CS/ECE 374 ✧ Spring 2017

Homework 2

Due Wednesday, February 8, 2017 at 10am

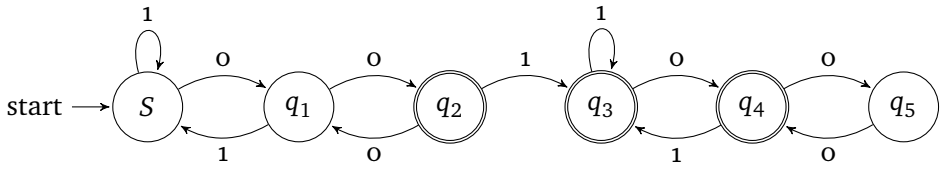
Groups of up to three people can submit joint solutions. Each problem should be submitted by exactly one person, and the beginning of the homework should clearly state the Gradescope names and email addresses of each group member. In addition, whoever submits the homework must tell Gradescope who their other group members are.

The following unnumbered problems are not for submission or grading. No solutions for them will be provided but you can discuss them on Piazza.

- Suppose $N_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$ and $N_2 = (Q_2, \Sigma, \delta_2, s_2, A_2)$ are NFAs. Formally describe a DFA that accepts the language $L(N_1) \setminus L(N_2)$. This combines subset construction and product construction to give you practice with formalism.
- Suppose $M = (Q, \Sigma, \delta, s, A)$ is a DFA. For states $p, q \in Q$ (p can be same as q) argue that $L_{p,q} = \{w \mid \delta^*(p, w) = q\}$ is regular. Recall that $\text{PREFIX}(L) = \{w \mid wx \in L, x \in \Sigma^*\}$ is the set of all prefixes of strings in L. Express $\text{PREFIX}(L(M))$ as $\cup_{q \in Z} L_{s,q}$ for a suitable set of states $Z \subseteq Q$. Why does this prove that $\text{PREFIX}(L(M))$ is regular whenever L is regular?
- For a language L let $\text{MID}(L) = \{w \mid xwy \in L, x, y \in \Sigma^*\}$. Prove that $\text{MID}(L)$ is regular if L is regular.

1. (a) Draw an NFA that accepts the language $\{w \mid$ there is exactly one block of 0s of even length$\}$. (A "block of 0s" is a maximal substring of 0s.)

Solution: One possible solution is



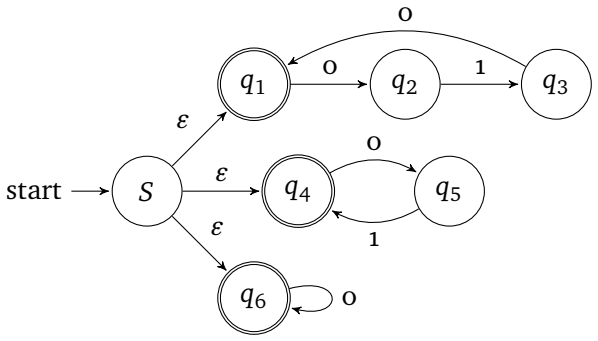
Explanation:

- Before encountering an even block of 0s, the automaton is in state S when it has just encountered a 1, q_1 when it has just encountered an odd number of contiguous 0s, and q_2 when it has just encountered an even number of contiguous 0s. If the automation encounters a 1 after an odd number of contiguous 0s, we return to S. Thus, state S and q_1 are not accepting states as they represent the case where there are no blocks of 0s of even length.
- After encountering a contiguous even block of 0s, the automaton is in state q_3 when it has just encountered a 1. q_4 when it has just encountered an odd number of contiguous 0s, and q_5 when it has just encountered an even number of contiguous 0s. Thus, q_5 is not an accepting state as it represents the case where there are more than one contiguous block of 0s of even length.

■

(b) i. Draw an NFA for the regular expression $(010)^* + (01)^* + 0^*$.

Solution: One possible solution is



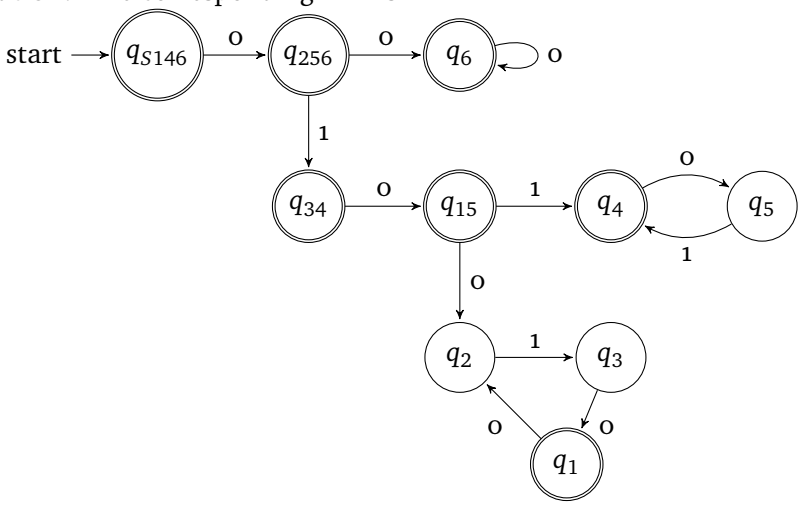
Explanation:

- We can break down our NFA into three parts, each representing a section of the regular expression. The NFA containing states q_1, q_2, and q_3 describes the regular expression $(010)^*$, q_4 and q_5 describes $(01)^*$, and q_6 describes 0^*.
- Since we would like our NFA to represent the union of these three regular expressions, we have ε transitions from the start state, S, to q_1, q_4, and q_6. Thus, if and only if a string is an element of either $(010)^*$, $(01)^*$, or $(0)^*$, will it be accepted by the corresponding part of the NFA and therefore accepted by the overall NFA.

■

ii. Now using the powerset construction (also called the subset construcion), design a DFA for the same language. Label the states of your DFA with names that are sets of states of your NFA.

Solution: The corresponding DFA is



To avoid cluttering the diagram, we have drawn neither the state $q_\emptyset$ representing the empty set of states of the NFA nor the transition arrows that point to it. The DFA should have all of the undrawn arrows point to a state $q_\emptyset$, which is not an accepting state. ■

Rubric: 10 points

- 4 points for part (a)
  - -1 for an incorrect edge or other minor error
  - -2 for too few states
  - 2 points overall for having the correct idea
- 2 points for (b.i)
  - -1 for first error
  - -.5 for next two errors
- 4 points for part (b.ii)
  - -1 for an incorrect edge or other minor error
  - -2 for too few states
  - 2 points overall for having the correct idea

2. This problem is to illustrate proofs of (the many) closure properties of regular languages.

   (a) For a language L let FUNKY$(L) = \{w \mid w \in L$ but no proper prefix of w is in $L\}$. Prove that if L is regular then FUNKY(L) is also regular using the following technique. Let $M = (Q, \Sigma, \delta, s, A)$ be a DFA accepting L. Describe a NFA N in terms of M that accepts FUNKY(L). Explain the construction of your NFA.

   Solution: Let L be a regular language. Suppose $M = (Q, \Sigma, \delta, s, A)$ is a DFA that accepts this language. We will make an NFA M' from M by removing all the edges that go out of a final state. We claim that $L(M')$ is the desired language. To sketch the proof, we observe that if M upon input w has to cross one of the removed edges, then in some step other than the last, it has been on a final state. That means that a proper prefix of w is also accepted by M. Therefore, $w \notin$ FUNKY(L). Moreover, if M never uses any of the removed edges upon input w and lands on a final state, then M' also takes the same path and lands on the same final state, and therefore accepts w. Thus, M' accepts all the strings in $L(M)$, except those that have a proper prefix in $L(M)$. In what follows, we state this proof formally. For that, we need the following lemmas.

   Lemma 1. *Let $M = (Q', \Sigma', \delta', s', A')$ be an NFA. Let $w, x \in \Sigma'^*$ be such that $w = xa$ where $a \in \Sigma$. Let q be an arbitrary state in Q'. Then*

$$\delta'^*(q, w) = \bigcup_{p \in \delta'^*(q,x)} \bigcup_{r \in \delta'(p,a)} \epsilon reach(r)$$

   Proof: Prove by induction on $|x|$. □

   Lemma 2. *Let $M = (Q', \Sigma', \delta', s', A')$ be an NFA. Let $w, x, y \in \Sigma'^*$ be such that $w = xy$. Let q be an arbitrary state in Q'. Then*

$$\delta'^*(q, w) = \bigcup_{p \in \delta'^*(q,x)} \delta'^*(p, y)$$

   Proof: Prove by induction on $|x|$ and application of Lemma 1. □

   Define an NFA $M' = (Q, \Sigma, \delta', s, A)$ where δ' is defined as follows:

$$\delta'(q, a) = \begin{cases} \emptyset & q \in A \\ \{\delta(q, a)\} & q \in Q \setminus A \end{cases}$$

   We will show that $L(M') =$ FUNKY(L). First, we prove that FUNKY$(L) \subseteq L(M')$. We make the following claim about the structure of M'.

   Claim 1. *Suppose that no proper prefix of w is accepted in M. Then $\delta'^*(s, w) = \{\delta^*(s, w)\}$.*

   Proof: We prove this claim by induction on length of w. If $|w| = 0$, then $w = \epsilon$. Therefore, $\delta^*(s, w) = \delta^*(s, \epsilon) = s$. By definition, $\delta'^*(s, w) = \delta'^*(s, \epsilon) = \epsilon$reach$(s) = \{s\}$. Hence, $\delta'^*(s, w) = \{\delta^*(s, w)\}$.

   Suppose $|w| > 0$ and that for any string z such that $|z| < |w|$ and no proper prefix of z is accepted by M, we have $\delta'^*(s, z) = \{\delta^*(s, z)\}$. Suppose $w = xa$ where $x \in \Sigma^*$ and $a \in \Sigma$. Since every proper prefix of x is also a proper prefix of w, thus no proper prefix

of x is accepted by M. Therefore, by induction hypothesis, $\delta'^*(s,x) = \{\delta^*(s,x)\}$. By Lemma 1,

$$\delta'^*(s,w) = \bigcup_{q \in \delta'^*(s,x)} \bigcup_{p \in \delta'(q,a)} \epsilon\text{reach}(p) = \bigcup_{q \in \delta'^*(s,x)} \delta'(q,a) = \delta'(\delta^*(s,x),a)$$

Since x is not accepted by M, the state $\delta^*(s,x)$ is not in A. Hence, $\delta'(\delta^*(s,x),a) = \{\delta(\delta^*(s,x),a)\} = \{\delta^*(s,w)\}$. Therefore, by induction, $\delta'^*(s,w) = \{\delta^*(s,w)\}$ for any string w. □

Let w be a string that is in L but no proper prefix of it is in L. By Claim 1, we know that $\delta'^*(s,w) = \{\delta^*(s,w)\}$. Since w is in L, we know that $\delta^*(s,w) \in A$. Therefore, $\delta'^*(s,w) \cap A \neq \emptyset$. Thus, M' accepts w. Since w was chosen arbitrarily from FUNKY(L), we conclude that FUNKY(L) $\subseteq L(M')$.

Next, we show that $L(M') \subseteq$ FUNKY(L). We make the following claim.

Claim 2. *If $q \in \delta'^*(s,w)$, then $\delta^*(s,w) = q$.*

Proof: We prove this by induction on $|w|$. If $|w| = 0$, then w is the empty string and therefore $q \in \delta'^*(s,w)$ is true only for $q \in \epsilon\text{reach}(s)$. The set $\epsilon\text{reach}(s)$ contains only s, by definition. Also, by definition of δ^*, we have $\delta^*(s,\epsilon) = s$. Therefore the claim is true for $|w| = 0$.

Suppose $|w| > 0$ and that for every string $z \in \Sigma^*$, where $|z| < |w|$ and $q \in \delta'^*(s,z)$ we have $\delta^*(s,z) = q$. Suppose $w = xa$ where $x \in \Sigma^*$ and $a \in \Sigma$. By definition, $q \in \delta'^*(s,w) = \bigcup_{p \in \delta'^*(s,x)} \bigcup_{r \in \delta'(p,a)} \epsilon\text{reach}(r) = \bigcup_{p \in \delta'^*(s,x)} \delta'(p,a)$. Thus, there exists $p \in \delta'^*(s,x)$, such that $q \in \delta'(p,a)$. Since $p \in \delta'^*(s,x)$, and by induction hypothesis, $\delta^*(s,x) = p$. By definition, $\delta^*(s,w) = \delta(\delta^*(s,x),a) = \delta(p,a)$. Since $\delta'(p,a)$ is non-empty and contains q, by definition of δ', we have $\delta'(p,a) = \{\delta(p,a)\}$ and $\delta(p,a) = q$. Therefore, $\delta^*(s,w) = q$. □

Let w' be an arbitrary string of $L(M')$. That means $q \in \delta'^*(s,w)$ for some $q \in A$. By Claim 2, we have $\delta^*(s,w) = q \in A$. Therefore, $w \in L(M)$. Thus, $L(M') \subseteq L(M)$. It remains to prove that no proper prefix of the strings in $L(M')$ are in $L(M)$.

Suppose $w \in L(M')$ and x is a proper prefix of w. Let $w = xay$ where $a \in \Sigma$ and $y \in \Sigma^*$. Suppose towards contradiction that $x \in L(M)$. Since $w \in L(M')$, we have $\delta'^*(s,w) \cap A$ is non-empty and in particular $\delta'^*(s,w)$ is non-empty. By Lemma 2, we have $\delta'^*(s,w) = \bigcup_{p \in \delta'^*(s,x)} \delta'^*(p,ay)$. Therefore, $\delta'^*(s,x)$ is non-empty. Suppose $p \in \delta'^*(s,x)$. Then, by Claim 2, $\delta^*(s,x) = p$. Since, x is accepted by M', the state p must be in A. Since, $r \in \delta'^*(s,x)$ would imply $\delta^*(s,x) = r$, we have $\delta'^*(s,x) = \{p\}$. Therefore, we have $\delta'^*(s,w) = \delta'^*(p,ay) = \bigcup_{r \in \delta'(p,a)} \bigcup_{s \in \epsilon\text{reach}(r)} \delta'^*(s,y)$. But since $p \in A$, by definition of δ', we have $\delta'(p,a) = \emptyset$. Therefore, $\delta'^*(s,w) = \emptyset$. This contradicts the fact that w is accepted by M'. This contradiction shows that no proper prefix of strings in $L(M')$ are accepted by M. This completes the proof.

Rubric: 10 points

- 8 points for a correct construction.
  - -1 for minor errors.
- 1 points for justifying that $L(M') \subseteq$ FUNKY(L).
- 1 points for justifying that FUNKY $\subseteq L(M')(L)$.

Solution (A ninja's solution): Let r be a regular expression for L. Define r' as $r(0+1)(0+1)^*$. The $L(r')$ is the set of strings that have a proper prefix in L. Therefore, $L \setminus L(r')$ is the set of strings in L that have no proper prefix in L. Since regular languages are closed under set difference, this language is regular. ■

(b) In Lab 3 we saw that $\mathit{insert}\mathtt{1}(L)$ is regular whenever L is regular. Here we consider a different proof technique. Let r be a regular expression. We would like to show that there is another regular expression r' such that $L(r') = \mathit{insert}\mathtt{1}(L(r))$.

i. For each of the base cases of regular expressions $\emptyset, \epsilon$ and $\{a\}, a \in \Sigma$ describe a regular expression for $\mathit{insert}\mathtt{1}(L(r))$.

ii. Suppose r_1 and r_2 are regular expressions, and r_1' and r_2' are regular expressions for the languages $\mathit{insert}\mathtt{1}(L(r_1))$ and $\mathit{insert}\mathtt{1}(L(r_2))$ respectively. Describe a regular expression for the language $\mathit{insert}\mathtt{1}(L(r_1 + r_2))$ using r_1, r_2, r_1', r_2'.

iii. Same as the previous part but now consider $L(r_1 r_2)$.

iv. Same as the previous part but now consider $L((r_1)^*)$.

Solution:

- Suppose r is empty. That means r represents the empty language. Inserting a $\mathtt{1}$ in every string of $\emptyset$ again results in $\emptyset$. This language is represented with an empty regular expression.
- If $r = \epsilon$ then $r' = \mathtt{1}$. Observe that there is only one option for inserting a $\mathtt{1}$ in ϵ and that makes the string $\mathtt{1}$ which is can be represented as the regular expression $\mathtt{1}$.
- If $r = \mathtt{a}$ where $\mathtt{a}$ is a letter of alphabet, then $r' = \mathtt{1a} + \mathtt{a1}$. Observe that there are two option for inserting a $\mathtt{1}$ in $\mathtt{a}$. That is either before or after the letter $\mathtt{a}$. That makes two strings $\mathtt{1a}$ and $\mathtt{a1}$ which can be represented as the regular expression $\mathtt{a1} + \mathtt{1a}$.
- Suppose $r = r_1 + r_2$ and r_1' and r_2' are regular expressions for $\mathit{insert}\mathtt{1}(L(r_1))$ and $\mathit{insert}\mathtt{1}(L(r_2))$, respectively. Then $r' = r_1' + r_2'$. For every string $w \in L(r)$, then $w \in L(r_1)$ or $w \in L(r_2)$. If we insert one $\mathtt{1}$ into w the new string is in $L(r_1')$ or $L(r_2')$. Therefore, $L(r_1' + r_2') \subseteq \mathit{insert}\mathtt{1}(L(r))$.
  Conversely, if $w' \in \mathit{insert}\mathtt{1}(L(r))$, then there exists $w \in L(r)$ such that w' is derived by inserting a $\mathtt{1}$ into w. Thus, either $w \in L(r_1)$ or $w \in L(r_2)$. If $w \in L(r_1)$, then $w' \in L(r_1')$ and if $w \in L(r_2)$, then $w' \in L(r_2')$. In either case, $w' \in L(r_1' + r_2')$. Hence $\mathit{insert}\mathtt{1}(L(r)) = L(r_1' + r_2')$.
- Suppose $r = r_1 r_2$. Then $r' = r_1' r_2 + r_1 r_2'$. Suppose $w \in L(r)$. That means $w = xy$ where $x \in L(r_1)$ and $y \in L(r_2)$. If we insert a $\mathtt{1}$ into w, it will be inserted into either x or y. If we insert it into x to obtain x', then $w = x'y$ where $x' \in L(r_1')$. Hence $w \in L(r_1' r_2)$. If we insert a $\mathtt{1}$ into y to obtain y', then $w' = xy'$ where $y' \in L(r_2')$. Thus $w' \in L(r_1 r_2')$. In either case $w \in L(r_1' r_2 + r_1 r_2')$. Hence, $\mathit{insert}\mathtt{1}(L) \subseteq L(r_1' r_2 + r_1 r_2')$.
  Conversely, suppose $w' \in L(r_1' r_2 + r_1 r_2')$. If $w' \in L(r_1' r_2)$, then $w' = x'y$ where $x' \in L(r_1')$ and $y \in L(r_2)$. Thus, there exists $x \in L(r_1)$ such that x' is obtained by inserting a $\mathtt{1}$ into x. Thus w' is obtained by inserting a $\mathtt{1}$ into w. Similarly, if $w' \in L(r_1 r_2')$, then w' is obtained by inserting a $\mathtt{1}$ into w. Hence, $L(r_1' r_2 + r_1 r_2') \subseteq \mathit{insert}\mathtt{1}(L)$.
- Suppose $r = r_1^*$. Then $r' = r_1^* r_1' r_1^* + \mathtt{1}$. Let $w \in L(r)$. If $w = \epsilon$, then inserting a $\mathtt{1}$ into w gives the string $\mathtt{1}$. If $w \in L(r_1 r_1^*)$, then there are strings $x_1, \ldots, x_k \in L(r_1)$

such that $w = x_1 x_2 \cdots x_k$. If we insert one 1 into w, it will be inserted into one of x_i's. Then, $w' = x_1 \cdots x_{i-1} x'_i x_{i+1} \cdots x_k$. Observe that $x_1 \cdots x_{i-1} \in L(r_1^*)$, $x'_i \in L(r'_1)$ and $x_{i+1} \cdots x_k \in L(r_1^*)$. Thus $w' \in L(r_1^* r'_1 r_1^*)$. Thus, $insert1(L) \subseteq L(r_1^* r'_1 r_1^* + 1)$.
Conversely, suppose $w' \in L(r_1^* r'_1 r_1^* + 1)$. If $w' = 1$, then w' can be obtained from ϵ by inserting a 1 into it. Observe that $\epsilon \in L(r_1^*)$. Thus $w' \in insert1(L)$. If $w' \in L(r_1^* r'_1 r_1^*)$, then there are strings $w = xy'z$ such that $x, z \in L(r_1^*)$ and $y' \in L(r'_1)$. Hence, there exists a string $y \in L(r_1)$ such that y' can be obtained from y by inserting a 1 into it. Therefore, w' can be obtained from $w = xyz$ by inserting a 1 into it. Observe that $xyz \in L(r_1^* r_1 r_1^*) \subseteq L(r_1^*)$. Thus $w' \in insert1(L)$.

■

Rubric: 10 points:

- 1 point for each of the three base cases.
- 2 points for union and concatenation.
- 3 points for Kleene star.

3. Recall that for any language L, $\overline{L} = \Sigma^* - L$ is the complement of L. In particular, for any NFA N, $\overline{L(N)}$ is the complement of $L(N)$.

   Let $N = (Q, \Sigma, \delta, s, A)$ be an NFA, and define the NFA $N_{\text{comp}} = (Q, \Sigma, \delta, s, Q \setminus A)$. In other words we simply complemented the accepting states of N to obtain N_{comp}. Note that if M is DFA then M_{comp} accepts $\Sigma^* - L(M)$. However things are trickier with NFAs.

   (a) Describe a concrete example of a machine N to show that $L(N_{\text{comp}}) \neq \overline{L(N)}$. You need to explain for your machine N what $\overline{L(N)}$ and $L(N_{\text{comp}})$ are.

   (b) Define an NFA that accepts $\overline{L(N)} - L(N_{\text{comp}})$, and explain how it works.

   (c) Define an NFA that accepts $L(N_{\text{comp}}) - \overline{L(N)}$, and explain how it works.

   Hint: For all three parts it is useful to classify strings in Σ^* based on whether N takes them to accepting and non-accepting states from s.

Solution: We first notice that for an NFA N, every string w may be classified into one of four types:

Type 1 w leads to no state in N. That is, every computation on w crashes.

Type 2 w leads to at least one accept state in N, but no non-accepting states.

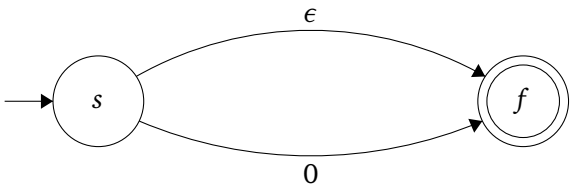
Type 3 w leads to at least one non-accepting state in N, but no accepting states.

Type 4 w leads to at least one accepting state in N and also leads to at least one non-accepting state of M.

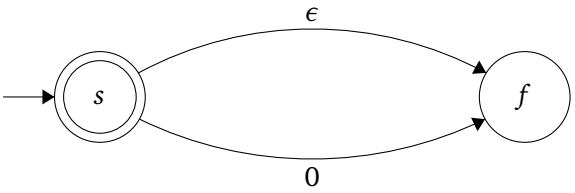
We now enunciate strings that are in $L(N)$ (i.e., accepted by N), $\overline{L(N)}$, and $L(N_{comp})$. Using our aforementioned classification:

- $L(N)$ contains strings of type 2 and 4, since M contains an accepting path for w in these cases.
- $\overline{L(N)}$ contains strings of type 1 and 3, since by definition, $\overline{L(N)} = \Sigma^* - L(N)$
- $L(N_{comp})$ contains strings of type 3 and 4, since if N contains a path for w to a non-accepting state, it will be turned into an accepting state in $\overline{N}$).

(a) We describe a language L represented by an NFA N ($L = L(N)$) such that swapping the accept and non-accept states in N yields an NFA N_{comp} that does NOT recognize $\overline{L(N)}$. The example is the following: consider the language L defined over $\Sigma = \{0\}$. Let N be an NFA such that $L(N) = L$ and is defined as follows:



Switching the accept and non accepting states in this NFA gives us N_{comp} as follows:



L(N) accepts $\epsilon \implies \overline{L(N)}$ rejects ϵ. However, $L(N_{comp})$ accepts the empty string ϵ and thus for this machine N, $\overline{L(N)} \neq L(N_{comp})$

(b) The set difference $\overline{L(N)} - L(N_{comp})$ contains strings of type 1 only. Hence, we want to accept all and only strings that lead nowhere in N. To do this, we build the powerset DFA M' that simulates N, but make the accepting state the one labeled with the empty set. More formally, we define the automaton $M' = (Q', \Sigma, \delta', s', A')$ by

- $Q' = P(Q)$.
- $\Sigma = \Sigma$.
- $\delta'(B, a) = \{\delta(q, a) : q \in B\}$.
- $s' = \{s\}$.
- $A' = \{\emptyset\}$.

Note that this is a DFA, but all DFAs are just NFAs, so it doesn't matter.

One plausible but incorrect approach is to modify N to obtain M' as follows: make all states non-accepting, and add all edges missing in N to point to a new accepting state in M'. While this does let previously crashed computations in N be accepted in M', such strings might already have been accepted in N via another path, and so should be rejected in M'.

(c) The set difference $L(N_{comp}) - \overline{L(N)}$ contains strings of type 4 only, so we can construct a DFA M'' that accepts exactly these strings by creating the powerset construction DFA M'', but a state (set of states belonging to N) C of the M'' is an accepting state if and only if it contains both an accepting and a non-accepting state of N. More formally, we define the automaton $M'' = (Q'', \Sigma, \delta'', s'', A'')$ by

- $Q'' = P(Q)$.
- $\Sigma = \Sigma$.
- $\delta''(B, a) = \{\delta(q, a) : q \in B\}$.
- $s'' = \{s\}$.
- $A'' = \{C : \exists\ p \in C \cap A \text{ and } \exists\ q \in C \cap (Q - A)\}$.

Alternate solution: Parts (b) and (c) can also be computed as follows: For any two arbitrary regular languages A and B, there exist NFA's N_A and N_B respectively that accept these languages. The DFAs corresponding to these NFA's M_A and M_B(say) can be constructed using subset construction. Now A-B can be defined by a new DFA M_{A-B} by following the rules of product construction on M_A and M_B. This procedure can be followed for both parts where A = $\overline{L(N)}$ and B = $L(N_{comp})$ in part (a) and vice versa for part (b).

Rubric: On an exam, on a 10-point scale, we would allocate: 2 points for correctly stating the answer to part (a), 4 points for each of parts (b) and (c) - +2 if you have the main idea, and the remaining +2 if you've correctly/accurately specified the automaton.

■

Solved problem

4. Let L be an arbitrary regular language. Prove that the language $\mathit{half}(L) := \{w \mid ww \in L\}$ is also regular.

Solution: Let $M = (\Sigma, Q, s, A, \delta)$ be an arbitrary DFA that accepts L. We define a new NFA $M' = (\Sigma, Q', s', A', \delta')$ with ε-transitions that accepts $\mathit{half}(L)$, as follows:

$$\begin{aligned} Q' &= (Q \times Q \times Q) \cup \{s'\} \\ s' &\text{ is an explicit state in } Q' \\ A' &= \{(h,h,q) \mid h \in Q \text{ and } q \in A\} \\ \delta'(s', \varepsilon) &= \{(s,h,h) \mid h \in Q\} \\ \delta'((p,h,q),a) &= \{(\delta(p,a),h,\delta(q,a))\} \end{aligned}$$

M' reads its input string w and simulates M reading the input string ww. Specifically, M' simultaneously simulates two copies of M, one reading the left half of ww starting at the usual start state s, and the other reading the right half of ww starting at some intermediate state h.

- The new start state s' non-deterministically guesses the "halfway" state $h = \delta^*(s,w)$ without reading any input; this is the only non-determinism in M'.
- State (p,h,q) means the following:
  - The left copy of M (which started at state s) is now in state p.
  - The initial guess for the halfway state is h.
  - The right copy of M (which started at state h) is now in state q.
- M' accepts if and only if the left copy of M ends at state h (so the initial non-deterministic guess $h = \delta^*(s,w)$ was correct) and the right copy of M ends in an accepting state.

■

Rubric: 5 points =
- \+ 1 for a formal, complete, and unambiguous description of a DFA or NFA
  - – No points for the rest of the problem if this is missing.
- \+ 3 for a correct NFA
  - – −1 for a single mistake in the description (for example a typo)
- \+ 1 for a *brief* English justification. We explicitly do *not* want a formal proof of correctness, but we do want one or two sentences explaining how the NFA works.