CS3800: Theory of Computation — Summer II '22 — Drew van der Poel

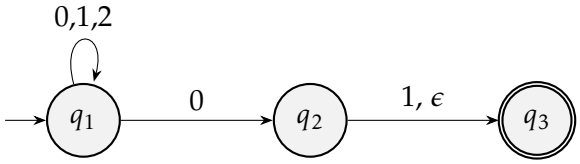
Homework 2
Due Friday, July 22 at 11:59pm via Gradescope

Name: Steve Liu
Collaborators: Ares Do, Jamie Lin, Eric Chapelaine, Michael Fang, HwiJoon Lee, Jaron Cui

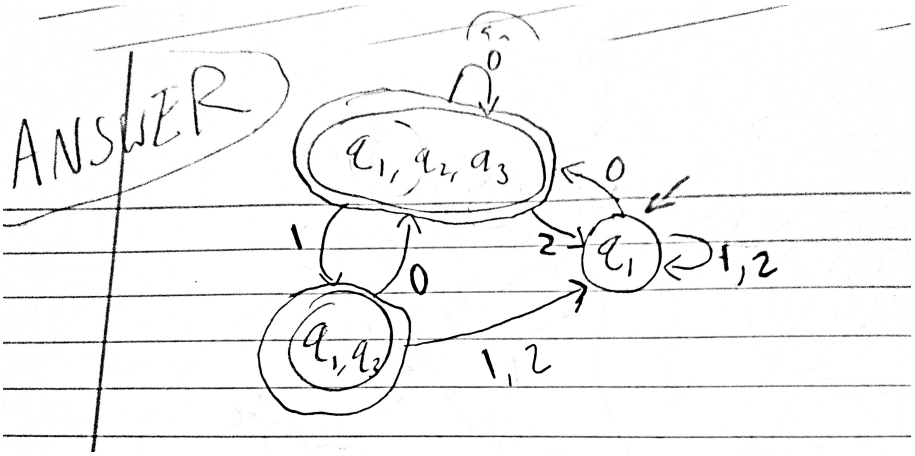
- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- This assignment is due Friday, July 22 at 11:59pm via Gradescope. No late assignments will be accepted. Make sure to submit something before the deadline.
- Solutions must be typeset. If you need to draw any diagrams, you may draw them by hand as long as they are embedded in the PDF. I recommend using the source file for this assignment to get started.
- I encourage you to work with your classmates on the homework problems. *If you do collaborate, you must write all solutions by yourself, in your own words.* Do not submit anything you cannot explain. Please list all your collaborators in your solution for each problem by filling in the `yourcollaborators` command.
- Finding solutions to homework problems on the web, or by asking students not enrolled in the class is strictly forbidden.

Problem 1. *Converting NFAs to DFAs (*6 points*)*

Convert the following NFA to a DFA. Your DFA should be as simple as possible (e.g. any states which cannot be reached from the initial state should be removed). Note $\Sigma = \{0, 1, 2\}$.



Solution:



WORK FOR PROBLEM ON NEXT PAGE

δ	0	1	2	ε
q_1	$\{q_1, q_2\}$	$\{q_1\}$	$\{q_1\}$	$\emptyset$
q_2	$\emptyset$	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

$E(q_1) = \{q_1\}$

$E(q_2) = \{q_2, q_3\}$

$E(q_3) = \{q_3\}$

	0	1	2
~~q_1, q_2, q_3~~	~~q_1, q_2, q_3~~	~~q_1, q_3~~	~~q_1~~
~~q_1, q_2~~	~~q_1, q_2, q_3~~	~~q_1, q_3~~	q_1
q_1, q_3	q_1, q_2, q_3	q_1	q_1
~~q_2, q_3~~	$\emptyset$	q_3	$\emptyset$
q_1	q_1, q_2, q_3	q_1	q_1
~~q_2~~	$\emptyset$	$\{q_3\}$	$\emptyset$
~~q_3~~	$\emptyset$	$\emptyset$	$\emptyset$
~~$\emptyset$~~	$\emptyset$	$\emptyset$	$\emptyset$

$E(\delta(q_1,q_2,q_3),0) = E(\delta(q_1,0)) \cup E(\delta(q_2,0)) \cup E(\delta(q_3,0)) = E(q_1,q_2) \cup \emptyset = \{q_1,q_2,q_3\}$

$E(\delta(q_1,q_2,q_3),1) = E(\delta(q_1,1)) \cup E(\delta(q_2,1)) \cup E(\delta(q_3,1)) = E(q_1,q_3,\emptyset) = \{q_1,q_3\}$

$E(\delta(q_1,q_2,q_3),2) = E(\delta(q_1,2)) \cup E(\delta(q_2,2)) \cup E(\delta(q_3,2)) = E(q_1,\emptyset,\emptyset) = \{q_1\}$

$E(\delta(q_1,q_2),0) = E(\delta(q_1,0)) \cup E(\delta(q_2,0)) = E(\{q_1,q_2\} \cup \emptyset) = \{q_1,q_2,q_3\}$

$E(\delta(q_1,q_2),1) = E(\delta(q_1,1)) \cup E(\delta(q_2,1)) = E(q_1 \cup q_3) = \{q_1,q_3\}$

$E(\delta(q_1,q_2),2) = E(\delta(q_1,2)) \cup E(\delta(q_2,2)) = E(q_1 \cup \emptyset) = \{q_1\}$

$E(\delta(q_1,q_3),0) = E(\delta(q_1,0)) \cup E(\delta(q_3,0)) = E(\{q_1,q_2\} \cup \emptyset) = \{q_1,q_2,q_3\}$

$E(\delta(q_1,q_3),1) = E(\delta(q_1,1)) \cup E(\delta(q_3,1)) = E(q_3 \cup \emptyset) = \{q_3\}$

$E(\delta(q_1,q_3),2) = E(\delta(q_1,2)) = E(q_1) = \{q_1\}$

$E(\delta(q_2,q_3),0) = E(\delta(q_2,0)) \cup \delta(q_3,0) = \emptyset$

$E(\delta(q_2,q_3),1) = E(q_3) = \{q_3\}$

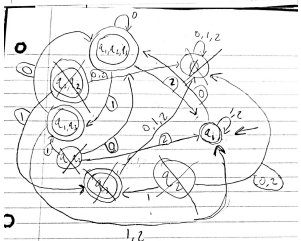
$E(\delta(q_2,q_3),2) = \emptyset$

$E(\delta(q_1),0) = E(q_1,q_2) = \{q_1,q_2,q_3\}$

$E(\delta(q_1),1) = E(q_1) = \{q_1\}$

$E(\delta(q_1),2) = E(q_1) = \{q_1\}$

$E(\delta(q_2),0) = \emptyset$, $E(\delta(q_2),1) = q_3$, $E(\delta(q_2),2) = \emptyset$



Problem 2. *Regular Expressions (*6 points*)*

Give a regular expression for the language of words over $\Sigma = \{0, 1\}$ which have an equal number of copies of 01 and 10 as substrings (note: these copies need not be disjoint). Show why your regular expression accepts the words 10101, 0101110, 111, and 111111011. Explain why it does not generate 0101. Provide brief justification for why your regular expression works as desired.

Solution:
REGULAR EXPRESSION: 0(0 ∪ 1)*0 ∪ 1(0 ∪ 1)*1 ∪ ϵ ∪ 1 ∪ 0

10101 → 1(010)1 is in 1(0 ∪ 1)*1 → you have the left and right side 1's with 010 in the middle which is in (0 ∪ 1)*

0101110 → 0(10111)0 is in 0(0 ∪ 1)*0 → you have the left and right side 0's with 10111 in the middle which is in (0 ∪ 1)*

111 → 1(1)1 is in 1(0 ∪ 1)*1 → you have the left and right side 1's with 1 in the middle which is in (0 ∪ 1)*

111111011 → 1(1111101)1 is in 1(0 ∪ 1)*1 → you have the left and right side 1's with 1111101 in the middle which is in (0 ∪ 1)*

0101 cannot be generated because it does not belong in 0(0 ∪ 1)*0 as it does not start or end in 0, nor 1(0 ∪ 1)*1 for the same reason (it does not start or end with 1), nor is it an empty string, nor the string 1, and it is not the string 0.

EXPLANATION: Getting the easiest cases out of the way, this language should accept the empty string, 1, and 0 since they all have zero 01s and 10s. For other strings, we can break down the possibilities into two separate cases:

Case 1: The string begins with a 1, the next input in the string can either be a 0 or a 1. We can disregard all 1's until the first 0 since the number of 01's and 10's do not change. Once we reach our first 0, then we get to a case where there are one 10's and zero 01's in the string. Turns out, because of the "non-disjoint" rule of this language, if we let n be the length of the string, the number of 10's can be counted in the first $n-1$ substring and the number of 01 can be counted in the last $n-1$ substring (i.e. take the string, 10101 has 2 01's and 2 10's, the first $n-1$th substring = 1010 has 2 10's and last $n-1th$ substring = 0101 has 2 01's). As a result of this rule, the number of 10's will always be exactly one greater than the number of 01's if the string starts with a 1 and ends in a 0 because the first 0 will always be able to create a 10 but the last 0 will never be able to create a matching 01. In order to guarantee that the first and last zeroes create an equivalent number 10's and 01's, the last input of the string has to be a 1 in order to match the last 0 in the string.

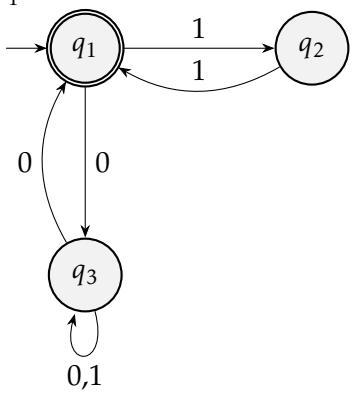
Case 2: The string begins with a 0. This case is entirely symmetrical to the first case with the same

idea but this time the number of 01's will always be exactly one greater than the number 10's if the string begins with 0 and ends in 1 for the same symmetrical reason mentioned in Case 1. To remediate this issue, we can basically do the same thing by enforcing that the last input of the string has to be a 0.

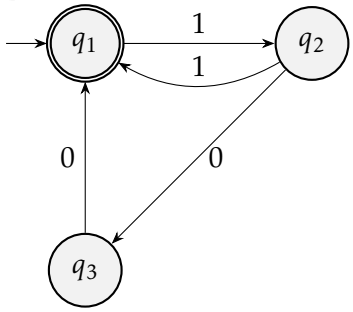
Problem 3. *NFA to RegEx (*9 points*)*

Consider the following NFAs. For each, convert it to a regular expression. Show all intermediate steps when building and simplifying the GNFA. You can omit transitions labeled with $\emptyset$.

D_1:

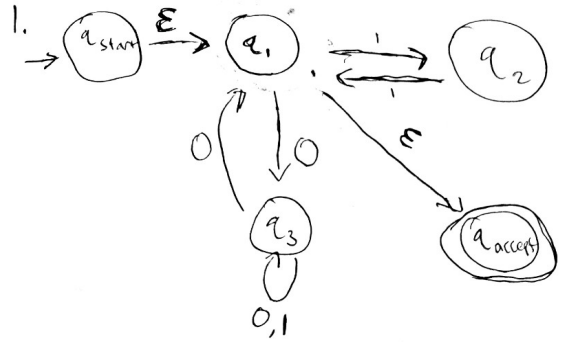


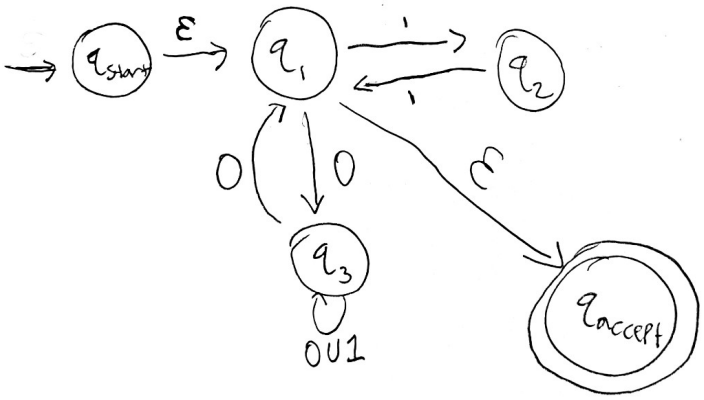
D_2:



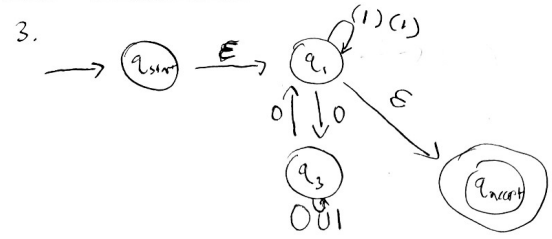
Solution:
D1: $(11 \cup 0(0 \cup 1)^*0)^*$
D2: $(11 \cup 100)^*$
1st image is D1, 2nd image is D2

1.

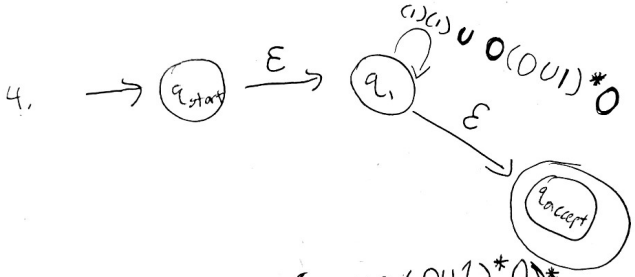

q_{start}
ε
q_1
1
1
q_2
0
0
q_3
0,1
ε
q_{accept}


q_{start}
ε
q_1
1
1
q_2
0
0
q_3
0U1
ε
q_{accept}

3.

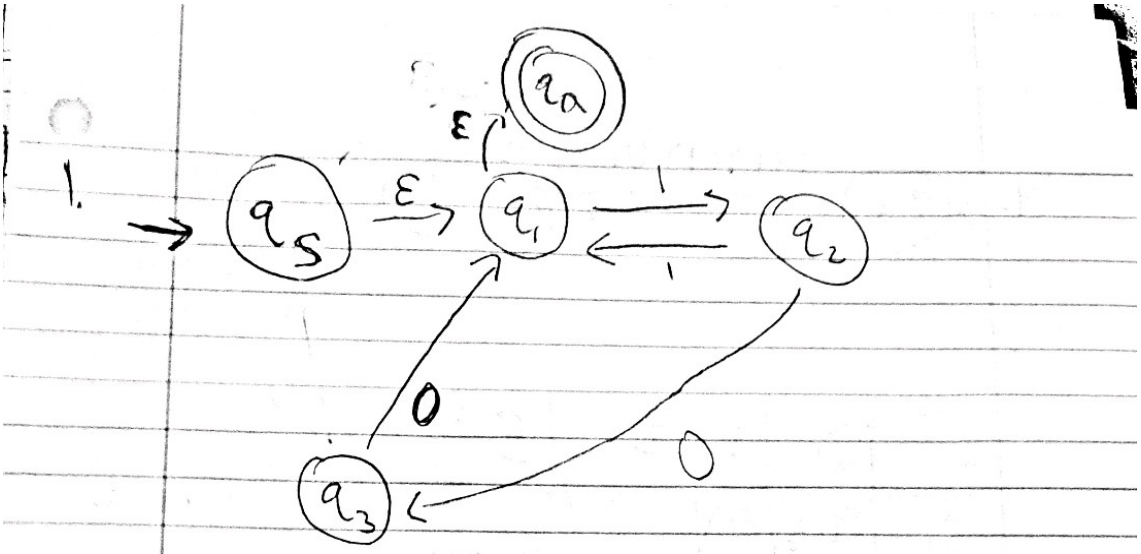

q_{start}
ε
q_1
(1)(1)
0
0
q_3
0U1
ε
q_{accept}

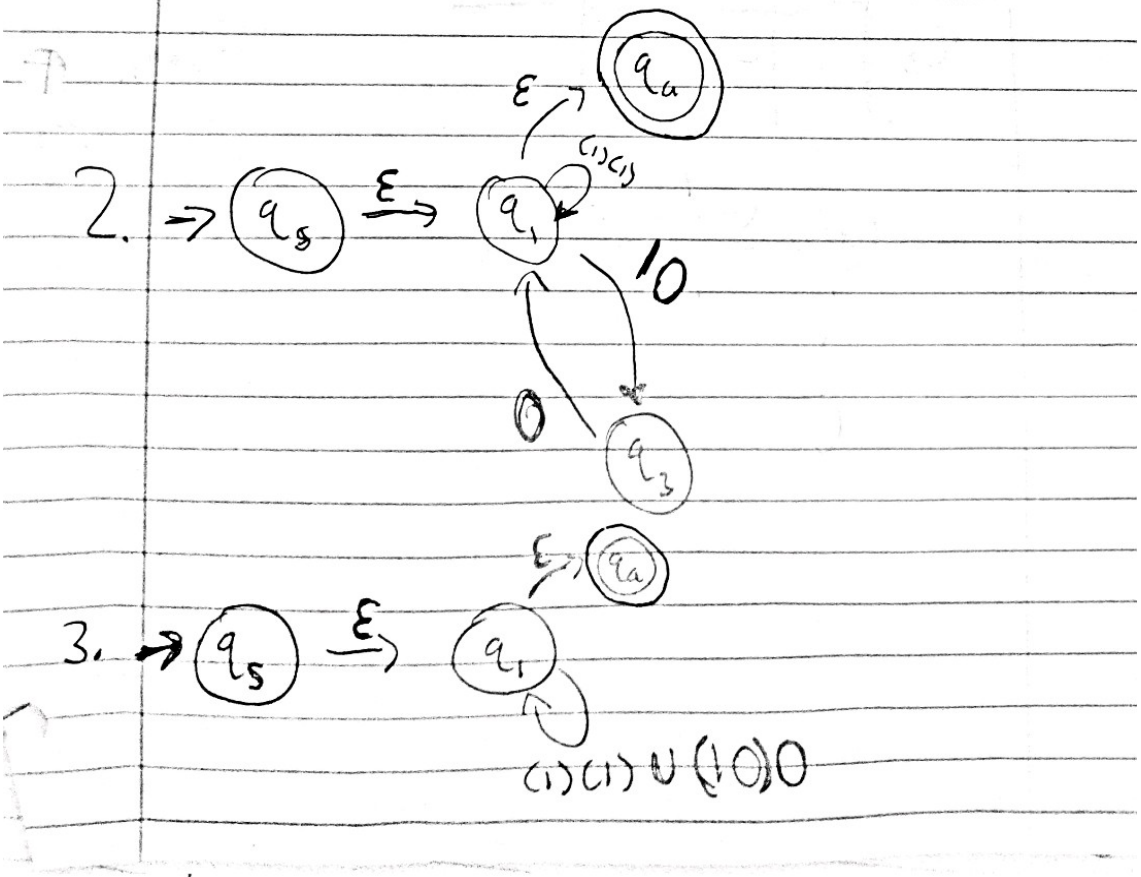
4.

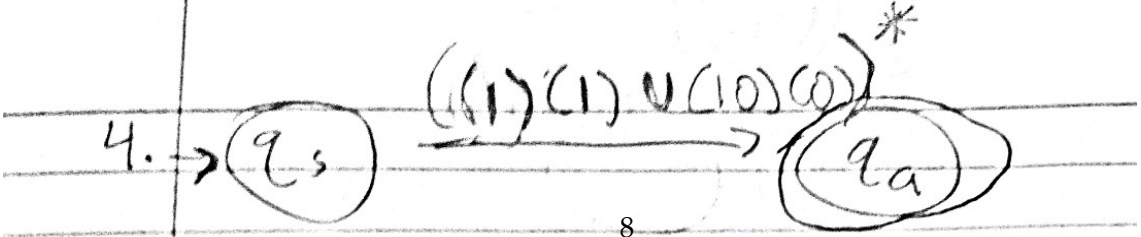

q_{start}
ε
q_1
(1)(1) U 0(0U1)*0
ε
q_{accept}

5.


q_{start}
((1)(1) U 0(0U1)*0)*
q_{accept}


1.
q_s
q_1
q_2
q_3
q_a
ε
ε
1
1
0
0


2.
q_s
q_1
q_3
q_a
ε
ε
(1)(1)
10
0
3.
q_s
q_1
q_a
ε
ε
(1)(1) ∪ (10)0


4.
q_s
q_a
((1)(1) ∪ (10)(0))*

Problem 4. *Non-Regular Languages (*9 points*)*

For each of the following languages over $\Sigma = \{g, q\}$, state whether or not it is regular. If it is, give a regular expression that describes it. If not, prove why using the Pumping Lemma. Note that we assume $\mathbb{N}$ (the natural numbers) to be the non-negative integers ($\{0, 1, 2, \ldots\}$).

(a) **[3 pts.]** $L_1 = \{w | w \in \{0,1\}^*;$ number of 00 substrings is the same as the number of 11 substrings$\}$

Solution:
Assume true, that L_1 is a regular language so the Pumping Lemma applies to L_1. Let P be the pumping length. Let $S = (00)^P(11)^P$. We know that via the Pumping Lemma, $\left|xy\right| \leq P$ which means that y must consist of some number of pairs of only zeroes for string S. We also know that the Pumping Lemma states $xy^iz \subseteq L_1$ which is not true when $i = 3$ for S since $xyyyz \not\subseteq L_1$ as we will add more 00's to the string resulting in it having more 00's than we will have 11's. Thus, by contradiction, L_1 does not uphold the Pumping Lemma which means that L_1 is not regular.

(b) **[3 pts.]** $L_2 = \{0^n 1^m | n, m \geq 5; n, m \in \mathbb{N}\}$

Solution:
There has to be atleast five 0's in the string and five 1's in the string where all of the zeroes must be before all of the ones. Thus the regular expression for this language would be: (00000)(0)*(11111)(1)*

(c) **[3 pts.]** $L_3 = \{w | w \in \{0,1\}^*;$ the length of w is odd and contains a 1 as the middle character$\}$

Solution:
Assume true, that L_3 is a regular language so the Pumping Lemma applies to L_3. Let P be the pumping length. Let $S = (00)^P 1 (00)^P$. We know that via the Pumping Lemma, $\left|xy\right| \leq P$, which means that y must consist of some number of pairs of only zeroes for string S. However, we also know that the Pumping Lemma states $xy^iz \subseteq L_3$ which is not true when $i = 3$ for S since adding any extra zeroes on the left side of S will result in the left side having more 0s than the right side which means that the string no longer has a 1 in the middle. Thus, by contradiction, L_3 does not uphold the Pumping Lemma which means that L_3 is not regular.