Assignment 12, Authomata Theory

Oleg Sivokon

<2015-09-07 Mon>

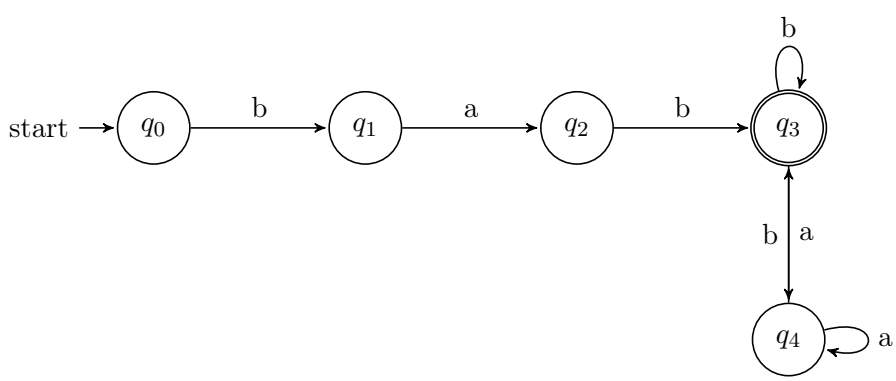
Contents

1 Problems

1.1 Problem 1

1. Build an NFA for the language over alphabet $\{a, b\}$ where words must start with bab and end in b.
2. Build an NFA for the language over alphabet $\{a, b, c\}$, defined as follows: $L = \{w \mid \exists n, m, k \in \mathbb{N}.(w = a^n b^m c^k \wedge \#(w) \mod 2 = 0)\}$.
3. Build an NFA for the language over alphabet $\{a, b\}$, s.t. it contains all words with substring aba repeated at least twice.

1.1.1 Answer 1



The nodes where automaton dies are not shown.

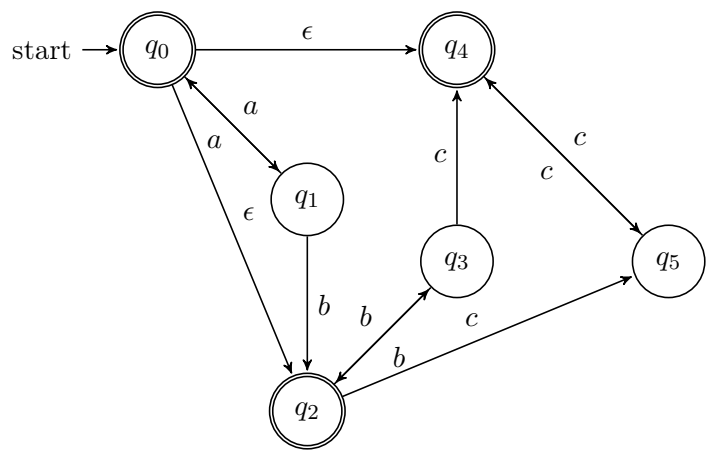
The language accepted by this automaton must start with the prefix bab as can be seen in diagram above. The only accepting state has transitions pointing at it only on inputs b, thus all words accepted by this language must end in b.

Conversely, if the words start with the prefix bab, then we must reach the statate $\hat{\delta}(bab, q_0) = q_4$. From q_4 the input can be either accepted, since it already ends in any number of b, or it may follow through to q_4 and

whenever b is encountered in the input—return to q_3. Since all execution path will thus lead to the accepting state on b or to rejecting state on a, I conclude that all words with prefix bab and ending in b are accepted by the described automaton.

1.1.2 Answer 2

A regular expression to summarize the effort: $(((aa)^*((bb)^*(cc)^*)+(b(bb)^*c(cc)^*)))+(a(aa)^*((b(bb)^*(cc)^*)+(b(bb)^*(cc)^*)))$.

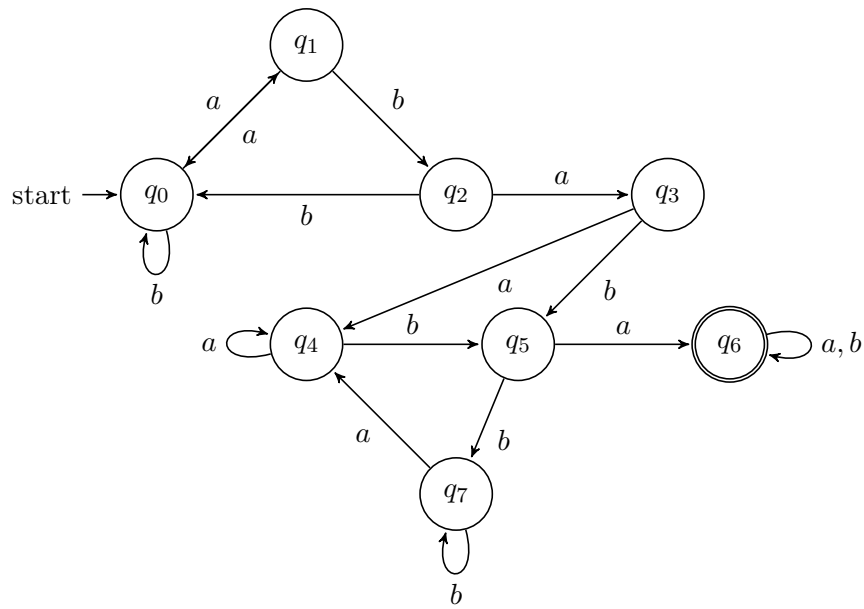


Essentially, what happens in this diagram is as follows:

1. On first input we decide whether the prefix starts with a, b or c.
2. Once decided, we parse more of the prefix.
   - If the prefix was c, we make sure there is even number of c.
   - If the prefix started with b, we create two branches, one will pick the non-accepting state of the bit of automaton processing c prefix once we counted even number of b inputs, and an accepting state otherwise.
   - If our prefix was a, then we will proceed similarly to the previous case, however, we will switch roles: on odd number of $a's$, we will switch to an accepting state and to a non-accepting state otherwise.

1.1.3 Answer 3

The diagram below can be also given by the regular expression: $((ab)^*b^*)^*aba(ba+(((ab)^*b^*)aba))(a+b)^*$. An easier, but a sloppier way to write the same reg-exp would be: $(a+b)^*(ababa+(aba(a+b)^*aba))(a+b)^*$.



Put in words: skip over repetitions of bb possibly preceded by a, until encountering aba substring. Once that happens, consider the prefix of the second aba substring found. If the next input is b, continue matching, else—bail out and essentially repeat the previously described procedure.

1.2 Problem 2

Prove or disprove that pairs of regular expressions to follow accept the same language.

1. $(0(10^*)^*)^* + 1^*$ and $(1+0)^*$.
2. $(1+0)^+$ and $(0^*1)^*(1^*0)^+ + (0^*1)^+$.

3. $1^*(0^*10^*)^*$ and $(101^*)^*1^*$.

1.2.1 Answer 3

Two expressions are not equivalent. $(1+0)^*$ matches any binary string, while $(0(10^*)^*)^* + 1^*$ doesn't match any binary string containing with a prefix 00 or more consequtive zeros.

1.2.2 Answer 4

Two expressions are equivalent. $(0^*1)^+$ will match any binary at least one character long string edning in 1, while $(1^*0)^+$ will match any binary string at least one character long ending in 0. The union of these two expressions will match all binary strings of length at least 1, which is equivalent to $(1+0)^+$. The $(0^*1)^*$ of the second expression plays no role (is redundant).

1.2.3 Answer 5

Teo expressions are not equivalent. $(101^*)^*1^*$ will not match string containing 00 or more consequent zeros as a substring, while this is not a problem for $1^*(0^*10^*)^*$.

1.3 Problem 3

Write a regular expression for the language over alphabet $\{a, b\}$ s.t. all words in this language start with either aa or bbb and none of them contains substring bab.

1.3.1 Answer 6

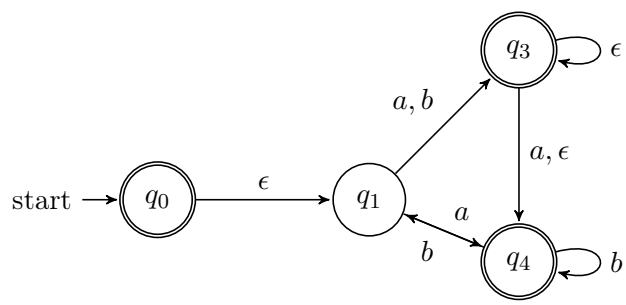
The desired regex is $(bbb^+)^*(aa^+b^*)^*$.

1.4 Problem 4

Write an algorithm which accepts a regular expression r and produces a language $\overline{L[r]}$.

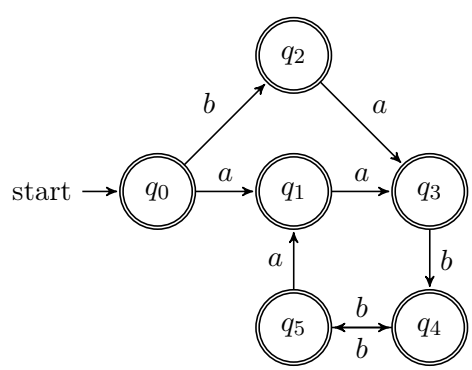
1.5 Problem 5

Build a DFA from given NFA:



1.5.1 Answer 7

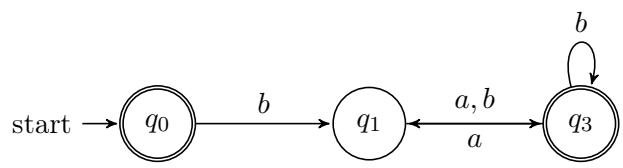
The corresponding DFA can be written as:



Nodes where automata dies are not shown.

1.6 Problem 6

Write a regular expression for the diagram below:



Nodes where the automata dies are not shown.

1.6.1 Answer 8

The regular expression for the diagram above: $\epsilon + b((a+b)b^*)^+$.

1.7 Problem 7

Given regular expression r and L, a language over Σ which designates regular expression $r\Sigma^*$. Prove that unless $L = \Sigma^*$ and $L = \emptyset$, there doesn't exist a regular expression s s.t. $s\Sigma^*$ designates $\overline{L}$.