COMS W3261: Computer Science Theory.
Instructor: Tal Malkin

Handout 5: Example from Class (Subset Construction, and more)

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

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

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). So at that point in class, this could seem like some intuition that the language may not be regular. On the other hand, we didn'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 – the language is regular. Figuring it out with just the definition of DFAs is possible, but requires some creativity and intuition (coming up with the right approach and solution), and mathematical sophistication (proving that it works).[1] In contrast, coming up with an NFA is much easier, and from there we can get a DFA by applying the subset construction, which can be done systematically without requiring creativity. This is one way that different characterizations of regular languages (such as via NFAs) are helpful, even though ultimately they are equivalent to (have the same computational power as) DFAs.

In this note we show several ways to prove L is regular and to construct a DFA for it, along the way practicing some of what we learned in class, such as the subset construction.

1 Proof of regularity using closure

From the definition of the L (and the definition of concatenation of languages), we see that L can be expressed as a concatenation of two languages:

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

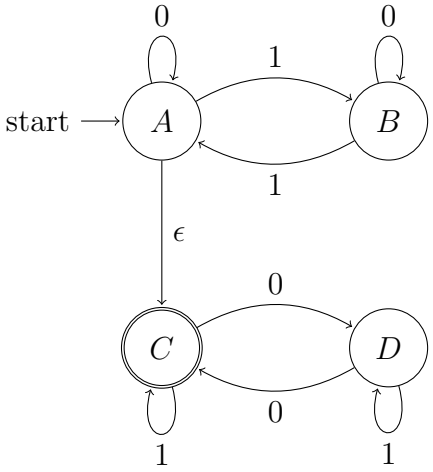
It is easy to show that each of the underlying languages is regular (we've seen this in class several times). Since we proved that the class of regular languages is closed under concatenation, we can conclude that L is regular.

If we further want to actually create a DFA for L, we could do it by first constructing a DFA for each of the underlying languages, then using the construction for an NFA recognizing the concatenation of two regular languages, and then applying the subset construction. This is essentially what we do in the next sections.

[1] See Section 4 for a characterization of strings in the language –one coud potentially prove this characterization directy from the language definition.

2 An NFA for this language

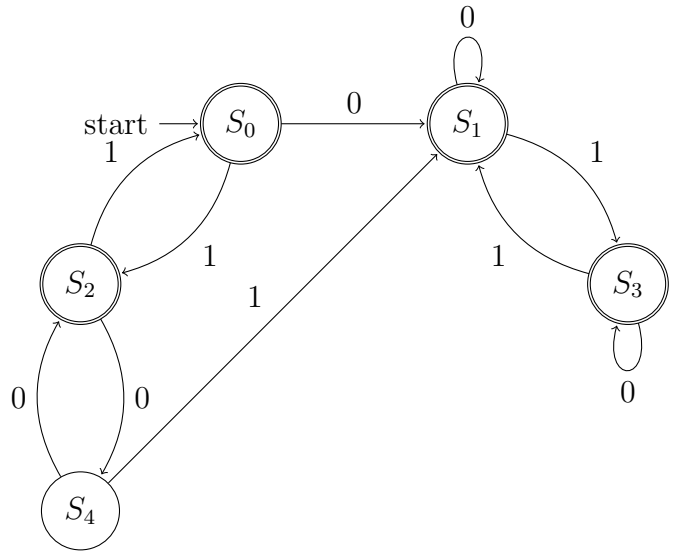
We saw the following NFA for this language in class – it's the same NFA you get by applying the construction for closure under concatenation (as just mentioned above), but you can also directly construct the NFA (we saw it in class before we even defined concatenation of languages). Our informal correctness proof 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 and only if there exists some parsing that works, namely if and only if there exists some accepting computation path of this NFA. But this is exactly the definition of NFA acceptance, so the NFA recognizes L.



3 From NFA to DFA: Subset Construction

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, that transforms an NFA to an equivalent DFA. 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 A, and then might also take a ε-transition to reach C, and a 0 transition from C goes to D. Therefore, in the DFA, a 0 transition from $S_0 = \{A, C\}$ goes to a new state $S_1 = \{A, C, D\}$. Next, in the NFA, a 1 transition from A goes to B and a 1 transition from C goes to C, so therefore in the DFA, a 1 transition from $S_0 = \{A, C\}$ will go to a new state $S_2 = \{B, C\}$ We continue to add 0 and 1 transitions from every new state, until there are no more states to add. The resulting DFA is the following.



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

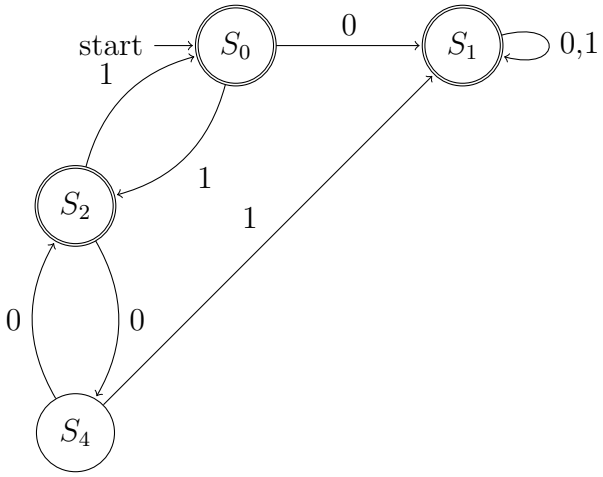
- S_0 corresponds to $\{A, C\}$
- S_1 corresponds to $\{A, C, D\}$
- S_2 corresponds to $\{B, C\}$
- S_3 corresponds to $\{B, C, D\}$
- S_4 corresponds to $\{B, D\}$

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_3 as accepting, while S_4 is not.

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).

4 Bonus: Understanding and Improving this DFA

[This section is not part of the required class material]. By looking at the above DFA, we may notice that we can collapse the states S_1, S_3 to the same state, since once you get to either of them, 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, it is possible (but hard) to come up with the above DFA directly from the language. Here's one possible explanation of how (or the meaning of each state). First, notice that if a string w has an even number of 0s then it is in the language, since we can parse it into xy where $x = \varepsilon$ and $y = w$.[2] Next, we focus on strings w with an odd number of 0s. We claim that such a string is in the language if and only if it has a prefix with an even number of 1s and an odd number of 0s. Indeed. if w with an odd number of 0s has such a prefix, we can write $w = xy$ where x is that prefix and y to be the rest of the string. The number of 1s in x is even, and the number of 0s in y is the total number in w minus the number of 0s in x, namely odd minus odd, which is even. Hence this string is in L. Conversely, if a string w with an odd number of 0s has a parsing $w = xy$ that puts it in the language, the prefix x has an even number of 1s (by definition), and a number of 0s that is the total in w minus the number of 0s in y, namely odd minus even, which is odd. Overall, we proved that a string is in the language if and only if it has an even number of 0s, or it has a prefix with an even number of 1s and an odd number of 0s.

With this in mind, the state diagram becomes clearer. Each state corresponds to the parity of each symbol in the string read so far, except that the first time we get to even 1s and odd 0s we don't care anymore about the rest of the string, it will be accepted. Specifically, state S_1 corresponds to any string which has a prefix where the parity of the number of 0/1 is odd/even respectively, and the rest of the states correspond to strings without such a prefix, where the parity of the number of 0/1s is even/even (state S_0), even/odd (state S_2), and odd/odd (state S_4).

It can be shown that these four types of strings exactly correspond to the four equivalence classes of $\sim_L$ from the Myhill-Nerode theorem (see handout 3). However, even without figuring this out, you can still use the Myhill-Nerode theorem to prove that the above 4-state DFA is the smallest possible one for this language, as we show next.

[2] A similar argument shows that a string with an even number of 1s is also in the language, but we don't need this fact here, it suffices to check the cases of even 0s and odd 0s.

Applying the Myhill-Nerode theorem to show minimality

To show that the above 4-state DFA is minimal, we need to show four different strings that are all in different equivalence classes under $\sim_L$. We take one string that ends up in each of the states, and then prove that any pair of these strings has a distinguishing extension (namely no two states are equivalent) – using the DFA as a guide makes it easier to identify such extensions. Concretely, let's take the following four strings:

$$\{\varepsilon, 0, 1, 10\}$$

and the following distinguishing extensions:

- For 10 and any of the other three strings: take the extension ε (check $10 \notin L$ while the other strings are in L)
- For $\varepsilon, 0$: take 10 (check $10 \notin L$ while $010 \in L$)
- For $\varepsilon, 1$: take 0 (check $0 \in L$ while $10 \notin L$)
- For $0, 1$: take 0 (check $00 \in L$ while $10 \notin L$)

We have shown four distinguishable strings, so we know any DFA for L must have at least four states. Since we already have a 4-state DFA, we know it is minimal.

5 Regular Expression for this language

We can also use regular expressions to prove that the language L is regular:

- Regular expression generating $\{x \mid x \text{ has an even number of 1s}\}$: $0^*(10^*10^*)^*$
- Regular expression generating $\{y \mid y \text{ has an even number of 0s}\}$: $1^*(01^*01^*)^*$

Since L is the concatenation of the above two languages, a regular expression generating L is:

$$0^*(10^*10^*)^*1^*(01^*01^*)^*$$

Of course, there are other possible regular expressions for L. If you want to translate the above to a DFA for L, you can first construct an NFA that is equivalent to this regular expression, using the constructions we saw in class when we proved closure of regular languages under the regular operations, and then apply the subset construction to get a DFA. What you get using these constructions is too large to include here.