NAME: _____

SPIRE ID: _____

COMPSCI 501
Formal Language Theory
Solutions to Midterm Spring 2024

D. A. M. Barrington 3 April 2024

DIRECTIONS:

- Answer the problems on the exam pages.
- There are seven problems on pages 2-7, some with multiple parts, for 100 total points plus 5 extra credit. Final scale will be determined after the exam.
- The supplemental page 8 has definitions for your use and should not be handed in.
- If you need extra space use the back of a page – both sides will be scanned.
- No books, notes, calculators, or collaboration.

1	/20
2	/30
3	/10
4	/10
5	/10
6	/10
7	/10+5
Total	/100+5

Question 1 (20): These are ten true/false statements, with no justification needed or wanted (2 points each):

- (a, 2) The intersection of any finite collection of CFL's is decidable, but it is not necessarily a CFL itself.

  TRUE. Any CFL is decidable, and any intersection of TD languages is TD. But in discussion we saw a pair of CFL's whose intersection is not a CFL.

  76% correct.

- (b, 2) Let X and Y be two languages such that there exists some function f such that for all strings w, $(w \in X) \leftrightarrow (f(w) \in Y)$. Then it must be the case that $X \leq_m Y$.

  FALSE. This would be true if we said that f were computable, but we didn't.

  38% correct. We don't feel guilty about this one at all. Read the text, not what you imagine it says!

- (c, 2) There exists an undecidable language with a unary alphabet, that is, with $|\Sigma| = 1$ where Σ is the input alphabet.

  TRUE. There is a computable bijection from unary strings to binary strings, so any language over a binary alphabet can be mapped to a unary language.

  64% correct.

- (d, 2) Let A and B be two languages such that B is undecidable, and there exists a reduction showing $A \leq_m B$. Then A must also be undecidable.

  FALSE. As long as A and B each have a string in it, and each have a string not in it, any TD language A can be reduced to B. It doesn't matter how difficult it might be to decide membership in B for any other strings.

  81% correct.

- (e, 2) Recall that an n-bit string w is **incompressible** if its Kolmogorov complexity satisfies $K(w) \geq n$. If v and w are both incompressible n-bit strings, then the string vw is also incompressible.

  FALSE. If v and w are the same string, for example, vw can be described as "two copies of v", and so $K(vw) \leq n + c$, for some constant c representing the size of the TM that makes the two copies. As long as n is large enough, $K(vw)$ is less the length of vw, $2n$, and so is not incompressible.

  59% correct.

- (f, 2) Let Σ be any non-empty alphabet. Then the language $L = \{a^n b^n : n \geq 0, a \in \Sigma, b \in \Sigma\}$ must not be regular.

  FALSE. This fails if Σ has only one letter, since in that case a and b must be the same letter and $L = (aa)^*$.

  40% correct. This was genuinely tricky.

- (g, 2) It is possible to convert any context-free grammar G into an NFA N such that $L(G) = L(N)$.

  FALSE. The language of an NFA must be regular, and we know that there exist grammars with non-regular languages.

  90% correct.

- (h, 2) Let f be any function from $\mathbf{N} \times \mathbf{N}$ to $\{0, 1\}$, where $\mathbf{N}$ is the set of natural numbers. Then there does not exist a function d from $\mathbf{N}$ to $\{0, 1\}$ such that for any n, $d(n) \neq f(n, n)$.

  FALSE. The function exists, since we can define $d(n)$ to be $1 - f(n, n)$. The point of diagonalization is that the function $d(n)$ is not equal to the function $f(i, n)$ for any i.

  86% correct. Better reading comprehension on this one!

- (i, 2) For any string w, define a TM to be a **w-printer** if, on any input, it halts with w on its tape. Then there exists some Turing machine M such that, on any input w, M halts with the description of some w-printer on its tape.

  TRUE. We did this in lecture as part of our proof of the Recursion Theorem.

  94% correct.

- (j, 2) Let M be any Turing machine and let t be any partial function computed by it, so that for any input x, $t(x)$ is the string left on M's tape if it halts (if it doesn't halt, then $t(x)$ is not defined). Then there exists a Turing machine R such that for any input string w, R accepts w if and only if the value $t(w)$ is not defined.

  FALSE. The set of strings on which M halts is an arbitrary TR language, and this R would only exist if that language were also co-TR.

  79% correct.

Question 2 (30): These are five true-false questions, with brief justification required. Three points for each correct boolean answer, and up to three points per question for the justification:

- (a, 6) Let c be any positive natural. A c-PDA is a (nondeterministic) pushdown automaton such that its stack never contains more than c characters. Then the language of any c-PDA is regular.

  TRUE. There are only a finite number of possible contents of the stack. We can convert the c-PDA to an NFA with a larger state set, using the state set to specify the stack contents. Since the language of the c-PDA is thus also the language of an NFA, it is regular.

  Mean score 4.4/6.0, 51% full credit, 77% correct boolean. Some people overlooked that the c-PDA is nondeterministic, so in order to convert it to a DFA you need to go through an NFA.

- (b, 6) Let X be any non-empty TD language whose complement $\overline{X}$ is also non-empty. Then $\overline{X} \leq_m X$.

  TRUE. Let y be an element of X and let n be an element of $\overline{X}$. Our reduction f on input w first determines whether $w \in X$, using the assumed decider for X. If $w \in X$, then we set $f(x) = n$, and if $w \notin X$, we set $f(w)$ to be y.

  Mean score 4.0/6.0, 22% full credit, 83% correct boolean. To show this, you need to show a mapping from strings to strings that meets the conditions. Lots of people wanted to "reverse the answer", or "reverse the states", when you need to change strings to strings rather than change a TM to a TM.

- (c, 6) Let X be any non-empty TR language whose complement $\overline{X}$ is also non-empty. Then $\overline{X} \leq_m X$.

  FALSE. Suppose X is A_{TM}. If there were a reduction from $\overline{A_{TM}}$ to A_{TM}, the language $\overline{A_{TM}}$ would be TR, since the TR languages are closed downward under $\leq_m$. But we know that $\overline{A_{TM}}$ is not TR.

  Mean score 5.1/6.0, 67% full credit, 92% correct boolean. I'm glad that this one went so well.

- (d, 6) Recall that PCP is the language of finite sets of dominoes that contain a match. Then the language PCP is TR-complete.

  TRUE. We know that PCP is TR because we can search, using a TM, for any sequence of dominoes that forms a match. The reduction $A_{TM} \leq_m PCP$ was carried out in Sipser and in the lectures.

  Mean score 3.9/6.0, 14% full credit, 81% correct boolean. There are two parts to being TR-complete, and in general I gave two of the three points for a good justification of one of them.

- (e, 6) The language EQ_{CFG} is co-TR, but not TD.

  TRUE. An input to EQ_{CFG} would be a pair of grammars (G, H), with the true instances being grammars such that $L(G) = L(H)$. A pair is *not* in EQ_{CFG} if and only if there exists a string w that is in one of the languages $L(G)$ or $L(H)$, but not in the other. The complement of EQ_{CFG} is TR, since (with a TM) we can search for such a string w, then verify that w is in one language and not the other, using the known decider for the language A_{CFG}.

  It remains to show that EQ_{CFG} is not TD. But we know that ALL_{CFG} is not TD, from Sipser and from lectures. We can reduce ALL_{CFG} to EQ_{CFG} by mapping an arbitrary TM M to the pair (M, Z) where Z is a TM that accepts any input.

  Mean score 3.9/6.0, 24% full credit, 71% correct boolean. A common mistake was to confuse finding a string in one language but not in the other, which you need to prove that the languages are different, with proving that languages are equal. The latter is not TR, as you can't determine that they are equal without looking at infinitely many cases.

Definitions: Some of Questions 3-5 deal with two new definitions. If X is any language over some alphabet Σ, we define the **cube root** of X, called $CR(X)$, to be the language $\{w : www \in X\}$. Similarly, the **cube** of X, called $Cube(X)$, is the language $\{www : w \in X\}$.

Question 3 (10): Prove that if X is any regular language over any finite alphabet Σ, then $CR(X)$ is also a regular language.

Let D be a DFA for the language X. For every state q of D, build a new DFA D_q which is identical to D except that q is the start state of D_q. Build a DFA E that is the direct product of all the machines D_q for each state. Given any input string w, $\delta^*_E(q_0, w)$ is a tuple containing $\delta^*_D(q, w)$ for each q. For each tuple, determine (off-line) whether the www is accepted by D, by seeing to what state p is taken by w from q_0 in D, then to what state r is taken by w from p, and finally to what state s is taken by w from r. We mark each tuple as a final state in E if and only if s is a final state of D. This is a DFA whose language is the cube root of X.

Here is an alternate proof, which uses more states but may have been easier to come up with. Define an NFA with ε-moves to each of n^2 states, one for each pair (p, q) of states of D. In state (p, q), we want to accept if and only if w takes q_0 to p, from p to q, and from q to a final state. If any string www is in X, there will be exactly one pair (p, q) for which this happens, and if www is not in X, there will be none. For each state (p, q), then, we can build a DFA whose states are the product three copies of D, with the appropriate start states, so that this DFA will accept w if and only those three conditions are satisfied.

Mean score 3.7/10, 4% full credit. One of the full-credit answers used the first proof above, and the other two used the second. In general, I gave 2/10 for fragmentary or fundamentally misguided answers, and 4/10 for some plausible approach – only a few beyond the correct answers got more than 4/10.

Question 4 (10): Prove that there exists a context-free language Y (over the alphabet $\{0, 1\}$) such that $Cube(Y)$ is not context-free.

Let Y be the language 0^*1. So $Cube(Y)$ is the set of strings of the form $0^n10^n10^1$ for all naturals n, and it is easy using the CFLPL that this language is not a CFL. Let p be the alleged pumping length of any G that is claimed to have $L(G) = \{0^n10^n10^n : n \geq 0\}$. We choose $w = 0^p10^p10^p$, and let $w = uvxyz$ be the alleged division into five strings matching the rules of the CFLPL. The string vxy must have length at most p, so it can contain at most one 1 and at most two of the three groups of 0's. If we pump down to the string uxz, the removed strings v and y must, between them, have at least one letter. If a 1 is removed, uxz cannot be in the language since it needs three 1's. If one or more 0's are removed, since at most of the three groups can be affected, the three groups cannot all have the same size and the string uxz is not in the language. Since the language does not obey the CFLPL, it is not a CFL.

Mean score 7.5/10, 55% full credit. Most people (except the ones who reversed $CR(X)$ and $Cube(X)$, were able to recognize that this was a standard CFLPL proof, and carry it out.

Question 5 (10): Here are two more problems about the cube root and cube languages defined above:

- (a) Prove that if Z is a TR language, prove that both $CR(Z)$ and $Cube(Z)$ are also TR languages.

  A TM whose language is $CR(Z)$ takes an input w and runs the machine for Z with input www. A TM whose language is $Cube(Z)$ takes an input w, rejects if it is not of the form uuu for some string u, and otherwise runs the machine for Z with input u.

  Mean score 3.9/5.0, 54% full credit. You do need to specify that your $Cube(Z)$ machine rejects its input if it is not of the form uuu. I think the nicest proof, for both arguments, took an enumerator for Z and created new enumerators for the two new languages.

- (b) Prove that if Q is the language of an LBA, then $CR(Q)$ is also the language of an LBA.

  To make an LBA for $CR(Q)$, we first take an LBA M for Q and expand its tape alphabet so that we can think of its tape as containing three separate strings over M's input alphabet. These strings will operate as a single tape, with the "virtual head" being on one of the three sections. (The actual head may have to run from one end to the other to deal with action on the virtual head exactly at one of the boundaries.) Once this is set up, the new PDA places two other copies on the virtual tape to put www on it. Then it simulates P on www, accepting if P accepts. Since the virtual head is restricted to remain within the space originally occupied by www, the simulated computation remains within the space originally occupied by w, following the rules for an LBA.

  Mean score 2.1/6.0, 12% full credit. The main problem was that if you are taking an input w and trying to run the new input www on it, you need some kind of argument to say that a string with $3n$ letters fits into n cells on the LBA's tape.

Question 6 (10): In this problem we define a **Silly TM** to be a one-tape machine with two read heads and no ability to change letters on its tape. On a given time step, each head may either move right or stay where it is. Prove that the language E_{SillyTM} is undecidable.

Given any TM M and input w, build a Silly TM whose language is any valid ACH of M on w. Begin by checking, character by character, whether the input begins with the start configuration c_0 for input w. If it is, move Head 1 past it, leaving Head 2 in place. If it is not, just reject. For each succeeding phase (if it succeeds) read configuration c_i on Head 2 while passing over the following string with Head 1. Reject unless the configuration seen on Head 1 follows from the one seen on Head 2, according to the rules of M. This means that each new letter is equal to the corresponding old one seen, unless the read head is there. If it is there, the two heads must check that the new letters on Head 1 properly result from what Head 2 sees. Head 1 may also read a new blank symbol as long as both tapes reach the ends of their configurations right after that. This continues until Head 1 reaches the end of the input, and the Silly TM accepts if and only if this last configuration is in the accepting state.

Mean score 3.7/10, 14% full credit. This was kind of a bloodbath. If you started by trying to run a TM on a SillyTM, which cannot modify its tape, I basically stopped reading and gave you 2/10. To get more than 4/10, you had to recognize that this was one of the relatively few cases where we are proving undecidability for a language that is about machines strictly weaker than TM's. As you should realize, every *such case has involved accepting computation histories. I gave 6/10 for answers that amounted to "mumble mumble accepting computation histories mumble", since that was the only track to be on. One clever solution (which got 9/10 because it has some serious mistakes) was to reduce the PCP problem to $\overline{E_{SillyTM}}$, by accepting strings of dominoes of the form (t_i, b_i) that form a match. One head reads the letters of the t strings, and the other reads the same letters of the b strings, each in the right position to read the matching letters at one time. One of the heads, in its spare time, has to also verify that the pairs of strings given are actually valid dominoes. (This proof does not contradict what I said above about ACH's, because the PCP proof used the ACH method.)*

Question 7 (10+5): Given a collection of Turing machines $\{M_i\} = \{M_0, M_1, M_2, \ldots\}$, $\{M_i\}$ is defined to be a **computable collection** if there exists a computable function $q : \{1\}^* \to \Sigma^*$ such that $q(1^n) = \langle M_n \rangle$ (that is, q takes a natural number and maps it to the description of the n'th Turing machine in the collection). In this case we also call $\{L(M_i)\}$ a **computable collection of TR languages**.

- (a) Is the union of any countable collection of TR languages necessarily TR? Prove your answer.

  It is not. Any language whatsoever, since it contains only a countable number of strings, is the union of a collection of singleton sets $\{w\}$, and each of these is clearly TR. We know that there exist non-TR languages.

  Mean score 2.7/5.0, 9% full credit, 62% correct boolean. My most common comment on this one (and probably on all three parts of Q7) was "not really a proof". It is true, for example, that if you tried to use the correct argument for Q7b on Q7a, it would not work because a TM cannot start running any of the given TM's if it doesn't know what they are. But that in itself does not *prove that the statement of 7a is not true. To refute the proposition that all such languages are TR, you need to come up with an example of a non-TR language that is a countable union of TR languages.*

- (b) Is the union of any countable *computable* collection of TR languages necessarily TR? Prove your answer.

  It is. We make a TM M^* such that $L(M^*) = \sum_{i=0}^{\infty} L_1$. On input w, M^* runs each of the machines M_i on w in parallel by dovetailing, accepting if and only one of the M_i's accepts. (For the dovetailing, we could, for each n in series, run each of the first n machines for n steps each.)

  Mean score 3.8/5.0, 47% correct, 91% correct boolean. Pretty much any answer using the word "dovetailing" was good enough. But there were a number of bogus reduction proofs.

- (c, extra credit) Is the intersection of a countable computable collection of TR languages necessarily TR? Prove your answer.

  It is not. We define a countable collection $\{M_i\}$ whose intersection is the non-TR language $\overline{A_{TM}}$. We define the machine M_i to take input $\langle M, w \rangle$, run M on w for i steps, and accept *unless* M has accepted w within that time.

  Let INT be the intersection of the languages of all the languages $\{M_i\}$. Consider any pair $\langle M, w \rangle$. If this M accepts w in t steps, then this pair is *not* in INT because it is not in any $L(M_i)$ for any i with $i \leq t$. But if M does not accept w, this pair *is* in INT because it is in $L(M_i)$ for all i.

  Mean score 1.8/5.0, 4% full credit, 41% correct boolean. That is, 41% of all papers said "no", but a Iarge fraction of the others didn't answer the boolean question at all. There were lots of bad proofs of the correct answer, often involving statements of the form "run M on w, if it reject then...", ignoring the possibility that it might not halt. We made this one extra credit because we thought it was genuinely hard to prove that a specific non-TR language was the intersection of a computable countable collection of TR languages.

Supplemental Page for COMPSCI Midterm Spring 2024

Language A_{TM} (similarly A_{REG}, etc.) Set of pairs (M, w) such that M accepts w

Language ALL_{TM}**:** (similarly ALL_{REG}, etc.) Set of machines that accept all possible strings over their alphabet

Computable Function: Function f from strings to strings such that some Turing machine, on any input w, always halts with $f(w)$ on its tape

Countable Set: Any set that is either finite or has a bijection with the set of natural numbers

Context-Free Grammar (CFG): Grammar where rules allow single non-terminals to be replaced by strings

Context-Free Language (CFL): Definable by a context-free grammar or a PDA

co-TR: A language A is co-TR if and only if its complement $\overline{A}$ is TR.

Language E_{TM}**:** (similarly E_{REG}, etc.) Set of machines with empty languages

Language EQ_{TM}**:** (similarly EQ_{REG}, etc.) Set of pairs of machines with equal languages, *i.e.*, $\{(M_1, M_2) : L(M_1) = L(M_2)\}$.

Linear Bounded Automaton (LBA): A machine like a one-tape Turing machine, but with no additional space to the right of its input.

Mapping Reduction ($\leq_m$): $A \leq_m B$ means that there exists a computable function f such that $\forall w : w \in A \leftrightarrow f(w) \in B$

Pushdown Automaton (PDA): Nondeterministic finite-state machine with an added stack

Language REG_{TM}**:** (similarly REG_{REG}, etc.) Set of machines M such that $L(M)$ is a regular language

Regular Language: Can be defined by a DFA, NFA, or regular expression

Turing Decidable (TD): Is the language of a TM that always halts

Turing Recognizable (TR): Is the language of any TM