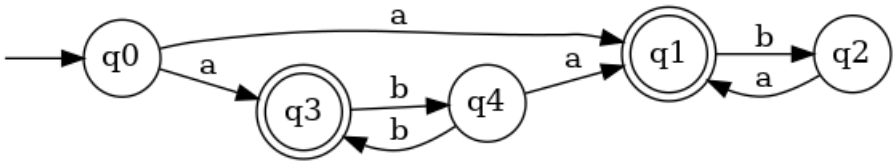
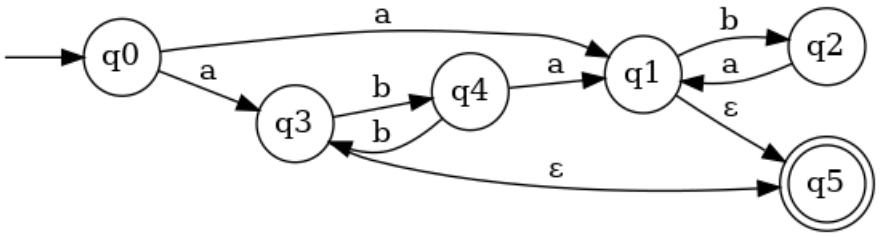
Give a regular expression for the language accepted by this NFA using the construction from class.



Answer: $a(ba)^*|a(bb)^*(ba(ba^*)|\epsilon)$

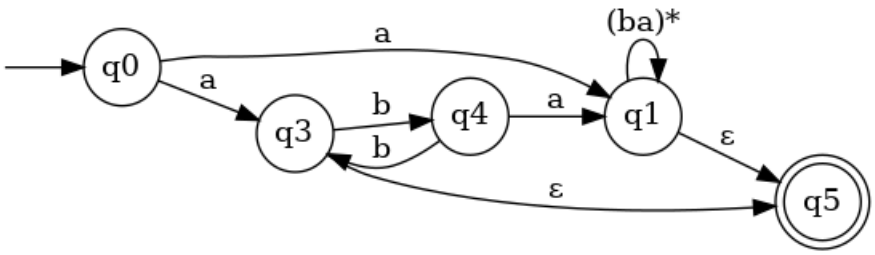
Discussion: The construction from class involves replacing two-transition sequences with single transitions labeled with regular expressions, working to eliminate all of the states other than the start and final states.

This construction requires a single final state, so first transform the NFA so that there's only one final state:

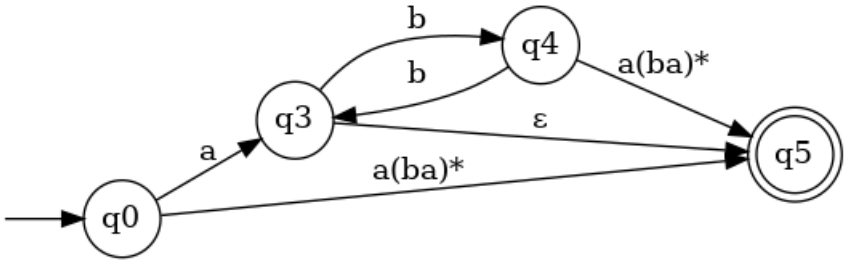


First, change the labels for $q_i \longrightarrow q_i$ transitions to include $*$ in order to reflect the loop. But there aren't any of those at the moment.

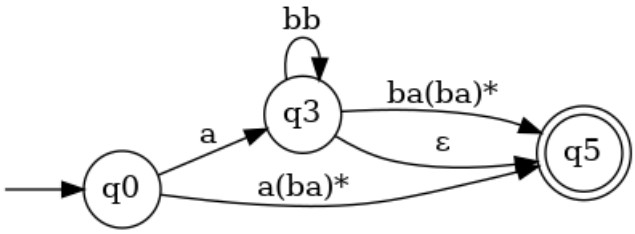
Next, look for transitions of the form $q_i \longrightarrow q_j \longrightarrow q_i$ (a cycle) where q_j has a single transition in and a single transition out. $q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_1$ is such a situation. Eliminate q_2 and replace the sequence of transitions with $q_1 \xrightarrow{ba} q_1$. Since it is a loop, add the $*$.



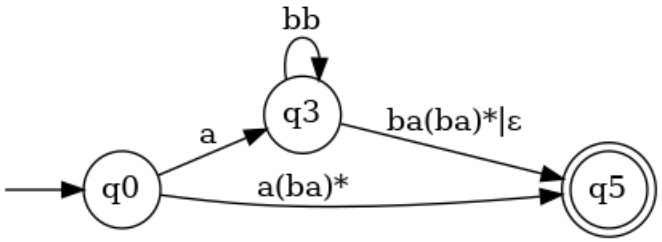
There aren't any more states with only one transition in and out, so consider the next step up: those with three transitions, either two in and one out or vice versa. q_1 qualifies. There are two pairs of transitions involving q_1: $q_0 \xrightarrow{a} q_1 \xrightarrow{\epsilon} q_5$ and $q_4 \xrightarrow{a} q_1 \xrightarrow{\epsilon} q_5$.



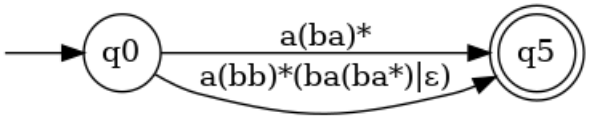
q_4 is another 3-transition state, with two pairs of transitions involving it: $q_3 \xrightarrow{b} q_4 \xrightarrow{a(ba)^*} q_5$ and $q_3 \xrightarrow{b} q_4 \xrightarrow{b} q_3$.



Two transitions between the same states can be combined with |.

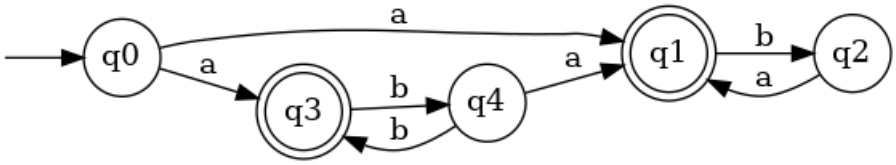


Now q_3 has a single transition in and out.

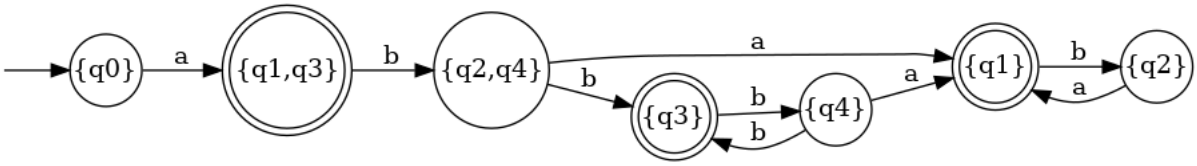


Now we can easily read the regular expression, so it's not really necessary to combine the last two transitions.

Apply the NFA-to-DFA conversion algorithm to construct a DFA that accepts the same language.

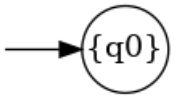


Answer:



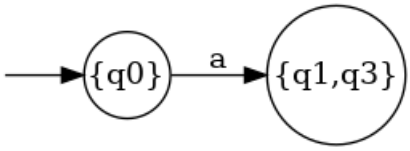
Discussion: You can imagine your finger pointing to the current state as you trace through a DFA for a particular string, and if your finger ends up on a final state when the string is complete, it is accepted. In an NFA you many need many fingers at once, as there can be more than one transition applicable at one time. The algorithm for converting a NFA to a DFA converts multiple fingers to one — the DFA states correspond to the sets of NFA states where there can be fingers pointed after a string has been read.

The start state for the DFA is the set of the start state for the NFA and anything reachable from the NFA's start state via ϵ-transitions.

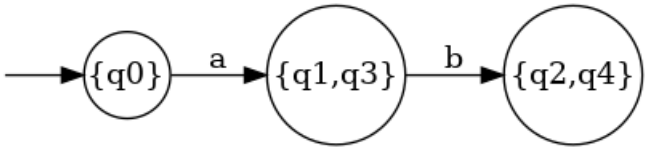


Choose one of the DFA states where the out transitions haven't been considered yet, and add them: for each NFA state in the DFA state, consider where a transition involving symbol σ ends up. Remember that if there isn't a transition involving σ, that string isn't accepted — it's a dead end.

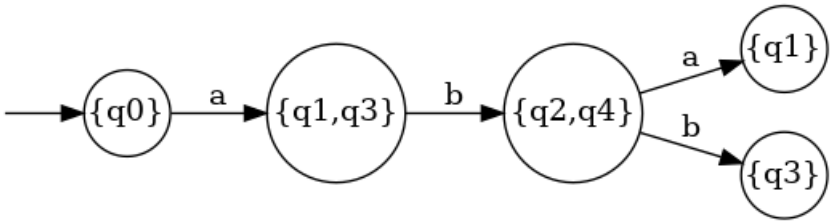
$q_0 \xrightarrow{a} q_1$ and $q_0 \xrightarrow{a} q_2$, but there aren't any transitions from q_0 involving b. Don't forget to also include any states reachable from q_1 and q_2 with ϵ-transitions.



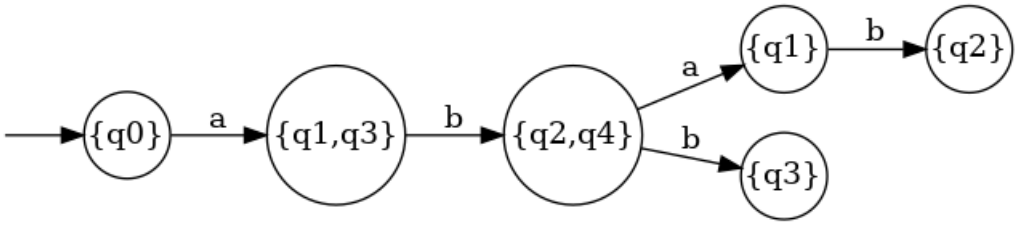
There aren't any transitions leaving q_1 or q_3 for a. For b, $q_1 \xrightarrow{b} q_2$ and $q_3 \xrightarrow{b} q_4$. Also include any states reachable from q_2 and q_4 with ϵ-transitions.



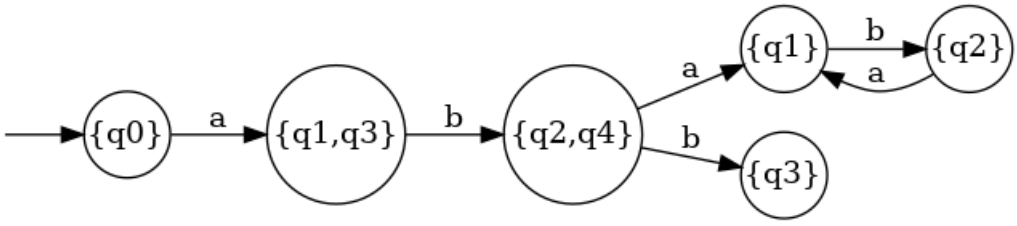
For a, $q_2 \xrightarrow{a} q_1$ and $q_4 \xrightarrow{a} q_1$. For b, $q_4 \xrightarrow{b} q_3$. Don't forget to also include any states reachable from q_1 and q_3 with ϵ-transitions.



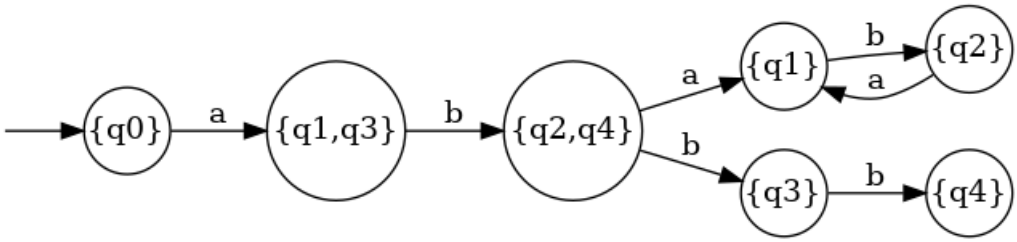
From q_1, only $q_1 \xrightarrow{b} q_2$ applies. Don't forget to also include any states reachable from q_2 with ϵ-transitions.



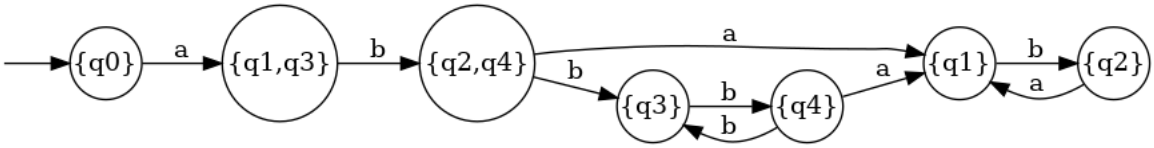
From q_2, only $q_2 \xrightarrow{a} q_1$ applies. Don't forget to also include any states reachable from q_1 with ϵ-transitions.



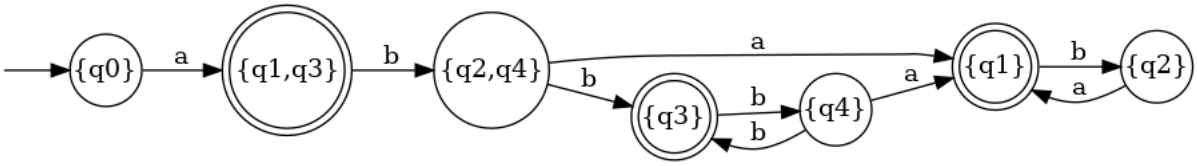
From q_3, only $q_3 \xrightarrow{b} q_4$ applies. Don't forget to also include any states reachable from q_4 with ϵ-transitions.



From q_4, $q_4 \xrightarrow{a} q_1$ and $q_4 \xrightarrow{b} q_3$ Don't forget to also include any states reachable from q_1 or q_3 with ϵ-transitions.

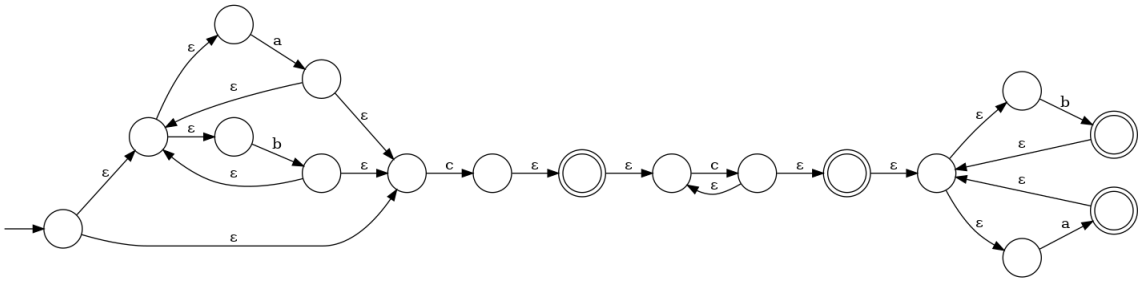


The last step is to indicate the final states. Any NFA only requires a finger to end up in a final state, not all of them, so any state of the DFA that includes one of the NFA final states should be final.



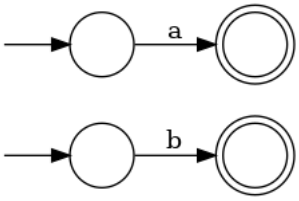
Draw an NFA that accepts $L((a|b)^*cc^*(a|b)^*)$ using the construction from Theorem 3.3.

Answer:

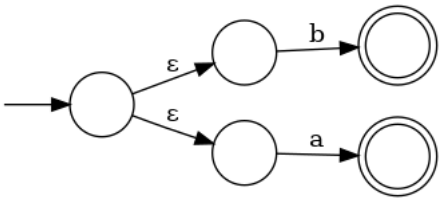


Discussion: The proof of Theorem 3.3 gives NFAs for each of the basic regular expressions, including ϵ and a single symbol σ, plus how to construct NFAs from $r_1|r_2$, r_1r_2, and r^* from NFAs for r_1, r_2, and r. ($*$ isn't in the book but was discussed in class and can be found in the examples from class.)

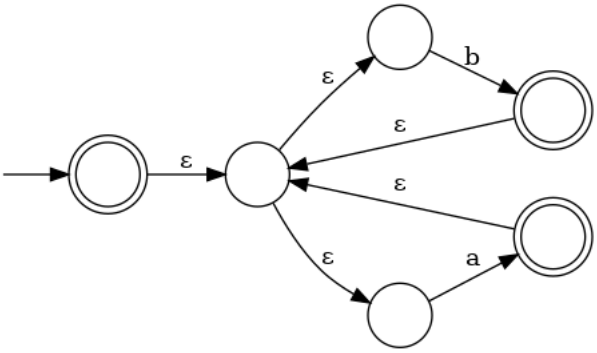
Start with NFAs for a and b:



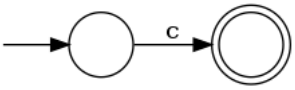
$a|b$ involves a new start state with ϵ-transitions:



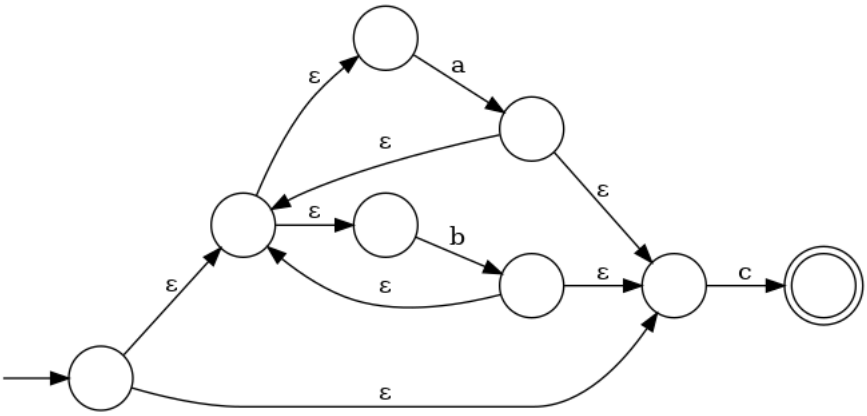
$(a|b)^*$ involves a new start (and final) state with an ϵ-transition, plus ϵ-transitions linking final states back to the beginning:



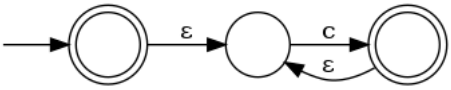
Next, c:



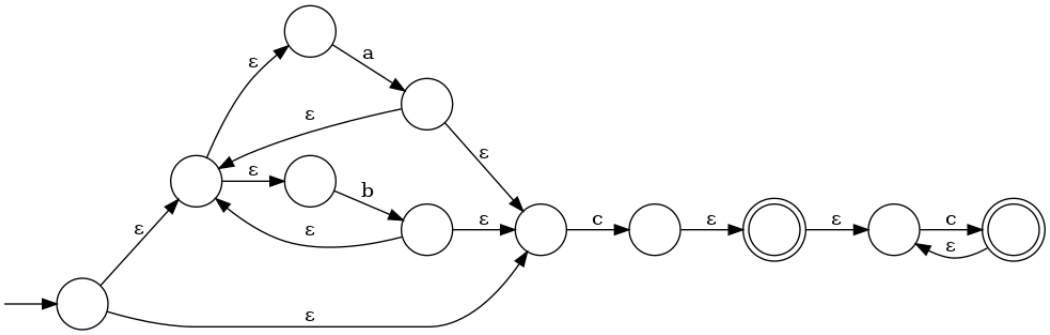
For $(a|b)^*c$, connect the final states from $(a|b)^*$ to the start state of c:



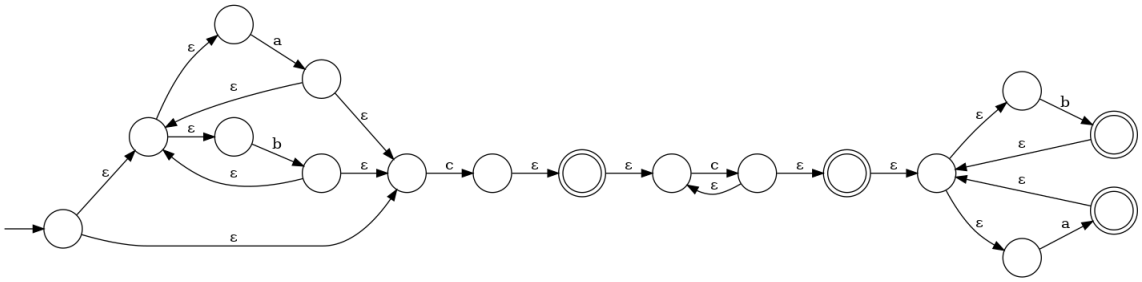
Next, c^*:



$(a|b)^*cc^*$:



And finally, $(a|b)^*cc^*(a|b)^*$:



Give a context-free grammar for the language $L = \{\ a^n b^m \mid n > 2m \in \mathbb{N}\ \}$.

Answer:

$$S \longrightarrow aaSb$$
$$S \longrightarrow A$$
$$A \longrightarrow aA$$
$$A \longrightarrow a$$

Discussion: When there are constraints on the number of different symbols, they need to be generated in pairs. Here, in addition to the pattern of as followed by bs, there need to be at least twice as many as as bs. Start with generating exactly twice as many as:

$$S \longrightarrow aaSb$$
$$S \longrightarrow \epsilon$$

Each time $S \longrightarrow aaSb$ is used, we get two more as at the beginning and one more b at the end. $S \longrightarrow \epsilon$ allows the process to end — otherwise there is no way to eliminate all of the non-terminals.

Exactly twice as many as as bs isn't legal, we need at least one more. Forcing the last S to be replaced by an a instead of ϵ ensures that:

$$S \longrightarrow aaSb$$
$$S \longrightarrow a$$

But this produces $a^{2m+1}b^m$ and we need any number of as $< 2m$. A rule of the form $A \longrightarrow aA$ produces any number of a, but do that means $S \longrightarrow aS$ or do we need to introduce a new non-terminal? Since it doesn't matter which order we generate as-matched-with-bs vs just as, $S \longrightarrow aS$ would work. But that's definitely an ambiguous grammar, and unambiguous grammars are nice. So it is preferable to fix the order for the as (matched as first):

$$S \longrightarrow aaSb$$
$$S \longrightarrow A$$
$$A \longrightarrow aA$$
$$A \longrightarrow a$$

$S \longrightarrow A$ allows the switch from matched as to additional as.

Now we can generate any number of as $> 2m$. The last step is to check base cases — is ϵ permitted in the language? What about the case of $m = 0$? $m = 0$ is possible, but since $n > 2m$, there must then be at least one a. The grammar does not allow ϵ because the shortest derivation is $S \Longrightarrow A \Longrightarrow a$, and it does allow a. So it seems like problem solved.

Write a BNF grammar for the following.

```
<html>
<head><title>the title</title></head>
<body>
  <p>paragraph text</p>
  <ul>
    <li> list item
    <li> <p>another list item</p>
    <li> <p>list item</p>
         <p>second paragraph</p>
      <ul>
        <li> nested item
      </ul>
  </ul>
</body>
</html>
```

Answer:

⟨html⟩ ::= "<html>" ⟨head⟩ ⟨body⟩ "</html>"
⟨head⟩ ::= "<head>" ⟨title⟩ "</head>"
⟨title⟩ ::= "<title>" **text** "</title>"
⟨body⟩ ::= "<body>" [⟨p⟩ | ⟨ul⟩] ... "</body>"
⟨p⟩ ::= " " **text** " "
⟨ul⟩ ::= "
" [⟨li⟩] ... "
"
⟨li⟩ ::= "<li>" (**text** | [⟨p⟩ | ⟨ul⟩] ...)

Discussion: A convenient structure is a rule for each tag. Start with some of the simpler ones. Let **text** be plain text without any more tags (defining that is beyond the intent of this question).

⟨html⟩ ::= "<html>" ⟨head⟩ ⟨body⟩ "</html>"
⟨head⟩ ::= "<head>" ⟨title⟩ "</head>"
⟨title⟩ ::= "<title>" **text** "</title>"
⟨p⟩ ::= " " **text** " "

The body can contain any number of `p` and `ul` tags in any order.

⟨body⟩ ::= "<body>" [⟨p⟩ | ⟨ul⟩] ... "</body>"

A list can only have list items.

⟨ul⟩ ::= “
” [⟨li⟩] . . . “”

List items can be just text or a combination of paragraphs and lists.

⟨li⟩ ::= “<li>” (**text** | [⟨p⟩ | ⟨ul⟩] . . .)