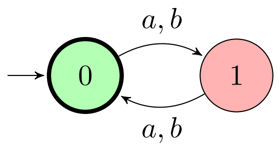
COSC341 Tutorial 4

The theme of this tutorial is to get some hands-on experience with the various concepts we've introduced so far. Unless otherwise specified use the alphabet $\Sigma = \{a, b\}$.

1. *The language* EVEN *consists of all strings of even length. Design a DFA that accepts it, a regular grammar that generates it, and a regular expression that describes it.*



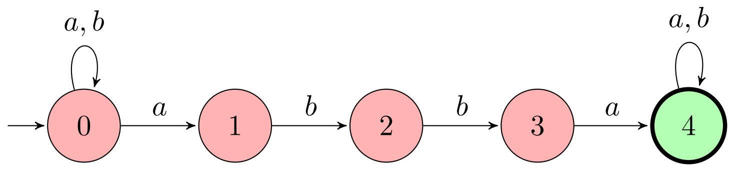
$$\begin{aligned} S &\to \epsilon \mid aO \mid bO \\ O &\to aS \mid bS. \end{aligned}$$

$$((a + b)(a + b))^* \,.$$

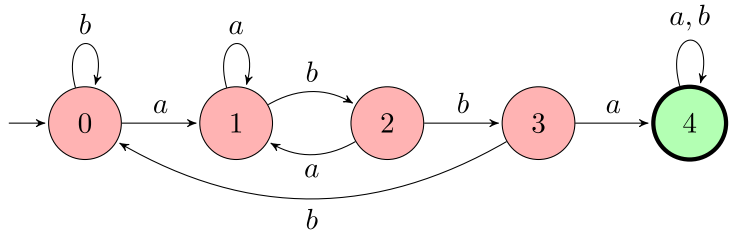
2. *The language* HAS-$abba$ *consists of all strings that contain* $abba$ *as a consecutive substring. Describe a simple regular expression for it, and a simple NFA that accepts it. Can you find a DFA that accepts it?*

$$(a + b)^* abba (a + b)^*.$$

For an NFA, the genie can just wait on the start state until it is sure you are about to type $abba$, then proceed along to the accepting state.

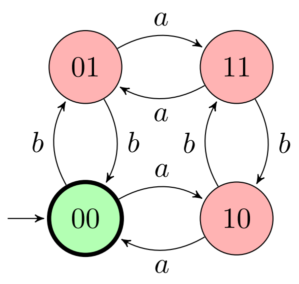


To do this in a DFA think of the state labels as the length of the prefix of $abba$ that we currently see at the end of the string. The "correct" next letters still move us along one, but we need to work out in detail what happens with "incorrect" letters.



3. *The language* EVEN-EVEN *consists of all strings containing both an even number of a's and an even number of b's. Design a DFA that accepts it, a regular grammar that generates it, and a regular expression that describes it.*

   The idea for the DFA is to use four states each of which codes the pair consisting of the parity of the number of a's seen so far, and the number of b's seen so far (using 0 for even and 1 for odd). That is:

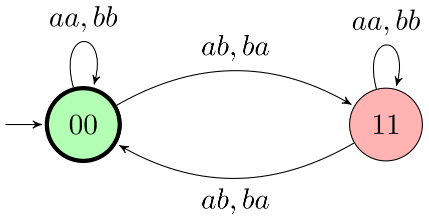


   Converting to a regular grammar is standard - introduce one non-terminal for each state and the corresponding rules arising from transitions.

   A regular expression is a bit more complicated. It's probably easiest to start with an answer and then to explain it:

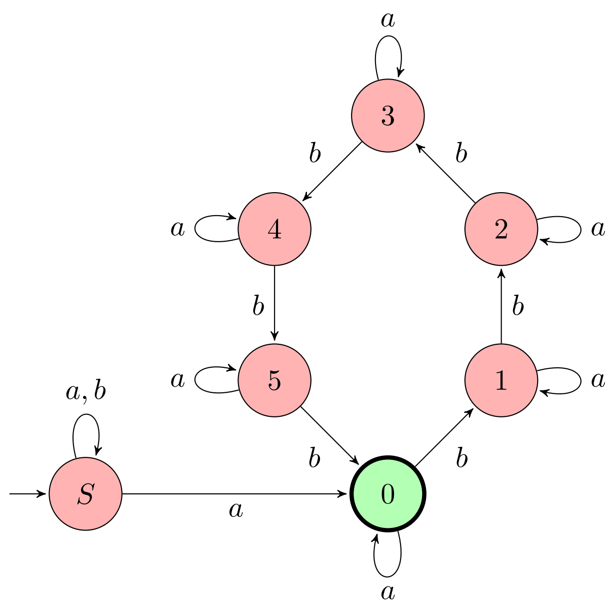
$$(aa + bb + (ab + ba)(aa + bb)^*(ab + ba))^* .$$

   The idea is this: we start at 00 and after two steps are either back there $(aa + bb)$ or have moved to 11 $(ab + ba)$. In the first case we are accepting, and we can just repeat this as often as we like. In the second case, we must at some point in the future return to 00 for the first time – that will be with an even number of symbols and some pairs (aa and bb) just bring us back to 11, but eventually we need an ab or ba to take us back to 00. Perhaps its simplest to think in terms of an automaton that takes four different sorts of letters (representing pairs of letters from the original alphabet). Now there are only two states:



4. *The language* MULTI-6-b *(I'm open to better names) consists of all strings with the following property: for some a in the string, the number of b's that follow it is a multiple of 6. Describe an NFA that accepts this language. Try to describe a DFA or regular expression for it.*

   To be clear the idea is the following. To check whether a string belongs to the language you look at each a in the string and count the number of b's to its right. If any of these numbers are multiples of 6 then the string is in, if not, it's out. The genie can identify the critical a (if there is one), and move over to a cycle of 6 states (the one it moves to is accepting) where each of these states has an a loop and b moves one step forward in the cycle.



Designing a DFA or regular expression seems much more difficult - we'll learn why in the next few lectures.

5. *Is there a DFA over the one-letter alphabet* $\{a\}$ *that accepts all, and only, those strings whose length is a power of two?*

   No – consider the structure of a one-letter DFA. Each state has a single outgoing arrow. Following the path from the start state that those transitions describe we must eventually return to some state that we've already visited (since there are only a finite number of states in the machine), forming a sort of lollipop shape. If there are no accepting states in the loop that describes the candy part of the lollipop then we only accept a finite set of strings so we don't accept all powers of 2. However, if there are accepting states in the lollipop then, if N is the length of the lollipop we find that eventually (once we're surely in the lollipop), whenever we accept x we also accept $x + N$. But, choose a power of 2 big enough to be sure that we're in the loop and bigger than the length of the loop – it's accepted but the next power of 2 is more than the length of the loop away, so the next thing we accept won't be a power of 2.