Exam 1. October 27, 2021

CS181: Fall 2021

Guidelines:

- The exam is closed book and closed notes. Do not open the exam until instructed to do so.
- Write your solutions clearly and when asked to do so, provide complete proofs. You may use results and algorithms from class without proofs 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 to them in. Even if you don't solve the problems fully, 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, only the corresponding white space will be used for grading.
- Most importantly, make sure you adhere to the policies for academic honesty set out on the course webpage. The policies will be enforced strictly and any cheating reported with the score automatically becoming zero.
- Write clearly and legibly. All the best!

Problem	Points	Maximum
1		10
2		3
3		3
4		3
5		3
6		3
Total		25

Name	
UID	

1 Problem

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

1. True or False: The binary encoding of natural numbers to strings (i.e., elements of $\{0,1\}^*$) is prefix-free. If true, justify your answer in a sentence. If false, give an example to show it is not prefix-free. [1 point]

   Solution. False. To show that this encoding is not prefix-free, it suffices to find two strings x, y such that $E(x)$ is a prefix of $E(y)$. For example, $x = 3$ encoded as $E(x) = 11$, and $y = 7$ encoded as $E(y) = 111$. □

2. True or False: Let $E : S \to \{0,1\}^*$ be an encoding of some set of objects such that for any $i, j \in S$, $|E(i)| = |E(j)|$ (i.e., the lengths of all encodings of elements is the same). Then, E is also prefix-free. If true, justify your answer in one or two sentences. If false, give an example to show it is not prefix-free. [1.5 point]

   Solution. True. Notice that if $|s| = |s'|$ and $s \neq s'$ for some strings s, s', then neither s nor s' is a prefix of the other one. Hence, for $i, j \in S$, if $i \neq j$, then $E(i)$ cannot be a prefix of $E(j)$. □

3. Is the following sequence of lines a valid NAND-CIRC-PROGRAM. Justify your answer in a sentence. [1 point]

   (a) $A = NAND(x[0], C)$
   (b) $B = NAND(x[1], A)$
   (c) $C = NAND(x[2], B)$
   (d) $y[0] = NAND(x[3], C)$.

   Solution. False. Here, variable C depends on variable A (through variable B), while A depends on C, so the corresponding directed graph has a cycle. □

4. In class we saw a way to build a Boolean circuit for any function $f : \{0,1\}^n \to \{0,1\}$ of size at most $O(n2^n)$. However, this method can yield better size circuits for some functions.

   Define $ExactlyOne_n : \{0,1\}^n \to \{0,1\}$ by $ExactlyOne_n(x)$ is 1 if exactly one bit of the input string is 1 and is 0 otherwise. If one follows the approach in class to build a Boolean circuit for $ExactlyOne_n$, how many gates would be used? You may use big-O notation. Explain your answer in a sentence or two (no need for full proof). [1.5 points]

   (Just as an example for the definition of $ExactlyOne$, for $n = 3$, we would have $ExactlyOne_3(010) = 1$, but $ExactlyOne_3(011) = ExactlyOne_3(111) = 0$.)

Solution. $O(n^2)$ gates. First, we have n gates to compute all negations $\neg x_i$. Then, we have n different assignments where the function evaluates to 1, and for each of those we need to compute AND of n literals which requires $n-1$ AND gates. Finally, we have to compute OR of those n values which is another $n-1$ gates. In total, we have $n+n(n-1)+n-1 = O(n^2)$ gates. □

5. What is a lower bound on the minimum number of Boolean gates required to compute all functions $f : \{0,1\}^n \to \{0,1\}^n$? No need to justify your answer [1 point]

   (For full credit, your lower bound should be correct and as high as possible ignoring constant factors; don't say 1 is a lower bound :)).

   Solution. $\Omega(2^n)$. In homework, we showed that for functions $f : \{0,1\}^n \to \{0,1\}^m$ we need at least $\delta m 2^n/n$ gates for $\delta = 1/120$. Setting $m = n$ gives us a lower bound of $2^n/120$. □

6. State the Physical Extended Church Turing Thesis. [1 point]

   Solution. If a function $F : \{0,1\}^n \to \{0,1\}^m$ can be computed in the physical world using s amount of physical resources, then it can be computed by a Boolean circuit program using roughly s gates. □

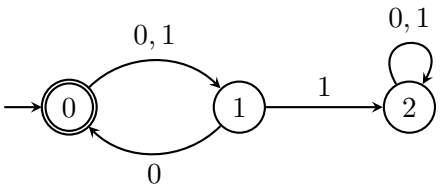
7. True or False: If $f, g : \{0,1\}^* \to \{0,1\}$ are computable by a DFA, then so is the function $h : \{0,1\}^* \to \{0,1\}$ defined as the XOR of f, g. That is $h(x) = XOR(f(x), g(x))$. [1.5 points]

   Justify your answer in one or two sentences. (The main idea for true or false is enough).

   [Recall that for two bits $a, b \in \{0,1\}$, $XOR(a,b)$ is 1 if exactly one of a, b is 1 and is 0 otherwise.]

   Solution. True. This can be shown by a Cartesian product construction we used for the intersection of languages. A short description is the following. In a DFA for h, our states are pairs of states (i, j') of DFAs that compute f and g. Transitions follow the transition tables of both DFAs simultaneously, and accepting states are those pairs where exactly one element is an accepting state in the corresponding DFA. □

8. Describe in words the set of strings accepted by the following automata. Please give a succinct description in a sentence or two (just writing down a few example strings is not enough). [1.5 points]



   Solution. Words of even length in which every second symbol (if any) is 0. □

2 Problem

Define the function $NAEqual : \{0,1\}^3 \to \{0,1\}$ as the function where $NAEqual(a,b,c)$ is 1 if not all bits a,b,c are equal to each other and is 0 otherwise. In other words, $NAEqual(0,0,0) = NAEqual(1,1,1) = 0$ and the function evaluates to 1 on all other inputs.

Design a AND/OR/NOT circuit to compute $NAEqual$. You can draw the circuit or write it as a AND/OR/NOT program. You don't have to prove your construction works. [3 points]

Solution. The function is a negation of "all inputs are ones or all inputs are zeros". This can be computed by the following AON program.

1. $NEG_0 = NOT(x[0])$
2. $NEG_1 = NOT(x[1])$
3. $NEG_2 = NOT(x[2])$
4. $ONES_0 = AND(x[0], x[1])$
5. $ONES_1 = AND(ONES_1, x[2])$
6. $ZEROS_0 = AND(NEG_0, NEG_1)$
7. $ZEROS_1 = AND(ZEROS_0, NEG_2)$
8. $AllEqual = OR(ONES_1, ZEROS_1)$
9. $y[0] = NOT(AllEqual)$

□

3 Problem

Show that $\{NAEqual, 0, 1\}$ is a universal set of gates. Here, $NAEqual : \{0,1\}^3 \to \{0,1\}$ is the function defined above and 0 is the constant 0 function and 1 denotes the constant one function. [3 points]

[Recall that to show a set of gates universal it suffices to show that they can simulate either the entire set (AND, OR, NOT) or the single $(NAND)$ function.]

You can draw or write down the formulas for the computations as we did in class. You don't have to prove your construction works.

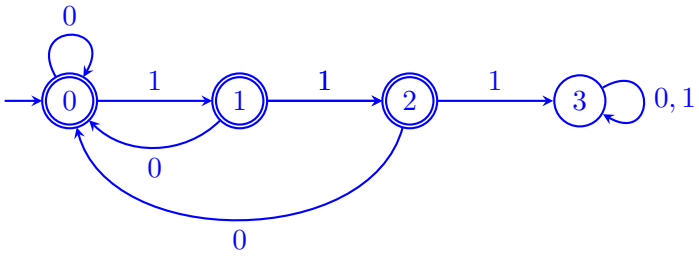
Solution. Since $NAND$ is a universal gate, it suffices to compute $NAND$ using $\{NAEqual, 0, 1\}$. This can be easily done as $NAND(x, y) = NAEqual(x, y, 1)$. Indeed, to make values $x, y, 1$ not-all-equal, we need at least one of x, y to equal 0 which is the same as for $NAND$ function. □

4 Problem

Design a DFA that accepts all strings x that do not contain three consecutive 1's in them. That is, draw a DFA D such that $D(x) = 1$ if and only if x **does not** contain 111 as a substring. You don't have to prove that your constuction works, just a correct diagram (and a sentence or two explaining your diagram) would be enough for full-credit.

For instance, $1010, 10001010, 11001011, \ldots, 1010, 10001010, 10101011$ should be accepted (i.e., $D(x) = 1$ for these strings as x) by the DFA whereas strings such as $111, 0111, 1110, 01011100$ should not be accepted. [3 points]

Solution. The following DFA computes the language described above.

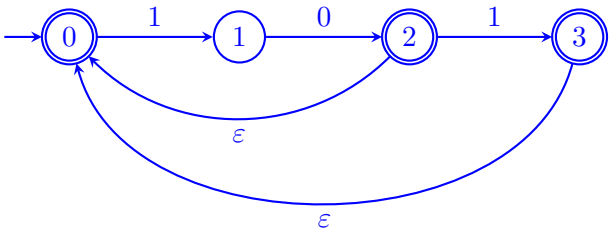


□

5 Problem

Design a NFA that accepts the language $\{10, 101\}^*$. You don't have to prove that your constuction works, just a correct diagram (and a sentence or two explaining your diagram) would be enough for full-credit. [3 points]

Solution. The following NFA computes the language described above.



□

6 Problem

For a language L, define $DROPANY(L)$ to be the language containing all strings that can be obtained by removing one symbol from a string in L. Thus, $DROPANY(L) = \{ac : abc \in L, a, c \in \{0,1\}^*, b \in \{0,1\}\}$. Show that if there is a DFA for L, then there is a NFA for $DROPANY(L)$. [3 points]

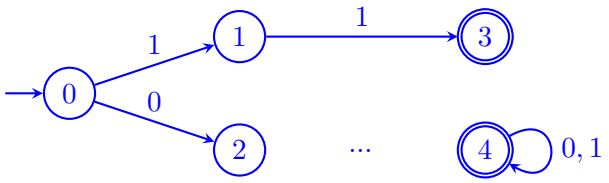
(As a simple example of the operation, if $L = \{0, 010\}$, then $DROPANY(L) = \{\varepsilon, 10, 00, 01\}$.)

[Hint: Try to guess 'where' the drop occurs. One idea is have two copies of the DFA for L, and then connect them based on a 'guess' for where the drop is.]

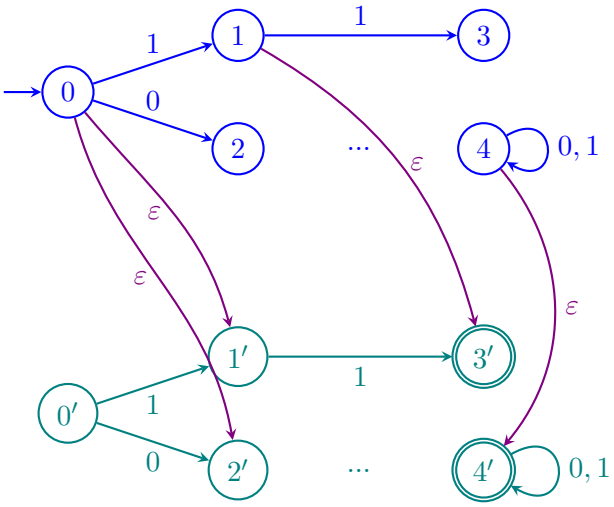
Solution. Let D be a DFA that computes L. Let $\{0, 1, \ldots, C-1\}$ be its states, T the transition function, and S the set of accepting states. First, we make a copy D' of D with states $\{0', 1', \ldots (C-1)'\}$. Its transition function T' and set of accepting states S' are defined analogously to those of D, i.e., $T'(i', a) = j'$ if and only if $T(i, a) = j$, and $i' \in S'$ if and only if $i \in S$. Now, we add transitions on empty that correspond to a dropped symbol in a word. For any transition $T(i, a) = j$ in D, we add an ε-transition from i to j'.

Our NFA N consists of D, D', their original transitions, and transitions on empty defined above. 0 is a start state, and S' is a set of accepting states. N processes words as follows. First, it follows the computation path in D. In the place of the dropped symbol, it uses ε-transition to the next state but in D' (as if it read some symbol and made a transition). Finally, N processes the remaining suffix in D' which is the same as computation in D.

Let the following schematic diagram represent D (only some of the transitions are shown).



Then, N will have the following structure.



□