Exam 1. October 26, 2022

CS181: Fall 2022

Guidelines:

- The exam is closed book and closed notes. Do not open the exam until instructed to do so.
- Write your solutions clearly and provide complete proofs when asked to do so. You may use results and algorithms from class without proof or details as long as you state what you are using.
- I recommend taking a quick look at all the questions first and then deciding what order to tackle them in. Even if you solve the problems partially, attempts that show some understanding of the questions and relevant topics will get partial credit.
- You can use extra sheets for scratch work, but **you can only use the white space (it should be more than enough) on the exam sheets for your final solutions**. The exams will be scanned into gradescope, so for each problem, we will use only the corresponding white space for grading.
- Most importantly, make sure you adhere to the policies for academic honesty set out on the course webpage. We will enforce the policies strictly, and any cheating reported with the score automatically becomes zero.
- Write legibly and with a dark pencil or pen. All the best.

Problem	Points	Maximum
1		7
2		3
3		3
4		3
5		3
6		3
7		3
Total		25

Name	
UID	

1 Problem

The answers to the following should fit in the white space below the question.

1. Consider the mapping $M : \mathbb{N} \to \mathbb{N}^*$ defined as follows: $M(0) = (0), M(1) = (1)$ and for any natural number $n > 1$, $M(n)$ is the prime divisors of n written in increasing order. For example, $M(12) = (2, 3)$. True or False: M is an encoding. If true, justify your answer in a sentence. If false, give an example to show it is not an encoding. [1 point]

   False. For example, $M(12) = (2, 3)$ and $M(6) = (2, 3)$.

2. Describe the approach we followed in class to convert any encoding $E : \mathcal{O} \to \{0, 1\}^*$ to a new, prefix-free encoding. [1 point]

   You duplicate each bit of the output of E and append with 01 at the end.

3. Let $E : \mathcal{O} \to \{0, 1\}^*$ be a mapping of some set of objects. Define $flipE : \mathcal{O} \to \{0, 1\}^*$ as the mapping obtained by flipping each bit of the output under E. For example, if $E(x) = 100$ for some x, then $flipE(x) = 011$ and so on.

   True or False: If E is a valid encoding, so is $flipE$. If true, prove your statement. If false, give an example to show this is not true. [1 point]

   True: Suppose $flipE$ is not an encoding, then we must have two distinct objects $x \neq y \in \mathcal{O}$ such that $flipE(x) = flipE(y)$. However, the $flip$ operation itself ensures that this can happen only if $E(x) = E(y)$, a contradiction.

   An alternate solution: Let D be a decoder for E, i.e., for all x, $D(E(x)) = x$. Now, you can get a decoder for $flipE$ as follows $flipD(y) = D(flip(y))$ (so decode the flipped string).

4. A function $f : \{0,1\}^n \to \{0,1\}$ is called a t-junta if the function depends on only t coordinates. That is, there is some subset $I \subseteq [n]$ of size t such that if $I = \{i_1 < i_2 < \cdots < i_t\}$ are the elements of I in increasing order, then $f(x) = h((x_{i_1}, x_{i_2}, \cdots, x_{i_t}))$ for some other function $h : \{0,1\}^t \to \{0,1\}$. How many gates suffice to compute a t-junta on n bits? Justify your answer in a sentence. [1 point]

   Since f only depends on t coordinates, we could use the construction in class to get a circuit of size at most $O(t2^t)$. If you use the better upper bound we stated without proof, you would get a circuit of size $O(2^t/t)$. Either answer will get full-credit. The point is that the size does not depend on n.

5. For a string $x \in \{0,1\}^n$, let $flip(x)$ denote the string obtained by flipping each bit of x. For example, if $n = 3$, and $x = 100$, then $flip(x) = 011$. Call a function $f : \{0,1\}^n \to \{0,1\}$ *self-flippant* if $f(x) = f(flip(x))$ for all inputs $x \in \{0,1\}^n$. What is the number of self-flippant functions? You don't have to justify your answer. [1 point]

   The number of functions would be $2^{2^{n-1}}$. This is because we can pair up elements $(x, flip(x))$ into one group. There would be 2^{n-1} such pairs. For each pair we have two options to choose from - either say the function value is 0 or 1. Therefore, the number of functions would be $2^{(2^{n-1})}$.

6. Define the Odd-MaxBit function $OMB : \{0,1\}^5 \to \{0,1\}$ as the function where $OMB((x_0, x_1, x_2, x_3, x_4))$ is 1 if the largest index i such that $x_i = 1$ is odd. For example, $OMB(01000) = OMB(11000) = OMB(01010) = 1$, while $OMB(10000) = OMB(00001) = 0$. Describe a Boolean circuit for computing the OMB function. You can draw or write down a logical formula or describe it in words (unambiguously). You don't have to prove your construction works. [2 points]

   On an input $x = (x_0, x_1, x_2, x_3, x_4)$, $OMB(x) = 1$ is one only if either $x_1 = 1$ and $x_2 = x_3 = x_4 = 0$ or if $x_3 = 1$ and $x_4 = 0$. Following this logic, you can build a circuit out of the following expression:

   $$OMB((x_0, x_1, x_2, x_3, x_4)) = OR(AND(x_1, NOT(x_2), NOT(x_3), NOT(x_4)), AND(x_3, NOT(x_4))).$$

   [Here we are taking the natural convention that $OMB((0,0,0,0,0)) = 0$ (as there is no index with a 1). If you adopt a different convention also it'll be fine - as long as your circuit gets all the other strings correctly, you will get full-credit.]

2 Problem

Define the function $NOTTWO : \{0,1\}^3 \to \{0,1\}$ as the function where $NOTTWO(a,b,c)$ is 1 if and only if the number of bits among a, b, c that are 1 is not two. In other words, $NOTTWO(1,1,0) = NOTTWO(1,0,1) = NOTTWO(0,1,1) = 0$ and the function evaluates to 1 on all other inputs.

Show that $\{NOTTWO, 0, 1\}$ is a universal set of gates. Here, 0 denotes the constant 0 function and 1 denotes the constant one function. [3 points]

Solution You can implement the NAND function using $NOTTWO$ as follows:

$$NAND(a,b) = NOTTWO(a,b,0).$$

This is because if both $a = b = 1$, then $NOTTWO(a,b,0) = 0$, whereas if either a or b is 0, then there is at most one 1 so $NOTTWO(a,b,0) = 1$ as desired.

As $NAND$ itself a universal gate, $\{NOTTWO, 0\}$ is universal.

3 Problem

For intgers $0 < k < n$, define the function $Exact_k : \{0,1\}^n \to \{0,1\}$ as the function such that $Exact_k(x) = 1$ if and only if the number of 1's in x is exactly k.

Show that $Exact_k$ can be computed by Boolean circuits of size much smaller than the bound we saw in class.

(So, for example, if $k = 2$, the general theorem in class would say it can be computed with $O(n2^n)$ gates; but $Exact_2$ can be computed with at most $O(n^3)$ gates.)

You can describe your construction as we did in class or draw them (with explanations to interpret them). Your construction should work for general k, and you don't have to prove your construction works to get full credit. [3 points]

Solution Observe that on an input x, $Exact_k(x)$ is 1 if and only if x has exactly k 1's. So the number of inputs on which the function $Exact_k$ evaluates to 1 is exactly $\binom{n}{k}$ (the latter being the number of strings in $\{0,1\}^n$ that have exactly k 1's).

Therefore, if we follow the construction from class: we would need at most $2n-1$ gates for each one of the strings that have exactly k 1's (to implement the E_α functions from class). We then need $\binom{n}{k}$ or gates to implement a giant or over all the strings with exactly k 1's.

Thus, there is a Boolean circuit for computing $Exact_k$ with at most $\binom{n}{k} \cdot (2n-1)$ gates.

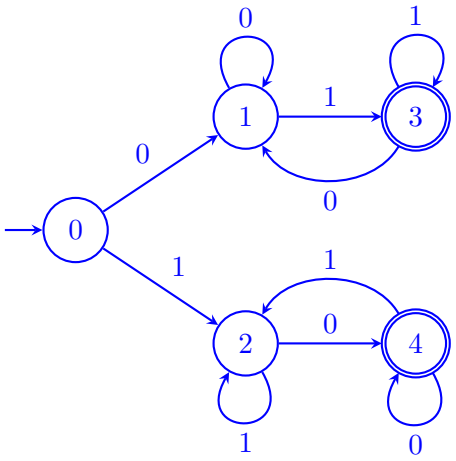
[This question and solution are similar in spirit to Problem 1.4 from Exam 1 F21. You can write this even as $O(n^{k+1})$ to get full credit.]

4 Problem

Design a DFA that accepts all strings x with different starting and ending bits. For example, $10, 111110, 010111$ should be accepted, but strings like $1010101, 0000, 0000110$ should not be accepted.

You don't have to prove that your construction works. A correct diagram and a sentence or two explaining your diagram would be enough for full credit. [3 points]

Solution The DFA construction for problem 4 is exactly the same as example 6 in lecture 7 (DFA that accepts same start and end bits), with the only difference being where the accept states are located:



[We once again adopt the convention that the empty string should not be accepted. If you adopt a different convention, that will be fine too. As long as you get other strings correctly, you'll get full credit.]

5 Problem

For a language L, define $ODD(L)$ to be the language containing all strings of even length such that the substring formed by the bits with odd-numbered indices is in L (recall that our indexing starts with 0). That is, a string $x = x_0x_1x_2 \cdots x_{2n-1} \in ODD(L)$ if and only if $x_1x_3x_5 \cdots x_{2n-1} \in L$.

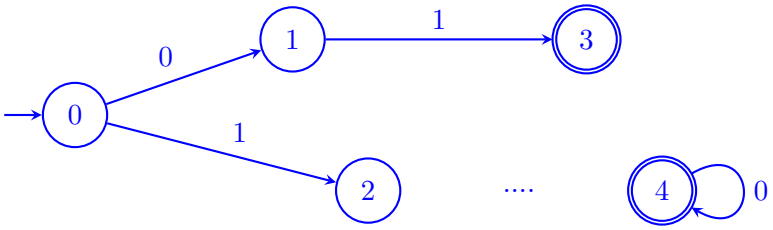
For example if $10 \in L$, then $0100, 1100, 0110, 1110$ would all be in $ODD(L)$.

Show that if a language L is computable by a DFA, then $ODD(L)$ is also computable by a DFA. You don't have to prove that your construction works. A correct diagram (and a sentence or two explaining your diagram) would be enough for full credit. [3 points]

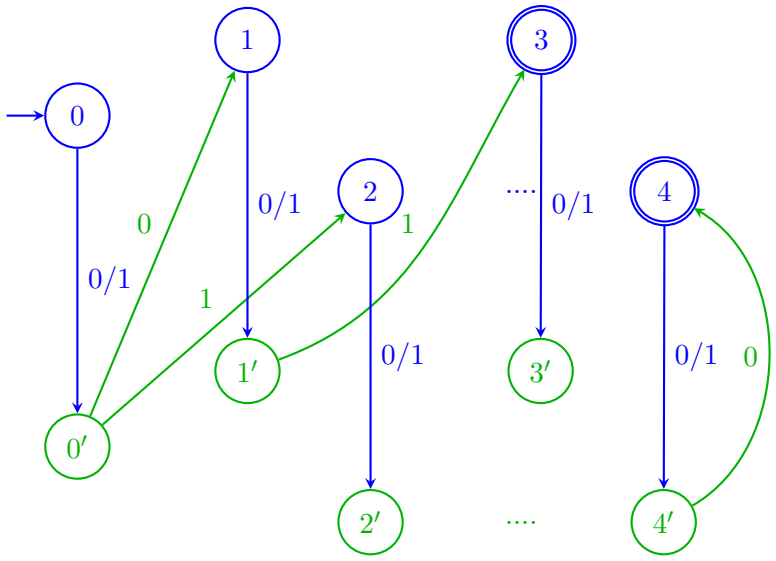
[Hint: One approach is to create separate copies for each state of the original DFA and design transitions to perform steps of the original DFA only for every other bit.]

Solution: We follow the hint by making two copies of our DFA states and connect them together using the following procedure: Given a DFA D that computes L and has nodes q, we will denote D' as a copy of the DFA with nodes q'. Starting with our initial state q_0, we remove any original outbound edges (including self loops) and instead draw the transition to q_0' with label 0/1. Next replace each edge of the form (q_0', q_i') with transition (q_0', q_i) using the same label. We repeat this process so that for all q_i we have exactly one edge labeled 0/1 to q_i', and all edges of the form (q_i', q_j') have been replaced with (q_i', q_j). We keep the accept states only for DFA D (since we only accept even-length strings).

Pictorially, let the following schematic diagram represent D (only some of the transitions are shown).



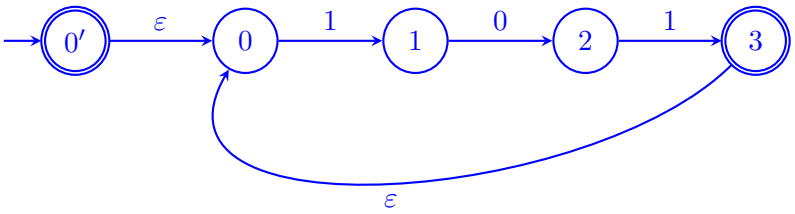
Applying our procedure to D and D' produces the following DFA:


0
1
2
3
4
0′
1′
2′
3′
4′
0/1
0
1
0/1
1
0/1
....
0/1
0/1
0
....

6 Problem

Design an NFA that accepts the language $\{101\}^*$. You don't have to prove that your construction works. A correct diagram and a sentence or two explaining your diagram would be enough for full credit. [3 points]

Solution: The construction can be achieved by first writing the DFA for $L = \{101\}$, then using the closure property for the Kleene star operator.



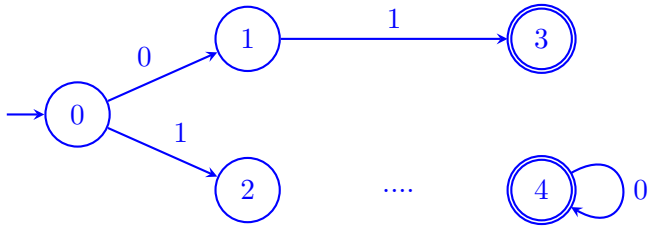
7 Problem

For a language L, define $ADDANY(L)$ as the language containing all strings we can obtain by adding one symbol to a string in L. Thus, $ADDANY(L) = \{abc : ac \in L, a, c \in \{0,1\}^*, b \in \{0,1\}\}$. Show that if a language L is computable by a DFA, then $ADDANY(L)$ is computable by an NFA. You don't have to prove that your construction works. A correct diagram and a sentence or two explaining your diagram would be enough for full credit. [3 points]

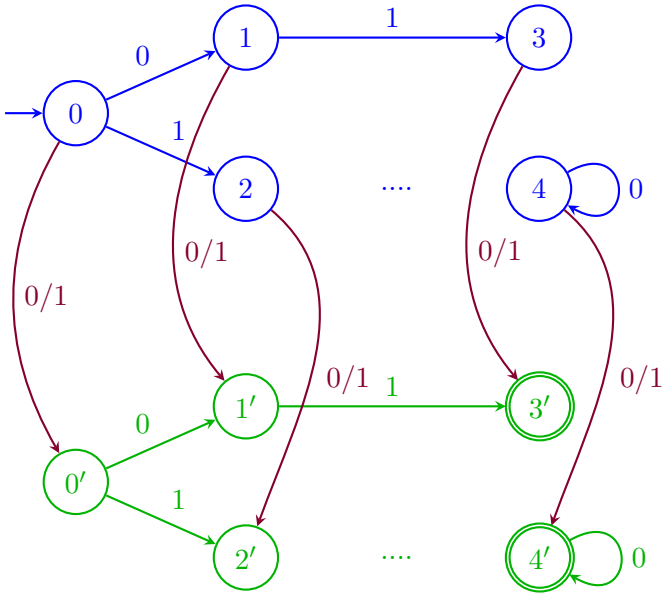
(As an example of the operation, if $L = \{01\}$, then $ADDANY(L) = \{001, 101, 001, 011, 010, 011\}$.)

Solution: Suppose we have a DFA D that computes L. We first make a copy of this DFA, D'. We next design a NFA as follows: The states are the union of D, D'. The starting node is the starting node of D. The accepting states are the accepting states in D' only (those in D will no longer be considered accept). In addition, for each node q in D, we add additional transitions (draw new edges) with labels 0/1 input to the corresponding node q' in D'.

Pictorially, let the following schematic diagram represent D (only some of the transitions are shown).



The NFA for computing $ADDANY(L)$ would be the following:



[The construction is similar in spirit to what happened in Problem 6 of Exam 1, F21.]