(1) Answers and brief explanations for the “HW3 Online Part” have been made visible on TopHat.

(2 and 3) Convert the following NFA $N$ (at upper left in the following diagram) into a DFA $M$ such that $L(M) = L(N)$ using the algorithm from the text and lecture notes. You must expressly use the algorithm and you must show the scratchwork used to derive your answer. Then (3) convert $N$ into a regular expression $r$ such that $L(N) = L(r)$ using the algorithm from class.

The following diagram includes much of the answers to both problems, with partial steps of the FA-to-regexp algorithm done “graphically” at upper right.

1
2
3
4

\[\text{Start} \quad 1 \quad 2 \quad 3 \quad 4\]

\[\epsilon \quad a \quad b \quad e \quad a + b\]

\[1 \quad 2 \quad 3 \quad 4\]

\[a, b, e, a + b, a + ab\]

\[1 \quad 2 \quad 3 \quad 4\]

\[a + bb \quad b + a \quad a + a(b + a) \quad a + aab\]

Then use your DFA to help answer the following four questions:

(a) Can $N$ process the string $baa$ to any one of its four states? Trace out a computation of the DFA showing yes or no. **Answer:** The DFA processes $baa$ to its set-state $q = \{1, 2, 3, 4\}$. This means that the NFA can process $baa$ to any of those states.

(b) Is there a string $y$ such that $N$ does not accept the string $baa \cdot y$? Again, demonstrate your answer using $M$ and traces. **Answer:** After processing $baa$ to $q$, the string $y = babb$ gives the further trace

\[(q, b, \{1, 2, 3\}, a, \{1, 3, 4\}, b, \{2, 3\}, b, \{1\})\]

The set-state $\{1\}$ is not accepting, so the answer is yes.
(c) Is there a string \( z \) that \( N \) cannot process? \textit{Answer:} No, because the DFA \( M \) does not have a dead state—at least, not a dead state that can be reached from its start state.

(d) If the start state of \( N \) had been state 3 not state 1, then would the DFA have a dead state? \textit{Answer:} Yes: The start state of the DFA would just be \([3]\), not \([2,3]\) since one does not follow \( \varepsilon \)-arrows backwards. And \( \delta([3],b) = \emptyset \) because \( N \) cannot process \( b \) from state 3. So that \( \emptyset \) would be a reachable dead state.

(In total this problem was worth \( 18 + 4 \times 3 = 30 \) pts.; adding 24 for (3) made 74 on the set)

Here is how the answers are obtained. For (2), we can optically calculate what lecture called the “\( \delta \)-underbar” function, but since LaTeX has no simple “underbar” macro we just write “\( \delta \)”: \[
\begin{array}{c|c}
\delta(1,a) &=& \{1\} \\
\delta(2,a) &=& \emptyset \\
\delta(3,a) &=& \{3,4,1\} \\
\delta(4,a) &=& \{2\}
\end{array} \quad \begin{array}{c|c}
\delta(1,b) &=& \{2,3\} \\
\delta(2,b) &=& \{1\} \\
\delta(3,b) &=& \emptyset \\
\delta(4,b) &=& \{3\}
\end{array}
\]

It does not matter what order one writes a set, so \([3,4,1]\) is the same as \([1,3,4]\). The order came because the \( \varepsilon \)-arc triggered a reminder that “whenever 4 then also 1”—writing this reminder is IMHO a good habit. It would have been OK to write \( \delta(2,a) = \{3,4,1\} \) via the leading \( \varepsilon \)-arc but there is no need—and IMHO doing so might have made one forget the more-important use of the other \( \varepsilon \)-arc to include state 1. Since there is no \( \varepsilon \)-arc \textit{out of} state 1, the start state of the DFA is just \([1]\). Expanding it to start a breadth-first search uncovers the DFA states as follows:

\[
\begin{align*}
\Delta([1],a) &= \delta(1,a) = \{1\}; \quad \text{(not a new state)} \\
\Delta([1],b) &= \delta(1,b) = \{2,3\}; \quad \text{(new state, must expand)} \\
\Delta([2,3],a) &= \delta(2,a) \cup \delta(3,a) = \emptyset \cup \{3,4,1\} = \{1,3,4\}; \quad \text{(new)} \\
\Delta([2,3],b) &= \delta(2,b) \cup \delta(3,b) = \{1\} \cup \emptyset = \{1\}; \\
\Delta([1,3,4],a) &= \{1\} \cup \{3,4,1\} \cup \{2\} = \{1,2,3,4\}; \quad \text{(new)} \\
\Delta([1,3,4],b) &= \{2,3\} \cup \emptyset \cup \{3\} = \{2,3\}; \\
\Delta([1,2,3,4],a) &= \{1,2,3,4\}; \quad \text{(note we came in on} \ a) \\
\Delta([1,2,3,4],b) &= \{2,3\} \cup \{1\} \cup \emptyset \cup \{3\} = \{1,2,3\}; \quad \text{(new, dang)} \\
\Delta([1,2,3],a) &= \{1\} \cup \emptyset \cup \{3,4,1\} = \{1,3,4\}; \quad \text{(again)} \\
\Delta([1,2,3],b) &= \{1,2,3\}; \quad \text{(itself means same-old, same-old, so done!)}
\end{align*}
\]

The final states are those that contain 3, so all but the start state \([1]\). Note that since there isn’t any state that includes 2 but not 3, it does not matter if you made 2 an accepting state in the NFA, which the \( \varepsilon \)-arc would have justified doing.

For problem (3), first note that if the arc from \([2,3]\) on \( b \) had gone to any of the other states rather than \([1]\), then the accepting states would have all formed a “nirvana cluster”
and could have been reduced to just one “nirvana state” yielding an easy machine and expression

\( r_0 = a^*b(a + b)^* \). This expression \( r_0 \) is comprehensive but it is not sound, because as we saw in part (2b) strings like \( bbaabb \) can go in quite deep and yet not be accepted. Nor is it correct to replace the triangle of accepting states by a big loop \((\{3, 4, 1\}, aa^*bb^*a, \{3, 4, 1\})\) because that messes up on computations that go into \{1, 2, 3, 4\} or \{1, 2, 3\} and stay there. It is hence IMHO better not to mess with the DFA even though there is an urge to consider DFAs as simpler. The NFA has just one accepting state.

Although the code said to re-number it 2 right away, in fact it is always OK to eliminate any higher-numbered non-accepting states first. This will diverge from the above diagram, which eliminated state 2 first because it has no outgoing arc to state 4 and hence adds no work to the next step. Here is the diagram for eliminating state 4 first, then swapping the labels 2 and 3 (if you care) before eliminating 3.

In terms of the code form of the algorithm, the original “T-matrix” for the NFA is:

\[
\begin{bmatrix}
a & b & \emptyset & \emptyset \\
b & \emptyset & \epsilon & \emptyset \\
\emptyset & \emptyset & a & a \\
\epsilon & a & b & \emptyset \\
\end{bmatrix}
\]

The three uses of \( \emptyset \) on the main diagonal in \( T(2, 2) \), \( T(3, 3) \), and \( T(4, 4) \) could equally well be \( \epsilon \), and \( T(1, 1) \) could even be \( a + \epsilon \). But the usages of \( \epsilon \) and \( \emptyset \) in the other entries have to be exactly as-is.

When eliminating state 4 we have \( \text{In : (3, a)} \) and \( \text{Out : [\epsilon, 1], [a, 2], [b, 3]} \). We therefore only need to update \( T(3, 1) \), \( T(3, 2) \), and \( T(3, 3) \). The lines of code executed are:

\[
T(3, 1) := T(3, 1) + T(3, 4) \cdot T(4, 4)^* \cdot T(4, 1) \\
= \emptyset + a \cdot (\emptyset)^* \cdot \epsilon = a \cdot \epsilon = a
\]

\[
T(3, 2) := T(3, 2) + T(3, 4) \cdot T(4, 4)^* \cdot T(4, 2) \\
= \emptyset + a \cdot \epsilon \cdot a = aa
\]

\[
T(3, 3) := T(3, 3) + T(3, 4) \cdot T(4, 4)^* \cdot T(4, 3) \\
= a + ab \quad \text{(don’t forget the a that was already there)}
\]

The new \( T \) matrix with 3 states here has just a changed last row and deleted fourth row and column. At right we swap the second and third rows and columns to effect the swap of the labels 2 and 3.
Now we have \( l_n : (1, b), (2, aa) \) and \( o_u : [b, 1], [c, 2] \). We therefore need to update all of \( T(1, 1) \), \( T(1, 2) \), \( T(2, 1) \), and \( T(2, 2) \). The one solace is that as before, the state we are bypassing and eliminating has no self-loop. Again I avoid writing subscripts “new” and “old” or \( T' \) and \( T'' \) etc.—we’re used to how variables get updated as we trace code execution. Here goes (again, we are using the right-hand matrix):

\[
T(1, 1) := T(1, 1) + T(1, 3) \cdot T(3, 3)^\star \cdot T(3, 1)
= a + b \cdot (\emptyset^\star) \cdot b = a + bb
\]

\[
T(1, 2) := T(1, 2) + T(1, 3) \cdot T(3, 3)^\star \cdot T(3, 2)
= \emptyset + b \cdot \epsilon \cdot \epsilon = b
\]

\[
T(2, 1) := T(2, 1) + T(2, 3) \cdot T(3, 3)^\star \cdot T(3, 1)
= a + aa \cdot b = a + aab
\]

\[
T(2, 2) := T(2, 2) + T(2, 3) \cdot T(3, 3)^\star \cdot T(3, 2)
= a + ab + aa
\]

In the final diagram, we want \( L_{1,2} \). Using the form \( L_{1,2} = L_{1,1}T(1, 2)T(2, 2)^\star \), we get:

\[
L_{1,2} = (T(1, 1) + T(1, 2)T(2, 2)^\star T(2, 1))'T(1, 2)T(2, 2)^\star, \quad \text{so}
\]

\[
L(N) = (a + bb + b(a + aa + ab)')(a + aab)^\star b(a + aa + ab)^\star.
\]