This problem explores Turing machines, properties of the \textit{RE} and \textit{R} languages, and the limits of \textit{RE} and \textit{R} languages. This will be your first experience exploring the limits of computation, and I hope that you find it exciting!

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 30 possible points. It is weighted at 5\% of your total grade.

Good luck, and have fun!

\textbf{Due Monday, November 17th at 2:15 PM}

\section*{A Note on Turing Machine Design}

Some questions in this problem set will ask you to design Turing machines that solve various problems. In some cases, we will want you to write out the states and transitions within the Turing machine, and in other cases you will only need to provide a high-level description.

If a problem asks you to \textbf{draw the state-transition diagram for a Turing machine}, we expect you to draw out a concrete Turing machine by showing the states in that Turing machine and the individual transitions between them. If a question asks you to do this, as a courtesy to your TAs, please include with your Turing machines the following information:

\begin{itemize}
  \item A short, one-paragraph description of the high-level operation of the machine.
  \item A brief description of any subroutines in the Turing machine or any groups of states in the Turing machine that represent storing a constant in the TM's finite-state control.
\end{itemize}

For simplicity, you may assume that all missing transitions implicitly cause the TM to reject.

If a problem asks you to \textbf{give a high-level description of a Turing machine}, you can just provide a high-level description of the machine along the lines of what we did in lecture. More generally, unless you are specifically asked to give a state-transition diagram, any time that you are asked to design a Turing machine, you are encouraged to do so by giving a high-level description.

Unless stated otherwise, any TM you design should be a \textbf{deterministic} TM.

If you have any questions about this, please feel free to ask!
Problem One: The Collatz Conjecture (9 Points)

In last Friday's lecture, we discussed the Collatz conjecture, which claims that the following procedure (called the hailstone sequence) terminates for all positive natural numbers $n$:

- If $n = 1$, stop.
- If $n$ is even, set $n = n / 2$.
- If $n$ is odd, set $n = 3n + 1$.
- Repeat.

In lecture, we claimed that it was possible to build a TM for the language $L = \{ 1^n | \text{the hailstone sequence terminates for } n \}$ over the alphabet $\Sigma = \{ 1 \}$. In this problem, you will do exactly that. The first two parts to this question ask you to design key subroutines for the TM, and the final piece asks you to put everything together to assemble the final machine.

i. Draw the state transition diagram for a Turing machine subroutine that, given a tape holding $1^{2n}$ surrounded by infinitely many blanks, ends with $1^n$ written on the tape, surrounded by infinitely many blanks. You can assume the tape head begins reading the first 1, and your TM should end with the tape head reading the first 1 of the result. For example, given this initial configuration:

   ... 1 1 1 1 1 1 1 1 1 1 ...

   The TM would end with this configuration:

   ... 1 1 1 1 ...

   You can assume that there are an even number of 1s on the tape at startup and can have your TM behave however you'd like if this isn't the case. Please provide a description of your TM as discussed at the start of this problem set. (To give you a rough sense for how big a TM you'll need to build, our solution has between 5 – 10 states. If you have significantly more than this many states, you may want to rethink your design.)

ii. Draw the state transition diagram for a Turing machine subroutine that, given a tape holding $1^s$ surrounded by infinitely many blanks, ends with $1^{3s+1}$ written on the tape, surrounded by infinitely many blanks. You can assume that the tape head begins reading the first 1, and your TM should end with the tape head reading the first 1 of the result. For example, given this configuration:

   ... 1 1 1 ...

   The TM would end with this configuration:

   ... 1 1 1 1 1 1 1 1 1 1 ...

   Provide a description of your TM as discussed at the start of this problem set. (To give you a rough sense for how big a TM you'll need to build, our solution has between 5 – 10 states. If you have significantly more than this many states, you may want to rethink your design.)
iii. Using your TMs from parts (i) and (ii) as subroutines, draw the state transition diagram for a Turing machine $M$ that recognizes $L$. You do not need to copy your machines from part (i) and (ii) into the resulting machine. Instead, you can introduce “phantom states” that stand for the entry or exit states of those subroutines and then add transitions into or out of those states. (Check our TM for checking whether a number is composite as a reference.) Please provide a description of your TM as discussed at the start of this problem set. (To give you a rough sense for how big a TM you’ll need to build, our solution has between 5 – 10 states. If you have significantly more than this many states, you may want to rethink your design.)

Problem Two: Manipulating Encodings (5 Points)

In what follows, you can assume that $\Sigma = \{0, 1\}$. In Monday's lecture, we discussed string encodings of objects and ways in which TMs could manipulate those encodings. To help give you a better feeling for why this is possible, this question asks you to design two TM subroutines to perform common manipulations on encodings.

When discussing encodings, we saw that it was possible to take two encodings of objects $\langle O_1 \rangle$ and $\langle O_2 \rangle$ and combine them together to form a single string $\langle O_1, O_2 \rangle$ that encodes both of those objects. The specific encoding scheme we suggested was the following: the string $\langle O_1, O_2 \rangle$ is the string formed by

- doubling each character in $\langle O_1 \rangle$ (i.e. $0$ becomes $00$ and $1$ becomes $11$),
- then writing out the string $01$ as a delimiter, and finally
- writing out the description of $\langle O_2 \rangle$ unchanged.

For example, suppose that $\langle O_1 \rangle = 1010$ and $\langle O_2 \rangle = 11111$. The encoding $\langle O_1, O_2 \rangle$ would then be the string $\underline{11001100011111}$. (I’ve underlined the parts of the encoding corresponding to $\langle O_1 \rangle$ and $\langle O_2 \rangle$)

In order for this representation to be useful, Turing machines need to be to extract the first and second part of an encoded pair. This problem asks you to design TMs that do precisely these tasks.

i. Draw the state transition diagram for a TM subroutine that, given an encoding $\langle O_1, O_2 \rangle$ of two objects, ends with the string $\langle O_1 \rangle$ written on its tape, surrounded by infinitely many blanks. You can assume that the tape head begins reading the first character of $\langle O_1, O_2 \rangle$, and should design the TM so it ends with its tape head reading the first character of $\langle O_1 \rangle$. The input will be surrounded by infinitely many blanks.

For example, given this initial configuration:

```
... 1 1 0 0 0 0 1 1 0 1 0 1 1 0 ...
```

The TM should end in this configuration:

```
... 1 0 0 1 ...
```

You can assume that the encoding is properly formatted and can have your TM behave however you’d like if this isn’t the case. Please provide a description of your TM as discussed at the start of this problem set. (For reference, our solution has between 5 – 10 states. If you have significantly more than this, you might want to change your approach.)
ii. Draw the state transition diagram for a TM subroutine that, given an encoding \((O_1, O_2)\) of two objects, ends with the string \((O_2)\) written on its tape, surrounded by infinitely many blanks. You can assume that the tape head begins reading the first character of \((O_1, O_2)\), and should design the TM so it ends with its tape head reading the first character of \((O_2)\). The input will be surrounded by infinitely many blanks.

For example, given this initial configuration:

```
... 1 1 0 0 0 0 1 1 0 1 0 1 1 0 ...
```

The TM should end in this configuration:

```
... 0 1 1 0 ...
```

You can assume that the encoding is properly formatted and can have your TM behave however you'd like if this isn't the case. Please provide a description of your TM as discussed at the start of this problem set.
Problem Three: R and RE Languages (6 Points)

We have covered a lot of terminology and concepts in the past few days pertaining to Turing machines and R and RE languages. These problems are designed to explore some of the nuances of how Turing machines, languages, decidability, and recognizability all relate to one another. Please don't hesitate to ask if you're having trouble answering these questions – we hope that by working through them, you'll get a much better understanding of key computability concepts.

i. Give a high-level description of a TM \( M \) such that \( \mathcal{L}(M) \in \text{R} \), but \( M \) is not a decider. This shows that just because a TM's language is decidable, it's not necessarily the case that the TM itself must be a decider.

ii. Only languages can be decidable or recognizable; there's no such thing as an “undecidable string” or “unrecognizable string.” Prove that for every string \( w \), there's an \text{R} language containing \( w \) and an \text{RE} language containing \( w \).

iii. Prove that for every language \( L \), there is a decider \( M^+ \) that accepts every string in \( L \) and a decider \( M^- \) that rejects every string not in \( L \). Explain why this result doesn't prove that every language is decidable.

Problem Four: \( L_D \) (3 Points)

Here's another perspective of the proof that \( L_D \notin \text{RE} \). Let \( TM \) be the set of all descriptions of Turing machines. We can define a new function \( \mathcal{F} : TM \to \wp(TM) \) as follows:

\[
\mathcal{F}(\langle M \rangle) = \mathcal{L}(M) \cap TM
\]

In other words, \( \mathcal{F}(\langle M \rangle) \) is the set of all Turing machine descriptions in the language of \( M \).

Trace through the proof of Cantor's theorem from earlier in the quarter, assuming that the choice of the function \( f \) in the proof is the function \( \mathcal{F} \). What is the set \( D \) that is produced in the course of the proof? How does it relate to \( L_D \)?
Problem Five: Why Decidability and Recognizability? (6 Points)

There are two classes of languages associated with Turing machines – the \textbf{RE} languages, which can be recognized by a Turing machine, and the \textbf{R} languages, which can be decided by a Turing machine. Why didn’t we talk about a model of computation that accepted just the \textbf{R} languages and nothing else? After all, having such a model of computation would be useful – if we could reason about automata that just accept the \textbf{R} languages, it would be easier to see what problems are and are not decidable.

It turns out, interestingly, that assuming the Church-Turing thesis is true, there cannot be an effective method of computation that decides precisely the \textbf{R} languages. This problem explores why.

Suppose, for the sake of contradiction, that there an automaton called a \textbf{hypothetical machine} (or HM for short) that has the computational power to decide precisely the \textbf{R} languages. That is, \( L \in \textbf{R} \) iff there is a HM that decides \( L \). We can’t assume anything about how HMs work – perhaps they use an infinite tape and a finite-state control, or perhaps they just use magic widgets – so we can’t design concrete HMs. However, we will make the following (reasonable) assumptions about HMs:

\begin{itemize}
  \item Any \textbf{R} language is accepted by some HM, and each HM accepts an \textbf{R} language.
  \item All HMs halt on all inputs. That is, once started, an HM will always eventually terminate.
  \item Since HMs are a type of automaton, each HM is finite and can be encoded as a string. For any HM \( H \), we will let the encoding of \( H \) be represented by \( \langle H \rangle \).
  \item HMs are an effective model of computation. The Church-Turing thesis says that the Turing machine is at least as powerful as a HM, so there is some Turing machine \( U_{\text{HM}} \) that takes as input a description of a HM \( H \) and some string \( w \), then accepts if \( H \) accepts \( w \) and rejects if \( H \) rejects \( w \). Note that \( U_{\text{HM}} \) can never loop infinitely, because \( H \) is a hypothetical machine and always eventually accepts or rejects. More specifically, \( U_{\text{HM}} \) is the decider “On input \( \langle H, w \rangle \), where \( H \) is an HM and \( w \) is a string, simulate the execution of \( H \) on \( w \). \( H \) always terminates, so if \( H \) accepts \( w \), accept the input; otherwise, \( H \) rejects \( w \), so reject the input.”
\end{itemize}

Unfortunately, these four properties are impossible to satisfy simultaneously.

\begin{itemize}
  \item Consider the language \( \text{REJECT}_{\text{HM}} = \{ \langle H \rangle \mid H \text{ is a HM that rejects } \langle H \rangle \} \). Prove that \( \text{REJECT}_{\text{HM}} \) is decidable.
  \item Prove that there is no HM that decides \( \text{REJECT}_{\text{HM}} \).
\end{itemize}

Your result from (ii) allows us to prove that there is no class of automaton like the HM that decides precisely the \textbf{R} languages. If one were to exist, then it should be able to decide all of the \textbf{R} languages, including \( \text{REJECT}_{\text{HM}} \). However, there is no HM that accepts the decidable language \( \text{REJECT}_{\text{HM}} \). This means that one of our assumptions must have been wrong, so at least one of the following must be true:

\begin{itemize}
  \item HMs do not accept precisely the \textbf{R} languages, or
  \item HMs do not halt on all inputs, or
  \item HMs cannot be encoded as strings (meaning they lack finite descriptions), or
  \item HMs cannot be simulated by a TM (they are not effective models of computation)
  \item The Church-Turing thesis is incorrect.
\end{itemize}
Problem Six: 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/1Z0fKd-DNJKqV3RpBGgHs3iSsefOKN--h6TjC6PUDcPo/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 1: TMs and Regular Languages (1 Point Extra Credit)

We can measure the amount of time that a Turing machine takes to run on some string $w$ by counting the total number of transitions the Turing machine makes when running on $w$. Let's denote the number of transitions that $M$ takes when run on string $w$ by $T(M, w)$

Let $M$ be a Turing machine with input alphabet $\Sigma$. Prove that if there is some fixed constant $n$ such that $T(M, w) \leq n$ for any $w \in \Sigma^*$, then $L(M)$ is regular. (Intuitively, this means that if there's some fixed upper bound to the amount of time that a TM takes to run, then its language must be regular.)

Extra Credit Problem 2: Building Encodings (1 Point Extra Credit)

Draw the state transition diagram for a Turing machine that, given two encodings $\langle O_1 \rangle$ and $\langle O_2 \rangle$ separated by a blank, ends with the string $\langle O_1, O_2 \rangle$ written on its tape, surrounded by infinitely many blanks. You can assume that the tape head begins reading the first character of $\langle O_1, O_2 \rangle$, and should design the TM so it ends with its tape head reading the first character of $\langle O_1, O_2 \rangle$. The input will be surrounded by infinitely many blanks.

For example, given this initial configuration:

```
... 1 1 0 0 0 1 ...
```

The TM should end in this configuration:

```
... 1 1 1 1 0 0 0 1 0 0 1 ...
```

You can assume that the encoding is properly formatted and can have your TM behave however you'd like if this isn't the case. Please provide a description of your TM as discussed at the start of this problem set.