Universidad Autónoma de Chiapas

Facultad de Contaduría y Administración Campus 1

Licenciatura en Ingeniería y Desarrollo de tecnologías de Software

18-8-2024

Act. 1.1 Investigar analizador Léxico y Lenguajes Regulares. -2024

COMPILADORES

Alumna:

A221641 Karla Carolina Lopez Ruiz

DOCENTE

Dr. Luis Gutiérrez Alfaro

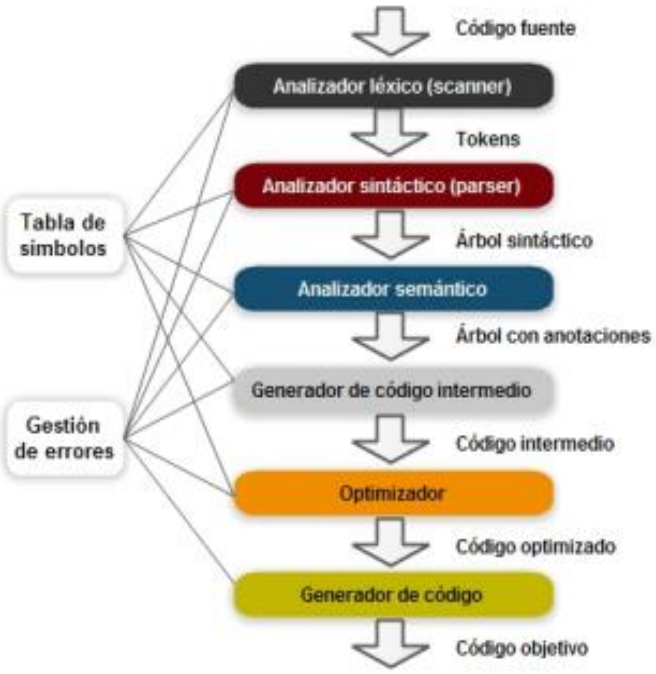
18/08/2024

Tuxtla Gutierrez, Chiapas

La importancia que es un analizador léxico;

Un analizador léxico es crucial en la compilación, pues actúa como la primera fase del proceso. Su principal función es leer el código fuente y convertirlo en una secuencia de tokens (componentes léxicos) que el analizador sintáctico utilizará posteriormente. Este proceso incluye la eliminación de espacios en blanco y comentarios, y la validación de símbolos del lenguaje. El analizador léxico actúa normalmente como un procedimiento que es llamado por el analizador sintáctico cuando éste necesita un nuevo token. El analizador léxico devuelve, al analizador sintáctico que lo llamó, una representación del token que ha encontrado en el texto fuente. Para ello habrá saltado los espacios y los comentarios que pudiera haber delante del token. La representación que devuelve el analizador léxico depende de la implementación, pero normalmente se compone de dos informaciones: el tipo de token y su valor o dirección en una tabla.

¿Qué es un token?
Es una agrupación de caracteres reconocidos por el analizador léxico que constituyen los símbolos con los que se forman las sentencias del lenguaje. Lo que el analizador léxico devuelve al analizador sintáctico es el nombre de ese símbolo junto con el valor del atributo (si ese token lo necesita, ya que no todos los tokens llevan atributo como, por ejemplo: una palabra reservada "if").



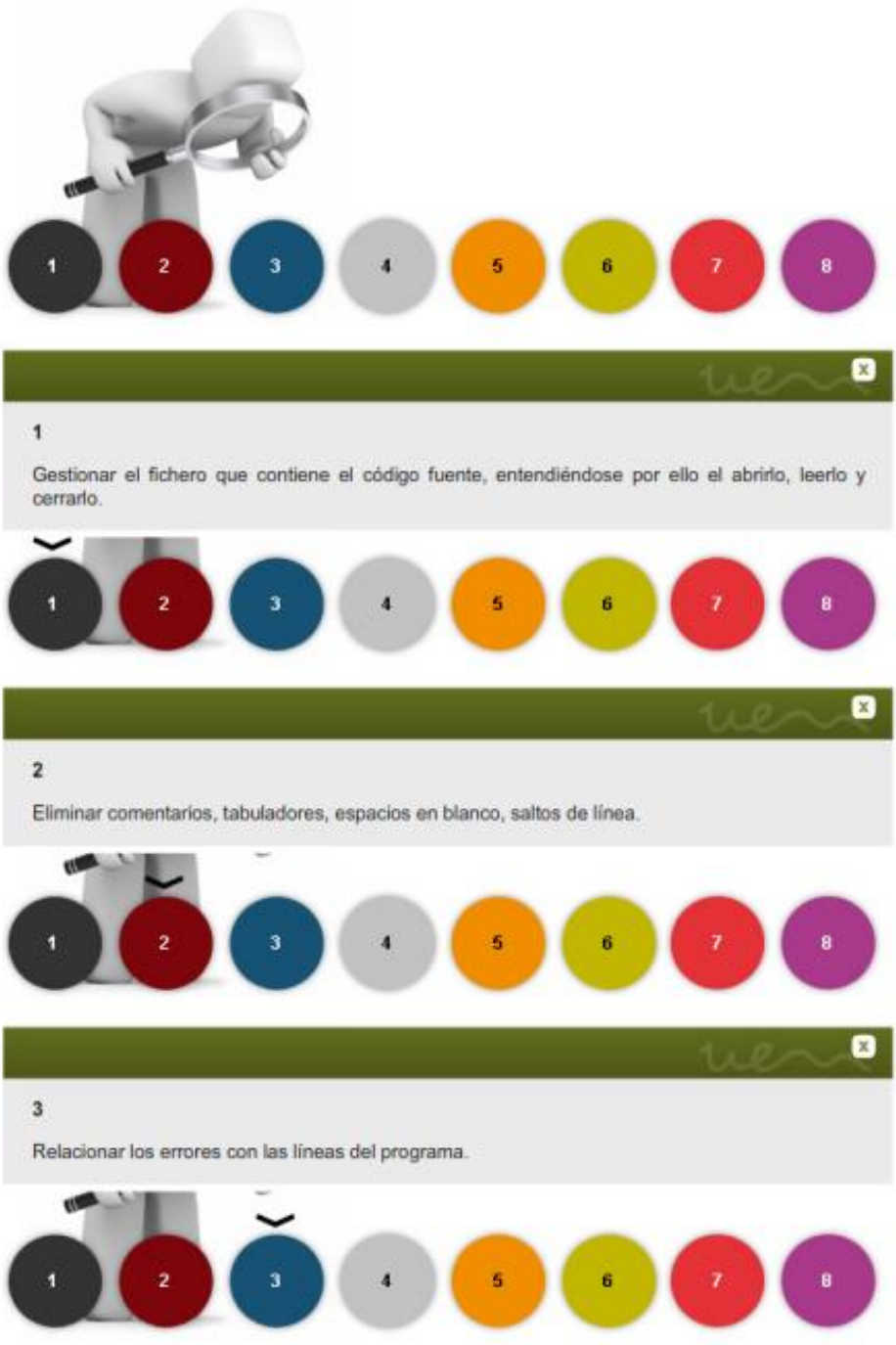
Sánchez Dueñas, G. y Valverde Andreu, J. A. (2022). Compiladores e intérpretes: un enfoque pragmático: (1 ed.). Ediciones Díaz de Santos. Recuperado de https://elibro.net/es/ereader/uachiapas/269960?page=58.

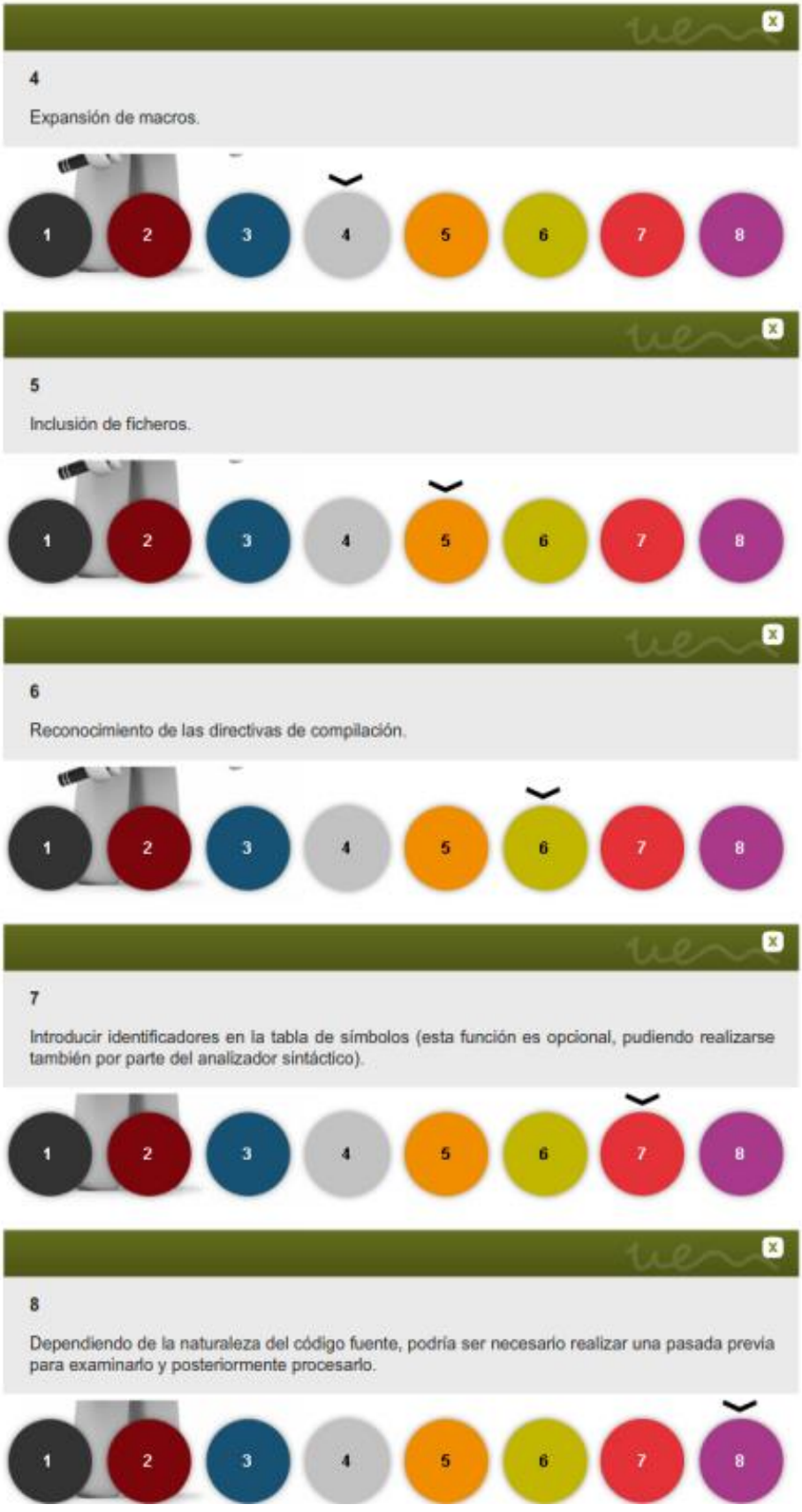
Funciones del Analizador Léxico

Figura 2.5: Esquema por etapas definitivo de un traductor

La fase inicial de un compilador, es decir, su análisis lexicográfico, también llamado análisis léxico. Las técnicas que se utilizan para la construcción de analizadores léxicos también se pueden implementar en otras áreas como, por ejemplo, a lenguajes de consulta y sistemas de recuperación de información. Sin importar el área de aplicación, el problema básico es la especificación y diseño de programas que ejecuten las acciones activadas por palabras que siguen ciertos patrones dentro de cadenas a reconocer. Debido a que la programación dirigida por patrones es sin du- da alguna de gran utilidad, existen gran cantidad de metalenguajes8 que permiten la programación de pares que tengan la forma patrón-acción, de tal manera que la acción se ejecute una vez se encuentre el patrón dentro de una cadena. La fase de análisis léxico de un compilador tiene como función leer el programa fuente como un conjunto de caracteres y separarlo en tokens9. Como básicamente la función de un analizador léxico es un caso especial de coincidencia de patrones, se necesita estudiar métodos de especificación y re- conocimiento de patrones en la medida en que se aplican al proceso de análisis léxico. Estos métodos son principalmente las expresiones regulares y autómatas finitos10 un analizador léxico debe funcionar de manera tan eficiente como sea posible. Por esto, también se debe prestar mucha atención a los detalles prácticos de la estructura del analizador.

El analizador léxico realiza varias funciones, siendo la fundamental la de agrupar los caracteres que va leyendo uno a uno del programa fuente y formar los tokens. Las otras funciones que realiza el analizador léxico son:





Vega Castro, R. A. (2008). Compilador de pseudocódigo como herramienta para el aprendizaje en la construcción de algoritmos: (1 ed.). Madrid, Bubok Publishing S.L. Recuperado de https://elibro.net/es/ereader/uachiapas/260295?page=27.

Componentes léxicos, patrones y lexema.

Componentes léxicos (Tokens): Son las unidades mínimas de información con significado en la compilación, como identificadores o palabras clave.

Constituyen los símbolos terminales de la gramática:

- Palabras reservadas.
- Identificadores.
- Operadores y constantes.
- Símbolos de puntuación y especiales.

Patrón: Es la regla que define qué secuencias de caracteres se pueden considerar como un determinado token. Como por ejemplo el nombre de un identificador, o el valor de un número. Un token puede tener uno o infinitos lexemas. Por ejemplo, las palabras reservadas tienen un solo lexema, mientras que los identificadores o los números tienen infinitos.

Lexema: Es la secuencia de caracteres en el código fuente que coincide con un patrón específico. Es la forma de describir los tipos de lexema. Esto se realiza utilizando expresiones regulares.

Ejemplo:

Token (Componente léxico)	Lexema	Patrón
While	While	While
Suma	+	+
Identificador	a, valor, b	[a-zA-Z]+
Número	5, 3, 25, 56	[0-9]+(\.[0-9]+)?

Lenguajes Regulares.

La clase de los lenguajes regulares contiene los lenguajes estructuralmente más simples dentro de la Jerarquía de Chomsky. A día de hoy, se han descrito lenguajes todavía más simples de los descritos originalmente por Chomsky, los que conforman lo que se conoce como Jerarquía Subregular y que no alcanzan ni siquiera el nivel de complejidad de los lenguajes de los que nos ocuparemos aquí. Aunque no es totalmente irrelevante para la lingüística, en este texto dejaremos de lado la Jerarquía Subregular. Los lenguajes regulares propiamente dichos, en cambio, sí tienen interés lingüístico, ya que, como veremos más adelante, ciertas propiedades de la fonología y quizá la morfología de las lenguas naturales son modelables con lenguajes de esta complejidad estructural. De forma intuitiva, un lenguaje regular puede caracterizarse como aquel cuyas cadenas contienen dependencias lineales, de tal modo que la presencia de un símbolo u otro en una posición determinada de la cadena depende exclusivamente del símbolo que lo precede inmediatamente

- Explicar el Lema de Bombeo para lenguajes regulares con un ejemplo.

El lema de bombeo es una propiedad fundamental de los lenguajes regulares que establece que para cualquier lenguaje regular, existe un número entero ppp tal que cualquier cadena sss en el lenguaje, con longitud al menos ppp, se puede dividir en tres partes s=xyzs = xyzs=xyz, cumpliendo con ciertas condiciones que permiten "bombear" la cadena sin salir del lenguaje.

Ejemplo: Consideremos el lenguaje L={anbn∣n≥0}L = \{a^n b^n | n \geq 0\}L={anbn∣n≥0}. Para una cadena s=aaabbbs = aaabbbs=aaabbb, el lema de bombeo demuestra que no es regular, pues no se puede dividir en xyzxyzxyz cumpliendo las condiciones del lema sin romper la estructura del lenguaje.

- Explicar las propiedades de cerradura de lenguajes regulares con un ejemplo.

Los lenguajes regulares son cerrados bajo operaciones como la unión, concatenación, y estrella de Kleene. Esto significa que si aplicas cualquiera de estas operaciones a lenguajes regulares, el resultado también será un lenguaje regular.

Ejemplo: Si L1L_1L1 y L2L_2L2 son lenguajes regulares, entonces su unión L1∪L2L_1 \cup L_2L1∪L2 también es regular.

- Explicar las propiedades de decisión de lenguajes regulares con un ejemplo.

Las propiedades de decisión se refieren a la capacidad de determinar si ciertas condiciones son verdaderas para lenguajes regulares, como la pertenencia de una cadena a un lenguaje, o si dos lenguajes son equivalentes. Estas decisiones pueden resolverse de manera efectiva utilizando autómatas finitos.

Ejemplo: Se puede decidir si una cadena pertenece a un lenguaje regular construyendo un autómata finito para ese lenguaje y verificando si el autómata acepta la cadena.

- Explicar el proceso de determinación de equivalencias entre estados y lenguajes regulares con un ejemplo

El proceso de determinar equivalencias entre estados se basa en la minimización de autómatas finitos deterministas (DFA). Dos estados son equivalentes si, para cualquier cadena de entrada, llevan al mismo estado final (aceptación o rechazo). Se puede usar la partición de estados para identificar y combinar estados equivalentes, simplificando el DFA.

Decimos que dos estados p y q son equivalentes si para todas las entradas w, ˆδ (p, w) es un estado final si y solo si ˆδ (q, w) es un estado final también. Si dos estados no son equivalentes entonces decimos que sin distinguibles. Lo que quiere decir que el estado p es distinguible del estado q si existe al menos una cadena w tal que uno de ˆδ (p, w) y ˆδ (q, w) es un estado final y el otro no. Para encontrar los estados que son equivalentes es necesario hacer un esfuerzo por encontrar r los pares de estados que son distinguibles. Si esta tarea se realiza correctamente mediante el algoritmo llamado" llenado de tabla", entonces cualquier par de estados que no encontremos es equivalente. Este algoritmo consiste en un descubrimiento recursivo de pares distinguibles en un DFA A = (Q, Σ, δ, q0, F). BASE: si p es un estado final y q es no final, entonces el par {p, q} es distinguible. INDUCCION: ´ p y q son estados que ara algún símbolo de entrada a, r = ˆδ(p, a) y s = ˆδ(q, a) son un par de estados que se sabe son distinguibles. Entonces {p, q} son un par de estados distinguibles. La razón por la que esta regla tiene sentido es que debe de existir alguna cadena w que distingue entre r y

s; lo que significa que uno de ˆδ(r, w) y ˆδ(s, w) es final. Entonces la cadena aw debe distinguir p de q ya que ˆδ(p, aw) y ˆδ(q, w) son el mismo par de estados que ˆδ(r, w) y ˆδ(s, w). De lo anterior se desprende el siguiente teorema. Teorema 2 Si dos estados no se distinguen por el algoritmo de llenado de tabla, entonces los estados son equivalentes. La figura 2 muestra el resultado del algoritmo de llenado de tabla

sobre el autómata de la figura 1. En la tabla las x indican los pares de estados distinguibles, y los cuadros en blanco indican que ese par es equivalente. Inicialmente no hay x en la tabla. Para el caso base, ya que C es el único estado final marcamos con x cada par que inicia a C. Ahora que tenemos algunos estados distinguibles podemos descubrir otros. Por ejemplo, ya que {C, H} es distinguible, y los estados E y F van a H y C respectivamente al recibí la entrada 0, entonces sabemos que {E, F} es también un par distinguible. Todos los pares de la tabla se pueden encontrar de manera similar a excepción de {A, G}. Para encontrar este par hay que observar que al recibir la entrada 1 pasan a ser F y E respectivamente, y hemos establecido con anterioridad que {E, F} son distinguibles.

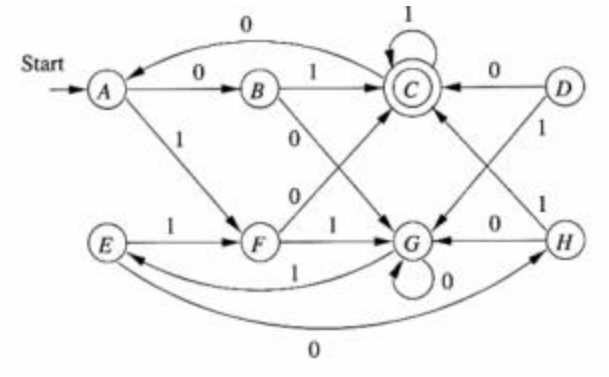


Figure 1: Un autómata con estados equivalentes.

	A	B	C	D	E	F	G
B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x

Figure 2: Tabla de inequivalencias de estados.

- Explicar el proceso de minimización de DFA

Minimizar un DFA implica reducir el número de estados del autómata sin cambiar el lenguaje que reconoce. Esto se logra identificando estados equivalentes y combinándolos, lo que resulta en un autómata más eficiente pero funcionalmente idéntico.

Estas explicaciones resumen conceptos importantes sobre analizadores léxicos y lenguajes regulares, basadas en fuentes confiables y académicas

Esto significa que por cada DFA podemos encontrar otro DFA equivalente que tenga tan pocos estados como cualquier otro DFA que acepte el mismo lenguaje. Los siguientes teoremas nos permitirán definir un algoritmo para minimizar DFA.

Teorema 3 *La equivalencia de estados es transitiva, lo que quiere decir que si en algún DFA A = (Q, Σ, δ, q0, F) encontramos que los estados p y q son equivalentes, y encontramos también que los estados q y r son equivalentes, entonces p y r tienen que ser equivalentes.*

Teorema 4 *Si creamos para cada estado q de un DFA un bloque que consista de q y de todos sus estados equivalentes, entonces los diferentes bloques de estados forman una partición del conjunto de estados. Esto es, cada estado esta exactamente en un bloque. Todos los miembros del bloque son equivalentes, y ningún par de los estados escogidos de diferentes bloques son equivalentes.*

Con esta información el algoritmo de minimización de n DFA A = (Q, Σ, δ, q0, F) es como sigue:

1. Usar el algoritmo de llenado de tabla para encontrar todos los pares de estados equivalentes.

2. Particionar el conjunto de estados Q en bloques de estados mutuamente equivalentes.

3. Construir el DFA minimizado usando los bloques como estados.

Conclusión

La teoría de lenguajes regulares y autómatas es fundamental en la informática, especialmente en el diseño de compiladores y la comprensión de algoritmos eficientes. El analizador léxico, como primer componente de un compilador, convierte el código fuente en tokens, permitiendo una interpretación más sencilla por las siguientes fases del compilador. Los lenguajes regulares, descritos por expresiones regulares y reconocidos por autómatas finitos, presentan propiedades robustas, como la cerradura bajo operaciones como la unión, concatenación y complemento, lo que asegura que cualquier combinación de lenguajes regulares sigue siendo regular. Además, las propiedades de decisión permiten verificar características clave, como la pertenencia de una cadena o la equivalencia entre lenguajes. El Lema de Bombeo nos ayuda a identificar lenguajes que no son regulares, mostrando las limitaciones de estos sistemas. Finalmente, la minimización de DFA asegura que los autómatas sean lo más simples y eficientes posible, sin perder su capacidad para reconocer un lenguaje. En conjunto, estos conceptos son esenciales para optimizar y comprender la estructura subyacente de los lenguajes formales y sus aplicaciones prácticas en la computación.

REFERENCIAS

Sánchez Dueñas, G. y Valverde Andreu, J. A. (2022). Compiladores e intérpretes: un enfoque pragmático: (1 ed.). Ediciones Díaz de Santos. Recuperado de https://elibro.net/es/ereader/uachiapas/269960?page=58.

Vega Castro, R. A. (2008). Compilador de pseudocódigo como herramienta para el aprendizaje en la construcción de algoritmos: (1 ed.). Madrid, Bubok Publishing S.L. Recuperado de https://elibro.net/es/ereader/uachiapas/260295?page=27.

Ciencias computacionales Propedeutico: Autómatas Propiedades de los Lenguajes Regulares. (n.d.). Retrieved August 18, 2024, from https://posgrados.inaoep.mx/archivos/PosCsComputacionales/Curso_Propedeutico/Automatas/04_Automatas_PropiedadesLenguajesRegulares/CAPTUL1.PDF

George B. Dantzig. Reminiscences about the origins of linear programming. Operations Research Letters, 1(2):43–48, 1982.

(2012). CONCEPTOS BÁSICOS DEL ANÁLISIS LÉXICO: (1 ed.). a Universidad Europea de Madrid, S.L.U. Recuperado de https://www.cartagena99.com/recursos/alumnos/apuntes/ININF2_M4_U2_T1.pdf