MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF THE REPUBLIC OF MOLDOVA

Technical University of Moldova Faculty of Computers, Informatics and Microelectronics Department of Software and Automation Engineering

Postoronca Dumitru FAF-233

Report

Individual Work №1

of LFA

Checked by:
Cojuhari Irina, *university assistant*
DISA, FCIM, UTM

Chișinău – 2024

Conditions of the Task

1. Create the alorithm for convertion of the given number from the keyboard to base 2, 8 and 16 Design a DFA or NFA (with or without e-transitions) that accepts all strings over the alphabet $,c,0,1,2,3,4,5,6,7,8,9,. that correspond to valid currency amounts. A valid string is either a dollar sign followed by a number which has no leading 0's, and may have a decimal point in which case it must be followed by exactly two decimal digits, OR a one or two-digit amount followed by the cent sign c. The single exception to this rule is that strings which begin with "$0." and are followed by exactly two digits are also acceptable. Thus, $432.63, $1, $0.29, 47c, 2c are all accepted, but $021, $4.3, $8.63c, $0.0 are not accepted.

2. Find a grammar for a simple arithmetic expression in a programming language, and show the parse tree for sample expressions:

   Variant 25: **((a-b)+c)**

Visual Solution

I designed an NFA to represent the currency convertion states. I tried to pass the test cases specified in condition through the NFA and all of them passed.

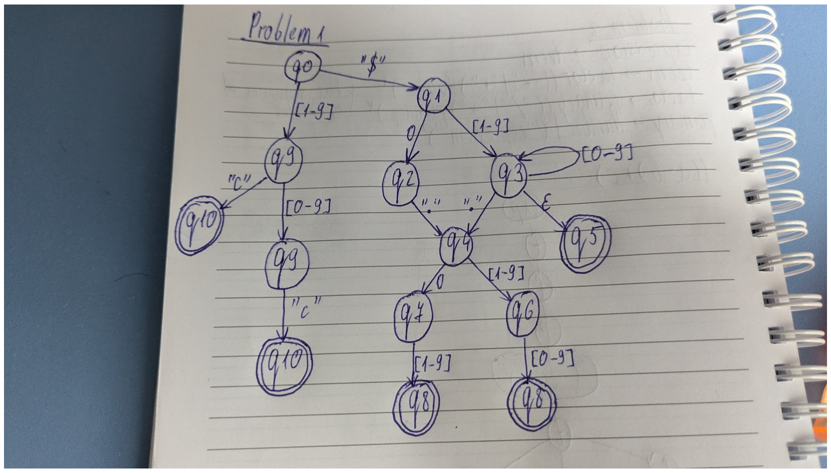

Figure 1: NFA for currency convertion

I specified the rules of the grammar and the parse tree below in Figure 2. You can see some uncommon representations like ()*. This is uses to show that a node can contain as children multiple factors/terms including only one factor/term with operations. As inspiration I use a good book where the author explain how to craft interpreters and take in consideration the edge cases [1]

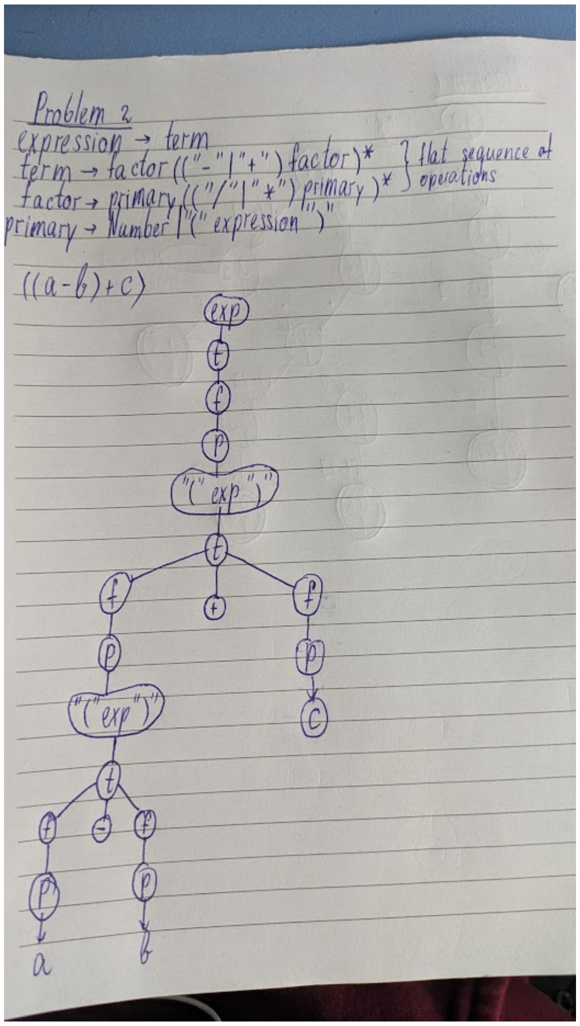
Problem 2
expression → term
term → factor (("-"|"+") factor)*
factor → primary (("/"|"*") primary)*
flat sequence of operations
primary → Number | "(" expression ")"
((a-b)+c)
exp
t
f
p
"(" exp ")"
+
-
a
b
c

Figure 2: Grammar and parse tree for the expression

Also in order to avoid ambiguity in priority of operations I specified *term* rule and *factor* rule

Conclusions

I learned during this individual work how to properly analyse how to build a NFA and to create a good grammar to suit algorithmic expressions. I tried to build a grammar suitable for all variants listed on ELSE and to build the parse tree for my variant.

References

[1] Robert Nystrom (2021) *Crafting-Interpreters https://craftinginterpreters.com/*