CS 3100, Fall 2020, Final Exam, 100 points
27 questions of 3 points each (81 points) are provided.
Three non-multiple-choice questions (19 points : 6, 6, 7 pts)
Please answer on Canvas in the Quiz called "Final Exam" available there (12/9/20 thru 12/10/20). Fill answers matching questions given here.

Please choose up to two options per question. For each correct option chosen, you will earn 1.5 points. Please do not choose more than two options—you will then get zero points for the whole question. All questions as well as the two correct options in each question have the same point value. Thus, (1) Choosing two wrong options results in zero points for the question; (2) choosing one right and one wrong results in 1.5 points (for the right option).

1. Consider these two regular expressions:

   $R_1 = (01^*0)^*$
   $R_2 = (0^*1^*0)^*$

   Let $L(R)$ mean the language of regular expression R.

   Choose from this list:

   (a) $L(R_1)$ contains 1010 ☐
   (b) $L(R_2)$ contains 1010 ☐
   (c) $L(R_2)$ contains 1110 ☐
   (d) $L(R_1)$ contains 000 ☐

2. Consider the same regular expressions, i.e., $R_1 = (01^*0)^*$ and $R_2 = (0^*1^*0)^*$: Consider the DFAs presented in Fig-A and Fig-B (states that do not fully decode are assumed to transition to a black-hole state). Also let $L(A)$ mean "the language of Fig-A's DFA" and $L(B)$ mean "the language of Fig-B's DFA". Now choose your answers:

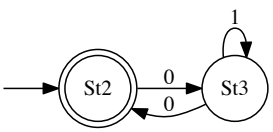


Fig-A:

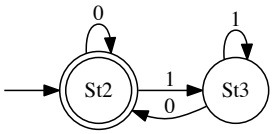
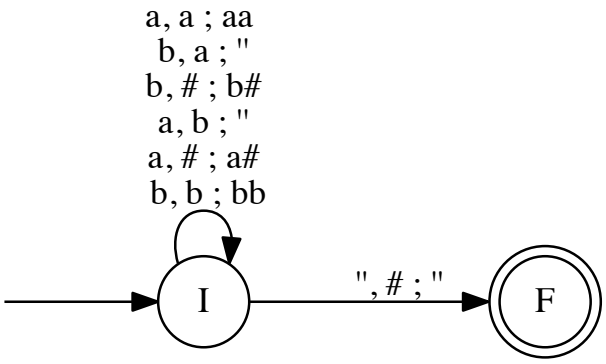


Fig-B:

   (a) $L(A) = L(R_1)$ ☐
   (b) $L(B) \subseteq L(R_1)$ ☐
   (c) $L(A) \cap L(B) \cap L(R_1) \cap L(R_2)$ is infinite ☐
   (d) $L(B) \subseteq L(A)$ ☐

3. Consider the PDA "X" below and let its language be $L(X)$. Let P stand for the polynomial-time language family (in the sense of complexity). Select your options:



:

(a) $bab \in L(X)$ ☐

(b) X can be recommended as a PDA for recognizing $\{a^n b^n \ : \ n \geq 0\}$. ☐

(c) $baab \in L(X)$ ☐

(d) $L(X) \in P$ ☐

4. Consider the following CFG

```
S -> M C | A N
M -> a M b | ''
N -> b N c | ''
A -> a A | ''
C -> c C | ''
```

Choose answers from this list:

(a) L(S) is regular. ☐

(b) A direct CFG to PDA conversion of this grammar will yield a "working" state that has a self-loop labeled with more than 10 cases ☐

(c) This grammar is ambiguous but it can also be disambiguated ☐

(d) abbcc is in the language of this grammar ☐

5. Pick all true assertions (One convention: we always use the tightest language classification; thus if a language is not regular, then only will we begin calling it context-free, etc.)

   (a) Recursively Enumerable languages are closed under complementation ☐

   (b) Recursively Enumerable languages are closed under intersection ☐

   (c) Context-free languages are not closed under concatenation ☐

   (d) Every recursively enumerable language is infinite ☐

6. These are true of Lambda conversions.

   (a) $(\lambda x.(\lambda y.(y + x))) \rightarrow (\lambda x.(\lambda x.(x + x)))$ via the α rule. ☐

   (b) $(YY) \rightarrow Y(YY)$ ☐

   (c) $(\lambda x.x)(\lambda y.y) \rightarrow (\lambda z.z)$ via α and β rules. ☐

   (d) The combinators S, K, and I are a Turing-complete; we cannot model any one of these combinators in terms of the other two. ☐

7. Given these languages, pick the correct answers from the list below:

   - $L_1 = \{w \in \{a, b\}^* : \#_a(w) < \#_b(w)\}$
   - $L_2 = \{(ab)^n : n \geq 0\}$
   - $L_3 = \{a^{n^2} : \ n \geq 0\}$
   - $L_4 = \{a^i b^j c^k : \text{if } (i = 3) \text{ then } (j = k)\}$

   (a) L_1 and L_4 are context-free. ☐

   (b) L_2 and L_3 are context-free. ☐

   (c) L_2 and L_4 are context-free. ☐

   (d) One of the languages in this list is not Recursive. ☐

8. Let the language (over $\Sigma = \{0, \#\}$) defined by the following CFG be called L.

```
S -> T T | U
T -> 0 T | T 0 | #
U -> 0 U 0 0 | #
```

   These are true of L's language:

   (a) L is regular. ☐

   (b) L is context-free. ☐

   (c) L would be regular if `S` goes to `T` instead of to `TT`. ☐

   (d) The language of `T` is not regular, as it has both types of linearity. ☐

9. Identify the correct answers

   (a) If a language L is NP-Complete and $\overline{L}$ is in NP, then for every language $L' \in NP$, it is also the case that $L' \in CoNP$ (and vice-versa). ☐

   (b) The complexity of 3COLOR is exponential while 2COLOR is polynomial ☐

   (c) For a new language X, if we can show that $3SAT \leq_P X$, then we still do not know whether X is NP-complete ☐

   (d) All functions have exactly one fixpoint ☐

10. Choose the correct facts from this list:

    (a) A Turing machine accepts an input only when it has read its input fully and the control state goes from its starting state q0 in Q to some terminal state f in F (from f, there are no further moves) ☐

    (b) In general, it is not possible to tell whether a TM will ever read its input fully ☐

    (c) A Turing machine started in state q0 in Q accepts its input when it attains the state f in F from which it has no further moves ☐

    (d) A Turing machine started in state q0 in Q accepts its input if it attains the state f in F at least once ☐

11. Choose from the list below those languages that are RE but not recursive:

    (a) The language $Halt_{TM}$ which is the set of machine, word pairs $\langle M, w \rangle$ such that Turing machine M halts on word w ☐

    (b) The language { } ☐

    (c) The set of Turing machine descriptions whose language contains the string 101 ☐

    (d) The language of Turing machines whose tape has a fixed length. ☐

12. Choose from these facts:

    (a) Every NP-complete language is recursively enumerable ☐

    (b) If an NDTM has a language-equivalent DTM, then P=NP ☐

    (c) Every regular language L is in P because there is a deterministic P-time decider for it ☐

    (d) In the Chomsky hierarchy we studied consisting of DFA, PDA, and TM, all have non-deterministic versions that can be converted to corresponding deterministic versions ☐

13. Let us consider the Post correspondence problem (PCP). One can enumerate all PCP instances that have solutions. Given this, choose all the assertions below that are true.

    The set of PCP instances that have solutions is:

    (a) Not even recursively enumerable (a procedure does not exist) ☐

    (b) Not recursive (no algorithm exists) ☐

    (c) Recursively enumerable (a procedure exists) ☐

    (d) Context-free (a CFG parser exists) ☐

14. In computability theory, when we are creating "a mapping reduction," the following are implied:

    (a) We are showing that a "new problem" is being reduced to an old, and impossible-to-solve problem. ☐

    (b) We are showing that given an old impossible-to-solve problem "A", a mapping can be created to the new problem "B" that is at least as hard as A. ☐

    (c) Both the "old problem" and the "new problem" must be equally hard to solve. ☐

    (d) If a decider is shown to exist for the new problem, that would result in a contradiction. ☐

15. Suppose someone produced a proof, from first principles, for A_{TM} being non-recursive. They want to use that to show that $Halt_{TM}$ is non-recursive via a mapping reduction function f. The following two steps accomplish a sound reduction from A_{TM} to $Halt_{TM}$.

    (a) Show that for every $\langle M, w\rangle$ pair, $\langle M, w\rangle \in Halt_{TM}$ iff $f(\langle M, w\rangle) \in A_{TM}$ ☐

    (b) Show that for every $\langle M, w\rangle$ pair, $\langle M, w\rangle \in A_{TM}$ if $f(\langle M, w\rangle) \in Halt_{TM}$ ☐

    (c) Show that for every $\langle M, w\rangle$ pair, $\langle M, w\rangle \in A_{TM}$ iff $f(\langle M, w\rangle) \in Halt_{TM}$ ☐

    (d) Show that function f is computable ☐

16. Consider the following claimed mapping reduction proof to show that checking whether a TM has a *context-free language* is not decidable. Let a decider for a context-free language, if one exists, be called D_{cfl}.

```
M'(x) {
  if x is of the form 0^n 1^n then goto accept_M' ;
  Run M on w ;
  If M accepts w, goto accept_M' ;
  If M rejects w, goto reject_M' ; }
```

Pick out the two correct arguments from below:

(a) Under the condition "M does not accept w," the language of `M'` is context-free; Under the condition "M accepts w," the language of `M'` is Σ^* which is regular. Thus, the existence of D_{cfl} allows us to tell whether the language of `M'` is a CFL or not, and hence decide A_{TM}. ☐

(b) Under the condition "M rejects w," the language of `M'` is also context-free. This makes it impossible to finish the "if and only if" of a mapping reduction. ☐

(c) If we make the initial test
`if x is of the form ''ww'' then goto accept_M'`
the proof would be fixed. Here, `w` is a member of $\{0,1\}^*$. ☐

(d) We must make the initial test
`if x is of the form ww^R then goto accept_M'`
to fix the proof. ☐

17. Let PCP be all PCP instances that have a solution. Then:

(a) In PCP, if the tiles are required to be used *without* repetition, the problem will still be undecidable ☐

(b) If all the tiles carried only strings over $\{1\}$ (singleton alphabet), then the problem would be decidable. ☐

(c) The PCP solver included within Jove is, in principle, an algorithm ☐

(d) PCP is a good starting point for proving results such as Grammar Ambiguity being not recursive. For this, we reduce PCP to Grammar Ambiguity. ☐

18. In 1975, Pratt established the result "PRIMES are in NP." In 2002, Agrawal, Kayal, and Saxena established the result "PRIMES are in P."

(a) We now have a P-time algorithm that checks for primality ☐

(b) We can now factor numbers into their prime factors in P-time ☐

(c) Checking whether a given number is Prime in P-time is not the same as factoring numbers in P-time (which is still open) ☐

(d) We have now the result P=NP ☐

19. Consider BDDs over variables x and y for the Boolean functions *and*, *or*, and *xor*. The following are true:

    (a) The BDD for *or* will have only one path to the "1" node starting from the root. ☐
    (b) The BDD for *xor* will have two paths to the "1" node starting from the root. ☐
    (c) The BDD for *and* will have one path to the "1" node starting from the root. ☐
    (d) BDDs always have a polynomial number of nodes in them, even though Boolean satisfiability checking is NP-complete. ☐

20. (Part-1): If a Deterministic Turing Machine M is programmed to check if two strings w_1 and w_2 (both of length N) are equal by zig-zagging along the string, the machine M will take **this much runtime** in Big-O notation. (Part-2): A TM that writes into $O(N^3)$ cells at least once must take this much time.

    (a) $O(N)$ for Part-1. ☐
    (b) $O(N^2)$ for Part-1. ☐
    (c) At least $O(N^3)$ time for Part-2. ☐
    (d) Can be lower than $O(N^3)$ time for Part-2. ☐

21. To preserve one's own reputation, one must **not** do these:

    (a) Try to develop an algorithm to check for CFG ambiguity ☐
    (b) Develop a derivatives-based pattern matcher ☐
    (c) Develop a P-time algorithm to check for Boolean satisfiability ☐
    (d) Refuse to solve the Traveling Salesperson problem ☐

22. New languages L are typically shown NPC by these two steps

    (a) Showing that L is in NP ☐
    (b) Showing that L can be P-time mapping-reduced to an existing NP-complete problem ☐
    (c) Showing that all the NP problems can be mapping-reduced to L ☐
    (d) Showing that some NPC problem can be P-time mapping-reduced to L ☐

23. A new problem L can be shown to be NP-hard. This is not sufficient as an NP-completeness proof for L because:

    (a) L being NP-hard means that L is in NP ☐
    (b) L can be undecidable ☐
    (c) We must additionally show that L is in NP ☐
    (d) (c) *and* the statement "The first language shown NP-hard was the K-Clique problem" ☐

24. By "3CNF," "3SAT," and "DNF" (disjunctive normal form; Boolean formulae such as `(abc+def)`), the following are meant:

    (a) When a 3CNF formula is converted to DNF, the size of the resulting formula can be exponential with respect to the input formula ☐

    (b) The satisfiability checking of DNF formulae is also NP-complete ☐

    (c) A 3SAT formula that is not a contradiction (always false) ☐

    (d) When a 3CNF formula is converted to DNF, the size of the resulting formula grows polynomially with respect to the input formula ☐

25. If we are given a DIMACS file containing these lines

```
 1  2 0
 1 -2 0
 ...many more clauses...
-1  2 0
-1 -2 0
```

    where "many more clauses" are lines are of the form "`p q 0`" with `p` and `q` being arbitrary positive or negative integers. Here is what we can then conclude about this SAT problem:

    (a) Such CNF formulae can be decided in Polynomial time ☐

    (b) The above-shown CNF formula is satisfiable ☐

    (c) The above formula is a 2CNF formula actually ☐

    (d) No 2CNF formula can ever have a 3CNF representation ☐

26. Given that 3SAT has been shown to be NP-Complete (NPC), to show that the K-Clique problem is NPC, here are all the additional steps needed:

    (a) Showing that 3SAT is in NP ☐

    (b) Showing that K-Clique is in NP ☐

    (c) Showing that K-Clique can be reduced to 3SAT via a P-time reduction ☐

    (d) Showing that 3SAT can be reduced to K-Clique via a P-time reduction ☐

27. Consider the 3CNF formula `(a+b+c) (a+!b+!c)(!a+!b+!c)`
that is being converted into a graph for the purposes of an NP-completeness proof of 3COLOR. The following are true of the graph that is generated from this formula or a small variant of this formula.

    (a) We can conjoin clauses involving `a,b,c` to the original formula to ensure that 3-coloring is blocked. ☐

    (b) The impossibility of 3-coloring implies the impossibility of 4-coloring. ☐

    (c) The impossibility of 3-coloring implies the impossibility of 2-coloring. ☐

    (d) 2-coloring is in P, and a good way to argue that is through depth-first search (DFS). ☐

LONG ANSWER PROBLEMS

1. (6 points, Mapping Reductions) This machine `M'` was generated based on a given machine M and its input word w in a mapping-reduction construction.

```
M' (x) {
Run M on w;
If M accepts w
   if (x==101) accept;
   else reject;
else accept;
}
```

The person who wrote the proof did not specify whether the A-domain of the mapping reduction function f was A_{TM} or $Halt_{TM}$ (one of these is true). The person also did not specify whether the B co-domain was E_{TM} (the language of TMs with an empty language) or NE_{TM} (the language of TMs with a non-empty language); again, one of these is true. Your task is to find out the A and B, accompanied by clear reasons.

(a) What is A, and why?

(b) What is B, and why?

(c) With these A and B chosen, why is it the case that $x \in A \Rightarrow f(x) \in B$?

(d) With these A and B chosen, why is it the case that $x \notin A \Rightarrow f(x) \notin B$?

2. (6 points, Fixpoint Theory) Treat the 3x+1 code as the recursive program:

```
tep(x) = if (x==1) then 1 elif odd(x) then tep(3*x+1) else tep(x/2)
```

Now, use the notation on Page 287 of our book, and explain how you can convert this recursive definition into the "prefact" form shown on Page 293. Show all necessary steps, calling the result "`pretep`." `pretep` is the function whose fixpoint `tep` is. How will the call `Ye(pretep)(1)` unfold? Unfolding of recursive calls is shown on Page 290; use that style of presentation. In other words, show how `Ye(pretep)` evolves into `pretep(Ye(pretep))` and then continues to "munge" on the `1` input, till the final answer is obtained.

How to answer: Provide a set of derivations of the form

```
tep =  Lambda x : (if...)

tep = ...after doing one more Lambda-lifting...


...steps involving applying Ye to tep as the first arg, and 1 as the second arg...

-->

...

-->

1     (final answer)
```

3. (7 points, NPC) Given the 3CNF formula

$$\phi = (a + b + c) \cdot (a + b + !c) \cdot (a + !b + c) \cdot (a + !b + !c) \cdot (!a + b + c) \cdot (!a + b + !c) \cdot (!a + !b + !c)$$

Demonstrate your understanding of the k-clique reduction by describing these items below:

(a) The satisfying instance of ϕ, presented as

$a = \ldots$
$b = \ldots$
$c = \ldots$

(b) The nodes, one from each island, that are going to be on this 7-clique (Remember, there will be a 7-clique that will be formed.); present the nodes as follows, picking:

Nodes $= (p, q, r, s, t, u, v)$, with $p, q, \ldots, v$ suitably replaced.

(c) A portion of the "island graph." In particular, describe all the edges between Island-1 and Island-7, where Island-1 is formed from $(a + b + c)$ and Island-7 is formed from $(!a + !b + !c)$. Present these edges as follows:

Edges $= \{(p, q), (p, r), (q, r), \ldots\}$, with $p, q, \ldots, v$ suitably replaced.

(d) Any extra details you would like to write (in 2-3 sentences max).

Hints and additional requirements:

- First, draw out some of the 7-clique that is allowed to be formed, on a piece of a paper. (It will get very busy very soon, so only draw some of the clique.) This clique is directly based on the satisfying instance.
- Then, looking at this drawing and the satisfying instance, present that clique as requested below.
  - List the seven nodes of the clique in the fashion "$(p, q, !r, s, !t, u, v)$", picking one node per island. (Of course, $p, !r$, etc., are merely examples—we have "a," "!b" etc as the real node names; use the real literal names.).
  - If you can choose a and b from Island-1, pick a from Island-1; then move on to Island-2, and do the same. *In other words, pick the lexically first (ignoring "!") literal from each island. That is, !a comes before b, a comes before !b, etc.*
  - List all the edges between Island-1 and Island-7, where Island-1 is formed from $(a + b + c)$ and Island-7 is formed from $(!a + !b + !c)$. List the edges by mentioning pairs of nodes such as "$(p, !q)$". *List only those edges where the* **literals** *at both end-points are true in the satisfying assignment of ϕ.*