NAME: _____

COMPSCI 501
Formal Language Theory
SOLUTIONS to Midterm Spring 2023

D. A. M. Barrington 10 April 2023

DIRECTIONS:

- Answer the problems on the exam pages.
- There are eight problems on pages 2-8, some with multiple parts, for 100 total points plus 10 extra credit. Final scale will be determined after the exam.
- The supplemental page 9 has definitions for your use and should not be handed in.
- If you need extra space use the back of a page.
- No books, notes, calculators, or collaboration.

1	/20
2	/30
3	/10
4	/10
5	/10
6	/10
7	/10
8	/+10
Total	/100+10

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

- (a, 2) The language $\{\langle M\rangle : L(M) \in CFL\}$, where M is a TM, is Turing decidable.
  FALSE, by Rice-Myhill-Shapiro, since this is a non-trivial property of languages.
- (b, 2) The language $\{\langle M\rangle : L(M) \in CFL\}$, where M is a TM, is Turing recognizable.
  FALSE, we map $A_{TM} \leq_m CFL_{TM}$ by mapping $\langle M, w\rangle$ to a machine that takes its input x, accepts if x is in $\{0^n1^n0^n\}$, and otherwise runs M on$w$.
  Literally 90% of you got this one wrong. Rice-Myhill-Shapiro tells you that nontrivial properties of languages are undecidable. Most *of these languages are neither TR nor co-TR – the example that should come to mind is REG_{TM}, which is neither.*
- (c, 2) If $A \leq_m B$, and B is not co-TR, it is possible that A is Turing recognizable.
  TRUE, let A be $\overline{A_{TM}}$, let B be something worse like Π_2-complete.
- (d, 2) If Σ is a finite alphabet, then every string in Σ^* is finite.
  TRUE, Σ^* is defined to be all finite strings over Σ.
  32% of you got it wrong. it was a trick question, of sorts, but the trick was that the statement is trivially true.
- (e, 2) If X and Y are each *nonempty* TD languages, then $X \leq_m Y$ must be true.
  FALSE, if $Y = \Sigma^*$ and X isn't, you can't make a reduction. *47% of you got this wrong – this was fairly tricky because you had to know* why *it's not true that any two TD languages are reducible to one another.*
- (f, 2) If X is a language, and both X and its complement $\overline{X}$ are both context-free, it is possible X is not a regular language.
  TRUE, let $X = \{0^10^n\}$, both X and $\overline{X}$ are CFL's, but X is not regular.
- (g, 2) If A is not a CFL, and $A \leq_m B$, then it is possible that B could be a CFL.
  TRUE, if A and B are non-trivial decidable languages, $A \leq_m B$ is true. So we could have $A = \{0^n1^n0^n\}$ and $B = 0^*1^*$.
  A small majority of you got this wrong. The lesson here is that if you apply a definition that categorizes objects very loosely, the result won't tell you about finer distinctions. The reduction $\leq_m$ tells you about languages being TD or TR, but it doesn't distinguish between decidable languages (unless they are $\emptyset$ or Σ^*) it won't say anything useful about whether languages are CFL's or not.
- (h, 2) Let G be a context-free grammar, let p be a pumping length for it in the CFLPL, and assume that $L(G)$ is an infinite language. Then it is possible that $L(G)$ contains no strings w such that $p \leq |w| \leq 2p$.
  FALSE, since $L(G)$ is infinite, there must be a string of length more than $2p$, keep pumping that string down until it is shorter than $2p$, the next string after that is in the given range.
- (i, 2) Recall that $K(x)$ is the length of the minimal description of a binary string x. It is not the case that for every string, such that x is a member of any regular language, $K(x)$ is strictly less than $|x|$.
  TRUE, Sipser proves that there exists at least one incompressible string, and every string is a member of some regular language.

- (j, 2) There exists a Turing machine M that accepts a string if and only that string is same length as the description of M.

  TRUE, by the Recursion Theorem, obtain your own description and compare it to your input.

  97% of you got this one right.

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) The following language is undecidable: $Q = \{\langle G, x\rangle : \exists u : \exists y : uxy \in L(G)\}$. This language Y is the set of pairs consisting of a context-free grammar G and a string x such that G can generate *some* superstring of x.

  FALSE, use G to build a grammar G' for the language $\Sigma^* x \Sigma^*$. Then we can test whether G is in Q by testing whether G' is in the decidable language E_{CFG}.

  The intuition here should be that this language Q is similar to E_{CFG}, which is decidable.

- (b, 6) The following language is decidable: $R = \{\langle G, x\rangle : \forall u : \forall y : uxy \in L(G)\}$. This language Y is the set of pairs consisting of a context-free grammar G and a string x such that G can generate *every* superstring of x.

  FALSE, we show $ALL_{CFG} \leq_m R$, by mapping G to a pair $(G', \$)$, with $L(G') = L(G)\$L(G)$, and $\$$ is a new terminal.

  In this case the intuition should be that R is similar to ALL_{CFG}, which is undecidable.

- (c, 6) Let $\Sigma = \{0, 1\}$. Let f be any function from Σ to Σ^* (from letters to strings). We extend f to a **homomorphism** f from Σ^* to Σ^* by defining $f(a_1 a_2 \ldots a_n)$ as $f(a_1)f(a_2)\ldots f(a_n)$. Then if S is any regular language, it is possible that the language $f(S) = \{f(w) : w \in S\}$ is not a regular language.

  FALSE, let D be a DFA with $L(D) = S$, and we will build an GNFA N where $L(N) = f(S)$. We let N have the same states, start state, and final states. If D has a move from p to q on the letter a, N has a move from p to q on the string $f(a)$. A second proof, that is perhaps even better, is that you can take a regular expression for S and build a regular expression for $f(S)$ by substituting the regular expression for the string $f(a)$ in place of every occurrence of every letter a.

- (d, 6) Let f be a homomorphism of languages as defined in Question 2(c). Then if T is a regular language, then it is possible that the language $f^{-1}(T) = \{w : f(w) \in T\}$ is not a regular language.

  FALSE, again let D be a DFA with $L(D) = T$, and build D' with the states, start state, and final states of D, but with a transition function so that $\delta'(p, a)$ is $\delta^*(p, f(a))$.

- (e, 6) The following language is undecidable: U is the set of all pairs $\langle M, w\rangle$ such that M is a single-tape TM, w is a string, and M accepts w while never modifying the part of the tape containing the input.

  TRUE, we map $A_{TM} \leq_m U$ by mapping (M, w) to a pair (N, x) where N moves past its input, writes a new tape character $\$$, writes down w in its tape to the right of the $\$$, and then runs M on this new copy of w, using the $\$$ as the left end of its tape. If it accepts, it does so without modifying the tape containing its original input.

  My first thought on this one was to copy w onto a fresh section of the tape, then ignore the first one and work with the second one. The problem is that I don't see how to copy on a one-tape TM without keeping track of where you are in the input, so you know which character to copy. Normally you do this by marking the character where you are, but this requires modifying the input. So I took one of the three justification points if you did this.

Definitions: Some of Questions 3-8 deal with a new definition. If L_1 and L_2 are any two languages over the same alphabet, the **quotient language** L_1/L_2 is the set of all strings x such that there exists a string w such that $xw \in L_1$ and $w \in L_2$. An important special case is when $L_2 = \{w\}$, where w is a specific string, and we write this quotient language as L_1/w.

Question 3 (10): Prove that if V is a regular language, and w is any fixed string, then the language V/w is regular.

Given a DFA D with $L(D) = V$, we build a new DFA D' with $L(D') = V/w$. We have D' have the same states, start state, and transition function, but a different final state set. For any state p in D, let $f(p)$ be the state $\delta^*(p, w)$, the state that D winds up in if it starts in p and sees w. Then we make each state p final in D' if and only if $f(p)$ is a final state of D. This way any string x is accepted by D' if and only if xw is accepted by D.

There were 24% ten-point answers for this. This is a case where you should use your freedom to look at a regular language which whatever model works best for you. It was hard to prove this from a regular expression, but much easier from a DFA. Here's a valid but somewhat silly proof from regular expressions. Using an induction on all regular expressions, you can construct and verify a function that takes a regular expression R and gives back a regular expression whose language is $L(R)/a$, where a is any letter. If you prove that, you can get a regular expression for $L(R)/w$ for any string w, by induction.

Question 4 (10): Prove that is L is a context-free language, and w is any fixed string, then the language L/w is context-free. (**Hint:** It may be easier to solve this by generalizing. If you assume that if L is a context-free language and that B is regular, and you prove L/B is context-free, this suffices – why?)

Following the hint, let X be a PDA for L and let Y be a DFA for B. We'll assume that X is a PDA made by a top-down parser from a grammar, so that it can only accept by removing a special symbol from its stack. We make a new PDA Z with state set that is two copies of $X \times Y$. The computation proceeds in two phases. In the first phase, Z processes all of its input in X, having Y stay in its start state. In the second phase, Z nondeterministically chooses zero or more letters from the input alphabet, processes these letters in X but reading ϵ instead of reading the letter from the input, and processes the "imagined letters" in Y. Z accepts if it accepts in both machines at the end of the second phase, because X has processed a string uv, where u is the actual input it read for Z and v is the imagined string, and Y has processed v. So $uv \in L$ and $v \in B$.

The reason that V/w is a special case of V/B is that we can choose B to be the regular language $\{w\}$.

I gave one *ten-point answer on this one, and that was generous. The key idea that pretty much everyone missed was to run the PDA for L and the DFA for B in parallel, altering it so that you guess when x has ended and w has begun, and start the PDA only then. Several people wanted to use the stack to store w to check it later for membership in B, but they were doing this at the same time that they were using the stack on the PDA to determine whether xw was in L.*

Question 5 (10): A Turing machine M, with one tape, is defined to **cycle** on any input w if there exists a configuration C such that if started in w, M reaches C *more than once*. Prove that if L is a TR language, then there exists a Turing machine M such that $L(M) = L$ and M does not cycle on any input w.

Since we know L is TR, we know that some one-tape machine M' exists with $L(M') = L$. Modify M so that before each step of M', it copies the entire tape (up to the end of the used portion of the tape) one space to the right, leaving some marker character in the vacated space. It will accept or reject if and only if M does, but if M reaches the same configuration C at different times, M' will not be in the same configuration each time because each time it will be displaced to the right by different amounts.

There are two different valid approaches to solve this that I can see. The solution above modifies M so that it never repeats any configuration at all *– along with keeping a log of all the prior configurations, you could keep a counter on the tape that records the number of steps taken, or just shift the whole tape over one space for every step of M. The other approach is to use the log of prior configurations to detect when M might be about to cycle, stop it from doing so, and reject. Of course this won't change the language of the machine because that input wasn't going to accept.*

Question 6 (10): Consider the language $DOUBLE_{TM}$, the set of Turing machines $\langle M \rangle$ such that $L(M)$ contains a string of the form ww. Is $DOUBLE_{TM}$ a TR language? Is it a co-TR language? Prove your answers. Do not use the result of Question 7.

It is TR but not co-TR. We prove that $DOUBLE_{TM}$ is not co-TR by mapping $A_{TM} \leq_m DOUBLE_{TM}$. We map the pair $\langle M, w \rangle$ to a machine M' that rejects all non-empty strings and, if its input is ϵ, runs M on w. Thus if M accepts w, $L(M') = \{\epsilon\}$ and M' is in $DOUBLE_{TM}$, but if M does not accept w, $L(M') = \emptyset$ and M' is not in $DOUBLE_{TM}$.

To show that $DOUBLE_{TM}$ is TR, we build a machine X that accepts input $\langle M \rangle$ iff M is a machine that accepts any string of the form ww. X runs M in parallel on every string of this form, using dovetailing.

There were lots of reductions that were backward, and/or drawing the wrong conclusions about reductions being downward closed for TR or co-TR, and upward closed for non-TR and non-co-TR.

The reason we told you not to use the result of Q7 is that if you had a decider for $DOUBLE_{TM}$, you could use it to decide $DOUBLE_{CFL}$. We wanted to isolate the easier problem Q6.

Question 7 (10): Prove that the language $DOUBLE_{CFG}$, the set of context-free grammars $\langle G \rangle$ such that $L(G)$ contains a string of the form ww, is undecidable. (**Hint:** Make a reduction $PCP \leq_m DOUBLE_{CFG}$ as follows. Let the set of dominoes for the PCP problem be $\{[\frac{x_1}{y_1}], \ldots, [\frac{x_k}{y_k}]\}$ and let a_i be a unique terminal for each domino. Build the grammar G with a rule $S \to A'B'$ and, for each i with $1 \leq i \leq k$, rules $A' \to x_i A a_i$, $A \to x_i A a_i$, $A \to \epsilon$, $B' \to y_i B a_i$, $B \to y_i B a_i$, and $B \to \epsilon$. Prove that the PCP instance has a match if and only if $L(G)$ contains a string of the form ww.

A string generated by G must be of the form uv, where u was generated by A and v by B. A generates any string of the form $x_1 \ldots x_n a_n \ldots a_1$, with $n \leq 1$, and B generates any string of the form $y_1 \ldots y_n a_n \ldots a_1$. The only way the string uv could be of the form ww is if u and v are the same string, and this can happen if and only if there is a match in the PCP instance. If there is any match, there will be a such a string uv. It's clear that there is a computable function to build G from the PCP instance. Since the language PCP is undecidable, and we have proved $PCP \leq DOUBLE_{CFG}$, we know that $DOUBLE_{CFG}$ is undecidable.

27% of you got ten-point answers. A lot of the wrong ones got confused about the strings generated by G. It's important that you have the string of a's on each side, so that there could be a match only if the two sides use the tops and bottoms of the same dominoes.

Question 8 (10 extra credit): In this problem, we will prove that the language REG_{CFG}, the set of context-free grammars G (with $\Sigma = \{0, 1\}$) such that $L(G)$ is regular, is undecidable. We will build a reduction $ALL_{CFG} \leq_m REG_{CFG}$, as follows. Throughout the problem, Σ denotes $\{0, 1\}$. Let Z be the language $\{0^n 1^n : n \geq 0\}$, which we know not to be regular. Given a grammar G, with alphabet Σ, we define a grammar G' (over the alphabet $\{0, 1, \#\}$) such that $L(G') = Z\#\Sigma^* \cup \Sigma^*\#L(G)$.

1. Describe how to build the grammar G' from the grammar G.
   $S_0 \to Z\#A$, $S_0 \to A\#S$, $Z \to 0Z1$, $Z \to \epsilon$, $A \to 0A$, $A \to 1A$, $A \to \epsilon$, **and all the other rules of G.**
   This was a very straightforward question, even though it was part of an XC question. 24% of you gave no answer for Q8.
2. Prove that if $\langle G \rangle \in ALL_{CFG}$, then $L(G')$ is a regular language.
   Since every string in $\{0, 1\}^*$ is in $L(G)$, and every string in $L(G')$ must have exactly one #, $L(G') = \Sigma^*\#\Sigma^*$, which is clearly regular.
   You needed to refer to $\Sigma^*\#\Sigma^*$ to get full credit.
3. Suppose that some string $w \in \Sigma^*$ is not in $L(G)$. Prove that the quotient language $L(G')/\#w$ is not regular.
   The quotient language is the set of all strings x such that $x\#w$ is in $L(G')$. Such a string cannot be generated from the rule $S_0 \to A\#S$, so it can only come from $S_0 \to Z\#A$. This means that the string x must be a member of the language Z (and every string of Z will be there) and $L(G')/\#w$ is equal to the non-regular language Z.
   I was generous if you said "$L(G')/\#w$ comes from the non-regular language Z", rather than showing that it's equal to Z, even if you didn't quote the right closure properties.
4. Using the result of Question 3, whether you solved it or not, explain how we may now conclude that REG_{CFG} is undecidable.
   If a grammar is not in ALL_{CFG}, there must exist a string w with $w \notin L(G)$. But if $L(G')$ were a regular language, by the result of Question 3, $L(G')/\#w$

would be a regular language, and it isn't. This completes the proof that $ALL_{CFG} \leq_m REG_{CFG}$, Since we proved in lecture that ALL_{CFG} is undecidable, we know that REG_{CFG} is also undecidable.

There were 15% of you that got full credit for Q8.