



Tecnológico Nacional de México
**INSTITUTO TECNOLÓGICO
DE PACHUCA**

Documentación del Analizador Léxico y Sintáctico

INGENIERIA EN SISTEMAS COMPUTACIONALES

NOMBRE DE LA ASIGNATURA

Lenguajes y Autómatas I

Docente:

Rodolfo Baumé Lazcano

Alumnos:

García Lugo Itzel Paola

Quintos Cabeza Diego

Vera Rodríguez Héctor Gabriel

Fecha

30/05/2024

Introducción

Este documento describe el diseño e implementación de un analizador léxico y sintáctico para expresiones aritméticas simples. El analizador es capaz de procesar expresiones que incluyen números enteros y de punto flotante, operadores aritméticos básicos (+, -, *, /) y paréntesis. Además, el analizador puede identificar y reportar errores léxicos y sintácticos.

Diseño del Analizador

Analizador Léxico (Lexer)

El analizador léxico se encarga de dividir la entrada en una secuencia de tokens. Los tokens son definidos mediante expresiones regulares y pueden representar números, operadores, paréntesis, espacios y caracteres inesperados. El Lexer genera una lista de tokens que se utilizarán posteriormente en el análisis sintáctico.

Especificación de Tokens

NUMBER: Números enteros o de punto flotante (\d+(\.\d*)?)

ADD: Operador de suma (\+)

SUB: Operador de resta (-)

MUL: Operador de multiplicación (*)

DIV: Operador de división (/)

LPAREN: Paréntesis izquierdo (\()

RPAREN: Paréntesis derecho (\))

SKIP: Espacios y tabuladores ([\t]+)

MISMATCH: Cualquier otro carácter (.)

Clase Lexer

La clase Lexer tiene las siguientes responsabilidades:

- `tokenize`: Convierte la cadena de entrada en una lista de tokens utilizando las expresiones regulares definidas.
- `next_token`: Devuelve el siguiente token de la lista de tokens.

Analizador Sintáctico (Parser)

El analizador sintáctico utiliza un enfoque de análisis descendente predictivo para evaluar las expresiones aritméticas. Las principales funciones del analizador sintáctico son:

- `factor`: Maneja números y expresiones entre paréntesis.
- `term`: Maneja multiplicación y división.
- `expr`: Maneja suma y resta.

Clase Parser:

La clase Parser tiene las siguientes responsabilidades:

- `eat`: Verifica y consume el token esperado, avanzando al siguiente token. Si el token actual no es el esperado, lanza un `SyntaxError`.
- `factor`: Procesa un número o una expresión entre paréntesis.
- `term`: Procesa términos que pueden incluir operadores de multiplicación y división.
- `expr`: Procesa expresiones que pueden incluir operadores de suma y resta.
- `parse`: Inicia el análisis sintáctico y devuelve el resultado de la evaluación de la expresión.

Casos de Prueba

Expresión válida con números enteros:

Entrada: "60 / 3 (60-40)"

Salida esperada: Resultado del análisis sintáctico: 400

Expresión válida con números de punto flotante:

Entrada: "89.72 + 67.56 * (4 /10)"

Salida esperada: Resultado del análisis sintáctico: 116.744

Error léxico (carácter inesperado):

Entrada: "389 + 504 a"

Salida esperada: Error de runtime: Error léxico: Carácter inesperado 'a' en posición 11

Error sintáctico (token inesperado):

Entrada: "3 + 5 * (10 -)"

Salida esperada: Error de sintaxis: Error sintáctico: Se esperaba NUMBER pero se encontró RPAREN en la posición [['RPAREN', ')']]

Expresión con espacios y tabuladores:

Entrada: " 3 + 5 * (10 - 4)"

Salida esperada: Resultado del análisis sintáctico: 33

```
GNU Octave (GUI).Ink  lenguaje.py  sintactico.py X
sintactico > sintactico.py > ...
1 import re
2
3 # Definición de tokens
4 token_specification = [
5     ('NUMBER', r'\d+(\.\d*)?'), # Números enteros o de punto flotante
6     ('ADD', r'\+'), # Suma
7     ('SUB', r'\-'), # Resta
8     ('MUL', r'\*'), # Multiplicación
9     ('DIV', r'\/'), # División
10    ('LPAREN', r'\('), # Paréntesis izquierdo
11    ('RPAREN', r'\)'), # Paréntesis derecho
12    ('SKIP', r'[ \t]+'), # Espacios y tabuladores
13    ('MISMATCH', r'.'), # Cualquier otro carácter
14 ]
15
16 tok_regex = '|'.join('%s' % pair[0] for pair in token_specification)
17
18 class Lexer:
19     def __init__(self, code):
20         self.tokens = []
21         self.tokenize(code)
22
23     def tokenize(self, code):
24         for mo in re.finditer(tok_regex, code):
25             kind = mo.lastgroup
26             value = mo.group()
27             if kind == 'NUMBER':
28                 value = float(value) if '.' in value else int(value)
29             elif kind == 'SKIP':
30                 continue
31             elif kind == 'MISMATCH':
32                 raise RuntimeError(f'{value!r} inesperado en {mo.start()}')
```

```
GNU Octave (GUI).Ink  lenguaje.py  sintactico.py X
sintactico > sintactico.py > ...
18 class Lexer:
19     def tokenize(self, code):
20         for mo in re.finditer(tok_regex, code):
21             kind = mo.lastgroup
22             value = mo.group()
23             if kind == 'NUMBER':
24                 value = float(value) if '.' in value else int(value)
25             elif kind == 'SKIP':
26                 continue
27             elif kind == 'MISMATCH':
28                 raise RuntimeError(f'{value!r} inesperado en {mo.start()}')
29             self.tokens.append((kind, value))
30
31     def next_token(self):
32         return self.tokens.pop(0) if self.tokens else ('EOF', None)
33
34 class Parser:
35     def __init__(self, lexer):
36         self.lexer = lexer
37         self.current_token = lexer.next_token()
38
39     def eat(self, token_type):
40         if self.current_token[0] == token_type:
41             self.current_token = self.lexer.next_token()
42         else:
43             raise SyntaxError(f"Se esperaba {token_type} pero se encontró {self.current_token[0]}")
44
45     def factor(self):
46         if self.current_token[0] == 'NUMBER':
47             value = self.current_token[1]
48             self.eat('NUMBER')
49             return value
50         elif self.current_token[0] == 'LPAREN':
51             self.eat('LPAREN')
52             result = self.expr()
53             self.eat('RPAREN')
54             return result
55         else:
56             raise SyntaxError(f"Token inesperado {self.current_token[0]}")
57
58
59
60
61
```

```
GNU Octave (GUI).lnk  lenguaje.py  sintactico.py X
sintactico > sintactico.py > ...

38 class Parser:
39     # base syntactical (tokens: multiplicative, self.current_token[0])
40
41
42     def term(self):
43         result = self.factor()
44         while self.current_token[0] in ('MUL', 'DIV'):
45             if self.current_token[0] == 'MUL':
46                 self.eat('MUL')
47                 result *= self.factor()
48             elif self.current_token[0] == 'DIV':
49                 self.eat('DIV')
50                 result /= self.factor()
51         return result
52
53     def expr(self):
54         result = self.term()
55         while self.current_token[0] in ('ADD', 'SUB'):
56             if self.current_token[0] == 'ADD':
57                 self.eat('ADD')
58                 result += self.term()
59             elif self.current_token[0] == 'SUB':
60                 self.eat('SUB')
61                 result -= self.term()
62         return result
63
64     def parse(self):
65         return self.expr()
66
67 # Uso del analizador léxico y sintáctico
68 code = "33 + 5 * (10 - 4)"
69 lexer = Lexer(code)
70
71 # Mostrar los tokens generados por el analizador léxico
```

```
# Uso del analizador léxico y sintáctico
code = "33 + 5 * (10 - 4)"
lexer = Lexer(code)

# Mostrar los tokens generados por el analizador léxico
print("Tokens generados por el analizador léxico:")
for token in lexer.tokens:
    print(token)

parser = Parser(lexer)

try:
    result = parser.parse()
    print(f"\nResultado del análisis sintáctico: {result}")
except SyntaxError as e:
    print(f"Error de sintaxis: {e}")
except RuntimeError as e:
    print(f"Error de runtime: {e}")
```

Caso de prueba 1.

```
OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR

('NUMBER', 60)
('DIV', '/')
('NUMBER', 3)
('MUL', '*')
('LPAREN', '(')
('NUMBER', 60)
('SUB', '-')
('NUMBER', 40)
('RPAREN', ')')

Resultado del análisis sintáctico: 400.0
PS C:\Users\gabor\OneDrive\Documentos\SEMESTRE 6TO\Automatas\analizador sintactico>
```

Caso de prueba 2.

```
OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR

('NUMBER', 89.72)
('ADD', '+')
('NUMBER', 67.56)
('MUL', '*')
('LPAREN', '(')
('NUMBER', 4)
('DIV', '/')
('NUMBER', 10)
('RPAREN', ')')

Resultado del análisis sintáctico: 116.744
PS C:\Users\gabor\OneDrive\Documentos\SEMESTRE 6TO\Automatas\analizador sintactico>
```

Caso de prueba 3.

```
OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR

PS C:\Users\gabor\OneDrive\Documentos\SEMESTRE 6TO\Automatas\analizador sintactico> & C:/Users/gabor/AppData/Local/Microsoft/Windows/
Automatas/analizador sintactico/analizador.py
Traceback (most recent call last):
  File "c:\Users\gabor\OneDrive\Documentos\SEMESTRE 6TO\Automatas\analizador sintactico\analizador.py", line 93, in <module>
    lexer = Lexer(code)
    ~~~~~
  File "c:\Users\gabor\OneDrive\Documentos\SEMESTRE 6TO\Automatas\analizador sintactico\analizador.py", line 21, in __init__
    self.tokenize(code)
  File "c:\Users\gabor\OneDrive\Documentos\SEMESTRE 6TO\Automatas\analizador sintactico\analizador.py", line 32, in tokenize
    raise RuntimeError(f'{value!r} valor inesperado')
RuntimeError: 'a' valor inesperado
PS C:\Users\gabor\OneDrive\Documentos\SEMESTRE 6TO\Automatas\analizador sintactico>
```

Caso de prueba 4.

```
OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR

Automatas/analizador sintactico/analizador.py"
Tokens generados por el analizador léxico:
('NUMBER', 3)
('ADD', '+')
('NUMBER', 5)
('MUL', '*')
('LPAREN', '(')
('NUMBER', 10)
('SUB', '-')
('RPAREN', ')')
Error de sintaxis: Token inesperado RPAREN
PS C:\Users\gabor\OneDrive\Documentos\SEMESTRE 6TO\Automatas\analizador sintactico>
```

Caso de prueba 5.

```
OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SEARCH ERROR

('NUMBER', 3)
('ADD', '+')
('NUMBER', 5)
('MUL', '*')
('LPAREN', '(')
('NUMBER', 10)
('SUB', '-')
('NUMBER', 4)
('RPAREN', ')')

Resultado del análisis sintáctico: 33
PS C:\Users\gabor\OneDrive\Documentos\SEMESTRE 6TO\Automatas\analizador sintactico>
```