ECE 468
Problem Set 1: Regular expressions and finite automata

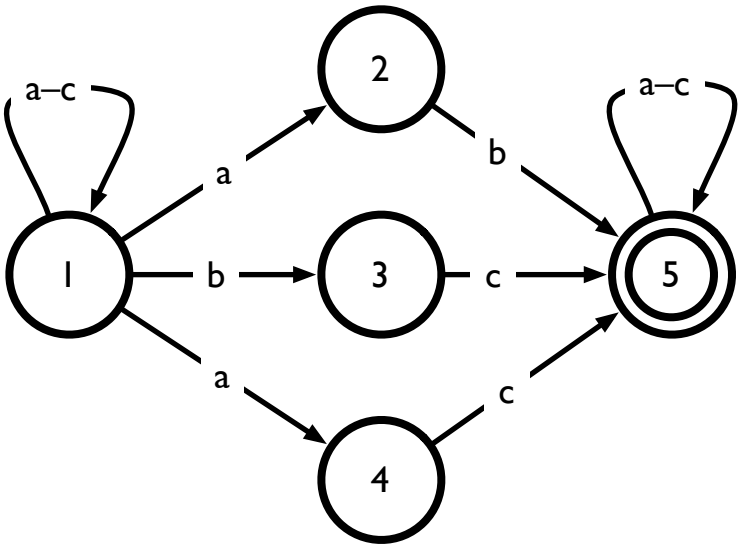
1. For strings containing the letters 'a', 'b', and 'c', give a regular expression that captures all strings that have at least two (different, consecutive) letters in alphabetical order.

   Answer: We can capture this by specifying the three possible substrings that should make our regular expression match: 'ab', 'bc' and 'ac'. We need *one* of those substrings to appear somewhere in the string we are trying to match:

   $$[a-c]^*((ab)|(bc)|(ac))[a-c]^*$$

2. Give a *non-deterministic* finite automaton that captures the regular expression from above. Show the automaton in graphical form.

   Answer: There are many possible non-deterministic automata that could capture the regular expression. Here's one possibility:



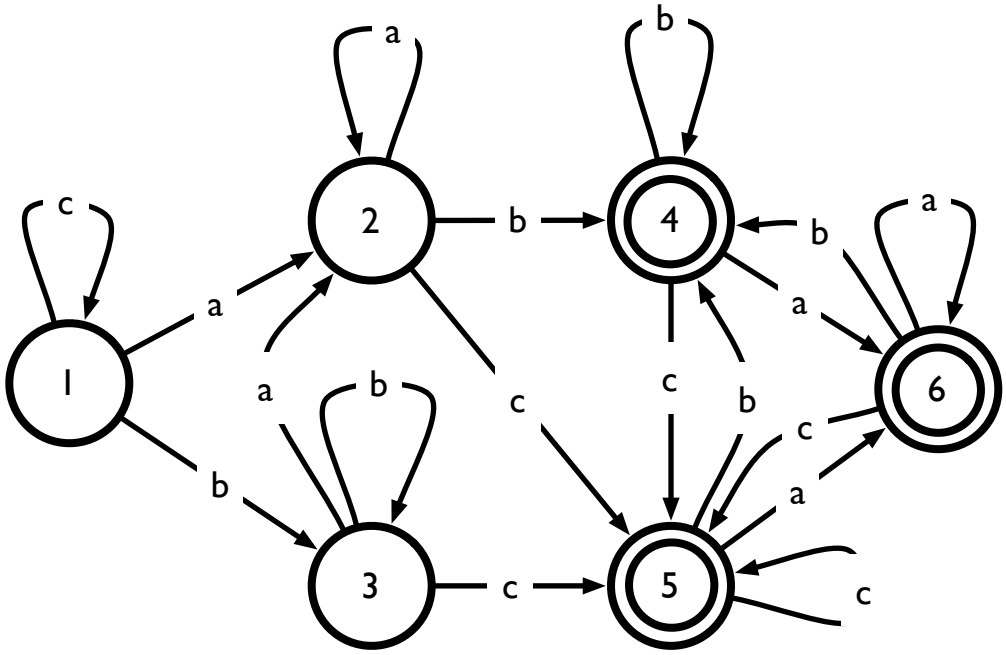
   Note that this automaton takes heavy advantage of non-determinism. State 1 lets it capture all characters, and then, at any point, it can guess whether an 'a' or 'b' should begin the alphabetical substring.

3. Using the construction described in class, give a *deterministic* version of the automaton. You only need to show the transition table.

Answer: Here's the transition table we get from applying the nfa-to-dfa construction from class. I've included an extra column at the end to let me rename the states (which will come in handy in the next question).

State	a	b	c	Final?	New name
1	1, 2, 4	1, 3	1	No	1
1, 2, 4	1, 2, 4	1, 3, 5	1, 5	No	2
1, 3	1, 2, 4	1, 3	1, 5	No	3
1, 3, 5	1, 2, 4, 5	1, 3, 5	1, 5	Yes	4
1, 5	1, 2, 4, 5	1, 3, 5	1, 5	Yes	5
1, 2, 4, 5	1, 2, 4, 5	1, 3, 5	1, 5	Yes	6

While you did not have to provide a graphical representation of this DFA, here it is:



4. Give a *reduced* version of the finite automaton, using the algorithm we used in class. You only need to show the state transition diagram.

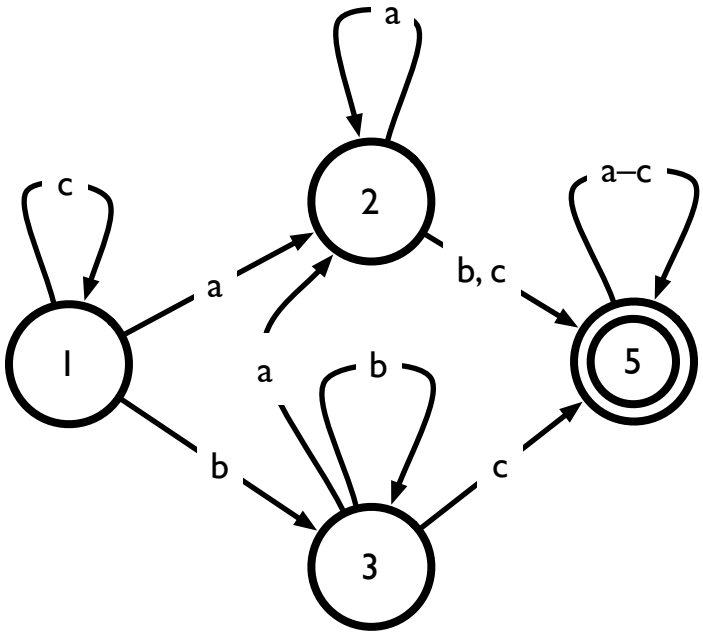
Answer: Using the algorithm from class, we start by combining all the non-final states (1, 2, 3) and all the final states (4, 5, 6). Interestingly, we immediately see that all of the final states cannot be distinguished: on any character, you just go to another final state (in other words, once you get to a final state, you don't leave). Intuitively, this isn't surprising: once you've seen the alphabetical sequence (putting

you in a final state), you're done—whatever else the string has in it doesn't matter. For the non-final states, however, from state 2, if you see a 'b' you go to a final state, while from states 1 and 3 you do not, so we can definitely split state 2 off from states 1 and 2. From state 3, if you see a 'c', you go to a final state, while from state 1 you do not, so we can definitely split state 3 off from state 1.

That leaves us with the following transition table:

State	**a**	**b**	**c**	**Final?**
1	2	3	1	No
2	2	4	4	No
3	2	3	4	No
4	4	4	4	Yes

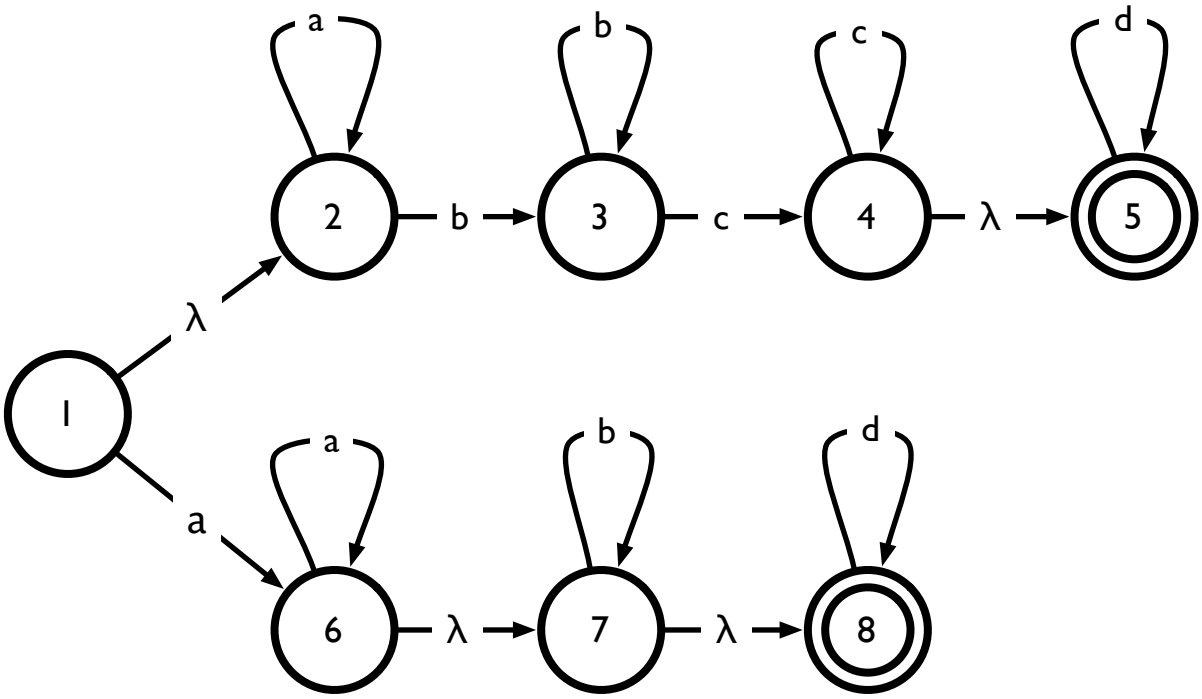
And the following reduced DFA:



5. Build a *reduced, deterministic* automaton for the following regular expression:

$$(a^*b^+c^+d^*)|(a^+b^*d^*)$$

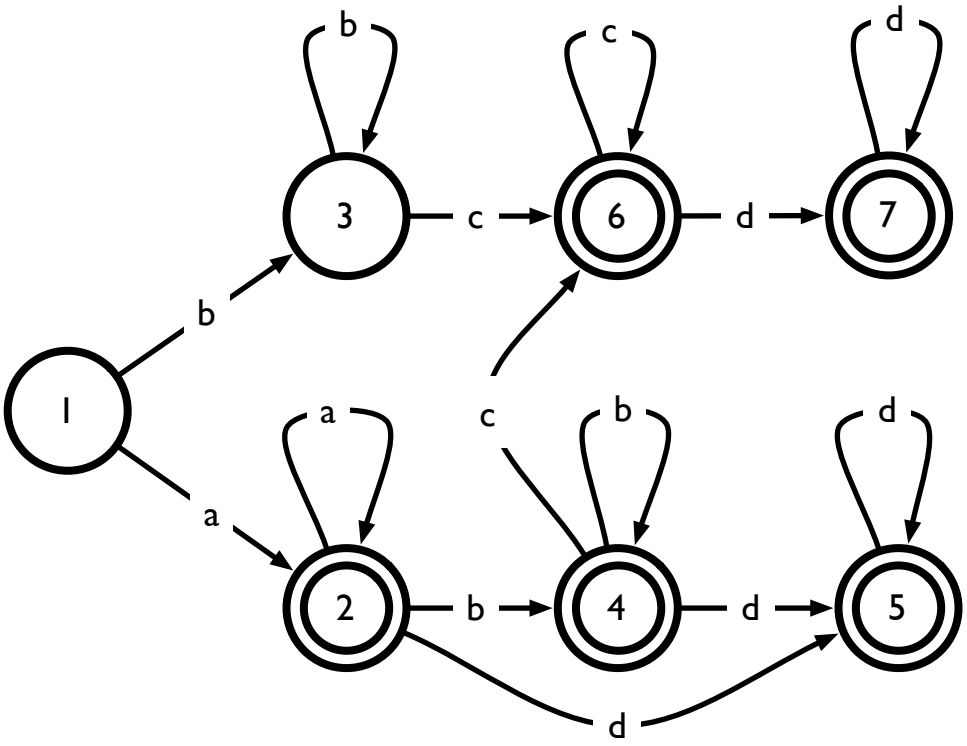
Answer: First, we build the NFA for this regular expression. Here's one possible NFA:



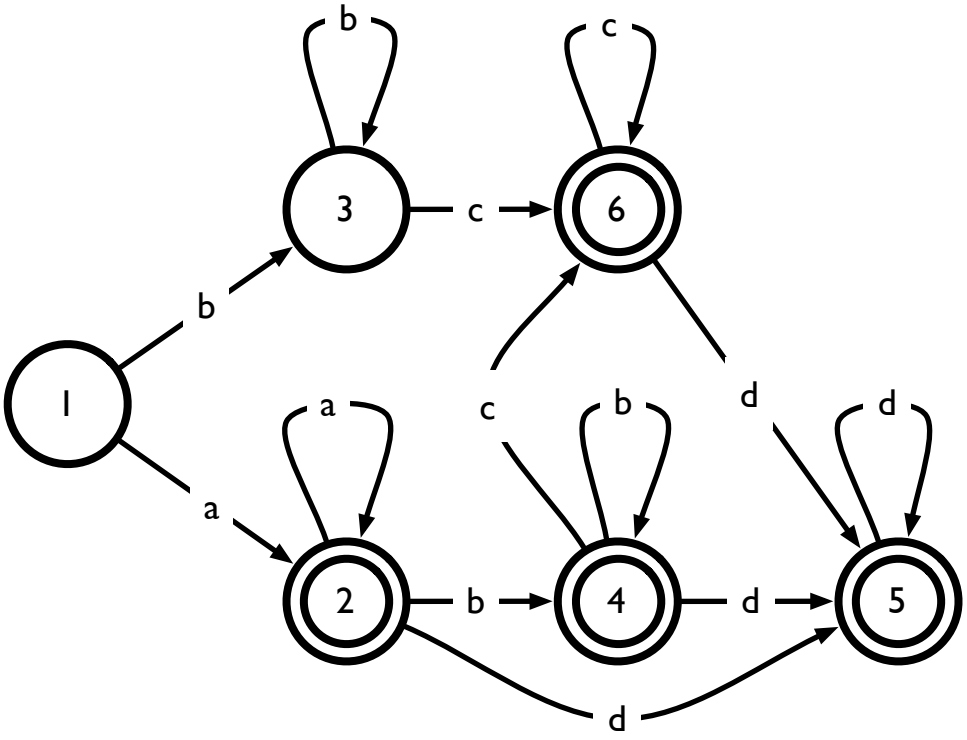
Next, we use the subset construction to produce a corresponding DFA:

State	**a**	**b**	**c**	**d**	**Final?**	**New name**
1, 2	2, 6, 7, 8	3	err	err	No	1
2, 6, 7, 8	2, 6, 7, 8	3, 7, 8	err	8	Yes	2
3	err	3	4, 5	err	No	3
3, 7, 8	err	3, 7, 8	4, 5	8	Yes	4
8	err	err	err	8	yes	5
4, 5	err	err	4, 5	5	yes	6
5	err	err	err	5	yes	7

In graphical form:



When reducing this DFA, we begin by putting together all the final states and all the non-final states. We immediately see that we can split apart the two non-final states (1 and 3), because on a 'c', 1 goes to error, while 3 goes to the merged final state. On a 'b', states 6, 7 and 8 from the merged final state go to error, while 2 and 4 both stay in the merged final state, so we can split those apart, leaving us with two merged final states: {2, 4} and {5, 6, 7}. If we look at the first of these, we see that on an 'a', 2 stays in {2, 4}, while 4 goes to error, so we can split them apart. If we look at {5, 6, 7}, we see that on a 'c', 6 stays in {5, 6, 7} while 5 and 7 go to error, so we can split 6 off. The only merged state we are left with is {5, 7}. We see that there is no way to prove they are different: on 'd', they both stay in {5, 7}, while on any other character, they go to error. So in the reduced DFA, we can merge 5 and 7. That gives us the following reduced DFA:


b
c
3
c
6
b
1
a
c
b
d
d
a
2
b
4
d
5
d