CS 4510 Automata and Complexity Jan 12

Lecture 2

Lecturer: Frederic Faulkner *Scribe(s):*

1 DFA's

In the same way that we simplify computers down to *models*, we also need simplify what we mean by a computational problem or task. In this class we will deal exclusively with *language membership*. (Recall that a language is just a set of strings.) In other words, every task we attempt, every problem we solve, and every function we compute will be formulated as the question "For some language L, and some string x, is $x \in L$?". Then we classify different languages L by how hard it is to to answer the question "Is x in L?".

Definition 1 *A language L is said to be regular if there exists some DFA D such that $L(D) = L$, i.e. D accepts L.*

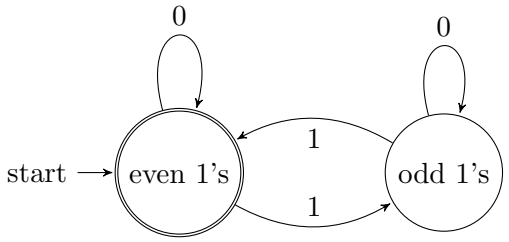
Thus, last class we saw two languages that are regular:

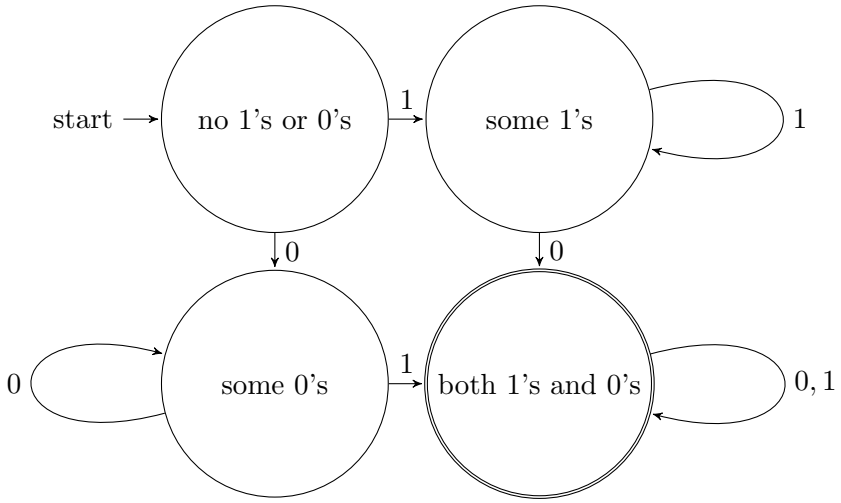
$$L_1 = \{x \mid x \text{ is a binary string with an even number of 1's}\}$$

$$L_2 = \{x \mid x \text{ is a binary string containing at least one 1 and one 0}\}$$

(Technically, last class we had the two languages "even number of vowels" and "strings containing at least one letter and one non-letter". The two languages shown here are roughly equivalent, but with input alphabet changed from "any character on a keyboard" to "just 0 or 1". The set $\{0, 1\}$ is the *binary alphabet*, and it is very popular in theoretical computer science.)

We know that L_1 and L_2 are regular, because they each have some DFA that accepts them, as seen below.





1.1 Brief aside about homework

Before continuing, let's discuss some things that are relevant to the homework. Firstly, when are specifying a DFA on the homework, there are multiple ways you can do so. You can draw a state diagram like the ones above, you can draw a table like in lecture 1, or you can even just write a paragraph of English text. HOWEVER! Make sure that all 5 parts of a DFA are present, implicitly or explicitly, in your answer (states, alphabet, start state, final states, transition function). You do not have to list them out one by one, but the information should be present somewhere.

In other words, the diagrams above would be sufficient for full points on the homework because all five parts of a DFA are present. On the other hand, the following table from last week is missing some information, namely there is no indication of which states are starting or accepting:

input / state	eventHappened=1	eventHappened=0
isEven=True	isEven=False	isEven=True
isEven=False	isEven=True	isEven=False

Also, here is a piece of advice when designing DFA's: you do not need to write pseudocode before designing a DFA (although I think it can be helpful). However, you should do your best to give your states meaningful names. With clear names, either the transition function almost writes itself, or it becomes clear that you need to choose different states.

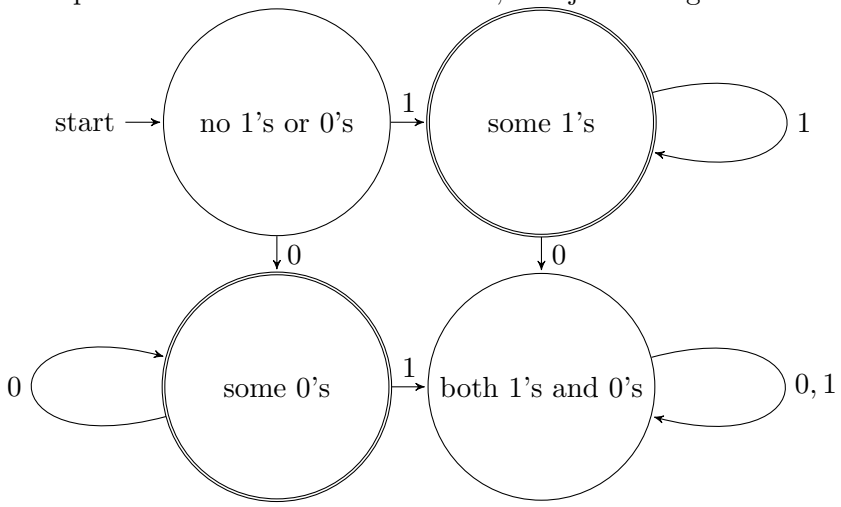
1.2 Back to DFA's

Consider the following language:

$$L_3 = \{x \mid x \text{ does not contain both 0's and 1's}\}$$

.

To handle this language, we need to keep track of whether or not we have already seen a 0 and/or a 1. However, this is the same thing we had to keep track of for L_2! So we can in fact keep the same states and transitions, and just change the final states like so:

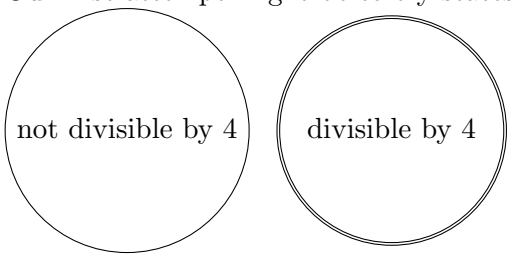


1.3 Length mod 4

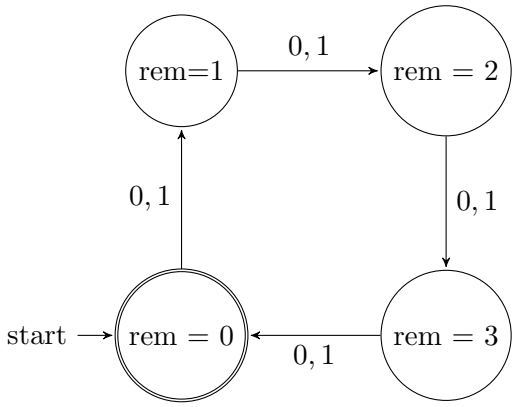
How would you prove that the following language is regular?

$$L_3 = \{x | x \text{ has length divisible by } 4\}$$

Our first attempt might be to try states like the following:



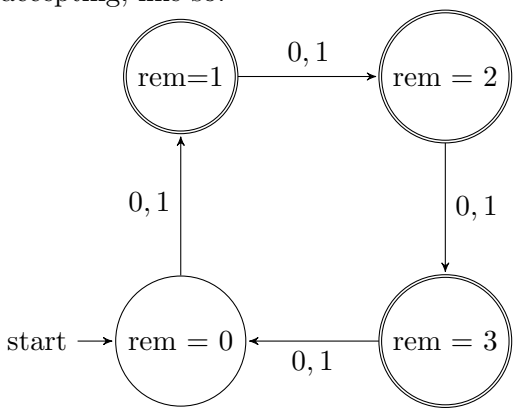
However, when it comes time to connect them, we run into an issue. If you add 1 to number divisible by 4, clearly it is no longer divisible by 4. However, if you add 1 to a number that is not divisible by 4, now what? 2+1 = 3 (not divisible by 4), but 3+1=4(divisible by 4). Clearly our states are not tracking all of the relevant information! Rather than proceeding to add states at random, let's stop and rethink our approach. What we really need to keep track of is the *remainder* mod 4. That leads to the following machine:



Now consider the following language:

$$L_4 = \{x | \text{ the length of } x \text{ is not divisible by } 4\}$$

Again, we need to keep track of the remainder mod 4, just like before. In this case we know that if we are in state "rem=0" we are divisible by 4. Therefore, if we are in *any other state*, we know that we are NOT divisible by 4. So we can just invert which states are accepting, like so:



This naturally leads to the following theorem:

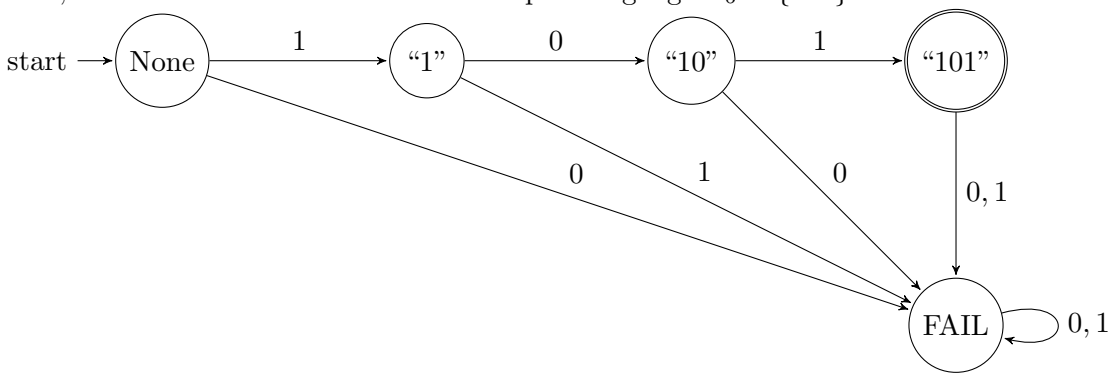
Theorem 2 *Let L be any regular language. Then $\overline{L}$ (the compliment of L) is also a regular language.*

Proof. Let $D = (Q, \Sigma, q_0, F, \delta)$ be a DFA for L (which we know exists, because L is regular, and all regular languages have DFA's). Then let $D' = (Q, \Sigma, q_0, Q - F, \delta)$. (In other words, swap the accepting and non-accepting states.) D' clearly accepts exactly those strings which are rejected (and rejects the strings that are accepted) by D, since for any input x, D and D' end in the same state, which means one will reject and the other will accept. Since we have given a DFA for $\overline{L}$, $\overline{L}$ is regular.

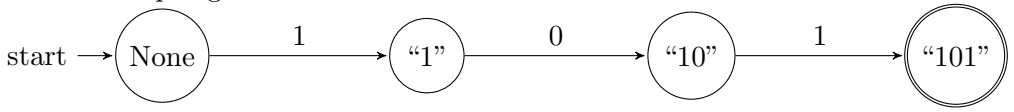
Now onto some harder languages. Let's prove that the following language is regular:

$$L_5 = \{x | x \text{ is a binary string ending in } 101\}$$

First, consider the DFA for the much simpler language $L_6 = \{101\}$:

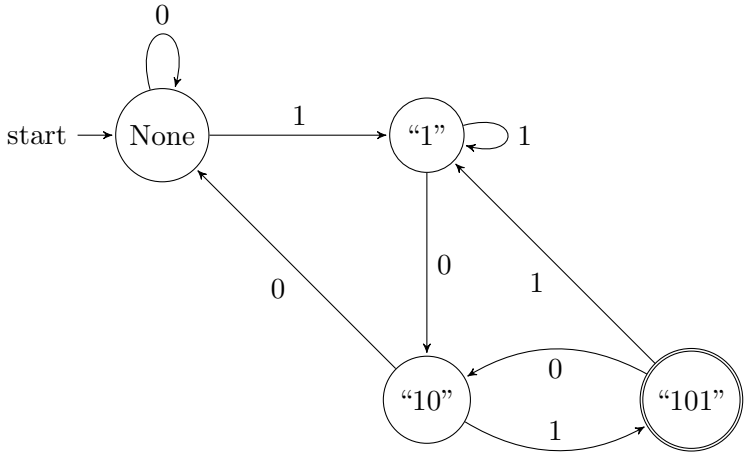


Naturally, for L_5, it feels natural to keep the top part of the above DFA, to track our progress for the suffix 101, however, we no longer need a permanent fail state, since no matter what we have read so far, if the next three letters are "101" we need to be able to reach some accepting state. So let's start there.



So the question is, what is the rest of the transition function, i.e. where do we go when we see an input that isn't "the right one", so to speak.

For example, suppose that the last three characters we have read are in fact 101, but then we read another 0. What do we do? We do not have to throw away all our progress and return to the beginning. After all, the string now ends in 1010, and in particular, it ends in 10, so we should be in the third state. Similar if we had see a 1 instead, the end of the string would look like 1011, which means we still are partway there, because we end in a 1, so we would return the second state. The full machine then looks like so:

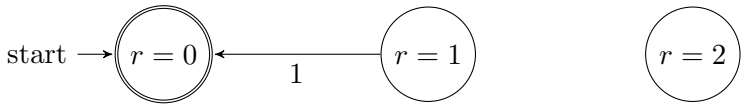


Let's try one more. Consider the following language:

$$L_7 = \{x | x \text{ is divisible by 3 when interpreted as a binary number}\}$$

In other words, L_7 contains strings like $11, 110, 1001, 1100$, etc. since these are binary representations of $3, 6, 9$ and 12.

Just like before, we want our states to represent the possible remainders mod 3. However, it might not be obvious that we can have a consistent transition function. But we can! Suppose that as we are processing the string, so far we have just read a bunch of characters, and resulting number has remainder 1 mod 3. Then we read another "1" as input. For example, maybe we had just read 110111 (=55), but now we have 1101111. Notice what just happened: we shifted our original number to the right, then added a 1. But this is the same as multiplying the original number by 2 and adding 1. So, by the rules of modular arithmetic, if the original remainder was 1, the new remainder is $1 \times 2 + 1 = 3 \equiv 0$. Thus we have done the first transition for our machine!



Here is the full machine: