CS 501: Formal Language Theory **Spring 2023**

Final Examination SOLUTIONS

Released: 5/24/2023, 6:00 pm EST Time Limit: 120 minutes Due: 5/24/2023, 8:00 pm EST

Note: LaTeX template courtesy of UC Berkeley EECS dept.

Instructions. This final contains seven questions on pages 1-9, for a total of 100 points. You have a total of 120 minutes to complete it. There will be a supplemental sheet with some definitions on it.

The final is an **individual effort**. You are required to write your entire attempt yourself, and are forbidden from consulting anyone else. Failure to abide by this will result in immediate failure from the course, among other consequences.

- This is a closed-book exam, with no books, notes, calculators, or collaboration.

Submissions. Please write your answers on the test sheet. You may use the backs of pages, but let us know in the indicated place for each question where we can find the rest of your answer. Page 10 and 11 is a supplemental sheet with useful information – do not put answers on it.

Name:_____

SPIRE ID:_____

1. **(10 × 2 points) Unjustified True/False Questions.** For each of the following questions, indicate simply whether it is TRUE or FALSE. *No justification needed or wanted.*

   (a) Let $\Sigma = \{0, 1\}$. There exists a bijection from all strings over Σ to all languages over Σ.

   FALSE. Diagonalization proves exactly that this cannot be done.

   (b) If X is a context-free language over the alphabet $\{a, b, c, \ldots, z\}$ that contains all of the strings $a^n b^n c^n$ for all n with $n \geq 0$, then it is not the case that it must contain at least one other string in $a^* b^* c^*$.

   FALSE. If it didn't, you could intersect it with the regular language $a^* b^* c^*$ and get a context-free grammar for expression for the famously known non-CFL.

   (c) If Y is any infinite TR language, and n is any natural, it is possible that there does not exist a string in Y that is compressible by n bits.

   FALSE. Use the Recursion Theorem to find a machine M that finds its own description, find the length m of that description, and run an enumerator for Y to find the first string longer than $m + n$.

   (d) Let Z be a language in the class NC^1. Then there exists a family of boolean *formulas* $\{C_n : n \geq 0\}$ such that each C_n decides membership in Z for strings of length n, the depth of C_n is $O(\log n)$, the size of C_n is polynomial in n, and its gates are binary AND, binary OR, and unary NOT. Note that a circuit is a **formula** if it is a tree, meaning that a given gate may feed into only one other gate.

   TRUE. The definition tells us that there is a circuit family meeting all these conditions except being a formula. But we can convert the circuit into a formula by duplicating gates. The size of the resulting formula is at most 2^d where d is the depth, and $2^{O(\log n)}$ is polynomial in n.

   (e) Let D_7-SAT be the set of satisfiable formulas with depth at most 7, unbounded fan-in OR and NOT gates, and unary NOT gates. Then, like CIRCUIT-SAT, D_7-SAT is NP-complete (assuming $P \neq NP$).

   TRUE. It's clear that D_7-SAT is in NP, since we can guess a satisfying instance. We know that D_7-SAT is hard for NP because it is a special case of 3-SAT.

   (f) It is not the case that the class L is a strict subset of the context-free languages.

   TRUE. It is not a subset at all, because $\{a^n b^n c^n : n \geq 0\}$ is in L but not context-free.

   (g) It is not the case that the class of context-free languages is closed under union, concatenation, Kleene star, and intersection.

   TRUE. It is closed for the first three, but $\{a^n b^n c^n : n \leq o\}$ is the intersection of two CFL's, as we showed in a discussion this term.

   (h) Let SUBGRAPH-ISO be the set of two undirected graphs G and H where G has a subgraph isomorphic to H. Then SUBGRAPH-ISO is NP-complete (assuming $P \neq NP$).

   TRUE: CLIQUE $\leq_p$ SUBGRAPH-ISO*, if $\langle G, i \rangle$ is an instance of CLIQUE, then reduction is simply to $\langle G, K_i \rangle$ where K_i is a complete graph with i. And it's obvious that SUBGRAPH-ISO is in the class NP.*

   (i) If X is a language in NP, then it is possible that the language X^* is not in NP.

   FALSE. On an input w, guess the partition of w into substrings, then verify that each substring is in X. This is possible if and only if w is in X^*, and can be done in NP.

   (j) Using the Recursion Theorem, we can build a Turing machine B where we can obtain its own description, find out whether it accepts, and reverse the answer.

   FALSE. Sipser does this in Chapter 6 in so many words, but he is doing it under the assumption that the halting problem is decidable. If we could do this, we would have a contradiction.

2. **(5 × 6 points) Justified True/False Questions.** For each of the following questions, indicate whether it is TRUE or FALSE, *and provide a brief justification (i.e. either a proof or a counterexample).*

   (a) The language $X = \{\langle M \rangle : M$ is a single tape TM and there exists a number k such that M never moves right past the k'th cell, on any input$\}$ is not TR.

   FALSE. It is TR because we can successively test for each k whether M runs within k steps on each input string of length $\leq k$, since letters after the first k are never seen. We run each computation with a clock, until we accept, reject, tries to go too far right, or runs until we know that it is in a loop.

   (b) The language X from Question 2(a) is TD.

   FALSE. We show $A_{TM} \leq_m X$. Given an input $\langle M, w \rangle$, we make a machine N that erases its input and runs M on w, with two changes. If the computation of M on w rejects, N runs to the right forever. Also, it keeps a marker to the right of the input, and moves it one space to the right every for every step of M's computation. In this way, if the computation of M on w never halts, N is not in X. If M does accept w, N uses a fixed amount of space on any input, and thus N is in X.

   (c) Let f be the function that takes a binary number and outputs its unary representation. Then f is a polynomial-time computable function if and only if $P = NP$.

   FALSE. This function f cannot be polynomial-time computable no matter whether $P = NP$. Since its output is size 2^n when the input size is n, it cannot complete its computation in polynomial time.

   (d) On Homework #2, we proved the given any directed graph G and any nodes s and t, we can construct a grammar whose language is non-empty if and only if there is a path from s to t. Then, assuming that this construction can be carried in logspace, and if the language E_{CFL} were in the class L, it would still be possible that L is a strict subset of NL (assuming $L \neq NL$).

   FALSE. We know that REACH is complete for NL under log-space reductions. The result of Question 6 lets us map any directed graph to a grammar, proving REACH $\leq_L \overline{E_{CFL}}$*, as long as we know that the reduction can be computed in log-space. The latter is true because the description of the grammar can be produced by loops over i, j, and k, checking G for each edge to see whether it is there. If E_{CFL} were in L, downward closure of L under log-space reductions would put an NL-complete language into L, proving $L = NL$.*

   (e) If every language in the class P can be decided by a polynomial-size log-space uniform family of circuits, then $L = P$ must be true, even assuming $L \neq P$.

   FALSE. The statement that P is contained in U_L-PSIZE is true unconditionally, while $L = P$ is unknown and here we are assuming it to be false. I was disappointed that hardly anyone recognized this important definition that occurs here verbatim.

3. **(10 points) MAX-2SAT.** The language MAX-2SAT is the set of all pairs $\langle \phi, k \rangle$ where ϕ is a 2-CNF formula (a set of clauses, each the OR of one or two literals) and there exists a setting of the variables making k or more of the clauses true. Prove that MAX-2SAT is NP-complete. (**Hint:** Reduce from 3-SAT. For any clause ψ in the 3-CNF formula, with literals ℓ_1, ℓ_2, and ℓ_3, let c be a new variable and consider the ten 2-CNF clauses $\{\ell_1, \ell_2, \ell_3, c, \neg\ell_1 \vee \neg\ell_2, \neg\ell_1 \vee \neg\ell_3, \neg\ell_2 \vee \neg\ell_3, \ell_1 \vee \neg c, \ell_2 \vee \neg c, \ell_3 \vee \neg c\}$. How many of these clauses are satisfied if ψ is satisfied? How many could be satisfied if ψ is not satisfied? Note that the list of clauses above is symmetric in ℓ_1, ℓ_2, and ℓ_3, which might make counting easier.)

   If all three of the ℓ_i's are false, we get four clauses if c is true and six if c is false. If exactly one of the ℓ_i's is true, we get six with c true and seven with c false. If exactly two of the ℓ_i's are true, we get seven each way. If all three ℓ_i's are true, we get seven with c true and six with c false. So if there are m clauses in all in ϕ, so that we have $10m$ clauses in ψ, we can pick the c's to get $7m$ satisfied clauses in ψ if and only if the setting of the original literal satisfies ϕ. This completes the reduction to show 3-SAT $\leq_p$ MAX-2SAT. It's obvious that MAX-2SAT is in NP, since we can use the setting making k or more of the clauses as a certificate, and it's easy to verify this in polynomial time. So MAX-2SAT is NP-complete.

4. **(10 points) Bottleneck Sets.** Let BOTTLENECK-SET be the language $\{\langle G, s, t, S \rangle : G$ is a directed graph, s and t are nodes, and for any node x in G, (there is a path from s to t without using x) if and only if $x \notin S$ $\}$. Hence S is exactly those nodes that can be deleted, individually, to prevent the path from s to t. Prove that BOTTLENECK-SET is complete for the class NL under log-space reductions.

   To prove it in NL, we must prove that there is a path without x whenever $x \in S$, and that there is no path without x whenever $x \notin L$. We can guess and verify the paths that should be there, and use Immerman-Szelepczenyi to guess and verify the absence of the paths that should not be there. For the hardness, we can reduce the complement of REACH to BOTTLENECK-SET by mapping $\langle G, s, t \rangle$ to $\langle G, s, t, S \rangle$, where S contains all the nodes.

5. **(10 points) Equivalence of Regular Expressions.** Prove that the language EQ_{REG} is PSPACE-complete under poly-time reductions.

   We first use poly-time reductions to convert REG to NFA. For the hardness result, we quote the result that ALL_{NFA} is hard, and prove $ALL_{NFA} \leq_p EQ_{NFA}$ by noting that the former is a special case of the latter. This argument is complicated by the fact that the computation graph involved in our reduction (from a generic PSPACE machine) is exponential size. We can guess a string, though, without writing it down, and guess its computation path through the computation graph while remembering only one node of the graph at a time.

   To show membership in PSPACE, we follow Example 8.4 in Sipser to show that the complement of EQ_{NFA} is in NPSPACE, by guessing a string w and running the two corresponding DFA's on w, implicitly one time step at a time, to show that one is in the language of one DFA and the other is not. This uses polynomial space because we can maintain two DFA states (there is one bit per state of each of the two NFA's), and the length of w is polynomial. Most of you claimed that the regular expressions could be converted into equivalent DFA's in poly-time, which is provably false because those DFA's have exponential size.

6. **(10 points) Rotational Closure.** If U is any language, the **rotational closure** of U, called $RC(U)$, is that set of all strings w such that there exist strings x and y, $w = yx$, and $xy \in U$. For example, if $U = \{abc\}$, $RC(U) = \{abc, bca, cab\}$. A class C of languages is said to be **closed under rotational closure** if $RC(U) \in C$ whenever $U \in C$.

   (a) (6) Prove that the class of regular languages is closed under rotational closure.

   (b) (2) Prove that the class P is closed under rotational closure.

   (c) (2) Prove that the class of TD languages is closed under rotational closure.

   (a) If D is a DFA, we build an NFA N such that $L(N) = RC(L(D))$. This suffices to show that $RC(U)$ is regular whenever U is regular. Given an input w, N first guesses a state p, then guesses when it has read the string y. If $\delta^*(p, y)$ is not a final state, it rejects. It then jumps to the start state i and reads the rest of the string, x. If $\delta^*(i, x, p)$ it accepts – otherwise it rejects. If w can be written as yx where xy is in $L(D)$, and makes the right guesses of y and p, it will accept. And if it accepts, it must have read a string y and then x, such that xy was accepted.

   (b) Given any input w, our algorithm computes each of the strings y and x for any parsing of $w = yx$. It then tests whether xy is in U. If any of these strings xy are in U, it accepts, and otherwise it rejects. The time is n times the time to test n strings of length n for membership in U – this is n times a polynomial in n, which is polynomial.

   (c) This proceeds exactly as for (b). Since we have a decider for U, we can run n different tests for membership in U, and each one will terminate in a finite time.

7. **(10 points) LINEAR-SAT.** Define a **linear TM** to be a Turing machine M, with two tapes, and a constant c such that M always halts within cn steps on any input of length n. Define the language LINEAR-SAT to be the set of all linear TM's M such that there exists some string w that M accepts. What is the complexity of the language LINEAR-SAT?

   It is TR but not TD. It is TR because we can search forever for any string w, and we can test each candidate in finite time. It is not TD because $A_{TM} \leq_m \text{LINEAR} - \text{SAT}$. For the reduction, on any input $\langle M, w \rangle$, we design a linear TM that checks whether its input is an accepting computation history of M on w. We can do this with two tapes and linear time, first moving one tape, and then both, to keep putting one tape at the head position of configuration c_i, and the other at the corresponding position of c_{i+1}.

   The name "LINEAR-SAT" was deliberately misleading, suggesting that this might be NP-complete. Perhaps a less misleading name would be "$\overline{E}_{LINEAR}$". In the event, hardly any of you considered the input size of this language – it is the description length of the linear-time TM rather than any string running on it.