CS208 Tutorial 3: Finite Automata Theory

---

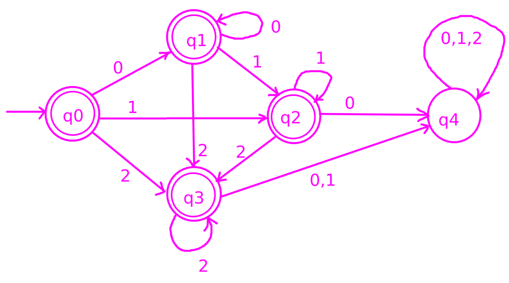
1. Often, we are required to find ways to accept sets of strings with various properties. We've seen in class that DFAs and NFAs are ways to achieve this, with an equivalence between NFAs and DFAs. In this question and the next, we'll specify some properties of sets of strings and you will be required to construct (small) DFAs/NFAs to accept these sets (or languages). We aren't insisting that you should find the smallest (in terms of number of states) automata, but try to use as few states as you can.

   Let $\Sigma = \{0, 1, 2\}$. Construct DFAs for recognizing each of the following languages.

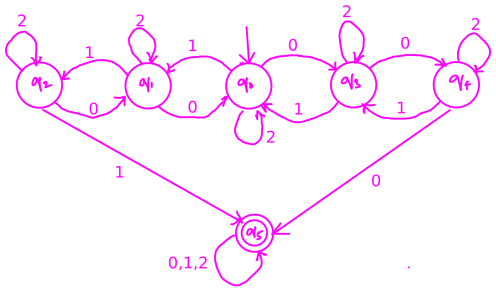
   1. $L_1 = \{w \in \Sigma^* \mid w$ doesn't have any pair of consecutive decreasing letters (numbers)$\}$. For example $010 \notin L$ but $00012 \in L$ and $222 \in L$.
   2. $L_2 = \{w \in \Sigma^* \mid w = u.v,\ u \in \Sigma^+,\ v \in \Sigma^*,\ \ n_0(u) > n_1(u) + 2$ or $n_1(u) > n_0(u) + 2\}$, where $n_i(u)$ denotes the count of i's in u, for $i \in \{0, 1\}$. For example, $0010221020012 \in L$ but $012012012 \notin L$.
   3. $L_3 = \{0^n \mid n > 0, n^3 + n^2 + n + 1 = 0 \mod 3\}$

Solution:

Possible DFAs are given below:

1. 

q_1, q_2, q_3 remember that we've not seen any decreasing sequence of letters so far, and the last letter seen was 0, 1, 2, respectively. All of these strings are accepted. q_4 remembers that we have seen a decreasing sequence, and hence anything we see subsequently cannot make us accept the string. So q_4 is a sink or trap state – once you reach there, you can't escape it.

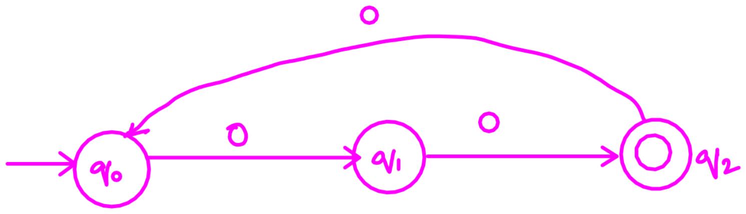
2. 

For a prefix u of w seen so far, we need to remember if $n_0(u) - n_1(u)$ is $0, 1, -1, 2, -2$ or greater than 2 or less than -2. Once the prefix u satisfies $n_0(u) - n_1(u) > 2$ or $n_0(u) - n_1(u) < -2$, we can immediately accept the word, regardless of what subsequent letters are seen. States q_0, q_1, q_2, q_3, q_4 remember if $n_0(u) - n_1(u) = 0, -1, -2, 1, 2$ respectively, where u is the (prefix of the) input word seen so far. State q_5 simply remembers whether $|n_0(u) - n_1(u)| > 2$.

3. Notice that $n^3 + n^2 + n + 1 = 0 \mod 3$ is equivalent to $(n^2+1) \cdot (n+1) = 0 \mod 3$. Therefore, $\big((n^2+1) \mod 3\big) \cdot \big((n+1) \mod 3\big)$ has to be equal to 0.

   Given that $n \mod 3 \in \{0, 1, 2\}$, it follows that we require $n^2 = 2 \mod 3$ or $n = 2 \mod 3$. However, for no n is $n^2 = 2 \mod 3$. Why so? Because this would require $\big((n$

$\bmod 3) \times (n \bmod 3)) = 2 \bmod 3$. Since $n \bmod 3 \in \{0,1,2\}$, $(n \bmod 3) \times (n \bmod 3)$ can only be in $\{0,1\} \bmod 3$. Therefore, we must have $n = 2 \bmod 3$. The DFA is now easily obtained as follows:



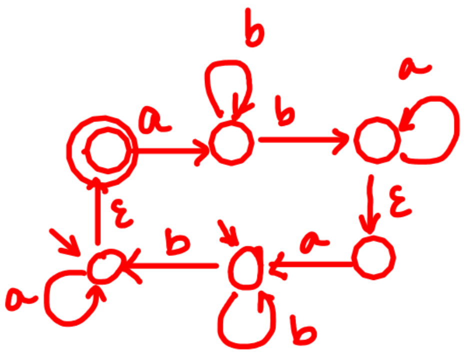
2. Let $\Sigma = \{a, b\}$. Construct NFAs, possibly with ε-transitions for each of the following languages.

   1. $L_4 = \{w \in \Sigma^* \mid n_{ab}(w) \text{ is even}\}$, where $n_{ab}(w)$ denotes the count of times ab appears in w as consecutive letters.
   2. $L_5 = \{w \in \Sigma^* \mid w$ contains the "pattern" abba (as consecutive letters) followed by the "pattern" baba, possibly in an overlapping manner$\}$. For example, $abababbaabb, babaabbabb \notin L$ but $abababbaba, ababbaabbaba \in L$.

Now try constructing a DFA that recognizes L_5 using the subset construction and ε-edge removal on the NFA constructed above. Do you see an exponential blow-up in the count of states as you do this conversion?

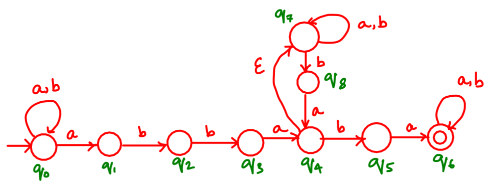
Solution: Possible NFAs are given below. These are not the only solutions. In fact, the first subquestion has a fairly simple DFA that can be directly constructed as well.

1.



The solution given here illustrates the convenience provided by NFAs with ε-transitions. We have two copies of an NFA that reads a sequence of as and bs with a leading a, a single ab change and possibly a trailing sequence of 0 or more as. These two NFAs can be seen in the top row (read left to right) and in the bottom row (read right to left). We simply connect them using ε-transitions, so that we read words with an even count of ab changes. The acceptance state of the overall NFA is therefore the starting state of the top copy of the NFA. The start states of the overall NFA are chosen so that we can accept a leading sequence of 0 or more as, or even such a sequence of bs followed by as (i.e. strings that don't contain a single ab change).

2.



The NFA is quite self-explanatory. This clearly shows the convenience of NFAs with ε-transitions. You can capture the intent of the problem so clearly that a look at the NFA tells you what it's designed to accept. This is not (so easily) the case if you determinize this NFA.

The NFA to DFA construction is left to you as a standard exercise.

3. **Take-away question: Propositional formulas and NFAs**

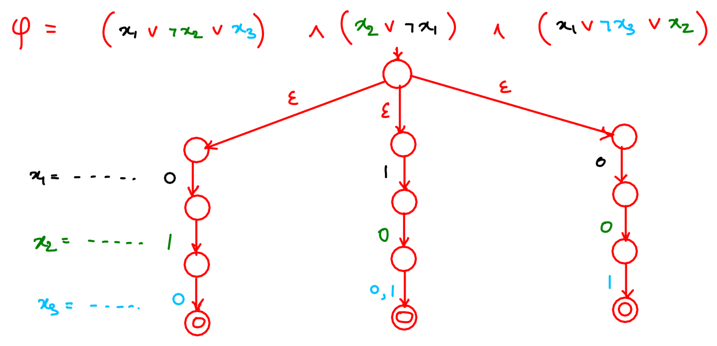
For a CNF formula φ with m variables and c clauses, show that you can construct in polynomial time an NFA with $O(cm)$ states that accepts all **falsifying or non-satisfying assignments**, represented as boolean strings of length m. You can assume that the formula φ is over variables $x_1 x_2 \dots x_m$, and you can assume that the NFA is fed as input the word $v_1 \cdot v_2 \cdot \dots \cdot v_m$, where $v_i \in \{0, 1\}$ and v_i is interpreted as the value of propositional variables x_i, for all $i \in \{1, \dots m\}$.

Can you construct an NFA in time polynomial in c and m that accepts all and only **satisfying assignments** of the CNF formula φ?

Solution:

1. On input ϕ, construct an NFA that non-deterministically picks one of the c clauses (via ε-transitions), reads the input of length m, and accepts if it does not satisfy the clause. For each clause, we need exactly m states, so the NFA has $O(cm)$ states. It is clear that we can construct the NFA in polynomial time, hence it is of polynomial size.

   Now consider any falsifying assignment of φ. At least one clause must be falsified by this assignment. Hence all literals of this clause must evaluate to 0. The NFA can non-deterministically choose this clause and accept the assignment string. Conversely, if the NFA accepts an assignment string, then there is a clause of φ that is falsified by the assignment. So, the assignment is a falsifying assignment of φ. Hence, the NFA accepts all and only the nonsatisfying assignments of φ. For example: let $\phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_1) \wedge (x_1 \vee \neg x_3 \vee x_2)$, the corresponding NFA is as shown below.


$\varphi = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_1) \wedge (x_1 \vee \neg x_3 \vee x_2)$
ε
ε
ε
x_1 = - - - - - 0
x_2 = - - - - - 1
x_3 = - - - - - 0
1
0
0,1
0
0
1

2. Unless we have a polynomial time algorithm for checking satisfiability of CNF formulas (aka **P vs NP** problem), such a construction is not possible. Indeed, if we could do this, then there would be a way to check in time polynomial in c and m whether φ is (un)satisfiable. Why? Simply construct the NFA for φ and check in the graph corresponding to the NFA whether any final state can be reached from any initial state

by any path. If so, there is a satisfying assignment, else the formula is unsatisfiable. Note that searching for a path in the graph can be done using any graph search algorithm like BFS or DFS in time that is polynomial in the size of the NFA (viewed as a graph). If the NFA's size is polynomial in c and m, this search (BFS/DFS) will also take time polynomial in c and m.