Universidad Autónoma del Estado de Hidalgo

REPORTE DE PRÁCTICA NO. 0

LENGUAJES FORMALES

ALUMNO: Axel Aldahir Gutiérrez Gómez
Dr. Eduardo Cornejo-Velázquez

1. Introducción

En esta practica desarrollaremos el contendio aprendido a partir de visualizar la lista de reproducción de 13 videos compartidos por el profesor a través de la plataforma garza, haremos un resumén además de una serie de preguntas resaltando los puntos más interesantes

2. Herramientas empleadas

Elementos utilizados en la practica

1. Youtube utilizado para ver los videos
2. Overleafpara la edición del texto
3. Libros de la biblioteca universitaria en la UAEH

3. Desarrollo

Análisis de requisitos

Los requisitos para la practica son desarollar un resumén para cada uno de los videos de la lista de reproducción compartida por el profesor, además de realizar al menos 5 preguntas enfocadas a cada video para poder aclara los puntos más importantes y las dudas que nos genere.

Video 1: Lenguajes formales desde CERO — Palabra, alfabeto y clausura de Kleene

Un lenguaje esta basado en un alfabeto, un alfabeto es un conjunto finito de símbolos, representado con el símbolo Σ
El primer paso para los lenguajes formales es establecer un alfabeto basado en nuestras necesidades, un ejemplo seria:
Σ español = abcdefghijklmnñopqrstuvwxyz
Una vez definido el alfabeto vamos al segundo paso, que es formar palabras o cadenas, son secuencias finitas de símbolos de un alfabeto.
Pueden existir las cadenas vacías, no formadas por ningún símbolo, puede ser lambda
Propiedades de las palabras:

- Longitud de las palabras calculando el número de elementos
- Apariciones, indicando cuantas veces se repite un símbolo dentro de la palabra
- Ordenación, las palabras más grandes son las que van antes, y si son del mismo tamaño se define por el lugar de las letras en el alfabeto, las que estén antes van primero
- Combinaciones, la longitud cuando se eleva a 1 es el número de palabras, la combinatoria se determina con el número al que se eleva el alfabeto, entre mayor más combinaciones, y también puede ser 0, eso nos da como resultado las cadenas vacías, y todas las combinaciones se pueden englobar en la clausura de Klee, que se representa elevando el símbolo alfa Σ con un asterisco

Preguntas:
1. ¿Qué es un alfabeto en lenguajes formales? **Respuesta:** Es un conjunto finito de símbolos utilizados para formar palabras o cadenas.
2. ¿Qué es una palabra en un lenguaje formal? **Respuesta:** Es una secuencia finita de símbolos pertenecientes a un alfabeto.
3. ¿Cuál es la diferencia entre una palabra y una cadena vacía? **Respuesta:** Una palabra tiene al menos un símbolo, mientras que la cadena vacía no tiene ninguno y se representa como lambda (λ).
4. ¿Qué es la clausura de Kleene? **Respuesta:** Es el conjunto de todas las cadenas posibles, incluyendo la vacía, que se pueden formar con un alfabeto dado.
5. ¿Cómo se calcula la longitud de una palabra? **Respuesta:** Contando el número de símbolos que la componen.

Video 2 Operadores con palabras, lenguajes formales II

Operaciones con lenguajes formales
Concatenación, es la cantidad de símbolos de la palabra 1 que se combinan con la palabra dos para poder formar una palabra conjunta entre ambos, se representaría como m símbolos más n símbolos
Potencia, la potencia n-ésima de una palabra es el resultado de concatenar esta palabra consigo misma n veces
Prefijo, sufijos y segmentos, dadas dos palabras de un segmento x e y, siendo y un segmento de x si existen las variables u y v antes y después de y, como en la siguiente forma:
x = u * y * v
Si u es igual a lambda entonces y es un prefijo de x, y si v es igual a lambda entonces y es un sufijo de x.

Reverso: Se interpreta que para palabras se eleva al reverso, simplemente es escribir la palabra del lenguaje de froma contraria.
Lenguaje: es un subconjunto del alfabeto, se define que es lo que solicitamos, puede ser finito o infinito, además también etá el conjunto vacío
Preguntas:
1. ¿Qué es la concatenación de palabras? **Respuesta:** Es la combinación de dos palabras para formar una nueva, agregando los símbolos de una después de la otra.
2. ¿Qué es la potencia de una palabra? **Respuesta:** Es la repetición de la palabra consigo misma n veces.
3. ¿Qué es un prefijo de una palabra? **Respuesta:** Es una subcadena inicial de la palabra.
4. ¿Qué es el reverso de una palabra? **Respuesta:** Es la misma palabra escrita en orden inverso.
5. ¿Qué es un lenguaje formal? **Respuesta:** Es un subconjunto del conjunto de todas las posibles palabras formadas con un alfabeto.

Video 3 Operaciones con lenguajes y aplicaciones, lenguajes formales III

Podemos aplicar las operaciones booleanas de conjuntos.
Los autómatas y compiladores son fundamentales en la teoría de la computación, pues permiten modelar y procesar lenguajes formales utilizados en el desarrollo de software y en la optimización de procesos computacionales. Los autómatas finitos deterministas (AFD) y no deterministas (AFND) son estructuras matemáticas utilizadas para reconocer lenguajes regulares, mientras que los autómatas de pila se emplean en lenguajes libres de contexto. Los compiladores, por otro lado, hacen uso de estas estructuras para analizar, traducir y optimizar programas escritos en lenguajes de alto nivel, convirtiéndolos en código ejecutable mediante técnicas como análisis léxico, sintáctico y semántico.
Las operaciones con lenguajes incluyen la unión, concatenación, intersección y complemento, las cuales permiten manipular y transformar conjuntos de cadenas definidas por expresiones regulares o gramáticas formales. Estas operaciones son esenciales en la construcción de analizadores léxicos y sintácticos dentro de los compiladores, facilitando la detección de errores y la optimización del código fuente. Además, la conversión entre diferentes modelos de reconocimiento de lenguaje, como la transformación de expresiones regulares en autómatas finitos o de gramáticas en autómatas de pila, juega un papel clave en la implementación de parsers eficientes en compiladores modernos.
Las aplicaciones de los autómatas y compiladores abarcan diversas áreas como el diseño de lenguajes de programación, la verificación formal de sistemas y la inteligencia artificial. En el ámbito del procesamiento del lenguaje natural, los autómatas permiten el reconocimiento y análisis de estructuras lingüísticas, mientras que en la ciberseguridad se utilizan para la detección de patrones maliciosos en el tráfico de red. Asimismo, los compiladores modernos implementan optimizaciones avanzadas basadas en teoría de autómatas para mejorar el rendimiento y la portabilidad del software, permitiendo el desarrollo de sistemas más eficientes y seguros en distintos entornos computacionales.
Preguntas:
1. ¿Cuáles son las principales operaciones con lenguajes? **Respuesta:** Unión, concatenación, intersección y complemento.
2. ¿Qué es un autómata finito determinista (AFD)? **Respuesta:** Es un modelo matemático de cómputo que acepta o rechaza cadenas mediante estados finitos con transiciones únicas por símbolo.
3. ¿Cómo se usa un autómata en un compilador? **Respuesta:** Se usa en el análisis léxico y sintáctico para reconocer y estructurar el código fuente.
4. ¿Qué es un autómata de pila? **Respuesta:** Es un autómata que utiliza una pila para manejar lenguajes libres de contexto.
5. ¿En qué áreas se aplican los autómatas y compiladores? **Respuesta:** En diseño de lenguajes de programación, verificación de modelos, inteligencia artificial y seguridad informática.

Video 4 Descubre los autómatas: el corazón de la computación

Un automata es una maquina abstracta que se utiliza para modelar y describir procesos de calculo, compuesta de estados y transiciones entre ellos que le indican al autolmata como reaccionar, recibe inputs y devuelve una respuesta, se utiliza la estructura de los grafos, los nodos son los estados, mientras los arcos las transiciones.

Tenemos automatas finitos, número finito de estados, ideales para reconocer textos, autómatas de fpila, ideales para procesar su información, se usan para analisis sintactico de lenguajes de programación, maquinas de turing, son cómo supercomputadoras, pueden resolver cualquier problema que pueda ser resuelto por un ordenador.
Preguntas:
1. ¿Qué es un autómata? **Respuesta:** Es una máquina abstracta que procesa entradas y cambia de estado según reglas predefinidas.
2. ¿Cómo se representa gráficamente un autómata? **Respuesta:** Mediante un grafo donde los nodos son estados y las aristas son transiciones.
3. ¿Qué tipos de autómatas existen? **Respuesta:** Autómatas finitos, autómatas de pila y máquinas de Turing.
4. ¿Para qué se usan los autómatas finitos? **Respuesta:** Para el reconocimiento de patrones y procesamiento de texto.
5. ¿Cómo se diferencia una máquina de Turing de un autómata finito? **Respuesta:** Una máquina de Turing tiene una memoria infinita y puede resolver problemas más complejos.

Video 5: Qué es un automata finito determinista (AFD)

Es una maquina abstracta compuesta por 5 elementos, Q es el conjunto de estados, Σ es el alfabeto del automata, delta es una función de transición que recoge todas las transiciones del automata, q0 es el estado incial de cualquier cadena, y F es un subconjunto de Q que se refiere a los estados finales. El automata es finito y para cada estado puede existir como mucho una transición por cada símbolo. Se representan mediante tablas de transiciones, colocamos cómo filas los estados del automata y cómo columnas los símbolos con los que trabaja, y se pone en cada fila el resultado de a dónde nos lleva seguuir el camino que muestra un simbolo desde el nodo hasta el siguiente nodo con el que tenga contacto, podría decirse que muestra el resultado de trasladarse desde un nodo a otro por un camino con un símbolo, si no cuentan con un camino a otra ubucación con el símbolo se marca vacio, , se pone una flecha a la derecha indicando cuál es el nodo del que parten y una felcha hacia la izquierda indicando cuales son los nodos finales. Se representan con grafos.
Preguntas:
1. ¿Cuáles son los cinco elementos de un AFD? **Respuesta:** Un conjunto de estados (Q), un alfabeto (Σ), una función de transición (δ), un estado inicial (q_0) y un conjunto de estados finales (F).
2. ¿Cómo se representa un AFD? **Respuesta:** Mediante una tabla de transiciones o un diagrama de estados.
3. ¿Qué condición debe cumplir un AFD en cuanto a las transiciones? **Respuesta:** Para cada estado y cada símbolo de entrada, debe haber una única transición posible.
4. ¿Cómo se determina si una cadena es aceptada por un AFD? **Respuesta:** Siguiendo las transiciones del autómata desde el estado inicial hasta un estado final.
5. ¿Cuál es la principal ventaja de un AFD frente a un AFND? **Respuesta:** Su ejecución es más eficiente porque no requiere evaluar múltiples caminos simultáneamente.

Video 6 Autómatas Finitos No Deterministas

Los autómatas finitos no deterministas (AFND) son un modelo teórico fundamental en la teoría de la computación y el diseño de compiladores. A diferencia de los autómatas finitos deterministas (AFD), los AFND permiten transiciones múltiples desde un mismo estado para un símbolo de entrada dado. Esto proporciona mayor flexibilidad en la representación de lenguajes regulares, aunque requiere técnicas adicionales para su simulación en implementaciones prácticas.
Las transiciones en un AFND pueden llevarse a cabo sin una única trayectoria definida, lo que significa que un autómata puede estar en múltiples estados simultáneamente. Esta característica es útil en la especificación y análisis de lenguajes formales, especialmente en la representación de expresiones regulares y en la construcción de analizadores léxicos. Sin embargo, para su implementación práctica, es común convertirlos en AFD mediante el algoritmo de subconjuntos.
Los AFND encuentran aplicaciones en diversas áreas de la informática, como la verificación de modelos, el reconocimiento de patrones y la inteligencia artificial. Aunque su implementación directa puede ser ineficiente

en algunos casos, su capacidad de expresión y facilidad de construcción los hacen herramientas valiosas en el diseño de lenguajes de programación y compiladores.
1. ¿Cuál es la diferencia fundamental entre un AFD y un AFND? **Respuesta:** Un AFD tiene una única transición por cada símbolo de entrada desde un estado, mientras que un AFND puede tener múltiples transiciones para un mismo símbolo o incluso transiciones epsilon.
2. ¿Cómo se define formalmente un AFND? **Respuesta:** Se define como una tupla $(Q, \Sigma, \delta, q_0, F)$, donde Q es el conjunto de estados, Σ el alfabeto, δ la función de transición, q_0 el estado inicial y F el conjunto de estados de aceptación.
3. ¿Por qué los AFND son útiles en el diseño de compiladores? **Respuesta:** Permiten representar de manera más compacta los lenguajes regulares y se utilizan en la construcción de analizadores léxicos.
4. ¿Cuál es la principal desventaja de los AFND en la implementación? **Respuesta:** La ejecución requiere evaluar múltiples caminos posibles simultáneamente, lo que puede ser ineficiente en hardware.
5. ¿Cómo se puede convertir un AFND en un AFD? **Respuesta:** Se usa el algoritmo de subconjuntos, que transforma los estados del AFND en conjuntos de estados en el AFD.

Video 7 Conversión de AFND a AFD

La conversión de un AFND a un AFD es un proceso clave en la teoría de autómatas, pues permite la implementación eficiente de reconocedores de lenguajes regulares. El método más utilizado para esta conversión es el algoritmo de subconjuntos, que consiste en construir un nuevo autómata determinista donde cada estado representa un conjunto de estados del AFND original.
El algoritmo de subconjuntos inicia con el estado inicial del AFD, que corresponde al conjunto de estados alcanzables desde el estado inicial del AFND mediante transiciones epsilon. Luego, para cada símbolo del alfabeto, se calcula el conjunto de estados alcanzables desde cada conjunto de estados existentes en el AFD. Este proceso se repite hasta que no se generen nuevos estados.
Una vez completado, el AFD resultante puede ser minimizado para reducir el número de estados, lo que mejora su eficiencia en aplicaciones prácticas. Aunque el número de estados en el AFD puede crecer exponencialmente en comparación con el AFND original, este método garantiza un autómata determinista que reconoce el mismo lenguaje.
La conversión de AFND a AFD es esencial en la implementación de analizadores léxicos en compiladores. Estos analizadores requieren un reconocimiento rápido y eficiente de tokens, lo que hace necesario el uso de AFD en lugar de AFND, ya que los AFD pueden evaluarse en tiempo constante para cada símbolo de entrada.
En la práctica, los generadores de analizadores léxicos como Lex utilizan este proceso de conversión para optimizar el rendimiento del reconocimiento de patrones en código fuente. De esta manera, los compiladores pueden procesar eficientemente el código de entrada antes de su análisis sintáctico y semántico.

Video 8 Autómatas de Transiciones Épsilon

Los autómatas de transiciones epsilon (AFND-) son una extensión de los AFND que permiten transiciones sin consumir símbolos de entrada. Estas transiciones epsilon facilitan la construcción y combinación de autómatas, especialmente en la representación de expresiones regulares y gramáticas libres de contexto.
Un AFND- puede moverse entre estados sin leer un símbolo, lo que permite una mayor flexibilidad en la especificación de lenguajes. Sin embargo, esto también introduce complejidad en su análisis y ejecución, ya que requiere el seguimiento de múltiples estados simultáneamente.
Para convertir un AFND- en un AFD o en un AFND sin transiciones epsilon, se debe calcular la cerradura epsilon de cada estado. La cerradura epsilon representa todos los estados alcanzables mediante transiciones epsilon desde un estado dado, lo que permite transformar el autómata en uno equivalente sin transiciones epsilon.
Los AFND- son ampliamente utilizados en la construcción de analizadores léxicos y en la representación de expresiones regulares. Su flexibilidad en la definición de lenguajes los hace herramientas valiosas en el diseño y optimización de compiladores.

Video 9 Conversión de un AFND con transiciones vacías a un AFD

Para convertir un AFND con transiciones vacías (AFND-) en un AFD, es necesario eliminar las transiciones epsilon utilizando la cerradura epsilon. Este proceso consiste en determinar todos los estados alcanzables desde cualquier estado dado sin consumir un símbolo de entrada.
El primer paso es calcular la cerradura epsilon de cada estado en el AFND-. Luego, se construye un nuevo autómata donde cada estado representa un conjunto de estados del original, similar al algoritmo de subconjuntos usado en la conversión de AFND a AFD.
A continuación, se definen las transiciones en el nuevo autómata. Para cada conjunto de estados en el AFD, se determina a qué conjuntos se puede llegar al leer cada símbolo del alfabeto, asegurándose de aplicar la cerradura epsilon.
Finalmente, los estados de aceptación en el AFD se determinan en función de si contienen al menos un estado de aceptación del AFND original. Este proceso garantiza que el nuevo autómata sea determinista y equivalente en comportamiento al original.
La eliminación de transiciones epsilon simplifica el análisis y la implementación de los autómatas en compiladores y otras aplicaciones, permitiendo un procesamiento más eficiente de cadenas de entrada.
1. ¿Qué son las transiciones epsilon? **Respuesta:** Son transiciones que permiten moverse entre estados sin consumir símbolos de entrada.
2. ¿Qué es la cerradura epsilon de un estado? **Respuesta:** Es el conjunto de estados alcanzables desde un estado dado solo mediante transiciones epsilon.
3. ¿Cuál es el primer paso para eliminar las transiciones epsilon? **Respuesta:** Calcular la cerradura epsilon de cada estado en el autómata original.
4. ¿Cómo se determinan los estados finales en el AFD resultante? **Respuesta:** Un estado en el AFD es final si contiene algún estado final del AFND original.
5. ¿Por qué es importante eliminar transiciones epsilon? **Respuesta:** Permite simplificar la ejecución del autómata y facilita su implementación en compiladores.

Video 10 Pattern Matching con Autómatas para Mejorar Algoritmos

El pattern matching es una técnica fundamental en informática para la búsqueda de patrones dentro de cadenas de texto. Los autómatas finitos son herramientas eficaces para este propósito, ya que permiten realizar búsquedas eficientes en tiempo lineal respecto al tamaño de la entrada.
Uno de los enfoques más conocidos para el pattern matching con autómatas es el algoritmo de Aho-Corasick. Este método construye un autómata de búsqueda basado en un conjunto de patrones, permitiendo encontrar múltiples coincidencias en una sola pasada del texto.
Otro método comúnmente usado es el algoritmo de Knuth-Morris-Pratt (KMP), que emplea un autómata parcial para realizar coincidencias de manera más eficiente, evitando comparaciones redundantes dentro del texto de búsqueda.
El uso de autómatas en pattern matching permite optimizar la detección de palabras clave en motores de búsqueda, detección de firmas en ciberseguridad y procesamiento de texto en lenguajes de programación.
Además, los autómatas pueden ser utilizados en la compresión de datos y en la inteligencia artificial para el reconocimiento de patrones complejos en secuencias de datos.
En resumen, el uso de autómatas en pattern matching mejora significativamente la eficiencia de los algoritmos, permitiendo una búsqueda rápida y efectiva en grandes volúmenes de datos.
1. ¿Qué es el pattern matching? **Respuesta:** Es la técnica de buscar patrones dentro de un texto de entrada.
2. ¿Cómo ayuda un autómata en pattern matching? **Respuesta:** Reduce la complejidad de la búsqueda a tiempo lineal respecto al tamaño de la entrada.
3. ¿Cuál es el propósito del algoritmo de Aho-Corasick? **Respuesta:** Permite buscar múltiples patrones simultáneamente en un texto.
4. ¿Qué ventaja tiene el algoritmo KMP sobre la búsqueda ingenua? **Respuesta:** Evita comparaciones redundantes al usar información previa de la cadena.
5. ¿En qué áreas se usa pattern matching con autómatas? **Respuesta:** En motores de búsqueda, ciberseguridad y procesamiento de texto.

Video 11 Clases de Equivalencia en Autómatas y Lenguajes Formales

Las clases de equivalencia en autómatas y lenguajes formales son fundamentales para el estudio de la minimización y clasificación de lenguajes regulares. Estas clases agrupan cadenas que son indistinguibles en términos de aceptación por un autómata dado.
Dos cadenas pertenecen a la misma clase de equivalencia si, al extenderlas con cualquier otra cadena, producen resultados equivalentes respecto a la aceptación por el autómata. Este concepto es clave en la prueba de que un lenguaje es o no regular.
El estudio de clases de equivalencia permite la construcción de autómatas minimizados y la demostración de propiedades fundamentales de los lenguajes formales, optimizando su análisis y representación en sistemas computacionales.
1. ¿Qué es una clase de equivalencia en autómatas? **Respuesta:** Es un conjunto de cadenas indistinguibles por el autómata.
2. ¿Cómo se usan las clases de equivalencia en la minimización de autómatas? **Respuesta:** Permiten identificar estados redundantes y combinarlos.
3. ¿Cómo se relacionan las clases de equivalencia con la regularidad de un lenguaje? **Respuesta:** Un lenguaje es regular si tiene un número finito de clases de equivalencia.
4. ¿Qué método se usa para encontrar clases de equivalencia en un autómata? **Respuesta:** El método de partición sucesiva.
5. ¿Por qué es útil estudiar clases de equivalencia en lenguajes formales? **Respuesta:** Facilita la prueba de regularidad y minimización de autómatas.

Video 12 Demostración de que un Lenguaje es Regular por el Teorema de Myhill-Nerode

El teorema de Myhill-Nerode establece una caracterización fundamental de los lenguajes regulares en términos de clases de equivalencia. Este teorema afirma que un lenguaje es regular si y solo si tiene un número finito de clases de equivalencia con respecto a la relación de indistinguibilidad.
Para demostrar que un lenguaje es regular mediante este teorema, se deben identificar las clases de equivalencia de las cadenas respecto al lenguaje dado. Si el número de estas clases es finito, entonces el lenguaje es regular y puede ser reconocido por un autómata finito determinista.
Este método proporciona una forma rigurosa y sistemática para probar la regularidad de un lenguaje, facilitando la construcción de autómatas minimizados y su implementación en compiladores y aplicaciones de procesamiento de texto.
1. ¿Qué establece el teorema de Myhill-Nerode? **Respuesta:** Un lenguaje es regular si y solo si tiene un número finito de clases de equivalencia.
2. ¿Cómo se usa este teorema para demostrar que un lenguaje es regular? **Respuesta:** Se identifica un número finito de clases de equivalencia y se construye un AFD.
3. ¿Cómo se usa para demostrar que un lenguaje no es regular? **Respuesta:** Se prueba que el número de clases de equivalencia es infinito.
4. ¿Qué relación tiene con la minimización de autómatas? **Respuesta:** Permite construir el autómata mínimo equivalente a un lenguaje dado.
5. ¿Por qué este teorema es más preciso que el lema del bombeo? **Respuesta:** Porque proporciona una caracterización exacta de los lenguajes regulares.

Video 13 Demostración de que un Lenguaje No es Regular por el Teorema de Myhill-Nerode

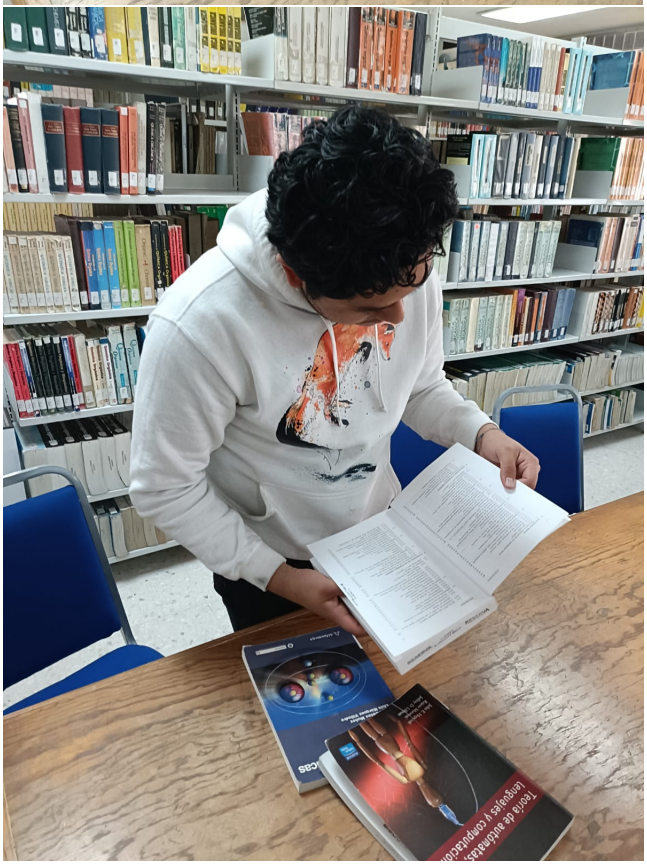
Para demostrar que un lenguaje no es regular utilizando el teorema de Myhill-Nerode, se busca mostrar que existen infinitas clases de equivalencia en la relación de indistinguibilidad de cadenas.
El proceso inicia identificando cadenas distintas que no pueden ser agrupadas en un número finito de clases sin afectar la aceptación del lenguaje. Si se demuestra que el número de clases es infinito, el lenguaje no puede ser reconocido por un autómata finito.

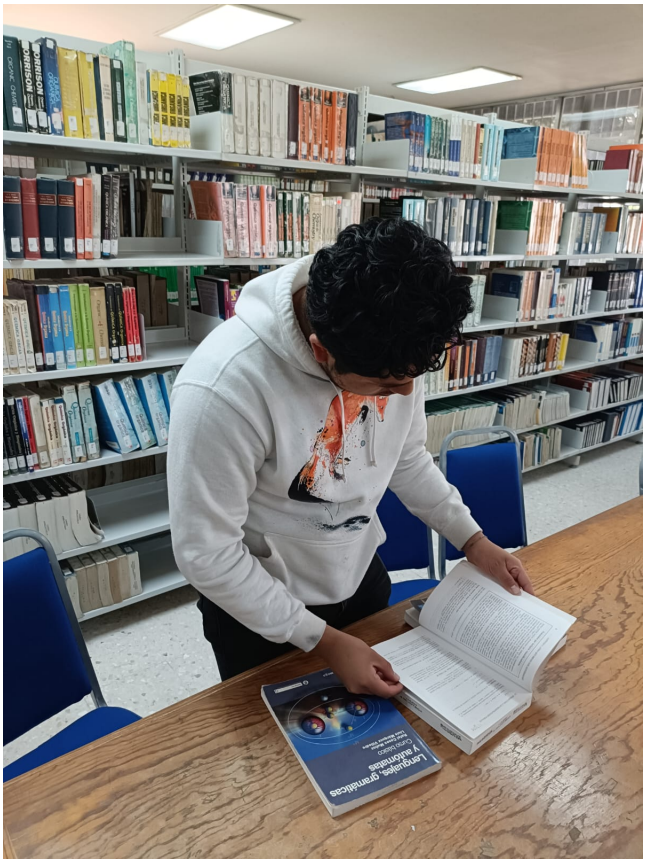
Este método es más formal y preciso que el lema del bombeo, permitiendo caracterizar con mayor claridad los límites de los lenguajes regulares.
Las demostraciones con este teorema son utilizadas en la teoría de la computación para clasificar lenguajes y determinar sus propiedades computacionales.

Video 14 Minimización de Estados de un Autómata Explicada desde Cero

La minimización de autómatas es un proceso clave para optimizar la representación de lenguajes regulares y mejorar la eficiencia en su implementación computacional.
El primer paso en la minimización consiste en eliminar los estados inaccesibles del autómata, es decir, aquellos que no pueden ser alcanzados desde el estado inicial.
Luego, se identifican y combinan estados equivalentes, es decir, aquellos que no pueden ser distinguidos por ninguna cadena de entrada. Esto se logra mediante la construcción de clases de equivalencia.
El algoritmo de minimización de estados, como el método de partición de Hopcroft, permite reducir el tamaño del autómata de manera óptima, garantizando una implementación más eficiente.
La minimización de autómatas es esencial en el diseño de analizadores léxicos en compiladores, donde un autómata más pequeño y eficiente mejora el rendimiento del reconocimiento de tokens.
Finalmente, la minimización de estados también tiene aplicaciones en áreas como la inteligencia artificial y la verificación formal de sistemas, donde se requiere optimización en el procesamiento de lenguajes formales.
1. ¿Cuál es el primer paso en la minimización de autómatas? **Respuesta:** Eliminar los estados inaccesibles.
2. ¿Cómo se agrupan los estados equivalentes? **Respuesta:** Se usa el método de partición sucesiva.
3. ¿Qué algoritmo se usa comúnmente para minimizar autómatas? **Respuesta:** El algoritmo de Hopcroft.
4. ¿Por qué es importante minimizar un autómata? **Respuesta:** Reduce el espacio de almacenamiento y mejora la eficiencia computacional.
5. ¿En qué aplicaciones se usa la minimización de autómatas? **Respuesta:** En analizadores léxicos, verificación formal y optimización de compiladores.

Fotos con los libros consultados




Langues et Grammaires
y Gramática

4. Conclusiones

Realmente los automatas y compiladores son un tema muy, muy extenso, y aunque no es tan complicado de comprender la parte derealizarlso realmente si es más compleja, el establecer relaciones que hagan que se llegue a un objetivo sin repetir patrones o equivocarse en ellos es complicado, imcluso desde el lenguaje formal es complicado analizar la idea incial que necesita ser cumplida al diseñar cada uno de ellos, por ahora lo veo más que nada cómo un desafio logico, y uno mucho más grande al momento de programarse, dare un repaso a los temás de los videos y practicare cómo realizar los automatas para poder entenderlos de mejor fomra, y en cuanto a la programación me es complicado entenderla en ocasiones, pero tratare de enfocarme en conocer su funcionamiento y aplicación.

Referencias Bibliográficas

References

[1] Grabowska, S.; Saniuk, S. (**2022**). Business models in the industry 4.0 environment—results of web of science bibliometric analysis. *J. Open Innov. Technol. Mark. Complex*, *8(1)*, 19.

[2] Codemath. (2023, 4 diciembre). *Operaciones con Palabras — Lenguajes Formales II [Vídeo]*. YouTube. https://www.youtube.com/watch?v=MXDl4Ts$_E$Z0

[3] Codemath. (2023b, diciembre 15). *Operaciones con Lenguajes y Aplicaciones — Lenguajes Formales III [Vídeo]*. YouTube. https://www.youtube.com/watch?v=uU-fNuwbmZg

[4] Codemath. (2024, 29 enero). *Descubre los autómatas: el corazón de la computación [Vídeo]*. YouTube. https://www.youtube.com/watch?v=pMIwci0kMv0

[5] Codemath. (2024b, febrero 4). *Qué es un Autómata Finito Determinista (AFD) [Vídeo]*. YouTube. https://www.youtube.com/watch?v=d9aEE-uLmNE

[6] Codemath. (2024c, abril 23). *Qué es un Autómata Finito No Determinista (AFND) [Vídeo]*. YouTube. https://www.youtube.com/watch?v=dIgKBNuaglE

[7] Codemath. (2024d, abril 29). *Convertir un Autómata NO Determinista (AFND) a Determinista (AFD) [Vídeo]*. YouTube. https://www.youtube.com/watch?v=hzJ8CNdPElc

[8] Codemath. (2024e, mayo 5). *Qué es un Autómata con Transiciones Epsilon [Vídeo]*. YouTube. https://www.youtube.com/watch?v=71P3daDZWlQ

[9] Codemath. (2024f, mayo 11). *Convertir un AFND con Transiciones a un AFND [Vídeo]*. YouTube. https://www.youtube.com/watch?v=1yKBT8gWN-Y

[10] Codemath. (2024g, mayo 27). *Pattern Matching con Autómatas: Mejora tus Algoritmos [Vídeo]*. YouTube. https://www.youtube.com/watch?v=22XqyZLhKPg

[11] Codemath. (2024h, junio 22). *Clases de Equivalencia en Autómatas y Lenguajes Formales [Vídeo]*. YouTube. https://www.youtube.com/watch?v=JuTuMe8Q58c

[12] Codemath. (2024i, julio 1). *Demostrar que un Lenguaje es Regular - Teorema de Myhill-Nerode [Vídeo]*. YouTube. https://www.youtube.com/watch?v=gYOvlrjRBwg

[13] Codemath. (2024j, julio 8). *Demostrar que un Lenguaje NO es Regular - Teorema de Myhill-Nerode [Vídeo]*. YouTube. https://www.youtube.com/watch?v=FPWpCq20g0o

[14] Codemath. (2025, 13 febrero). *Minimización de estados de un autómata explicada desde cero [Vídeo]*. YouTube. https://www.youtube.com/watch?v=gd6uyNXsqcw