Problem Set 8

What problems are beyond our capacity to solve? Why are they so hard? And why is anything that we've discussed this quarter at all practically relevant? In this problem set, you'll explore the absolute limits of computing power.

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

This problem set has 34 possible points. It is weighted at 5% of your total grade.

Good luck, and have fun!

Due Monday, December 1st at 2:15 PM
Problem One: A Non-Paradoxical Machine (2 Points)

In Friday's lecture, we saw this TM:

\[ M = \text{“On input } w:\]
\[ \text{Have } M \text{ obtain its own description } \langle M \rangle. \]
\[ \text{Try to recognize whether } M \text{ accepts } w. \text{ (This step might loop if } M \text{ doesn't accept } w) \]
\[ \text{If } M \text{ determines that it will accept } w, \text{ then } M \text{ rejects } w. \]
\[ \text{If } M \text{ determines that it will not accept } w, \text{ then } M \text{ accepts } w. \]

We spent quite a lot of time talking about this TM and its properties because its behavior is definitely not obvious. In this question, we want you to prove some properties about this machine before you continue on to work with other self-referential machines.

Prove that \( M \) loops on all inputs. (Hint: Use a proof by contradiction. What if it halts?)

Problem Two: Password Checking (3 Points)

If you're an undergraduate here, you've probably noticed that the dorm staff have master keys they can use to unlock any of the doors in the residences. That way, if you ever lock yourself out of your room, you can, sheepishly, ask for help back in. (Not that I've ever done that or anything.)

Compare this to a password system. When you log onto a website with a password, you have the presumption that your password is the only possible password that will log you in. There shouldn't be a “master key” password that can unlock any account, since that would be a huge security vulnerability. But how could you tell? If you had the source code to the password checking system, could you figure out whether your password was the only password that would grant you access to the system?

Let's frame this in terms of Turing machines. If we wanted to build a TM password checker, “entering your password” would correspond to starting up the TM on some string, and “gaining access” would mean that the TM accepts your string. Let \( p \in \Sigma^* \) be your password. A TM that would work as a valid password checker would be a TM \( M \) where \( \mathcal{L}(M) = \{ p \} \); the TM accepts your string, and it doesn't accept anything else.

Given a TM, is there some way you could tell whether the TM was a valid password checker? Let \( p \in \Sigma^* \) be your password and consider the following language:

\[ L = \{ \langle M \rangle \mid M \text{ is a TM and } \mathcal{L}(M) = \{ p \} \} \]

Prove that \( L \) is undecidable (that is, \( L \notin \mathbb{R} \)). This means that there's no algorithm that can mechanically check whether a TM is suitable as a password checker.

Problem Three: Checking Regularity (3 Points)

In lecture, we proved that this language is not decidable:

\[ L = \{ \langle M \rangle \mid M \text{ is a TM and } \mathcal{L}(M) \text{ is regular } \} \]

As we mentioned, not only is this language undecidable; it's also not an \( \mathbb{RE} \) language either.

Prove that \( L \notin \mathbb{RE} \) by constructing a self-referential machine. As a hint, if the machine tries to recognize that it has a regular language and goes into an infinite loop, what do you know about whether \( M \) actually has a regular language? And in that case, what is the language of \( M \)?
**Problem Four: Equivalent TMs (6 Points)**

If you've taken CS106A, CS106B, or CS107, you've probably noticed that we have a lot of section leaders and TAs on staff. This is partially so that we can provide lots of one-on-one support and assistance in those courses, but part of it is also due to the fact that it's really hard to grade programming assignments. This question explores why.

When teaching a programming class, it would be really nice if we could fully autograde student programming submissions. Ideally, we'd like to be able to write our own reference solution to one of the programming problems, then check, for each student submission, whether that submission is in some way “equivalent” to our reference solution. If it is, then the submission must be correct, and if it isn't, then the submission must be incorrect.

Let's reformulate this as an equivalent problem about Turing machines. Suppose we have a student-submitted TM $M_1$ and a reference TM $M_2$. We'd like to be able to check whether these TMs have the same languages, that is, whether $\mathcal{L}(M_1) = \mathcal{L}(M_2)$. (This isn't perfectly analogous to our original problem, but it's a close enough match.) We'd like to see whether we can write a TM that can check whether these two TMs have the same language, and, if not, at least whether we can write a TM that checks whether these two TMs have different languages.

Consider the following language $EQ_{TM}$:

$$EQ_{TM} = \{ \langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } \mathcal{L}(M_1) = \mathcal{L}(M_2) \}$$

In other words, $EQ_{TM}$ is the set of all pairs of TMs that have the same language.

It turns out that this is a frighteningly hard problem to solve, and in this question you'll see why.

i. Using a self-referential Turing machine, prove that $EQ_{TM} \notin RE$. (Hint: Try making a machine that asks whether its language is empty.)

ii. Using a self-referential Turing machine, prove that $EQ_{TM} \notin co-RE$. (Hint: Prove that the complement of the language cannot be RE.)

Your results from parts (i) and (ii) of this problem show that the problem of testing whether two TMs have the same language is one of the “monster problems” that lies outside of what TMs can ever hope to accomplish. There's no procedure you can follow to check whether two TMs have the same language, and there's no procedure you can follow to check whether two TMs have different languages!

A note: If you have a copy of the Sipser textbook, in Chapter Five, Sipser proves that this language is neither RE nor co-RE using mapping reductions. You're welcome to read over these proofs if you'd like, but your answers to this question should use self-reference as a mechanism for showing these results.
Problem Five: This Program is Not Responding (1 Point)

Most operating systems provide some functionality to detect programs that are looping infinitely. Typically, they display a dialog box containing a message like these shown below:

These messages give the user the option to terminate the program or to let the program keep running in the hopes that it stops looping. An ideal OS would shut down any program that had gone into an infinite loop, since these programs just waste system resources (processor time, battery power, etc.) that could be better spent by other programs. It makes more sense for the OS to automatically detect programs that have gone into an infinite loop.

Why does the operating system have to display a message like this? Briefly justify your answer.

Problem Six: Verifiers and NTMs (8 Points)

Verifiers and NTMs give new perspectives on how to show that a language is \( \text{RE} \). In this question, we’ll ask you to design verifiers and NTMs for a few languages to get a better sense for how they relate.

Let \( L_{me} = \{ \langle M \rangle \mid M \text{ is a TM and } \mathcal{L}(M) \neq \emptyset \} \). In other words, \( L_{me} \) is the language of all Turing machines whose languages are nonempty.

i. Design a verifier for \( L_{me} \), then briefly justify why your verifier is correct. Specifically, you should design a TM \( V \) with the following property:

\[ \langle M \rangle \in L_{me} \iff \text{there is a certificate } c \in \Sigma^* \text{ such that } V \text{ accepts } \langle M, c \rangle \]

No formal proofs are necessary, but you should briefly justify both directions of the implication.

ii. Design an NTM for \( L_{me} \), then briefly justify why your NTM is correct. Specifically, you should design an NTM \( N \) with the following property:

\[ \langle M \rangle \in L_{me} \iff \text{there is a series of choices that causes } N \text{ to accept } \langle M \rangle \]

No formal proofs are necessary, but you should briefly justify both directions of the implication.

iii. A palindrome number is a number \( n \) that, in base 10, is the same when read forwards and backwards. Let \( L = \{ \langle n \rangle \mid n \in \mathbb{N} \text{ and some nonzero multiple of } n \text{ is a palindrome number } \} \). Design a verifier or NTM for \( L \), which shows that \( L \in \text{RE} \). Then, briefly justify your answer, making sure to justify both directions of the necessary implication.

iv. Let \( L_1 \) and \( L_2 \) be any \( \text{RE} \) languages. It turns out that their union, \( L_1 \cup L_2 \), is also an \( \text{RE} \) language. Given \( L_1 \) and \( L_2 \), design a verifier or NTM for \( L_1 \cup L_2 \), which shows that \( L_1 \cup L_2 \in \text{RE} \). Then, briefly justify your answer, making sure to justify both directions of the necessary implication.
Problem Seven: The Big Picture (10 Points)

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 12 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, co-RE, or none of these.

1. $\Sigma^*$
2. $EQ_{TM}$ (defined earlier in this problem set.)
3. $\{a^n | n \in \mathbb{N}\}$
4. $\{a^n | n \in \mathbb{N}$ and is a multiple of 137. $\}$
5. $\{a^n | n \in \mathbb{N}\} \cup \{a^n | n \in \mathbb{N}$ and is a multiple of 137. $\}$
6. $\{a^n | n \in \mathbb{N}$ and is not prime. $\}$
7. $\{M | M$ is a Turing machine and $L(M) = L_D. \}$
8. $\{M | M$ is a Turing machine and $L(M) = A_{TM.} \}$
9. $\{M, n | M$ is a TM, $n \in \mathbb{N}$, and $M$ accepts all strings of length at most $n. \}$
10. $\{M, n | M$ is a TM, $n \in \mathbb{N}$, and $M$ rejects all strings of length at most $n. \}$
11. $\{M, n | M$ is a TM, $n \in \mathbb{N}$, and $M$ loops on all strings of length at most $n. \}$
12. $\{M_1, M_2, M_3, w | M_1, M_2, and M_3$ are TMs, $w$ is a string, and at least two of $M_1, M_2,$ and $M_3$ accept $w. \}$
Problem Eight: Course Feedback (1 Point)
We want this course to be as good as it can be, and we'd appreciate your feedback on how we're doing. For a free point, please answer the feedback questions available online at

https://docs.google.com/forms/d/1Hmf9n89a2jPgBmM8ZL_HmSyz6y-Jj9voSnPw5dzoy8/viewform

We'll award full credit for any answers you give, as long as you answer all of the questions.

If you are working in a group, please have every member of the team fill this form out independently. We read over this feedback to get a sense for how to tune and improve the course, so any and all feedback is welcome.

Extra Credit Problem: Very Sparse Languages (1 Point)
Let $L$ be an RE language with the following property: for every natural number $n$, there is exactly one string of length $n$ contained in $L$.

Prove that $L$ is decidable (that is, $L \in R$).