Math Circles – Finite Automata
Question Sheet 3 (Solutions)

Nickolas Rollick – nrollick@uwaterloo.ca

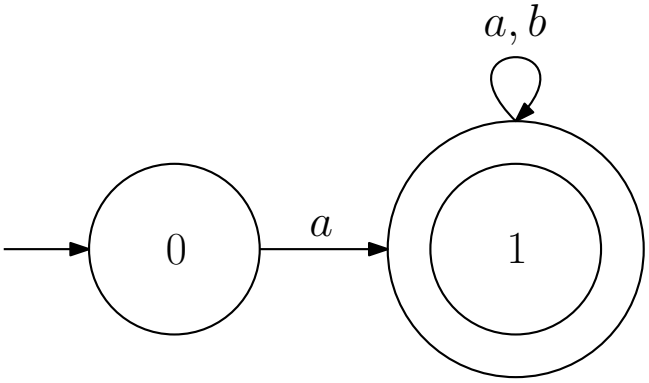
November 21, 2018

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

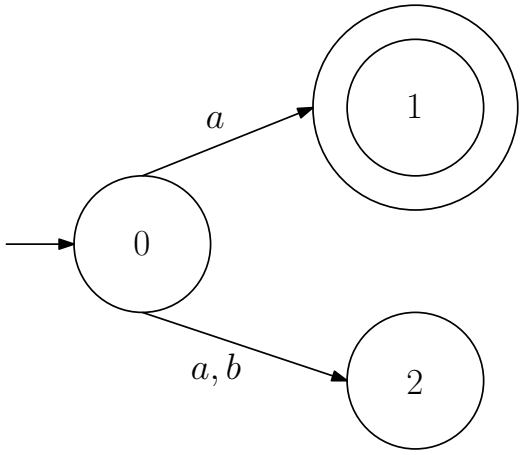
1. Describe all strings accepted by each of the following NFAs:

   (a)



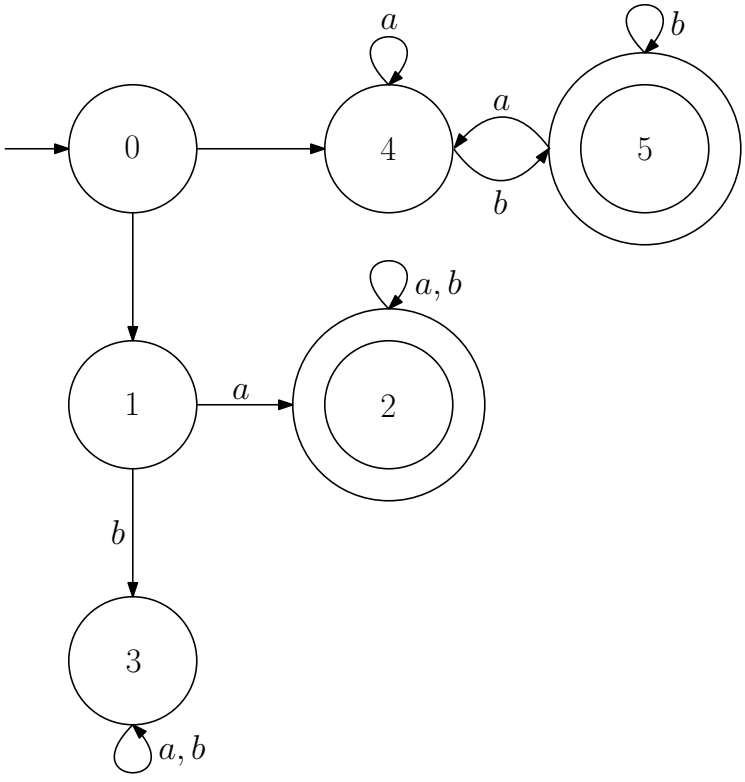
   Solution: This NFA accepts all the strings that start with a.

(b)



Solution: This NFA accepts the string a, and nothing else.

(c)

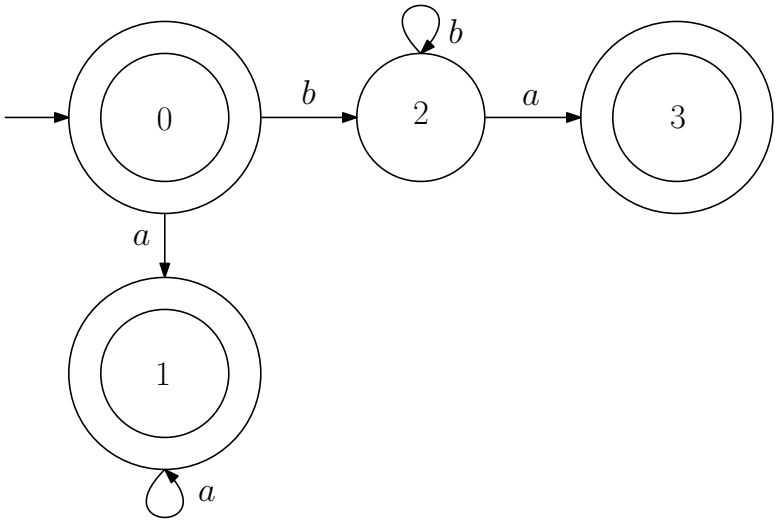


Solution: This NFA accepts the strings that either start with a, or end with b.

2. Draw an NFA satisfying each set of conditions:

   (a) It has four states, and accepts the language of strings that have no bs, or else have one or more bs followed by a single a.

   Solution:

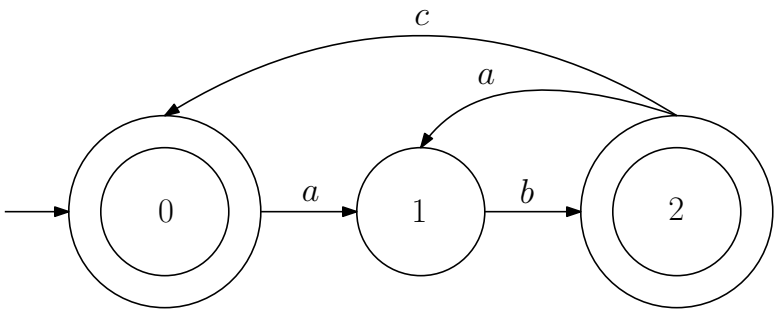


   (b) It has three states and allows three possible symbols: a, b, and c. It accepts the language defined by the following conditions:

   - The empty string is in the language.
   - Given a string in the language, adding ab or abc to the end gives another string in the language.

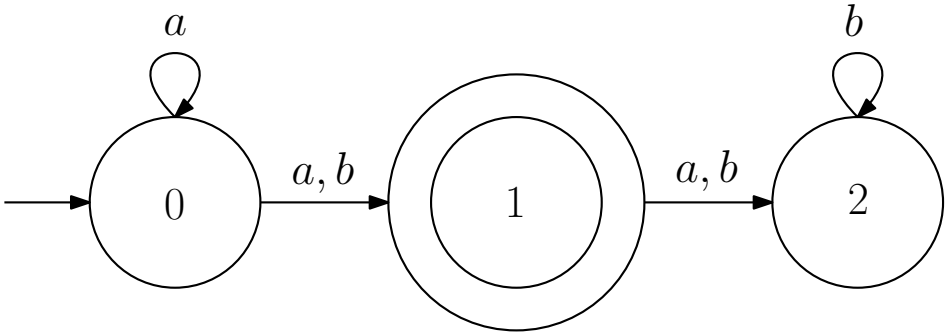
   (For example, ab, abc, $abcab$, $abab$, $abcabc$ all belong to this language.)

Solution:

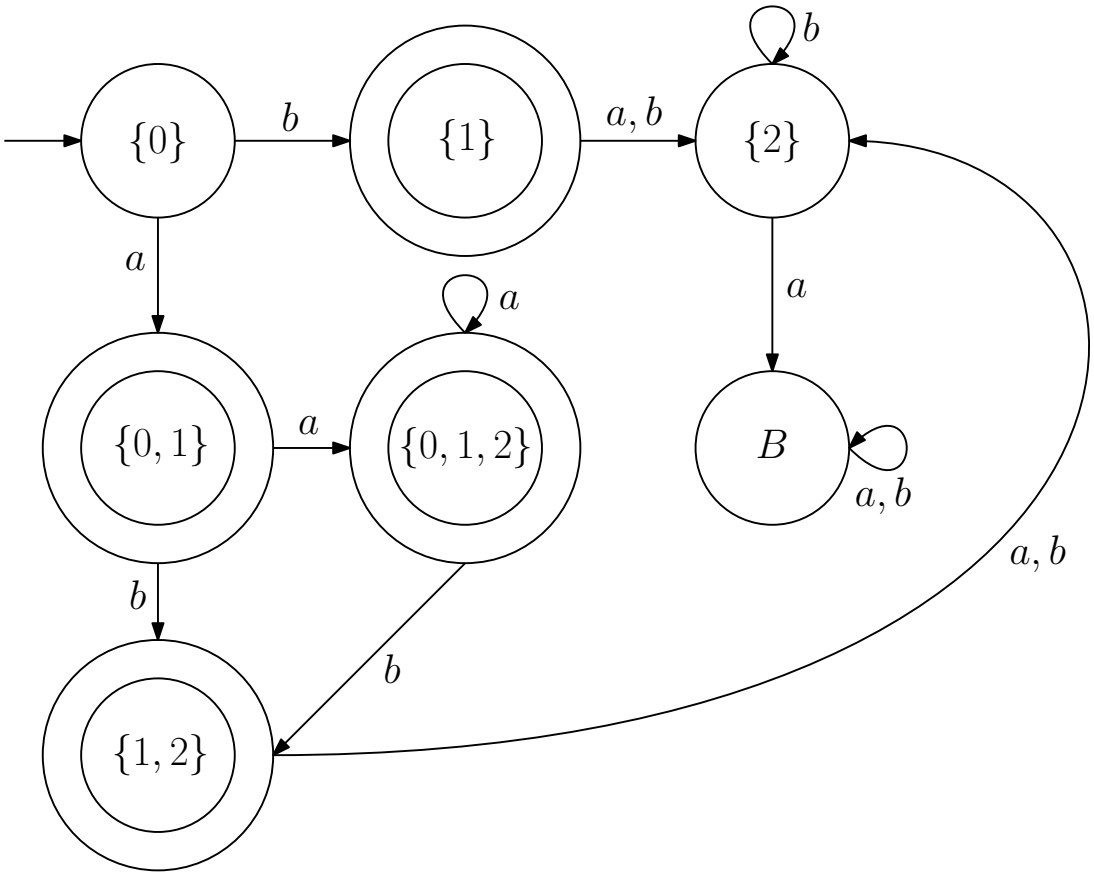


3. Given the following NFAs, construct DFAs that accept the same language:

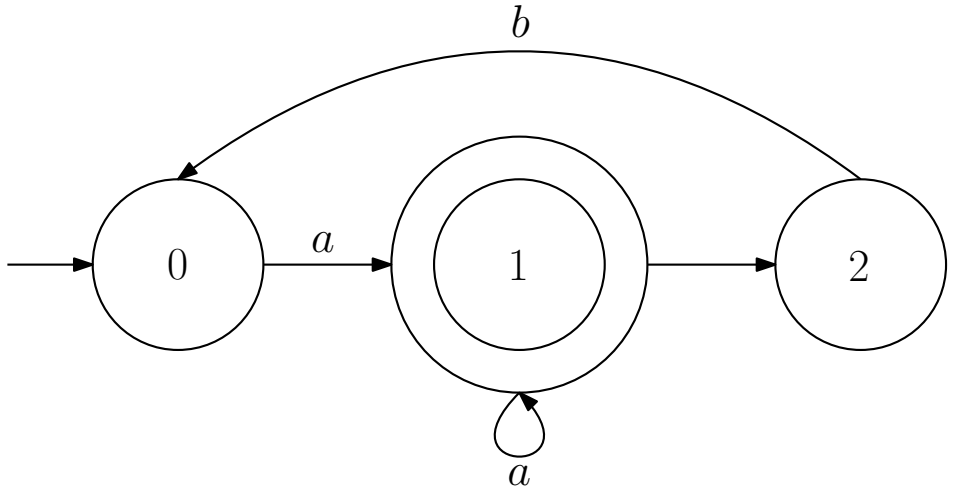
(a)



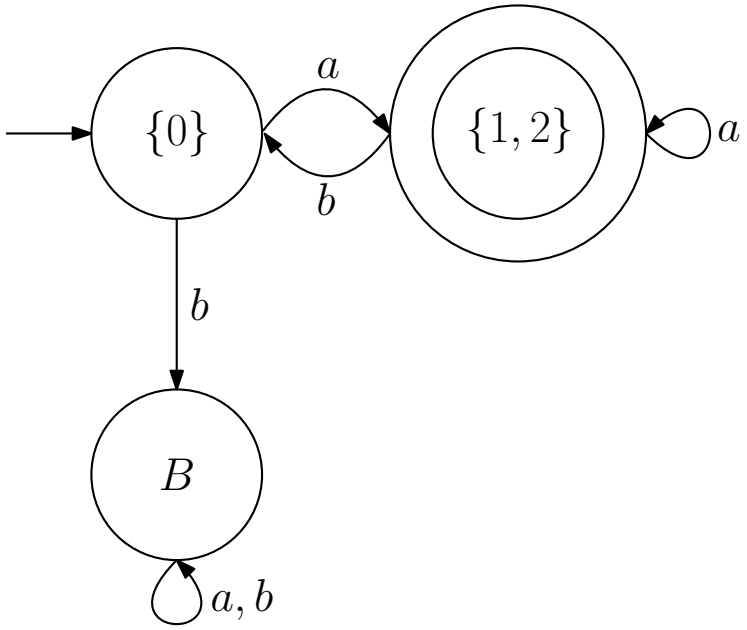
Solution: As discussed during the session, designing a DFA that keeps track of all "possible universes" leads to:



(b)



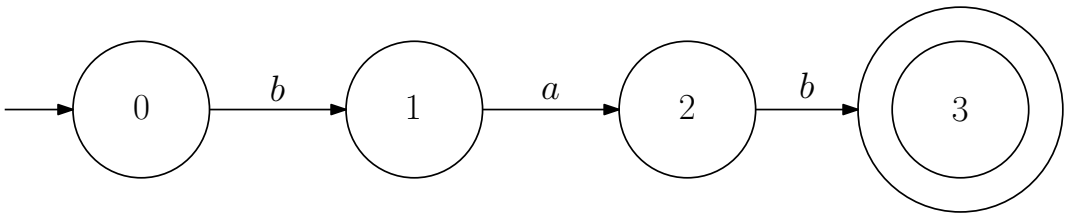
Solution: Again, we can follow the same "possible universes" construction that we discussed during the session. The accepting states in the new DFA are the ones with a 1 in them, since state 1 is the only accepting state of the NFA.



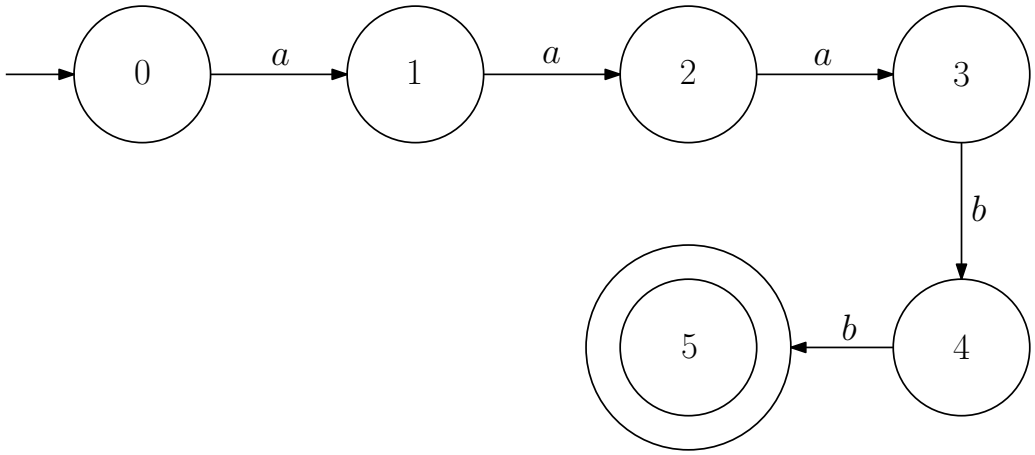
Extra Questions

4. Here, we will see that every *finite* language built out of strings of as and bs is regular. In other words, given any finite collection of strings, there is some DFA that accepts exactly those strings and no others.

   (a) Draw an NFA with four states and three arrows that accepts the string *bab* and nothing else. Now draw an NFA with six states and five arrows that accepts the string *aaabb* and nothing else.
   Solution: For *bab*:



   For *aaabb*:



   (b) Building on the pattern from the previous part, suppose we are given any string of as and bs (maybe it's *aaababbaababababba*). Describe how to build an NFA that accepts that string, and nothing else.

   Solution: If the string has n symbols in it, we build an NFA that has $n+1$ states, labelled 0 up to n, and n arrows. There is one arrow from state 0 to state 1 labelled with the first letter of the string, one from state 1 to state 2 labelled with the second letter, and so on. Only the last state n is an accepting state. Unless the input to the NFA is exactly the specified string, the NFA will either not make it to state n and reject the string, or it will read a letter that it can't handle, and break down.

(c) Given any single string, explain why there is a **D**FA (not just an **N**FA) accepting that string, and nothing else.

Solution: In the previous part, we showed there was an NFA that accepts the given string, and nothing else. Applying the "parallel universes" construction, we can take that NFA and get a DFA accepting the same language (so exactly that string, and nothing else).

(d) Last time, we saw that for any two DFAs, accepting any two languages, we can build another DFA accepting the strings in either language. Using this, explain how we can take any **two** strings and build a DFA accepting those two strings and nothing else.

Solution: Given any two strings, we can apply the previous part to find two DFAs, each of which accepts exactly one of the strings, and nothing else. Applying the construction from last time, we can build a new DFA accepting only the strings accepted by either DFA. In other words, this new DFA accepts the two given strings, and nothing else.

(e) Building on the idea from the previous part, explain how we can take any finite number of strings and build a DFA accepting those strings and nothing else.

Solution: Suppose we have a finite collection of strings, which we'll call $w_1, \ldots, w_n$. By part (c), we can find DFAs accepting w_1 and nothing else, w_2 and nothing else, w_3 and nothing else, etc. Taking the DFA accepting w_1 and nothing else and the one accepting w_2 and nothing else, we build a new DFA accepting the strings that either of these DFAs accept. In other words, this new DFA accepts w_1, w_2, and nothing else.

Next, we take this new DFA and the one accepting w_3 and apply the same construction, giving a DFA accepting w_1, w_2, w_3, and nothing else. Repeating this over and over again, we end up with a DFA that accepts exactly $w_1, \ldots, w_n$.

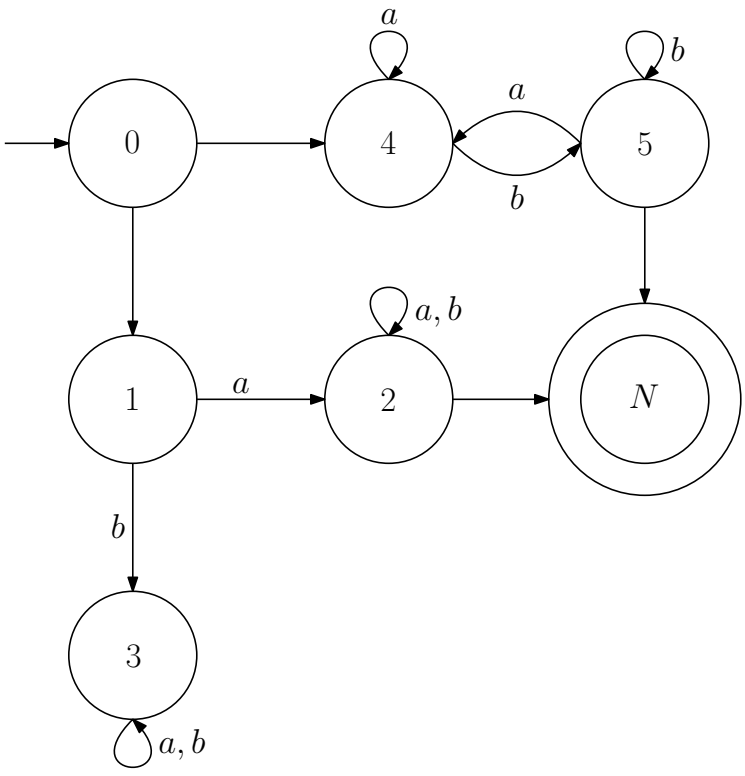
5. Given any NFA, explain how to construct a new NFA with exactly one accepting state that accepts the same language. (**Hint:** Try this on a few examples first – the NFA in Question 1(c) is a good place to start.)

   Solution: Here is how the construction works. Take the original NFA and add in one new state (this will be the accepting state of the new NFA). Take all the accepting states of the original NFA and add a blank arrow from those states to the new state. Then make all states of this new NFA rejecting states, except for the new one we just added in.

   Why does this accept the same language? We have to check that it accepts and rejects exactly the same strings as the original NFA. First, suppose a string is accepted by the original NFA. This means there is some collection of choices the original NFA can make when reading the string, where it ends up at an accepting state. Since we haven't taken away any arrows in the new NFA, the new NFA can make the same choices and end up at the same state. The only problem is that this is not an accepting state any more. But in the new NFA, there must be a blank arrow from this state to the new accepting state, so this new NFA can choose to follow the blank arrow after reading the string and accept it.

   Now suppose we have a string that is rejected by the original NFA. This means no matter what choices the original NFA makes when reading the string, it will always be rejected. Note that the only extra choices the new NFA can make are to follow the blank arrows to the new accepting state. But since there are no arrows coming out of that state, if the NFA still has letters left to read when it does this, it will break and reject the string. So the only way the new NFA can accept a string that the old NFA rejected is if it follows one of the new blank arrows *after reading all the symbols*. The only way it can do that is if it passes through an accepting state for the old NFA after reading all the symbols, which would cause the string to be accepted by the original NFA too.

   An example might be useful in making this clearer. Here is what we get when applying this construction to the NFA from Question 1(c). The new accepting state has been labelled N (for "new").



6. For any string, the *reverse* of that string is obtained by reversing the order of all the symbols. For example, the reverse of *aabab* is *babaa*, and the reverse of *abba* is *abba* again. Given a language, we can define the *reverse language* to be what you get if you reverse all the strings in the original language.

   If a language is regular, explain why the reverse language is also regular. (**Hint:** Starting with a DFA accepting the original language, turn it into an NFA with a single accepting state that accepts the same language, using the previous question. Then convert that NFA into a new NFA accepting the reverse language.)

   Solution: Since every NFA can be converted into a DFA accepting the same language, it is enough to build an NFA accepting the reverse language.

   If a language is regular, we can find a DFA that accepts the language. Since a DFA is a special kind of NFA, we can apply the previous question to turn this DFA into an NFA with a single accepting state that also accepts the language.

   Finally, we turn this NFA into an NFA accepting the reverse language. All we need to do is change the initial (starting) state of the NFA into an accepting state, change the unique accepting state of the NFA into an initial state, and reverse **all** of the arrows.

   Then, given a string in the original language, there is some way the original NFA can accept it, by following a collection of arrows from the initial state to the single accepting state. By starting at the accepting state and following all those arrows backwards, we end up back at the initial state, which means that our new NFA accepts the reverse string. The argument works the other way too: if the new NFA accepts a string, then in the original NFA, there is some way to start at the accepting state, follow the arrows backwards, and end up at the initial state after reading the string. This means the original NFA accepts the reverse of this string, so that the string we started with is in the reverse language.