UNIVERSIDAD AUTÓNOMA DE CHIAPAS, TUXTLA GUTIÉRREZ A
2024-01-29 FACULTAD DE SISTEMAS.
INGENIERÍA EN DESARROLLO Y TECNOLOGÍAS DE SOFTWARE.

SEXTO SEMESTRE.

ALUMNO GABRIEL HASSAN BRUNO SANCHEZ
MATRICULA: A210483

MATERIA: COMPILADORES
ACTIVIDAD: ACTIVIDAD 1.1
SUBCOMPETENCIA: SUBCOMPETENCIA 1 ANÁLISIS LÉXICO.
PROFESOR: DR. LUIS GUTIÉRREZ ALFARO

ANALIZADOR LÉXICO Y LENGUAJE REGULAR.

ÍNDICE:

INTRODUCCIÓN:

En ésta investigación se desarrollan los conceptos y funcionalidad del analizador léxico como herramienta de programación. Se habla de los componentes léxicos que forman parte de éste, los patrones y los lexemas así como también se toca el tema de lenguajes regulares en el cual se explica el lema de bombeo, las propiedades de cerradura, las propiedades de decisión y el proceso de equivalencia entre estados y lenguajes regulares así como el proceso de minimización de DFA.

DESARROLLO:

Funciones del analizador léxico:

La principal función del analizador léxico es leer los caracteres de entrada y crear como salida una secuencia de componentes léxicos que posteriormente serán utilizados por el analizador sintáctico para su análisis.
En éstos sentidos podemos decir que el analizador léxico es la primera fase de un compilador.
Ésta interacción se lleva a cabo convirtiendo al analizador léxico como una subrutina del analizador sintáctico recibiendo la orden de leer la secuencia de carácteres de entrada hasta que sea capaz de identificar el siguiente componente léxico.

Funciones secundarias del analizador léxico:

El analizador léxico al ser el componente que lee el texto fuente éste puede llevar a cabo ciertas acciones secundarias como eliminar carácteres innecesarios como los espacios en blanco o comentarios. También es capaz de relacionar los códigos de error con el programa fuente de manera que se pueda asociar el error con una línea de código específica.
En algunos programas el analizador léxico se encarga de hacer una copia del código fuente con los errores ya identificados, de igual manera “Si el lenguaje fuente es la base de algunas funciones de pre procesamiento de macros, entonces esas funciones del

preprocesador también se pueden aplicar al hacer el análisis léxico. “(2.1. Función Del Analizador Léxico, s. f.)

Componentes léxico, patrones y lexema:

Token:

Es un par de datos que se componen por el nombre de token y un atributo de información que es opcional. El nombre de token es la representación abstracta que representa un tipo de unidad léxica. Como ejemplo podríamos decir el identificador de algún operador: “+”

Patrón:

Decimos que es la descripción de la secuencia o forma que puede tomar un lexema. En el caso de una palabra clave el patrón es sólo la secuencia de carácteres que conforman a la palabra clave, pero en otros casos el patrón puede convertirse en una secuencia bastante compleja.

Lexema:

Es una secuencia de carácteres que el analizador léxico identifica como una instancia de un token, puesto que dicha secuencia encaja con el patrón de dicho token.

LENGUAJE REGULARES:

Lema de bombeo:

El Lema de bombeo se utiliza para demostrar que un lenguaje NO es regular. Más éste no demuestra que que un lenguaje sí es regular o suficiente. Es decir, pasar o cumplir con el lema de bombeo no es más que un requisito a cumplir para un lenguaje que si es regular y suficiente, para demostrar que un lenguaje si es regular se debe proceder con el AFD que sí lo demuestra. Fallar en el Lema de bombeo demuestra que un lenguaje no puede ser aceptado por un autómata finito determinístico.

se define como:

Sea L un conjunto regular, entonces existe un $n \in N$ tal que $\forall\ z \in L$, si $|z|=n$, entonces z se puede expresar de la forma z = uvw donde:

$$|uv| \leq n$$

$|v|=1$

$\forall\ i \geq 0\ uv^iw \in L$

además n puede ser el número de estados de cualquier autómata que acepte el lenguaje L.

pasos a seguir:

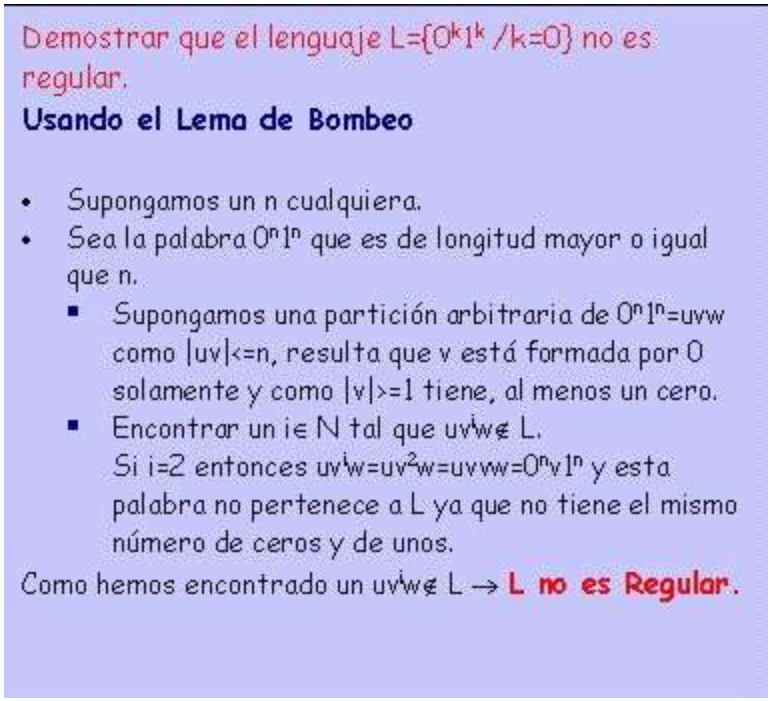
- Suponer un $n \in N$ arbitrario, que se supone que cumple las condiciones del lema.
- Encontrar una palabra que puede depender de n, de longitud mayor o igual que n para la que sea imposible encontrar una partición como la del lema. Para demostrar que una palabra z no cumple las condiciones del lema lo que hay que hacer es:

- Suponer una partición arbitraria de x, $x=uvw$ tal que $|uv| \leq n$, $|v|=1$

- Encontrar un $i \in N$ tal que $uv^iw \notin L$

Ejemplo:

1.- Demostrar que el lenguaje $L=\{0k1k /k=0\}$ no es regular.



Propiedades de cerradura en lenguajes regulares:

- Unión: L ∪ M lenguajes con cadenas en L, M o en ambos.
- Intersección: L ∩ M lenguajes con cadenas en ambos.
- Complemento: L cadenas que no están en L.
- Diferencia: L\M o L − M.
- Inversion: L^R t = {w^R : w ∈ L}
- Cerradura: L*
- Concatenación: L.M
- Homomorfismo (substitución): h(L) = {h(w) ∈ L}h es un homomorfismo.
- Homomorfismo inverso (sustitución inversa): h −1 (L) = {w ∈ Σ : h(w) ∈ L, h : Σ →} es un homomorfismo
- Unión: la unión de lenguajes regulares es regular. Sea L = L(E) y M = L(F). Entonces L(E + F) = L ∪ M , por la definición de “+” en RE.
- Complemento: Si L es un lenguaje regular sobre Σ, entonces también lo es L = Σ∗ L. Todos los estados son de aceptación excepto los F.

Ejemplo:

Sea L definido por el siguiente DFA (el lenguaje de cadenas que terminan en 01):

- Las cadenas de un número diferente de 0’s y 1’s es difícil probarlo con el pumping lemma. Sin embargo, ya probamos que L = 0n1 n no es regular. M = 0n1 m, n 6= m es L. Como L no es regular su complemento tampoco.
- Intersecci´on: Si L y M son regulares, entonces tambi´en L ∩ M. Usando las leyes de Morgan: L ∩ M = L ∪ M.

- Para esto también se puede construir un autómata que simula AL y AM en paralelo y llega a un estado de aceptación si AL y AM también lo hacen.

Propiedades de decisión de lenguajes regulares:

Algunas de las preguntas que nos podemos hacer acerca de lenguajes son si el lenguaje es vacío, si una cadena particular pertenece al lenguaje o si dos descripciones definen el mismo lenguaje. También podemos cuestionarnos acerca del costo computacional requerido para resolver estas preguntas o por ejemplo el requerido para hacer la conversión entre una representación a otra.

Análisis de complejidad y ejemplo:

Transformar un -NFA a un DFA:

- Si el -NFA tiene n arcos. Para calcular ECLOSE(p) se requieren seguir a lo más en 2 arcos y el DFA tiene a lo más 2n estados.
- Para cada símbolo y cada subconjunto el calcular la función de transición para todos los estados, requiere a lo más de 3 pasos, lo cual nos da una complejidad total de O(n3 2n).
- En general el número de estados del DFA es de orden lineal (digamos s), por lo que en la práctica la complejidad se reduce a O(n 3 s).
- Decidir si un lenguaje es vacío: el probar si existe un camino entre un estado inicial a uno final o de aceptación, es simplemente un problema de ver si un nodo está conectado en un grafo, lo cual tiene una complejidad de O(n 2).
- Probar por pertenencia a un lenguaje regular: ver si una cadena es miembro del lenguaje. Si la cadena w es de longitud n para un DFA, esto es de complejidad O(n). Si es un NFA de s estados, entonces la complejidad es: O(ns2). Si es un -NFA entonces la complejidad es O(ns3).

Proceso de determinación de equivalencias entre estados:

(información tomada de Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2006). Automata theory, languages, and computation. International Edition, 24.)

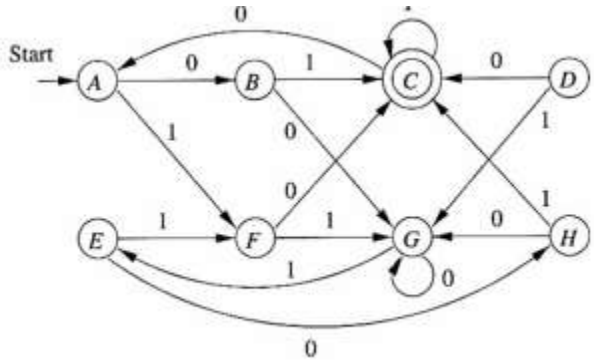
Lo que queremos saber es si dos autómatas diferentes definen el mismo lenguaje.

Primero definiremos lo que son estados equivalentes.

Dos estados p y q dentro de un autómata son *equivalentes*:
$p \equiv q \Leftrightarrow \forall w \in \Sigma^* : \hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$.

Si no, entonces se dice que son *distinguibles*. Osea que p y q son distinguibles si: $\exists w : \hat{\delta}(p, w) \in F \wedge \hat{\delta}(q, w) \notin F$ o viceversa.

ejemplo:



$$\hat{\delta}(C,\epsilon) \in F, \hat{\delta}(G,\epsilon) \notin F \Rightarrow C \neq G$$
$$\hat{\delta}(A,01) = C \in F, \hat{\delta}(G,01) = E \notin F \Rightarrow A \neq G$$
$$\hat{\delta}(A,\epsilon) = A \notin F, \hat{\delta}(E,\epsilon) = E \notin F$$
$$\hat{\delta}(A,1) = F = \hat{\delta}(E,1) \therefore \hat{\delta}(A,1x) = \hat{\delta}(E,1x) = \hat{\delta}(F,1x)$$
$$\hat{\delta}(A,00) = G = \hat{\delta}(E,00)$$
$$\hat{\delta}(A,01) = C = \hat{\delta}(E,01)$$
$$\therefore A \equiv E$$

otra forma

También podemos encontrar los pares equivalentes usando el algoritmo de llenado de tabla (*table-filling algorithm*).

Base: Si $p \in F \wedge q \neq F \Rightarrow p \neq q$

Inducción: Si $\exists a \in \Sigma : \delta(p,a) \neq \delta(q,a) \Rightarrow p \neq q$

Por ejemplo, para el DFA anterior:

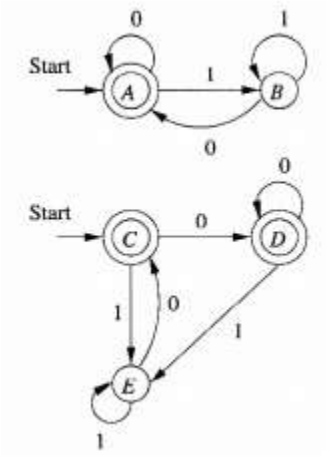
	A	B	C	D	E	F	G
B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x

Sea L y M dos lenguajes regulares (dados en alguna forma).

Convierte L y M a DFA's

Junta los dos DFA's

Prueba si los dos estados iniciales son equivalentes, en cuyo caso $L = M$. Si no son equivalentes, entonces $L \neq M$.



Los dos DFA's aceptan: $L(\epsilon + (0+1)^*0)$. Si los consideramos a los dos como un solo autómata, el algoritmo de llenado de tabla nos da:

B	x			
C		x		
D		x		
E	x		x	x
	A	B	C	D

Lo que nos deja los siguientes pares de estados equivalentes: $\{A, C\}, \{A, D\}, \{C, D\}$ y $\{B, E\}$. Como A y C son equivalentes, entonces los dos autómatas son equivalentes.

MINIMIZACIÓN DE UN DFA

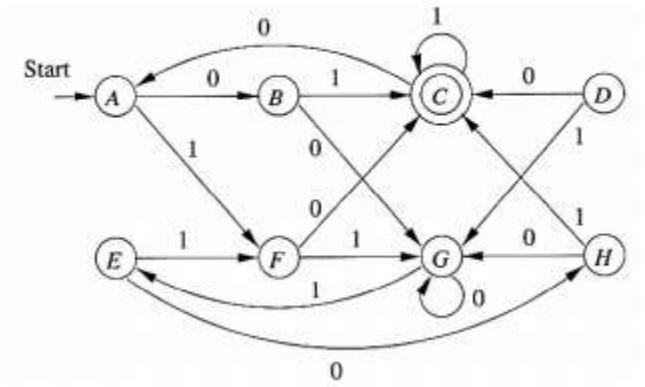
Una consecuencia de encontrar estados equivalentes, es que podemos reemplazar los estados equivalentes por un solo estado representado por su unión.

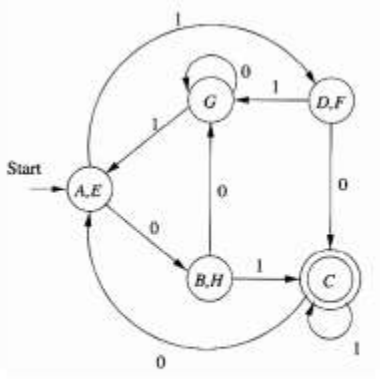
Por otro lado, la relación de equivalencia cumple con ser reflexiva, simétrica y transitiva.

La prueba de equivalencia de estados es transitiva, osea que si encontramos que p y q son equivalentes y q y r son equivalentes, entonces p y r son equivalentes.

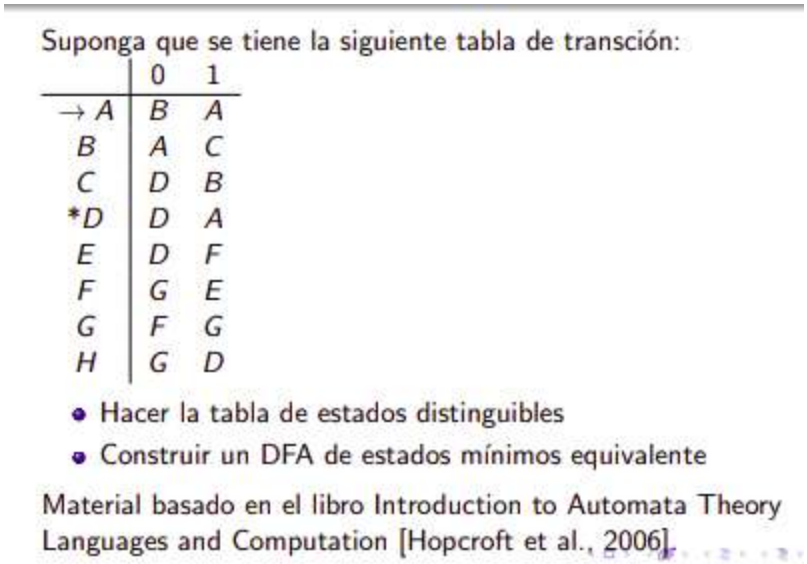
ejemplo

Minimizar el siguiente autómata usando los estados equivalentes.





Nota: no podemos aplicar el algoritmo de llenado de tabla a un NFA.



Suponga que se tiene la siguiente tabla de transción:

	0	1
$\rightarrow A$	B	A
B	A	C
C	D	B
$*D$	D	A
E	D	F
F	G	E
G	F	G
H	G	D

- Hacer la tabla de estados distinguibles
- Construir un DFA de estados mínimos equivalente

Material basado en el libro Introduction to Automata Theory Languages and Computation [Hopcroft et al., 2006].

CONCLUSIONES:

De ésta forma se han desarrollado los conceptos propuestos en una investigación que satisface con lo requerido y presenta en ella información confiable y nos permite comprender el espectro de los analizadores léxicos y sintácticos, su funcionamiento y utilidad así como retomar las bases de las expresiones regulares y sus lenguajes, base que determinará la correcta construcción de un analizador léxico y nos permite entender de esa forma las características de sus propiedades como lo son los tokens, los patrones y los lexemas.

BIBLIOGRAFÍA:

Se utilizó APA 7ma edición.

2.1. Función del Analizador Léxico. (s. f.).

http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro32/autocontenido/autoc

on/21_funcin_del_analizador_lxico.html

LEMA DE BOMBEO PARA LOS LENGUAJES REGULARES. (s. f.).

https://ccia.ugr.es/~rosa/tutormc/teoria/LEMA%20DE%20BOMBEO.htm

Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2006). Automata theory, languages, and computation. International Edition, 24.