(1) This problem is “HW2 Online Part” on TopHat, worth 20 pts. as before. Here is the NFA for the first five short-answer questions on it, which will help cut down scrolling:

(2) Call the following two DFAs \( M_1 \) (the one with three states that looks like \( N \) above) and \( M_2 \). Use the Cartesian product construction to design DFAs \( M_3 \) and \( M_4 \) such that \( \text{L}(M_3) = \text{L}(M_1) \cap \text{L}(M_2) \) and \( \text{L}(M_4) = \text{L}(M_1) \triangle \text{L}(M_2) \), where \( \triangle \) is symmetric difference. (18 + 3 = 21 pts.)

Notice that the state \( B_2 \) is never entered when we start from \( A_1 \). Hence we do not need to include it in the DFA \( M_3 \). The reason simply stated is that \( M_1 \) can only go to state 2 on a \( b \) but \( M_2 \) can only go to state \( B \) on an \( a \), so there’s never a way to do both. For a similar reason the machine \( M_3 \) can never get back to \( A_1 \), but since it starts there, that state is included.

The machines \( M_1 \) has \( F_1 = \{ 1, 3 \} \) (\( s = 1 \) here) while \( M_2 \) has \( F_2 = \{ B \} \). So \( M_3 \) has \( F_3 = F_1 \times F_2 = \{ (1, B), (3, B) \} \). OK, because I stated \( M_1 \) to be the machine first seen going down the page but we usually put rows before columns, the diagram shows \( F_2 \times F_2 \) without the parentheses as \( \{ B_1, B_3 \} \). So long as a clear indication is given, one can be flexible with notation.

For \( M_4 \) the arcs and states are the same, except that now we want the XOR (exclusive or) of being a final state of \( M_1 \) vis-à-vis \( M_2 \). So \( B \) goes with 2 and \( A \) goes with 1 and 3 to give \( F_4 = \{ A_1, A_3, B_2 \} \)—but since the state \( B_2 \) is a “no-show” we just get \( F_4 = \{ A_1, A_3 \} \).

(3) For each of the following languages \( A \), write a regular expression \( r \) such that \( \text{L}(r) = A \), and then give an NFA \( N_r \) such that \( \text{L}(N_r) = A \). Well, if you give a DFA, that counts as an NFA, but in one or two cases you may find the NFA easier to build especially once you have \( r \). For part (b), note that a string can be broken uniquely into maximal “blocks” of consecutive letters. For instance, in “Tennessee” the blocks are \( T, e, nn, e \) again, \( ss \), and \( ee \).
(a) The language of strings over \{a, b\} in which every b is followed immediately by at least one a.

(b) The language of strings over \{a, b\} in which every a belongs to a “block” of at least 2 a’s and every b belongs to a block of at least 3 b’s.

(c) The language of strings over \{a, b\} with no block of 3 [added: “or more”] b’s, and in which every block of 2 b’s has an odd number of chars before it. (6 + 6 + 12 = 24 pts., for 65 total)

**Answer:**

(a) It is as if the “characters” are really a and ba, so \(r_a = (a + ba)^*\). Note that strings with no b’s should count because a condition beginning “for every b” defaults to true if the domain of b’s is empty.

(b) Now the “characters” are aaa and bbbb, so \(r_b = (aaa \cup bbbb)^*\).

(c) A little reasoning says that the characters come in twos except at the end you can have an optional final a or b. The second rule says that bb is a bad character. So \(r_c = (aa \cup ab \cup ba)^*(e \cup a \cup b)\). Note that you can get two consecutive bs when ab is followed by ba or by the allowed final b, but in both cases the previous number of chars is odd.

Here are the machines. The first version of \(N_b\) at right is suggested by the form of \(r_b\), but the second version is also easy to conceive. In fact it is deterministic and needs only to fill in a dead state with the arcs not shown going to it in order to be a formal DFA.

Extra: In the NFA \(N_c\) at lower left, it does not matter whether the states \(t_3\) and \(u_3\) are accepting—the state \(v_3\) always covers their cases. Without the “or more” in 3(c), blocks of 4 or more b’s are considered OK. What makes this “annoying” is that it now matters whether you came from \(t_3\) back to \(s_3\) on a or on b: with the latter, following up with bbaa or just bbb and end-of-string will be OK, but not if we came back on a. Thus we have to split off instructions \((t_3, a, s_3)\) and \((t_3, b, s'_3)\) and duplicate the other arcs from \(s'_3\). At this point it is easier just to build the DFA: the states you need are s, “1-b-even,” “1-b-odd,” “2-b’s-even,” “2-b’s-odd,” “3 b’s,” and “4-or-more b’s” with reference to the current block; all but “2-b’s-even” and “3 b’s” are accepting.