

Facultad de Ingeniería
Depto. de Informática y Sistemas
Área de Informática
Lenguajes Formales y Autómatas Sección 02
Inga. Damaris Campos

Walter Francisco Melendez Aguilar - 1174722
Mario Miguel Arevalo Perez - 1072123

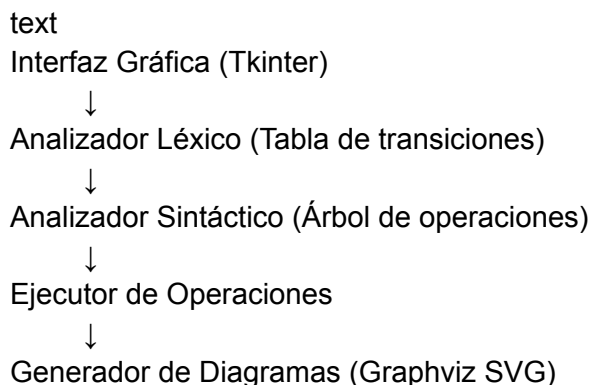
Manual Técnico - Analizador de Operaciones Aritméticas

Descripción del Sistema

Propósito

El Analizador de Operaciones Aritméticas es una aplicación desarrollada en Python que permite analizar, ejecutar y visualizar operaciones matemáticas descritas en un lenguaje XML personalizado. El sistema incluye un analizador léxico, un analizador sintáctico y un generador de diagramas de árbol.

Arquitectura General



Componentes Técnicos

1. Analizador Léxico ([analizador_lexico.py](#))

Tabla de Transiciones Implementada

text

Estado 0: Inicial

Estado 1: Reconocimiento de palabras/reservadas

Estado 2: Reconocimiento de números enteros

Estado 3: Punto decimal

Estado 4: Símbolo de apertura '<'

Estado 5: Símbolo de cierre '>'

Estado 6: Símbolo de cierre completo '</'

Estado 7: Igual '='

Estado 8: Números decimales

Tokens Reconocidos

- **PALABRA_RESERVADA**: OPERACION, NUMERO, SUMA, RESTA, MULTIPLICACION, DIVISION, POTENCIA, RAIZ, INVERSO, MOD
- **NUMERO_ENTERO**: Números sin punto decimal
- **NUMERO_DECIMAL**: Números con punto decimal
- **SIMBOLO_APERTURA**: '<'
- **SIMBOLO_CIERRE**: '>'
- **SIMBOLO_CIERRE_COMPLETO**: '</'
- **IGUAL**: '='

2. Analizador Sintáctico (**analizador_sintactico.py**)

Estructura del Árbol de Sintaxis

```
python
```

```
class NodoArbol:
```

```
    tipo: 'OPERACION' | 'NUMERO'
```

```
    valor: str | float
```

```
    hijos: List[NodoArbol]
```

```
    resultado: float
```

Gramática del Lenguaje

```
text
```

```
OPERACION → <Operacion= TIPO_OP> CONTENIDO </Operacion>
```

```
CONTENIDO → (NUMERO | OPERACION)*
```

```
NUMERO → <Numero> VALOR </Numero>
```

```
TIPO_OP → SUMA | RESTA | MULTIPLICACION | DIVISION | POTENCIA | RAIZ |  
INVERSO | MOD
```

3. Ejecutor de Operaciones

Algoritmo de Ejecución

```
python
```

```
def ejecutar(nodo):
```

```
    if nodo.tipo == 'NUMERO':
```

```
        return float(nodo.valor)
```

```
    else:
```

```
        resultados_hijos = [ejecutar(hijo) for hijo in nodo.hijos]
```

```
        return aplicar_operacion(nodo.valor, resultados_hijos)
```

Operaciones Implementadas

- **SUMA**: `sum(resultados_hijos)`
- **RESTA**: `resultados_hijos[0] - sum(resultados_hijos[1:])`

- **MULTIPLICACION:** producto de todos los elementos
- **DIVISION:** resultados_hijos[0] / resultados_hijos[1] / ...
- **POTENCIA:** base ^ exponente
- **RAIZ:** raiz_n(radicando)
- **INVERSO:** 1 / numero
- **MOD:** dividendo % divisor

4. Generador de Diagramas (**generador_arbol_graphviz.py**)

Especificaciones Graphviz

- **Formato de salida:** SVG
- **Orientación:** Top-Bottom (vertical)
- **Nodos:** Cajas con bordes redondeados
- **Colores:**
 - Números: Verde (#e8f5e8)
 - Suma/Resta: Azul (#e3f2fd)
 - Multiplicación/División: Naranja (#fff3e0)
 - Potencia/Raíz: Lila (#f3e5f5)

Estructura de Archivos

```

text
proyecto/
src/
├── analizador_lexico.py      # Analizador léxico con tabla de transiciones
├── analizador_sintactico.py  # Analizador sintáctico y ejecutor
├── generador_arbol_graphviz.py # Generador de diagramas SVG
├── proyecto.py              # Interfaz de usuario Tkinter
documentos
├── manual_tecnico.pdf        # Este manual
├── manual_de_usuario.pdf     # Manual usuario
├── requirements.txt          # Dependencias
ejemplos/                    # Ejemplos de uso
├── base.txt

```

Requisitos del Sistema

Software Requerido

- **Python:** 3.8 o superior
- **Graphviz:** Para generación de diagramas (opcional)

Instalación de Graphviz

```

bash
# Windows: Descargar desde https://graphviz.org/download/

```

Linux (Ubuntu/Debian): sudo apt-get install graphviz

macOS: brew install graphviz

Dependencias Python

txt

tkinter (incluido en Python estándar)

subprocess (incluido en Python estándar)

tempfile (incluido en Python estándar)

os (incluido en Python estándar)

Formatos de Entrada/Salida

Formato de Entrada

xml

<Operacion= TIPO_OPERACION>

<Numero> valor1 </Numero>

<Numero> valor2 </Numero>

<Operacion= OPERACION_ANIDADA>

...

</Operacion>

</Operacion>

Formatos de Salida

1. **Resultados.html**: Tabla con resultados de operaciones
2. **Errores.html**: Reporte de errores léxicos
3. **arbol_operacion_X.svg**: Diagramas de árbol en SVG

Algoritmos Implementados

1. Análisis Léxico

- **Método**: Tabla de transiciones con estados
- **Complejidad**: $O(n)$ donde n es la longitud del texto
- **Manejo de errores**: Reporte de posición (línea, columna)

2. Análisis Sintáctico

- **Método**: Recursivo por descendencia
- **Estructura**: Árbol de sintaxis abstracta (AST)
- **Validación**: Estructura XML bien formada

3. Generación de Diagramas

- **Herramienta**: Graphviz DOT language

- **Enfoque:** Recorrido en profundidad del AST
- **Personalización:** Colores y estilos por tipo de nodo

Esquema de Colores

Diagramas SVG

- **Números:** Fondo verde claro, borde verde
- **Operaciones básicas:** Fondo azul claro, borde azul
- **Operaciones avanzadas:** Fondo naranja claro, borde naranja
- **Operaciones especiales:** Fondo lila claro, borde morado

Interfaz de Usuario

- **Fondo principal:** #f5f5f5
- **Botones:** Tema por defecto de Tkinter
- **Área de texto:** Fondo blanco, texto negro

Manejo de Errores

Errores Léxicos

- Caracteres no reconocidos
- Números mal formados
- Símbolos incompletos

Errores de Ejecución

- División por cero
- Operaciones con operandos insuficientes
- Valores no numéricos

Errores de Graphviz

- Graphviz no instalado
- Permisos de escritura
- Memoria insuficiente

Flujo de Datos

text

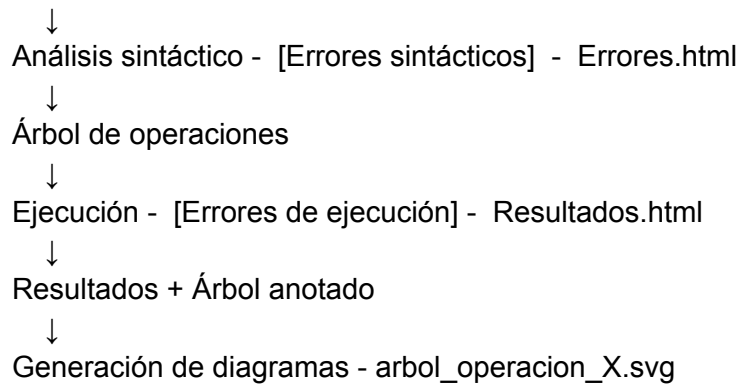
Texto de entrada



Análisis léxico - [Errores léxicos] - Errores.html



Lista de tokens



Consideraciones de Rendimiento

Optimizaciones Implementadas

1. **Tabla de transiciones:** Búsqueda $O(1)$ para tokens
2. **Árbol de operaciones:** Estructura eficiente para ejecución
3. **Generación SVG:** Un solo proceso Graphviz por operación

Límites Conocidos

- Anidamiento muy profundo puede causar recursion limits
- Archivos muy grandes pueden consumir memoria considerable
- Graphviz puede ser lento con árboles muy complejos

Mantenimiento y Extensión

Agregar Nueva Operación

1. Agregar token en `analizador_lexico.py`
2. Implementar lógica en `EjecutorOperaciones.ejecutar()`
3. Definir colores en `GeneradorArbolGraphviz._obtener_color()`

Modificar Esquema de Colores

Editar métodos en `GeneradorArbolGraphviz`:

- `_obtener_color()`
- `_obtener_borde()`
- `_obtener_color_linea()`

Extender Formato de Entrada

Modificar gramática en `AnalizadorSintactico.parsear_operacion()` y `parsear_numero()`

Desarrollado para el curso de Lenguajes Formales y Automatas - 2025S2