Unidad 4. Análisis Léxico

Tecnológico Nacional de México Campus Matamoros

Materia: Lenguajes Y Autómatas I

Profesor: María Guadalupe Hernández Compeán

Integrantes:

Nicole Rodríguez González

Víctor Hugo Vázquez Gómez

23 marzo 2020

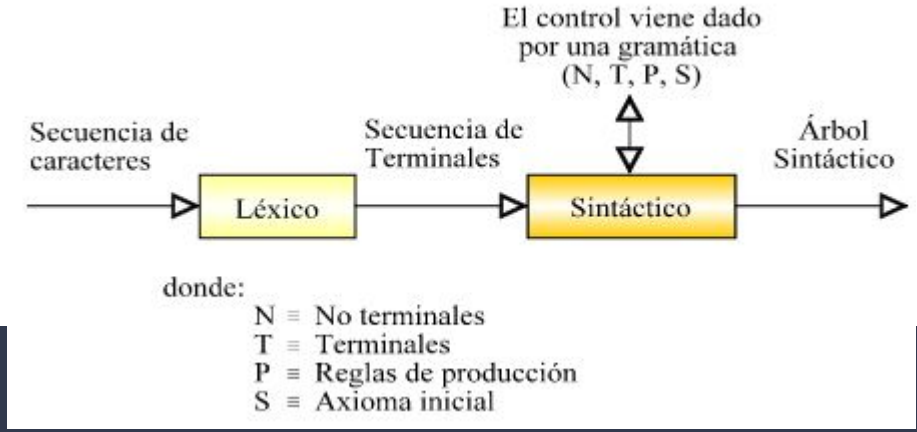
Introducción

Esta es una investigación de la primera fase de un compilador, el analizador léxico. Intentaremos entender las técnicas utilizadas para construir dichos analizadores, investigaremos también sus aplicaciones, qué tipo de errores podemos cometer a la hora de realizar el analizador léxico.

Nos basamos en el libro de compiladores escrito por Sergio Gálvez Rojas y Miguel Ángel Mora Mata. En él nos dice:

El analizador léxico es el que se encarga de buscar componentes léxicos o palabras que componen el programa fuente, según unas reglas o patrones.

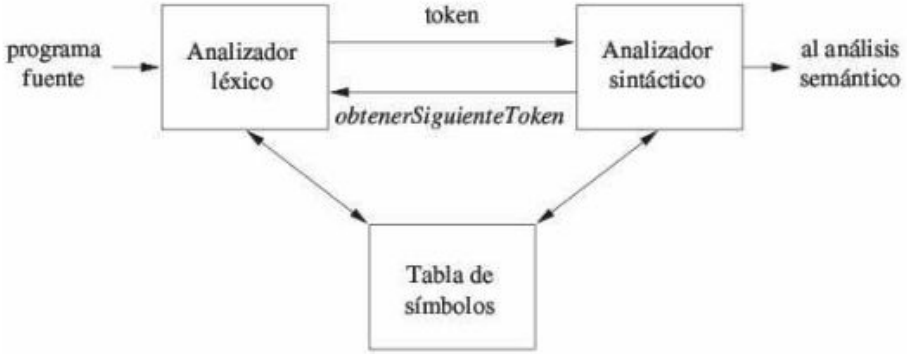
La entrada en sí es una secuencia de caracteres que se encuentra con una codificación tal como la UTF-8 o la ASCII. El analizador lo que hace con esta secuencia es dividir en palabras con significado propio para después convertirlas en una secuencia que el análizador sintáctico pueda entender.



Funciones del analizador léxico

La función del analizador léxico es la de leer los caracteres de la entrada del programa fuente, agruparlos en lexemas y producir una salida de tokens para cada lexema en el programa.

Cuando el analizador léxico descubre un lexema que constituye a un identificador, debe introducir ese lexema a la tabla de símbolos.

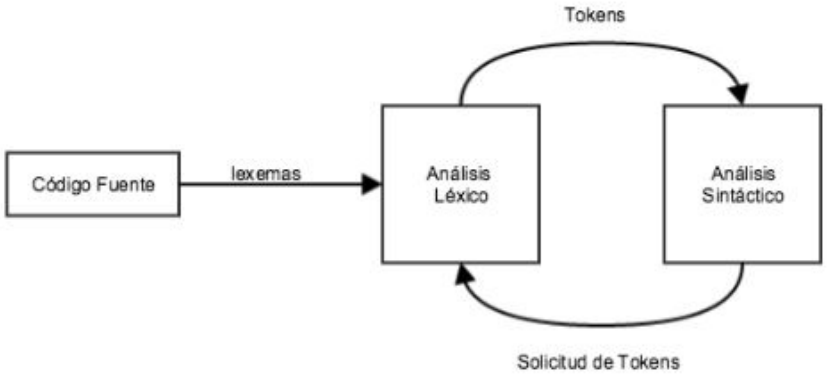


El analizador léxico es la parte del compilador que lee el texto de origen, y como tal realiza otras tareas además de las ya mencionadas. Tales como:

- Eliminar comentarios.
- Eliminar espacios en blanco (como el tabulador, nueva línea u otros caracteres que se usan para separar tokens de entrada).
- Correlacionar mensajes de error generados por el compilador con el programa fuente.

Algunas veces, los analizadores léxicos se dividen en una cascada de dos procesos:

- El *escaneo,* que consiste en los procesos simples que no requieren la determinación de tokens de entrada, como la eliminación de comentarios y la compactación de los caracteres de espacio en blanco consecutivos en uno solo.
- El propio análisis léxico es la porción más compleja, en donde escanear produce la secuencia de tokens como salida.



Componentes léxicos, patrones y lexemas

Desde un punto de vista muy general, podemos abstraer el programa que implementa un análisis léxico gráfico mediante una estructura como:

(Expresión regular)$_1$ {acción a ejecutar} $_1$
(Expresión regular)$_2$ {acción a ejecutar}$_2$
(Expresión regular)$_3$ {acción a ejecutar}$_3$
... ...
(Expresión regular)$_n$ {acción a ejecutar}$_n$

donde cada acción a ejecutar es un fragmento de programa que describe cuál ha de ser la acción del analizador léxico cuando la secuencia de entrada coincida con la expresión regular.

Al hablar sobre el análisis léxico, utilizamos tres términos distintos:

• 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 sintáctico. 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, el 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.

Por ejemplo, si se necesita construir un analizador léxico que reconozca los números enteros, los números reales y los identificadores de usuario en minúsculas, se puede proponer una estructura como:

Expresión Regular	Terminal asociado
$(0 \ldots 9)^+$	NUM_ENT
$(0 \ldots 9)^* \, . \, (0 \ldots 9)^+$	NUM_REAL
(a ... z) (a ... z 0 ... 9)*	ID

Asociado a la categoría gramatical de número entero se tiene el token **NUM_ENT** que puede equivaler, p.ej. al número 280; asociado a la categoría gramatical número real se tiene el token **NUM_REAL** que equivale al número 281; y la categoría gramatical identificador de usuario tiene el token ID que equivale al número 282. Así, la estructura expresión regular-acción sería la siguiente:

$(0 \ldots 9)^+$	{ return 280;}
$(0 \ldots 9)^* \, . \, (0 \ldots 9)^+$	{ return 281;}
(a ... z)* (a ... z 0...9)	{ return 282;}
“ ”	{ }

De esta manera, un analizador léxico que obedeciera a esta estructura, si durante su ejecución se encuentra con la cadena:

95.7 99 cont

intentará leer el lexema más grande de forma que, aunque el texto “95" encaja con el primer patrón, el punto y los dígitos que le siguen “.7" hacen que el analizador decida reconocer “95.7" como un todo, en lugar de reconocer de manera independiente “95" por un lado y “.7" por otro; así se retorna el token **NUM_REAL**. Resulta evidente que un comportamiento distinto al expuesto sería una fuente de problemas. A continuación el patrón “ ” y la acción asociada permiten ignorar los espacios en blanco. El “99" coincide con el patrón de **NUM_ENT**, y la palabra “cont” con **ID**.

Creación de Tabla de Tokens

Existen diferentes tipos de tokens y a cada uno se le puede asociar un tipo y, en algunos casos, un valor. Los tokens se pueden agrupar en dos categorías:

Cadenas específicas, como las palabras reservadas (if, while, ...), signos de puntuación (., ,, =, ...), operadores aritméticos (+,*, ...) y lógicos (AND, OR, NOT, ...), etc. Habitualmente, las cadenas específicas no tienen asociado ningún valor, sólo su tipo.
Cadenas no específicas, como los identificadores o las constantes numéricas o de texto. Las cadenas no específicas siempre tienen tipo y valor. Por ejemplo, si dato es el nombre de una variable, el tipo del token será identificador y su valor será dato.

Consideraciones sobre la tabla de tokens

La tabla de tokens tiene dos funciones principales:

- Efectuar chequeos semánticos.
- Generación de código.

La tabla de tokens puede iniciarse con cierta información útil como:

- **Constantes:** PI, E, etc.
- **Funciones de librería:** EXP, LOG, etc.
- **Palabras reservadas:** Esto facilita el trabajo lexicográfico que tras reconocer un identificador lo busca en la tabla de tokens y si es la palabra reservada devuelve el token asociado.

La tabla de símbolos consta de una estructura llamada símbolo. Las operaciones que puede realizar son:

- **Crear:** Crea una tabla vacía.
- **Insertar:** Parte de una tabla de símbolo y de un nodo, lo que hace es añadir ese nodo a la cabeza de la tabla dado un par clave-valor.
- **Buscar:** Busca el nodo que contiene el nombre que le paso por parámetro, es decir, dada la clave de un elemento, encontrar su valor.
- **Imprimir:** Devuelve una lista con los valores que tiene los identificadores de usuario, es decir recorre la tabla de símbolos. Este procedimiento no es necesario pero se añade por claridad, y a efectos de resumen y depuración
- **Cambio de valor:** Buscar el elemento y cambiar su valor.
- **Borrado:** Eliminar un elemento de la tabla.
- **Longitud de búsqueda** (o tiempo de acceso)

Procedimiento de creación:

El final de la tabla de símbolos se usa como una lista al revés. Cuando aparece {se añade un bloque y se pone el puntero al final de la lista. Cuando llega} se vacía la lista correspondiente al bloque que terminó, copiando al principio de la tabla.

Inserción: se añade a la lista del final de la tabla de símbolos, se corre el puntero y se aumenta en 1 el número de
elementos de ese bloque.
Búsqueda: se busca en el bloque presente sólo (si es una declaración) o en él y sus antepasados en otro caso.

Errores léxicos

Sin la ayuda de los demás componentes es difícil para un analizador léxico saber que hay un error en el código fuente. Por ejemplo, si la cadena **fi** se encuentra por primera vez en un program a en C en el siguiente contexto:

fi (a = f (x)) . . .

un analizador léxico no puede saber si fi es una palabra clave if mal escrita, o un identificador de una función no declarada. Como fi es un lexema válido para el token **id**, el analizador léxico debe regresar el token **id** al analizador sintáctico y dejar que alguna otra fase del compilador (quizá el analizador sintáctico en este caso) mande un error debido a la transposición de las letras.

Posibles acciones de recuperación de errores son:

1. Eliminar un carácter del resto de la entrada.

2. Insertar un carácter faltante en el resto de la entrada.

3. Sustituir un carácter por otro.

4. Transponer dos caracteres adyacentes.

Generadores de analizadores Léxicos.

Aproximaciones para construir un analizador lexicográfico

Hay tres mecanismos básicos para construir un analizador lexicográfico:

- Ad hoc. Consiste en la codificación de un programa reconocedor que no sigue los formalismos propios de la teoría de autómatas. Este tipo de construcciones es muy propensa a errores y difícil de mantener.

- Mediante la implementación manual de los autómatas finitos. Este mecanismo consiste en construir los patrones necesarios para cada categoría léxica, construir sus autómatas finitos individuales, fusionarlos por opcionalidad y, finalmente, implementar los autómatas resultantes.

- Mediante un metacompilador. En este caso, se utiliza un programa especial que tiene como entrada pares de la forma (expresión regular, acción). El metacompilador genera todos los autómatas finitos, los convierte a autómata finito determinista, y lo implementa en C. El programa C así generado se compila y se genera un ejecutable que es el analizador léxico de nuestro lenguaje.

Aplicaciones (Caso de estudio).

Algunas aplicaciones de los analizadores léxicos son:

- El analizador léxico divide la entrada en componentes léxicos.
- Especificamos las categorías mediante expresiones regulares.
- Para reconocer los lenguajes asociados a las expresiones regulares empleamos autómatas de estados finitos(AFD).
- se pueden crear los AFD directamente a partir de la expresión regular.
- El analizador léxico utiliza la máquina discriminadora determinista.
- El tratamiento de errores en nivel léxico es muy simple.
- Se pueden emplear las ideas de los analizadores léxicos para facilitar el tratamiento de ficheros de texto.

Conclusión grupal

Los analizadores léxicos son como una aplicación en la que los compiladores que se encargan de verificar si el texto que se escribió en un formato aceptado para todo el programa que se escribe en ese lenguaje de programación y también se encarga de verificar que tenga coherencia, los analizadores léxicos nos sirven para resolver los problemas que pueden ocurrir a causa de que el programa no esté bien estructurado o esté mal escrito.