Assignment 3
Regular Languages

Josefin Ulfenborg
940806-5960
yunalescca@gmail.com

2017-05-02

1 Task 1

At first I tried to prove that the regular expressions were equal, in the sense that they generate the same language, by using the algebraic laws. But after some time of trying, and consulting during the individual help, I concluded that instead of using algebra, I will try to explain this with words and examples.

So the expressions I have are:

$$(aa^\star b^\star b)^\star = \varepsilon + a(a+b)^\star b$$

The left hand side (LHS) states that I can read this expression zero or more times, and the right hand side (RHS) states that I can either read nothing, or read the alternative expression once. I will continue in list form, with each new item representing supporting argument for why these two are equal.

- For the most basic case: on the LHS we can read this expression zero or more times, so the most simplest case is reading it zero times. That is, reading nothing. This corresponds to the RHS, as we can choose to read nothing.

- If we read the expression on the LHS once, we have $(aa^\star b^\star b)$. So from here on the most simplest case is just reading ab (i.e, not looping on a's or b's). This a word we can also generate on the RHS, by reading a, not looping on (a+b), and then finally reading b.

- For the LHS, say we choose to loop on a and b once respectively, and still only read the whole thing once, this means we will produce the word aabb.

  For the RHS, if we do not choose nothing, we start by reading a. Then we can choose to *either* read a or b, and this option is available for as many times as we want. This means we could first read one a, and then read one b, and finally finishing with the b. This will produce the word aabb, same as above.

- Another word that can be produced on the LHS is aababbab, that is, we loop the whole expression three times: first we read aab, then abb, and finally ab. These we put together in order to get the word aababbab.

  Now I will examine if this word is possible to produce on the RHS. So we already have the starting a and the finishing b, which means we have to produce the sub word ababba. This can easily be done by looping the $(a+b)^\star$ six times. For every time we loop this, we can *choose* whether or not we want to read the a or the b. So on the first loop, we read the a, on the second loop we read the b and so on, until we have produced ababba.

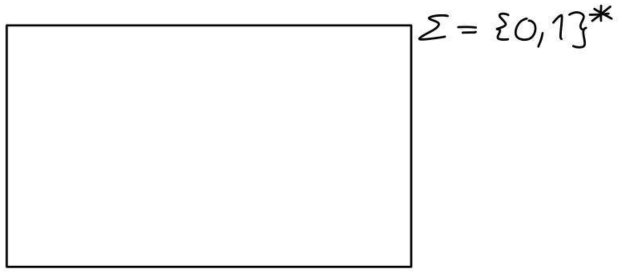
- What the LHS is saying, really, is that if we choose to read the expression at least once, it has to start with an a and end with b. Between these two symbols we can read as many a's and b's as we want, and in any order as well, because we have the star outside of the parenthesis.

What the RHS is saying is, that if we choose to not read nothing, we have to read a word that will always start with an a and end with b. Between these two letters, is that we can read a *or* b as many times as we want. This is essentially the same description as I provided for the LHS expression, in the sense that these two descriptions will always produce the same language.

To wrap up, these two expressions are equivalent with support of the above arguments and examples.

2 Task 2

a) So for the first example, $\mathcal{L}_1$ is regular, and $\mathcal{L}_2$ is not, and the question asks whether their union can be a regular language, that is $\mathcal{L}_1 \cup \mathcal{L}_2$. If I have the following universe:



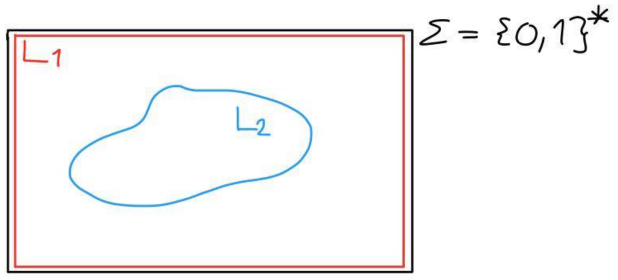
then my non-empty sets $\mathcal{L}_1$ and $\mathcal{L}_2$ will be sets in this universe. Examples of some regular languages are

- $\varnothing$, the empty language (this has to be ruled out, however, since it states that both languages are non-empty)
- $\{\varepsilon\}$
- $\Sigma^\star$, that is, the universe itself.

So for instance, since $\mathcal{L}_1$ is a regular language, it can be any regular language, so it can be $\Sigma^\star$. And since $\mathcal{L}_2$ is any non-regular language, all I know is that it's a subset of my universe, and so a subset of $\mathcal{L}_1$. So if I union these two languages, then it's clear that I will get the whole universe as my language, that is:

$$\mathcal{L}_1 \cup \mathcal{L}_2 = \mathcal{L}_1$$

Hence, the union of a regular language and a non-regular language can be a regular language. I have made an illustration of this which can be seen here (I have drawn $\mathcal{L}_1$ slightly inside the frames of $\Sigma^\star$ for clarification):



b) So for the b-part of this task both $\mathcal{L}_1$ and $\mathcal{L}_2$ are non-regular languages. It also specifically says that their intersection must be non-empty.

At first I thought of defining the two languages like so:

$$\mathcal{L}_1 = \left\{0^n 1^n | n \geq 0\right\}$$

$$\mathcal{L}_2 = \left\{0^n 1^m | n \leq m\right\}$$

but when I did the intersection of these two I realized that

$$\mathcal{L}_1 \cap \mathcal{L}_2 = \mathcal{L}_1$$

which is non-regular language. Hence, I chose discarded that example for now, and see if I could find another example.

So I tried to re-define $\mathcal{L}_2$ like this:

$$\mathcal{L}_2 = \left\{1^n 0^n | n \geq 0\right\}$$

The words in these languages are

$$\mathcal{L}_1 : \left\{\varepsilon, 01, 0011, ...\right\}$$

$$\mathcal{L}_2 : \left\{\varepsilon, 10, 1100, ...\right\}$$

This means that the intersection will result in

$$\mathcal{L}_1 \cap \mathcal{L}_2 = \left\{\varepsilon\right\}$$

which is a regular language. This is not empty, like the language $\varnothing$, but it is the language of the empty word.

To conclude, for both a and b, the resulting languages **can** be regular.

3 Task 3

For the third exercise I am to prove that the given language is not regular. I will do this by the Pumping Lemma, and I will use the guidelines given in the slides and the exercise class (also in the book), where you look at it as a "game" with player moves and opponent moves. By using the Pumping Lemma I must first assume that the language I choose *is* regular, because the Pumping Lemma only holds for regular languages. The game rules are as follows:

1. Player 1 (our move) chooses the language
2. Player 2 (the opponent) picks an $n \geq 0$. We only know that there exists an n such that it is greater than or equal to 0, but we do not know which n this is.
3. Player 1 (our move) now gets to pick w such that $w \in \mathcal{L}$. Now we must show the length of w such that $|w| \geq n$
4. Player 2 now gets to pick x,y,z such that $x, y, z = w \wedge y \notin \varepsilon \wedge |xy| \leq n$. Again we do not now what values x,y,z have, we only know *there exists* some value for them.
5. Player 1 (our move) now gets to choose a k such that $xy^kz \notin \mathcal{L}$
6. At last we get to conclude that the language is not regular.

Now, the language we are given states that the number of zeros should be less than twice the number of ones. That is, say we have five 1's, then the number of 0's should be nine or less. So for the very step, I assume that this language is in fact regular.

1. The language I choose is the language given in the assignment, that is $\mathcal{L} = \{w \in \{0, 1\}^\star \mid \#0(w) < 2 * \#1(w)\}$. This is all words with 0's and 1's in no particular order, where the number of 0's should be less than the twice amount of 1's. That is, 01, and 011 are words in the language, but 001 and 000011 are not.
2. Let $n \geq 0$
3. Choose w. Now, at first I thought of letting $w = 0^{2n-1}1^n$. But then I realized that I could make it even simpler for myself. My only task here is to pick a w, but this could be any w as long as it's a part of the language (and as long as it helps me when we split the words up in the three parts). This means that the word $w = 0^n1^n$ works just as fine.

   Now I can also show that the length is $\geq n$, like so: $n + n = 2n \geq n$
4. For the next part I assume there exists some x,y,z $\in \Sigma^\star$, such that xyz = w, y $\notin \varepsilon$ and $|xy| \leq n$. An important note is that, although I do not know exactly what x,y,z is, I have chosen w in such a way that, since we know $|xy| \leq n$ and that xy comes at the front of my w, then xy has to consist of only 0's (in particular only the n first 0's).

5. Now I get to choose k. Since y is nonempty, y consists of at least one zero, however, we can choose k = 0 meaning that we don't go into the loop explained during the lectures (so we don't pump any more 0's). Anyhow, if k = 1, then we have the original word xyz, which is the w I picked. What I want to try and prove now is that there can in fact be, say, two 0's when in fact I only have one 1.

   So if I pick k = 2, then that means $xy^kz = xyyz$, and if $\mathcal{L}$ is regular, then this word should also be in $\mathcal{L}$. Now, I still know that $\#1(z) = \#1(w) = n$, but now $\#0(xyy) \neq \#0(w)$, this means I have at least one more 0 than I had originally, since y is nonempty and has to consist of at least one 0. The condition is that **for all k**, xy^kz should be in $\mathcal{L}$. But if $\#0(xyy) \neq \#0(w) < 2 * \#1(w)$, then $\#0(xyy)$ does not have to be $< 2 * \#1(z)$ anymore, which means that $xyyz \notin \mathcal{L}$. For instance, if the number of 0's in xy was one, and the number of 1's in z was one, then by adding at least one 0, xyy consists of at least two 0's now, which is not less than the double amount of 1's.

6. Hence, $\mathcal{L}$ is not regular.

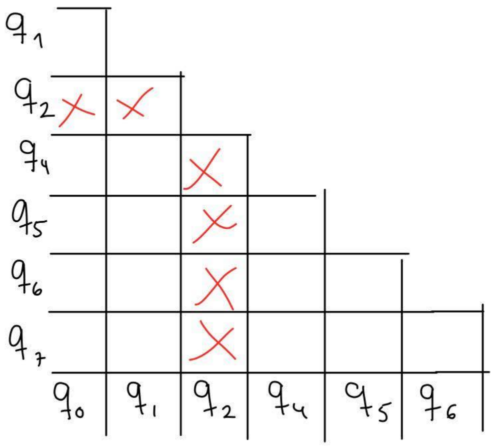
4 Task 4

The first thing I want to change in the table is to remove the state q_3 altogether, because it is not accessible from the other states. This gives me the new table

	0	1
$\rightarrow q_0$	q_1	q_7
q_1	q_6	q_2
$\star q_2$	q_0	q_2
q_4	q_5	q_7
q_5	q_6	q_2
q_6	q_6	q_4
q_7	q_6	q_2

- The first step is, since we only have one accepting state, cross all boxes for q_2, because by reading nothing, from q_2 we will end up in a final state, but for every other state we will not, so these are distinguishable.
- Now that I have some distinguishable pairs, I can move on to find the rest of them.

This is the table I have now:



	q_0	q_1	q_2	q_4	q_5	q_6
q_1						
q_2	X	X				
q_4			X			
q_5			X			
q_6			X			
q_7			X			

This means that the distinguishable pairs I have now are:

- (q_2, q_0)
- (q_2, q_1)
- (q_2, q_4)
- (q_2, q_5)

- (q_2, q_6)
- (q_2, q_7)

So in order to disclose the other distinguishable pairs, I will use the recursive step approach used in the slides. That is, *Let p and q be such that for some a, δ(p,a) = r and δ(q,a) = s with (r,s) known to be distinguishable. Then (p,q) are also distinguishable.* That is, my (r,s) will be any of the above listed for starters.

So, to start off, I will choose to start to look at my state q_0, and see how this state will transition when it reads 0 or 1.

$$\delta(q_0, 0) = q_1$$

$$\delta(q_0, 1) = q_7$$

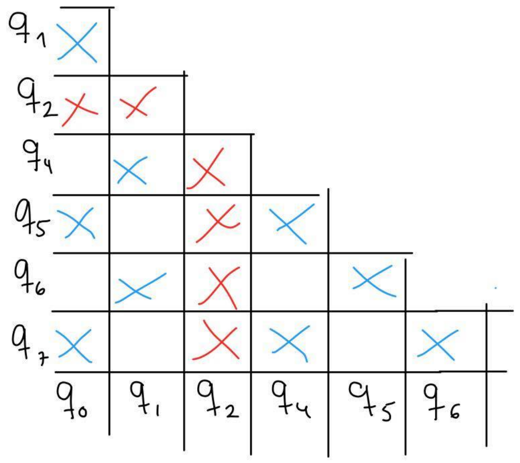
These will be my r's when comparing to another state. So, the state I'll start comparing to will be q_1, so I check the transitions for this too.

$$\delta(q_1, 0) = q_6$$

$$\delta(q_1, 1) = q_2$$

First off, the pair (q_1, q_6) is not in the list of so far discovered distinguishable pairs. That is, this box in my above table is blank, which means that from this information only I can't not tell if (q_0, q_1) is distinguishable or not. However, the pair (q_7, q_2) is in the list, so if this pair is distinguishable, then (q_0, q_1) will be too. So I put a cross in that box.

I will continue with this first recursive step q_0 up to q_6, then when I have done one round of this I will start over with q_0 as well to see if any new pair of states have been marked. The first round of checks leaves me with this table (marked with blue):

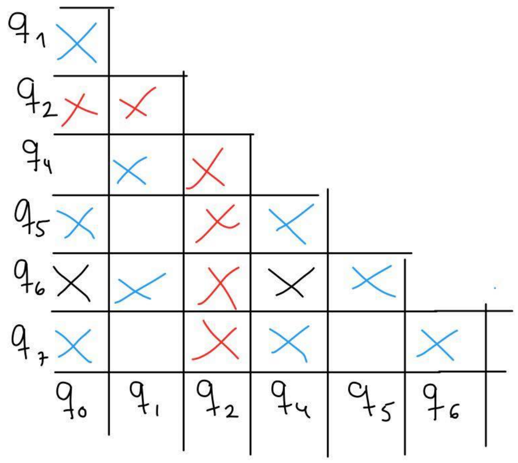


q_1	X					
q_2	X	X				
q_4		X	X			
q_5	X		X	X		
q_6		X	X		X	
q_7	X		X	X		X
	q_0	q_1	q_2	q_4	q_5	q_6

So to add to my existing list of distinguishable pairs, I will now also have the follow pairs as distinguishable:

- (q_0, q_1)
- (q_0, q_5)
- (q_0, q_7)
- (q_1, q_4)
- (q_1, q_6)
- (q_4, q_5)
- (q_4, q_7)
- (q_5, q_6)
- (q_6, q_7)

So now for round 2 I will also have to see if the resulting (r,s) is any of the, in total, 15 pairs. This leaves me with the following, and resulting, table (marked with black):



q_1	X					
q_2	X	X				
q_4		X	X			
q_5	X		X	X		
q_6	X	X	X	X	X	
q_7	X		X	X		X
	q_0	q_1	q_2	q_4	q_5	q_6

The next step is to actually minimize the automaton. From the book: "Then, partition the remaining states into blocks, so that all states in the same block are equivalent, and no pair of states from different blocks are equivalent." This means that my first partition will be

- (q_0, q_4)

and these two states are not equivalent with any other state, so that partition is already done. My next pair of equivalent states are

- (q_1, q_5)

but q_1 is also equivalent to q_7, which means the resulting partition will be

- (q_1, q_5, q_7)

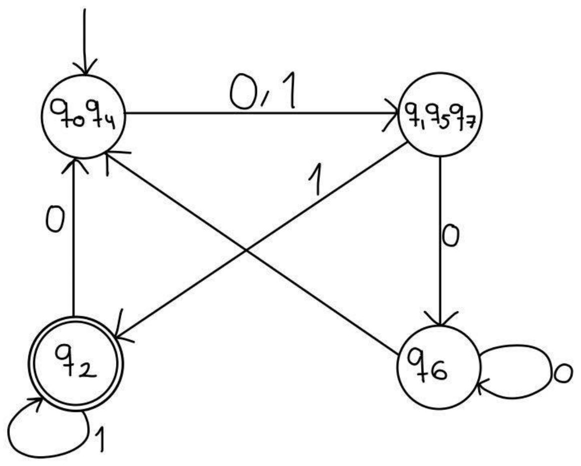
Then I also have (q_5, q_7), but since this is already a part of the above partition I do not have to take any extra actions.

Another, alone, equivalent state is in fact q_6. This state is distinguishable from any other state and so it must be equivalent only to itself.

The last equivalent state is actually the finishing state, since it's only equivalent to itself and that means my four partitions will be

- $(q_0, q4)$
- (q_1, q_5, q_7)
- (q_6)
- (q_2)

These are the states that I will use for my new, minimized automaton. The starting state will be the one containing q_0, so (q_0, q_4). The finishing state will still be q_2. When I write down the transitions, I simply see where q_0 goes on input 0, and then where q_4 would go. This I do for all states, leaving me with the minimized automaton:


q0q4
q1q5q7
q2
q6
0,1
1
0
0
0
1