

Universidad Técnica Particular de Loja



INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN

**TEORIA DE AUTOMATAS
Y COMPILADORES**

Elaborado por:

Edisson Chamba

Abril- agosto 2025

Documentación del Proyecto: Analizador Léxico con Autómata Finito Determinista

Descripción general

Este proyecto implementa un analizador léxico que detecta si los tokens de un archivo de texto (simulando código fuente) son válidos, utilizando un autómata finito determinista (AFD) diseñado y programado en Python. El sistema detecta variables, números, cadenas de texto, operadores y símbolos comunes de un lenguaje tipo C/JavaScript.

Archivos del proyecto

- automata_completo_con_cadenas.jff: Autómata visual en formato JFLAP.
- automata_completo_con_cadenas.txt: Transiciones del AFD en formato de texto.
- prueba.txt: Archivo de entrada con líneas de código a analizar.
- verificador_final.py: Script en Python que carga el AFD y valida los tokens del archivo.

Link repositorio de GitHub con archivos adjuntos.

<https://github.com/edisson2407/automatas1.git>

Flujo de funcionamiento

1. Se diseña un AFD que reconoce los tokens válidos.
2. El AFD se transcribe como una lista de transiciones en Python.
3. Un archivo de prueba contiene líneas de código simuladas.
4. El verificador:
 - Lee cada línea.
 - Extrae tokens con expresiones regulares.
 - Valida cada token recorriendo el AFD.
 - Muestra si es válido () o inválido ().

Componentes del código (verificador_final.py)

1. Transiciones: Representan los estados y reglas del AFD.
2. Construcción del AFD: Se usa un diccionario de diccionarios.
3. Estados finales: {21, 22}, indican aceptación de un token.
4. extraer_tokens(): Usa expresiones regulares para dividir una línea en tokens.
5. validar_token(): Recorre el autómata para verificar si el token es válido.
6. validar_archivo(): Lee el archivo, extrae tokens y los valida uno por uno.

Archivo de prueba (prueba.txt)

Contiene líneas con:

- Asignaciones: $x = 5$;
- Condicionales: `if (edad >= 18) { ... }`
- Cadenas: "Bienvenido al sistema."
- Operaciones: $10 + 15, 6 * 7, 10 == 10$, etc.

Resultados esperados

Al ejecutar el script, muestra en consola:

- Línea 1: $x = 5$;
- Token válido: 'x'
- Token válido: '='
- Token válido: '5'
- Token válido: ';'
- Línea 2: nombre = "Juan" ;
- Token válido: 'nombre'
- Token válido: '='
- Token válido: '"Juan"'
- Token válido: ';'
- Línea 3: apellido = "Pérez" ;
- Token válido: 'apellido'
- Token válido: '='
- Token válido: '"Pérez"'
- Token válido: ';'
- Línea 4: edad = 25 ;
- Token válido: 'edad'
- Token válido: '='
- Token válido: '25'
- Token válido: ';'
- Línea 5: peso = 70.5 ;
- Token válido: 'peso'
- Token válido: '='
- Token válido: '70.5'
- Token válido: ';'
- Línea 6: altura = 1.75 ;
- Token válido: 'altura'
- Token válido: '='
- Token válido: '1.75'
- Token válido: ';'

Possibles mejoras o extensiones

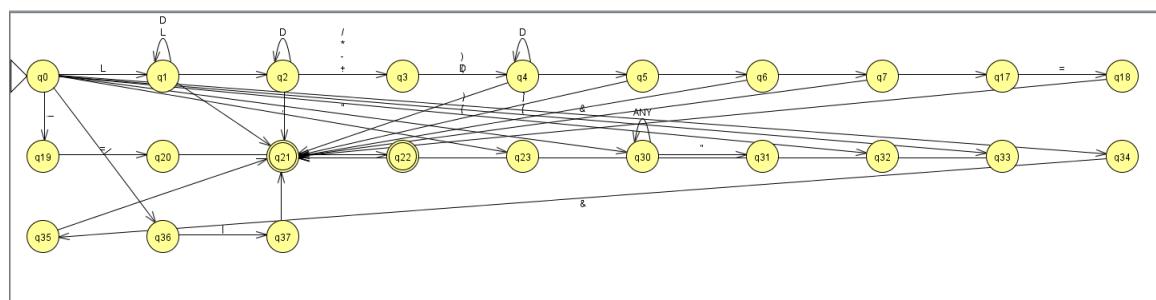
- Reconocer comentarios como // texto o /* bloque */
- Reconocer tipos de datos como int, float, bool
- Reportar número de línea y tipo de error

Autómata

El autómata finito determinista (AFD) diseñado para este proyecto permite reconocer tokens válidos usados comúnmente en lenguajes de programación como identificadores, números, operadores, símbolos y cadenas.

Este autómata fue definido visualmente en JFLAP (.jff) y convertido a una tabla de transiciones para su uso en Python.

El estado inicial es 0, y los estados de aceptación son 21 y 22.



Definición del lenguaje

El lenguaje definido está compuesto por los siguientes elementos léxicos:

- Identificadores: combinaciones de letras, números y guiones bajos que inician con letra o '_'
- Números enteros y decimales: como 25, 3.14
- Cadenas de texto: rodeadas por comillas dobles, como "Hola Mundo"
- Operadores: +, -, *, /, =, ==, !=, >, <, &&, ||
- Delimitadores y símbolos: ;, (), { }, []

Sintaxis de las instrucciones

La sintaxis utilizada en el archivo de prueba simula un lenguaje estructurado similar a C o JavaScript.

Ejemplos:

- Asignación: variable = valor ;
- Condicional if: if (condición) { instrucciones } else { instrucciones }
- Expresiones: suma = 10 + 20 ; promedio = (a + b) / 2 ;
- Comparaciones: edad >= 18 ; valor == 10 ;
- Composición de bloques: bloque = { [1, 2, 3] }

Conclusión

Este proyecto demuestra cómo aplicar autómatas finitos para la etapa léxica de un compilador, separando y validando cada componente básico del código fuente. La implementación combina teoría de lenguajes formales con programación práctica en Python.