COMS W3261: Computer Science Theory.
Instructor: Tal Malkin

Handout 2: Example from Class – the Subset Construction

In class, we considered the following language over the alphabet $\Sigma = \{0, 1\}$:

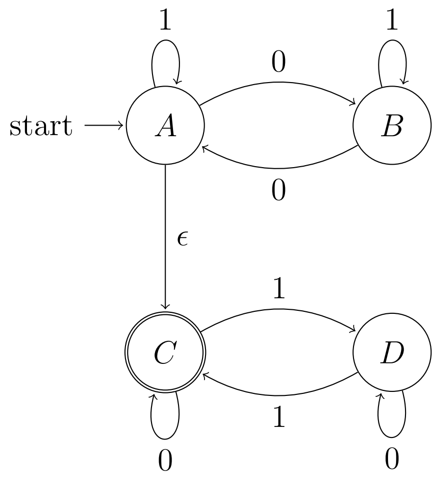
$$L = \{xy \mid x \text{ has an even number of 0s, } y \text{ has an even number of 1s}\}$$

The first time we saw this example, we had just defined DFAs and regular langauges, and asked whether this language is regular. Based on what we learned up to that point, answering this question was hard. Coming up with a DFA seems hard, 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 – a DFA can't go back). 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). Some students had indeed succeeded to do that (impressive!) and we discuss this towards the end of this note. In contrast, 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).

In this note we show how to transform the NFA we saw in clsas for this language to a DFA, serving as another example of applying the subset construction.

An NFA for this language

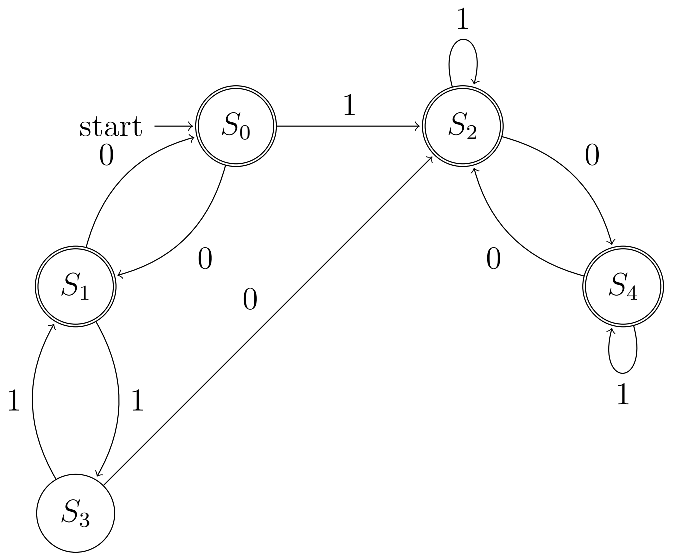
Here's an NFA we saw in class for this language. The intuition was that for this NFA, the ε-transition can be taken at most once in any possible computation path on a string, and that corresponds to where we parse the string. The computation path ends in an accepting state if that parsing works. A string is in the laguange if there exists some parsing that works, namely if there exists some accepting computation path of this NFA. But this is exactly the definition of NFA acceptance, so the NFA recognizes this language.



Transforming the NFA to a DFA for this language

We 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. In this construction, each state in the DFA corresponds to a subset of states of the NFA. Transitions are 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. We now apply the subset construction to the NFA above, to get a DFA.

The start state, which we will denote by S_0, corresponds to $\{A, C\}$, since in the NFA the start state is A, and C is ε-reachable from it. Now we must add one transition labeled 0 and one transition labeled 1 going out from S_0. In the NFA, a 0 transition from A goes to B, and a 0 transition from C goes to C, so therefore in the DFA, a 0 transition from $S_0 = \{A, C\}$ would go to a new state $S_1 = \{B, C\}$. In the NFA, a 1-transition from A goes to A, and then might also take a ε-transition to reach C, and a 1 transition from C goes to D, so in the DFA, a 1 transition from S_0 would go to a new state $S_2 = \{A, C, D\}$. We continue to add 0 and 1 transitions from every new state, until there are no more states to add. This results in the following DFA (check that you understand why):



where the correspondence between DFA states and subsets of NFA states is as follows:

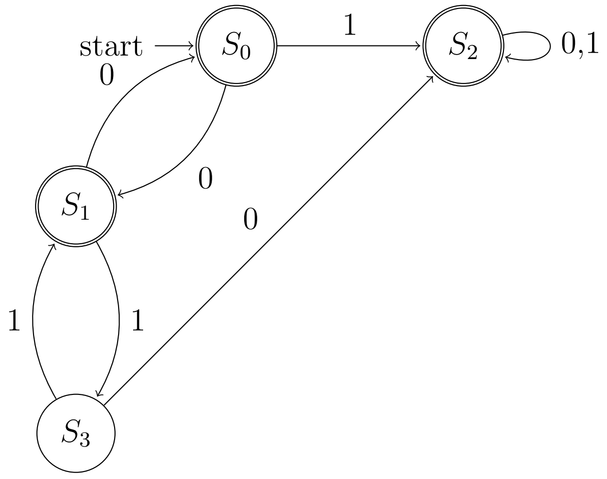
- S_0 corresponds to $\{A, C\}$
- S_1 corresponds to $\{B, C\}$
- S_2 corresponds to $\{A, C, D\}$
- S_3 corresponds to $\{B, D\}$
- S_4 corresponds to $\{B, C, D\}$

The accepting states are all those corresponding to a subset containing at least one accepting state from the NFA. In this case the only accepting state in the NFA is C, and so we mark S_0, S_1, S_2, S_4 as accepting.

Note that 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). But as we were constructing this DFA, only 5 such states were reachable from the start state (after adding $\{S_0, \ldots, S_4\}$ there were no transitions that need to go to new states not previously encountered).

Bonus: Understanding and Improving this DFA

[The following is not part of the class material]. By looking at the above DFA, we may notice that we can collapse the states S_2, S_4 to the same state, since once you get to S_2 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 1 is accepted – can you prove that indeed any string that starts with 1 can be parsed as xy satisfying the required properties?

As mentioned above, it is possible (but hard) to come up with the above DFA directly from the langauge. Here's one possible explanation of how (or the meaning of each state). First, notice that if a string has either an even number of 0s or an even number of 1s, then it will be in the language since we can take y or x to be the empty string, respectively. So the only strings which may possibly be rejected are those with an odd number of zeros and an odd number of ones. Furthermore, if a string w has a prefix which has an even number of 0s and an odd number of 1s, then $w \in L$. Indeed, if such a w has an even number of 1s or 0s then it will be accepted and otherwise it has an odd number of 1s and 0s. In that case if we take x to be the prefix with an even number of 0s and odd number of 1s, then y will have an even number of 1s, so w is in the language. Conversely, if a string w with an odd number of 0s and 1s has a parsing as xy that puts it in the language, it must be the case that x has an even number of 0s (by definition) and an odd number of 1s (since y has an even number of 1s), so that this condition characterizes the language (it is an if and only if).

With this in mind, the state diagram becomes clearer. Each state corresponds to the parity of each symbol in the string read so far: state S_0 refers to even 0s and even 1s, state S_1 is odd 0s and even 1s, state S_3 is odd 1s and odd 0s, and state S_2 is even 0s and odd 1s, at which point we don't care what the rest of the string is, it will be accepted.

Finally, we note that there is a systematic way to minimize any DFA to the best (smallest) possible one for the language. It can be proven that the above DFA is indeed minimal for our example langauge. We will discuss this later.