CS323 Documentation

About 2 pages

1. **Problem Statement**

   This assignment implements a lexical analyzer (lexer) for Rat26S source code, which reads an input .rat program and returns tokens one at a time in a record containing the token type and the lexeme (character sequence from the source code). The tokens include identifiers, integers, and real values recognized by a Finite State Machine (FSM), as well as Rat26S keywords, operators, and separators.

2. **How to use your program**

   1. *Code a program using Rat26S syntax with a .rat file format*
   2. *Using the terminal command run*
      a. *"python lexer.py"*
      b. *Enter the file path for the input file*
      c. *Enter the file path for the output file*
   3. *Using a text editor, open the newly created text file*
      a. *or using the terminal command "cat <Output File>.txt"*
   4. *The output file will contain the record of recognized tokens and their respective lexeme*

3. **Design of your program**

   The Major Components of the program include:

   Token Record (Token dataclass)

   - *Type: keyword, identifier, integer, real, operator, separator, and unknown*
   - *Lexeme: the character sequence from the source code*

   Lexer (Lexer Class)

   - *The lexer will process the source code file using a pointer (position) calls on the next_token() method that returns one token per call*

Other methods include:

- *_peek(k) – which peeks at the next input*
- *_advance() – which takes one character as input and moves the pointer*
- *_skip_ignored() – which skips whitespaces and block comments*
- *_scan_identifier_fsm() – Finite State Machine which scans for identifiers*

- *_scan_number_fsm() – Finite State Machine which scans for integers and reals*
- *_scan_operator() – which recognizes Rat26S operators*

Main Driver

- *Run_lexer(input_path, output_path) – which will .rat file into a string, repeatedly calling the next_token() until the end of file, and writes a record table of token/lexeme pairs to the output file.*

The Data Structures Used

- ***Sets** for fast testing:*
  - *Keywords: reserved words like if, while, integer, real, etc*
  - *Separators: symbols like "(", ")", "{", "}", ",", ";", "@"*
  - *Operators: Operators include =, <=, ==, etc*
  - *Whitespace: spaces, tabs, newlines "\n", returns, etc*
- ***String Input Buffer**: Source code loaded into a single string*
- ***Integer Index Pointer**: **position** tracks the pointer's position*
- ***Token Record**: Each output token, a type and its lexeme*

REs:

L = [A-Za-z]

D = [0-9]

ID = L (l|D|_)*

INT = D+

Real = D+.D+

NFSM using Thompson:

INTEGERS

Σ = {d}

Q = {1, 2, 3, 4, 5, 6}

q0 = 1

F = {6}

N:

State	*D*	*ε*
1	*{2}*	*{}*
2	*{}*	*{5}*
3	*{4}*	*{}*
4	*{}*	*{3, 6}*
5	*{}*	*{3, 6}*
6	*{}*	*{}*

DFSM

Σ = {d, other}

Q = {0, 1, 2}
q0 = 0
F = {1}
N:

State	*d*	*Other*
0	*1*	*2*
1	*1*	*2*
2	*2*	*2*

REAL
Σ = {d, .}
Q = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}
q0 = 1
F = {14}
N:

State	*d*	*.*	*ε*
1	*{2}*	*{}*	*{}*
2	*{}*	*{}*	*{5}*
3	*{4}*	*{}*	*{}*
4	*{}*	*{}*	*{3, 6}*
5	*{}*	*{}*	*{3, 6}*
6	*{}*	*{}*	*{7}*
7	*{}*	*{8}*	*{}*
8	*{}*	*{}*	*{9}*
9	*{10}*	*{}*	*{}*
10	*{}*	*{}*	*{13}*
11	*{12}*	*{}*	*{}*
12	*{}*	*{}*	*{11, 14}*
13	*{}*	*{}*	*{11, 14}*
14	*{}*	*{}*	*{}*

DFSM

Σ = {d, ., other}

Q = {0, 1, 2, 3, 4}

q0 = 0

F = {3}

N:

State	*D*	*.*	*Other*
0	*1*	*4*	*4*
1	*1*	*2*	*4*
2	*3*	*4*	*4*
3	*3*	*4*	*4*
4	*4*	*4*	*4*

IDENTIFIER

Σ = { l, d, _}

Q = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

q0 = 1

F = {4}

N:

State	*l*	*d*	*_*	*ε*
1	*{2}*	*{}*	*{}*	*{}*
2	*{}*	*{}*	*{}*	*{3}*
3	*{}*	*{}*	*{}*	*{4, 5}*
4	*{}*	*{}*	*{}*	*{}*
5	*{}*	*{}*	*{}*	*{6, 8, 10}*
6	*{7}*	*{}*	*{}*	*{}*
7	*{}*	*{}*	*{}*	*{12}*
8	*{}*	*{9}*	*{}*	*{}*
9	*{}*	*{}*	*{}*	*{12}*
10	*{}*	*{}*	*{11}*	*{}*
11	*{}*	*{}*	*{}*	*{12}*
12	*{}*	*{}*	*{}*	*{4, 5}*

DFSM
Σ = { l, d, _}
Q = {0, 1, 2}
q0 = 0
F = {1}
N:

State	*l*	*d*	*_*	*Other*
0	*1*	*2*	*2*	*2*
1	*1*	*1*	*1*	*2*
2	*2*	*2*	*2*	*2*

4. **Any Limitation**
   <All features are running according to the assignment but you limit your program due to resource limitations, such as Maximum number of lines in the source code, size of the identifier, integer etc. ***Say 'None' if there is no limitation****>*

5. **Any shortcomings**
   <Anything you could NOT implement although that is required by the Assignment. ***Say 'None' if there is no shortcoming****>*