All questions due Wednesday, August 12th at 11:59PM.

Note: Because this problem set is due on the last day of class, no late periods may be used and no late submissions will be accepted. Sorry about that! On the plus side, we’ll release solutions as soon as the problem set comes due.

This final problem set is designed as a capstone of the past seven weeks in this course. We’ll be concluding some long-running threads as well as exploring our newfound intuitions for the landscape of computability.

Before attempting the last problem on this problem set, we strongly recommend reading over the Guide to the Lava Diagram that is available on the course website, which provides a ton of extra background that you might find useful here.

As always, please feel free to drop by office hours or ask questions on Piazza if you have any questions. We’d be happy to help out.

Good luck, and have fun!
1 A Visit to the Fourth Dimension

This question explores a class of graphs called the hypercube graphs whose nodes and edges correspond to the vertices and edges of squares, cubes, tesseracts, and even more exotic higher-dimensional objects. Hypercube graphs have a ton of applications throughout CS theory and if you study high-performance computing, parallel programming, binary encoding schemes, or combinatorics, you’ll likely see them make an appearance.

We’ll begin this exploration by asking you to translate a statement into first-order logic.

i. Given the predicates
   \( x \in y \), which states that \( x \) is an element of \( y \), and
   \( x \notin y \), which states that \( x \) is not an element of \( y \),

   along with the constant symbols \( S \) and \( T \), which represent some two sets, translate the statement \( |S \Delta T| = 1 \) into first-order logic.

   Write your answer here.

   In zero dimensions, we have a point. In one dimension, we have a line. In two dimensions, we have a square. In three dimensions, we have a cube. And in four dimensions, we have a tesseract.

   How do we even think about what a tesseract is? For this, we can turn to graph theory. The hypercube graph of order \( k \), denoted \( Q_k \), is defined as follows:

   • The nodes of \( Q_k \) are the elements of \( \wp([k]) \). (Refer to Problem Set Three for the definition of \( [k] \).)
   • There is an edge between a pair of nodes \( S \) and \( T \) if (and only if) \( |S \Delta T| = 1 \).

   That definition might seem like a mouthful, but it makes a lot more sense once you have a visual intuition.

   ii. Draw the graphs \( Q_0 \), \( Q_1 \), \( Q_2 \), and \( Q_3 \). Label each node with the set it corresponds to. Then, explain why \( Q_0 \) is a good approximation of a point, \( Q_1 \) is a good approximation of a line, \( Q_2 \) is a good approximation of a square, and \( Q_3 \) is a good approximation of a cube.

      You may need to shuffle around the nodes of \( Q_3 \) to see why it’s a good proxy for a cube.

      Write your answer here.

   Though we won’t make you draw this one out, the tesseract is modeled by \( Q_4 \). There are many ways to draw \( Q_4 \); one of them is shown to the right. We’ve omitted the labels on the nodes for convenience.

   Isn’t that just wild? You took this dense mathematical definition that we gave you and extracted from that a new intuition for shapes in the fourth dimension and beyond! Imagine if we had given you this problem on the first day of class and think about how far you’ve come since then.
2 Equivalence Classes and Regular Languages, Part Two

On Problem Set Five, you explored the indistinguishability relation for \( L \), denoted \( \equiv_L \), defined as

\[
x \equiv_L y \quad \text{if} \quad \forall w \in \Sigma^*, (xw \in L \leftrightarrow yw \in L).
\]

You specifically proved that for any language \( L \), the relation \( \equiv_L \) is an equivalence relation and that any DFA for \( L \) must have at least \( I(\equiv_L) \) states. In this problem, you’re going to prove an amazing result:

**Theorem:** If \( L \) is a language where \( I(\equiv_L) \) is finite, then \( L \) is regular.

In other words, if you know absolutely nothing about a language other than there are finitely many equivalence classes of the \( \equiv_L \) relation, then somewhere out there, there must be a DFA for \( L \)!

Let \( L \) be an arbitrary language over some alphabet \( \Sigma \) where \( I(\equiv_L) \) is finite. We are going to prove that \( L \) is regular by defining a 5-tuple \( (Q, \Sigma, \delta, q_0, F) \) for this language \( L \). The key insight behind this proof is how to choose \( Q \). Specifically, we will choose \( Q \) to be the set of equivalence classes of \( \equiv_L \):

\[
Q = \{[w]_{\equiv_L} \mid w \in \Sigma^*\}.
\]

It might seem strange to have the states of a DFA be sets, but then again, you’ve seen something like this before. When we worked through the subset construction in lecture, we created a DFA whose states literally were sets of states of some particular NFA.

i. Explain why \( Q \) is finite. This should take you at most a sentence or two.

We now need to figure out how to pick a start state and wire up our transitions. Our goal will be to define \( q_0 \) and \( \delta \) so that our DFA has the following property: if you run \( w \) through this DFA, the state you end up in corresponds to \([w]_{\equiv_L}\). It turns out that choosing \( q_0 \) and \( \delta \) as follows makes this work:

\[
q_0 = [\varepsilon]_{\equiv_L}, \quad \delta([x]_{\equiv_L}, a) = [xa]_{\equiv_L}.
\]

Of course, you shouldn’t take our word for it. You should prove that these choices make everything work!

ii. Prove that for any string \( w \in \Sigma^* \), we have \( \delta^*(w) = [w]_{\equiv_L} \).

Need a refresher on the definition of \( \delta^* \)? Check Problem Set Five. Perhaps the proof strategy you used there to reason about \( \delta^* \) could come in handy here as well.

To seal the deal, we need to choose our set of accepting states. We’ll define \( F \) as follows:

\[
F = \{[w]_{\equiv_L} \mid \exists x \in [w]_{\equiv_L}, x \in L\}.
\]

In other words, \( F \) is the set of all equivalence classes containing at least one string in \( L \).

iii. On Problem Set Five, you saw that we can formally define \( L(D) = \{w \in \Sigma^* \mid \delta^*(w) \in F\} \). Prove that with this choice of \( F \), we have \( L(D) = L \).

There is a ton of formal notation here, but at the end of the day, this question is just asking you to prove that two sets are equal. Think way back to Problem Set One. What’s the easiest way to do this?

Your proof should use the formal definitions provided here and not higher-level concepts like “the DFA accepts \( w \)” or “run the DFA on \( w \)”.

Write your answer here.

By combining the two theorems you’ve explored about indistinguishability – the one you proved last time, and the one from above – we get this fundamental result:
**Theorem (Myhill-Nerode):** A language $L$ is regular if and only if $I(\equiv_L)$ is finite. Furthermore, if $I(\equiv_L)$ is finite, the smallest possible DFA for $L$ has exactly $I(\equiv_L)$ states.

This result formalizes the intuition we’ve had about regular languages corresponding to problems you can solve with only finite memory. The “memory” you need corresponds to remembering which equivalence class the string you’ve seen so far happens to fall into. If you talk to CS theory folk and mention “the Myhill-Nerode theorem,” they’ll assume you’re talking about the above theorem! The version we saw in lecture is just a special case of this more general one.
3  co-RE, Disjoint Unions, and Terrifyingly Difficult Problems

When we first saw that $A_{TM}$ is undecidable, we could at least take consolation in the fact that $A_{TM}$ is recognizable. Then we discovered that $L_D$ is unrecognizable, and up to now we haven’t had a comforting fact to fall back on. But there is something about $L_D$ we can console ourselves with: the complement of $L_D$, the language $\overline{L}_D$, is an RE language.* In other words, while there’s no general way to prove that some TM will not accept its own encoding, there is a general way to prove that some TM will accept its own encoding.

The fact that $L_D$ isn’t an RE language while $\overline{L}_D$ is still in RE suggests that there might be a bit more to the computability landscape than just R and RE. And indeed there is: the same way that $A_{TM}$ is the poster child of an RE language, the language $L_D$ is the poster child of a co-RE language. Formally speaking, the class co-RE consists of all the languages whose complement is an RE language:

$$\text{co-RE} = \{ L \mid \overline{L} \in \text{RE} \}.$$  

Intuitively speaking, the co-RE languages are languages where if you have a string that isn’t in the language, there’s some easy way to prove that it isn’t in the language.

i. Prove that $A_{TM} \notin \text{co-RE}$

Provide a proof here.

*While we’re not going to ask you to formally prove that $\overline{L}_D$ is an RE language, we would definitely recommend taking a few minutes to ponder how you’d build a recognizer or verifier for it.
At this point, we have an RE language that’s not in co-RE ($A_{TM}$) and a co-RE languages that’s not in RE ($L_D$). As a coda to our treatment of unsolvable problems, let’s see a language that’s neither in RE nor co-RE. In other words, there’s no general way to prove that strings in that language are indeed in that language, nor is there way to prove that strings not in that language are indeed not in that language. Yikes!

In what follows, let’s assume all languages are over the alphabet $\Sigma = \{0, 1\}$. Given two languages $A$ and $B$ over $\Sigma$, the disjoint union of $A$ and $B$, denoted $A \uplus B$, is the language

$$A \uplus B = \emptyset A \cup 1B$$

For example, if $A = \{1, 10, 100, 1000\}$ and $B = \{\epsilon, 0, 1, 00, 01, 10, 11\}$, then $A \uplus B$ is the language

$$A \uplus B = \{ 01, 010, 0100, 01000, 1, 10, 11, 100, 101, 110, 111 \}$$

Notice how each string in $A \uplus B$ is tagged with which language it originated in. Any string that starts with 0 came from $A$, and any string that starts with 1 came from $B$.

ii. Let $A$ and $B$ be languages where $A \uplus B \in$ RE. Show that $A \in$ RE. To do so, assume you have code for a verifier or recognizer for $A \uplus B$, use it to write code for a verifier or recognizer for $A$, then, in a sentence or two, explain why that code has the required properties. No formal proof is required.

Write your answer here.
iii. Let $A$ and $B$ be languages where $A \cup B \in \text{co-RE}$. Show that $A \in \text{co-RE}$, structuring your answer along the lines of what you did in part (ii) of this problem.

*If you have a language in co-RE, what can you say about that language? If you want to prove that a language is in co-RE, what do you need to prove about that language?*

Write your answer here.
The results you proved in parts (ii) and (iii) of this problem can be extended to show that if $A \cup B \in \text{RE}$, then $B \in \text{RE}$, and if $A \cup B \in \text{co-RE}$, then $B \in \text{co-RE}$, though you don’t need to prove this.

iv. Let $L$ be an undecidable language. Prove $L \cup \overline{L} \notin \text{RE}$ and $L \cup \overline{L} \notin \text{co-RE}$.

Provide a proof here.

Your result from part (iv) shows there are problems harder than $A_T$ and $L_D$; one example is $L_D \cup \overline{L}_D$. But even this nightmarish problem sits within a class of problems that can still be touched by computing power, if only in a very weak sense. Specifically, $L_D \cup \overline{L}_D$ is what’s called a $\Delta^0_2$ language, where $\Delta^0_2$ is a class of languages that contains all of RE and co-RE, plus a bunch of other problems. And $\Delta^0_2$ itself sits inside even larger classes of languages, stretching outward to infinity. Want to learn more? Take Phil 152!
4 The Lava Diagram

Below is a Venn diagram showing the overlap of different classes of languages we’ve studied so far. We have also provided you a list of twelve numbered languages. For each of those languages, draw where in the Venn diagram that language belongs. As an example, we’ve indicated where Language 1 and Language 2 should go. No proofs or justifications are necessary – the purpose of this problem is to help you build a better intuition for what makes a language regular, \( R \), \( RE \), or none of these.

We \textit{strongly} recommend reading over the Guide to the Lava Diagram before starting this problem.

To submit your answers, edit the file \texttt{LavaDiagram.h} in the \texttt{src/} directory of the starter files for this problem set.

1. \( \Sigma^* \)
2. \( L_D \)
3. \( \{ a^n \mid n \in \mathbb{N} \} \)
4. \( \{ a^n \mid n \in \mathbb{N} \text{ and is a multiple of } 137 \} \)
5. \( \{ 1^n + 1^m \mid 1^n + 1^m \mid m, n \in \mathbb{N} \} \)
6. \( \{ \langle M \rangle \mid M \text{ is a Turing machine and } L(M) \neq \emptyset \} \)
7. \( \{ \langle M \rangle \mid M \text{ is a Turing machine and } L(M) = \emptyset \} \)
8. \( \{ \langle M \rangle \mid M \text{ is a Turing machine and } L(M) = L_D \} \)
9. \( \{ \langle M, n \rangle \mid M \text{ is a TM, } n \in \mathbb{N}, \text{ and } M \text{ accepts all strings in its input alphabet of length at most } n \} \)
10. \( \{ \langle M, n \rangle \mid M \text{ is a TM, } n \in \mathbb{N}, \text{ and } M \text{ rejects all strings in its input alphabet of length at most } n \} \)
11. \( \{ \langle M, n \rangle \mid M \text{ is a TM, } n \in \mathbb{N}, \text{ and } M \text{ loops on all strings in its input alphabet of length at most } n \} \)
12. \( \{ \langle M_1, M_2, M_3, w \rangle \mid M_1, M_2, \text{ and } M_3 \text{ are TMs, } w \text{ is a string, and at least two of } M_1, M_2, \text{ and } M_3 \text{ accept } w. \} \)

Submit your edited \texttt{LavaDiagram.h} file on Gradescope.
Optional Fun Problem: Back to the Fourth Dimension (Extra Credit)

Picking up right where we left off in Problem 1, we saw that the tesseract is modeled by $Q_4$. Here is again one way of drawing $Q_4$. It’s hard to look at that picture and to imagine what it would “feel like” to hold it in your hand. But by using automorphisms, which you saw in the Problem Set 4, we can get a better intuition.

From your lived experience you know that if you pick up a cube, you can twist and turn it around and it has all sorts of symmetries. From a graph theory perspective, that would lead us to think that $Q_3$ should have many automorphisms, since each automorphism corresponds to a symmetry. So here’s a question: does $Q_4$ have the same sort of symmetries that you’d expect of a cube or a square?

A graph $G = (V, E)$ is called node-symmetric if, for any two nodes $u, v \in V$, there is an automorphism $\sigma$ of $G$ where $\sigma(u) = v$. Intuitively, this means that all the nodes in $G$ “look the same,” since for any pair of nodes there’s a symmetry of the graph (an automorphism) that makes the first node look like the second.

Prove that for every natural number $k$, the graph $Q_k$ is node-symmetric. You can assume that symmetric difference is commutative ($S \Delta T = T \Delta S$) and associative ($(S \Delta T) \Delta R = S \Delta (T \Delta R)$).

This result is beautiful and ties together concepts from throughout the course. We’d like to encourage everyone to at least take a stab at solving it. Here’s some guidance to get started:

This problem is much, much easier to solve if you take the time to work through some examples. Can you find an automorphism of $Q_2$ that maps $\emptyset$ to $\{0, 1\}$? To answer that question, try connecting it back to an actual square and its symmetries, see what happens to the corners, and see if you can use that to define an automorphism – there are two different choices here, one of which has a very simple interpretation. Then, find an automorphism of $Q_3$ that maps $\{1\}$ to $\{0, 2\}$ by thinking about what that means in terms of symmetries of a square. You should aim to find a general pattern here before moving on.

Once you’ve found a pattern of what these automorphisms look like, find a general formula for an automorphism $\sigma$ of $Q_k$ that maps some set $S$ to some set $T$. It should be pretty short and shouldn’t require a piecewise definition. Then, write down a list of everything you need to prove in order to show that $\sigma$ is indeed an automorphism; you did this on Problem Set Four, so perhaps that would be useful as a starting point. Then, go prove all those properties. In doing so, look back to Problem Set One or Problem Set Two. Perhaps there are some nice results from there you could use here?

Oh, and there’s no need to use induction here. ☺️

Write your answer here.

Grand Challenge Problem: $P \neq NP$ (Worth an A+, $1,000,000, and a Ph.D)

Prove or disprove: $P = NP$.

Take fifteen minutes and try this. Seriously. And if you can’t crack this problem, feel free to submit your best effort, or the silliest answer you can think of.

Write your answer here.