MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF REPUBLIC OF MOLDOVA
TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS

Laboratory work 4:
Formal Languages and Finite Automata

Elaborated: **st. gr. FAF-223**	**Vieru Mihai**
Verified: **asist. univ.**	**Cretu Dumitru**

Chișinău, 2024

Content

RegexToString

The way I have solved this lab is that I have different methods that break the problems into pieces and solve each one of them.

Parentheses:

```
def handleParanthesis(self):
    self.add_state()
    self.coursor += 1
    new_non_terminal = next(self.iterator)

    while self.string[self.coursor] != ")":
        if self.string[self.coursor] == "|":
            self.grammar += "\n" + self.current_non_terminal + " → "
            self.coursor += 1

        if self.coursor + 1 < len(self.string) and
         ↪  self.string[self.coursor+1].isalpha():
            intermediary_terminal = next(self.iterator)
            self.grammar += self.string[self.coursor] +
             ↪  intermediary_terminal
            self.grammar += "\n" + intermediary_terminal + " → "
        elif self.string[self.coursor-1].isalpha():
            self.grammar += self.string[self.coursor] +
             ↪  self.current_non_terminal
        else:
            self.grammar += self.string[self.coursor] + new_non_terminal
        self.coursor += 1

    self.previous_non_terminal = self.current_non_terminal
    self.current_non_terminal = new_non_terminal
```

Here I check all of the characters untill ")". Inside I handle what happens when I encounter the —, what I do when there multiple characters one after another, and what happens to singular characters.

Star:

```
def handleStar(self):
    self.grammar += "\n" + self.previous_non_terminal + " → " +
     ↪  self.current_non_terminal
    self.grammar += "\n" +  self.current_non_terminal + " → " +
     ↪  self.previous_non_terminal
```

Here is how I handle start, is acording to how I parse normal terminals and that being I create a new state and add the terminal into the non terminal directly, so for example S → aB, thus the way to handle star is that I make a make a direct move S → B, then return back if the user wants to repeat B → S.

Plus:

```
def handlePlus(self):
    self.grammar += "\n" +  self.current_non_terminal + " → " +
     ↪  self.previous_non_terminal
```

Here I just do a loop back to the last state.

QuestionAndPower:

```
def handlePowerAndQuestion(self):
        char_id = 0
        while char_id < len(self.string):
            char = self.string[char_id]
            token = ""
            start_pos = char_id
            if char == "(":
                while char != ")" and char_id < len(self.string):
                    char = self.string[char_id]
                    token += char
                    char_id += 1
            else:
                token += char
                char_id += 1

            if char_id < len(self.string):
                if self.string[char_id] == "^":
```

```
                        char_id += 1
                        number_token = ""
                        while char_id < len(self.string):
                            char = self.string[char_id]
                            if not char.isdigit():
                                break
                            number_token += self.string[char_id]
                            char_id += 1
                        self.string = self.string[:start_pos] +
                         ↪  str(token)*int(number_token) + self.string[char_id:]
                        print(self.string)

                    if self.string[char_id] == "?":
                        self.string = self.string[:start_pos] + f"({token}| )"
                         ↪  + self.string[char_id+1:]
                        print(self.string)
```

Now the way I handle the power and ? symbol is that I break the regex into tokens, then look at what the next value of the token is and based on what the next one is, I do what's needed. (I understood after I finished doing this that this with recursion would've been a better way to do this, but oh well...). For the power symbol I just replace that word with the amount of times its repeated, so for a^3, I would replace with aaa, for $(a \text{ or } b)^4$, I would replace with (a or b)(a or b)(a or b)(a or b).

Then result grammar I take and use the code from the first laboratory to recursivelly create strings.

Conclusion:

This laboratory has provided me with valuable insights into problem-solving strategies and algorithmic approaches. Through the process of tackling various challenges within the scope of parsing grammars, I have honed my ability to break down complex problems into manageable components and devise systematic solutions for each part.

One of the key techniques I employed throughout the laboratory is the method of decomposition. By breaking down the parsing tasks into smaller, more manageable sub-problems, such as handling parentheses, star, plus, question, and power symbols, I was able to effectively address each aspect of the grammar parsing process. For instance, in handling parentheses, I meticulously examined each character until encountering the closing parenthesis, dynamically constructing grammar rules based on the encountered symbols.

Furthermore, I leveraged iterative approaches to handle repetitive tasks, such as the star and plus symbols. For example, when encountering the star symbol, I efficiently managed the repetition by creating direct transitions between non-terminals, allowing for recursive expansion when needed. Similarly, the plus symbol was addressed by establishing a loop back to the previous state, ensuring seamless progression within the grammar structure.

In addressing more complex operations, such as power and question symbols, I utilized tokenization and conditional logic to adaptively process the input string. Despite recognizing the potential for improvement, such as incorporating recursion for a more elegant solution, I successfully implemented a systematic approach to handle these symbols. By parsing the grammar tokens and dynamically adjusting the output based on the subsequent characters, I effectively managed the transformation of the input grammar.

In conclusion, this laboratory experience has not only enhanced my understanding of grammar parsing algorithms but also reinforced my problem-solving skills and adaptability in approaching computational challenges. Through the utilization of systematic methods, iterative processes, and adaptive strategies, I have acquired valuable insights that will undoubtedly contribute to my academic and professional growth in the field of computer science and algorithm design.