UNIVERSIDAD AUTÓNOMA DE CHIAPAS
FACULTAD DE CONTADURÍA Y ADMINISTRACIÓN. CAMPUS I.
LICENCIATURA EN INGENIERÍA EN DESARROLLO Y TECNOLOGÍAS DE SOFTWARE.

COMPILADORES

ACT. 1.3 INVESTIGAR LOS CONCEPTOS DEL ANALIZADOR LÉXICO.

Ana Gabriela Casanova Hernández

Docente:
DR. LUIS GUTIÉRREZ ALFARO.

Tuxtla Gutiérrez, Chiapas. 22 de agosto de 2024

INDICE

Introducción

Un **analizador léxico** es como el primer filtro de un compilador. Su función principal es leer el código fuente de un programa y dividirlo en unidades significativas llamadas **tokens**. Estos tokens son las palabras clave, identificadores, números, operadores y otros símbolos básicos que conforman el lenguaje de programación.

Imagina el código fuente como una oración: el analizador léxico es como el cerebro que descompone esa oración en palabras individuales. Cada palabra (token) tiene un significado específico en el contexto del lenguaje de programación.

1.1Expresiones regulares.

Una expresión regular es un modelo con el que el motor de expresiones regulares intenta buscar una coincidencia en el texto de entrada. Un modelo consta de uno o más literales de carácter, operadores o estructuras.

Ejemplos de sus usos:

- Comandos de búsqueda, e.g., grep de UNIX
- Sistemas de formateo de texto: Usan notación de tipo expresión regular para describir patrones
- Convierte la expresión regular a un DFA o un NFA y simula el autómata en el archivo de búsqueda
- Generadores de analizadores léxicos. Como Lex o Flex.
- Los analizadores léxicos son parte de un compilador. Dividen el programa fuente en unidades lógicas (tokens), como while, n ´umeros, signos (+, −, <, etc.)
- Produce un DFA que reconoce el token

Las expresiones regulares denotan lenguajes. Por ejemplo, la expresión regular: $01* + 10*$ denota todas las cadenas que son o un 0 seguido de cualquier cantidad de 1's o un 1 seguido de cualquier cantidad de 0's.

Operaciones de los lenguajes:

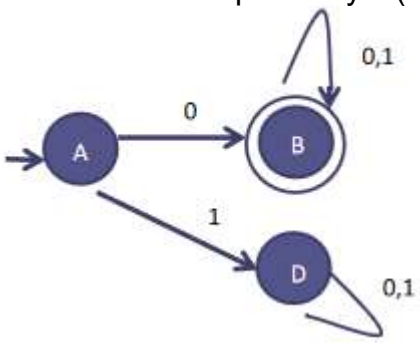
1 Unión: Si L y M son dos lenguajes, su unión se denota por L ∪ M (e.g., L = {11, 00}, M = {0, 1}, LcuoM = {0, 1, 00, 11})

2 Concatenación: La concatenación es: LM o L.M (e.g., LM = {110, 111, 000, 001})

3 Cerradura (o cerradura de Kleene): Si L es un lenguaje su cerradura se denota por: L∗ (L0 = {}, L1 = L, L2 = LL. Si L = {0, 11}, L2 = {00, 011, 110, 1111})

1.2 Autómatas.

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

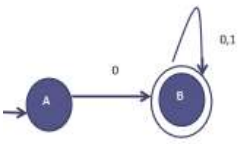
(B,1) = B (D,0) = D (D,1) = D

1.2.1 Autómatas no determinísticos.

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 ∆.

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

SOLUCIÓN:

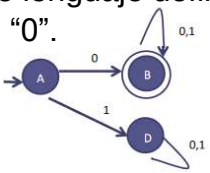


1.2.2 Autómatas determinísticos.

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.

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

SOLUCIÓN:



1.3 Matrices de transición.

Una matriz de transición es una forma de representar la función de transición de un autómata finito. Cada fila de la matriz corresponde a un estado, y cada columna corresponde a un símbolo de entrada. El valor en la celda (i, j) indica el estado al que se pasa desde el estado i al leer el símbolo j.

Ejemplo: Para el ejemplo de la máquina expendedora, la matriz de transición sería:

	10	20
Q0	Q1	Q1
Q1	Q1	Q0

Explicación de la matriz:

- Si estamos en el estado q0 (esperando 10 centavos) y leemos un 10, pasamos al estado q1 (esperando 20 centavos).
- Si estamos en el estado q0 y leemos un 20, también pasamos a q1 (ya que se dispensa la bebida y se vuelve al estado inicial).
- Si estamos en el estado q1 y leemos un 10, nos quedamos en q1 (seguimos esperando 20 centavos).
- Si estamos en el estado q1 y leemos un 20, pasamos a q0 (se dispensa la bebida y se vuelve al estado inicial).

1.4 Tabla de símbolos.

Almacena todos los nombres declarados en el programa y sus atributos (tipo, valor, direcciones, parámetros, etc).

Se usa en las distintas fases del compilador.

ESTRUCTURA DE DATOS:

Almacena información sobre:

Los identificadores.

Las palabras reservadas.

Las constantes.

Contiene una entrada para cada uno de los símbolos definidos en el programa fuente.

Sobre los identificadores, y opcionalmente sobre las palabras reservadas y las constantes.

Información sobre el lexema, tipo de datos, ámbito y dirección en memoria.

Por cada entrada en la tabla de símbolos habrá que guardar:

Lexema correspondiente.

Tipo. (depende de la implementación)

Ámbito. (depende de la implementación)

Dirección de memoria asignada.

Forma. (depende de la implementación).

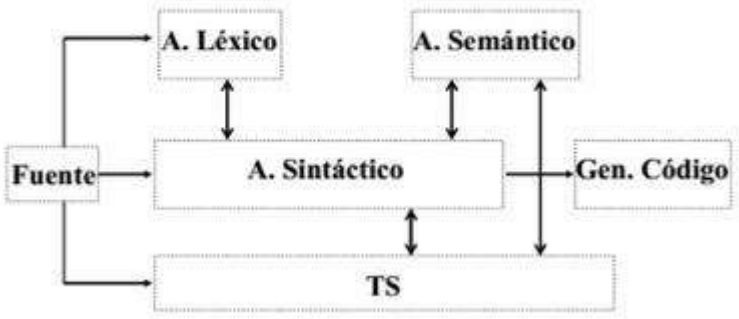
¿UTILIDAD DE LA TABLA DE SÍMBOLOS?

Analizador Léxico: Pasa en el token y la entrada de la TS creada.

Analizador Sintáctico y Semántico: Busca el token y si no lo encuentra crea una nueva entrada.

¿UTILIDAD DE LA TABLA DE SÍMBOLOS? Esquema general

Esquema general



DATOS QUE SE ALMACENAN:

Para un array:

- Tipo de los elementos.

- Número de elementos.
- Límites inferior y superior.

Para una función:

- Número de parámetros.
- Tipo de los parámetros.
- Forma de paso de parámetros.
- Tipo de retorno.

OPERACIONES PRINCIPALES

- **I**nsertar: introduce un símbolo tras una declaración.
- Buscar: recupera información asociada a un símbolo.
- Eliminar: borra la información de un símbolo cuando ya no se utiliza

ELEMPLO DE USO I

Declaración previa al uso de variables:

En las declaraciones, inserción en la TS

Aparición de una variable en una sentencia, búsqueda en la TS:

Si se encuentra Fue declarada

Si no se encuentra Error de compilación.

EJEMPLOS DE USO II

Acceso a una posición de un array:

– Declaración > Inserción en la TS

– Acceso a un array > Búsqueda en la TS

1. Comprobación de tipo array

2. Comprobación acceso a una posición válida:

```
PROGRAM Ejemplo
VAR a,b: INTEGER;
    PROCEDURE proc1(VAR x: INTEGER);
    VAR b: INTEGER;
    BEGIN
        b:=3;
        x:=b*2+a;
    END

    PROCEDURE proc2(VAR d: INTEGER);
    VAR y: INTEGER;
        b: BOOLEAN;
        FUNCTION fun(x: INTEGER): BOOLEAN;
        VAR a: INTEGER;
    BEGIN
        a:= 2;
        fun:=(y MOD a)= b;
```

TABLA DE SIMBOLOS

Operador	Significado
!	Negación
+	Suma
-	Resto
*	Multiplicación
/	División
%	Módulo
<	Menor
<=	Menor igual
>	Mayor
>=	Mayor igual
!=	Diferente
&&	Conjunción Lógica(Y)
||	Disyunción Lógica(O)
==	Igualdad

EJEMPLO →

1.6 Diferentes herramientas automáticas para generar analizadores léxicos.

Lex es una herramienta de los sistemas UNIX/Linux que nos va a permitir generar código C que luego podremos compilar y enlazar con nuestro programa.

Internamente Lex va a actuar como un autómata que localizará las expresiones regulares que le describamos, y una vez reconocida la cadena representada por dicha expresión regular, ejecutará el código asociado a esa regla.

Las Variables, Funciones, Procedimientos y Macros internas de Lex

Lex incorpora algunas variables, funciones, procedimientos y macros que nos permiten realizar de una forma más sencilla todo el procesamiento. Como variables, las más utilizadas son:

Variable	Tipo	Descriptión
Yytext	char * o char []	Contiene la cadena de texto del fichero de entrada que ha encajado con la expresión regular descrita en la regla.
Yylength	Int	Longitud de yytext. yylength = strlen (yytext).
Yyin	FILE*	Referencia al fichero de entrada.
Yyval yylval	struct	Contienen la estructura de datos de la pila con la que trabaja YACC. Sirve para el intercambio de información entre ambas herramientas

1.7 Gramática libre de Contexto

Una gramática libre de contexto es una notación para describir lenguajes. Son más capaces que los autómatas finitos o las expresiones regulares, pero no pueden definir todos los posibles lenguajes. Son útiles para estructuras anidadas, e.g. paréntesis en lenguajes de programación.

Una gramática libre de contexto se define con G = (V, T, P, S) donde:

• V es un conjunto de variables

• T es un conjunto de terminales

• P es un conjunto finito de producciones de la forma $A \rightarrow \alpha$, donde A es una variables y $\alpha \in (V \cup T)*$

• S es una variable designada llamada el símbolo inicial

- **Definir el concepto de gramática libre de contexto (CFG), gramática ambigua y forma enunciativa.**

  No todas las gramáticas proveen estructuras únicas. Cuando esto ocurre, a veces es posible rediseñará la gramática para crear una estructura única, pero no todo el tiempo. En ese caso se tiene gramáticas

  ambiguas. Por ejemplo, en la gramática:

  $E \rightarrow I$
  $E \rightarrow E + E$
  $E \rightarrow E * E$
  $E \rightarrow (E)$
  . . .

  la sentencia $E + E * E$ tiene dos derivaciones:

  $E \Rightarrow E + E \Rightarrow E + E * E$ y $E \Rightarrow E * E \Rightarrow E + E * E$ lo cual nos da dos ´arboles de parseo.

  Sea G = (V, T, P, S) una CFG. Decimos que G es ambigua si existe una cadena en $T*$ que tenga m´as de

un árbol de parseo. Si todas las cadenas en L(G) tienen a lo más un ´árbol de parseo, se dice que G es no

ambigua. Por ejemplo, la cadena a + a ∗ a tiene dos ´arboles de parseo.

- **Explicar las características de gramáticas de lenguajes.**

Si G(V, T, P, S) es una CFG, entonces el lenguaje de G es: L(G) = {w ∈ T ∗ : S ∗ ⇒G w}, ósea el conjunto de cadenas sobre T ∗ derivadas del símbolo inicial. Si G es una CFG al L(G) se llama lenguaje libre de contexto. Por ejemplo, L(Gpal) es un lenguaje libre de contexto.

Teorema 1 L(Gpal) = {w ∈ {0, 1}∗ : w = wR}

Para probar el Teorema 1 se utiliza inducción: (⇒) Suponemos w = wR. Mostramos por inducción en |w| que w ∈ L(Gpal).

• Base: |w| = 0 or |w| = 1. Entonces w es , 0 o 1. Como P → , P → 1 y P → 0 son producciones, concluimos que P ∗ ⇒G w en todos los casos base.

• Inducción: Suponemos |w| ≥ 2. Como w = wR, tenemos que w = 0x0 o w = 1x1 y que x = xR.

• Si w = 0x0 sabemos de la hipótesis inductiva que P ∗ ⇒ x, entonces P ⇒ 0P 0 ∗ ⇒ 0x0 = w, entonces w ∈ L(Gpal).

• El caso para w = 1x1 es similar.

- **Explicar el proceso de construcción de gramáticas de lenguajes.**

Las gramáticas de lenguajes permiten:

Describir la sintaxis: Definen las estructuras válidas de un lenguaje.

Generar cadenas: Permiten crear nuevas cadenas que pertenecen al lenguaje.

Analizar cadenas: Se pueden utilizar para determinar si una cadena dada pertenece al lenguaje y cuál es su estructura sintáctica.

Ejemplo: La gramática de un lenguaje de programación define la estructura de los programas, como la sintaxis de las variables, expresiones, sentencias, etc.

- **Explicar el proceso de construcción de formas enunciativas.**

  Sea G = (V, T, P, S) una CFG y $\alpha \in (V \cup T)*$. Si $S *\Rightarrow \alpha$ decimos que α está en forma de sentencia (sentential form).

  Si $S \Rightarrow lm\ \alpha$ decimos que α es una forma de sentencia izquierda (left-sentencial form), y si $S \Rightarrow rm\ \alpha$ decimos que α es una forma de sentencia derecha (right-sentencial form). L(G) son las formas de sentencia que estan en $T*$.

  Si tomamos la gramática del lenguaje sencillo, $E * (I + E)$ es una forma de sentencia ya que: $E \Rightarrow E * E \Rightarrow E * (E) \Rightarrow E * (E + E) \Rightarrow E * (I + E)$.

  Esta derivación no es ni más a la izquierda ni más a la derecha. Por otro lado: $a * E$ es una forma de sentencia izquierda, ya que:

  $E \Rightarrow lm\ E * E \Rightarrow lm\ I * E \Rightarrow lm\ a * E$ y $E * (E + E)$ es una forma de sentencia derecha, ya que: $E \Rightarrow rm\ E * E \Rightarrow rm\ E * (E) \Rightarrow rm\ E * (E + E)$.

- **Explicar el proceso de remoción de ambigüedad de gramáticas.**

  La ambigüedad se puede eliminar de una gramática mediante:

  Prioridad de operadores: Estableciendo un orden de evaluación para los operadores.

  Asociatividad de operadores: Especificando si los operadores son izquierda o derecha asociativos.

  Factorización: Reorganizando las reglas de producción.

  Ejemplo: Para eliminar la ambigüedad en el ejemplo anterior, se pueden agregar reglas para establecer la precedencia de los operadores, como que la multiplicación y la división tienen mayor precedencia que la suma y la resta.

- **Describir la normalización de CFG.**

  La normalización de una CFG consiste en transformar la gramática en una forma estándar, como la Forma Normal de Chomsky. Esto facilita el análisis y la comparación de gramáticas.

  Ejemplo: La Forma Normal de Chomsky requiere que todas las reglas tengan la forma $A \rightarrow BC$ o $A \rightarrow a$, donde A, B, C son símbolos no terminales y a es un símbolo terminal.

- **Explicar el proceso de normalización de CFG.**

  1. Eliminar producciones ε (cadenas vacías).
  2. Eliminar producciones unitarias (A → B).
  3. Transformar las producciones en la forma requerida.

     Ejemplo: La gramática original podría transformarse en la Forma Normal de Chomsky aplicando estas reglas.

CONCLUSIÓN

El estudio de los analizadores léxicos, autómatas finitos, gramáticas libres de contexto y herramientas asociadas constituye un pilar fundamental en la construcción de compiladores y, en general, en el procesamiento de lenguajes formales. Estos conceptos interrelacionados permiten modelar, reconocer y analizar la estructura sintáctica de los lenguajes de programación, así como de otros sistemas formales.

Las expresiones regulares, como herramienta de descripción de patrones, y los autómatas finitos, como modelos de computación, son esenciales para la tarea de identificar y clasificar los componentes léxicos de un programa. Por su parte, las gramáticas libres de contexto proporcionan un formalismo riguroso para describir la estructura sintáctica de un lenguaje, permitiendo la construcción de analizadores sintácticos capaces de verificar la corrección gramatical de un programa.

La implementación de estos conceptos se ve facilitada por herramientas como Flex, que automatizan la generación de analizadores léxicos a partir de especificaciones basadas en expresiones regulares. Además, la normalización de gramáticas libres de contexto contribuye a simplificar el análisis sintáctico y a mejorar la eficiencia de los compiladores.

En resumen, los temas abordados en este análisis son de vital importancia para el desarrollo de herramientas de software que permiten la traducción y ejecución de programas de computadora. Su comprensión profunda es esencial para cualquier profesional interesado en el diseño y construcción de compiladores, intérpretes y otros sistemas de procesamiento de lenguajes.

REFERENCIAS

- Dean K. (1995). “Teoría de Autómatas y Lenguajes Formales”. Edit. Prentice Hall, España.
- ¨ Hopcroft J. E., Ullman J.D. (2007). “Introducción a la teoría de autómatas, lenguajes y computación”. 3ª ed. Edit. Pearson Educación, Madrid.
- ¨ Linz P. (2001) "An Introduction to Formal Languages and Automata", 3rd Edition, J.A. Bartlett.
- 4.3 La tabla de símbolos. (s. f.) http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro32/autocontenido/aut ocon/43__la_tabla_de_smbolos.html
- ¨ Martin J. (2004). “Lenguajes Formales y Teoría de la computación”. 3ª ed. Edit. MacGraw-Hill Interamericana de México.
- Nº, C., Adolfo, A., Fuente, J., Manuel, J., Lovelle, C., Ortín, F., Raúl, S., Castanedo, I., Cándida, M., Díez, L., Emilio, J., & Gayo, L. (2006). *Tablas de Símbolos de Procesadores de Lenguaje*. http://di002.edv.uniovi.es/~cueva/publicaciones/monografias/41_TablasDeSimbolos.pdf
- Bavera, F., Nordio, D., Arroyo, M., & Aguirre, J. (n.d.). *JTLex un Generador de Analizadores Léxicos Traductores*. https://dc.exa.unrc.edu.ar/staff/fbavera/papers/TesisJTLex-Bavera-Nordio-02.pdf
- *Ciencias computacionales Propedéutico: Teoría de Autómatas y Lenguajes Formales Gramáticas libres de contexto y lenguajes*. (n.d.). https://posgrados.inaoep.mx/archivos/PosCsComputacionales/Curso_Propedeutico/Automatas/05_Automatas_GramaticasLibresContextoLenguajes/CAPTUL1.PDF