apl. Prof. Dr. Thomas Noll
Christoph Matheja, Matthias Volk

Compiler Construction 2016

— Series 2 —

Hand in until May 10th before the exercise class.

General remarks

- Please hand in your solutions to the programming exercises as a single .ZIP named

  `ex2_MATRNO1_MATRNO2_MATRNO3`.

  This file should include the complete framework provided to you via our webpage.

Exercise 1 (2 Points)

Consider the comparison of the complexities for DFA and NFA method given on the last slide of lecture two.

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

Exercise 2 (2 Points)

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

- 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 (4 Points)

Let $\alpha_1 = \texttt{if}$, $\alpha_2 = \Sigma(\Sigma|N)^*$ 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 the product automaton $\mathfrak{A} = \mathfrak{A}_1 \otimes \mathfrak{A}_2$.

(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 `ifdef`. Provide the run by giving the corresponding configurations.

Exercise 4 (4 Points)

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

Hint: as before we provide a framework which can be downloaded from the course webpage.

Implement `lexer.BacktrackingDFA.run(String)`, the method that, given an input string, performs the steps of the backtracking automaton as discussed in the lecture and returns a list of symbols.

Test your implementation! For example, given the following input

```
1  /* A random walk */
2  int x = 10;
3  int s = 0;
4  while ( x > 0 ) {
5      int b = read() % 2; // randomness by user input
6      if (b == 1) {
7          x = x + 1;
8      } else {
9          x = x - 1;
10     }
11     s++;
12 }
13 write("I stopped walking after: ");
14 write(s);
15 write(" steps");
```

your implementation should generate a list of symbols like this:

```
(INT, int), (ID, x), (ASSIGN, =), (NUMBER, 10), (SEMICOLON, ;),
(INT, int), (ID, s), (ASSIGN, =), (NUMBER, 0), (SEMICOLON, ;),
(WHILE, while), (LPAR, (), (ID, x), (GT, >), (NUMBER, 0), (RPAR, )), (LBRACE, {),
(INT, int), (ID, b), (ASSIGN, =), (READ, read), (LPAR, (), (RPAR, )),
    (MOD, %), (NUMBER, 2), (SEMICOLON, ;),
(IF, if), (LPAR, (), (ID, b), (EQ, ==), (NUMBER, 1), (RPAR, )), (LBRACE, {),
(ID, x), (ASSIGN, =), (ID, x), (PLUS, +), (NUMBER, 1), (SEMICOLON, ;),
(RBRACE, }), (ELSE, else), (LBRACE, {),
(ID, x), (ASSIGN, =), (ID, x), (MINUS, -), (NUMBER, 1), (SEMICOLON, ;),
(RBRACE, }),
(ID, s), (INC, ++), (SEMICOLON, ;),
(RBRACE, }),
(WRITE, write), (LPAR, (), (STRING, "I stopped walking after: "), (RPAR, )),
    (SEMICOLON, ;),
(WRITE, write), (LPAR, (), (ID, s), (RPAR, )), (SEMICOLON, ;),
(WRITE, write), (LPAR, (), (STRING, " steps"), (RPAR, )), (SEMICOLON, ;), (EOF, $)
```