Math Circles – Finite Automata
Question Sheet 2 (Solutions)

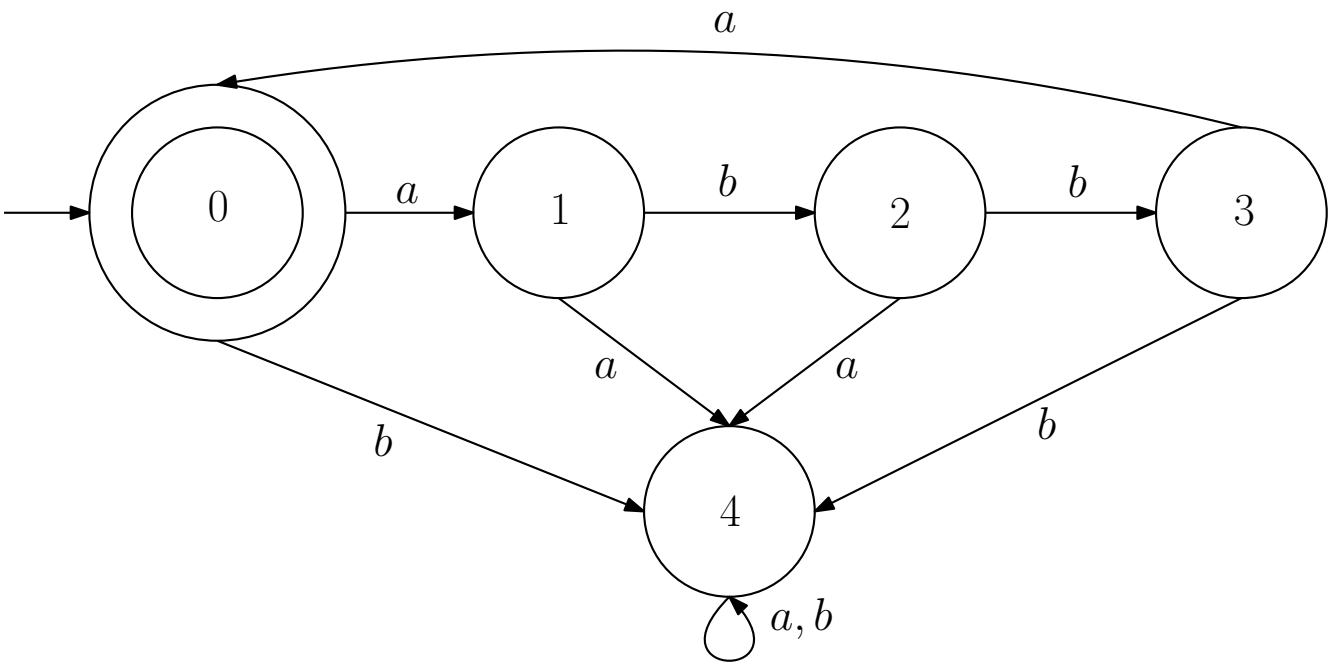
Nickolas Rollick – nrollick@uwaterloo.ca

November 14, 2018

Note: These solutions may give you the *answers* to all the problems, but they usually won't tell you how to *get* the answer. All the fun and profit lies in finding the answers for yourself... Also be aware that some questions have more than one solution – this will only provide you with one of them!

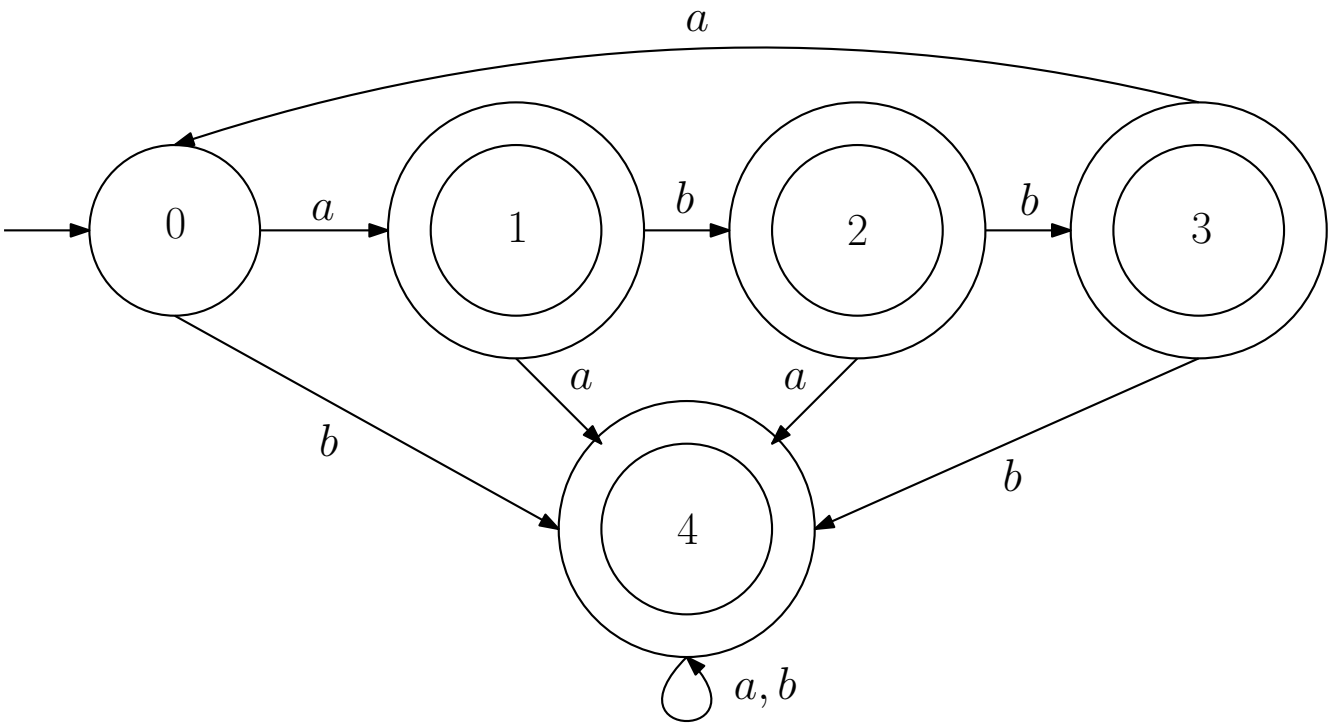
Questions from Lesson

1. Consider the DFA from last time, accepting the *abba* language:

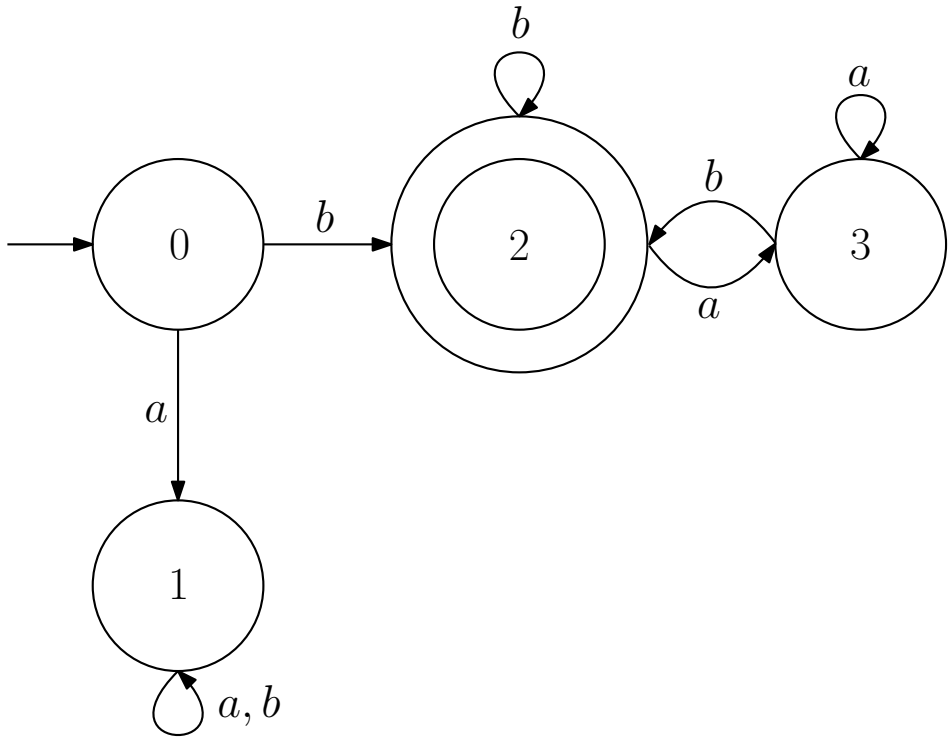


Write down a DFA accepting the *complement* of this language (the strings NOT accepted by the original DFA).

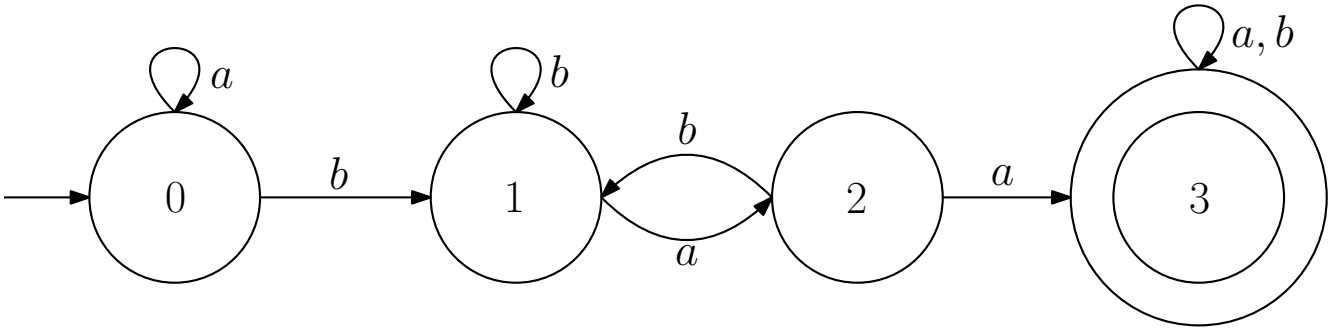
Solution: As mentioned during the session, the easiest way to do this is to change all the accepting states into rejecting states and all rejecting states into accepting states:



2. Here is a DFA accepting the strings that start and end with b:

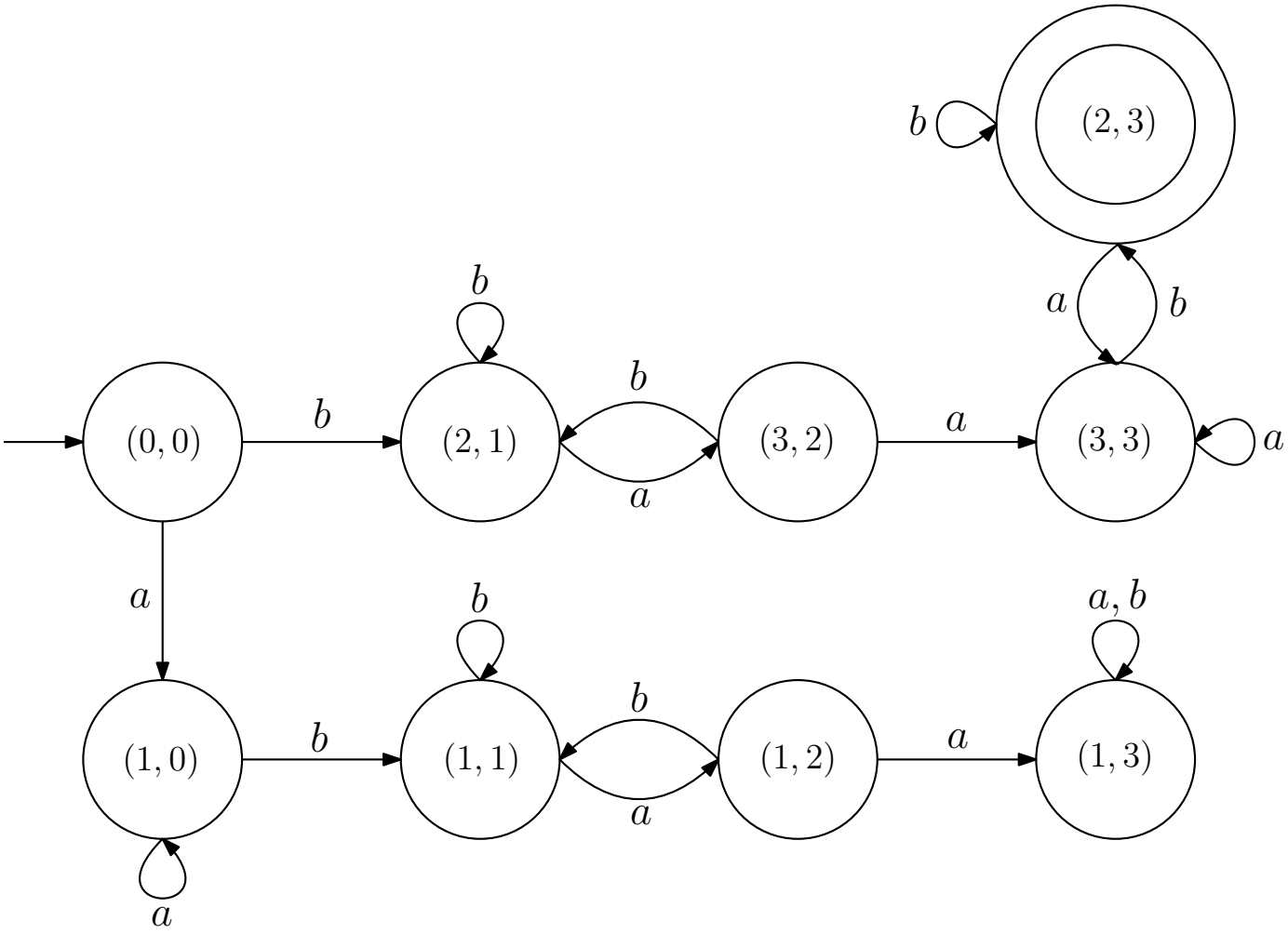


Here is a DFA accepting the strings containing baa inside them:



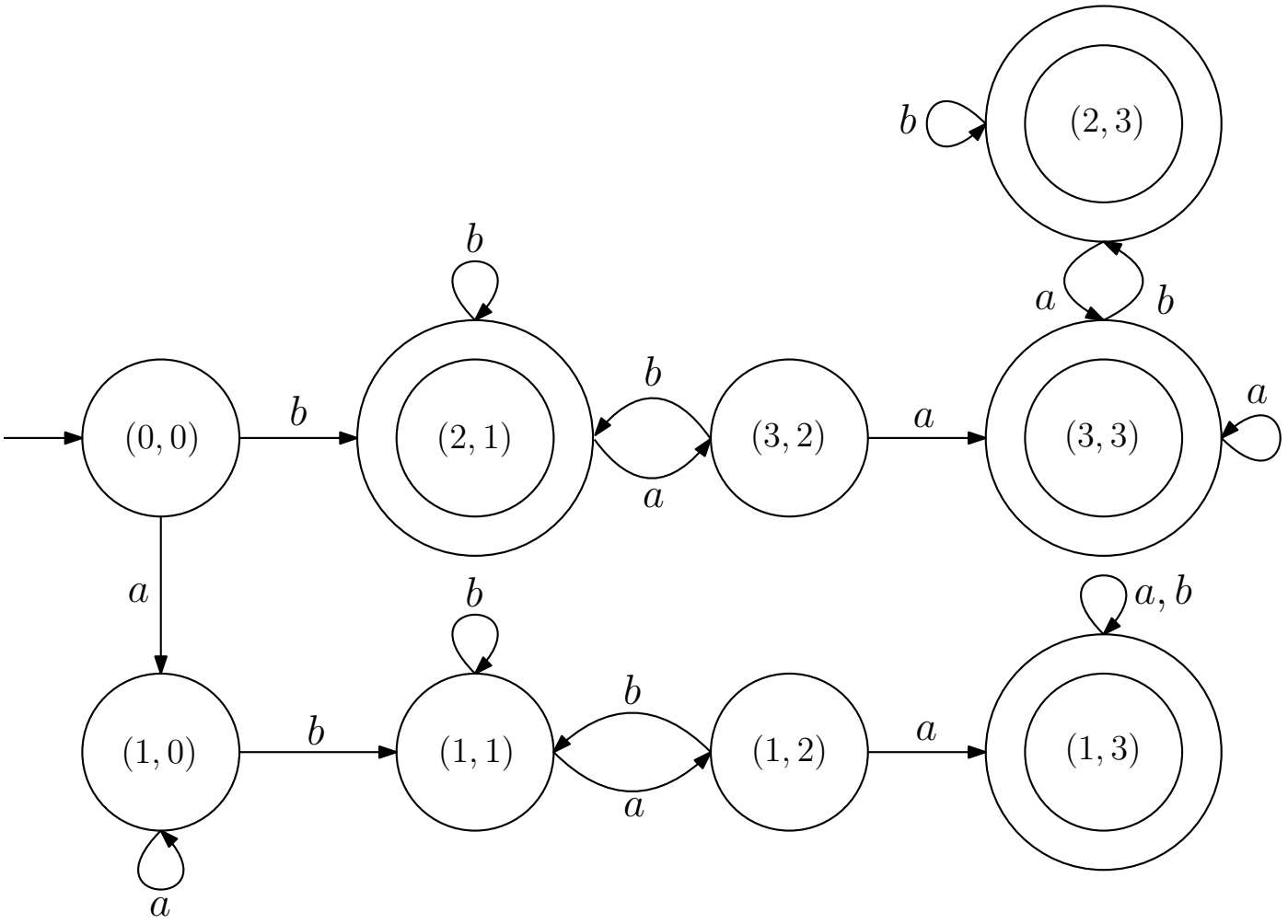
(a) Can you build a DFA accepting the *intersection* of these two languages (the strings accepted by *both* DFAs)?

Solution: We did this one during the session as well. Here it is:



(b) Can you build a DFA accepting the *union* of these two languages (the strings accepted by *either* DFA)?

Solution: This was also done in our session:



3. Is every language a regular language? If not, can you provide an example of a language that is not regular?

   Solution: As discussed in our session, not every language is regular. The language of strings with some number of as followed by an *equal* number of bs is an example. To see why, suppose there were a DFA accepting this language. Then there are two different strings a^m and a^n that put the DFA into the same state if given as input. (There are only finitely many states, but infinitely many strings with a bunch of as in a row, so this must happen). Call this state q.

   The DFA must accept both $a^m b^m$ and $a^n b^n$, since both strings belong to the language. So if the DFA is in state q and reads m letter bs in a row, it ends up in some accepting state r. But then the DFA ends up in state r after reading $a^m b^n$ as well, so it must accept that string. Since $a^m b^n$ is not in the language, a DFA accepting this language cannot exist.

Extra Questions

4. The language of *legal bracketings* is a collection of strings using the letters a (left bracket) and b (right bracket) following the rule that every b in the string must have a matching a coming before it.

   Which of the following strings belong to this language?

   (a) a
   (b) ba
   (c) $abab$
   (d) $abba$
   (e) $aabb$
   (f) $ababb$
   (g) $abaababb$

   Is the language of legal bracketings regular? Why or why not?

   Solution: The strings $abab$, $aabb$, and $abaababb$ belong to the language, while a, ba, $abba$, and $ababb$ do not.

   The language of legal bracketings is not regular, for exactly the same reason that the language in the previous solution is not regular. Indeed, we can use the same argument! Notice that all strings with some number of as followed by an equal number of bs belong to the language of legal bracketings (it's some number of open brackets followed by the same number of closed brackets).

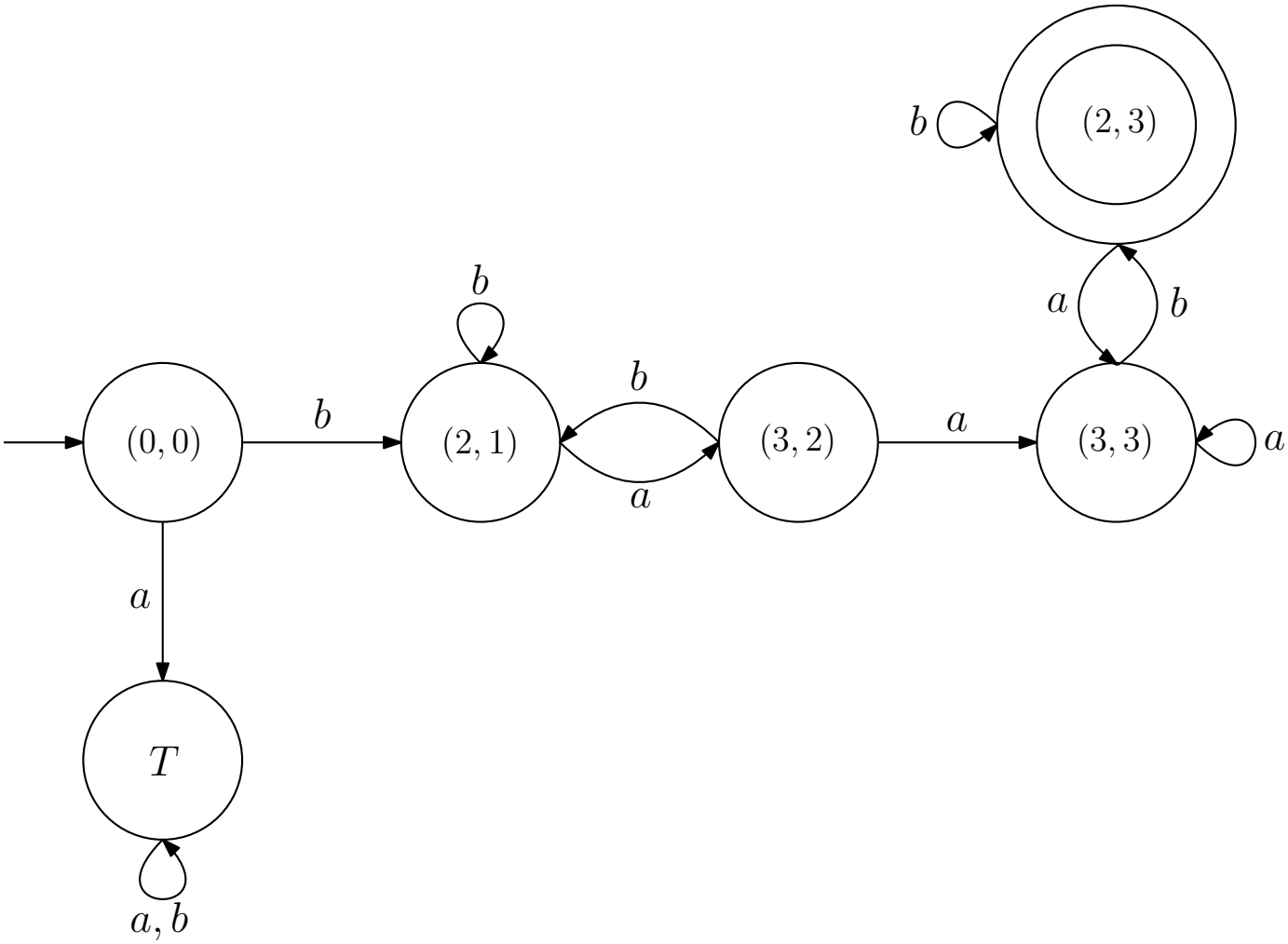
   If there were a DFA accepting the language of legal bracketings, it must have a finite number of states. Therefore, there are two different strings a^m and a^n, where the DFA ends up in the same state q after reading both. Again, both $a^m b^m$ and $a^n b^n$ are in the language of legal bracketings, so the DFA must accept both strings.

   So, if we start the DFA and give it the string a^n, it ends up in state q. If we follow that with b^n, it must end up in some accepting state, because $a^n b^n$ must be accepted. But then $a^m b^n$ must be accepted as well, since the DFA ends up in state q after reading a^m and goes to the same accepting state after reading b^n. Since $a^m b^n$ is not a legal bracketing, no such DFA can exist.

5. The steps we talked about for building a DFA accepting the union or intersection of two regular languages always work, but sometimes it creates more states than we really need. When we built a DFA accepting the strings that start and end with b, and contain baa somewhere inside, the resulting DFA had nine states.

   Can you write down a DFA accepting the same language, but with only six states?

   Solution: The key is to notice that when the DFA we constructed reads an a, there is no way the string can possibly be accepted. So, we can collapse all of the states in that part of the DFA into a single "trash can" state (labelled T here).



6. Suppose we have two regular languages, which we'll call L_1 and L_2. Consider the language of all strings belonging to L_1 but not L_2. Is this language always regular?

   If so, describe a process for building a DFA accepting this language, given DFAs accepting L_1 and L_2.

   If not, explain why not.

   Solution: Yes, this language is always regular. To see why, suppose we start with a DFA accepting L_1 and a DFA accepting L_2. In today's session, we argued that we can build a DFA accepting all the strings *not* in L_2, just by flipping all the accepting and rejecting states.

   So now we have a DFA accepting exactly the strings in L_1, and a DFA accepting exactly the strings NOT in L_2. Now, we can always build a DFA accepting only the strings accepted by both of these DFAs, by keeping track of the states of both machines at the same time (like we did in Question 2). The accepting states of this new DFA are the ones labelled with states that are *both* accepting states for the original DFAs.

   This new DFA we have at the end accepts strings belonging to L_1 and not in L_2.