

CSEN1002 Compilers Lab, Spring Term 2024

Task 1: Regular Expressions to Non-Deterministic Finite Automata

Due: Week starting 17.02.2024

## 1 Objective

For this task you need to implement Thompson's construction for converting a regular expression to an equivalent NFA. Description of Thompson's construction can be found in Chapter 3 of the textbook and at [https://en.wikipedia.org/wiki/Thompson's\\_construction](https://en.wikipedia.org/wiki/Thompson's_construction).

## 2 Requirements

- We make the following assumptions for simplicity.
  - a) The alphabet  $\Sigma$  of the regular expression is always a subset of the Latin alphabet, not including **e**.
  - b) Regular expressions do not include  $\emptyset$ .
  - c) The empty string  $\varepsilon$  is represented by **e**.
  - d)  $\circ$  is represented by **.** and  $\cup$  by **|**.
  - e) Regular expressions are represented in *postfix* notation.
  - f) States of the resulting NFA are numbers.
  - g) For a postfix regular expression  $R$ , states *introduced* by NFA equivalent to a prefix of  $R$  are smaller (as numbers) than states *introduced* by NFA equivalent to longer prefixes of  $R$ . For operators (such as union and  $*$ ) which introduce a start and an accept state, the start state is smaller (as a number) than the accept state.
  - h) Following Thompson's construction, concatenation involves merging the accept state of the first (left) NFA and the start state of the second (right) NFA; the resulting merged state is the accept state of the first NFA.
- You should implement a class constructor **RegExToNfa** and a method **toString**.
- **RegExToNfa** takes one parameter which is a string of the form  $A\#R$ .  $A$  is a string representation of an alphabet  $\Sigma$ , a semicolon-separated sequence of alphabetically sorted symbols, and  $R$  is a postfix regular expression over  $\Sigma$ . **RegExToNfa** constructs the NFA to  $R$  as per Thompson's construction.
- **toString** returns a string describing the NFA resulting from Thompson's construction. A string describing the NFA resulting from Thompson's construction is of the form  $Q\#A\#T\#I\#F$ .
  - $Q$  is a string representation of the set of states; a semicolon-separated sequence of sorted integer literals.

- $A$  is a string representation of the input alphabet; a semicolon-separated sequence of alphabetically sorted symbols
- $T$  is a string representation of the transition function.  $T$  is a semicolon-separated sequence of triples. Each triple is a string representing a single transition; a comma-separated sequence  $i, a, j$  where  $i$  is a state of  $Q$ ,  $a$  a symbol of  $A$  or  $\epsilon$ , and  $j$  a state of  $Q$  representing a transition from  $i$  to  $j$  on input  $a$ . These triples are sorted by the source state  $i$ , then (if the same state has more than one outgoing transition) by the input  $a$ , and then (if multiple triples share the same source state and input, due to non-determinism) by the destination state  $j$ .
- $I$  is an integer literal representing the initial state.
- $F$  is a string representation of the set of accept states; a semicolon-separated sequence of sorted integer literals.
- For example, `toString`, being invoked on a `RegexToNfa` object representing the regular expression `a;b#ab|`, should return the string

`0;1;2;3;4;5#a;b#0,a,1;1,e,5;2,b,3;3,e,5;4,e,0;4,e,2#4#5`

- Important Details:

- Your implementation should be done within the template file “`RegexToNfa.java`” (uploaded to the CMS).
- You are not allowed to change package, file, constructor, or method names/signatures.
- You are allowed to implement as many helper classes/methods within the same file (if needed).
- Public test cases have been provided on the CMS for you to test your implementation.
- Please ensure that the public test cases run correctly without modification before coming to the lab to maintain a smooth evaluation process.
- Private test cases will be uploaded before your session and will have the same structure as the public test cases.

### 3 Evaluation

- Your implementation will be tested by ten input regular expressions.
- You get one point for each correct output of `toString`; hence, a maximum of ten points.
- The evaluation will take place during your lab sessions of the week starting Saturday, February 17.

### 4 Online Submission

- You should submit your code at the following link.

<https://forms.gle/5a8bZTG6ThNfw24z8>

- Submit one Java file (`RegexToNfa.java`) containing executable code.
- **Online submission is due by the end of your lab session.**