Laboratory Work Nr.3 Lexer Scanner

Cretu Cristian

April 1, 2024

Theory

Regular expressions, often abbreviated as regex or regexp, are powerful tools used in computer science and programming for pattern matching within strings of text. They provide a concise and flexible means of searching, extracting, and manipulating textual data based on specific patterns. Utilized across various programming languages and text processing utilities, regular expressions enable developers to perform tasks such as validation of input, searching for specific patterns within large datasets, and text manipulation with precision and efficiency. The syntax of regular expressions consists of a combination of literal characters and metacharacters, forming patterns that define the desired matches. With their versatility and widespread adoption, regular expressions serve as an indispensable tool for tasks ranging from simple string manipulation to complex data extraction and transformation. However, mastering regular expressions requires understanding their syntax, metacharacters, and application-specific nuances to leverage their full potential effectively.

Objectives

1. Write and cover what regular expressions are, what they are used for.
2. Take your variant code
3. Write a code that will generate valid combinations of symbols conform given regular expressions (examples will be shown).
4. In case you have an example, where symbol may be written undefined number of times, take a limit of 5 times (to evade generation of extremely long combinations).
5. Bonus point: write a function that will show sequence of processing regular expression (like, what you do first, second and so on)

Implementation description

RegEx Operators

I have created custom functions for handling different Regex operators, particularly the +, |, ∗, ^ operators.

For example. here is the OR operator:

```
def or_operator(options: list[str] = None):
    symbol = random.choice(options)
    print(f'Choose {symbol}')
    return symbol
```

Here is the Star Operator

```
def star_operator(symbol: str):
    times = random.randint(0, infty)
    answer = symbol*times
    print(f'Repeat {symbol} {times} times')
    return answer
```

After defining these functions, we can simply construct our Regular Expression based on the defined rules by adding the result of sequential operators computations.

```
def first_regex():
    "0(P|Q|R)+2(3|4)"
    answer = ""
    answer += const_operator("0")
    choice = or_operator(["P", "Q", "R"])
    answer += plus_operator(choice)
    answer += "2"
    answer += or_operator(["3", "4"])
    return answer


def second_regex():
    "A*B(C|D|E)F(G|H|i)^2"
    answer = ""
    answer += star_operator("A")
    answer += const_operator("B")
    answer += or_operator(["C", "D", "E"])
    answer += const_operator("F")
    answer += n_operator(or_operator(["G", "H", "i"]), 2)
    return answer


def third_regex():
    "J+K(L|M|N)*0?(P|Q)^3"
    answer = ""
    answer += plus_operator("J")
    answer += const_operator("K")
```

```
    answer += star_operator(or_operator(["L", "M", "N"]))
    answer += question_operator("O")
    answer += n_operator(or_operator(["P", "Q"]), 3)
    return answer
```

Results

First Regular Expression: $O(P|Q|R)^+2(3|4)$

```
Constant O
Choose Q from ['P', 'Q', 'R']
Repeat Q 2 times
Constant 2
Choose 3 from ['3', '4']
OQQ23
```

Second Regular Expression: $A^*B(C|D|E)F(G|H|i)^2$

```
Repeat A 5 times
Constant B
Choose D from ['C', 'D', 'E']
Constant F
Choose G from ['G', 'H', 'i']
Repeat G 2 times
AAAAABDFGG
```

Third Regular Expression: $J^+K(L|M|N)^*O?(P|Q)^3$

```
Repeat J 2 times
Constant K
Choose L from ['L', 'M', 'N']
Repeat L 2 times
Not Empty O
Choose P from ['P', 'Q']
Repeat P 3 times
JJKLLOPPP
```

Conclusions

In conclusion, this laboratory work provided valuable insights into regular expressions, emphasizing their role in pattern matching within text. Practical implementation involved creating custom functions for handling regex operators and generating valid symbol combinations. Exploring operators like *, +, —, ?, and ^ enhanced understanding of pattern definition and repetition

control. Future enhancements could include supporting additional regex operators, optimizing code for efficiency, and implementing error handling mechanisms. Overall, this experience deepened comprehension of regex principles and their practical application, setting a foundation for further exploration in future projects.