Intermediate Math Circles
Wednesday, November 7, 2018
Finite Automata I

Nickolas Rollick – nrollick@uwaterloo.ca

Modelling Computers

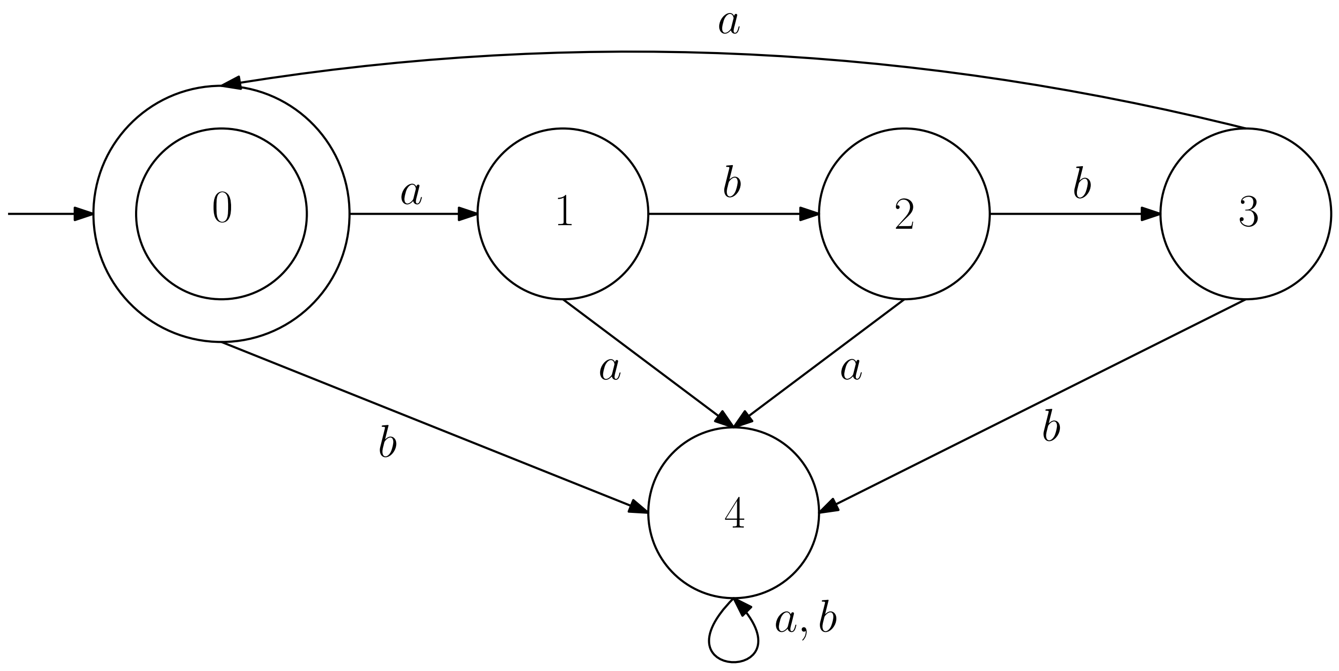
Imagine, if you can, a time before computers. At some point, the whole idea of a computer was just that – an idea in someone's mind. Before a computer could be built, mathematical models of computing had to be developed and studied. Over the next few weeks, we'll be looking at the simplest of these models, the *finite automata*. What's really cool about this topic is that it's pure problem-solving, the kind you can get your hands on right away.

Deterministic Finite Automata (DFAs)

What is the simplest kind of computer? You could argue that it's the old-fashioned punch-card machine. You feed the punchcard into the computer one row at a time, and the machine reads only one thing at a time. Every time the computer reads something, it clunks from one *state* to another. When it has finished reading the entire card, it will output either "yes" or "no", depending on whether the card is accepted or rejected.

This idea forms the essence of a *deterministic finite automaton*, or DFA for short. It's a machine with a finite number of states, with one of those states being the special *initial state*, where the machine starts. It reads a string of letters one at a time, and changes states according to what state it is currently in and what letter it is reading. At the end, the machine will accept or reject the string, depending on what state it finishes at.

The easiest way to explain is by example. This is a picture of a DFA:



This machine has five states, represented by circles in the picture, labelled 0 through 4. You'll notice that there's an arrow leading into state 0 from nowhere, which tells us the machine starts in state 0. You'll also notice that state 0 has a double circle, which tells us it's an *accepting state*. In other words, if the machine stops at state 0, it accepts the input (hence the double circle), and if it stops at any other state, it doesn't. A DFA can have more than one accepting state; it just so happens that this DFA has only one.

Every state in this DFA has two arrows coming out of it, one labelled a and one labelled b. Okay, technically state 4 has only one arrow coming out of it, but it really represents two arrows going to the same place. These arrows tell us how the machine changes states when it reads letters. So, for example, if the machine is in state 1 and it reads an a, it moves to state 4. If the machine is in state 1 and reads a b, it moves to state 2 instead.

It just so happens that this DFA only reads as and bs, but there are DFAs that take as, bs, and cs as input, or any number and combination of symbols whatsoever. The only rule is that there needs to be an arrow coming out of every state for every allowed symbol, so that the machine knows what to do when it reads each new symbol.

All right, let's see what happens when we give this DFA some input. Suppose we like sheep, so we feed *baa* into the machine. It starts in state 0 (because that's where the arrow from nowhere is), and the first thing we read is a b. So, the machine moves into state 4. Now it's in state 4 and reads an a, so where does it go? State 4 again. Finally, the machine reads another a, so what state does it finish in? State 4. This state does not have a double circle, so *baa* is rejected by the machine.

While tracing through this example, you might have noticed that state 4 is a *garbage state*. Once you're in state 4, you can never get out, no matter what symbols come in to the machine, and the input will be rejected.

Alright, now it's your turn. I'm a big fan of the band ABBA, so I'm going to give *abba* to the machine. What state does it start in again? What state is it in after the first a? After the first b? The second b? The final a? So, is *abba* accepted or rejected?

All right, time for something a little more interesting... Usually, we're not just interested in whether a single input (called a *string*) is accepted or rejected; we want to know *all* the strings that a given DFA accepts.

That's your challenge: in groups now, try and figure out exactly what strings this DFA accepts.

How did it go? It turns out that this DFA accepts what I like to call "the *abba* language". We already mentioned that *abba* is accepted, but so is *abbaabba*, and so is *abbaabbaabba*, and so on. It can get tiring to write out all those letters, so we'll usually just write $(abba)^2$ instead of *abbaabba*, and $(abba)^3$ instead of *abbaabbaabba*, and so on. It's pretty easy to convince yourself that anything of the form $(abba)^m$ gets accepted, where m can be any positive whole number.

But is that it? What happens if you put in any other string? You can convince yourself that if the string ever stops repeating *abba*, the machine ends up in the garbage state 4, and the string is rejected. Also, if the string fails to fully repeat *abba*, it will finish in one of states 1, 2, 3, so that the string is also rejected.

But there's one sticky point left: what happens if you run the machine with no input? After all, no input is a form of input... Will it accept or reject the "empty string"?

Great! So in conclusion, this DFA accepts exactly the strings where *abba* is repeated a certain number of times (possibly 0, giving the empty string). This collection of strings defines a *language*: the language defined by a DFA is exactly the strings that it accepts.

There – that's a DFA in action. In general, to build a DFA, we just have to follow these rules:

1. The machine must have at least one state (but only finitely many). These are represented by circles (single or double).

2. Some of the states are accepting states; these get double circles. The rest are rejecting states and get single circles.

3. The machine has exactly one starting state. This state should have an arrow coming into it from nowhere.

4. For every allowed symbol, every state should have exactly one arrow coming out of it labelled with that symbol.

Building DFAs

All right, enough rules! It's time to get some practice with building DFAs of your own. To keep things simple, let's assume the only input symbols are a and b.

First, let's do one together: let's build a DFA that accepts exactly the strings that start *and* end with b.

Ready for the next one? Construct a DFA that accepts exactly the strings that contain baa somewhere inside. So, for example, $ababaa$ should be accepted, and $bbaabbb$ works too, but not $baba$.

Now, for a really fun one... Can you find a DFA that accepts only the strings with exactly 2 as and more than 2 bs?

Let's go into the bonus round: find a DFA that accepts only strings where every run of as has length 2 or 3. In other words, the only strings we want are ones where the as always come 2 in a row or 3 in a row.

Conclusion

To recap, in today's session you learned about one of the most basic models of computing – the DFA. More importantly, you had practice with constructing DFAs accepting a given language (a given collection of strings). Next time, we will perform a much more careful investigation of the languages accepted by a DFA. In particular, you will think about how to take languages accepted by a DFA and use them to build new, related languages. Further down the road, we will investigate the limitations of a DFA, answering the question: are there any languages that cannot be accepted by some DFA?