UNIVERSIDAD AUTÓNOMA DE CHIAPAS

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN CAMPUS 1

ACTIVIDAD. Act. 1.3 Investigar los Conceptos del analizador léxico

ING. EN DESARROLLO Y TECNOLOGÍAS DE SOFTWARE

ALUMNO: Gabriel Omar Fuentes Chacon.

DOCENTE: Mtro. Luis Gutiérrez Alfaro

Fecha de entrega: 25/08/2023

INDICE

INTRODUCCION

Durante esta investigación se podrá explicar, las expresiones regulares representan una forma algebraica equivalente a los autómatas, y tienen un papel crucial en la descripción de patrones en texto. Aunque simples, son poderosas para definir lenguajes regulares y permiten expresar las cadenas que se desean aceptar, un ejemplo ilustrativo es el lenguaje L1, compuesto por repeticiones de "ab" en diversas longitudes.

Autómatas, una disciplina anclada en la Informática Teórica, destaca por su naturaleza multidisciplinaria y su resistencia a los cambios tecnológicos. Son utilizados para comprender sistemas cambiantes en respuesta a eventos y acciones, un ejemplo sería un autómata de reconocimiento de patrones para el procesamiento de voz a texto, el panorama de los autómatas incluye los deterministas y no deterministas. Los primeros, regidos por transiciones únicas, trazan rutas específicas en respuesta a entradas. Los segundos, más flexibles, permiten múltiples trayectorias, resultando en un proceso más complejo de análisis.

La "Tabla de Símbolos" emerge como una estructura clave en la construcción de compiladores. Actúa como una base de datos que rastrea y gestiona información vital del programa, como variables, funciones y constantes. Esta tabla es fundamental para el análisis, optimización y generación de código preciso, las herramientas automáticas para generar analizadores léxicos simplifican la identificación de componentes básicos en el código fuente. Estas herramientas, como Flex, ANTLR y otras, se basan en expresiones regulares para describir patrones y producir automáticamente el código necesario para el análisis.

Las Gramáticas Libres de Contexto (CFG) son esenciales en la teoría de lenguajes y la construcción de compiladores. Estas describen la estructura sintáctica de un lenguaje mediante reglas de producción. Se distingue entre gramáticas ambiguas y no ambiguas, y la Forma Enunciativa simplifica su comprensión. Las características de las gramáticas abarcan la generación de cadenas válidas, la evitación de ambigüedad y la precisión en la descripción sintáctica, esta construcción de gramáticas y formas enunciativas involucra la definición de símbolos, reglas y categorización. La remoción de ambigüedad busca asegurar interpretaciones únicas, mientras que la normalización de CFG simplifica el análisis, todo este conjunto de conceptos y herramientas conforma el entramado de la Gramática Libre de Contexto, un pilar en la teoría de lenguajes y la construcción de compiladores.

DESAROLLO

Expresiones regulares.

Es un equivalente algebraico para un autómata, es utilizado en muchos lugares como un lenguaje para describir patrones en texto que son sencillos pero muy útiles, pueden definir exactamente los mismos lenguajes que los autómatas pueden describir, los lenguajes regulares, ofrecen algo que los autómatas no, expresan las cadenas que queremos aceptar

Como ejemplo podemos tomar lo siguiente:

L1={ab, abab, ababab, abababab,…}

Las palabras de L1 son simplemente repeticiones de “ab”, cualquier número de veces, esta “regularidad” consiste en que las palabras contienen “ab” algún número de veces.

Una expresión regular es como una fórmula hecha con letras y símbolos nos tiene que contribuir, para describir cómo debe lucir algo que estás buscando, se utiliza mayormente para trabajar con palabras o frases y así hallar patrones especiales o hacer cosas como encontrar, buscar, cambiar y revisar si algo es válido, además las expresiones regulares están conformadas por letras normales y símbolos especiales que representan grupos de letras, repeticiones, diferentes opciones y otras cosas parecidas.

Autómatas

El estudio de la teoría de autómatas y de los lenguajes formales se puede ubicar en el campo científico de la Informática Teórica, un campo clásico y multidisciplinar dentro de los estudios universitarios de Informática. Es un campo clásico debido no solo a su antigüedad (anterior a la construcción de los primeros ordenadores) sino, sobre todo, a que sus contenidos principales no dependen de los rápidos avances tecnológicos que han hecho que otras ramas de la Informática deban adaptarse a los nuevos tiempos a un ritmo vertiginoso. Es multidisciplinar porque en sus cimientos encontramos campos tan aparentemente dispares como la lingüística, las matemáticas o la electrónica, como ejemplo podríamos tomar los siguiente para explicar de mejor manera un Autómata de Reconocimiento de Patrones, siendo un sistema de reconocimiento de voz que convierte palabras habladas en texto escrito, este sistema puede ser modelado como un autómata, sus estados podrían ser fonemas o sonidos individuales, y las transiciones entre los estados representarían las secuencias de sonidos en una palabra se compondría por:

Estados: Fonemas y silencio

Estado inicial: Silencio

Transiciones:

Si se detecta un fonema "ka", se transiciona a otro estado.

Si le sigue el fonema "t", la máquina permanece en ese estado.

Luego, si escucha "a", cambia de estado nuevamente.

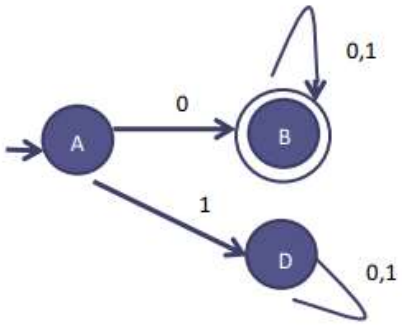
Después de detectar "t", "i" y "e", el autómata podría finalizar en un estado de reconocimiento de la palabra "catie".

Los autómatas son usados para explicar cómo funcionan sistemas que cambian su condición cuando ocurren ciertas acciones o situaciones. Estos autómatas pueden volverse bastante elaborados y se emplean en campos como la computación, el estudio de lenguajes, la electrónica y otros, con el propósito de crear representaciones y entender mejor sistemas que evolucionan con el tiempo y tomar decisiones.

Segundo ejemplo:

Autómata es una máquina matemática M formada por 5 elementos M = (Σ, Q, s, F, δ) donde Σ es un alfabeto de entrada, Q es un conjunto finito de estados, s es el estado inicial, F es un conjunto de estados finales o de aceptación y δ (delta) es una relación de transición.

Ejemplo: Σ ={0,1} s = A Q= {A,B,D} F = {B} δ: (A,0) = B (A,1) = D (B,0) = B (B,1) = B (D,0) = D (D,1) = D



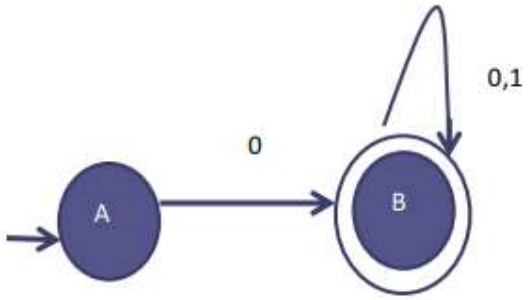
Autómatas no determinísticos.

Autómata Finito No Determinista (AFND) es un autómata finito en donde δ no es necesariamente una función de transición, es decir, que para cada par (estado actual y símbolo de entrada) le corresponde cero, uno, dos o más estados siguientes, Normalmente la relación de transición para un AFND se denota con Δ.

Son herramientas matemáticas que describen situaciones donde un mismo estado puede tener múltiples cambios en respuesta a una entrada, estos autómatas, desde un estado específico, al enfrentar una entrada, hay varias opciones posibles, esto se puede ver como una "elección" entre diferentes caminos a seguir, se caracteriza por ser en lugar de una secuencia de estados única, se consideran múltiples secuencias potenciales, pero esto no significa que los autómatas no determinísticos hagan cosas fuera de lo normal, más bien, ofrecen una manera más flexible de describir sistemas donde las transiciones no son predecibles, en la teoría de la computación y para definir lenguajes y gramáticas, a pesar de esto son más complejos de analizar que los autómatas

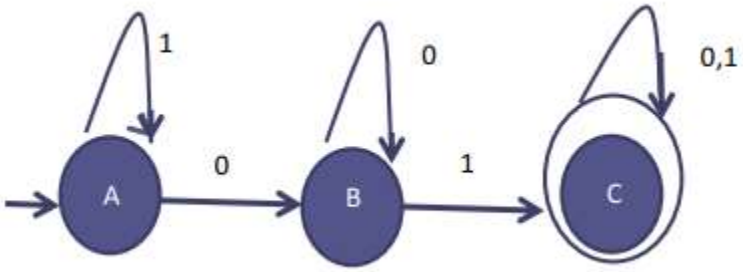
determinísticos debido a las distintas rutas, a menudo, se convierten en versiones deterministas para facilitar su uso en sistemas informáticos y otras aplicaciones prácticas.

Ejemplo:
Obtenga un AFND dado el siguiente lenguaje definido en el alfabeto Σ= {0,1}. El conjunto de cadenas que inician en 0.



Segundo ejemplo:

Obtenga un AFND dado el siguiente lenguaje definido en el alfabeto Σ= {0,1}. El conjunto de cadenas que contienen a la sub-cadena ”01”.



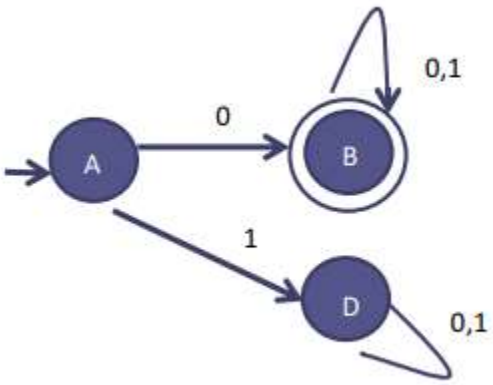
Autómatas deterministicos.

Autómata Finito Determinista (AFD) es un autómata finito en donde δ (delta) es una función de transición, es decir, que para cada par (estado actual y símbolo de entrada) le corresponde un único estado siguiente, cuando le das información al Autómata Finito Determinista (AFD), este comienza en un estado específico, luego, para cada elemento de la información, el autómata sigue caminos predefinidos para moverse entre los diferentes estados. Una vez que toda la información ha sido procesada, el AFD puede terminar en un estado especial, que es como un punto final, si la información cumple con ciertas condiciones que se han establecido.

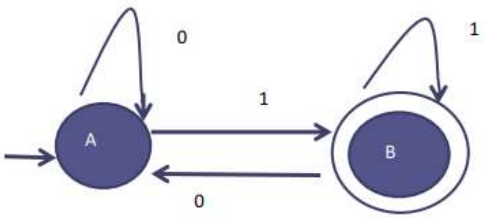
Los AFD son herramientas esenciales en la teoría de lenguajes y autómatas. Se utilizan para describir y detectar patrones y estructuras en secuencias de símbolos, lo que tiene usos en la creación de programas de traducción, el análisis de texto y en muchas otras áreas de la informática.

Ejemplo:

Obtenga un AFD dado el siguiente lenguaje definido en el alfabeto Σ= {0,1}. El conjunto de cadenas que inician en “0”



Segundo ejercicio: Obtenga un AFD dado el siguiente lenguaje definido en el alfabeto Σ= {0,1}. El conjunto de cadenas que terminan en “1”.



Aspectos esenciales de un AFD:

Estados Limitados: Un AFD se compone de un conjunto finito de estados, cada uno representando una situación específica en el sistema que se está describiendo.

Conjunto de Entrada: También contiene un grupo finito de símbolos, conocido como "conjunto de entrada", que refleja las posibles entradas o eventos que pueden ocurrir y que determinarán cómo el autómata cambia entre estados.

Reglas de Cambio: La operación de un AFD sigue reglas precisas de cambio definidas por una función de transición. Esta función describe cómo el autómata se mueve de un estado a otro en respuesta

a una entrada particular. Cada estado tiene una transición específica para cada símbolo en el conjunto de entrada.

Inicio Definido: Hay un estado de partida desde el cual el autómata comienza a funcionar cuando se le presenta una entrada.

Estados de Cumplimiento o Finalización: Puede haber uno o varios estados de cumplimiento o finalización, que señalan que el autómata ha alcanzado un estado deseado o ha logrado completar una tarea específica.

Matrices de transición.

Básicamente, una matriz de transición es una especie de tabla organizada con filas y columnas. En cada casilla de esta tabla se coloca un número que representa la probabilidad o el ritmo de cambio entre dos estados específicos, estos números en la matriz nos cuentan cómo las probabilidades o los ritmos de cambio en un sistema se distribuyen entre las diferentes opciones de cambio, por ejemplo, si hablamos de autómatas, una matriz de transición podría mostrar cómo un autómata se desplaza entre sus distintos estados al recibir diferentes señales de entrada, una matriz de transición podría demostrar cómo las probabilidades evolucionan en una cadena de Márkov, que es un proceso estocástico que cambia su estado a lo largo del tiempo.

las matrices de transición son una forma poderosa de visualizar y entender cómo ocurren los cambios en sistemas complicados. Además, nos ayudan a modelar cómo la evolución de un sistema puede impactar en su comportamiento futuro.

Un ejemplo que podemos tener sería el siguiente donde un autómata celular que vive en una cuadrícula y puede estar en los estados "vivo" o "muerto", la matriz de transición podría ser:

Vivo Muerto
Vivo 0.8 0.2
Muerto 0.3 0.7

En este caso, la matriz describe cómo los estados "vivo" y "muerto" evolucionan en cada paso de tiempo, si una persona está viva, hay un 80% de probabilidad de que este estado también esté vivo en el próximo paso

Tabla de símbolos

La "Tabla de Símbolos" o "Lista de Símbolos" desempeña un rol fundamental en la operación de un compilador, una herramienta esencial que traduce el código fuente creado por los programadores en un formato ejecutable por la computadora, la función principal de la lista de símbolos reside en la tarea de rastrear y gestionar la información crítica para el análisis y procesamiento del programa, podemos considerar la lista de símbolos como un equivalente a un diccionario o incluso una base de datos, donde se almacenan datos relevantes relacionados con variables, funciones y otros elementos presentes en el código fuente a medida que el compilador analiza el código fuente, la lista de símbolos se enriquece con detalles específicos, como los nombres de las variables, sus tipos de datos, ubicaciones en la memoria, información sobre funciones, valores constantes y otros atributos que están intrínsecamente conectados con la semántica del programa, es muy importancia de la tabla de símbolos porque radica en su papel esencial de mantener un registro minucioso de los elementos utilizados en el programa, permitiendo que el compilador realice tareas vitales como verificaciones de consistencia, optimizaciones del código y generación precisa del código final, como última instancia, la tabla de símbolos empodera al compilador para llevar a cabo su trabajo de manera eficiente y exacta, lo que es fundamental para la correcta ejecución del programa en la computadora.

Ejemplo: Constantes en Java

Supongamos que tenemos este código en Java:

javaCopy code

final int LIMITE = 100;

final double PI = 3.14159;

Tabla de símbolos

Nombre	**Tipo**	**Valor**	**Ubicación en Memoria**
LIMITE	int	100	3000
PI	double	3.14159	3004

En cada ejemplo, la tabla de símbolos almacena información relevante sobre las variables, funciones o constantes presentes en el código fuente, permitiendo que el compilador realice un seguimiento y una gestión adecuados de los elementos del programa durante el proceso de compilación.

Diferentes herramientas automáticas para generar analizadores léxicos.

Las herramientas automáticas para crear analizadores léxicos hacen más fácil la tarea en el proceso de construcción del compilador que se encarga de reconocer las partes básicas del código fuente. Estas herramientas permiten a los programadores describir cómo lucen estas partes (como números, nombres, signos, etc.) usando patrones especiales llamados expresiones regulares. A partir de estas descripciones, la herramienta genera automáticamente el código necesario para llevar a cabo esta identificación, esto resulta en un ahorro de tiempo y disminuye la probabilidad de cometer errores al hacer el análisis manualmente, estas herramientas como Flex, ANTLR y similares, ofrecen diversas capacidades y se ajustan a diferentes lenguajes de programación, dándole a los desarrolladores la opción de elegir la herramienta que mejor se adapte a sus requerimientos.

En el ámbito de la compilación y el procesamiento de lenguajes, hay diversas herramientas automáticas para crear analizadores léxicos, estas herramientas simplifican la elaboración de la sección del compilador encargada de examinar y comprender las partes básicas, como nombres, palabras clave, números y símbolos, en el código fuente.

Las herramientas con las que se cuentan:

Ragel: Ragel es una herramienta que genera máquinas de estados finitos para analizar texto. Aunque no se centra únicamente en análisis léxico, puede ser utilizado para generar componentes léxicos de manera eficiente.

Pygments: Pygments es una herramienta de resaltado de sintaxis escrita en Python que puede utilizarse para analizar y resaltar la sintaxis de varios lenguajes de programación. Aunque no es una herramienta puramente para generar analizadores léxicos, se utiliza comúnmente para resaltar la sintaxis en editores de código y sistemas de documentación.

Flex (Fast Lexical Analyzer Generator): Flex es una herramienta ampliamente utilizada para generar analizadores léxicos en C o C++. Permite definir patrones de expresiones regulares que coinciden con los componentes léxicos y generar automáticamente el código necesario para el análisis léxico.

ANTLR (ANother Tool for Language Recognition): ANTLR es una herramienta más completa que permite generar analizadores léxicos y sintácticos. Permite definir gramáticas contextuales y generar analizadores en varios lenguajes de programación, como Java, C# y Python.

Lex: JLex es una herramienta similar a Flex, pero está diseñada para generar analizadores léxicos en Java. Permite definir patrones de expresiones regulares y generar código Java para el análisis léxico.

Lex/Flex++: Lex y Flex++ son versiones extendidas de las herramientas Lex y Flex, respectivamente, que admiten C++. Son útiles cuando se desea trabajar con características específicas de C++ en el análisis léxico.

Gramática libre de Contexto

- **Concepto de Gramática Libre de Contexto (CFG):** Las Gramáticas Libres de Contexto (CFG) son herramientas formales usadas para describir la estructura sintáctica de un lenguaje. Se componen de reglas de producción que indican cómo combinar símbolos no terminales y terminales para crear cadenas válidas en el lenguaje. Son esenciales en teoría de lenguajes formales y en la construcción de compiladores.

- **Diferencia entre Gramática Ambigua y No Ambigua:** Una gramática se considera ambigua si puede dar lugar a múltiples interpretaciones de una misma cadena. En contraste, una gramática no ambigua proporciona una única interpretación. La ambigüedad puede complicar el análisis sintáctico y la comprensión del lenguaje.

- **Explicación de la Forma Enunciativa con un ejemplo:** La Forma Enunciativa es una representación estandarizada de una CFG que facilita su comprensión. Clasifica las reglas de producción en categorías como sustantivos, verbos, etc., lo que ayuda a identificar partes del discurso. Por ejemplo, en "El gato come el ratón", se categorizan como: Artículo + Sustantivo + Verbo + Artículo + Sustantivo.

Características de Gramáticas de Lenguajes:

- **Descripción de las Propiedades Clave:** Las gramáticas de lenguajes deben generar cadenas válidas y rechazar inválidas. Deben evitar ambigüedad para un análisis y comprensión sencillos, y ser lo suficientemente expresivas para describir la sintaxis con precisión.

- **Ejemplo de Características:** Si consideramos un lenguaje que suma números, una gramática cumpliría generando "2 + 3" y rechazando "+ 4 5".

Proceso de Construcción de Gramáticas de Lenguajes:

- **Pasos para Crear Gramáticas:** Implica definir símbolos no terminales y terminales, establecer reglas de producción para su combinación, y verificar que la gramática genere el lenguaje y evite la ambigüedad.

- **Ejemplo del Proceso:** Si deseamos una gramática para expresiones matemáticas simples, definiremos símbolos para expresiones, operadores y números, estableceremos reglas para combinarlos y formar expresiones válidas.

Proceso de Construcción de Formas Enunciativas:

- **Construcción de Formas Enunciativas:** Implica agrupar reglas en categorías como sustantivos y verbos, facilitando el análisis sintáctico.

- **Ejemplo del Proceso:** En una gramática para describir acciones, categorizaremos reglas como "Sujeto" y "Verbo", simplificando el análisis de frases como "El gato duerme".

Proceso de Remoción de Ambigüedad de Gramáticas:

- **Abordaje de Ambigüedad:** Implica redefinir reglas o reorganizar la estructura para garantizar una sola interpretación válida de cada cadena.

- **Ejemplo del Proceso:** Si una gramática para operaciones matemáticas es ambigua, redefiniremos reglas para asegurar interpretaciones únicas.

Descripción de la Normalización de CFG:

- **Normalización de CFG:** Implica reformular la gramática para cumplir propiedades estándar, simplificando el análisis sintáctico y la comprensión.
- **Ejemplo de Normalización:** Si hay reglas redundantes o ambiguas en la gramática, las redefiniremos para eliminar problemas de ambigüedad y redundancia.

Proceso de Normalización de CFG:

- **Pasos para Normalizar CFG:** Implica ajustar reglas y estructura para cumplir con estándares definidos y mejorar la calidad sintáctica y semántica de la gramática.

Los ejemplos mostrados a continuación corresponden a los lenguajes libres del contexto que fueron introducidos en los ejemplos 1, 2 y 3 del apunte de autómatas de pila.

Ejemplo 1:

La siguiente gramática genera las cadenas del lenguaje L1 = {wcwR

/ w ∈ {a, b}*

}

G1 = ({A}, {a, b, c}, P1, S1), y P1 contiene las siguientes producciones

S1 → A

A → aAa

A → bAb

A → c

Estas gramáticas, conocidas también como gramáticas de tipo 2 o gramáticas independientes del contexto, son las que generan los lenguajes libres o independientes del contexto. Los lenguajes libres del contexto son aquellos que pueden ser reconocidos por un autómata de pila determinístico o no determinístico.

Las gramáticas libres del contexto se escriben, frecuentemente, utilizando una notación conocida como BNF (Backus-Naur Form). BNF es la técnica más común para definir la sintaxis de los lenguajes de programación. En esta notación se deben seguir las siguientes convenciones: - los no terminales se escriben entre paréntesis angulares < > - los terminales se representan con cadenas de caracteres sin paréntesis angulares

el lado izquierdo de cada regla debe tener únicamente un no terminal (ya que es una gramática libre del contexto)

- el símbolo ::=, que se lee “se define como” o “se reescribe como”, se utiliza en lugar de →
- varias producciones del tipo

<A> ::= <B1>

<A> ::= <B2>

<A> ::= <Bn>

se pueden escribir como <A> ::= <B1> | <B2> | ... |<Bn>

Una Gramática Libre de Contexto (CFG) es una herramienta utilizada para modelar cómo se construyen las frases y expresiones en un lenguaje. Se compone de reglas que explican cómo combinar palabras y estructuras para formar oraciones válidas son fundamentales en la teoría de lenguajes y en la creación de compiladores, ya que ofrecen una forma ordenada y sistemática de establecer las reglas de un lenguaje, lo que es esencial para entender y procesar el código de programación de manera precisa.

Conclusión

En conclusión, hemos explorado el mundo de las expresiones regulares y los autómatas, dos herramientas fundamentales en la teoría de lenguajes y la construcción de compiladores. Las expresiones regulares, al ser una representación algebraica de los autómatas, desempeñan un papel esencial en la descripción de patrones en texto, permitiendo definir lenguajes regulares y expresar cadenas deseables, los autómatas, a su vez, se presentan como un campo multidisciplinario arraigado en la Informática Teórica, con aplicaciones en la comprensión de sistemas cambiantes, la distinción entre autómatas deterministas y no deterministas revela la interacción única entre rutas predeterminadas y flexibilidad en el análisis, respectivamente, por lo que la "Tabla de Símbolos" emerge como una estructura esencial en la construcción de compiladores, facilitando el seguimiento y la gestión de información crítica en el análisis y procesamiento de programas, además de la utilización de herramientas automáticas para generar analizadores léxicos, como Flex y ANTLR, representa un avance significativo en la simplificación del proceso de identificación de componentes básicos en el código fuente, por otro lado, las Gramáticas Libres de Contexto nos brindan una estructura organizada y precisa para describir la estructura sintáctica de un lenguaje, con su Forma Enunciativa simplificando la comprensión, para finalizar el conjunto de estos conceptos y herramientas se entrelazan para formar el sólido fundamento de la Gramática Libre de Contexto, que juega un papel esencial en la construcción de compiladores y en la comprensión de la estructura de los lenguajes, medida que continuamos avanzando en el mundo de la informática, es imperativo reconocer la importancia de estas bases teóricas para lograr un procesamiento preciso y eficiente del código y la información.

BIBLIOGRAFIAS

Sguerra, M. D. (2006). Las expresiones regulares. *INVENTUM*, *1*(1), 31-37.

Pérez, EM, Acevedo, JM y Silva, CF (2009). *Autómatas programables y sistemas de automatización* . Marcombo.

Alvarez, G. I., Ruiz, J., & García, P. (2009). Comparación de dos algoritmos recientes para inferencia gramatical de lenguajes regulares mediante autómatas no deterministas. *Ingeniería y competitividad*, *11*(1), 21-36.

LILIA, O. T. AUTÓMATAS DETERMINISTAS Y NO DETERMINISTAS (EJERCICIOS).

Ossa, J. C. (2013). Matrices de transición y patrones de variabilidad cognitiva. *Universitas Psychologica*, *12*(2), 559-570.

Léonard Schardijn, D. (2018). Comprobador automático de tablas de símbolos.

Fuente, A. A. J., Lovelle, J. M. C., Soler, F. O., Castanedo, R. I., Díez, M. C. L., & Gayo, J. E. L. TABLAS DE SÍMBOLOS.

Mazariegos, W. CLASIFICACION Y HERRAMIENTAS GENERALES PARA EL DISEÑO AUTOMÁTICO DE COMPONENTES ESPECÍFICOS DEL COMPILADOR.

i Arís, J. M. S. (1998). La gramática categorial como gramática universal. *Revista española de lingüística*, *28*(1), 89-114.

de Visualización Molecular-AVISMO, A. Una Gramática Libre de Contexto para el Lenguaje del.