Laboratory Work Nr.4

Regular expressions

Course: Formal Languages & Finite Automata

Author: Tofan Liviu

Group: FAF-223

Chişinău – 2024

Theory

A regular expression (shortened as regex or regexp), sometimes referred to as rational expression, is a sequence of characters that specifies a match pattern in text. Usually, such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation. Regular expression techniques are developed in theoretical computer science and formal language theory.

The concept of regular expressions began in the 1950s when the American mathematician Stephen Cole Kleene formalized the concept of a regular language. They came into common use with Unix text-processing utilities. Different syntaxes for writing regular expressions have existed since the 1980s, one being the POSIX standard and another, widely used, being the Perl syntax.

Regular expressions are used in search engines, in search and replace dialogs of word processors and text editors, in text processing utilities such as sed and AWK, and in lexical analysis. Regular expressions are supported in many programming languages. Library implementations are often called an "engine", and many of these are available for reuse.

Objectives

1. Write and cover what regular expressions are, what they are used for;

2. Below you will find 3 complex regular expressions per each variant. Take a variant depending on your number in the list of students and do the following:

    a. Write a code that will generate valid combinations of symbols conform given regular expressions (examples will be shown).

    b. 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);

    c. Bonus point: write a function that will show sequence of processing regular expression (like, what you do first, second and so on)

Implementation

I defined some functions for operators. These functions are utilized to construct strings adhering to predefined regex patterns in the first_regex(), second_regex(), and third_regex() functions.

choose(options):

Purpose: Randomly selects an option from a list of provided options.

Arguments: options - A list of strings representing available choices.

Behavior: Randomly selects one option from the provided list, prints the chosen option for reference, and returns the selected option as a string.

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

Figure 1. choose() function

plus(symbol):

Purpose: Generates a repeated sequence of a symbol a random number of times.

Arguments: symbol - A string representing the symbol to repeat.

Behavior: Generates a repeated sequence of the provided symbol a random number of times (between 1 and 5), prints the operation performed, and returns the resulting string.

```
def plus(symbol: str):
    init_symbol = symbol
    result = ""
    last_i = 1
    for i in range(1, random.randint( a: 1,  b: 5)):
        result += symbol
        last_i = i
    print(f'Repeat {init_symbol} {last_i} times')
    return result
```

Figure 2. plus() function

star(symbol):

Purpose: Generates a repeated sequence of a symbol a random number of times (up to 5 times).
Arguments: symbol - A string representing the symbol to repeat.

Behavior: Generates a repeated sequence of the provided symbol a random number of times (up to 5 times), prints the operation performed, and returns the resulting string.

```
def star(symbol: str):
    init_symbol = symbol
    result = ""
    repetitions = random.randint( a: 0,  b: 5)
    for i in range(0, repetitions):
        result += symbol
    print(f'Repeat {init_symbol} {repetitions} times')
    return result
```

Figure 3. star() function

power(symbol, n):

Purpose: Generates a repeated sequence of a symbol a specified number of times.

Arguments: symbol - A string representing the symbol to repeat, n - An integer specifying the number of times to repeat the symbol.

Behavior: Generates a repeated sequence of the provided symbol the specified number of times, prints the operation performed, and returns the resulting string.

```
def power(symbol: str, n: int):
    repeated_symbol = ""
    for i in range(0, n):
        repeated_symbol += symbol
    print(f'Repeat {symbol} {n} times')
    return repeated_symbol
```

Figure 4. power() function

const(symbol):

Purpose: Returns a constant symbol without repetition.

Arguments: symbol - A string representing the constant symbol.

Behavior: Prints the provided constant symbol, and returns it.

```
def const(symbol: str):
    print(f'Constant {symbol}')
    return symbol
```

Figure 5. const() function

find_chooses(regex):

Purpose: Extracts choices enclosed in parentheses from the given regular expression and returns the matched options along with the index to advance in the loop after processing the choice. This function is called in *find_operators()* while iterates through expression and finds “(“.

Arguments: regex - A string representing the regular expression containing the choices enclosed in parentheses.

Behavior: Searches for choices enclosed in parentheses within the provided regular expression. It extracts the matched options, removes any '|' symbols, and returns the matched options as a list along with the index representing the position to move in the loop after processing the choice.

```
def find_chooses(regex):
    stack = []
    for i, char in enumerate(regex):
        if char == "(":
            stack.append(i)
        elif char == ")":
            if stack:
                matches = []
                start_index = stack.pop()
                substring = regex[start_index + 1:i]
                index = len(substring) + 2
                substring = substring.replace("|", "")
                matches.extend(substring)
    return matches, index
```

Figure 6. find_chooses() function

find_operators(regex):

Purpose: Processes operators in the given regular expression and generates the corresponding textual representations.

Arguments: regex - A string representing the regular expression containing operators.

Behavior: Iterates through each character in the provided regular expression. It identifies operators and operands, performs the corresponding operations, and generates the textual representations accordingly. The processed representations are concatenated into a result string, which is returned after completing the iteration.

```
1 usage   LiviuTofan
def find_operators(regex):
    num = ""
    result = ""
    i = 0
    while i < len(regex):
        char = regex[i]
        # Find chooses
        if char == "(":
            result += const(num)
            matches, index = find_chooses(regex[i:])
            result += choose(matches)
            i += index  # Increment index to skip over processed characters
            num = ""
            continue
        # Check for constants
        elif char.isalnum():
            num += char
        # Perform * operation
        elif char == "*":
            if num != "":
                result += star(num)
            else:
                result += star(result[-1])
            num = ""
        # Perform + operation
        elif char == "+":
            if num != "":
                result += plus(num)
            else:
                result += plus(result[-1])
            num = ""
        # Perform power operation
        elif char == "^":
            if num != "":
                result += power(num, int(regex[i+1]))
            else:
                result += power(result[-1], int(regex[i + 1]))
            i += 1
            num = ""

        i += 1  # Increment index to move to the next character
    # Perform remaining operations if num is not empty
    if num:
        result += const(num)

    return result
```

Figure 7. find_operators() function

And finally perform all needed operations for all 3 given regular expressions in Variant – 1V:

```
regexes = ["(S|T)(U|V)W*Y+24", "L(M|N)O^3P*Q(2|3)", "L(M|N)O^3P*Q(2|3)"]
for regex in regexes:
    print("------------")
    print("Regex: ", regex)
    print("------------")
    result = find_operators(regex)
    print(result)
```

Figure 8. Execute functions for given regexes

Conclusion

In this laboratory work, we explored the concept of regular expressions (regex) and their practical implementation in Python. By defining functions to handle various regex operations such as choice, repetition, and constants, we gained a deeper understanding of how regex patterns can be constructed programmatically. Through the creation of specific functions for different regex patterns, we demonstrated the versatility and flexibility of regex in matching complex text patterns.

This laboratory work equipped us with the knowledge and skills to:

- Understand the syntax and usage of regular expressions.
- Implement regex patterns using Python functions.
- Handle operations such as choice, repetition, and constants within regex patterns.
- Construct custom regex patterns tailored to specific requirements.

By encapsulating regex functionality within functions, we've created a modular and reusable codebase that can be easily adapted to handle a wide range of regex patterns. This code can serve as a foundational tool for text processing tasks, such as data extraction, validation, and transformation, across various domains and applications. Overall, this laboratory work has provided a solid foundation for harnessing the power of regular expressions in practical programming tasks.