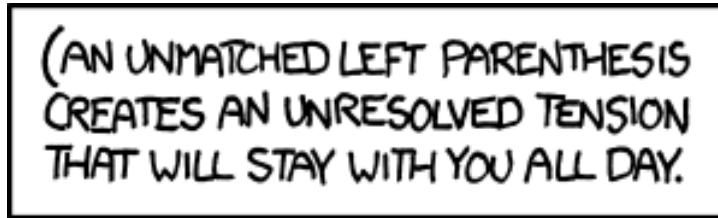


Compiler Construction 2016

— Solution Series 2 —



Credit: <https://xkcd.com/859/>

General Remarks

- Follow the naming convention for the zip file: `ex2_MATRNO1_MATRNO2_MATRNO3` and include the complete framework provided to you via our webpage.
- It is allowed to hand your solutions for the theoretical part via email **as a separately attached PDF file**.
- Please hand in your solutions in groups of 3 or 4.

Exercise 1

(2 Points)

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

- 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.
- 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.

Solution: _____

- For each regular expression there exists a DFA accepting the same language. We call two states of a DFA A , say s, s' , different if there exists a string v such that $\delta(s, v)$ is in a final state and $\delta(s', v)$ is not in a final state or vice-versa.

Consider the regular expression

$$r_n := \Sigma^* a \Sigma^n b$$

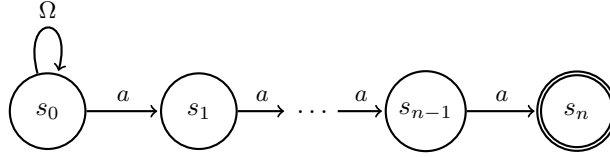
over the alphabet $\Sigma = \{a, b\}$. Let A_n be the corresponding DFA for r_n . Let $w = \alpha a \beta b$ and $w' = \alpha' a \beta' b$ with $|\beta| = |\beta'| = n$. After reading w and w' let the automaton be in the states s, s' , respectively. Let $\beta \neq \beta'$, thus there exist an index i such that $\beta(i) \neq \beta'(i)$ (the i th symbol of β). W.l.o.g assume $\beta(i) = a$ and $\beta'(i) = b$. The string $v = b^{i+1}$ distinguishes s and s' , as $\delta(s, v)$ is in a final state and $\delta(s', v)$ is not (since, $wv \in \mathcal{L}(r_n)$ and $w'v \notin \mathcal{L}(r_n)$).

Thus, for each distinct β there is a distinct state of A_n and there are 2^n different β 's.

- For alphabet Ω with $a \in \Omega$ consider the regular expression

$$\alpha_n := \Omega^* a^n.$$

The corresponding NFA A_n looks as follows:



The number of states of automaton A_n is $O(n)$. For all strings $w = w'a^n w''$, the number of reachable states increases by 1 till it becomes equal to $|Q|$ after reading $w'a^n$. Thus, the NFA method runs in $O(|\alpha_n| \cdot |w|)$.

Exercise 2

(2 Points)

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

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

Solution: _____

- If $(w_1, \dots, w_k)$ is a decomposition of w , then

$$w = w_1 \dots w_k \text{ and } \forall j \in \{1, \dots, k\} \exists i_j \in \{1, \dots, n\} : w_j \in \llbracket \alpha_{i_j} \rrbracket \quad (*)$$

If $\ell \in \{1, \dots, n\}$ such that $\llbracket \alpha_\ell \rrbracket = \emptyset$, then $\forall j \in \{1, \dots, k\} : w_j \notin \llbracket \alpha_\ell \rrbracket$, even if $w_j = \varepsilon$. Thus,

$$(*) \iff w = w_1 \dots w_k \text{ and } \forall j \in \{1, \dots, k\} \exists i_j \in \{1, \dots, \ell-1, \ell+1, \dots, n\} : w_j \in \llbracket \alpha_{i_j} \rrbracket$$

- Let $(w_1, \dots, w_k)$ be a decomposition of w respecting the *flm* principle, i.e. $w = w_1 \dots w_k$, and let α_ε be a regular expression with $\varepsilon \in \llbracket \alpha_\varepsilon \rrbracket$. Then $(w_1, \dots, w_k, w_\varepsilon)$ is a decomposition of w respecting the *flm* principle as well ($w = w_1 \dots w_k \varepsilon$).

Exercise 3

(4 Points)

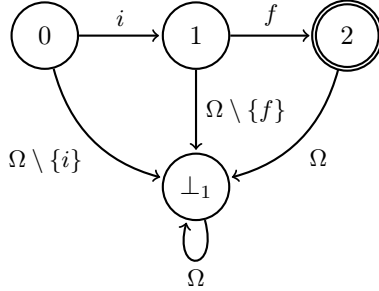
Let $\alpha_1 = \text{if}$, $\alpha_2 = \Sigma(\Sigma|N)^*$ where $\Sigma := (a| \dots |z|A| \dots |Z)$, $N := (0|1| \dots |9)$.

- Construct DFAs $\mathfrak{A}_i$ for α_i such that $\mathcal{L}(\mathfrak{A}_i) = \llbracket \alpha_i \rrbracket$.
- Construct the product automaton $\mathfrak{A} = \mathfrak{A}_1 \otimes \mathfrak{A}_2$.
- Determine the *first match* partitioning of the set of final states in $\mathfrak{A}$.
(The regular expressions are ordered (α_1, α_2) .)
- Determine the set of reachable and productive states in $\mathfrak{A}$.
- Compute the run of the corresponding backtracking DFA for input `ifdef`. Provide the run by giving the corresponding configurations.

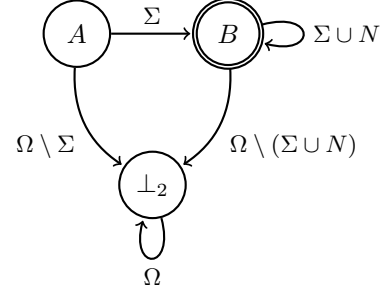
Solution:

- (a) Let $\Sigma \cup N \subseteq \Omega$. The DFA look as follows:

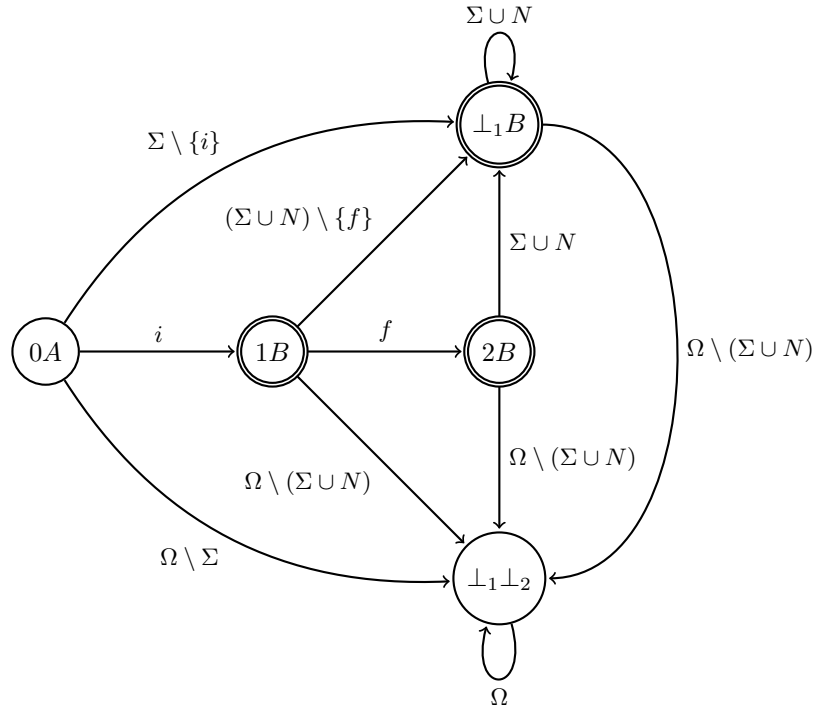
A_1 :



A_2 :



- (b) The product automaton $A = A_1 \otimes A_2$ looks as follows:



- (c) The partitioning of the final states $F = \{1B, 2B, \perp_1B\}$ into the *first match* partitioning is $F_1 = \{2B\}$ and $F_2 = \{1B, \perp_1B\}$.
- (e) All states are reachable and the only non-productive state is $\perp_1\perp_2$.
- (f) The run is:
 $(N, 0A.\text{ifdef}, \varepsilon) \vdash (\text{ID}, 1B.\text{fdef}, \varepsilon) \vdash (\text{IF}, 2B.\text{def}, \varepsilon) \vdash (\text{ID}, \perp_1B.\text{ef}, \varepsilon) \vdash (\text{ID}, \perp_1B.\text{f}, \varepsilon) \vdash (\text{ID}, \perp_1B, \varepsilon) \vdash$
output: ID

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, $)
```

Solution: _____

For the solution see the code in `While2JasminCompilerSolution`.

)