Universidad del Quindío Oct. 4/2003
Ingeniería de Sistemas y Computación
Teoría de la Computación

Profesor: Leonardo Hernández Rodríguez
leonardohernandez@uniquindio.edu.co
www.geocities.com/leoher314

Aplicaciónes de los AFD

Fases de un compilador

Un compilador es un programa de computador muy complejo de escribir. Se suele dividir en varias fases. Según Karen A. Lemone, las fases de un compilador son las siguientes:

- Análisis lexicográfico. Se conoce también como rastreo
- Análisis sintáctico. Se conoce también como análisis gramatical
- Análisis semántico
- Optimización
- Preparación para la generación de código
- Generación de código

Todas estas fases se estudiaran en la asignatura “Compiladores” el próximo semestre.

Analizador Léxico

Los programas de computador se conforman de una secuencia de palabras o tokens, como por ejemplo: identificadores, números enteros, operadores etcétera.

La primera fase de un compilador es el análisis léxico., este se encarga, entre otras funciones, de detectar o rastrear los tokens

del programa que está siendo compilado y de determinar el tipo de cada uno de estos tokens.

Ejemplo de entrada a un analizador léxico

La típica entrada e un analizador léxico es un archivo de texto con un programa fuente en algún lenguaje de programación, como el siguiente:

```
void main (void)
        int s;
        s = 0;
        for (  i = 1;  @ <= 100; i++)
                s = s + i;
        printf( “suma: %d“, s );
}
```

Obviamente la entrada a un analizador léxico puede tener errores de sintaxis.

Ejemplo de la lista de token producida por un analizador léxico

void	Identificador
main	Identificador
(	Paréntesis de abrir
void	Identificador
)	Paréntesis de cerrar
int	Identificador
s	Identificador
;	Punto y coma
s	Identificador
=	Operador de asignación
0	Entero
;	Punto y coma
for	Identificador
(	Paréntesis de abrir
i	Identificador
=	Operador de asignación

1	Entero
;	Punto y coma
@	Error
<=	Operador relacional
100	Entero
… etcetera	

Note que el único error que se ha detectado es el de tipo de token inválido. Otros errores como variables no declaradas y falta de llaves de abrir o de cerrar, escapan del alcance del análisis léxico. Estos errores se manejan en las otras fases del compilador.

Otros tipos de tokens son:

- Las cadenas de caracteres
- Los operadores multiplicativos: *, /, %
- Los operadores aditivos: +,-
- Los operadores lógicos: !, &&, ||

Función siguienteToken()

En la implementación de un analizador léxico es natural incluir la función siguienteToken(). Esta tiene las siguientes características:

Recibe:

- Una gran cadena de caracteres *a,* conteniendo un programa fuente completo.
- Una posición *i,* en la cadena de caracteres anterior

Retorna:

- El token que se encuentra en la cadena *a* a partir de la posición *i*
- El tipo de este token
- La posición inicial del siguiente token

Ejemplo 1

Datos recibidos por la función:

- La cadena *a*

v	o	i	d		m	a	i	n	(	…	x	=	1	2	3	+	y	…	}	\0
0	1	2	3	4	5	6	7	8	9	…	24	25	26	27	28	29	30	…	98	99

- La posición i = 26

Datos devueltos:

- El token que comienza en la posición 26, es decir el número 123.
- El tipo de token: un número entero
- La posición inicial del siguiente token, es decir la posición 29, ya que el siguiente token es el signo más (+).

Ejemplo 2

Datos recibidos por la función:

- La cadena *a*

v	o	i	d		m	a	i	n	(	…	x	=	1	2	3	+	y	…	}	\0
0	1	2	3	4	5	6	7	8	9	…	24	25	26	27	28	29	30	…	98	99

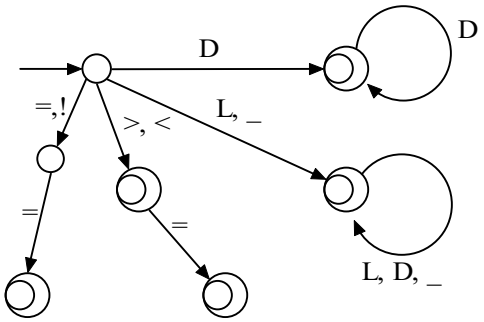
- La posición i = 5

Datos devueltos:

- El token que comienza en la posición 5, es decir el identificador *main*.
- El tipo de token: un identificador
- La posición inicial del siguiente token, es decir la posición 9, ya que el siguiente token es el signo paréntesis de abrir.

Implementación

Una forma eficiente de implementar de la función *siguienteToken* es basándose en un AFD, un gran AFD que represente todos los tipo de tokens existentes en el lenguaje a compilar. En el AFD de la figura se incluye como ejemplo los enteros, los identificadores y los operadores relacionales, pero deben incluirse todos los tipos de tokens. Por claridad se han omitido estados de rechazo de donde ya no pueda escapar el AFD y las transiciones hacia estos estados.



Ahora veamos una codificación en seudocódigo de la función *siguienteToken* basada en el AFD anterior:

```
función siguienteToken ( Cadena a[0..?], i )
        cadena b[0..80]
        b[0] ← a[i]
        j ← 1;
        si esDigito( a[i] ) entonces
                i ← i + 1
                mientras esDigito( a[i] )
                        b[i] ← a[i]
                        j ← j + 1
                        i ← i + 1
                b[j] ← NULO
                devolver b, “Entero”, i
        si esLetra( a[i] ) o a[i] = ‘_’ entonces
                …
                devolver b, “identificador”, i
        si a[i] = ‘<’  o  a[i] = ‘>’ entonces
                …
                devolver b, “Operador Relacional”,, i
        …
        b[j] = NULO
        devolver b, “Error”, i+1
```

Note como cada una de las rutas del AFD se reflejan en una estructura si … entonces y cada bucle del AFD se refleja en un bucle mientras.

Bibliografía

KELLEY, Dean. Teoría de autómatas y lenguajes formales. Prentice hall, Madrid, 1995.

LEMONE, Karen A. Introducción a los compiladores.