NAME: _____

SPIRE ID: _____

COMPSCI 250
Introduction to Computation
Solutions to Final Exam Fall 2023

D. A. M. Barrington and G. Parvini 12 December 2023

DIRECTIONS:

- Answer the problems on the exam pages.
- There are four problems on pages 2-9, some with multiple parts, for 125 total points plus 5 extra credit. Final scale will be determined after the exam.
- Page 10 contains useful definitions and is given to you separately – do not put answers on it!
- If you need extra space use the back of a page –both sides are scanned.
- If absolutely necessary, you may attach additional pages to be graded, but this is a big nuisance to us.
- No books, notes, calculators, or collaboration.
- Scrap paper during the exam may be used, if it starts out blank. Please transfer any answers to the exam pages.
- You may say "pass" for any question or lettered part of a question, except for the true/false and the extra credit questions, and receive 20% of the points.

1	/35
2	/10
3	/20
4	/40+5
5	/20
Total	/125+5

Question 1 (35): (Dog Proof) On Thanksgiving at Dave's house, his dogs Blaze and Rhonda were joined by his daughter's dog Gwen. The family observed the sounds made by the three dogs – barking, growling, and whining.

Given the set of dogs $D = \{B, G, R\}$, the set of sounds $S = \{ba, gr, wh\}$, and the set of numbers $N = \{0, 1, 2\}$, we define a function f from $D \times S$ to N such that if x is in D and y is in S, $f(x, y) = 0$ if dog x *never* makes sound y, $f(x, y) = 1$ if x *rarely* makes sound y, and $f(x, y) = 2$ if x *often* makes sound y.

We also define three propositions p as "$f(B, ba) = 2$", q as "$f(G, ba) = 2$", and r as "$f(R, ba) = 2$".

Question 1a (10): Translate the following five statements as indicated:

Statement I (to symbols): Either Blaze or Gwen, but not both of them, barks often.

$f(B, ba) = 2 \oplus f(G, ba) = 2$

Statement II (to English): $((f(R, ba) = 2) \vee (f(G, ba) = 2)) \wedge (f(B, ba) = 2)$

Either Rhonda or Gwen barks often, and Blaze barks often.

Statement III (to symbols): There is no dog that never barks.

$\neg\exists D : f(D, ba) = 2$

Statement IV (to English): $\forall n : (\exists d : f(d, gr) = n) \wedge (\exists s : f(R, s) = n) \wedge (\exists t : f(G, t) = n)$

For any number n in N, there is a dog whose growling frequency is n, there is a sound such that Rhonda's frequency for that sound is n, and there is a sound such that Gwen's frequency for that sound is also n.

Statement V (to symbols): Given any two sounds, Blaze has the same frequency for each.

$\forall s : \forall t : f(B, s) = f(B, t)$.

Question 1b (10): Using only Statement I and Statement II, determine the truth value of the three propositions p, q, and r. You may use either a truth table or deductive rules. If you do the latter, remember that for full credit you must verify that your solution satisfies both the rules.

The two statements translate to "$p \oplus q$" and "$(r \vee q) \wedge p$". From Right Separation on II, we get p. From p and the Definition of XOR, we get "$\neg q$". By Left Separation on II we get $r \vee q$, which we can rewrite by Definition of Implication as "$\neg q \rightarrow r$", and from this and $\neg q$ we get r by Modus Ponens. So p and r are true and q is false.

Question 1c (15): Using all of Statements I-V, determine the nine values of the function f. That is, for each pair (x, y) in the domain $D \times S$, find $f(x, y)$.

From 1b, we have $f(B, ba) = f(R, b) = 2$, and $f(G, ba) \neq 2$. By Specification on V twice, we get $f(B, gr) = f(B, wh) = 2$. We can rewrite III as $\forall x : f(x, ba) \neq 0$ and Specify this to get $f(B, ba) \neq 0$, which then gives us $f(B, ba) = 1$. Statement IV tells us that all three values occur for $f(\_, gr)$, for $f(R, \_)$, and $f(G, \_)$. So the values of $f(G, g)$ and $f(G, w)$ must be 0 and 2, and $f(G, gr) = 2$ is not possible because $f(B, gr) = 2$, so $f(G, gr) = 0$ and $f(R, gr) = 2$. Finally, we know that the values $f(R, gr)$ and $f(R, wh)$ are 1 and 0. $f(R, gr) = 0$ is impossible because $f(G, gr) = 0$, so $f(R, gr) = 1$ and $f(R, wh) = 0$.

Question 2 (10): (Induction 1) Prove, by induction, *either* one of the following two statements:

"There exists a natural m such that for any natural n, $(n \geq m) \rightarrow (n! < 3^n)$."

"There exists a natural m such that for any natural n, $(n \geq m) \rightarrow (3^n < n!)$."

(We could offer you extra credit for proving both, but we won't.)

(**Hint:** This can be done with ordinary induction, for an appropriate base case.)

The second statement is true. The given number m is 7, or any larger natural. For the base case, $3^6 = 729$ and $6! = 720$, so $3^n > n!$ is true for $n = 6$, but $3^7 = 2187$ and $7! = 5040$, so $3^n < n!$ for $n = 7$. The inductive hypothesis $P(n)$ is "$3^n < n!$", and the inductive goal $P(n + 1)$ is "$3^{n+1} < (n + 1)!$". We can reformulate the inductive goal as $3 \cdot 3^n < (n + 1)n!$, and since $n+1 > 3$, we know that $3 \cdot 3^n$ is less than $3 \cdot n!$, which in turn is less than $(n+1)n!$.

Note that the arithmetic in this solution was corrected on 30 April 2024.

We gave nothing better than 4/10 for attempting the wrong statement, because (a) you should know that factorials grow faster than exponentials from a data structures course, (b) we had similar problems on the homework, and (c) in order to finish this proof you have to accept some absurdity like any arbitrary natural n being less than 3. It might be reasonable to start the wrong proof, but you should see that it won't work and switch to the other one. When we first posted the exam, we had 2/10, the same as a pass, for proving the wrong question, but we relented somewhat.)

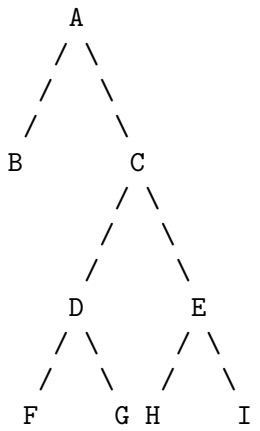
Question 3 (20): (Induction 2) Recall that a **rooted binary tree (RBT)** consists of either

- a single node, which is its root, or
- a new node, its root, which is connected to the roots of two other RBT's.

An **edge 3-coloring** is an assignment of colors (red, blue, or green) to the *edges* of an RBT. A coloring is **valid** if there is no node that is adjacent to two edges of the same color.

Question 3a (5):

Give a valid edge 3-coloring of the following RBT:



A greedy algorithm works: AB red, AC blue, CD red, CE green, DF blue, DG green, EG red, EI blue.

Question 3b (15):

Prove that given any RBT T, we can make a valid edge 3-coloring for T.

(**Hint:** Prove this by induction on the definition of RBT's, but let your $P(T)$ be "for any color c, we can make a valid edge 3-coloring of T such that the root has no adjacent edge of color c".)

Taking the hint, let $P(T)$ be the statement that for any color c, there is a valid edge 3-coloring where the root avoids c. For the base case, T has one node and no edges. It is a valid edge 3-coloring since every edge is colored, and we certainly avoid any of the three colors.

For the inductive step, we have a tree T formed from other RBT's U and V, where the IH tells us that $P(U)$ and $P(V)$ are true. Without loss of generality, let c be "red", so that we must color T so that its root A has no adjacent red edge. Let B and C be the roots of U and V respectively. Color AB blue and AC green. Using the IH, construct a valid edge 3-coloring of U such that C has no adjacent blue edge. Similarly, construct a valid coloring of V such that C has no adjacent green edge. Together with these two colorings and the coloring of the last two edges, this is a valid edge 3-coloring of T, and A has no adjacent red edge.

Many people approached the problem via an alternate definition of RBT's, or a variant of them, where the base case is a single node and the inductive case involves taking a leaf and making it an internal node by adding two edges and two new leaves below it. No one, of course, proved that the class of trees defined

in this way is the same as the class of RBT's, though this is actually true. I gave a lot of points for these approaches, but not full credit.

It's true that given any binary rooted tree, even if it has internal nodes with only one child, can be 3-edge-colored by a greedy algorithm. Of course, to get full credit, you would need to prove that this always works, which really should be some kind of induction on trees.

I was fine with letting $P(n)$ being "any RBT of depth at most n can be 3-edge-colored", so that in the inductive step you have a new root and two subtrees for which the IH holds. But the tricky part of the whole proof was to show that your arbitrary colorings of the two subtrees can be fitted together into a valid coloring of the new tree. One way to do this is via the hint, using the stronger IH to say that your subtree's root avoids the color used by the edge connected to it. The other is to point out that given any valid coloring, you can swap two of its colors and get a new valid coloring.

Question 4 (40+5): (Kleene Constructions) Let N be the following λ-NFA. The alphabet is $\Sigma = \{a, b\}$. The state set is $\{i, p, q, r\}$, the start state is i, and the final state set is $\{q, r\}$. The transitions are (i, a, p), (i, a, r), (p, b, p), (p, b, q), (p, b, r), (q, a, r), (q, λ, r), and (r, λ, p).

(a, 10) Apply the Killing λ-Moves Construction to find an ordinary NFA N' such that $L(N') = L(N)$. If you deviate from the construction given in lecture and the textbook, explain why your construction is still correct. The correct NFA has four a-moves and nine b-moves.

The λ-moves are transitive if we add one more, (q, λ, p). Since there is no λ-path from the start state to any final state, we do not change the final state set. The letter move (i, a, p) gives rise only to itself. The move (i, a, r) gives rise to itself and (i, a, p), which is redundant. The move (p, b, q) gives rise to nine *moves, any move to or from any state p, q, or r. This makes the other two b-moves in N redundant. Finally, (q, a, r) gives rise to itself and (q, a, p).*

(b, 10) Using the Subset Construction on N', create a DFA such that $L(D) = L(N')$.

The start state is $\{i\}$, which is non-final. The a-arrow goes to $\{p, r\}$ (which is final) and the b-arrow goes to the death state, which of course has both arrows to itself. The a-arrow from $\{p, r\}$ goes to the death state, and the b-arrow goes to $\{p, q, r\}$ (which is final). From $\{p, q, r\}$, the a-arrow goes to $\{p, r\}$ and the b-arrow goes to itself, so we are done with only four states.

(c, 5XC) Is your constructed DFA D minimal for its language? If so, argue that it is minimal, either using the Minimization Construction or by direct use of distinguishability. If not, find a DFA D' such that $L(D') = L(D)$ and D' is minimal, justifying your answer if you did not use the Minimization Algorithm.

It is minimal. If we run the Minimization Algorithm, we start with two non-final states $I = \{i\}$ and $D = \emptyset$, and two final states $PR = \{p, r\}$ and $PQR = \{p, q, r\}$. For the non-final states, I has behavior FN and D has behavior NN, so they must be split. For the final states, PR has behavior NF and PQR has behavior NN, so they must be split and our new partition has each state in its own class.

Similarly, we can separate I and D by appending a, and we can separate PR and PQR also by appending a. So any pair of states is $L(N)$-distinguishable, as either a separates them as above, or λ separates them if one is final the other is non-final.

(d, 10) Using the DFA D (or an equivalent D' if you found one), find a regular expression R such that $L(R) = L(D)$.

We don't need a new start state because there are no moves into I, but we need to add a new final state F, make PR and PQR non-final, and add transitions (PR, λ, F) and (PQR, λ, F).

We can delete the death state D with no other action, since it has no arrows into it. There are now two states to eliminate, and it's not immediately clear which is better to eliminate first.

If we eliminate PR first, we get transitions (I, ab, PQR), (I, a, F), and $(PQR, b + ab, PQR)$. Eliminating PQR then gives us a final expression of $(a + ab(b + ab)^*(a + \lambda))$.

If we eliminate PQR first, we get transitions (I, a, PR) (unchanged), (PR, bb^*a, PR), and $(PR, \lambda + bb^*, F)$, and eliminating PR gives us $a(bb^*a)^*(\lambda + bb^*)$. Some people correctly noted that $\lambda + bb^*$ is equivalent to b^*.

We got a lot of regular expressions with no explanation, and we generally dealt harshly with them. Partial credit came from applying a process and making clear what you were doing.

(e, 10) Compute a λ-NFA N'' such that $L(N'') = L(R)$. Use a construction starting from R rather than just using N. If you use a construction other than the one given in the lectures and the textbook, explain it and argue that it is correct.

Here is a solution for the first regular expression, which has only one use of Kleene star. State 1 is the start state, and state 8 is the final state. We have arrows $(1, a, 2)$, $(1, a, 7)$, $(2, b, 3)$, $(3, \lambda, 4)$, $(4, a, 5)$, $(4, b, 6)$, $(4.\lambda, 6)$, $(5, b, 6)$, $(6, \lambda, 4)$, $(6, \lambda, 7)$, $(7, a, 8)$, and $(7, \lambda, 8)$.

Question 5 (20): The following are ten true/false questions, with no explanation needed or wanted, no partial credit for wrong answers, and no penalty for guessing.

- (a) Let n be a positive natural, and let R be the equivalence relation of mod-n congruence on the *positive* naturals. Then R divides the positive naturals into exactly $n-1$ equivalence classes.
  FALSE. There are n sets. We have removed one element of the class $[0]$, but here are still lots of other elements in that class.

- (b) Let N be an NFA, and let N' be a new NFA that is identical to N except that every final state of N is non-final in N', and vice versa. Then $L(N)$ must be the complement of $L(N')$.
  FALSE. Take the example where i is the start state, and f is both the only final state and the only other state, and the only edge is (i, a, f). Then $L(N) = \{a\}$ but $L(N') = \{\lambda\}$.

- (c) Let Σ be any finite alphabet. Then the language $(\Sigma^0)^*$ is non-empty if and only if Σ itself is non-empty.
  FALSE. $\Sigma^0 = \{\lambda\}$ whether Σ is empty or not. So $(\Sigma^0)^*$ is also $\{\lambda\}$ whether Σ is empty or not.

- (d) Let x be any natural. Then if x is divisible by either 3 or 7, then it is also divisible by 21.
  FALSE. This would be true if x were divisible by both 3 and 7, but that's not what we said.

- (e) Let G be directed graph with positive edge labels, and let h be an admissible, consistent heuristic for some goal node g. Suppose we conduct two searches with s as start node and g as the goal node, one a uniform-cost search and the other an A^* search using h. Then both searches will return the same distance from s to g.
  TRUE. Both return the correct shortest distance, the only difference is that the A^* search might be faster.

- (f) Let M be an ordinary Turing machine with tape alphabet $\{a, b, \Box\}$, where $\Box$ is the blank symbol. Let p, q be states of M with $\delta(p, a) = (q, a, R)$ and $\delta(q, a) = (p, a, L)$. Then if we start M in configuration $\Box bpaab\Box$, M will never halt.
  TRUE. The first step goes to $\Box baqab\Box$, and the second step goes to the original configuration. So the machine alternates between these two configurations forever.

- (g) Let R and S be any two regular expressions. Then the expressions $(R + S^*)(R^* + S)$ and $(R^* + S)(R + S^*)$ may fail to have the same language.
  TRUE. An example: Let $R = \{a\}$ and $S = \{b\}$. Then baa is in the first language but not the second.

- (h) Let w be a binary string. Then the set of substrings of w is equal to the set of prefixes of suffixes of w.

  TRUE. The string x is a substring of w if and only if $x = uxv$ for some strings u and v. So if x is a substring of w, it is a prefix of ux, and ux is a suffix of w. On the other hand, if x is a prefix of a suffix of w, we can write the suffix as z, where $yz = w$, and if x is a prefix of z, we can write $z = xq$, and then we have $w = yxq$ and thus x is a substring of w.

- (i) Let M be a five-tape Turing machine. Then there exists an ordinary single-tape Turing machine M' such that $L(M') = L(M)$.

  TRUE. This was proved in Lecture 38 and in the text.

- (j) Let A and B be two finite sets, and let f be a surjective (onto) function from A to B that is not a bijection. Then there exists an injective (one-to-one) function g from B to A that is not a bijection.

  TRUE. This function f can only exist if A is larger than B, and in that case this g will exist, because there must be an injection when the domain is smaller or the same size, and there can't be a bijection if the domain is strictly smaller.