NAME: _____

SPIRE ID: _____

COMPSCI 501
Formal Language Theory
Solutions to Midterm Spring 2026

D. A. M. Barrington 1 April 2026

DIRECTIONS:

- Answer the problems on the exam pages.
- There are seven problems on pages 2-9, some with multiple parts, for 100 total points plus 5 extra credit. The scale is A = 80, B = 64, C = 48, D = 32, F = 16, norm = (raw - 16) * (400/64).
- The supplemental page 10 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) Every infinite language has a neutral letter. (See the definition in Question 3 or on the supplemental sheet.)

  FALSE. (74% correct) This is completely false – the simplest example is the set of all nonempty strings, which is certainly infinite, but fails the neutral letter test for every letter a, since a is in the language and ε is not.

- (b, 2) A **linear grammar** is a context-free grammar such that every rule has at most one non-terminal in its right-hand side. Given this definition, the language of every linear grammar is a regular language.

  FALSE (62% correct). $S \to aSb|\varepsilon$ is a linear grammar, but its language is our canonical non-regular language.

- (c, 2) If G is any context-free grammar, there exists a natural number p such that for any string $w \in L(G)$, there exists strings u, v, x, y, and z such that $|vxy| \leq p$, $|vy| > 0$, and for all natural numbers i, $uv^ixy^iz \in L(G)$.

  FALSE (38% correct). This is close to the CFLPL, but it leaves out the crucial condition that $|w| \geq p$. No finite nonempty language, for example, can possibly meet this condition, and all such languages are context-free.

- (d, 2) If L is a context-free language and R is a regular language, the symmetric difference $L\Delta R$ may fail to be a context-free language.

  TRUE (64% correct). If $R = \Sigma^*$, then $L\Delta R$ is just $\overline{L}$, and we have seen examples of context-free languages whose complements are not context-free.

- (e, 2) Let X_1, X_2, and X_3 be any three TR languages. Let Y be the language of strings that are in *exactly two* of these three languages. Then Y must also be TR.

  FALSE (55% correct). If it were "at least two", this would work, but the condition requires a string to *not* be in a particular TR language, which is in general not a TR condition. If X_1 and X_2 were each Σ^*, for example, Y would be the complement of X_3, so that if X_3 were A_{TM}, Y would not be TR.

- (f, 2) Let M be a Turing machine taking two arguments which we will view as natural numbers. Then we know that the language $\{j : \exists i : M \text{ accepts } (i, j)\}$ is TR because we can build a single Turing machine that takes its input and sequentially runs M on each input (i, j) and accepts if any of these computations accept.

  FALSE (43% correct). The result is true, but the argument is invalid because of the word "sequentially". If M runs forever on input $(0, j)$, this machine would never test $(1, j)$, or indeed any other input. If we ran those computations in parallel by dovetailing, this would be fine.

- (g, 2) The language $\{(M, w, n) : M \text{ accepts } w \text{ within } n \text{ steps }\}$ is undecidable.

  FALSE (83% correct). We can decide it by running M on w for n steps and see whether it accepts.

- (h, 2) The following is a valid proof of Rice's Theorem using the Recursion Theorem. Let P be a property of Turing machine descriptions such that if $L(M_1) = L(M_2)$ then $P(M_1) \leftrightarrow P(M_2)$, there exists a machine Y with $P(Y)$ true, and there exists a machine N with $P(N)$ false. Then if we could decide P, we could build a machine R that on input w, obtains its description, decides whether $P(R)$ is true, runs N on w if $P(R)$ is true, and runs Y on w if $P(R)$ is false. This is a contradiction and thus P cannot be decidable.

  TRUE (57% correct). This is exactly the argument of Problem 6.9 in Sipser.

- (i, 2) Because the language ALL_{TM} satisfies the rule $(M) \in ALL_{TM}$ if and only if $\forall w : w \in L(M)$, this language is in the Π_1 level of the Arithmetic Hierarchy.

  FALSE (55% correct). This is false because we proved in lecture that ALL_{TM} is complete for the Π_2 level, and thus is not in Π. The alleged Π description of this language has an undecidable predicate, "$w \in L(M)$", but to determine its level in the AH we would need to use only decidable predicates inside the quantifiers.

- (j, 2) If M is a Turing machine and w is a binary string, we say that a string x is **quadratic-time described** by (M, w) if M, on input w, halts with x on its tape in at most $|w|^2$ steps. The **quadratic-time Kolmogorov complexity** $QK(x)$ of x is the length of the shortest pair (M, w) that quadratic-time describes x. Then the function from x to $QK(x)$ is not a Turing-computable function.

  FALSE (50% correct). We can compute it, though it would take a while, by testing each pair in turn, in order of total length, until we find one that halts with x on its tape, Each test only takes a finite time, so we can do them all sequentially. Since a "hello world" program for a string w takes less than $|w|^2$ time, the search will always terminate.

  The mean grade was 11.62/20, and the highest grade was 16/20.

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) Consider the set $REPEAT$ of strings that ever repeat a five-letter word, that is, the set of strings w that can be written as $xuyuz$, where $|u| = 5$ and x, y, and z are arbitrary strings. Then $REPEAT$ is not a regular language.

  FALSE (17% full credit, mean score 3.14, 57% correct boolean). For any fixed string u, we make a regular expression $\Sigma^* u \Sigma^* u \Sigma^*$ containing all the strings that repeat the string u. Our language is the union of finitely many ($|\Sigma|^5$ of them) such languages, and we know that the regular languages are closed under (finite) union. *We could also make an NFA with a branch for each u, and on that branch it idles in one state until it guesses that u is starting, reads u in the next five steps, idles at a new state, guesses when to read the second occurrence of u, and then goes to a final state where it idles for the rest of the input.*

  You could make a DFA as well – it would need to remember which five-letter string it has ever seen, which would require $2^{|\Sigma|^5}$ states, a lot but still only finitely many.

- (b, 6) Consider the following language $COINCHECK$, consisting of strings over the alphabet $\{\#, N, D, Q\}$ of the form $u\#v$, where u and v are each strings without $\#$, and the numerical values of u and v are the same, where each N counts five cents, each D counts ten cents, and each Q counts 25 cents. Then $COINCHECK$ is not a context-free language.

  FALSE (40% full credit, mean score 4.00, 71% correct boolean). We can design a (deterministic) PDA that accepts a string if and only if it is in $COINCHECK$. We put a start symbol \$ on the stack, and read coins (N, D, or Q) until we run out, putting one, two, or five copies of the letter a onto the stack. We then read a $\#$, moving to a new state, in which we now read more coins, this time popping one, two, or five a's for each coins. Finally, if we can pop the \$ with the input done, we accept.

  This was an easy one, as long as you thought in terms of a PDA and not a grammar.

- (c, 6) Define the language $THISSIZE_{CFG}$ to be the set of pairs (G, n) such that G is a grammar, n is a natural, and there exists some string of length n in $L(G)$. Then the language $THISSIZE_{CFG}$ is not Turing decidable.

  FALSE (40% full credit, mean score 4.21, 76% correct boolean). Given G and n, there are finitely many strings of length n. A Turing machine can test each one for membership in $L(G)$, and accept if any of them are found to be there.

  Also pretty easy, as long as you realize that A_{CFG} is decidable.

- (d, 6) Let $DECIDER$ be the set of Turing machines that are deciders, that is $\{(M) : \forall w :$ M accepts w or M rejects $w\}$. Then the languages $DECIDER$ and ALL_{TM} are m-equivalent (meaning that each is $\leq_m$-reducible to the other).

  TRUE (12% full credit, mean score 3.36, 64% correct boolean). To show $DECIDER \leq_m ALL_{TM}$, given any machine M, make $f(M)$ by changing every reject state to another accept state. Then $f(M)$ accepts a string w if and only M either accepts or rejects w. To show $ALL_{TM} \leq DECIDER$, given any maching M, make $g(M)$ by replacing each reject state by a state that always moves right and stays in the same state, whatever letter it sees. So $g(M)$ cannot reject at all, and thus it "either accepts or rejects" if and only if M accepts.

  This was an exercise in formulating and justifying $\leq_m$ reductions. Some of you need to work on that before the final!

- (e, 6) An **insert-delete TM** or **IDTM** is like a Turing machine, except that on each step, rather than overwriting the character on the current cell, either (1) inserts a new tape cell (with content given by the transition function), (2) deletes the current cell, or (3) moves left or right without altering the tape. Then the language A_{IDTM} is not Turing decidable.

  TRUE (31% full credit, mean score 4.67, 95% correct boolean). We can prove $A_{TM} \leq_m A_{IDTM}$ by simulating an ordinary one-tape TM with an IDTM as follows. Whenever the ordinary TM M carries out a step, if it needs to change the current tape cell contents, it deletes the current cell, inserts a new cell with the desired content, then moves to the correct position in the new tape.

  I think some people were misinterpreting the new model. In order to get full credit, you need to talk about how the new model can simulate an arbitrary TM in order to show the give reduction. Some people got the reduction backward.

Question 3 (10): Recall that a formal language X has a **neutral letter** if there is a letter a such that for any strings u and v, $(uv \in X) \leftrightarrow (uav \in X)$, and $NEUT_{DFA}$ is the set of DFAs M such that $L(M)$ has a neutral letter. In this problem, you must prove that the language $NEUT_{DFA}$ is TD. In particular, first prove that $L(M)$ has a neutral letter if and only if M's minimal DFA M' has the property that its transition function δ satisfies $\exists a : \forall q : \delta(q, a) = q$. Then describe a decision procedure for $NEUT_{DFA}$ that could be implemented on a Turing machine that always halts.

(Mean score 6.05/10, 14% full credit.) Let M be any DFA and δ be the transition function of M's minimal DFA M'.

If a is a neutral letter for $L(M)$, then let q be any state of M', and let w be any string such that $\delta^*(q_0, w) = q$. The definition of a neutral letter tells us that w and wa are Myhill-Nerode equivalent for $L(M)$. The definition of the minimal DFA thus tells us that $\delta^*(q_0, wa)$ is also q, which implies that $\delta(q, a) = q$.

Conversely, suppose that every letter a fails to be a neutral letter, so that there exists two strings u and v such that $(uv \in X) \oplus (uav \in X)$. This means that u and ua are *not* Myhill-Nerode equivalent, and thus that $\delta^*(q_0, u) \neq \delta * (q_0, ua)$. Letting $q = \delta^*(q_o, u)$, we see that $\delta(q, a) \neq q$).

I gave six points for the equivalence and four for the decision procedure. The proof of the equivalence had to use the similarities between the definitions of the neutral letter and the minimal automaton. For the decision procedure, you needed to explain how you can test the property given an arbitrary DFA, which means to first minimize it and check for each of the loops.

Question 4 (10): Let Σ and Δ be any two finite alphabets. A **homomorphism** from Σ^* to Δ^* is a function defined by letting $f(a)$, for any letter a in Σ, be any string in Δ^*, and defining $f(w)$ so that $f(\varepsilon) = \varepsilon$ and $f(wa) = f(w)f(a)$ for any string w and any letter a.

- (a) Prove that if f is any homomorphism from Σ^* to Δ^*, and $X \subseteq \Sigma^*$ is any regular language, the language $f(X) = \{f(w) : w \in X\}$ is a regular language over the alphabet Δ.

  (Mean score 2.62/5, 19% full credit) Let R be a regular expression for X. We define a new regular expression $f(X)$ over the alphabet Δ, as follows. If $a \in \Sigma$, we let $f(a)$ be the expression for the string $f(a)$. We translate each of the regular expression operations as $f(R \cup S) = f(R) \cup f(S)$, $f(RS) = f(R)f(S)$, and $f(R^*) = f(R)^*$. By induction over all regular expressions over Σ, it is clear that for any such R, the expression $f(R)$ defined here has a language that is the result of applying the homomorphism f to the language $L(R)$.

  Lots of people started with X being given as a DFA, which is much harder to work with. To determine whether $w \in f(X)$, you can guess that w is a concatenation of strings of the form $f(a)$, such that the a's involved accept when fed into X's DFA. This will work, but it's important that you are creating an NFA (or a GNFA, which at least one person successfully used) instead of a DFA.

- (b) With f as above, prove that if Y is any regular language over the alphabet Δ, the language $f^{-1}(Y) = \{w : f(w) \in Y\}$ is a regular language over the alphabet Σ.

  (Mean score 2.26/5, 10% full credit) Let M be a DFA for the language Y. Create a new DFA M' with input alphabet Σ and the same start and final states as M. For each letter $a \in \Sigma$, and each state q of M', define $\delta(q, a)$ in M' to be $\delta^*(q, f(a))$ in M. By induction on strings $w \in \Sigma^*$, $\delta^*(q, w)$ in M' equals $\delta^*(q, f(w))$ in M. It follows that M' accepts w if and only if M accepts $f(w)$, and so M' is a DFA whose language is $f^{-1}(Y)$ and thus $f^{-1}(Y)$ is regular.

  There were a lot of problems with the definition, many assuming that the string homomorphism f had to have an inverse string homomorphism, which it needn't. This is the direction for which it makes the most sense to start with a DFA.

Question 5 (10): A **homomorphism** from strings to strings is defined in Question 4 above.

- (a) Prove that if X is any context-free language over the alphabet Σ, and f is any homomorphism from Σ^* to Δ^*, then the language $f(X) = \{f(w) : w \in X\}$ is also context-free.

  (Mean score 3.36/5, 36% full credit) Let G be a grammar for X. Make a new grammar H by first replacing each terminal a of G with a new non-terminal A, then for each $a \in \Sigma$, add a rule $A \to f(a)$. In any derivation in H, you can alter it to apply the new rules at the end, and in this way there is a string in $L(G)$ that is generated, and then acted on by f to make the result. And this method allows you to generate any string in $f(X)$ using H.

  This went better, though some people took the harder route of starting from a PDA and (hopefully) used the nondeterminism of the new PDA.

- (b) Prove that if Y is any context-free language over the alphabet Δ, and f is any homomorphism from Σ^* to Δ^*, then the language $f^{-1}(Y) = \{w : f(w) \in Y\}$ is also context-free.

  (Mean score 2.19/5, 10% full credit) Take a PDA M with $L(M) = Y$, and make a new PDA M' whose input alphabet is Σ as follows. M' starts with the same states of M. For every transition of M reading the letter $a \in \Sigma$, where $f(a) = w$ with $|w| = k$, we include all sequences of k transitions in M' such that the i'th transition in the sequence reads the i'th letter of w, and these new transitions may only be performed as part of the the sequence (we add new states to enforce this). So an accepting computation of M' must read a string z such that M would have an accepting computation that reads $f(z)$. So $z \in L(M') \leftrightarrow f(z) \in L(M)$, and thus $L(M') = f^{-1}(L(M))$, making the language $f^{-1}(M)$ context-free.

  Again, a lot of people invented a string homomorphism f^{-1} that need not exist. The common answer "run the PDA for Y on the string $f(w)$" is more or less right, but required some explanation of how you would do that.

Question 6 (10): In Question 2(e) we defined an **insert-delete TM** which, *instead of* rewriting an existing tape letter, can insert a new tape cell with given content, delete a tape cell, or move one space on the tape. In this problem we use an **insert TM** or **ITM**, which can insert cells and move, but not delete cells. Prove that A_{ITM} is not Turing decidable. You may refer to any arguments you have already made in Question 2(e) without repeating them.

(Mean score 3.83/10, 10% full credit) We need to simulate an ordinary, one-tape TM with an ITM, proving that $A_{TM} \leq_m A_{ITM}$. We have argued in Question 2(e) that the arbitrary TM can be simulated by an IDTM which is also allowed to delete letters. The way that we will "delete" letters with our ITM is to insert a new tape character d in front of the ordinary character to be deleted. At any time, the tape of our ITM will contain both d's and ordinary characters, never having two d's in a row. This tape will be representing the corresponding tape of the ordinary TM, which consists of the sequence of undeleted ordinary characters (where "undeleted" means that the character is not preceded by a d. We make sure that after simulating each step of the ordinary TM, the ITM is looking at an undeleted ordinary character. To implement the appropriate action of the ordinary TM: (1) if the ordinary TM does not change the letter it sees, the ITM just moves left or right to the next undeleted ordinary character, and (2) if the ordinary replaces the letter a that it sees, the ITM inserts a d before that letter, then inserts the new character b either before or after the string da.

This was much harder than Question 2(e), even though we told you that A_{ITM} was undecidable. I gave a fair number of points for setting up a correct reduction, but you needed to explain how your ITM, which can never delete cells, is going to simulate the tape of the ordinary TM. A common answer was to write down each new configuration in new tape space on each time step, which is a good idea, but overlooks the fact that the way we copied strings used rewriting of cells, usually by expanding the tape alphabet. I don't see a way to make this work without locally marking cells as having been deleted.

Question 7 (10+5): In Question 3 (and on the supplemental sheet) we defined what it means for a formal language to have a **neutral letter**. In this problem we consider the language $NEUT_{CFG}$, the set of all grammars whose languages have a neutral letter. But for the regular portion of this problem, we ask you to prove a related language to be undecidable. In particular, we define $NEUT^*_{CFG}$ to be the set of pairs (G, a) where a is an input letter of the grammar G and a is a neutral letter for $L(G)$.

It is possible to solve both of these problems *without* going into the proof that ALL_{CFG} is undecidable, but it may be useful to consider how it works. In the homework, we constructed a reduction from ALL_{CFG} to $COFINITE_{CFG}$, building a new grammar G' so that $L(G')$ lacked an infinite number of strings if $L(G)$ lacked even one. What happens to neutral letters in the new grammar's language?

- (a, 10) Prove that $ALL_{CFG} \leq_m NEUT^*_{CFG}$.
  (Mean score 4.17/10, 7% full credit) Given a grammar G over Σ, we need to create a new grammar H and a letter a, so that H will be over the alphabet $\Delta = \Sigma \cup \{a\}$, such that G accepts every string over Σ if and only if $L(H)$ has a as a neutral letter. We begin by adding all the rules of G to H, and then adding new rules to general any string in $\Delta^* a \Delta *$, for example, with rules $S \to E$, $E \to xE|Ex|a$, for every letter $x \in \Delta$. If every string over Σ is in $L(G)$, then every string over Δ is in $L(H)$, and $L(H)$ thus has a neutral letter. If there exists a string $w \in \Sigma^*$ that is not in $L(G)$, it is also not in $L(H)$ because using a new rule must generate at least one a. But then since $w \notin L(H)$ and $wa \in L(H)$, a is not a neutral letter for $L(H)$.
  The most disappointing result here was how few people were able to set up a correct reduction, when you are asked explicitly for a particular reduction. You needed to note the data types of both languages, so that your reduction is converting a grammar G into a pair (G', a). At that point, the similarity to the HW problem started to show for some of you. But it was hard for many of you to realize that all you may do is to define a mapping of grammars to grammars. (Things like "if $G \in ALL_{CFG}$, we should..." don't make sense because your reduction cannot tell whether G is in or not in that undecidable language.) A common answer was to allow the new grammar to form any strings made by inserting neutral letters into any string in $L(G)$. But this doesn't work because the result will have a neutral letter, whether or not there are strings not in $L(G)$.

- (b, 5XC) Prove that $NEUT_{CFG}$ is not TD.
  (Mean score 0.88, 2% full credit) There may be a better way to do this, but the construction above does not suffice to prove $ALL_{CFG} \leq_m NEUT_{CFG}$, because there may be some other neutral letter in $L(G)$ even if $L(G) \neq \Sigma^*$. So I (Dave) would do this by adapting the proof in lecture that $\overline{A_{TM}} \leq_m ALL_{CFG}$. In that proof, we take an input (M, w) and create a grammar $G_{M,w}$ that accepts all strings that are not accepting computations of M on w, using the format where every other configuration in the history is written backward. We can use the same reduction in this case to prove $\overline{A_{TM}} \leq NEUT_{CFG}$!. If w does not accept M, then $L(G_{M,w}) = \Sigma^*$, and this language has a neutral letter because every letter is neutral for Σ^*. But if M does accept w, there exists a string c that is not in $L(G_{M,w})$, and it is the *only* such string. So adding or deleting any letter of c results in a string that is in $L(G_{M,w})$, and the neutral letter property fails.

Only 62% of you attempted this at all, and only 16% got more than one point. Actually, the reduction from the HW problem in Joshua's solutions can be adapted to work in Question 7(b) without getting into the details of the ALL_{CFG} proof.