Manual Técnico
Luis Regalado 202131920
Analizador Léxico y Sintáctico

Software:

- **Lenguaje:** Java Script

JS es uno de los lenguajes de programacion mas utilizados en el desarrollo web. Se ejecuta directamente en el navegador del usuario, por lo que los desarrolladores no necesitan recargar la pagina para realizar tareas. Lenguaje de programación que se utiliza para hacer que las páginas web sean interactivas y dinámicas

- **FrameWork:** React

Es una biblioteca de JS que se utiliza para crear interfaces de usuario (UI) para aplicaciones web y moviles, permite combinar componentes escritos por diferentes personas, equipos y organizaciones

- **Sistema Operativo:** Linux Fedora 40

Un sistema operativo moderno y eficiente, ideal para desarrolladores por su robustez y herramientas avanzadas ya que ofrece un sistema operativo libre y de codigo abierto

- **IDE:** Visual Studio Code

Un editor de código ligero y potente que soporta múltiples lenguajes de programación, ideal para el desarrollo con Node.js.

- **GITHUB**

Es un sistema de control de versiones de código y gestión de proyectos, a su vez también funciona como una plataforma de estilo de red social

Estructura del Proyecto

El proyecto de Analizador Léxico y Sintáctico está organizado de la siguiente manera:

- **src/**: Contiene los archivos fuente del proyecto
    - **components/**: Componentes principales de la aplicación React que se reutilizan en diferentes partes del proyecto
    - **lexer/**: Implementación del analizador léxico utilizando archivos NLEX
    - **parser/**: Implementación del analizador sintáctico
    - **Class/:** arbol de nodos
    - **contexts/:** para buscar archivos
    - **pages:** Vistas de alto nivel que representan cada página principal de la aplicación.

    - **services/:** Módulos para realizar llamadas a APIs o manejar la lógica de interacción con fuentes externas de datos.
    - **styles/:** estilos que se da a la web con CSS
    - **utils:** logica para el lexico, sintactico y operaciones matematicas
- **public/**: Archivos estáticos y configuraciones públicas.
- **package.json**: Archivo de configuración para dependencias y scripts de npm.
- **assets/:** Archivos estáticos como imágenes, íconos, fuentes y estilos globales.

Autómata Finito Determinista (AFD) Sintactico

Expresión Regular Asociada

La expresión regular asociada al analizador léxico es:

[a-zA-Z_][a-zA-Z0-9_]*[\+\-*/\[\]\{\}\(\)””]*

Esta expresión regular describe identificadores que:

- Comienzan con una letra (mayúscula o minúscula) o un guión bajo (_).
- Pueden contener letras, dígitos o guiones bajos después del primer carácter.

Descripción del AFD

El AFD correspondiente consta de los siguientes elementos:

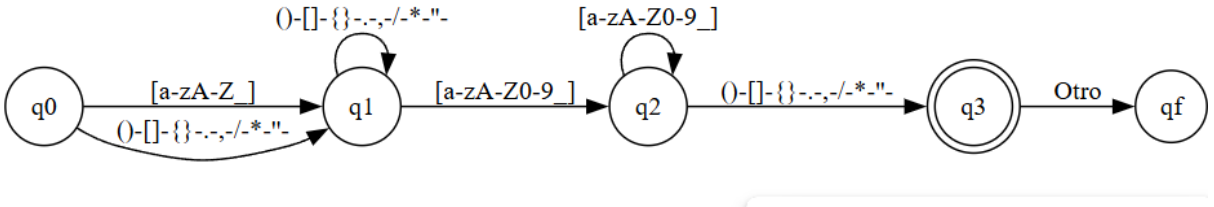
- **Estados**:
    - q0: Estado inicial.
    - q1: Estado de aceptación (identificador válido).
    - qf: Estado final (rechazo).
- **Transiciones**:
    - q0 -[a-zA-Z_]-> q1
    - q1 -[a-zA-Z0-9_]-> q1
    - q1 -[otro]-> qf

Método del Árbol

El método del árbol permite construir el AFD a partir de la expresión regular paso a paso:

- **Entrada inicial**: [a-zA-Z_][a-zA-Z0-9_]*
- **Descomposición**:
    - [a-zA-Z_]: Nodo principal para el primer carácter.
    - [a-zA-Z0-9_]*Nodo repetitivo para los siguientes caracteres.

- **Construcción del árbol**:
  - Raíz: La expresión regular completa.
  - Hijos: Componentes individuales de la expresión.
- **Conversión al AFD**:
  - Combinar nodos para representar estados y transiciones.

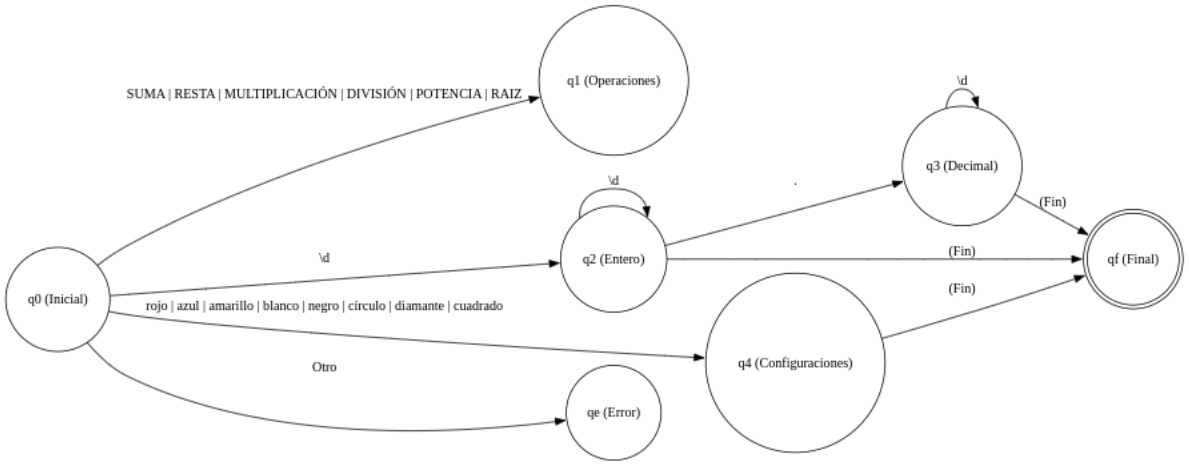


Autómata Finito Determinista (AFD) Léxico

Componentes del AFD:

- **Estados (Q):**
  - q0q_0q0: Estado inicial.
  - q1q_1q1: Reconociendo una operación válida (e.g., SUMA, MULTIPLICACIÓN).
  - q2q_2q2: Reconociendo valores numéricos enteros.
  - q3q_3q3: Reconociendo valores numéricos decimales.
  - q4q_4q4: Reconociendo configuraciones (e.g., rojo, blanco, círculo).
  - qeq_eqe: Estado de error (cuando un token no es válido).
  - qfq_fqf: Estado final.
- **Alfabeto (Σ\SigmaΣ):**
  - Letras (A-Z, a-z).
  - Dígitos (0-9).
  - Símbolos especiales (., ", :, ,, {, }).
- **Transiciones (δ\deltaδ):**
  - Las transiciones definen cómo el autómata se mueve entre los estados según el carácter leído.
- **Estado Inicial (q0q_0q0):**
  - Comienza en q0q_0q0.
- **Conjunto de Estados Finales (FFF):**
  - qfq_fqf: Estado final válido.

AFD:



Transiciones:

Estado Actual	Símbolo	Estado Siguiente
q_0	`SUMA`, `MULTIPLICACIÓN`	q_1
q_1	Fin de palabra	q_f
q_0	Dígito (`0-9`)	q_2
q_2	`.`	q_3
q_3	Dígito (`0-9`)	q_f
q_0	`rojo`, `blanco`, `círculo`	q_4
q_4	Fin de palabra	q_f
q_0	Otro	q_e

Expresión regular:
Operaciones matemáticas.
(SUMA|RESTA|MULTIPLICACIÓN|DIVISIÓN|POTENCIA|RAÍZ)

(SUMA|RESTA|MULTIPLICACIÓN|DIVISIÓN|POTENCIA|RAÍZ)

- SUMA: Identifica el operador de suma.
- RESTA: Identifica el operador de resta.
- MULTIPLICACIÓN: Identifica el operador de multiplicación.
- DIVISIÓN: Identifica el operador de división.
- POTENCIA: Identifica el operador de potencia.
- RAÍZ: Identifica el operador de raíz cuadrada.

Explicación:

- Los paréntesis () agrupan estas alternativas.

- El operador | indica que cualquiera de las palabras separadas por él es válida.

Reconocimiento de Valores Numéricos

\d+(\.\d+)?

- \d+: Reconoce uno o más dígitos consecutivos, representando un número entero.
- (\.\d+)?: Reconoce un punto decimal (.) seguido de uno o más dígitos, representando un número decimal. El signo de interrogación indica que esta parte es opcional.

Explicación:

- Los números enteros como 5 o 123 coinciden con \d+.
- Los números decimales como 3.14 o 0.001 coinciden con \.\d+, y la parte opcional permite que también se acepten números sin decimales.

Reconocimiento de Configuraciones

(red|blue|yellow|white|black|circle|diamond|box)

red, blue, yellow son las palabras que representan configuraciones válidas para el fondo, fuente o forma.

Unión:

(SUMA|RESTA|MULTIPLICACIÓN|DIVISIÓN|POTENCIA|RAIZ)|\d+(\.\d+)?|(red|blue|yellow|white|black|circle|diamond|box)

Gramática Libre de Contexto (GLC)

Notación BNF
La gramática que define las estructuras sintácticas del lenguaje aceptado es:

<programa> ::= <instruccion> | <instruccion> <programa>
<instruccion> ::= <declaracion> | <asignacion> | <condicional> | <ciclo>
<declaracion> ::= "int" <identificador> | "float" <identificador> | "string" <identificador>
<asignacion> ::= <identificador> "=" <expresion>
<condicional> ::= "if" "(" <condicion> ")" "{" <programa> "}"
<ciclo> ::= "while" "(" <condicion> ")" "{" <programa> "}"
<expresion> ::= <termino> | <termino> "+" <expresion>
<termino> ::= <factor> | <factor> "*" <termino>
<factor> ::= <identificador> | <numero>
<condicion> ::= <expresion> <operador_relacional> <expresion>
<operador_relacional> ::= "==" | "!=" | "<" | ">" | "<=" | ">="
<identificador> ::= [a-zA-Z_][a-zA-Z0-9_]*

<numero> ::= [0-9]+ | [0-9]+"."[0-9]+
<caracteres> ::= <[^>]+>|\w+|[|()=*""]|::=|\d+

Descripción

1. Representa la secuencia de instrucciones.
2. Define los tipos de instrucciones permitidas (declaración, asignación, condicionales y ciclos).
3. Modelan las operaciones aritméticas.
4. Describe las comparaciones lógicas.
5. Definen las unidades léxicas.