Assignment 11, Authomata Theory

Oleg Sivokon

<2015-09-04 Fri>

Contents

1 Problems

1.1 Problem 1

Given following languages over the alphabet $\{a, b\}$

- $L_1 = \emptyset$.
- $L_2 = \{\epsilon, aa\}$.
- $L_3 = \{\epsilon, a, aa, ab, abb\}$.
- $L_4 = \{aabb, aabbb, aa, aaa\}$.
- $L_5 = \{\epsilon, b, bbb, abab, abba, aabb\}$.
- $L_6 = \{\epsilon, bbbaa, baba, aaab, aabba, aa\}$.

1. What are the following languages:
   - $L_4 L_4$.
   - $(L_1 \cup L_2 \cup L_3)^R$.
   - $L_3 L_1 L_6$.
2. Define exponentiation as follows: $L^K = \{x \in L \mid \exists y \in K.(|y| = |x|)\}$. What are the languages $L_4^{L_5}$ and $L_6^{L_1}$?

1.1.1 Answer 1

1. Concatenation of L_4 with itself gives: $L_4 L_4 = \{aabbaabb, aabbaabbb, aabbaa, aabbaaa, aabbbaabb, aabbbaa, aabbbaaa, aaaabbb, aaaa, aaaaa, aaaaabb, aaaaabbb\}$
2. $(L_1 \cup L_2 \cup L_3)^R = \{\epsilon, a, aa, ba, bba\}$.
3. $L_3 L_1 L_6 = \emptyset$. This is so because there are no words in language L_1 to concatenate with.

```
:- use_module(library(lists)).

concatentated_member(L1, L2, L3) :-
    member(M1, L1), member(M2, L2),
    string_concat(M1, M2, L3).

concatentated(L1, L2, L3) :-
    findall(X, concatentated_member(L1, L2, X), X),
    list_to_set(X, L3).

assignment_11a :-
    X = ["aabb", "aabbb", "aa", "aaa"],
    concatentated(X, X, Y),
    [First | Rest] = Y,
    write("$$\\{"),
    write(First),
    maplist(format(',\\allowbreak ~s'), Rest),
    write("\\}$$").
```

$\{aabbaabb, aabbaabbb, aabbaa, aabbaaa, aabbbaabb, aabbbaabbb, aabbbaa, aabbbaaa, aaaabb, aaaabbb, aaaa, aaaaa, aaaaabb, aaaaabbb, aaaaaa\}$

1.1.2 Answer 2

1. $L_4^{L_5} = \{aaa, aabb\}$.
2. $L_6^{L_1} = \emptyset$.

1.2 Problem 2

Let L_1, L_2 and L_3 be languages over some alphabet Σ. Prove or disprove:

1. $(L_1 \cup L_2)L_3 = L_1L_3 \cup L_2L_3$.
2. $(L_1 \cap L_2)L_3 = L_1L_3 \cap L_2L_3$.

1.2.1 Answer 3

First, I will prove $(L_1 \cup L_2)L_3 \subset L_1L_3 \cup L_2L_3$. Assume to the contrary that there is $w \in (L_1 \cup L_2)L_3$ which is not in $L_1L_3 \cup L_2L_3$. Put $w = xy$ where $x \in (L_1 \cup L_2)$ and $y \in L_3$ (this implies $L_3 \neq \emptyset$ and at least one of $(L_1 \cup L_2) \neq \emptyset$. Suppose x comes from L_1, then it has to be in $L_1L_3 \cup L_2L_3$ because it is in L$_1$ L$_3$\$, similartly if it originates in L_2. Suppose now $L_3 = \emptyset$, then there is an empty set on both sides of equation (by definition of concatenation). Suppose both L_1 and L_2 are empty, then, again, we have emtpy set on both sides of the equation. Thus we showed that it is impossible for w not to be in the $L_1L_3 \cup L_2L_3$, hence the original argument must be true.

Similarly, to prove $L_1L_3 \cup L_2L_3 \subset (L_1 \cup L_2)L_3$, assume there exists $w \in L_1L_3 \cup L_2L_3$, not a amember of $(L_1 \cup L_2)L_3$. Again, $w = xy$ where $y \in L_3$ and x may be a member of L_1, L_2 or both. Suppose, again, the sets aren't empty. If w came from L_1L_3, then x came from L_1, but it is a member of $(L_1 \cup L_2)$ and similarly if it came from L_2. Since $y \in L_3$ and L_3 is present on both sides, it is not possible for w to not be a member of $(L_1 \cup L_2)L_3$. As in previous case, whenever L_3 or $(L_1 \cup L_2)$ are empty, both sides of equation contain an empty set. Hence we proved both directions, hence the conjecture is true.

1.2.2 Anser 4

This conjecture isn't generally true. Suppose $L_1 = L_3 = \{a\}$ and $L_3 = \{\epsilon, aa\}$. Then:

1. $(L_1 \cap L_2)L_3 = \emptyset$.
2. $L_1L_3 \cap L_2L_3 = \{aa\}$.

I.e. both sides of equation are not equal. This completes the proof.

1.3 Problem 3

An equivalence relation over Σ^* will be called invariant from right if $\forall z \in \Sigma^*.(xRy \implies xzRyz)$. Answer for every relation in $\{a, b\}^*$ whether the relation is an equivalence relation and whether it is invariant from right.

1. $xRy \iff |x| \geq |y|$.
2. $xRy \iff (|x| = |y| = 0 \vee x = qz, y = pz, |z| \geq 1)$.

1.3.1 Answer 5

Total order relation is not symmetric. Suppose $x = a$ and $y = ab$, then $x \geq y$ but not $y \geq x$. Since this relation is not an equivalence, it cannot be right invariant either.

1.3.2 Answer 6

This relation is an equivalence. It is transitive because whenever $x = qz$, $y = pz$ and $w = vz$, all of the below hold: xRy, yRw, xRw since they all have the last letter in common. This also holds trivially in case the length is zero, since $x = y = w = \epsilon$ in that case.

The relation is reflexive because whenever every string is either empty or its last symbol is equal to itself, i.e. xRx is always true.

The relation is symmetric because whenever $x = qz$ and $y = pz$ then both xRy and yRx hold (again, becuase x and y have the final letter in common, or are both the empty string).

The relation is also invariant from the right. The proof will proceed by induction on the string's length.

Base step: $\epsilon R \epsilon \implies \epsilon z R \epsilon z$ because R is reflexive and $z = \epsilon z$.

Inductive step: suppose the inductive hypotesis $xRy \implies xzRyz$, then suppose we concatenate the same character c to both x and y. This character must be the same by definition of R. Then $xcRyc \implies xczRycz$ because we can simply rename $xc = x_1$ adn $yc = y_1$ and obtain the inductive hypothesis restated using new terms: $x_1Ry_1 \implies x_1zRy_1z$. This completes the inductive step, and hence the proof is completed.

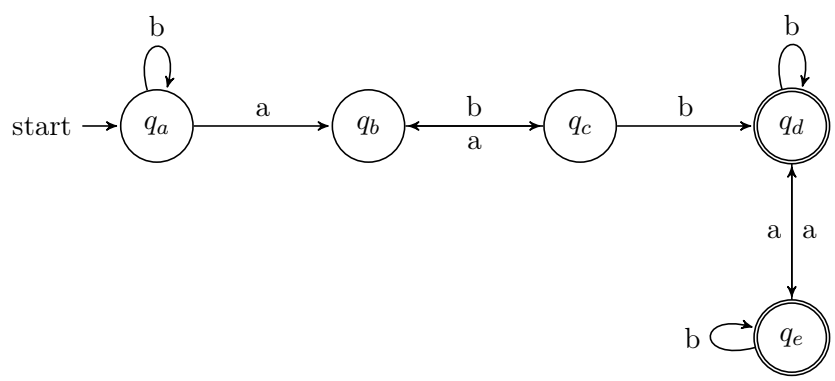
1.4 Problem 4

Build an DFA accepting languages over alphabet $\{a, b\}$ s.t.

1. The language contains all words with a substring abb, but never with the substring aa.
2. If the word contains substring bb must, it must be preceded by aba.

1.4.1 Answer 7

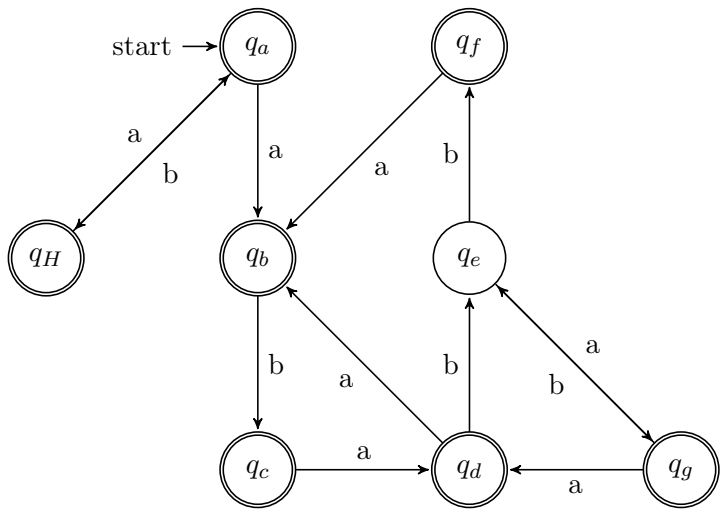
Below is the DFA for the first question:



Nodes where the automata dies are not shown.

1.4.2 Answer 8

Below is the DFA for the second question:



Nodes where the automata dies are not shown.

1.5 Problem 5

Given alphabet of binary strings and netation operation defined as follows:

$$\mathrm{Neg}(w) = \begin{cases} \epsilon & \text{if } w = \epsilon \\ 1 & \text{if } w = 0 \\ 0 & \text{if } w = 1 \\ 0.\,\mathrm{Neg}(y) & \text{if } w = ay \wedge a = 1 \\ 1.\,\mathrm{Neg}(y) & \text{if } w = ay \wedge a = 0\ . \end{cases}$$

And similarly for languages: $\mathrm{Neg}(L) = \{w \mid \mathrm{Neg}(w) \in L\}$.

1. Does there exist a language $\overline{L} = \mathrm{Neg}(L)$?
2. Does there exist a language $\overline{L} \setminus \{\epsilon\} = \mathrm{Neg}(L)$?

1.5.1 Answer 9

No, there cannot be such language. Suppose there was such a language, then ϵ must be either in it or in its complement. Suppose ϵ is part of L, then it must be also in its negation since $\epsilon = \text{Neg}(\epsilon)$. By symmetric reasoning ϵ cannot be in $\overline{L}$.

1.5.2 Answer 10

Yes, there exists such a language, for instance the language representing all odd numbers in base 2. Its complement would be a language representing all even numbers in base 2 and because words in L always end in 1 (with no other requirements) negating them will necessarily produce a word in $\overline{L}$.

No number is both odd and even, therefore negation produces exactly the complement of L. Since empty string is excluded, the only word which would be also a negation of itself is excluded as well.

1.6 Problem 6

1. Given DFAs $A = (\Sigma, Q, q_0, F, \delta)$ and $B = (\Sigma, Q, q_0, F \cap (Q \setminus q_0), \delta)$, is it also the case that $L(B) = L(A) \cap \Sigma^+$?
2. Given DFA $A = (\Sigma, Q, q_0, F, \delta)$ construct DFA B from A s.t. it accepts all words of the language of A which have length distinct from 1.

1.6.1 Answer 11

The conjecture is false. Consider the language of unary strings of even length described by the following transitions table:

δ	a
$*q_0$	q_1
q_1	q_0

In other words, $L(A) = \{\epsilon, aa, aaaa, aaaaaa, \dots\}$. However $F \setminus \{q_0\} = \emptyset$, i.e. no words are accepted by $L(B)$, while $L(A) \cap \Sigma^+ = \{aa, aaaa, aaaaaa, \dots\}$.

1.6.2 Answer 12

A DFA can accept words of length 1 in only these two scenarios:

1. It loops on some input in the first accepting state.
2. The state directly linked from the first state is accepting.

Below is the procedure to remove these accepting states while accepting all other inputs.

1. If the first state loops on some inputs add a new state q_1, remove all transitions from q_0 to itself and add transitions for the same inputs I_0 from q_0 to q_1. Add transitions on all inputs from q_1 back to itself.

   For any input which on which q_0 used to loop B now necessary makes at least two transitions before accepting it, possibly accepted string with prefixes that did not cause the automate to loop on q_0 are still accepted (nothing has changed).

2. For each accepting state q_n directly linked from q_0 we do the following:
   - change q_n from accepting to non-accepting.
   - add a new accepting state q_2.
   - remove transitions from states Q_i directly to q_n, for each transition removed add a new one for the same input from Q_i to q_2.

Algorithm 1 Ensure the language of this DFA has no words of length 1

```
procedure ensure-word-length-ne-one(Σ, Q, q_0, F, δ)
    loops ← {i | ∃i ∈ Σ.(δ(q_0, i) = q_0)}
    new ← make new accepting state
    backarcs ← {(v, w, i) | ∃v, w ∈ Q.(δ(q_0, j) = v = δ(w, i) ∧ i ≠ q_0)}
    for input ∈ loops do
                                        ▷ Remove all loops from the first state
        add-transition(q_0, new, input)
        remove-transition(q_0, q_0, input)
        add-transition(new, new, input)
    end for
    for input ∈ Σ do
                                ▷ Bounce back from the new state to the first state
        add-transition(new, q_0, input)
    end for
    for (source, destination, input) ∈ backarcs do
                                        ▷ For each state directly reachable
                                          from the first state reattach
                                          all inbound transitions
                                          to the new state
        new ← make new accepting state
        remove-transition(source, destination, input)
        add-transition(source, new, input)
        outbound ← {(i, w) | ∃w ∈ Q.(δ(w, i) = destination)}
        for (input, destination) ∈ outbound do
                                        ▷ Bounce back from the new state
                                          to the states immediately
                                          reachable from source state
            add-transition(new, destination, input)
        end for
    end for
end procedure
```

- for each outbound transition from q_n add new transition on the same input from q_2.

Another way to go about this is to simply construct an automaton accepting all words of $\Sigma * \setminus \{w \mid \#(w) = 1\}$ (example given below), but it feels like cheating.