CS F363 Compiler Construction

Week 2: Regular Expressions

In this lab session, your task is to write a C / C++ program to answer, "Can the given word w be generated by a given regular expression r?." The following procedure can answer this.

1. Convert the given regular expression r to an equivalent deterministic finite automaton (RE -> NFA with ϵ-transitions -> NFA without ϵ-transitions -> DFA).
2. Then run the obtained DFA with the given word w and check if the DFA accepts it.
3. If DFA accepts the word w , then output YES; otherwise, output NO.

Formal definition of Regular expression

- The class of regular expressions over Σ is defined recursively as follows:
  - The letter ϵ is a regular expression over Σ.
  - Every symbol in Σ is a regular expression over Σ.
  - If r_1 and r_2 are regular expressions over Σ, then so are $(r_1|r_2), (r_1r_2), (r_1)^*$ are regular expressions where
    - '|' indicates alternative or parallel paths (read it as r_1 or r_2).
    - '*' indicates closure (r^* means zero or more occurrences of r)
    - $(r_1r_2)$is the concatenation of regular expressions $r_1$and r_2.
  - The regular expressions are only those obtained using rules (1) and (2).

Assume that $\Sigma = \{a, b\}$.

In your C / C++ program, first generate an equivalent DFA for the given regular expression, then run the DFA for the given word to check if it can be generated from the regular expression.

Initially, you can consider NFAs for regular expressions $r = \epsilon,\ r = a$ and $r = b$ (base case):

NFA for $r = \epsilon$:

	ϵ
q_0	$\{q_0\}$

Where q_0 is the start and the final state.

NFA for $r = a$:

	a	b
q_1	$\{q_2\}$	$\emptyset$
q_2	$\emptyset$	$\emptyset$

Where q_1 is the start state and q_2 is the final state.

NFA for $r = b$:

	a	b
q_3	$\emptyset$	$\{q_4\}$
q_4	$\emptyset$	$\emptyset$

Where q_3 is the start state and q_4 is the final state.

Then incrementally construct NFAs (possibility with ϵ-transitions) for the larger expressions.

Once you obtained an NFA with ϵ-transtions, you convert it to an NFA with out ϵ-transtions (refer Theory of computation course for further details). Then convert the NFA to DFA (this was already done in the Theory of computation course).

Once you obtained an equivalent DFA M for the given regular expression r, run the given word w on M and output YES if M accepts w, otherwise output NO.

Examples:

1. Input: $r = \left(\left((a)\left(((a)|(b))^*\right)\right)\ \ \left((b)(a)\right)\right)$ and $w = abbaba$
   Output: YES

2. Input: $r = \left(\left((a)\left((a)(b)\right)^*\right)\left((b)(a)\right)\right)$ and $w = abbaba$
   Output: NO

3. Input: $r = \left(\left((a)\left(\left((a)(b)\right)\right)^*\right)(b)\right)$ and $w = aababb$
   Output: NO

Input format: Read the input from a .txt file, with the regular expression in the first line and word in the second line.

Output format: Print the answer (YES or NO) on the terminal along with the DFA in the tabular format (given in the above).