UNIVERSIDAD AUTÓNOMA DE CHIAPAS

FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN
CAMPUS 1

Licenciatura en Ingeniería en Desarrollo y Tecnologías de Software

Act. 1.1 Investigar analizador Léxico y Lenguajes Regulares

6 “M”

Materia: Compiladores

Docente: Luis Gutiérrez Alfaro

ALUMNO:
A 211387
Steven de Dios Montoya Hernández

TUXTLA GUTIÉRREZ, CHIAPAS
Sábado, 19 de Agosto de 2023, 23:59

Introducción

En los campos de la teoría informática y la lingüística, los lenguajes regulares son una categoría importante para describir patrones en cadenas. Estos lenguajes tienen ciertas propiedades matemáticas y teóricas que los hacen útiles para construir analizadores léxicos, compiladores y otras aplicaciones relacionadas con el procesamiento formal del lenguaje. Examinamos algunos de los aspectos importantes de los lenguajes regulares, desde el análisis léxico hasta las propiedades de decisión y las operaciones de cierre. Dividir el código en elementos básicos llamados tokens simplifica el proceso de traducción y análisis posterior. Este importante componente es especialmente relevante para aquellos que desean comprender el funcionamiento interno de los lenguajes de programación y cómo implementarlos de manera eficiente.

Funciones del analizador léxico.

El analizador léxico es la primera fase de un compilador.

Su principal función consiste en leer los caracteres de entrada y elaborar como salida una secuencia de componentes léxicos que utiliza el analizador sintáctico para hacer el análisis. Esta interacción, suele aplicarse convirtiendo al analizador léxico en una subrutina o corrutina del analizador sintáctico. Recibida la orden "obtén el siguiente componente léxico" del analizador sintáctico, el analizador léxico lee los caracteres de entrada hasta que pueda identificar el siguiente componente léxico.

Funciones secundarias.

El analizador léxico es la primera fase de un compilador.

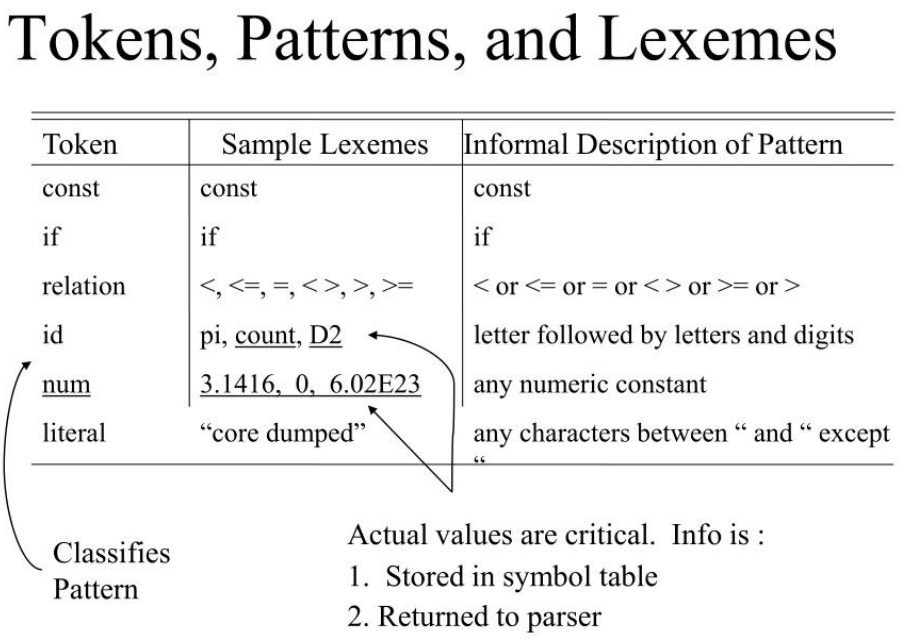
Su principal función consiste en leer los caracteres de entrada y elaborar como salida una secuencia de componentes léxicos que utiliza el analizador sintáctico para hacer el análisis. Esta interacción, suele aplicarse convirtiendo al analizador léxico en una subrutina o corrutina del analizador sintáctico. Recibida la orden "obtén el siguiente componente léxico" del analizador sintáctico, el analizador léxico lee los caracteres de entrada hasta que pueda identificar el siguiente componente léxico.

Componentes léxicos, patrones y lexema.

Un token es un par que consiste en un nombre de token y un valor de atributo opcional. El nombre del token es un símbolo abstracto que representa un tipo de unidad léxica; por ejemplo, una palabra clave específica o una secuencia de caracteres de entrada que denotan un identificador. Los nombres de los tokens son los símbolos de entrada que procesa el analizador sin táctico. A partir de este momento, en general escribiremos el nombre de un token en negrita. Con frecuencia nos referiremos a un token por su nombre

Un patrón es una descripción de la forma que pueden tomar los lexemas de un token. En el caso de una palabra clave como token, e l patrón es sólo la secuencia de caracteres que forman la palabra clave. Para los identificadores y algunos otros tokens, el patrón es una estructura más compleja que se relaciona mediante muchas cadenas.

Un lexema es una secuencia de caracteres en el programa fuente, que coinciden con el patrón para un token y que el analizador léxico identifica como una instancia de ese token.



Tema Lenguajes Regulares.

Lenguaje regular. En Lingüística, Matemáticas e Informática y en la jerarquía de Chomsky se refiere a los lenguajes de tipo 3, aquellos que pueden representarse mediante gramáticas regulares, autómatas finitos o expresiones regulares.

Son los lenguajes formales más simples, con los mecanismos de representación y reconocimiento más estudiados. Su aplicación práctica en la teoría y construcción de intérpretes y compiladores de lenguajes de programación o de especificación o formato de información, especialmente como microcomponentes del analizador lexicográfico que detecta los tókenes como constantes numéricas, cadenas de texto, operadores, palabras reservadas (keywords), separadores, etc. Pero también se puede apreciar su uso en máquinas expendedoras, teléfonos públicos, calculadoras y otros artefactos electromecánicos.

Al lenguaje generado por medio de una gramática regular. Son aquellos lenguajes cuyas cadenas están formadas por la concatenación de símbolos, en las cuales no hay relación entre una parte de la cadena y otra parte de la cadena.

Vemos que una gramática regular o de tipo 3 es aquella gramática donde las reglas de producción siguen la siguiente estructura: $A \rightarrow uB$ o $A \rightarrow u$ donde $u \in T^*$ y $A,B \in V$

Al lenguaje generado por medio de una gramática regular. Son aquellos lenguajes cuyas cadenas están formadas por la concatenación de símbolos, en las cuales no hay relación entre una parte de la cadena y otra parte de la cadena.

Vemos que una gramática regular o de tipo 3 es aquella gramática donde las reglas de producción siguen la siguiente estructura: $A \rightarrow uB$ o $A \rightarrow u$ donde $u \in T^*$ y $A,B \in V$

Los “lenguajes regulares” es la clase más pequeña, e incluye a los lenguajes más simples. Por ejemplo, el conjunto de todos los números binarios. Los “lenguajes libres de contexto” incluye a los LR. Por ejemplo, la mayoría de los lenguajes de programación. Los “lenguajes recursivamente enumerables” que incluyen a los dos anteriores

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

El Lema de Bombeo para lenguajes regulares dice que cualquier palabra suficientemente larga en un lenguaje regular contiene una sección que se puede repetir o eliminar y la palabra resultante sigue perteneciendo al lenguaje

5.2.2. Aplicación del lema de bombeo

Algo que se podría hacer con el lema de bombeo es usarlo para demostrar que un lenguaje es regular de la siguiente forma:

1. Identificar un lenguaje regular L
2. Escoger una longitud n para xy
3. Proponer una forma de cadena que dependa de n pero que sea mayor que n
4. Particionar la cadena w en xyz
5. Verificar que la partición cumpla con las restricciones:
    1. $xyz = w$
    2. $|xy| \leq n$
    3. $y \neq \varepsilon$
6. Generar cadenas bombeadas $xy^kz \in L$

Si se cumple para todas las formas de la cadena del lenguaje L podríamos concluir que es un lenguaje regular.

Cómo realmente usamos el lema de bombeo #

Si tratáramos de demostrar que un lenguaje es regular tendríamos que enumerar todas las formas y para todas demostrar que al bombear la cadena pertenece al lenguaje; esto no va a ser directo porque tendríamos que demostrar que tenemos una lista de todas las formas posibles de las cadenas del lenguaje, regresando a nuestro problema original. En realidad no usamos el lema de bombeo para demostrar que un lenguaje es regular… ¿Entonces para qué lo usamos? Lo usamos para demostrar que un lenguaje no es regular, vamos a partir de qué es regular y si para una de las formas del lenguaje regular no se cumple el lema de bombeo, habremos demostrado que un lenguaje no es regular.

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

Una propiedad de cerradura de una clase de lenguajes establece que lenguajes en dicha clase, una operación (e.g., unión) entre ellos produce otro lenguaje en la misma clase.

Cerradura de Union (A u B):La propiedad establece que la union de 2 lenguajes regulares tambien es un lenguaje regular.

Ejemplo: A={a,aa} y B=[b,bb].

Cerradura de Concatenación (A · B):establece que la concatenación de dos lenguajes regulares también es un lenguaje regular.
Ejemplo: Tomemos los lenguajes regulares A = {0, 1} y B = {00, 11}.

Cerradura de Concatenación (A · B):establece que la concatenación de dos lenguajes regulares también es un lenguaje regular.
Ejemplo: Tomemos los lenguajes regulares A = {0, 1} y B = {00, 11}.

Cerradura de Estrella (A):*establece que el cierre de Kleene de un lenguaje regular también es un lenguaje regular. El cierre de Kleene de un lenguaje A consiste en todas las posibles concatenaciones finitas de cadenas en A, incluyendo la cadena vacía ε.
Ejemplo:
A = {a, b}. El cierre de Kleene A* sería {ε, a, b, aa, ab, ba, bb, aaa, ...}, que también es un lenguaje regular.

Cerradura de Complemento (¬A):Esta propiedad establece que el complemento de un lenguaje regular A también es un lenguaje regular.
Ejemplo: Consideremos el lenguaje regular A = {00, 11}. Su complemento ¬A sería {ε, 0, 1, 01, 10, 001, 010, 011, 100, 101, 110, 111, ...}, que también es un lenguaje regular.

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

Propiedad de Pertinencia: Esta propiedad verifica si una cadena específica pertenece a un lenguaje regular.
Ejemplo: Lenguaje regular A = {dog, cat, bird}. Para verificar si la cadena "cat" pertenece a A, observamos sus elementos y confirmamos que efectivamente pertenece.

Propiedad de Vacío: Esta propiedad de decisión verifica si un lenguaje regular es vacío, es decir, no contiene ninguna cadena.
Ejemplo: Lenguaje regular A = {a, b}. Para verificar si A es vacío, simplemente comprobamos si contiene alguna cadena. Dado que A contiene las cadenas a y b, no está vacío.

Propiedad de Igualdad: Esta propiedad verifica si dos lenguajes regulares son iguales, es decir, contienen exactamente las mismas cadenas.
Ejemplo: Lenguaje regular A = {0, 1} y lenguaje regular B = {1, 0}. Para verificar si A = B, comparamos sus contenidos. Ambos lenguajes contienen las mismas cadenas, por lo que A = B.

Las propiedades de decisión son características que se aplican a lenguajes regulares y que permiten tomar decisiones algorítmicas sobre ciertas propiedades de estos lenguajes

Una propiedad de decisión para una clase de lenguajes es un algoritmo que toma una descripción formal del lenguaje (e.g., un DFA) y nos dice cuándo cierta propiedad de ese lenguaje se cumple o no

Propiedad de Subcadena: Esta propiedad verifica si un lenguaje regular contiene una subcadena específica.
Ejemplo: Lenguaje regular A = {aa, ab, ba, bb}. Para verificar si A contiene la subcadena "ab", observamos sus elementos y vemos que efectivamente contiene "ab".

Propiedad de Subcadena: Esta propiedad verifica si un lenguaje regular contiene una subcadena específica.
Ejemplo: Lenguaje regular A = {aa, ab, ba, bb}. Para verificar si A contiene la subcadena "ab", observamos sus elementos y vemos que efectivamente contiene "ab".

Propiedad de Prefijo: Esta propiedad verifica si un lenguaje regular contiene algún prefijo específico.
Ejemplo: Lenguaje regular A = {abc, def, gh}. Para verificar si A contiene el prefijo "de", observamos sus elementos y vemos que contiene la cadena "def".

Propiedad de Sufijo: Esta propiedad verifica si un lenguaje regular contiene algún sufijo específico.
Ejemplo: Lenguaje regular A = {abcd, ef, ghij}. Para verificar si A contiene el sufijo "ij", observamos sus elementos y vemos que contiene la cadena "ghij".

Propiedad de Finitud: Esta propiedad verifica si un lenguaje regular es finito, es decir, contiene una cantidad finita de cadenas.
Ejemplo: Lenguaje regular A = {red, blue, green}. Para verificar si A es finito, contamos sus elementos y vemos que contiene tres cadenas, por lo que es finito.

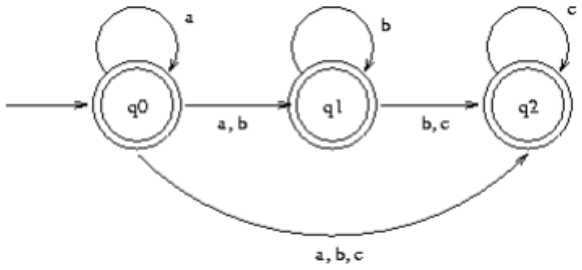
Explicar el proceso de determinación de equivalencias entre estados y lenguajes regulares con un ejemplo

La equivalencia entre AFD y AFN es clara entendiendo todo AFD como un caso particular de un AFN. En el otro sentido, a partir un AFN A=(Q,Σ, δ,q0,F) se puede construir otro AFD A'=(Q',Σ, δ', q0′, F') equivalente (que acepte el mismo lenguaje), de la siguiente forma:

- Q' = 2Q
- q0′ = {q0}
- F' = { q' ∈ Q' | q' ∩ F ≠ ∅ }
- δ'(q',a) = ∪(q ∈ q')δ(q,a) : q' ∈ Q, a ∈ Σ

Ejemplo:

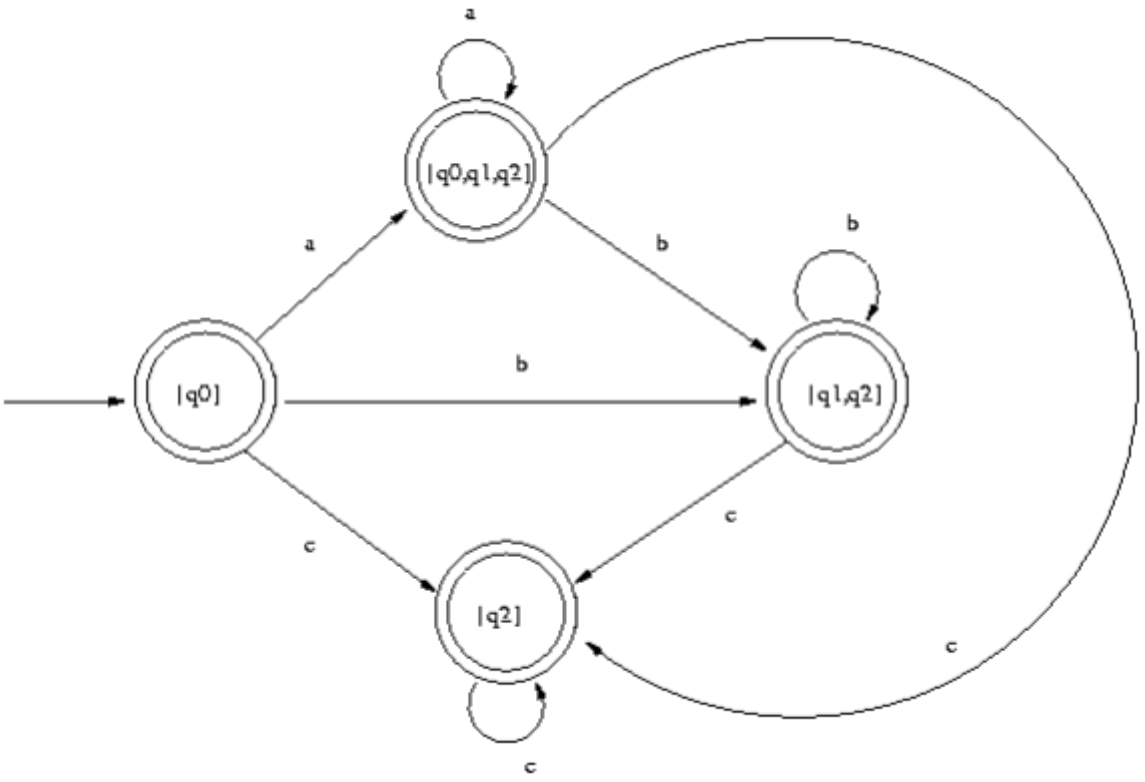
Dado el autómata de la figura 1, el proceso de construcción de un AFD equivalente parte del estado inicial {q0}, y determina el conjunto de estados alcanzables con cada símbolo del alfabeto. De esta forma, por ejemplo, al considerar el símbolo a se alcanzan los estados {q0, q1, q2}.



	a	b	c
$\{q_0\}$	$\{q_0,q_1,q_2\}$	$\{q_1,q_2\}$	$\{q_2\}$
$\{q_0,q_1,q_2\}$	$\{q_0,q_1,q_2\}$	$\{q_1,q_2\}$	$\{q_2\}$
$\{q_1,q_2\}$	$\varnothing$	$\{q_1,q_2\}$	$\{q_2\}$
$\{q_2\}$	$\varnothing$	$\varnothing$	$\{q_2\}$

Cada uno de los conjuntos de estados que aparezcan se considera como uno de los estados del AFD equivalente, determinandose para cada uno de ellos su función de transición. El proceso se repite mientras aparezcan nuevos estados. La figura 2 muestra la tabla de transiciones del AFD.

Tabla de transiciones del AFD equivalente al AFN de la primera figura a partir de esta tabla el diagrama de transiciones queda como muestra la siguiente figura.



Explicar el proceso de minimización de DFA

Minimización de DFA

Supongamos que hay un DFA D < Q, Σ, q0, δ, F > que reconoce un idioma L. Entonces se puede construir el DFA minimizado < Q', Σ, q0, δ', F' > para el idioma L como:

Paso 1: Dividiremos Q (conjunto de estados) en dos conjuntos. Un conjunto contendrá todos los estados finales y otro conjunto contendrá estados no finales. Esta partición se llama P 0 .

Paso 2: Inicialice k = 1

Paso 3: Encuentre P k dividiendo los diferentes conjuntos de P k-1 . En cada conjunto de P k-1 , tomaremos todos los pares de estados posibles. Si dos estados de un conjunto son distinguibles, dividiremos los conjuntos en diferentes conjuntos en P k .

Paso 4: Detener cuando P k = P k-1 (Sin cambios en la partición).

Paso 5: Todos los estados de un conjunto se fusionan en uno. El número de estados en DFA minimizado será igual al número de conjuntos en P k .

Para cada idioma regular, también existe un autómata mínimo que lo acepta, es decir, un DFA con un número mínimo de estados y este DFA es único (excepto que los estados pueden tener diferentes nombres). El DFA mínimo garantiza un costo computacional mínimo para tareas como la coincidencia de patrones . Minimización de DFA.

La minimización de DFA se realiza normalmente en tres pasos: eliminar estados muertos e inalcanzables (esto acelerará el siguiente paso), fusionar estados no distinguibles, opcionalmente, vuelva a crear un solo estado muerto (estado "receptor") si se requiere que el DFA resultante esté completo. Minimización de DFA.

Conclusión:

En resumen, los lenguajes regulares juegan un papel importante en la teoría de la computación y el procesamiento formal del lenguaje. Su investigación proporciona una comprensión más profunda de cómo se describen y manipulan las cadenas a través de patrones y reglas formales. A través de conceptos como análisis léxico, lemas de bomba, propiedades de cierre, propiedades de decisión, equivalencia de estado y minimización DFA, podemos abordar problemas fundamentales en la construcción de compiladores y la resolución de problemas relacionados. Acompañado por el procesamiento del lenguaje. Estos conceptos son fundamentales para el diseño y la implementación efectiva de los sistemas de procesamiento del lenguaje y juegan un papel importante en la optimización y eficiencia de los algoritmos utilizados en estos campos. Un analizador léxico es un punto de partida importante al construir un compilador. y procesamiento formal del lenguaje. La capacidad de identificar y clasificar tokens en el código fuente es fundamental para garantizar la precisión y la eficiencia de la traducción y el análisis. Comprender su función y su impacto en el procesamiento del lenguaje abre la puerta a la mejora de algoritmos, la optimización de programas y una comprensión más profunda de cómo las máquinas interpretan y ejecutan el software.

Bibliografía

En Fundamentos generales de programación, de Luis Joyanes Aguilar, 33-37. México: Mc Graw-Hill, 2013.
Aquilar, Luis joyanes. En Metodologia de la programación Diagramas de flujo, Algoritmos y programación Estructurada, 66-88. Mexico: Mc Graw-Hill, 1987.

En Fundamentos de Programación , de Manuel Santos. Ismael Patiño. Raul Carrasco, 37-40. México : AlfaOmega-Rama, 2006.
Chorda, Gloria de Antonio. Ramon. «Metodologia de la Programación.» 39-41. España: Ra-ma, s.f.

En Lenguaje Ensamblador para Mircrocomputadoras IBM, de J. Terry Godfrey, 73-76. México: Prentice Hall, 1991.

En Introducción a la computación y a la programación Estructurada, de Guillermo Levine, 115-118. Mexico, 1989.

En Sistema Operativos y Compiladores, de Jesus Salas Parilla, 149,150,173,174. México : Mc Graw-Hill, 1992.

Sethi R. Ullman J. Aho A. (1990). En Compiladores, Principios, Técnicas y Herramientas. Adisson-Wesley Iberoamericana, (86-89,94-108,115-131,164-187,190-193,200-201,209-212,221 -226,264-274,443-444.).

En Compiladores e Interpretes. Teoria y Practica, de M Moreno, M de la cruz, A Ortega y E Pulido, 78-85, 191-194, 199-204, 221-223. España: Pearson- Prentice Hall, 2006.

Autómatas, G. y., & de Ingeniería en Informática http://webdiis. unizar. es/asignaturas/LGA, C. C. (s/f). El lema de bombeo para lenguajes regulares. Unizar.es. Recuperado el 17 de agosto de 2023, de https://webdiis.unizar.es/asignaturas/LGA/material_2003_2004/Lema_Bombeo_Regulares.pdf