AOR Priv.-Doz. Dr. Thomas Noll

Friedrich Gretz, Christina Jansen

Exercise 1 (Time and Space Complexity): (2 Points)

Consider the comparison of the complexities for DFA and NFA method given in lecture three, slide 9.

a) Give an example regular expression, for which the DFA method requires the worst-case space complexity $\mathcal{O}(2^{|\alpha|})$. Argue why your answer is correct.

b) Give an example regular expression, for which the NFA method requires the worst-case time complexity $\mathcal{O}(|\alpha| \cdot |w|)$. Defend your answer shortly.

Exercise 2 (Longest First Match Principle): (4 Points)

a) For extended matching two principles have been introduced to resolve nondeterminism during analysis, the *longest match* principle and the *first match* principle. Argue why these principles are reasonable to use. Instead, we could have insisted on an unambiguous definition of the symbol classes, i.e. for regular expressions $\alpha_1, \ldots, \alpha_n$ it should hold $[\![\alpha_i]\!] \cap [\![\alpha_j]\!] = \emptyset$, for all $1 \leq i < j \leq n$. Why is this not a good idea from a practical point of view? Give examples to support your explanations.

b) Let $\alpha_1, \ldots, \alpha_n$ be regular expressions over Σ and $w \in \Sigma^*$. In the lecture it was assumed that $\varepsilon \notin [\![\alpha_i]\!] \neq \emptyset$ for all $i \in \{1, \ldots, n\}$. Show that these are reasonable assumptions by proving the following proposition:

- If $[\![\alpha_i]\!] = \emptyset$ for some $i \in \{1, \ldots, n\}$ there exists no *flm*-analysis of w w.r.t. $\alpha_1, \ldots, \alpha_n$ that is not a *flm*-analysis of w w.r.t. $\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n$ as well.
- If $\varepsilon \in [\![\alpha_i]\!]$ for some $i \in \{1, \ldots, n\}$ then the *flm*-analysis of w w.r.t. $\alpha_1, \ldots, \alpha_n$ is not unique (if it exists).

Exercise 3 (Backtracking DFA): (4 Points)

Let $\alpha_1 = write$, $\alpha_2 = \Sigma(\Sigma|N)^*$ where $\Sigma := (a|\ldots|z|A|\ldots|Z)$, $N := (0|1|\ldots|9)$.

a) Construct DFAs $\mathfrak{A}_i$ for α_i such that $\mathcal{L}(\mathfrak{A}_i) = [\![\alpha_i]\!]$.

b) Construct DFA $\mathfrak{A}$ such that $\mathcal{L}(\mathfrak{A}) = \mathcal{L}(\mathfrak{A}_1) \cup \mathcal{L}(\mathfrak{A}_2)$.

c) Determine the *first match* partitioning of the set of final states in $\mathfrak{A}$.
(The regular expressions are ordered (α_1, α_2).)

d) Determine the set of reachable and productive states in $\mathfrak{A}$.

e) Compute the run of the corresponding backtracking DFA for input $writeln()$. Provide the run by giving the corresponding configurations.

Exercise 4 (Lexer Implementation): (10 Points)

The goal of this exercise is to build our own lexer which transforms and input string into a list of tokens.

- For each token implement or generate a DFA.
- Build the parallel composition of these DFAs.
  Hint: You do not have to build one "monolithic" automaton, instead whenever the parallel automaton makes one step, you can mimic this by performing a transition in each of the individual automata.

- Partition the final states of the parallel automaton according to the first match principle.
- Implement the backtracking mechanism.

Given an input file with a *WHILE* program, e.g.:

```
/* GCD-Computation of x and y
    w/ WHILE */
int x; int y;
x = read();
y = read();
while ( x != y ) {
    if (y <= x) {
        y = y - x;
    } else {
        x = x - y;
    }
}
// Output result
write("GCD: ");
write(x);
```

your program should generate a list of token like this:

```
[INT, ID, SEMICOLON, INT, ID, SEMICOLON, ID, ASSIGN, READ, LPAR, RPAR, SEMICOLON,
 ID, ASSIGN, READ, LPAR, RPAR, SEMICOLON, WHILE, LPAR, ID, NEQ, ID, RPAR, LBRACE,
IF, LPAR, ID, LEQ, ID, RPAR, LBRACE, ID, ASSIGN, ID, MINUS, ID, SEMICOLON, RBRACE,
ELSE, LBRACE, ID, ASSIGN, ID, MINUS, ID, SEMICOLON, RBRACE, RBRACE, WRITE, LPAR,
STRING, RPAR, SEMICOLON, WRITE, LPAR, ID, RPAR, SEMICOLON]
```