CS 501: Formal Language Theory **Spring 2024**

Final Examination **Solutions**

Released: 5/15/2024, 6:00 pm EST Time Limit: 120 minutes Due: 5/15/2024, 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 with 5 extra credit. 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.

---

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) The circuit classes in the NC hierarchy satisfy the inclusions $NC^0 \subseteq AC^0 \subseteq NC^1 \subseteq AC^1 \subseteq \ldots$, and none of these inclusions are known to be strict.

   FALSE. Those inclusions are all correct, but we argued in lecture that $NC^0 \neq AC^0$, and we asserted the First-Saxe-Sipser-Ajtai theorem that says $AC^0 \neq NC^1$.

   (b) If a language Y and its complement $\overline{Y}$ are both context-free, then it is regular.

   FALSE. The language $\{a^n b^n : n \geq 0\}$ and its complement are both CFL's.

   (c) If there exists a linear-time algorithm for the vertex cover problem VC, then the classes NP and co-NP must be equal.

   TRUE. P = NP because we have an NP-complete problem in P, and P is closed under complementation.

   (d) It is not known whether the TQBF problem (the set of true quantified boolean formulas) is solvable in polynomial time.

   TRUE. This is widely conjectured to be false, since it would imply P = PSPACE, but this is unknown.

   (e) If a language is defined by a context-sensitive grammar, it can be decided by a deterministic Turing machine using $O(n^2)$ space.

   TRUE. Such a language is also the language of an NLBA and thus is in $NSPACE(n)$, and the result follows from Savitch's Theorem.

   (f) If $|\Sigma| = 1$, the Post Correspondence Problem over the alphabet Σ is Turing decidable.

   TRUE. We have a match if and only if there either is any domino with the same string on top and bottom, or two strings where one has the top string longer and the other has the bottom string longer.

   (g) With $\Sigma = \{0, 1\}$, any sufficiently long palindrome fails to be incompressible.

   TRUE. Such a string of length n can be described by giving its first $n/2$ or $(n+1)/$ bits, then indicating that it either an even-length or an odd-length palindrome. So $K(w) \leq n/2 + O(1)$, which is less than n for sufficiently large n.

   (h) The Regular Language Pumping Lemma may be used to prove a language to be regular, by showing that every string in the language satisfies the conclusion of the lemma.

   FALSE. Every regular language satisfies the RLPL, but the converse in not true in general.

   (i) Assuming the result of Question 4 below, the class POLYNBP is closed under complementation.

   TRUE, because we prove below that it is the same class as NL.

   (j) The set SELF of all Turing machines M that output their own descriptions is a TR language, but is not TD.

   TRUE. We can build a recognizer for this language that just runs M on itself and, if it halts, compares its output to its description. We can prove SELF to not be TD because because we can use the Recursion Theorem, given any M and w, to build a TM that outputs its own description if M accepts w and doesn't halt if not.

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) On the midterm, we defined a **computable collection of TR languages** $(L_1, L_2, L_3, \ldots)$, where each language L_i is the language of some TM M_i, and there is a single computable function q that produces a description of M_i on input i. Let INT be the set of all languages that are intersections of computable collections of TR languages. Then INT is complete for the Π_2 level of the Arithmetic Hierarchy.

   TRUE. Any element X of INT is in Π_2 because we can express $w \in X$ as $\forall i : \exists c : ACH(M_i, c, w)$. If Y is an arbitrary language in INT, so that $w \in Y$ is defined as $\forall u : \exists v : R(u, v, w)$ where R is a TD predicate, we can define a computable collection of Turing machines where M_i, on input w, searches for any string v such that $R(i, v, w)$ is true and accept if there is. The intersection of all the languages L_i for this collection is exactly Y.

   (b) We have seen that the language REACH, of triples (G, s, t) such that G is a directed graph and there is a path from s to t in G, is NL-complete under $\leq_L$ reductions. Define the language DAG-REACH to be the subset of REACH where each graph G must be a directed acyclic graph, with its nodes numbered such that any edge goes from smaller to larger node numbers. Then DAG-REACH is also NL-complete under $\leq_L$ reductions.

   TRUE. To see that DAG-PATH is in NL, a log-space NDTM can first verify the condition on the edges and node numbers of G and then guess a path from s to t. We can prove completeness by showing PATH $\leq_L$ DAG-PATH. Given an arbitrary directed graph G with n nodes, make a new graph H with nodes (i, u) where u is a node of G and i is a natural with $0 \leq i \leq n-1$. For every $i < n-1$, H has edges from (i, u) to $(i+1, u)$ and from $(i, u$ to $(i+1, v)$ for every edge (u, v) in G. We number the nodes of H with i as the high-order bits, so that H is clearly a DAG. Then there is a path from s to t in G if and only there is a path from $(0, s)$ to $(n-1, t)$ in H.

   (c) The language $OVERLAP_{CFG}$ is the set of all pairs (G, H) of CFG's such that $L(G) \cap L(H) \neq \emptyset$. Then the language $OVERLAP_{CFG}$ is Turing decidable.

   FALSE. We can reduce $PCP \leq_m OVERLAP_{CFG}$ by a mapping a PCP instance to a grammar G with rules $S \to t_i S a_i$ for all top strings t_i, and $S \to \varepsilon$, and a grammar H with rules $S \to b_i a_i$ for all bottom strings b_i, and $S \to \varepsilon$. Any string common to $L(G)$ and $L(H)$ could only come from a match in the PCP instance.

   (d) Define a **Sillier TM** to be a one-tape Turing machine M such that if q is a state and a and b are any two letters in M's tape alphabet, then $\delta(q, a) = \delta(q, b)$. Then the language $L(M)$ of any Sillier TM M is finite, and thus the language $A_{SillierTM}$ is TD.

   FALSE. Since the Sillier TM behaves the same on any input string, its language is either empty or Σ^*. So it is TD (since we can easily tell which), but the language need not be infinite.

   (e) Let S be an arbitrary *finite* set of undirected graphs. Define the language $Contains_S$ to be the set of undirected graphs G such that there exists a graph H in S such that H is a subgraph of G (see the definition on the supplemental sheet). Then, assuming $P \neq NP$, the language $Contains_S$ is not in the class P.

   FALSE. For any graph H in S, with k nodes, we can test all $\binom{n}{k}k! = O(n^k)$ possible assignments of vertices of H to vertices of G, and for each one test whether this assignment proves that H is a subgraph of G. Since k is a constant, this all happens in time $O(n^k)$ time, a polynomial. The total time will be n^ℓ, where ℓ is the number of nodes in the largest graph in S.

3. **(10 points + 5 XC) Wordle and Regular Languages.** Let W be the set of all five-letter strings over the alphabet $\Sigma = \{a, b, \ldots, z\}$ and let D be a arbitrary subset of W. We define the language C_D to be the set of all strings (of whatever length) that contain five consecutive letters that form a string in D.

   - (a, 5) Prove, by any method, that for any D, the language C_D is regular.
     There are lots of ways to do this, easiest if describe a regular expression that is the sum of $\Sigma^* w \Sigma^*$ for all $w \in D$
   - (b, 5) Prove a finite upper bound on the number of Myhill-Nerode classes in C_D, for any D. (Note that this result implies that of part (a), but you may want to do (a) separately in case your solution to (b) is wrong.)
     Every string in C_D is equivalent to any other, any other two strings with the last same four letters are equivalent, you can have a separate class for every string of length three or less, so this is $1 + 1 + 26 + 26^2 + 26^3 + 26 + 4$.
   - (c, 5XC) Prove that for any D with $|D| = n$, there are at most $5n + 1$ Myhill-Nerode classes for C_D. (Again, solving this also solves both (a) and (b), but there are easier ways to solve (b).)
     Induction on n. If $n = 0$, we need only one rejecting state and $1 \leq 5(0) + 1$. If $n = 1$, we need six states $\{0, 1, 2, 3, 4, 5\}$, where state i indicates that the we have not seen the word w in D yet, and that the last i letters have been a prefix of w. In this case $6 \leq 5(1) + 1$. Assume that we have a DFA for C_D, and we need to make a DFA for $C_{D'}$, where D' is D plus one new word w. We make four new states w_1, w_2, w_3, and w_4 where the input is taken to w_i if we have not yet seen a word in D', and the last i letters are a prefix of w. If is possible that one or more of these states coincide with other states for other strings. For example, if MOOSE was already in D, and w is the new word MOUSE, we don't need to add w_1 and w_2 because already have states for the prefixes M and MO. So from MO, it still goes to MOO on an O, but goes to the new state w_3 for MOU if the next letter is U.
     This construction takes at most $4n + 2$ states for $|D| = n$, which is better than $5n + 1$ as long as n is positive. I gave you the weaker bound because you might come up with a construction that meets that bound.

4. **(10 points) Nondeterministic Branching Programs.** In Discussion #10, we defined **branching programs** and the language POLYBP of languages defined by log-space uniform polynomial-size branching programs. Here we define **nondeterministic branching programs (NBP's)**, which are like branching programs but where a non-leaf node may have any number of 0-edges and any number of 1 edges, as well as **free edges** that may be taken whatever the input variable is. An input is in the language of the NBP if it is possible to go from the start node to an accepting leaf, using either ordinary edges matching the input or free edges. We define the class POLYNBP to the the set of languages of log-space uniform NBP families whose size (number of nodes) is bounded by a polynomial in their number of inputs. Prove that the class POLYNBP is equal to the class NL.

   To show that POLYNBP contains NL, we note that a log-space NDTM can first construct the NBP (since the family is log-space uniform), and then maintain its location as it goes from the start node to a leaf, following the rules according to the input, and guessing whether to take any free edges. The input is in the language of the NBP if and only if there is a path to an accepting node, and this NDTM can take this path if and only if it exists.

   To show that NL contains POLYBP, consider an arbitrary language in NL, defined by a log-space NDTM N with a clock as in Discussion #10. For any input size n make an NBP where each node represents a configuration of N on inputs of size n. The start node corresponds to the start configuration, and for each non-halting configuration, we define 0-edges and 1-edges for all moves that N might make from that configuration, for the appropriate input bit on the given cell of N's input tape. Finally we mark halting configurations as accepting or rejecting. The NBP can choose a path to an accepting node if and only if it is possible for N to accept the input.

5. **(10 points) Classroom Scheduling is NP-Complete.** A University registrar has n students and m courses for which she needs to assign final exam times, using k time slots. If x and y are two different courses, and there is any student in both courses, she may not give them the same time slot. Let SCHED be the set of tuples (S, C, A, k) such that:

   - S is a list of students,
   - C is a list of courses,
   - A is the set of pairs (s, c) in $S \times C$ such student s is in course c, and
   - k is the number of time slots, such that
   - It is possible to assign each course in C to a time slot with no student being assigned two courses with the same time slot.

   Prove that this language is NP-complete.

   To prove membership in NP, we construct a verifier that accepts a function $f : A \to \{1, \ldots, k\}$ if $\forall x : \forall y : ((x \neq y) \wedge f(x) = f(y)) \to \neg\exists s : (s, x) \in A \wedge (s, y) \in A$.

   To prove NP-hardness, we show 3-COLOR $\leq_p$ SCHED. Given an arbitrary undirected graph G with n vertices, we let C be the vertices of G, have one student s in S for each edge (u, v) of G and add (s, u) and (s, v) to A, and set $k = 3$. A schedule from A to $\{1, 2, 3\}$ exists that obeys the constraints if and only if the graph G is 3-colorable.

6. **(10 points) How Many a's and b's?** Let $\Sigma = \{a, b\}$ and define a language X that is a subset of the regular language a^*b^*. A string a^ib^j is in X if and only if $j \leq i \leq 2j$. That is, there are at least as many a's as there are b's, and there are not more a's than twice as many as there are b's. Find **both**:

   - A context-free grammar for X, and
   - A pushdown automaton for X.

   You may use standard constructions to convert one to the other, but we want *explicit* examples for both the CFG and PDA. Producing one and just claiming that the other exists will get you no points for the other.

   Grammar: $S \to aSb$, $S \to aaSb$, $S \to \varepsilon$. Clearly any derivation from this grammar will use the first rule some number x times and use the second rule some y times. We will have $2x + y$ a's and $x + y$ b's, and we will definitely have $x + y \leq 2x + y \leq 2(x + y)$. For any particular string a^ib^j following the rule, we can set $x = i - j$ and $y = 2j - 1$.

   PDA: We could use the top-down parser, but the following might be simpler. State 1 is the start state with only transition $(e, e, \$)$ to State 2. (I'm using e in place of ε.) State 2 has a transition (a, e, a) to State 3. State 3 has two transitions (a, e, a) and (a, e, e), both to State 2. Both States 2 and 3 also have transitions (e, e, e) to State 4. State 4 has a loop (b, a, e), and a transition $(e, \$, e)$ to the final State 5. The only way for the PDA to accept is to read the a's and put between half of them and all of them onto the stack. Then it has to match these a's exactly to b's from the input.

7. **(10 points) Implicitly Defined Circuits.** We're going to define a family of Boolean circuits, but they will be too large to be given as input in the normal way. An **Implicitly Defined Circuit (IDC)** with n inputs is defined by a poly-time Turing machine M which can be used to get an exponential number of NC^0 circuits, each with n inputs and n outputs. (Recall that an NC^0 circuit is one where each output depends on only a constant number of inputs.) For any input size n and any number i with $1 \leq i \leq 2^{p(n)}$, M outputs a description of the NC^0 circuit C_i. (Here $p(n)$ is a fixed polynomial in n.) The IDC is the composition of all the circuits, so that the n inputs feed into C_1, which feed into C_2, and so on, until the rightmost output bit of $C_{2^{p(n)}}$ is the output of the IDC.

   The language IDCVAL is the set of pairs (M, w) where M is such a Turing machine, w is a binary string, and the corresponding IDC accept w. Prove that the language IDCVAL is PSPACE-complete.

   We can compute the output of an IDC in PSPACE as follows. For each i, maintaining our count because i can be represented in only $p(n)$ bits, we take the n-bit input to C_i, find C_i using M, and compute its n-bit output. We simply return the rightmost bit of the last circuit.

   Given an arbitrary PSPACE deterministic TM with n input bits and space bound $p(n)$, we define a Cook-Levin like tableau with $2^{p(n)}$ rows, each a binary string of length $p(n)$. We can design an NC^0 circuit to convert a row of the tableau to the one above it, since each bit of the next row depends only on a constant number of bits in the previous row. Assuming we normalize the TM to run for exactly $2^{p(n)}$ steps and accepts by signaling in its rightmost bit, the corresponding IDC will compute the result of TM's computation.