Documentación del Analizador Léxico en Java

Sarah Sophía Olivares García

Septiembre 2024

1 Introducción

Este documento describe en detalle la implementación de un analizador léxico en Java, que es capaz de reconocer identificadores, números enteros, números decimales, operadores de asignación, operadores de suma, y manejar espacios en blanco. El objetivo principal de este programa es ilustrar el funcionamiento básico de un analizador léxico para un lenguaje sencillo.

2 Descripción del Programa

El programa está compuesto por varios métodos que permiten identificar diferentes tipos de tokens dentro de una cadena de texto. Los tokens que el programa puede reconocer son:

- **Identificadores**: Formados exclusivamente por letras.
- **Números enteros**: Secuencias de dígitos sin punto decimal.
- **Números decimales**: Secuencias de dígitos que contienen un punto decimal.
- **Operadores de asignación**: Actualmente se reconoce el operador `=`.
- **Operadores de suma**: Se reconocen los operadores `+` y `+=`.
- **Espacios en blanco**: Se manejan pero no son considerados como tokens.

3 Métodos Implementados

A continuación se detalla cada uno de los métodos implementados en el programa:

3.1 esLetra

Este método se encarga de verificar si un carácter dado es una letra del alfabeto.

```
public static boolean esLetra(char c) {
    return Character.isLetter(c);
}
```

3.2 esDigito

Este método verifica si un carácter dado es un dígito.

```
public static boolean esDigito(char c) {
    return Character.isDigit(c);
}
```

3.3 esEspacioEnBlanco

Este método identifica si un carácter es un espacio en blanco. Los espacios en blanco no son considerados tokens, pero son útiles para separar los tokens.

```
public static boolean esEspacioEnBlanco(char c) {
    return Character.isWhitespace(c);
}
```

3.4 esIdentificador

Este método determina si un token es un identificador, es decir, una secuencia de caracteres que contiene solo letras.

```
public static boolean esIdentificador(String token) {
    for (int i = 0; i < token.length(); i++) {
        if (!esLetra(token.charAt(i))) {
            return false;
        }
    }
    return true;
}
```

3.5 esEntero

Este método verifica si un token es un número entero.

```
public static boolean esEntero(String token) {
    for (int i = 0; i < token.length(); i++) {
        if (!esDigito(token.charAt(i))) {
            return false;
```

```
        }
    }
    return true;
}
```

3.6 esDecimal

Este método determina si un token es un número decimal, es decir, contiene un punto decimal además de dígitos.

```
public static boolean esDecimal(String token) {
    boolean puntoEncontrado = false;
    for (int i = 0; i < token.length(); i++) {
        char c = token.charAt(i);
        if (c == '.') {
            if (puntoEncontrado) {
                return false;
            }
            puntoEncontrado = true;
        } else if (!esDigito(c)) {
            return false;
        }
    }
    return puntoEncontrado;
}
```

3.7 esOperadorAsignacion

Este método identifica si un token es un operador de asignación (`=`).

```
public static boolean esOperadorAsignacion(String token) {
    return token.equals("=");
}
```

3.8 esOperadorSuma

Este método identifica si un token es un operador de suma (`+` o `+=`).

```
public static boolean esOperadorSuma(String token) {
    return token.equals("+") || token.equals("+=");
}
```

3.9 analizarCadena

Este método toma una cadena de texto como entrada, la divide en tokens utilizando los espacios en blanco como delimitadores, y luego utiliza los métodos anteriores para identificar cada token.

```
public static void analizarCadena(String cadena) {
    String[] tokens = cadena.split("\\s+");
    for (String token : tokens) {
        if (esIdentificador(token)) {
            System.out.println("El token \"" + token + "\" es un identificador v
        } else if (esEntero(token)) {
            System.out.println("El token \"" + token + "\" es un numero entero v
        } else if (esDecimal(token)) {
            System.out.println("El token \"" + token + "\" es un numero decimal
        } else if (esOperadorAsignacion(token)) {
            System.out.println("El token \"" + token + "\" es un operador de asi
        } else if (esOperadorSuma(token)) {
            System.out.println("El token \"" + token + "\" es un operador de sum
        } else {
            System.out.println("El token \"" + token + "\" no es v lido.");
        }
    }
}
```

4 Ejemplo de Ejecución

A continuación se muestra un ejemplo de ejecución del programa con la cadena de entrada:

```
a = 5 + 3.5 +=
```

El resultado de la ejecución sería:

```
El token "a" es un identificador valido.
El token "=" es un operador de asignacion.
El token "5" es un numero entero válido.
El token "+" es un operador de suma.
El token "3.5" es un numero decimal válido.
El token "+=" es un operador de suma.
```

5 Descripción del Método

El método de *elementos punteados* es un procedimiento sistemático que permite obtener un Autómata Finito Determinista (AFD) a partir de una expresión regular. Este método utiliza la noción de estados en los que se encuentran las subexpresiones de una expresión regular durante su evaluación.

El método de elementos punteados utiliza puntos o marcadores para indicar en qué posición de la expresión regular se encuentra el análisis. A medida que el autómata consume símbolos de entrada, los puntos avanzan por la expresión, determinando las transiciones entre estados. El objetivo es construir un AFD que acepte el mismo lenguaje que la expresión regular.

Los pasos para aplicar este método son los siguientes:

1. Colocar un punto (marcador) al principio de la expresión regular.
2. A medida que se leen los símbolos de entrada, mover el punto a la siguiente posición en la expresión.
3. Crear un estado del AFD para cada nueva posición del punto.
4. Definir las transiciones del AFD en función del avance del punto.

6 Ejemplo de Aplicación

Consideremos la siguiente expresión regular:

$$E = a(b|c)^*$$

A continuación, aplicamos el método de elementos punteados:

- Colocamos el punto al inicio de la expresión regular:
  $\cdot a(b|c)^*$
- Luego, avanzamos el punto después de la primera letra a:
  $a \cdot (b|c)^*$
- El siguiente avance del punto corresponde a las opciones b o c, lo que genera dos posibles transiciones:
  $ab \cdot (b|c)^*$
  $ac \cdot (b|c)^*$
- A medida que se repite la expresión $(b|c)^*$, el punto puede regresar a su posición anterior, indicando que se puede repetir la operación sobre los mismos símbolos.

7 Construcción del AFD

Utilizando el método anterior, el autómata finito determinista (AFD) correspondiente se puede representar gráficamente. En el siguiente diagrama, mostramos los estados generados y las transiciones entre ellos:

[shorten ¿=1pt, node distance=2cm, on grid, auto] [state, initial] (q_0)q_0; [state] (q_1)[$right = of q_0$]q_1; [state] (q_2)[$right = of q_1$]q_2; [state] (q_3)[$below = of q_1$]q_3; [state, accepting] (q_4)[$right = of q_2$]q_4; [-¿] (q_0)$edge[bend left]node a(q_1)(q_1)edge[bend left]node b(q_2)(q_1)edge[bend right]node[below]c(q_3)(q_2)edge[loopabove$

El autómata tiene los siguientes estados:

- q_0: Estado inicial, donde aún no se ha leído ningún símbolo.
- q_1: Después de leer el símbolo a.
- q_2: Después de leer b dentro de la repetición $(b|c)^*$.
- q_3: Después de leer c dentro de la repetición $(b|c)^*$.
- q_4: Estado de aceptación, donde se ha reconocido toda la expresión regular.

8 Conclusión

El método de elementos punteados proporciona un enfoque visual y sistemático para la construcción de autómatas a partir de expresiones regulares. Es particularmente útil en la creación de autómatas finitos deterministas (AFD) que son utilizados en el análisis léxico de lenguajes formales.

9 Conclusión

Este programa de analizador léxico básico permite reconocer diversos tipos de tokens dentro de una cadena de entrada. La implementación modular de cada método facilita la adición de nuevas reglas de análisis léxico si se requiere. El programa puede ser extendido para soportar más operadores y tipos de tokens según las necesidades del lenguaje que se desee analizar.