



ANÁLISIS LÉXICO

Ing. Max Cerna

Agenda

1. Análisis léxico
2. Ejemplos de análisis léxico
3. Lenguajes regulares
4. Lenguajes formales
5. Especificaciones léxicas

Análisis léxico

¿Qué queremos hacer?

```
if (i == j)
```

```
    Z = 0;
```

```
else
```

```
    Z = 1;
```

realmente es solo una cadena de caracteres:

```
\tif (i == j)\n\t\tz = 0;\n\telse\n\t\tz = 1;
```

META: dividir la cadena de entrada en subcadenas
Donde las subcadenas se llaman ***tokens***

Que es un Token?

Una secuencia de caracteres que se **agrupa** y **clasifica** como una unidad.

Clase de Token:

En idioma inglés:

En un lenguaje de programación:

Clase de Token

- **Identificador:** Cadenas de letras o dígitos, comenzando con una letra
- **Número entero:** Una cadena de dígitos no vacía
- **Palabra clave (keyword):** "else" o "if" o "begin" o ...
- **Espacio en blanco:** Una secuencia no vacía de espacios en blanco, saltos de línea y tabulaciones

¿Para qué sirven los Tokens?

Clasificar subcadenas de programas según su rol.

El análisis léxico produce un flujo de tokens que es entrada para el parser.

El analizador se basa en distinciones de tokens

- Un identificador se trata de forma diferente a una palabra clave.

Ejemplo

```
\tif (i == j)\n\t\tz = 0;\n\telse\n\t\tz = 1;
```

Ejercicio

Para el fragmento de código descrito a continuación, seleccione el número correcto de tokens:

`x = 0;\n\twhile (x < 10) {\n\ttx++;\n}`

- a) W = 9; K = 1; I = 3; N = 2; O = 9
- b) W = 11; K = 4; I = 0; N = 2; O = 9
- c) W = 9; K = 4; I = 0; N = 3; O = 9
- d) W = 11; K = 1; I = 3; N = 3; O = 9

Analizador Léxico

Una implementación de analizador léxico debe hacer 2 cosas:

- Reconocer las cadenas que correspondan a los tokens (los lexemas)
- Identificar la categoría de cada lexema

< token class, lexema >

Analizador Léxico

- Normalmente se descarta los tokens "poco interesantes" que no contribuyen al análisis.

Ejemplos: espacios en blanco, comentarios

Analizador Léxico

- Estructura léxica = clases (categorías) de tokens
- Podemos afirmar que un conjunto de cadenas pertenece a una clase de tokens

Ejemplos de análisis léxico

Ejemplo Analizador Léxico

Regla de FORTRAN: Los espacios en blanco son insignificantes

VAR1 es lo mismo que VA R1

Ejemplo Analizador Léxico

Regla de FORTRAN: Los espacios en blanco son insignificantes

VAR1 es lo mismo que VA R1

Terrible diseño

Ejemplo Analizador Léxico

DO 5 I = 1,25

DO 5 I = 1.25

Ejemplo Analizador Léxico

- El objetivo es dividir la cadena. Esto se implementa leyendo de izquierda a derecha, reconociendo un token a la vez.
- Es posible que se requiera "mirar hacia adelante (lookahead)" para decidir dónde termina un token y comienza el siguiente token

Ejemplo Analizador Léxico

```
if (i == j)
```

```
    Z = 0;
```

```
else
```

```
    Z = 1;
```

Ejemplo Analizador Léxico

PL/I: Las palabras clave (keywords) no están reservadas

IF ELSE THEN THEN = ELSE; ELSE ELSE = THEN

Ejemplo Analizador Léxico

PL/I: Las palabras clave (keywords) no estan reservadas

DECLARE (ARG1, . . . , ARGN)

¿Es DECLARE una palabra clave o un identificador de arreglo?

Ejemplo Analizador Léxico

C++ template

```
Foo<Bar>
```

C++ stream

```
cin >> var;
```

Lenguajes Regulares

Lenguaje

Sea el alfabeto Σ un conjunto de caracteres.

Un lenguaje sobre Σ es un conjunto de cadenas de caracteres extraídos de Σ

Notación

- Los lenguajes son conjuntos de cadenas.
- Necesitamos alguna notación para especificar qué conjuntos queremos.
- La notación estándar para los lenguajes regulares es expresiones regulares.

Lenguajes Regulares

Existen varios formalismos para especificar tokens.

Los lenguajes regulares son los más populares.

- Teoría simple y útil.
- Fácil de comprender
- Implementaciones eficientes

Notación de Expresiones Regulares

- **Epsilon**

$$\varepsilon = \{'''\}$$

Representa una cadena de longitud cero.

No es un conjunto vacío.

Notación de Expresiones Regulares

- **Carácter simple**

$$'c' = \{ "c" \}$$

- **Unión**

$$A + B = \{ s \mid s \in A \text{ or } s \in B \}$$

Notación de Expresiones Regulares

- Concatenación

$$AB = \{ab \mid a \in A \text{ and } b \in B \}$$

- Iteración

$$A^* = \bigcup_{i \geq 0} A_i \text{ donde } A_i \text{ es } AA \dots A \text{ } i \text{ veces}$$

Expresiones Regulares

Definición

Las expresiones regulares sobre Σ son el conjunto más pequeño de expresiones que incluyen

ϵ

'c' donde $c \in \Sigma$

$A+B$ donde A,B es una rexp sobre Σ

AB donde A,B es una rexp sobre Σ

A^* donde A es una rexp sobre Σ

Ejercicio

Elija los lenguajes regulares que sea equivalentes al lenguaje:

$(0+1)^*1(0+1)^*$ para $\Sigma = 0, 1$

- a) $(01+11)^*(0+1)^*$
- b) $(0+1)^*(10+11+1)(0+1)^*$
- c) $(1+0)^*1(1+0)^*$
- d) $(0+1)^*(0+1)(0+1)^*$

Lenguajes Formales

Lenguajes Formales

- Alfabeto = Caracteres en español
- Lenguaje = Oraciones en español


- Alfabeto = ASCII
- Lenguaje = Programas en C

Sintaxis y Semántica

La notación hasta ahora era imprecisa

$$AB = \{ab \mid a \in A \text{ and } b \in B\}$$

 B como una pieza de sintaxis

 B como un conjunto
(la semántica de la sintaxis)

Sintaxis y Semántica

La función de significado L asigna la sintaxis a la semántica

$$\varepsilon = \{''''\}$$

$$'c' = \{''c''\}$$

$$A + B = \{ A \cup B \}$$

$$AB = \{ab \mid a \in A \text{ and } b \in B \}$$

$$A^* = \bigcup_{i \geq 0} A_i$$

Sintaxis y Semántica

La función de significado L asigna la sintaxis a la semántica

$$L(\epsilon) = \{''''\}$$

$$L('c') = \{''c''\}$$

$$L(A + B) = \{ L(A) \cup L(B) \}$$

$$L(AB) = \{ab \mid a \in L(A) \text{ and } b \in L(B)\}$$

$$L(A^*) = \bigcup_{i \geq 0} L(A^i)$$

LENGUAJES FORMALES

¿Por qué usar una función de significado?

- Dejar claro qué es la sintaxis, qué es la semántica.
- Permite considerar la notación como un tema separado
- Porque las expresiones y significados no son 1-1*

LENGUAJES FORMALES

*El significado es muchos a uno: ¡nunca uno a muchos!

- Indica que una expresión sintáctica específica no puede tener múltiples significados semánticos diferentes.
- Esto sería una ambigüedad semántica y podría llevar a problemas de interpretación o ejecución en el lenguaje.

Ejemplo: **$x = y$**

no puede significar dos operaciones diferentes al mismo tiempo (ej: asignación y comparación).

Especificaciones léxicas

Expresiones Regulares

Se pueden describir ***tokens*** usando ***expresiones regulares***

Ejemplos

Token - Keywords

Abreviación: 'E' 'L' 'S' 'E' se puede escribir como 'ELSE'

keyword: "ELSE" or "IF" or "BEGIN" or ...

'ELSE' + 'IF' + 'BEGIN' +...

Token - Integer

Integer: una cadena no vacía de dígitos

digit = '0' + '1' + '2' + '3' + '4' + '5' + '6' + '7' + '8' + '9'

Abreviación: [0-3] = '0' + '1' + '2' + '3'

digit = [0-9]

integer = digit digit*

Abreviación: digit⁺ = digit digit*

Token - Identifier

Identifier: cadena de letras o números que inician con letra.

letter = 'a' + ... + 'z' + 'A' + ... + 'Z'

Abreviación: [a-z] = 'a' + ... + 'z'

identifier = letter (letter + digit)*

Token - Whitespace

Whitespace: cadena de no vacía de espacios en blanco, cambio de línea y tabulaciones.

`whitespace = (' ' + '\n' + '\t')+`

Ejemplo - Mail Address

ej: estudiante@url.edu.gt

$\Sigma = \text{letras} \cup \{',', '@'\}$

name = letter⁺

address = name '@' name '.' name('.'name+ ϵ)

Ejemplo - Phone Number

ej: +502-2341-5678

$\Sigma = \text{dígitos} \cup \{'-', '+'\}$

section = digit⁴

area = '+'digit³

phone = (area'-' ϵ)section'-'section

Ejemplo- Número decimal formato Pascal

digit = [0-9]

digits = digit⁺

opt_fraction = ('.'digits) + ϵ

opt_exponent = ('E'('+'+'-'+'+)'digits) + ϵ

num = digits opt_fraction opt_exponent