Intermediate Math Circles
Wednesday, November 21, 2018
Finite Automata III

Nickolas Rollick – nrollick@uwaterloo.ca

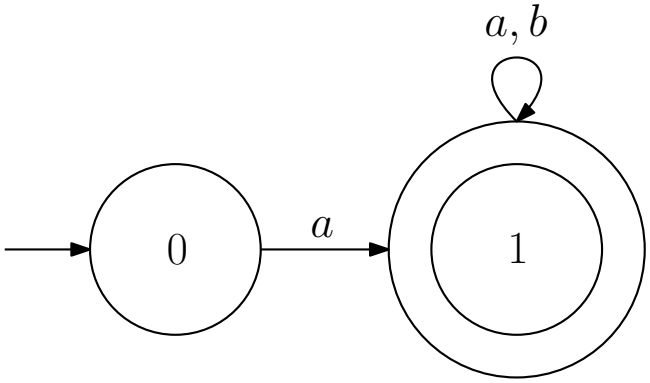
Non-Deterministic Finite Automata (NFAs)

The last time we met, we spent time exploring *regular languages*, the languages that can be accepted by some DFA. In particular, most of our time went into building new regular languages from old ones. At the end of our meeting, we learned that there are limitations to DFAs: not every language is regular. In particular, the language of legal bracketings, something we really want computers to recognize, cannot be accepted by any DFA!

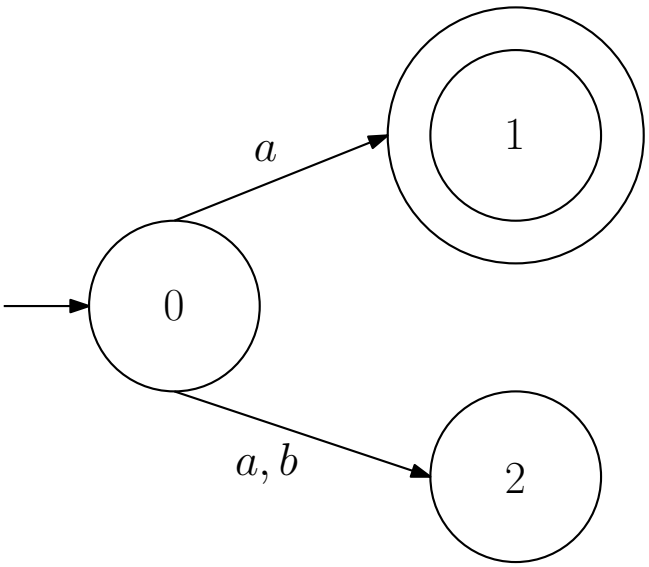
Intuitively, the problem seemed to be a lack of memory: DFAs have only finitely many states, and so they can only keep track of a finite amount of information. So, it seems like we might want to build a better model of a computer, offering more storage. That is definitely one way to proceed, but we'll explore a different avenue, one that sounds a little like *quantum computers*, if you know anything about those. At any rate, what we're about to describe is a little different from the computers you know and love.

We'll take a look at a *non*-deterministic finite automaton (or NFA for short). The picture of an NFA is a lot like a DFA, except that we're allowed to break some of the rules. Just like a DFA, we'll have a finite number of states, one of which is the initial state. Some of the states will be accepting states, and some will be rejecting states, as usual.

But now things get different. Remember, in a DFA, for every allowed symbol, there is exactly one arrow coming out of every state labelled with that symbol. Now, we're allowed to break that rule. For a given symbol, there might be *no* corresponding arrow, like in this NFA:

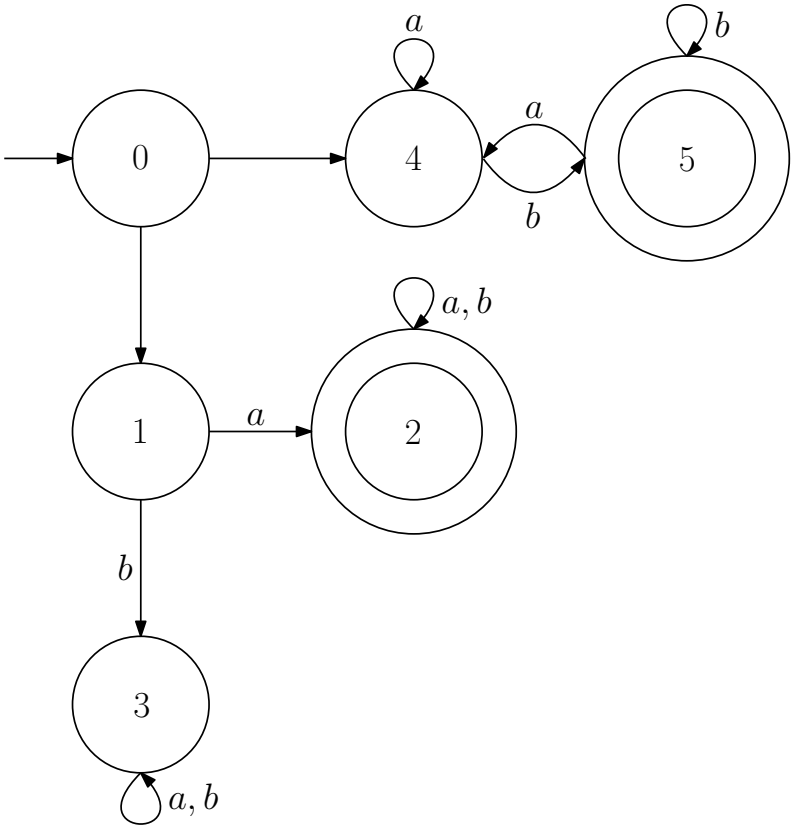


Here, there is no letter b coming out of state 0. We can also break this rule in a different way: there might be *more than one* arrow coming out of a state labelled with the same letter, like this NFA:



In this case, there are two arrows labelled with a coming out of state 0 – one to state 1, and one to state 2. There are also a lot of arrows missing... Can you point out where?

There is one more interesting feature of NFAs: you can have arrows not labelled with any symbol at all! For example:



All of these relaxed rules around arrows mean we have to be more careful with accepting strings. When you give a string to an NFA, several strange things might happen:

1. When the NFA reads a symbol, there might be no arrow to follow.
2. When the NFA reads a symbol, there might be multiple arrows to follow.

3. At any point, there might be one (or more) blank arrows to follow.

The first situation happens in the first NFA example, when trying to read the string b. There is no arrow coming out of state 0 labelled b. If you ever run into this situation, the NFA stops working and rejects the string.

The second situation happens in the second NFA example, when trying to read the string a, since there are two arrows coming out of state 0 labelled a. In such a situation, we let the NFA choose a path to follow. So it can either move into state 1 and accept the string, or into state 2 and reject it. This is where NFAs stop behaving like the computers we're familiar with – they aren't able to make choices.

The last situation happens in the third NFA example, when trying to read any string. Here, the NFA can move into either state 1 or state 4 from state 0 without reading any symbols. For example, let's say we input bb into the machine. The NFA starts in state 0, and can move to either 1 or 4, whichever it wants. Either way, its moves after that are completely determined. If it started by moving to state 1, it reads a b and goes to state 3, where it stays after reading another b. If it had moved to state 4 instead, it reads a b and goes to state 5, where it stays after reading the second b. In one situation, the string is accepted, and in the other, the string is rejected.

So now we have a dilemma. Sometimes, the NFA can make choices that lead to the string being accepted, and other choices lead to rejection. When should we say that the NFA accepts a string? The rule is: if it is *possible* for the NFA to accept the string (by *any* collection of choices), we say the string is accepted. Thus the second NFA accepts a and the third NFA accepts bb, even though it's possible for these NFAs to reject those strings if they make certain choices.

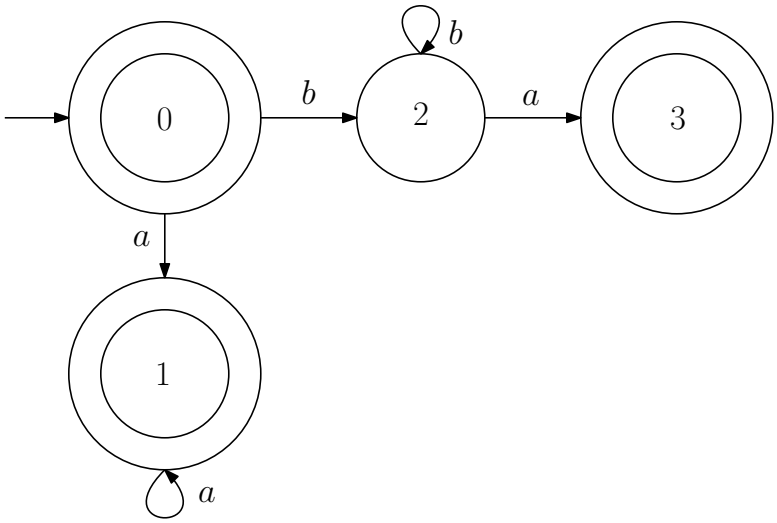
One way to think about it is in terms of "parallel universes": imagine that you gave the same string to the NFA in a bunch of different universes. In each universe, the NFA makes different choices when reading through the string, but every possible choice happens in some universe. If there is *any* universe where the NFA accepts the string, then the string belongs to the language defined by that NFA.

With that in mind, here's my next challenge for you: what are the languages accepted by the three NFAs we have just drawn?

The verdict? The first NFA accepts every string starting with a. The second NFA accepts only the string a (making it rather useless). The third NFA accepts all the strings that either start with a or end with b. If you don't see that right away, notice that as soon as one of the blank arrows is followed, the NFA must behave like a DFA. One of the branches is the DFA accepting all strings ending with b, and the other is the DFA accepting all strings starting with a.

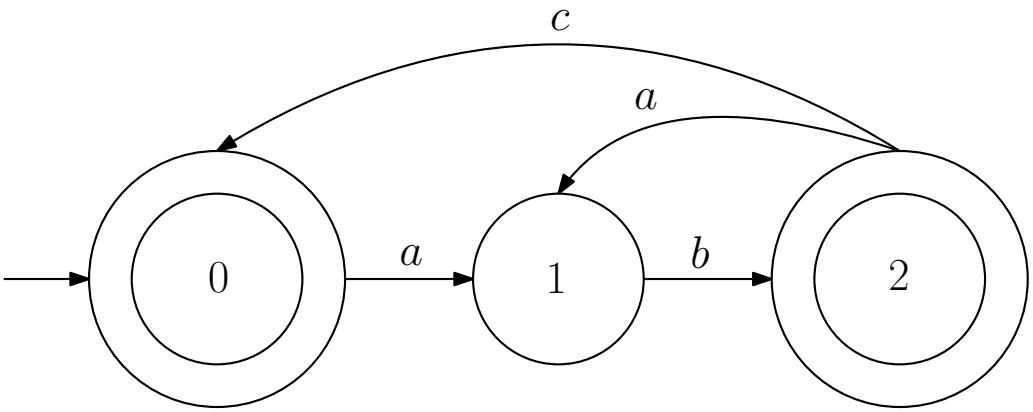
Okay, now let's give you a chance to build your own NFAs, accepting a certain language. First task: find an NFA accepting strings that have no bs, or else start with one or more bs, followed by a single a. Here's the catch: we want one with *exactly* four states.

Here's one possible answer:



Now, something trickier. Here, we're going to allow three symbols, a, b, and c. Build an NFA accepting the following language: it contains the empty string, and then gluing ab or abc to the end of a string in the language gives us another such string. So ab and abc are in the language, and so are $abab$, $ababc$, $abcab$, $abcabc$, and so on.

There's one more catch: we want an NFA with *three* states. You might have come up with this:



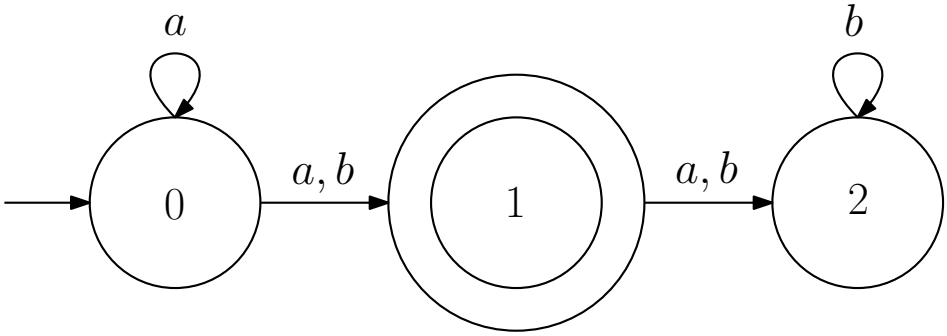
Now that you've had a chance to get familiar with building NFAs, we can start investigating the kind of languages that NFAs accept. We already know that any regular language (one accepted by a DFA) is also accepted by an NFA, since a DFA is already a special kind of NFA.

What Can NFAs Do?

But the million-dollar question is: have we gained anything new from this non-deterministic stuff? Are there languages out there that NFAs can accept but DFAs can't? What does everyone think? Do you have any gut instincts?

Contrary to what our gut says, it turns out that NFAs don't accept any new languages! Every language accepted by an NFA can be accepted by some DFA as well. The key lies in the fact that both types of machines have finitely many states. Remember the "parallel universe" discussion? When we want to know if a string is accepted by an NFA, we just give it to the machine in all possible universes, keep track of what happens in each one, and accept the string if it is accepted in any universe. The idea is to build a DFA that keeps track of exactly that information.

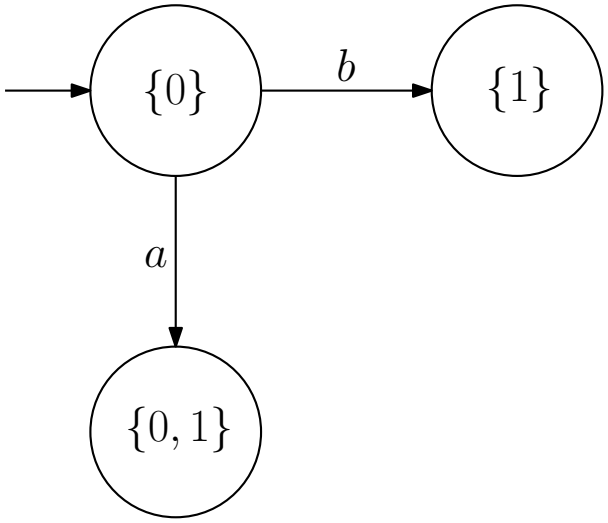
We'll do it together with one example, and then I'll let you try one of your own. Consider the following NFA:



We'll build a DFA that accepts exactly the same language using the parallel universe interpretation. In all possible universes, the NFA starts in state 0, the initial state. Therefore, the DFA modelling all the possible universes also starts in state 0. But we'll call the state $\{0\}$ instead; the numbers within the brackets keep track of all the possible states the NFA could be in.

All right, now we need to figure out what happens when some input comes in. Starting from state 0, if the NFA reads an a, it can move to either state 0 or state 1, but in all possible universes, it will be in one

of those two states. So we create a state $\{0,1\}$ in the DFA, and add an arrow labelled a from $\{0\}$ to $\{0,1\}$. On the other hand, if the NFA reads a b in state 0, it can only move to state 1. So, we create a state $\{1\}$ in the DFA, and an arrow from $\{0\}$ to $\{1\}$ labelled b. Here's what we have at this point:



Summarizing in a different way: we start in a situation where the NFA is in state 0 in all possible universes (represented by state $\{0\}$). In all these universes, if the NFA reads an a, it can be in either state 0 or 1 in any possible universe (represented by state $\{0,1\}$). On the other hand, in all these universes, if the NFA reads a b, it ends up in state 1 in all possible universes (represented by state $\{1\}$).

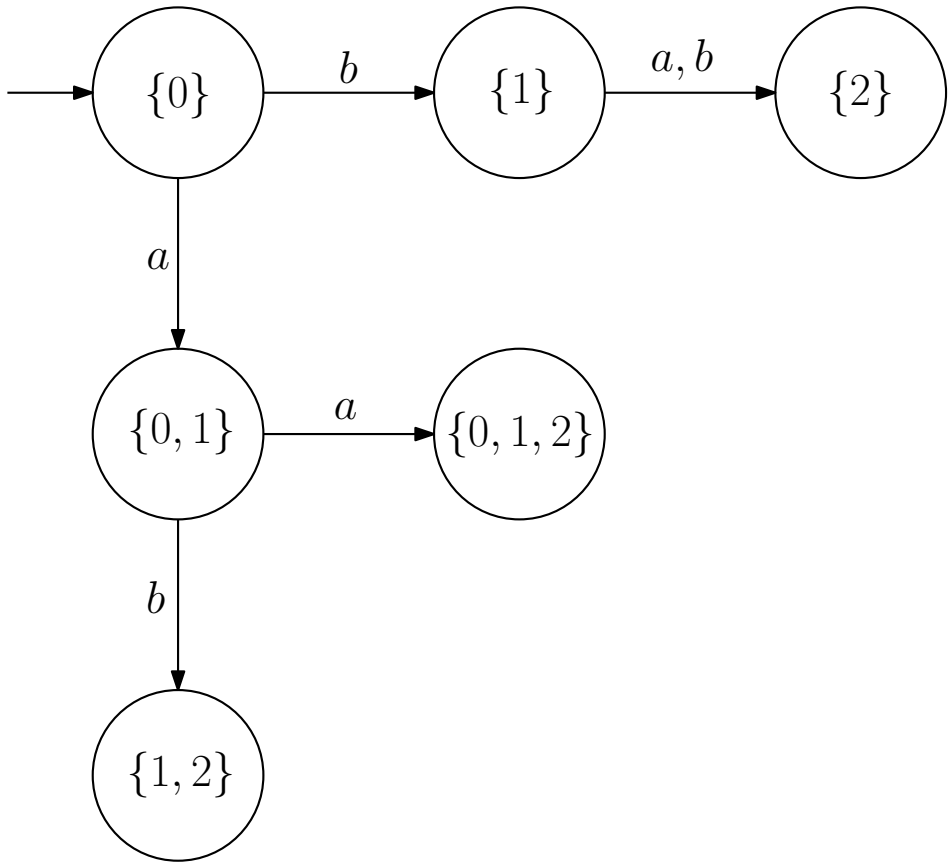
Okay, now it's your turn: we need some arrows coming out of state $\{1\}$. So suppose that the NFA is in state 1 in all possible universes, and reads an a. What states can it be in now?

Just state 2. In fact, the same thing goes if the NFA reads a b, in any possible universe. So we need a state $\{2\}$ in the DFA, and an arrow running from $\{1\}$ to $\{2\}$ labelled with a and b.

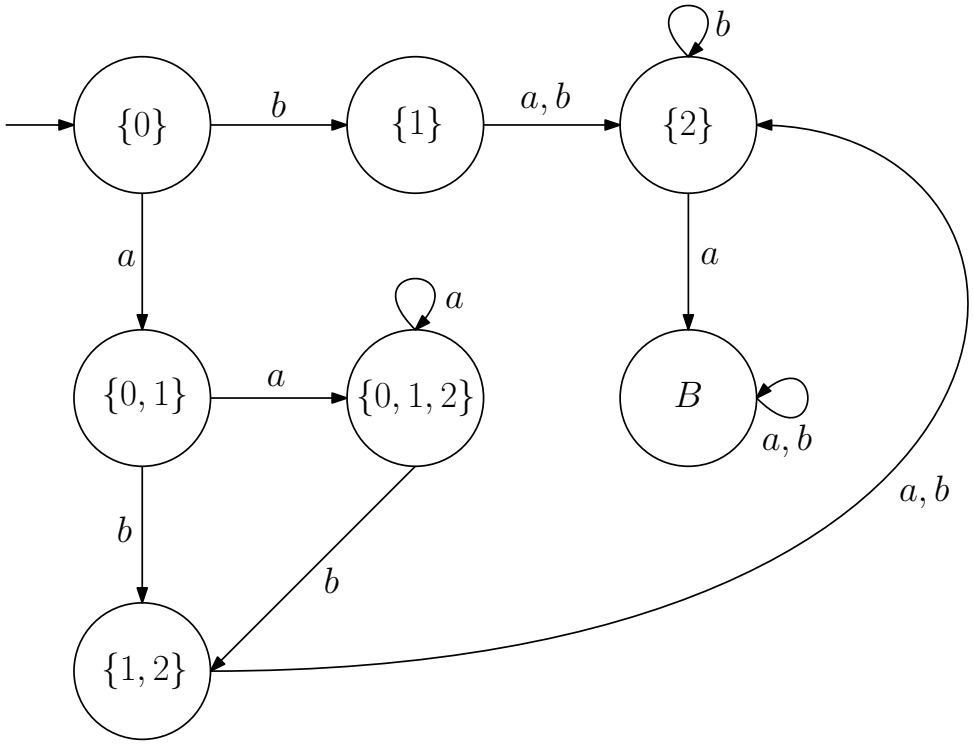
The arrows running out of $\{0,1\}$ are a bit more involved. Now the NFA can be in either state 0 or 1, and we need to figure out where it can be if it reads an a or a b. Let's say an a comes in. If the NFA was in state 0, it could either stay there or move to state 1. If the NFA was in state 1, it must move to state 2. So now we see that the NFA can be in any of the possible states! We create a state $\{0,1,2\}$ and draw an arrow labelled a from $\{0,1\}$ to $\{0,1,2\}$.

Now let's say a b comes in instead. The NFA started in either state 0 or state 1. If it's in state 0 and reads a b, it has to move to state 1. If it's in state 1 and reads a b, it has to move to state 2. So the only possible states it can be in now are 1 and 2. This means we create a state $\{1,2\}$ and an arrow labelled b from $\{0,1\}$ to $\{1,2\}$.

Here's what the picture looks like so far:



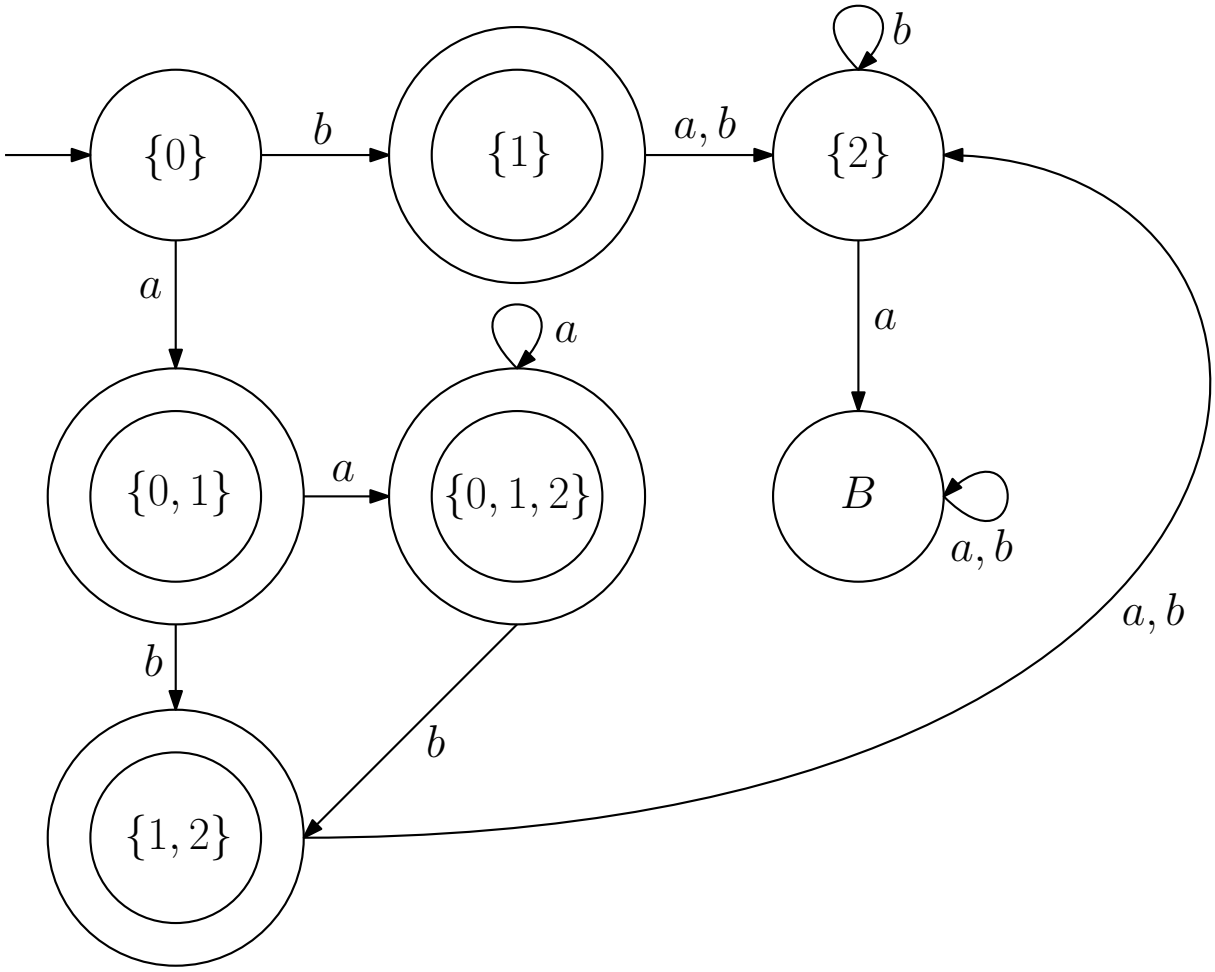
Do you think you get the idea? Take about five minutes to try and fill in the rest. Here's what I got at the end:



There's one part we should discuss. Note that there's a state called B, which stands for "broken". This represents the situation where the NFA breaks down in every possible universe (because it reads a symbol that it can't handle). This only happens when the NFA is in state 2 in every possible universe and reads an a. There's nothing it can possibly do with it, so it's forced to break down, and stays broken down no matter what other symbols come next (which is why B has an arrow labelled a, b pointing back to itself).

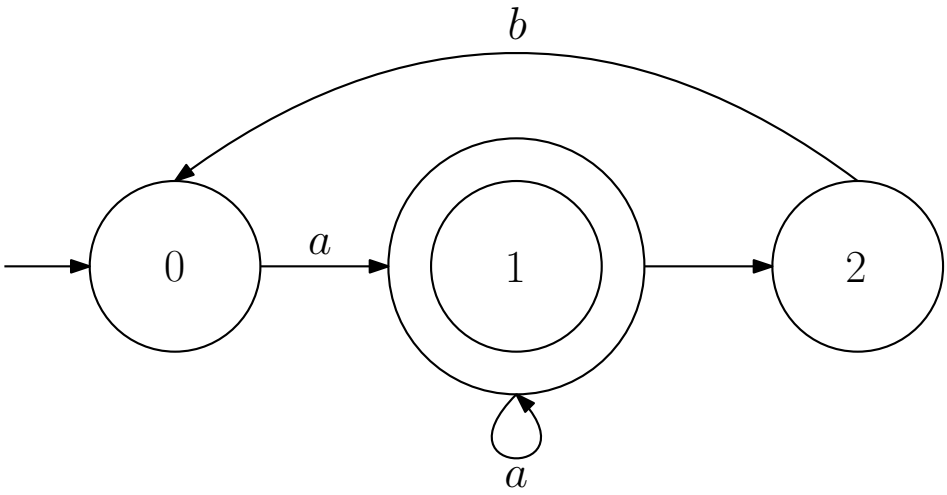
The rest is exactly as we discussed: each state of this DFA represents a collection of possible states the NFA can be in at any given time, and to figure out where to draw arrows, we consider what happens when the NFA starts in all those possible states and reads that letter.

The only thing missing is to decide what the accepting states of the new DFA should be. We want to accept a string if the NFA accepts it in some universe. In this case, state 1 is the only accepting state of the NFA, so the accepting states in the DFA should be all the ones that contain state 1 as a possibility. This gives us the final answer:



This DFA accepts the same language as our original NFA does. (For bonus points, what language is that?)

As our final activity, take the following NFA, and build a DFA that accepts the same language, using the same procedure we just mentioned:



All right, that's all we have time for! What did we do today? Knowing that there are limitations to what DFAs can do, we investigated what happens when you introduce non-determinism into the DFA model, giving us NFAs. Though we started with high hopes that this might allow us to accept more languages, it turns out any language accepted by an NFA is also accepted by a DFA. However, as you will explore on the question sheet, knowing that NFAs accept only regular languages gives us even more power to build new regular languages from old ones.

As you might guess, there is still lots to study in this area! If you are curious, take a look at *pushdown automata*: this is a variation on DFAs and NFAs that includes a *stack*, another form of memory. These machines are powerful enough to accept languages like the language of legal bracketings we mentioned before. However, even these machines have their limitations. The most powerful model of computing is the *Turing machine*, which formed the inspiration for the development of modern computers. Basically, whatever a Turing machine can do, a real computer can do as well!