



PROYECTO #1



MANUAL TÉCNICO



**EDUARDO JOSUÉ
GONZÁLEZ CIFUENTES**

COMPTILADORES 1

201900647



INDICE

Introducción	Pág. 3
Objetivos	Pág. 4
Especificación técnica	Pág. 5
Lógica del programa	Pág. 6
Créditos	Pág. 16



INTRODUCCION

SubSetify es una herramienta de software diseñada para simplificar y automatizar el proceso de conversión de AFN a AFD. Este proyecto nació de la necesidad de facilitar la implementación del método de subconjuntos en estudiantes de sistemas, haciendo hincapié en la eficiencia y precisión del análisis léxico.



OBJETIVOS

1. El principal objetivo de SubSetify es simplificar el proceso de conversión de AFN a AFD. Esto se logra mediante la automatización de tareas que suelen ser tediosas y propensas a errores cuando se realizan manualmente, permitiendo a los usuarios enfocarse en aspectos más conceptuales del análisis léxico.
2. El proyecto incluye la implementación del método de Thompson para la generación de Autómatas Finitos No Deterministas a partir de expresiones regulares. Este enfoque permite a los usuarios trabajar con expresiones regulares familiares y explorar la relación directa entre estas y los autómatas.
3. Fomentar la colaboración y el aprendizaje colaborativo en el campo de la programación y la ciencia de datos



ESPECIFICACIÓN TÉCNICA

REUERIMIENTOS HARDWARE

- Laptop o computadora de escritorio.
- Mínimo 4GB de Memoria RAM.
- 500GB de Disco Duro o Superior.
- Procesador Intel Core i3 o superior.

REQUERIMIENTOS SOFTWARE

- Sistema Operativo Windows 10 o superior.
- Java Runtime Enviroment (JRE) versión 8.2 o superior.
- Java Development Kit (JDK) version 8.2 o superior.
- Lenguaje de Programación Java.
- NetBeans IDE 8.2 o superior.
- Navegador web.
- Librería Java Cup.
- Librería Java Flex.
- Librería rsyntaxtextarea.
- Librería JfreeChart.
- Librería JTatto.
- Librería Quaqua.



LÓGICA DEL PROGRAMA

LENGUAJE SUBSETIFY

Si necesita información de como se estructura el lenguaje SubSetify sugiero consulta el manual de usuario donde se encuentra abordado ese tema.

ANÁLISIS LÉXICO SUBSETIFY

Acá está un listado de todos los tokens que se utilizaron para el análisis léxico del lenguaje SubSetify.

TOKEN	LEXEMA	DESCRIPCIÓN
"TkCONJ_R",	conj	Palabra R. CONJ
"TkPUNTOYCOMA",	;	Palabra R. Operador Simbolos
"TkDOSPUNTOS",	:	Palabra R. Operador Simbolos
"TkCOMA",	,	Palabra R. Operador Simbolos
"TkLLAVEA",	{	Palabra R. Operador Simbolos
"TkLLAVEC",	}	Palabra R. Operador Simbolos
"TkPARENTESISAbre",	(Palabra R. Operador Simbolos
"TkPARENTESISACierra",)	Palabra R. Operador Simbolos
"TkDelimitador",	"->"	Palabra R. Operador Simbolos
"TkOr",		Palabra R. Operador Simbolos
"TkMas",	+	Palabra R. Operador Simbolos
"TkInterrogacion",	?	Palabra R. Operador Simbolos
"TkConcatenacion",	.	Palabra R. Operador Simbolos
"TkGuionCurseado",	~	Palabra R. Operador Simbolos
"TkGuionNormal",	-	Palabra R. Operador Simbolos
"TkGuionBajo",	_	Palabra R. Operador Simbolos
"TkCerraduraKleen"	*	Palabra R. Operador Simbolos

Acá está un listado de todas las expresiones regulares que se utilizaron para el análisis léxico del lenguaje SubSetify.

TOKEN	LEXEMA	DESCRIPCIÓN
"TkENTERO",	52	"[0-9]{2,}"
"TkDECIMAL",	5	"([0-9]+\.[0-9]+ [0-9]+)"
"TkCADENA",	"HOLA MUNDO"	"[\"]([^\n]*[^\n])\""

MÉTODOS

Análisis SUBSETIFY

```

public static void analizarSubSetify(String entrada) {
    try {
        TokenList.clear();
        analyzerr.Lexer lexer = new analyzerr.Lexer(new StringReader(entrada));
        analyzerr.Parser parser = new analyzerr.Parser(lexer);
        parser.parse();
        System.out.println("Se analizó correctamente el archivo SS :D");

        textAreaGG2.setText(textAreaGG2.getText()+"\n" + ">> "+"Análisis Léxico Exitoso ");
        JOptionPane.showMessageDialog(null, "Análisis SS generado con éxito :D");

    } catch (Exception e) {
        System.out.println("Error fatal en compilación de entrada.");
        System.out.println(e);
    }
}

```

Se utilizaron Packets y Clases de Java para llevar a cabo la realización del algoritmo de thompson. A continuación, se dejará las clases mas importantes para la funcionalidad del proyecto.

Clase Main

Acá se manda a llamar la interfaz Grafica

```

public static void main(String[] args) {
    try {
        UIManager.setLookAndFeel(new McWinLookAndFeel());
    } catch (UnsupportedLookAndFeelException e) {
        System.err.println("No se pudo aplicar el Look and Feel deseado: " +
            e.getMessage());
    }

    GUI nuevaGUI = new GUI();
    nuevaGUI.setVisible(true);

    System.out.println(e);
}

```


Class NFA

```
public class NFA {
    State start, accept;
    String transition;

    public NFA(State start, State accept, String transition) {
        this.start = start;
        this.accept = accept;
        this.accept.transition = transition;
    }
    @Override
    public String toString() {
        return "NFA{start=" + start.label + ", accept=" + accept.label + ", transition='" +
        accept.transition + "'}";
    }
}
```

Variables Globales

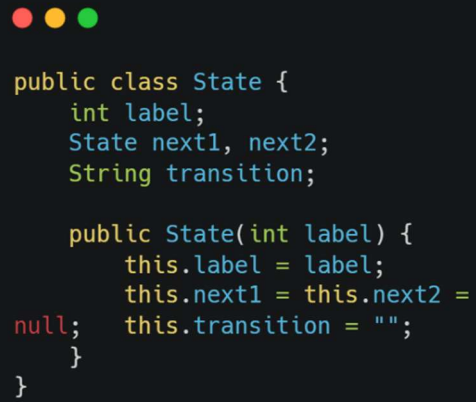
```
public class funca {

    //en uso
    public static LinkedList<Token> TokenList = new LinkedList<>();

    //en uso
    public static LinkedList<Erroror> ErrorList = new LinkedList<>();

    //      Id ER -- ER
    public static HashMap<String,ExpresionRegular > HashMapRegex = new HashMap<>
    ();
}
```

Class State



```
public class State {  
    int label;  
    State next1, next2;  
    String transition;  
  
    public State(int label) {  
        this.label = label;  
        this.next1 = this.next2 =  
null;    this.transition = "";  
    }  
}
```

Clase Thompson

```

private static int stateCounter = 0;
private static StringBuilder contenidoParentesis = new StringBuilder();
public static boolean dentroDeParentesis = false;
public static boolean modoLiteral = false;
private static StringBuilder literalBuilder = new StringBuilder();

public static NFA buildNFAFromRegex(String regex) {
    Stack<NFA> stack = new Stack<>();

    for (int f = regex.length() - 1; f >= 0; f--) {
        char c = regex.charAt(f);

        if (c == '*') {
            if (!stack.isEmpty()) {
                NFA nfa = stack.pop();
                stack.push(closure(nfa));
            } else {
                // Manejar el error
            }
        } else if (c == '.') {
            // Agregar este bloque para asegurarse de que todas las operaciones de concatenación se
            // completan
            while (stack.size() >= 2) {
                NFA a = stack.pop();
                NFA b = stack.pop();
                stack.push(concatenation(b, a));
            }
        } else if (c == '|') {
            if (stack.size() >= 2) {
                NFA bOr = stack.pop();
                NFA aOr = stack.pop();
                stack.push(or(aOr, bOr));
            } else {
                // Manejar el error
            }
        } else if (c == '^') {
            stack.push(basic(String.valueOf(c)));
        } else if (Character.isLetterOrDigit(c)) {
            // Verificar si estamos dentro de parentesis o no
            if (dentroDeParentesis) {
                //System.out.println(c);
                contenidoParentesis.append(c);
            } else {
                //System.out.println(c);
                stack.push(createToken(String.valueOf(c)));
            }
        } else if (c == '+') {
            // Manejar la cerradura positiva
            if (!stack.isEmpty()) {
                NFA nfa = stack.pop();
                stack.push(positiveClosure(nfa));
            } else {
                // Manejar el error
            }
        } else if (c == '(') {
            // Marcar que estamos dentro de un parentesis
            dentroDeParentesis = true;
            //System.out.println(c);

            contenidoParentesis = new StringBuilder();
        } else if (c == ')') {
            // Marcar que estamos fuera de un parentesis
            dentroDeParentesis = false;
            //System.out.println(dentroDeParentesis);
            // Agregar el contenido del parentesis como un token si hay contenido
            if (contenidoParentesis.length() > 0) {
                //System.out.println(contenidoParentesis);
                stack.push(basic(contenidoParentesis.reverse().toString()));
            }
        }

        System.out.println("Stack content: " + stack);
    }

    // Asegurarse de que todas las operaciones de concatenación pendientes se completan al final
    while (stack.size() >= 2) {
        NFA a = stack.pop();
        NFA b = stack.pop();
        stack.push(concatenation(b, a));
    }

    if (stack.size() == 1) {
        return stack.pop();
    } else {
        // Manejar el error
        return null;
    }
}

```

CRÉDITOS

Proyecto desarrollado por Eduardo González del Curso Organización de Lenguajes y Compiladores 1, en ciudad de Guatemala para la Universidad San Carlos de Guatemala el 17 de diciembre del año 2023.

Se adjunta repositorio donde se encuentra todo el código fuente.

https://github.com/La10gg/-OLC1-Proyecto1_201900647

