UNIVERSIDAD AUTONOMA DEL ESTADO

DE HIDALGO

PRACTICA 0

ALUMNO: HERNÁNDEZ LARA JOSÉ MANUEL

CATEDRATICO: DR. CORNEJO VELÁZQUEZ

LICINCIATURA EN CIENCIAS COMPUTACIONALES

SEMESTRE: 6° “3”

AUTOMATAS Y COMPILADORES

Introducción

La serie de videos aborda los lenguajes formales, comenzando con conceptos clave como "palabra" y "alfabeto", y explicando la cláusula de Kleene y las operaciones con palabras y lenguajes. También se exploran aplicaciones de los lenguajes formales y los autómatas, considerados el "corazón de la computación". Entre los temas tratados están los autómatas finitos deterministas, no deterministas, y aquellos con transiciones epsilon. Estos conceptos son esenciales en teoría de la computación, ya que ayudan a modelar y entender cómo funcionan los algoritmos y las máquinas.

Los lenguajes formales son fundamentales en la teoría de la computación y otras áreas porque proporcionan una base matemática para describir y analizar el comportamiento de los sistemas de procesamiento de información. Son esenciales para el diseño de compiladores, la verificación de programas y la definición de gramáticas en lenguajes de programación. Al utilizar reglas estrictas para la construcción de cadenas de símbolos, los lenguajes formales permiten modelar procesos lógicos y matemáticos de manera precisa y predecible, facilitando el desarrollo de algoritmos y la creación de autómatas.

Marco teórico

Los lenguajes formales son sistemas que describen estructuras sintácticas con precisión matemática. Se componen de un alfabeto (símbolos) y palabras (combinaciones de esos símbolos). Los lenguajes formales son cruciales para describir el comportamiento de sistemas computacionales y son la base de muchas aplicaciones, como los compiladores y los autómatas.

Importancia en Autómatas

Los autómatas son modelos abstractos que reconocen lenguajes formales. Hay diferentes tipos de autómatas, como:

Autómatas finitos deterministas (DFA): Procesan secuencias de entrada y determinan si una palabra pertenece al lenguaje, con un comportamiento predecible.

Autómatas finitos no deterministas (NFA): Permiten múltiples transiciones, lo que los hace más flexibles aunque conceptualmente más complejos.

Autómatas con transiciones epsilon (ε-NFA): Estos permiten transiciones sin consumir entrada, lo que agrega aún más flexibilidad.

Los autómatas son fundamentales en el reconocimiento de patrones y en el diseño de sistemas de procesamiento de datos, como en el filtrado de texto, la verificación de protocolos y el análisis de redes.

Importancia en Compiladores

Los compiladores son programas que traducen código fuente de lenguajes de programación de alto nivel a código máquina. Utilizan gramáticas, que son un tipo de lenguaje formal, para definir las reglas sintácticas del lenguaje fuente. Los autómatas juegan un papel importante en esta etapa, ya que el análisis léxico (fase inicial del compilador) usa autómatas finitos para identificar las unidades léxicas o tokens del código fuente. Posteriormente, en el análisis sintáctico, se usan gramáticas formales (normalmente gramáticas libres de contexto) para validar la estructura del código.

Herramientas empleadas

Para el desarrollo de esta práctica se acudió a la biblioteca digital de la universidad para consultar en libros los temas relacionados con los lenguajes formales y los autómatas y compiladores.

Desarrollo

En base a las fuentes consultadas se definieron los siguientes conceptos:

Palabra: Una palabra en un lenguaje formal es una secuencia finita de símbolos elegidos de un alfabeto.

1. Por ejemplo, si el alfabeto es {a, b}, algunas palabras posibles son "a", "b", "ab", "ba", etc. Las palabras se pueden usar para formar cadenas de entrada para autómatas o para expresar operaciones en lenguajes de programación.

2. Otro ejemplo sería un alfabeto binario {0, 1}, donde las palabras pueden ser "0", "1", "01", "10", "110", etc. Estas palabras se utilizan en el procesamiento de cadenas de entrada para autómatas o en la construcción de operaciones dentro de lenguajes de programación y sistemas formales.

Alfabeto: Es el conjunto finito de símbolos que se utilizan para formar palabras.

1. Por ejemplo, el alfabeto binario es {0, 1}, y el alfabeto de los caracteres latinos es {a, b, c,..., z}. El alfabeto define los símbolos válidos que se pueden utilizar en las palabras de un lenguaje formal.
2. Otro ejemplo sería el alfabeto de los números decimales {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, usado en sistemas numéricos y aritméticos.

Cláusula de Kleene: También conocida como *estrella de Kleene*, es una operación que permite formar la secuencia de todas las posibles combinaciones de palabras que se pueden generar a partir de un alfabeto. Si AAA es un conjunto de palabras, la estrella de Kleene A∗A^*A∗ incluye todas las palabras posibles formadas por concatenación de palabras de AAA, incluidas la palabra vacía ϵ\epsilonϵ.

1. Por ejemplo, si el alfabeto es {a, b}, {a,b}∗\{a, b\}^*{a,b}∗ incluye ϵ\epsilonϵ, "a", "b", "aa", "ab", etc.
2. Por ejemplo, si el alfabeto es {0, 1}, {0,1}∗\{0, 1\}^*{0,1}∗ incluiría ϵ\epsilonϵ, "0", "1", "00", "01", "10", "11", etc.

Operaciones con palabras: Estas son las operaciones que se pueden realizar con las palabras generadas por un alfabeto. Algunas de las más comunes son:

- **Concatenación**: Unir dos palabras en una sola.
  - Por ejemplo, concatenar "ab" y "ba" da "abba".
  - Unir "hello" y "world" da "helloworld".
- **Reverso**: Invertir el orden de los símbolos en una palabra.
  - El reverso de "abc" es "cba".
  - El reverso de "hello" es "olleh".
- **Subsecuencia**: Extraer una subsecuencia de símbolos de una palabra.
  - Por ejemplo, en "abcde", "ace" es una subsecuencia.
  - En la palabra "programming", "prmg" es una subsecuencia.

Operaciones con lenguajes: Los lenguajes formales también pueden someterse a diversas operaciones:

- **Unión**: Combinar todos los elementos de dos lenguajes.
  - Si L1={a,b}L_1 = \{a, b\}L1={a,b} y L2={b,c}L_2 = \{b, c\}L2={b,c}, la unión es {a,b,c}\{a, b, c\}{a,b,c}.
  - Si L1={x,y,z}L_1 = \{x, y, z\}L1={x,y,z} y L2={y,z,w}L_2 = \{y, z, w\}L2 ={y,z,w}, la unión es {x,y,z,w}\{x, y, z, w\}{x,y,z,w}.
- **Intersección**: Formar un conjunto con los elementos que comparten dos lenguajes.
  - Si L1={a,b}L_1 = \{a, b\}L1={a,b} y L2={b,c}L_2 = \{b, c\}L2={b,c}, la intersección es {b}\{b\}{b}.
  - Si L1={1,2,3,4}L_1 = \{1, 2, 3, 4\}L1={1,2,3,4} y L2={3,4,5,6}L_2 = \{3, 4, 5, 6\}L2={3,4,5,6}, la intersección es {3,4}\{3, 4\}{3,4}.
- **Complemento**: Tomar todas las palabras que no pertenecen a un lenguaje dado.
  - Supongamos que el alfabeto es {a,b}\{a, b\}{a,b} y el lenguaje LLL es {a,ab}\{a, ab\}{a,ab}. El complemento de LLL con respecto al alfabeto sería el conjunto de todas las posibles palabras sobre el alfabeto que no están en LLL. Entonces, el complemento de LLL sería {ϵ,b,ba,bb,aab,abb,…}\{\epsilon, b, ba, bb, aab, abb, \ldots\}{ϵ,b,ba,bb,aab,abb,…}, donde ϵ\epsilonϵ es la palabra vacía.
  - Si el alfabeto es {0,1}\{0, 1\}{0,1} y el lenguaje LLL es {0,1,01}\{0, 1, 01\}{0,1,01}, el complemento de LLL con respecto al alfabeto incluye todas las posibles palabras sobre el alfabeto {0,1}\{0, 1\}{0,1} que no están en LLL. Por ejemplo, el complemento de LLL sería {ϵ,00,10,11,000,010,100,110,…}\{\epsilon, 00, 10, 11, 000, 010, 100, 110, \ldots\}{ϵ,00,10,11,000,010,100,110,…}.
- **Concatenación**: Similar a la concatenación de palabras, pero aplicado a lenguajes.
  - Si L1={a}L_1 = \{a\}L1={a} y L2={b,c}L_2 = \{b, c\}L2={b,c}, su concatenación es {ab,ac}\{ab, ac\}{ab,ac}.
  - Supongamos que L1={x,y}L_1 = \{x, y\}L1={x,y} y L2={1,2}L_2 = \{1, 2\}L2={1,2}. La concatenación de L1L_1L1 y L2L_2L2 es el conjunto de todas las posibles combinaciones donde cada palabra en L1L_1L1 se concatena con cada palabra en L2L_2L2. Por lo tanto, la concatenación L1·L2L_1 \cdot L_2L1·L2 es: {x1,x2,y1,y2}\{x1, x2, y1, y2\}{x1,x2,y1,y2}

- **Estrella de Kleene**: Genera todas las posibles secuencias de concatenaciones de palabras de un lenguaje.
  - Supongamos que el lenguaje LLL es {a,b}\{a, b\}{a,b}. La estrella de Kleene de LLL, denotada como L∗L^*L∗, incluye todas las posibles secuencias de concatenaciones de las palabras de LLL, incluyendo la palabra vacía ϵ\epsilonϵ. Así que:

L∗={ϵ,a,b,aa,ab,ba,bb,aaa,aab,aba,abb,baa,bab,bba,bbb,…}L^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, \ldots\}L∗={ϵ,a,b,aa,ab,ba,bb,aaa,aab,aba,abb,baa,bab,bba,bbb,…}

- Considera el lenguaje L={0}L = \{0\}L={0}. La estrella de Kleene de LLL, denotada como L∗L^*L∗, incluye todas las posibles secuencias de concatenaciones de la palabra 000, incluyendo la palabra vacía ϵ\epsilonϵ. Así que: L∗={ϵ,0,00,000,0000,…}L^* = \{\epsilon, 0, 00, 000, 0000, \ldots\}L∗={ϵ,0,00,000,0000,…}

Aplicaciones de lenguajes formales: Los lenguajes formales se utilizan ampliamente en la informática, por ejemplo, en el diseño de compiladores, que traducen código de alto nivel a lenguaje máquina, y en la definición de gramáticas que especifican las reglas para construir programas válidos. También son esenciales en la verificación formal, donde se asegura que los sistemas cumplen con sus especificaciones.

Autómatas: el corazón de la computación: Un autómata es un modelo abstracto de máquina que puede realizar cálculos en función de una secuencia de entradas. Los autómatas están en el corazón de la computación porque sirven para modelar y analizar sistemas que procesan datos o ejecutan algoritmos. Los lenguajes formales y los autómatas están íntimamente conectados, ya que los autómatas aceptan o rechazan palabras de un lenguaje dependiendo de las reglas establecidas.

Autómata Finito Determinista (DFA): Es un autómata en el que para cada estado y símbolo de entrada hay una transición única a un nuevo estado. Es una máquina muy predecible y eficiente. Los DFA son usados en tareas como la validación de patrones o la construcción de filtros de texto, ya que procesan una palabra de principio a fin sin ambigüedad.

Ejemplo 1: DFA para aceptar cadenas que terminan en "01"

- **Alfabeto:** {0,1}\{0, 1\}{0,1}
- **Estados:** {q0,q1,q2}\{q_0, q_1, q_2\}{q0,q1,q2}
- **Estado inicial:** q0q_0q0
- **Estado(s) de aceptación:** q2q_2q2
- **Transiciones:**
  - Desde q0q_0q0:
    - Con 000 va a q1q_1q1
    - Con 111 permanece en q0q_0q0
  - Desde q1q_1q1:
    - Con 000 permanece en q1q_1q1
    - Con 111 va a q2q_2q2
  - Desde q2q_2q2:
    - Con 000 va a q1q_1q1

        - Con 111 permanece en q2q_2q2

Este DFA acepta cadenas que terminan en "01". Por ejemplo, las cadenas "01", "101", "0001" son aceptadas.

Ejemplo 2: DFA para aceptar cadenas que contienen un número par de "a"s

- **Alfabeto:** {a,b}\{a, b\}{a,b}
- **Estados:** {q0,q1}\{q_0, q_1\}{q0,q1}
- **Estado inicial:** q0q_0q0
- **Estado(s) de aceptación:** q0q_0q0
- **Transiciones:**
    - Desde q0q_0q0:
        - Con aaa va a q1q_1q1
        - Con bbb permanece en q0q_0q0
    - Desde q1q_1q1:
        - Con aaa va a q0q_0q0
        - Con bbb permanece en q1q_1q1

Este DFA acepta cadenas con un número par de "a"s. Por ejemplo, las cadenas "bb", "ab", "aabb", "bbaab" son aceptadas.

Autómata Finito No Determinista (NFA): A diferencia del DFA, un NFA puede tener varias transiciones posibles desde un estado dado con un símbolo de entrada. Esto da lugar a múltiples caminos para una misma palabra. Aunque parece menos eficiente que un DFA, ambos tienen la misma capacidad computacional, ya que cualquier NFA puede transformarse en un DFA equivalente.

Ejemplo 1: NFA para aceptar cadenas que contienen "01"

- **Alfabeto:** {0,1}\{0, 1\}{0,1}
- **Estados:** {q0,q1,q2}\{q_0, q_1, q_2\}{q0,q1,q2}
- **Estado inicial:** q0q_0q0
- **Estado(s) de aceptación:** q2q_2q2
- **Transiciones:**
    - Desde q0q_0q0:
        - Con 000 va a q0q_0q0 o a q1q_1q1 (transición no determinista)
        - Con 111 permanece en q0q_0q0
    - Desde q1q_1q1:
        - Con 111 va a q2q_2q2
    - Desde q2q_2q2:
        - Con 000 o 111 permanece en q2q_2q2

Este NFA acepta cadenas que contienen "01" en alguna parte. Por ejemplo, acepta "01", "1001", y "00101".

Ejemplo 2: NFA para aceptar cadenas que contienen al menos un "a" o una "b"

- **Alfabeto:** {a,b}\{a, b\}{a,b}
- **Estados:** {q0,q1,q2}\{q_0, q_1, q_2\}{q0,q1,q2}
- **Estado inicial:** q0q_0q0
- **Estado(s) de aceptación:** q1,q2q_1, q_2q1,q2
- **Transiciones:**
  - Desde q0q_0q0:
    - Con aaa va a q1q_1q1
    - Con bbb va a q2q_2q2
  - Desde q1q_1q1:
    - Con aaa o bbb permanece en q1q_1q1
  - Desde q2q_2q2:
    - Con aaa o bbb permanece en q2q_2q2

Este NFA acepta cualquier cadena que contenga al menos un "a" o una "b". Por ejemplo, acepta "a", "b", "ab", "ba", y "bbb".

Autómata con transiciones epsilon: Es un tipo especial de NFA que permite transiciones sin consumir un símbolo de entrada (transiciones epsilon, ϵ\epsilonϵ). Esto significa que el autómata puede cambiar de estado "gratuitamente" en ciertos momentos, lo que hace el diseño de los autómatas más flexible. Los autómatas con transiciones epsilon son útiles para optimizar ciertas tareas en el reconocimiento de lenguajes o la simulación de sistemas más complejos.

Ejemplo 1: ϵ-NFA para aceptar cadenas que contienen "ab"

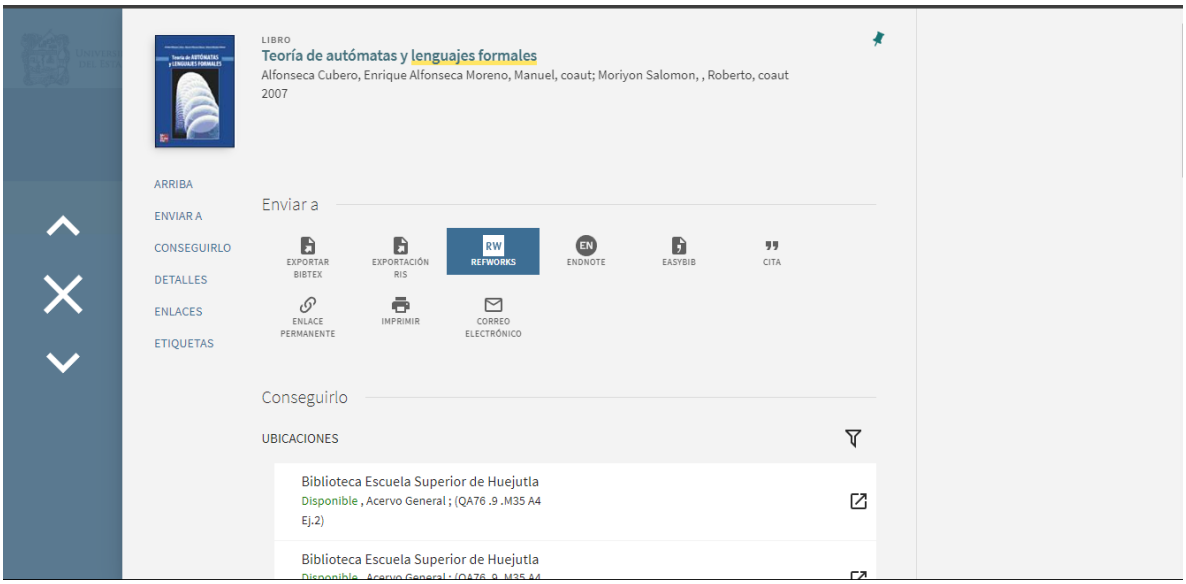
- **Alfabeto:** {a,b}\{a, b\}{a,b}
- **Estados:** {q0,q1,q2}\{q_0, q_1, q_2\}{q0,q1,q2}
- **Estado inicial:** q0q_0q0
- **Estado(s) de aceptación:** q2q_2q2
- **Transiciones:**
  - Desde q0q_0q0:
    - Con ϵ\epsilonϵ (transición epsilon) va a q1q_1q1
    - Con aaa va a q1q_1q1
  - Desde q1q_1q1:
    - Con bbb va a q2q_2q2
  - Desde q2q_2q2:
    - Con aaa o bbb permanece en q2q_2q2

Este ϵ-NFA acepta cadenas que contienen "ab" en alguna parte. La transición epsilon permite al autómata moverse de q0q_0q0 a q1q_1q1 sin consumir ningún símbolo, haciendo que la aceptación sea más flexible. Por ejemplo, acepta "ab", "aabb", "xab", y "xyzab".

Ejemplo 2: ϵ-NFA para aceptar cadenas sobre {0,1}\{0, 1\}{0,1} que contengan "01"

- **Alfabeto:** {0,1}\{0, 1\}{0,1}
- **Estados:** {q0,q1,q2}\{q_0, q_1, q_2\}{q0,q1,q2}
- **Estado inicial:** q0q_0q0
- **Estado(s) de aceptación:** q2q_2q2
- **Transiciones:**
  - Desde q0q_0q0:
    - Con 000 va a q1q_1q1
    - Con ϵ\epsilonϵ va a q2q_2q2
  - Desde q1q_1q1:
    - Con 111 va a q2q_2q2
  - Desde q2q_2q2:
    - Con 000 o 111 permanece en q2q_2q2

Este ϵ-NFA acepta cualquier cadena que contenga "01" en alguna parte. La transición epsilon desde q0q_0q0 a q2q_2q2 permite que el autómata acepte cadenas que no contienen "01", como "1", pero también permite transiciones hacia el estado de aceptación cuando "01" aparece. Por ejemplo, acepta "01", "1001", "0101", y "111".




LIBRO
Lenguajes formales y teoría de autómatas
Giró, Juan Vázquez, Juan, autor; Meloni, Brenda, autor; Constable, Leticia, autor
2015
ARRIBA
ENVIAR A
CONSEGUIRLO
DETALLES
ENLACES
ETIQUETAS
Enviar a
EXPORTAR BIBTEX
EXPORTACIÓN RIS
REFWORKS
ENDNOTE
EASYBIB
CITA
ENLACE PERMANENTE
IMPRIMIR
CORREO ELECTRÓNICO
Conseguirlo
UBICACIONES
Biblioteca Escuela Superior de Huejutla
Disponible , Acervo General ; (QA267 .3 G5 Ej.1)
Biblioteca Escuela Superior de Tizayuca
Disponible , Acervo General ; (QA267 .3 G5 Ej.1)


LIBRO
Teoría de automatas y lenguajes formales
García, Pedro, colab.
c2001
ARRIBA
ENVIAR A
CONSEGUIRLO
DETALLES
ENLACES
ETIQUETAS
Enviar a
EXPORTAR BIBTEX
EXPORTACIÓN RIS
REFWORKS
ENDNOTE
EASYBIB
CITA
ENLACE PERMANENTE
IMPRIMIR
CORREO ELECTRÓNICO
Conseguirlo
UBICACIONES
Biblioteca Central
Fuera de biblioteca , Acervo General ; (QA76 .9
.M35 T4 Ej.1)

Referencias

Giró, J., Vázquez, J., autor, Meloni, B., autor, & Constable, L., autor. (2015). *Lenguajes formales y teoría de autómatas*. Alfaomega.

Alfonseca Cubero, E., & Alfonseca Moreno, Manuel, coaut. (2007). *Teoría de autómatas y lenguajes formales*. McGraw-Hill.

García, Pedro, colab. (2001). *Teoría de automatas y lenguajes formales*. Alfaomega.

Massolo, A. (2019). Sesgos de razonamiento, lenguajes formales y enseñanza de la lógica. *Límite: revista de filosofía y psicología*, *14*.