COMS W3261: Computer Science Theory.
Instructor: Tal Malkin

DFAs and NFAs: Example from Class

This note summarizes what we have seen in class towards the end of lecture 3 (as well as a brief discussion at the end of lecture 2), with respect to one example, demonstrating the power of NFAs (and of DFAs). The note is detailed, recapping and expanding on the discussion and intuition. This is meant to help understanding, based on some student questions (not to overwhelm you with a long note!).

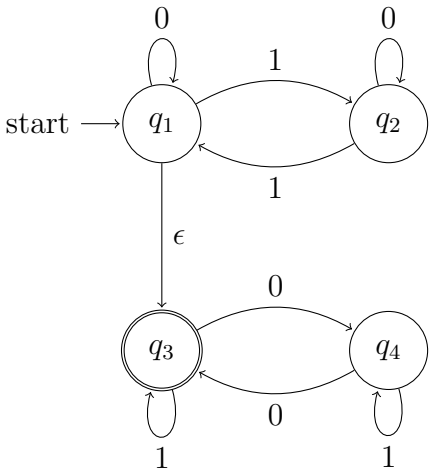
Consider the following language over the alphabet $\Sigma = \{0, 1\}$:

$$L = \{xy \mid x \text{ has an even number of 1s, } y \text{ has an even number of 0s}\}$$

Discussion. We saw this example at the end of lecture 2, and discussed whether or not this language is regular. Based on what we learned up to that point (just that a language is regular iff there exists a DFA that recognizes it), this was hard to answer. It seemed hard to come up with a DFA, because as we read the input from left to right we can't tell where we should parse the input (where x ends and y begins), and if we try one parsing and it doesn't work, we cannot go back on the input and try again (the input symbols have been consumed). On the other hand, we don't yet know how to prove a language is not regular, and perhaps there's some clever way to build a DFA for this language after all, some trick that will allow you, using only finitely many states, to figure out whether the input word can be parsed in this way? It turns out the answer is yes. But figuring it out with just the definition of DFAs requires some creativity and intuition (coming up with the right approach and solution), and mathematical sophistication (proving that it works). In contrast, as we saw in class and summarize below, coming up with an NFA is much easier, and applying the subset construction can be done systematically without requiring creativity. This is one way that NFAs are helpful, even though ultimately they have the same computational power as DFAs (recognizing regular languages).

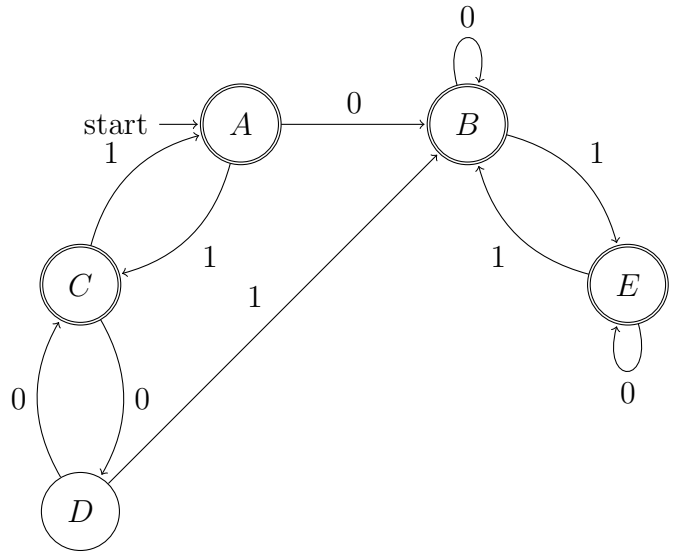
An NFA for this language

In lecture 3 we defined NFAs. Based on the definition of an NFA computation, it was relatively easy to come up with an NFA for this language, and the students suggested the following NFA:



Transforming the NFA to a DFA for this language

We also proved in class that for any language recognized by an NFA, there is a DFA recognizing the same language (namely, the language is regular). This is proved constructively, via the subset construction. Applying this subset construction to the NFA above, resulted in the following DFA:

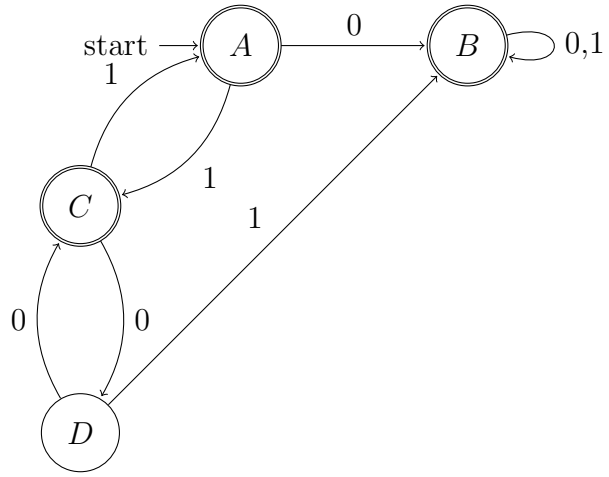


Recall that we obtained this DFA by constructing states where each state in the DFA corresponds to a subset of states in the NFA:

- A corresponds to $\{q_1, q_3\}$
- B corresponds to $\{q_1, q_3, q_4\}$
- C corresponds to $\{q_2, q_3\}$
- D corresponds to $\{q_2, q_4\}$
- E corresponds to $\{q_2, q_3, q_4\}$

We could in general have additional states in the DFA corresponding to all other subsets of states of the NFA (in this case, having 16 DFA states $\{A, \ldots, P\}$), but as we were constructing this DFA, only 5 such states were reachable from the start state (after adding $\{A, \ldots, E\}$ there were no transitions that need to go to new states not previously encountered). Recall that transitions were determined based on imagining all possible ways to perform the transition in the NFA, examining what possible subset of states in the NFA this would lead to, and then transitioning to the (unique) state in the DFA that corresponds to that subset. The accepting states are all states corresponding to a subset containing at least one accepting state from the NFA. In this case the only accepting state in the NFA is q_3, and thus states A, B, C, E in the NFA are accepting.

Looking at this DFA, we may notice that we can collapse the states B, E to the same state, since once you get to B your string will be accepted no matter what comes next. This gives the following smaller DFA:



We have explained above how we came up with this DFA. However, even without understanding how we got there, you can easily verify that the above is indeed a DFA, and you can try running it on various strings and checking that it indeed gives the correct output with respect to our example language.

You can further examine this DFA to understand the language better. For example, it is apparent from the DFA that any string that starts with 0 is accepted – can you prove that indeed any string that starts with 0 can be parsed as xy satisfying the required properties?

As mentioned above, one could come up with the above DFA directly from the language (and some students – at least last year – did). But going through an NFA is easier.