Formal Languages & Finite Automata

Laboratory work 4:

Regular expressions

Student: Grigoraș Dumitru
Verified: prof., Cojuhari Irina
univ. assist., Crețu Dumitru

Chișinău, 2024

Theory

Regular Expressions:

Regular expressions (regex or regexp) are powerful tools used for pattern matching and text manipulation. They are widely used in various fields, including programming, data validation, search algorithms, and lexical analysis. Regular expressions provide a concise and flexible way to define patterns in strings, allowing for efficient matching and extraction of information.

Definition and Syntax:

- A regular expression is a sequence of characters that define a search pattern. It can include literal characters, metacharacters, quantifiers, and grouping constructs.
- Metacharacters such as . **(dot), * (asterisk), + (plus), ? (question mark), ^ (caret), $ (dollar sign), \ (backslash), | (pipe), [] (character class), {} (curly braces), and () (parentheses)** have special meanings in regular expressions.
- Quantifiers such as *** (zero or more), + (one or more), ? (zero or one), {n} (exactly n occurrences), {n,} (n or more occurrences),** and **{n,m}** (between n and m occurrences) specify the number of times a pattern should repeat.
- Grouping constructs such as **(...)** (capturing group), **(?:...)** (non-capturing group), **(?=...)** (positive lookahead), **(?<=...)** (positive lookbehind), **(?!...)** (negative lookahead), and **(?<!...)** (negative lookbehind) allow for complex pattern matching and conditional logic.

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

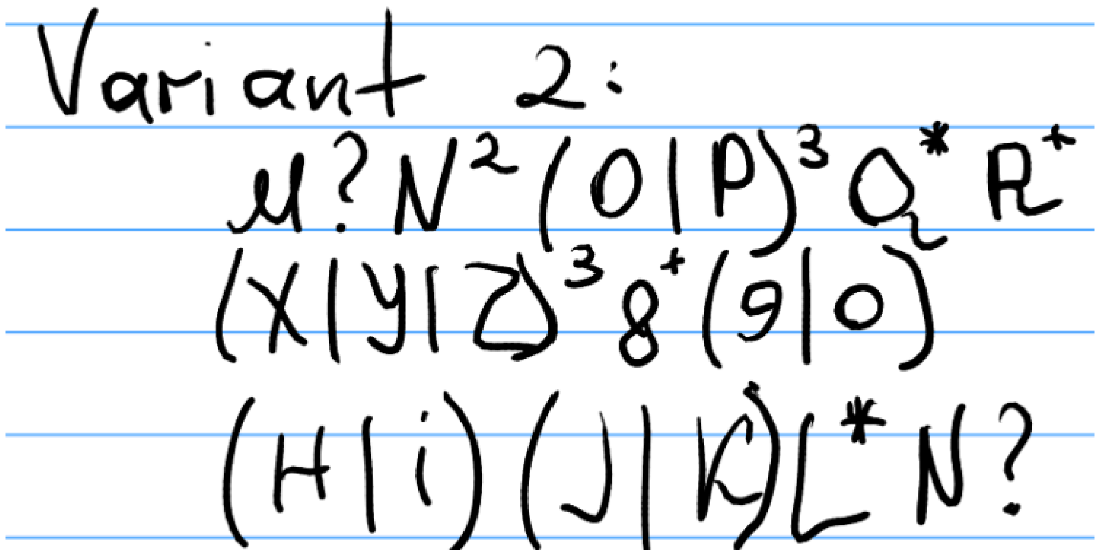
RegexWordGenerator class

RegexWordGenerator within the Lab4 package. The purpose of this class is to generate valid words based on a given regular expression pattern. Here's a breakdown of the key components and functionalities of the RegexWordGenerator class:

1. **Main Method:**
    - The main method is used to demonstrate the functionality of the generateValidWords method by generating valid words from a predefined regular expression pattern.
2. **generateValidWords Method:**
    - This method takes a regular expression pattern as input and returns a list of valid words that match the pattern.
    - Inside the method, a Random object is used to generate random characters and lengths based on certain rules specified by the regular expression.
    - The method iterates through each character in the regular expression pattern and constructs a valid word accordingly.
    - Specific rules are applied for characters such as 'M', 'N', 'P', 'R', 'X', 'Y', '8', '9', '0', 'H', 'i', 'J', 'K', 'L', and optional 'N' and 'L' characters based on random choices.
3. **Usage:**
    - The main method demonstrates how to use the generateValidWords method by providing a regular expression pattern and printing the valid words generated from that pattern.
    - The commented-out portion of the main method shows an alternative way to generate valid words using the Generex library, but the primary focus is on the custom generation logic implemented in the generateValidWords method.
    -

This RegexWordGenerator class is designed to provide a simple yet customizable approach to generate valid words based on regular expression patterns, allowing for flexibility in defining and generating language elements within specified patterns.

Results


Variant 2:

$M?N^2(O|P)^3Q^*R^+$
$(X|Y|Z)^3 8^+(9|0)$
$(H|i)(J|K)L^*N?$

1. **Output first Method:**

   NNPPPRXY809HHKJLNN

2. ***Output second Method:***

   NNOPPRYZZ800HJN
   MNNPPORZZY890HJ
   NNPPPQRRRRRYYY800iJN
   MNNPPPQQRRXXX890iJN
   MNNOOPQQRYXX890iKLLLL

Conclusion

The completion of the lab work involving the RegexWordGenerator class marks a successful exploration into the realm of regular expressions and their practical application in generating valid words based on specified patterns. Through this endeavor, several key observations and conclusions can be drawn:

The lab work deepened my understanding of regular expressions, their syntax, and their role in defining patterns for text matching and manipulation.

Custom Word Generation Logic:

By implementing a custom method to generate valid words based on a given regular expression pattern, I gained insights into the intricacies of constructing language elements within specified constraints.

Flexibility and Adaptability:

The flexibility of regular expressions allowed for the creation of complex patterns encompassing optional characters, repetitions, and choices, showcasing the adaptability of regular expressions to diverse language structures.

Randomized Generation Approach:

The use of randomization techniques within the generateValidWords method demonstrated a dynamic approach to word generation, where random choices and lengths added variability to the generated words while adhering to the pattern rules.

Practical Application of Theory:

This lab work exemplified the practical application of theoretical concepts learned in formal languages and automata theory, bridging the gap between theoretical understanding and hands-on implementation.

Future Exploration and Optimization:

The lab work opens avenues for further exploration, such as optimizing the word generation process, enhancing the randomness factor, and incorporating additional features for pattern manipulation and analysis.

Bibliography

[1] Formal Languages and Compiler Design Accessed February 14, 2024. https://else.fcim.utm.md/pluginfile.php/110457/mod_resource/content/0/Theme_1.pdf.

[2] Regular Language. Finite Automata. Accessed February 15, 2024. https://drive.google.com/file/d/1rBGyzDN5eWMXTNeUxLxmKsf7tyhHt9Jk/view