CS Theory (Fall '25) Assigned: Sept 23, 2025

Homework 2

Instructors: *William Pires and Toniann Pitassi* Due: Oct 2, 2025 at 11am

Collaboration: Collaboration with other students in the class homework is allowed. However, you must write up solutions by yourself and understand everything that you hand in. You may also consult other reference materials, but you may not seek out answers from other sources or use AI tools. **You are required to list who you collaborated with on each problem, including any TAs or students you discussed the problems with in office hours. Also list any reference materials consulted other than the lectures and textbook for our class.** See the course webpage for more details.

Formatting: Write the solution to each part of each problem on a separate page. Be sure to correctly indicate which page each problem appears on in GradeScope. Please do not write your name on any page of the submission; we are using anonymous grading in GradeScope.

Grading: There are 50 total possible points, not including the Bonus problem which is optional and for extra credit. For all problems except for the Bonus problem, you have the option of answering "Don't know" on any parts of the question, and you will receive 20 percent of the total marks for those parts.

0 Exercises

Here are some recommended exercises. You should not turn in your solutions for these, and we will not grade them.

(a) Exercises 1.11,1.14b,1.17

For Problem 1.11, the solution is in the book.

(b) Prove that a regular language L is finite if and only if it can be written as a regular expression that does not involve the '*' operation.

(a) 1.14b: There are many examples of NFAs that you could give, but here is one:

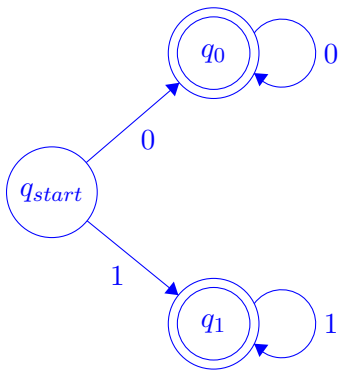

Figure 1: NFA solution for exercise (a, 1.14b)

This NFA accepts strings that are only 0s or only 1s. However, if we make the non-accept states to accept states, and vice versa, we would get an NFA that accepts no strings.

However, this does *not* imply that the set of all languages recognized by NFAs is not closed under complements. Because the set of languages recognized by NFAs is the same as the set of languages recognized by DFAs, the fact that the set of languages recognized by DFAs is closed under complements means that the set of languages recognized by NFAs is also closed under complements.

1.17: Here is one NFA that recognizes this language:

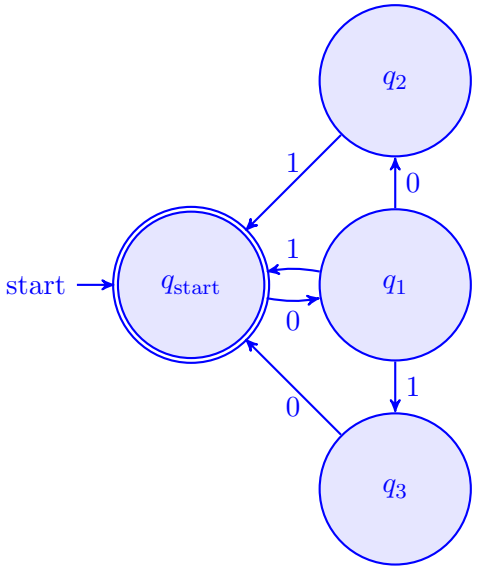

Figure 2: NFA solution for exercise (a, 1.17a)

When we turn this into a DFA, we get the following:

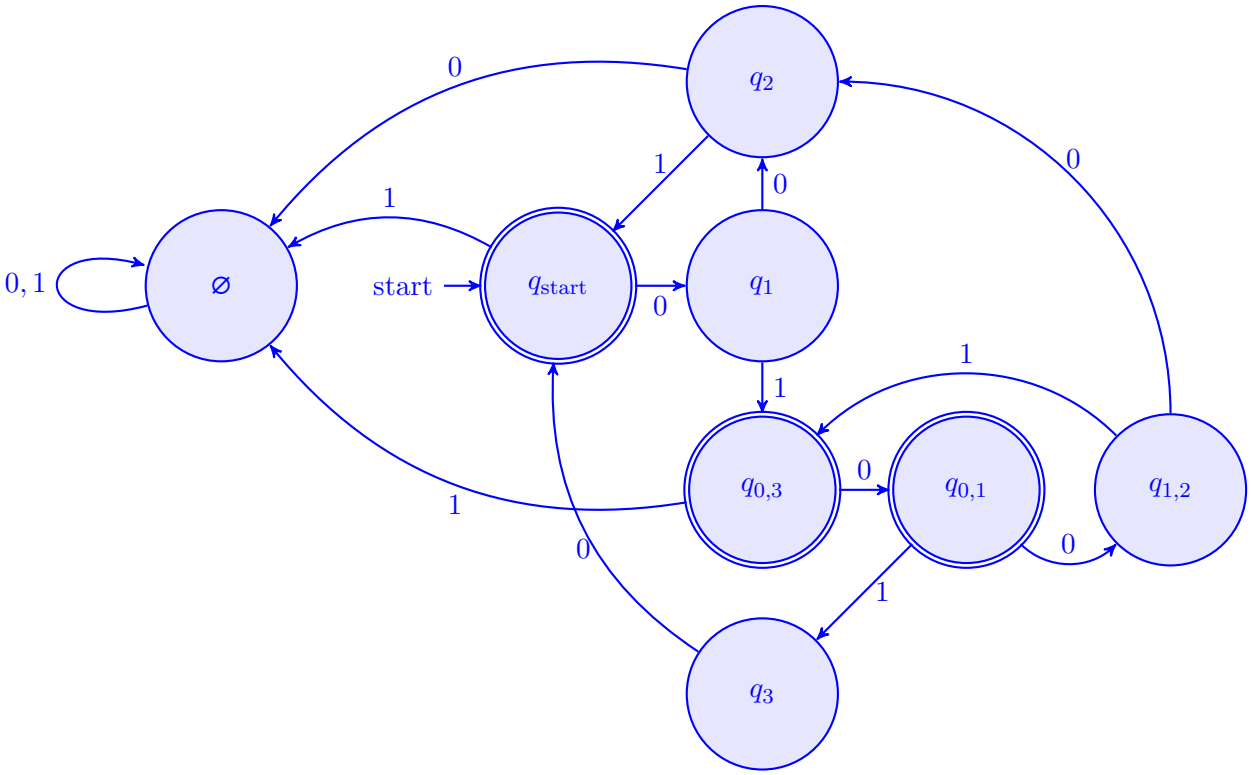

Figure 3: DFA solution for exercise (a, 1.17b)

(b) In this problem, we'll use the notation that, if R is a regular expression, $\mathcal{L}(R)$ is the language represented by that regular expression.

In the forward direction, we will show that a language is finite if it can be written as a regular expression without "*". We'll show this by induction. Suppose that all regular expressions of length less than ℓ without "*" represent finite languages. Then, consider a regular expression R of length ℓ. There are two cases: $R = R_1 \cup R_2$ or $R = R_1 \circ R_2$, where R_1 and R_2 are regular expressions of length less than ℓ.

In the first case, we know that $|\mathcal{L}(R)| \leq |\mathcal{L}(R_1)| + |\mathcal{L}(R_2)|$ (because of the general fact that $|A \cup B| \leq |A| + |B|$). Since both $\mathcal{L}(R_1)$ and $\mathcal{L}(R_2)$ are finite, $\mathcal{L}(R)$ is as well.

In the second case, we know that $|\mathcal{L}(R)| = |\mathcal{L}(R_1)| \cdot |\mathcal{L}(R_2)|$ (because $|A \times B| = |A| \cdot |B|$). Again, since both $\mathcal{L}(R_1)$ and $\mathcal{L}(R_2)$ are finite, $\mathcal{L}(R)$ is as well.

For the base case, we show that all regular expressions of length 1 without "*" represent finite languages. There are three possibilities: $R = a$ for $a \in \Sigma$, $R = \varepsilon$, or $R = \varnothing$. All three of these regular expressions represent languages of cardinality 1 or 0.

In the reverse direction, we will show that a language can be written as a regular expression without "*" if it is a finite language. Let $L = \{s_1, \ldots, s_n\}$ be a finite language. This is represented by the regular expression $s_1 \cup \cdots \cup s_n$.

1 Problems

1.1 NFA to DFA (10 points)

Consider the following NFA over the alphabet $\{a, b, c\}$.

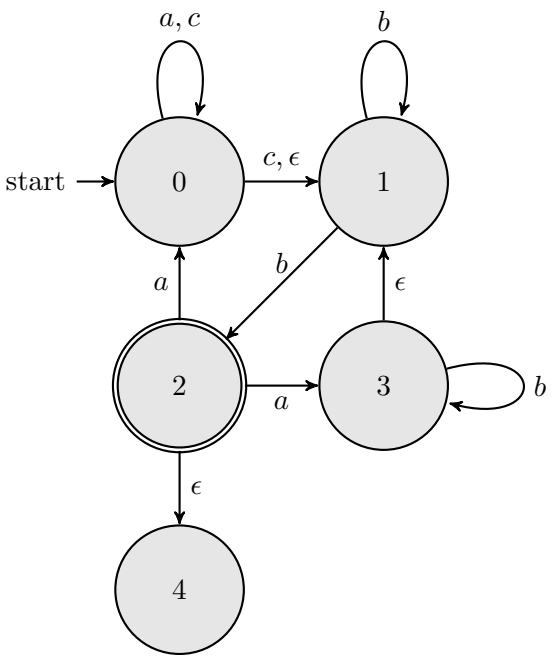

Figure 4: NFA for problem (1).

Convert this NFA into a DFA. Draw the final DFA after removing states that can't be reached from the start state. In the DFA, your states should be named as subsets of the states of the NFA. (I.e. your states should have names like $\varnothing, \{1\}, \{2, 3, 4\}$ etc...).

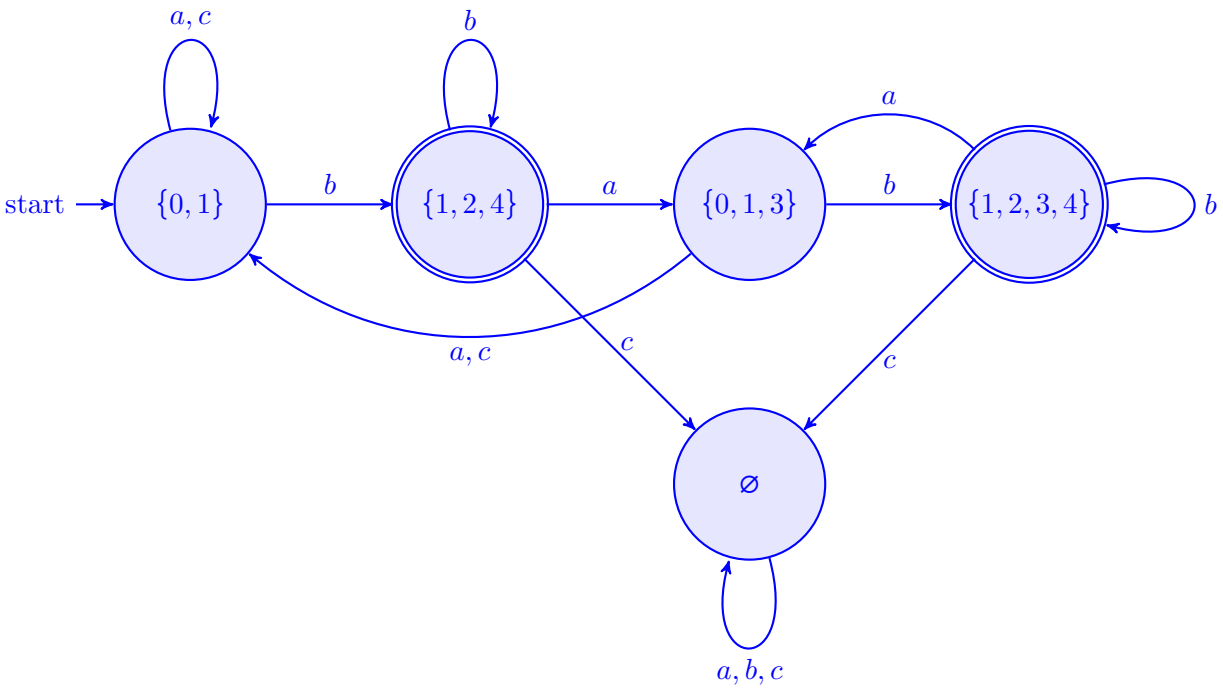

Figure 5: DFA solution for problem (1)

1.2 Strange Regular Language (8 points)

Let L_1, L_2 be regular languages over alphabet Σ. Define a new language

$$L = \{w \mid w = rst \text{ where } r \in L_1, s \in L_2 \text{ and } t \in L_1 \setminus L_2\}.$$

That is, L consists of all strings $w \in \Sigma^*$ such that w can be written as the concatenation of three strings, r, s, t such that r is in L_1, s is in L_2 and t is in L_1 but not in L_2. Prove that L is regular. (Hint: you may use without proof the four closure properties that we proved in class, but any other property or regular languages that you use, you need to prove.).

First, we will show that the set of regular languages is closed under intersection. We will build this from the closure properties that we already showed in class. Suppose L_1 and L_2 are regular. By De Morgan's Laws,

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}.$$

Since the set of regular languages is closed under complements, $\overline{L_1}$ and $\overline{L_2}$ are both regular. Since the set of regular languages is closed under unions, $\overline{L_1} \cup \overline{L_2}$ is regular. Finally, again using the fact that the set of regular languages is closed under complements, $\overline{\overline{L_1} \cup \overline{L_2}} = L_1 \cap L_2$ is regular.

Now, we will show that if L_1 and L_2 are regular, $L_1 \setminus L_2$ is regular. By definition, $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$. Since both L_1 and $\overline{L_2}$ are regular, and because we showed above that the intersection of regular languages is regular, $L_1 \setminus L_2$ is regular.

Finally, we observe that $L = L_1 \circ L_2 \circ (L_1 \setminus L_2)$. Since L_1, L_2, and $L_1 \setminus L_2$ are regular, and because the concatenation of regular languages is regular, L is regular.

1.3 Some weird properties of regular language (20 points)

The following two problems ask you to explore new ways to construct a new regular language L' from an existing regular language, L. For both questions, your proof should follow the same format as the proofs we did in class/book for showing that regular languages are closed under complement, union, concatenation, and star. First give a construction of an NFA for L', using a DFA for L, and then explain/prove correctness of your construction. Be sure to argue both directions of correctness: (i) first argue that every string in L' will be accepted by your NFA, and (ii) secondly argue that every string not in L' will be rejected by your NFA.

(a) Let L be a language over $\Sigma = \{a, b\}$. Define $\text{Swap}(L)$ to be the language consisting of all strings over $w \in \{a, b, c\}$ such that by replacing each c of w by a or b, we can obtain a string in L. Prove that if L is regular, then $\text{Swap}(L)$ is regular.

For example if $L = \{aab, bb\}$. Then $w = cc \in \text{Swap}(L)$ since we can swap the c's to get $bb \in L$. The string $w = cac$ is also in $\text{Swap}(L)$ since we can replace the first c by a, the second c by b and get $aab \in L$. However, the strings $cccc$, bcc, ac aren't in $\text{Swap}(L)$.

(b) Define $L' = \text{Leave-1-Out}(L)$ to be the language consisting of all strings $w \in \{0, 1\}^*$ such that by inserting *exactly* one symbol somewhere in w, we can obtain a string in L. Prove that if L is a regular language, then so is $\text{Leave-1-Out}(L)$.

For example, if L consists of all strings that start with 110, then $w = 1111$ is in $\text{Leave-1-Out}(L)$ since we can insert a 0 after the 2rd symbol of w to obtain 11011 which is in L. But $w = 001$ isn't in $\text{Leave-1-Out}(L)$ since we can't insert a symbol in w to make it start with 110.

If $L = \{1\}$, then $w = 1$ is *not* in $\text{Leave-1-Out}(L)$ since we need to insert *exactly* one symbol.

(a) Suppose L over $\Sigma = \{a, b\}$ is regular, which implies that L is recognized by a DFA $D = (Q, \Sigma, \delta, q_0, F)$. Our goal is to show $\text{Swap}(L)$ is regular by utilizing D to construct an NFA N that recognizes $\text{Swap}(L)$. The idea for how the NFA will determine if a string $w \in \{a, b, c\}^*$ is in $\text{Swap}(L)$ is:

1. The NFA needs to consider all the ways w can swap each c into an a or b. This gives us "candidate strings" that may or may not be in L. At least one candidate is in L iff w is in $\text{Swap}(L)$.
2. The NFA needs to nondeterministically guess which candidate string w' is actually in L and run the DFA D on it to verify. If D accepts the candidate string w', the NFA should accept w.

Let us construct such an NFA. We define the NFA $N = (Q', \Sigma', \delta', q_0', F')$ such that

1. The set of states, the set of accept states, and the start state are the same as that for the DFA D. That is, $Q' = Q$, $F' = F$, and $q_0' = q_0$.
2. The alphabet is $\Sigma' = \{a, b, c\}$.
3. The transition function $\delta' : Q \times (\Sigma' \cup \{\epsilon\}) \to \mathcal{P}(Q)$ is defined for any state $q \in Q$ and character $\sigma \in \Sigma'$ as
$$\delta'(q, \sigma) = \begin{cases} \{\delta(q, \sigma)\} & \sigma \in \{a, b\} \\ \{\delta(q, a)\} \cup \{\delta(q, b)\} & \sigma = c \end{cases}$$

For ϵ transitions, we define $\delta'(q, \epsilon) = \varnothing$ for any state $q \in Q$.

Observe that every branch of computation simulated by the NFA N on input w corresponds to the path of states traveled by the DFA D on an input w', where w' is some swapping of the c's in w into a or b. We can see this by considering any sequence of two states $r_i, r_{i+1} \in Q'$ where $r_{i+1} \in \delta'(r_i, w_{i+1})$. When $w_{i+1} \in \{a, b\}$, we have $r_{i+1} = \delta(r_i, w_{i+1})$. When $w_{i+1} = c$, we have either $r_{i+1} = \delta(r_i, a)$ or $r_{i+1} = \delta(r_i, b)$. This tells us that the NFA follows the same transitions as the DFA until it reads in a c, after which it nondeterministically replaces the c with an a or b and follows the corresponding transition in the DFA.

Therefore if $w \in \text{Swap}(L)$, then there exists some $w' \in L$ that can be obtained from swapping each c in w by a or b. Then one of the branches of computation the NFA simulates on w must be the path of states computed by the DFA D on input $w' \in L$, which ends in an accept state. Thus the NFA accepts $w \in \text{Swap}(L)$.

Furthermore, if $w \notin \text{Swap}(L)$, then $w' \notin L$ for any w' that can be obtained from swapping c's in w with a or b. It follows that every branch of computation the NFA simulates on input w must end in a reject state, since the DFA D will end in a reject state for all of the swapped strings w' obtained from w. Thus the NFA rejects $w \notin \text{Swap}(L)$.

We have shown that there exists an NFA that correctly recognizes $\text{Swap}(L)$, and therefore we conclude that $\text{Swap}(L)$ is regular.

(b) Suppose the language L over the alphabet Σ is regular, which implies that there exists a DFA $D = (Q, \Sigma, \delta, q_0, F)$ that recognizes L. We can rewrite $\text{Leave-1-Out}(L)$ as

$$\text{Leave-1-Out}(L) = \{xy \mid x\sigma y \in L \text{ for some symbol } \sigma \in \Sigma \text{ and strings } x, y \in \{0,1\}^*\}.$$

We want to show that $\text{Leave-1-Out}(L)$ is regular by constructing an NFA that recognizes it. One idea is where on input string w, the NFA nondeterminsitically chooses a w' satisfying $w \in \text{Leave-1-Out}(w')$ (so w' is w with a symbol inserted somewhere), then the NFA can check if w' is in L using the DFA D.

To do so, we want to construct an NFA where each branch of computation corresponds to (1) a symbol σ to add in, and (2) a place in the input string w to insert the symbol, that is a partition of w into two strings $w = x \circ y$. In this branch, the NFA would perform the following steps:

1. keep track of what state q_x the DFA D ends in after reading x,
2. use an epsilon transition transition to go to another state $q_{x\sigma}$ that is connected to q_x in the DFA (this lets the NFA simulate the DFA on $w' = x\sigma y$, where we transition as if we "added back" the "left-out" symbol σ),
3. then starting from this $q_{x\sigma}$ state, follow the DFA transitions for reading in y

More formally, we construct this NFA as $N = (Q', \Sigma', \delta', q_0', F')$ such that

1. The set of states $Q' = Q \times \{0, 1\}$ keeps track of both the state in Q and whether or not a symbol has been "added" yet (0 for false, 1 for true).
2. The alphabet is $\Sigma' = \{0, 1\}$ since this is how $\text{Leave-1-Out}(L)$ is defined.

3. Define the transition function $\delta' : Q' \times (\Sigma' \cup \{\epsilon\}) \to \mathcal{P}(Q')$, or equivalently $\delta' : Q \times \{0,1\} \times (\Sigma' \cup \{\epsilon\})$, so that for any state $(q,b) \in Q'$ and $s \in \Sigma' \cup \{\epsilon\}$,

$$\delta'((q,b),s) = \begin{cases} \{(\delta(q,s),b)\} & s \in \Sigma' = \{0,1\} \text{ (transitions corresponding to step 1 or 3)} \\ \bigcup_{\sigma \in \Sigma} \{(\delta(q,\sigma),1)\} & s = \epsilon, b = 0 \text{ (transitions corresponding to step 2)} \\ \varnothing & s = \epsilon, b = 1 \text{ (we do not allow "adding" a symbol more than once)} \end{cases}$$

4. The start state is $q_0' = (q_0, 0)$.
5. The accept states are $F' = F \times \{1\}$.

To prove that the NFA indeed recognizes Leave-1-Out(L), we will show that $w \in$ Leave-1-Out(L) if and only if w is accepted by the NFA.

(1) $w \in$ Leave-1-Out$(L) \implies w$ is accepted by the NFA

If $w \in$ Leave-1-Out(L), then there exists some symbol $\sigma \in \Sigma$ so that for some partition $w = xy$ where $x, y \in \{0,1\}^*$ are strings, we can undo the leave-1-out operation on w to get $w' = x\sigma y \in L$. Let the the lengths of x and y be $|x| = m$ and $|y| = n$ (we will use this to make indexing easier). Because $w' = x\sigma y$ is accepted by the DFA D, there is a sequence of DFA states $r_0, \ldots, r_m, t_0, t_1, \ldots, t_n$ such that

(i) $r_0 = q_0$, the start state of D

(ii) $r_i = \delta(r_{i-1}, x_i)$ for $1 \le i \le m$,
$t_0 = \delta(r_m, \sigma)$,
$t_i = \delta(t_{i-1}, y_i)$ for $1 \le i \le n$

(iii) $t_n \in F$ is an accept state in D

It follows that the sequence of NFA states $(r_0,0), \ldots, (r_m,0), (t_0,1), (t_1,1), \ldots, (t_n,1)$ form a valid path of computation for the NFA N on input $w = xy$. That is, using the conditions above satisfied by the sequence of DFA states, we can infer for the sequence of NFA states that:

(i) $(r_0,0) = (q_0,0)$, the start state of N

(ii) $(r_i,0) \in \delta'((r_{i-1},0), x_i)$ for $1 \le i \le m$
$(t_0,1) \in \delta'((r_m,0), \epsilon)$ because $t_0 = \delta(r_m, \sigma)$
$(t_i,1) \in \delta'((t_{i-1},1), y_i)$ for $1 \le i \le n$

(iii) $(t_n,1) \in F \times \{1\}$ is an accept state

Therefore $w = xy \in$ Leave-1-Out(L) is accepted by the NFA N.

(2) $w \in$ Leave-1-Out$(L) \impliedby w$ is accepted by the NFA

Now consider any $w \in \{0,1\}^*$ that is accepted by the NFA N. Then there is a path of computation through the sequence of NFA states $(r_0,0), \ldots, (r_m,0), (t_0,1), (t_1,1), \ldots, (t_n,1)$ such that, if $w = xy$ where x is the first m characters of w and y is the remaining n,

(i) $(r_0,0) = (q_0,0)$, the start state of the NFA N

(ii) $(r_i, 0) \in \delta'((r_{i-1}, 0), x_i)$ for $1 \le i \le m$
$(t_0, 1) \in \delta'((r_m, 0), \epsilon)$
$(t_i, 1) \in \delta'((t_{i-1}, 1), y_i)$ for $1 \le i \le n$

(iii) $(t_n, 1) \in F \times \{1\}$ is an accept state for N

Note that we are able to assume that a sequence of NFA states ending in accept follows such a format because: (1) the second component of the accept states in the NFA must be 1 (one symbol has been added), (2) we start in a state where the second component is 0 (one symbol has not been added), and (3) we can only change the second component during the first time we take an ϵ transition (one symbol is currently being added).

Using the definition of the NFA transition function δ' along with the properties satisfied by the sequence of NFA states, we can infer that the sequence of DFA states $r_0, \ldots, r_m, t_0, t_1, \ldots, t_n$ satisfy the following conditions:

(a) $r_0 = q_0$, the start state of the DFA D

(b) $r_i = \delta(r_{i-1}, x_i)$ for $1 \le i \le m$
$t_0 = \delta(r_m, \sigma)$ for some $\sigma \in \Sigma$ (this comes from the definition of δ')
$t_i = \delta(t_{i-1}, y_i)$ for $1 \le i \le n$

(c) $t_n \in F$ is an accept state for D

This tells us that $x\sigma y \in L$ for some symbol $\sigma \in \Sigma$, which implies that $w = xy \in \text{Leave-1-Out}(L)$.

We have constructed an NFA that accepts exactly the strings in Leave-1-Out(L), and thus the language is regular.

1.4 Regular Expressions (12 points)

Prove of disprove the following for regular expressions r, s and t.

(a) $s(rs \cup s)^* = s(rr^*s)^*$

(b) $(s^*r^* \cup rs) = (r \cup s^*)(s \cup r^*)$

(a) This is false. Let $r = a$, $s = b$, and $t = c$, where $a, b, c \in \Sigma$. Then, the string bb is accepted by the left hand side and rejected by the right hand side.

(b) This is true. First, we will show that any string that is accepted by the left hand side must also be accepted by the right hand side. There are two cases. In the first case, suppose the string x is accepted by s^*r^*. Then, the string can be broken into $x_1 \circ x_2$ where x_1 is accepted by s^* and x_2 is accepted by r^*. This means that x_1 will be accepted by $r \cup s^*$ and x_2 will be accepted by $s \cup r^*$, so x will be accepted by the right hand side.

In the second case, suppose x is accepted by rs. Then the string can be broken into $x_1 \circ x_2$ where x_1 is accepted by r and x_2 is accepted by s. This means that x_1 will be accepted by $r \cup s^*$ and x_2 will be accepted by $s \cup r^*$, so x will be accepted by the right hand side.

Second, we will show that any string that is accepted by the right hand side must also be accepted by the left hand side. Suppose x is accepted by the right hand side. Then, $x = x_1 \circ x_2$, where x_1 is accepted by $r \cup s^*$ and x_2 is accepted by $s \cup r^*$. There are now four cases:

Case 1: x_1 is accepted by r and x_2 is accepted by s. In this case, x is accepted by rs, so it is accepted by the left hand side.

Case 2: x_1 is accepted by r and x_2 is accepted by r^*. In this case, x is accepted by r^*, so it is accepted by the left hand side because $L(r^*) \subset L(s^*r^*)$.

Case 3: x_1 is accepted by s^* and x_2 is accepted by s. In this case, x is accepted by s^*, so it is accepted by the left hand side because $L(s^*) \subset L(s^*r^*)$.

Case 3: x_1 is accepted by s^* and x_2 is accepted by r^*. In this case, x is accepted by s^*r^*, so it is accepted by the left hand side.

An alternative argument can be made more algebraically. The proof sketch/idea is this: the RHS can be rewritten (by "multiplying it out") as $(rs \cup rr^* \cup ss^* \cup s^*r^*)$. Now this simplifies to just $(rs \cup s^*r^*)$ since both rr^* and ss^* are subsumed by r^*s^*.

2 Bonus Problem

A co-NFA is like an NFA, except it accepts an input $w \in \Sigma^*$ if and only if every possible state it could end up in when reading w (if the NFA does not abort) is an accept state. If a language L is recognized by a co-NFA, is L always regular?

Yes, any language recognized by a co-NFA is regular.

Proof. Let C_{NFA} be the NFA associated with co-NFA C (note that these two automata recognize different languages). Then the co-NFA C accepts $w \in \Sigma^*$ if and only if the following statement is true:

> "For each branch of computation, if the NFA C_{NFA} does not abort in the branch, then the final state is an accept state of C_{NFA}."

This is equivalent to saying that the co-NFA C rejects (does not accept) $w \notin \Sigma^*$ if and only if:

> "There exists a branch of computation where the NFA does not abort in the branch and the final state is a reject state."

We observe that this rejection criteria of the co-NFA is similar to the acceptance criteria of a normal NFA, which accepts if and only if there exists a branch of computation where the NFA does not abort in the branch and the final state is an accept state. The difference is that the accepts and rejects were switched.

Therefore we will prove that if L is recognized by a co-NFA C, then its complement $\overline{L} = \Sigma^* \setminus L$ is recognized by some NFA. This will imply that L is regular, and by the closure of regular languages under complement, it will also prove that L is regular.

Let the language L be recognized by the co-NFA C which runs the NFA $C_{NFA} = (Q, \Sigma, \delta, q_0, F)$. Now define the NFA N as $N = (Q, \Sigma, \delta, q_0, Q \setminus F)$. This new NFA switches the accept and reject states of the NFA C_{NFA}, but otherwise keeps the same states and transitions. We will show this NFA N recognizes the complement $\overline{L}$.

If $x \in \overline{L}$, then $x \notin L$. This implies that the co-NFA C rejects x. Therefore some branch of computation of C_{NFA} on x does not abort and ends in a reject state of C_{NFA}, which is some state not in F. Note

though that because the NFA's C_{NFA} and N share the same states and transitions, they must also share the same possible branches of computation (sequence of states to transition through) on input x. Therefore some branch of computation of N on x does not abort and ends in some state not in F. This means the branch of computation ends in a final state in $Q \backslash F$, so the NFA N will accept x. Thus N accepts $x \in \overline{L}$.

If $x \notin \overline{L}$, then $x \in L$. This implies that the co-NFA C accepts x. Therefore for each branch of computation of the NFA C_{NFA} on x, if the branch does not abort, then the final state is a accept state in C_{NFA}. In other words, every non-aborted branch of C_{NFA} on input x will end in a final state in F. However, we observed earlier that C_{NFA} and N have the same possible branches of computation. It follows that every non-aborted branch for N on x will end in a state in F, which is a reject state for N. Thus N rejects $x \notin \overline{L}$. (Note this includes the $x \in L$ where every branch in C_{NFA} aborts. In these cases, every branch of N's computations will also abort, so x will be rejected by N as it should.

We have shown that the NFA N recognizes $\overline{L}$, and so we can conclude from our earlier reasoning that L recognized by the co-NFA is regular. □

Alternative solution

Proof. We will prove that any language recognized by a co-NFA is regular by constructing a DFA for it.

Assume that the co-NFA for an NFA $N = (Q, \Sigma, \delta, q_0, F)$ recognizes L. Then $w \in L$ if and only if, when the co-NFA simulates N on w, it only has (non-aborting) accepting and aborting paths of computation. This also implies $w \notin L$ if and only if, when the co-NFA simulates N on w, that w has at least one non-aborting computation path that ends in reject.

We will construct a DFA using the NFA to DFA conversion where we us the power set of NFA states as the set of DFA states. Here, we keep track of the set of states that all possible branches of the NFA computation could be in as we read in each symbol of the input. Let the DFA 5-tuple be defined exactly how we did in class for the NFA to DFA conversion, except we will define the accept states differently. (Note that if we assigned the accept states the way we did in class, then the DFA would recognizes the language of the NFA N, rather than the language of the co-NFA that simulates N.)

Before assigning the accept states, we first observe that the final set of NFA states (after reading the entire input) can fall into the following categories:

1. The NFA aborted on every branch of computation. Then the final set of states is empty. The co-NFA accepts in this case, so the DFA should too.

2. The NFA did not abort on some branch of computation.

    i. In every non-aborted branch of computation, the NFA accepted. Here the final set of states is nonempty and contains only accept states of the NFA. The co-NFA accepts the string in this case, so the DFA should too.

    ii. In some non-aborted branch of computation, the NFA did not accept. Here, the final set of states contains at least one reject state. The co-NFA rejects this input, so the DFA should too.

Therefore we will assign the DFA accept states as the sets of NFA states containing only accept states (this includes the empty set). In other words, the set of DFA accept states is the power set $\mathcal{P}(F)$ of NFA accept states F.

We know from class that the state the DFA D ends in on input w will be exactly the set of states that the NFA N could end in on input w (which is empty if the NFA aborts everywhere). Because of our

definition of the DFA accept states, the DFA D accepts $w \in \Sigma^*$ if and only if all non-aborted branches of the NFA N run on w end in an accept state in the NFA. This is true if and only if w is accepted by the co-NFA that simulates N. Thus D recognizes the same language L as the co-NFA, and so L is regular. □