos lenguajes, λ corresponde a una cadena inicializada pero que no contiene caracteres.

Estructura y Funcionalidad

Un autómata lambda (λ-AF) tiene una estructura similar a la de otros autómatas finitos, pero se diferencia principalmente en su función de transición.

- La función de transición permite realizar transiciones utilizando símbolos de entrada y la cadena vacía (λ), lo que posibilita el movimiento entre estados sin consumir entrada.

Pattern Matching con Autómatas: Mejora tus Algoritmos

Comparación de Patrones

La coincidencia de patrones es una técnica crucial en la informática para identificar patrones específicos dentro de estructuras de datos más grandes, como texto o bases de datos. Por ejemplo, la búsqueda de palabras clave en Gmail o el uso de expresiones regulares para encontrar direcciones de correo electrónico dentro del texto.
Más allá de las aplicaciones de usuario, la comparación de patrones también es vital en campos como la biología para identificar de manera eficiente secuencias genéticas importantes.

Algoritmos para la Búsqueda de Patrones

El primer algoritmo introducido es el algoritmo Naive (también conocido como algoritmo Knife), que busca patrones específicos dentro de un texto binario.
Para implementar el algoritmo Naive, construimos un autómata de árbol de prefijos a partir de un conjunto de patrones. Este autómata consta de estados que representan todos los prefijos encontrados dentro del conjunto de patrones y utiliza transiciones basadas en símbolos de entrada.

Función de Transición y Representación del Estado

- El estado inicial representa una cadena vacía; los estados finales corresponden a palabras completas del conjunto de patrones.
- Las transiciones entre estados se definen en función de la concatenación de símbolos con prefijos existentes. Si no existe una transición válida para determinadas entradas, esto indica que esas combinaciones no coinciden con ningún prefijo.
- El lenguaje por la derecha de un estado q incluye todas las cadenas procesadas desde ese estado que dan como resultado alcanzar un estado final.

Clases de Equivalencia en Autómatas y Lenguajes Formales

Relaciones de Equivalencia en la Teoría de Conjuntos

En la teoría de conjuntos, una relación de equivalencia en el conjunto A debe satisfacer tres condiciones: reflexividad, simetría y transitividad.

- Reflexividad: Cada elemento se relaciona consigo mismo.
- Simetría: Si un elemento se relaciona con otro, entonces ocurre lo contrario.
- Transitividad: Si un elemento se relaciona con un segundo que se relaciona con un tercero, entonces el primero debe relacionarse con el tercero.

Clases de Equivalencia

Dada una relación de equivalencia sobre los estados de un autómata, los elementos que comparten comportamientos similares forman clases de equivalencia.
La colección de estas clases define el conjunto cociente bajo la relación r, determinando su índice en función de cuántas clases distintas existen.

Lenguajes Regulares y sus Propiedades

Los autómatas finitos procesan lenguajes regulares, que se caracterizan por tener un número finito de clases de equivalencia representadas por estados finitos.
Dos cadenas pertenecen a la misma clase si sus resultados cocientes producen idiomas idénticos; por lo tanto, la regularidad depende de tener un número finito de tales clases.

Demostrar que un Lenguaje es Regular - Teorema de Myhill-Nerode

Primero se define como $a^i b^j$, donde tanto i como j son mayores que 0. Esto significa que el lenguaje incluye palabras con al menos una "a" seguida de al menos una "b", como "ab", "aab", "aabb", etc.
Para demostrar que el lenguaje es regular, aplicamos la operación cociente sobre los símbolos del lenguaje de forma iterativa hasta que no surjan nuevos lenguajes.
Comenzando con el cociente de 'a' sobre L, si eliminamos todas las palabras que contienen más de una 'a', aún conservamos cierta estructura en L.

- Si las palabras originales contienen solo una "a", la aplicación del cociente da como resultado al menos una "b", lo que da lugar a un nuevo idioma $L \cup BA^+$.
- El lenguaje resultante debe incluir al menos una "b", pero puede tener un número ilimitado de ellas debido a las propiedades de cierre.

Otras Aplicaciones de los Cocientes

- Al aplicar el cociente de 'b' sobre L, dado que todas las palabras comienzan con **"a"**, esto da como resultado un conjunto vacío, denotado como L'_2.

Demostrar que un Lenguaje NO es Regular - Teorema de Myhill-Nerode

A diferencia de demostrar que un lenguaje es regular (lo que requiere mostrar un número finito de clases de equivalencia), demostrar que un lenguaje no es regular implica encontrar una cantidad infinita de clases de equivalencia.
Para este ejemplo, se establece que L no es regular porque ningún mecanismo finito puede asegurar un recuento igual de ceros y unos para todas las longitudes posibles.

Demostración de la No Regularidad

Para demostrar la no regularidad, se debe encontrar una secuencia infinita en el cierre sobre el alfabeto que genere palabras dentro de L y definir un nuevo conjunto A, que consiste en secuencias que comienzan con al menos un cero.

- La primera palabra del conjunto A es "0". Al aplicar la operación de cociente sobre esta palabra con respecto a L, el resultado es "1".
- Continuando con la siguiente palabra "00", al aplicar su cociente se llega a "111", lo que indica que las palabras anteriores pueden no permanecer en el cociente después de operaciones posteriores.
- Este proceso puede continuar indefinidamente; por lo tanto, cada palabra del conjunto A produce cocientes únicos en el lenguaje L.

Minimización de Estados de un Autómata Explicada Desde Cero

Tabla De Transiciones

		a	b
1	q0	q1	q5
2	q1	q2	q3
3	q2	q2	q3
4	q3	q4	q6
5	q4	q4	q7
6	q5	q5	q6

Figure 1: Tabla de estados minimizados

Para reducir los estados, se deben cumplir dos requisitos clave:

Accesibilidad

Cada estado debe ser accesible desde algún estado inicial. Por ejemplo, los estados q_7 y q_8 no son accesibles porque no hay transiciones entrantes.
Dado que q_7 y q_8 no contribuyen al procesamiento de ningún lenguaje debido a su aislamiento, pueden eliminarse del autómata.

Completitud

El segundo requisito es que el autómata debe estar completo; es decir, cada estado debe tener una transición definida para cada símbolo de su alfabeto (en este caso, 'a' y 'b').
Una vez satisfechos ambos requisitos (accesibilidad y completitud), podemos comenzar a aplicar el algoritmo de reducción.

Proceso de Minimización

El algoritmo implica dividir los estados en grupos según comportamientos similares. Cada iteración perfecciona estas particiones respetando dos principios fundamentales:

- No podemos tener menos clases que en iteraciones anteriores.
- Los estados que pertenecen a diferentes clases no pueden volver a fusionarse en una sola clase durante el proceso.

Primera Iteración de Partición

En la primera iteración de partición, los estados se agrupan en función de si son finales o no finales:

- Estados finales: q_2, q_4, q_6.
- Estados no finales: q_0, q_1, q_3, q_5.

Esta distinción es fundamental porque los estados finales se comportan de manera diferente: indican la aceptación de palabras procesadas en comparación con los estados no finales.

EVIDENCIAS FOTOGRÁFICAS

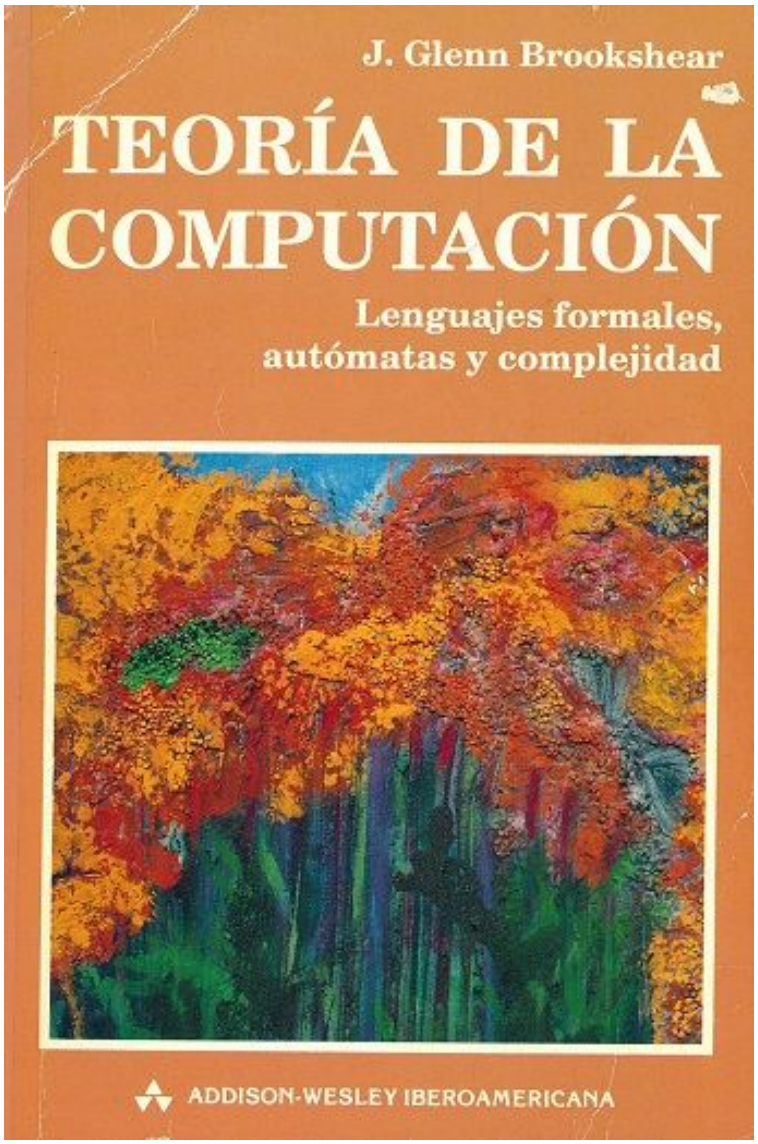

Figure 2: Teoría de la computación lenguajes formales, autómatas y complejidad

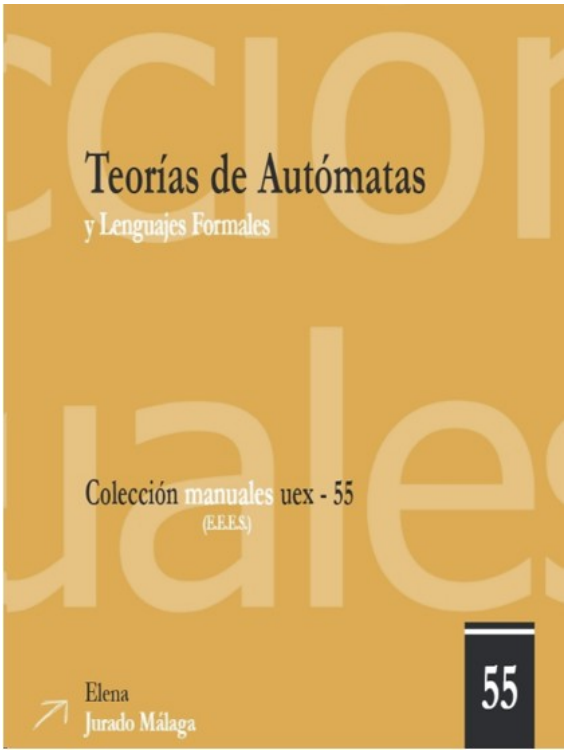

Figure 3: Teoría de autómatas y lenguajes formales

Figure 4: Lenguajes formales y teoría de autómatas

Preguntas y Respuestas sobre Lenguajes Formales y Autómatas

1. ¿Qué es la cláusula de Kleene y qué elementos puede incluir?

Respuesta: La cláusula de Kleene es un operador que permite incluir todos los caracteres de un alfabeto junto con la cadena vacía.

2. ¿Cuál es la diferencia entre prefijos y sufijos en las operaciones con palabras?

Respuesta:

- **Prefijos:** Se obtienen considerando los caracteres y posiciones de una palabra de **izquierda a derecha**, agregando un carácter más en cada paso.
- **Sufijos:** Se obtienen considerando los caracteres y posiciones de una palabra de **derecha a izquierda**, agregando un carácter más en cada paso.

3. ¿Cuáles son los cinco elementos de un Autómata Finito Determinista (AFD)?

Respuesta: Un **AFD** consta de cinco elementos:

1. **Q:** Conjunto de estados.
2. **E:** Alfabeto utilizado por el autómata.
3. **A:** Función de transición que define las transiciones entre estados según los símbolos de entrada.
4. $\mathbf{q_0}$**:** Estado inicial desde el que comienza el procesamiento.
5. **F:** Conjunto de estados finales donde se produce la aceptación.

4. ¿Cuál es la principal diferencia entre un DFA y un NFA en términos de transiciones?

Respuesta:

- En un **DFA**, cada estado tiene **una única transición** para cada símbolo del alfabeto.
- En un **NFA**, un estado puede tener **múltiples transiciones** para un mismo símbolo, permitiendo mayor flexibilidad en el procesamiento.

5. ¿Cómo se puede representar un DFA gráficamente?

Respuesta: Un **DFA** se puede representar mediante:

- **Tablas de transición:** donde las filas representan estados y las columnas representan símbolos del alfabeto.
- **Grafos dirigidos:** donde los estados se representan como **círculos** y las transiciones como **flechas** entre ellos.

Referencias Bibliográficas

References

[1] Brookshear, J. Glenn. *Teoría de La Computación Lenguajes Formales, Autómatas y Complejidad.* México: Pearson, 1999. Print.

[2] García, Pedro, colab. *Teoría de Autómatas y Lenguajes Formales.* México, D. F: Alfaomega, 2001. Print.

[3] Giró, Juan, et al. *Lenguajes Formales y Teoría de Autómatas.* Buenos Aires, Argentina: Alfaomega, 2015. Print.