Problem Set 8

What lies beyond what can be solved by a computer? What intuitions can we build about those sorts of problems? In this problem set, you will learn how to reason about the unsolvable.

In any question that asks for a proof, you **must** provide a rigorous mathematical proof. You cannot draw a picture or argue by intuition. You should, at the very least, state what type of proof you are using, and (if proceeding by contradiction, contrapositive, or induction) state exactly what it is that you are trying to show. If we specify that a proof must be done a certain way, you must use that particular proof technique; otherwise you may prove the result however you wish.

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

This problem set has 125 possible points. It is weighted at 7% of your total grade. The earlier questions serve as a warm-up for the later problems, so be aware that the difficulty increases over the course of this problem set.

Good luck, and have fun!

Due Monday, March 11th at 12:50 PM

Problem One: Understanding Mapping Reductions (24 Points)

We often use mapping reductions to determine how hard one problem is by relating it to some other problem we already know about. However, we have to be careful when doing so.

i. Below is an **incorrect** proof that $REGULAR_{TM}$ (the language of TMs whose language is regular) is **RE**, even though in lecture we proved that it was neither **RE** or co-**RE**. Identify what is wrong with this proof. You should be sure that you are 100% positive what is wrong with this proof before you attempt any other problems on this problem set, because the mistake made here is extremely common!

Theorem: $REGULAR_{TM} \in$ **RE**.

Proof: We exhibit a mapping reduction from A_{TM} to $REGULAR_{TM}$; since A_{TM} is **RE**, this proves that $REGULAR_{TM}$ is **RE** as well.

Given a TM/string pair $\langle M, w \rangle$, let the function $f(\langle M, w \rangle) = \langle N \rangle$, where N is the TM defined in terms of M and w as follows:

> N = "On input x:
> If x has the form $\mathtt{0}^n\mathtt{1}^n$ for some $n \in \mathbb{N}$, then N accepts x.
> Run M on w.
> If M accepts w, then N accepts x.
> If M rejects w, then N rejects x."

We claim f is computable and omit the details from this proof. We also claim that $\langle M, w \rangle \in A_{TM}$ iff $f(\langle M, w \rangle) \in REGULAR_{TM}$. To see this, note that $f(\langle M, w \rangle) = \langle N \rangle \in REGULAR_{TM}$ iff $\mathscr{L}(N)$ is regular. We claim that $\mathscr{L}(N) = \Sigma^*$ if M accepts w and otherwise is $\{ \mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N} \}$; from this, we have that $\mathscr{L}(N)$ is regular iff M accepts w. To see that this is true, note that if M accepts w, then N accepts all strings, either because they are immediately accepted because they have the form $\mathtt{0}^n\mathtt{1}^n$, or because they are accepted when M accepts w. Thus $\mathscr{L}(N) = \Sigma^*$. Otherwise, if M does not accept w, then N accepts x iff x has the form $\mathtt{0}^n\mathtt{1}^n$. Consequently, we have that N accepts x iff $x \in \{ \mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N} \}$, so $\mathscr{L}(N) = \{ \mathtt{0}^n\mathtt{1}^n \mid n \in \mathbb{N} \}$, as required.

We thus have that $\mathscr{L}(N)$ is regular iff M accepts w iff $\langle M, w \rangle \in A_{TM}$. Thus $\langle M, w \rangle \in A_{TM}$ iff $f(\langle M, w \rangle) \in REGULAR_{TM}$. Therefore, f is a mapping reduction from A_{TM} to $REGULAR_{TM}$, so $A_{TM} \leq_M REGULAR_{TM}$. Consequently, $REGULAR_{TM} \in$ **RE**. ■

ii. Below is an **incorrect** proof that L_D (the language of TMs that do not accept their own encodings) is **RE**, even though in lecture we proved that it was not **RE**. Identify what is wrong with this proof. You should be sure that you are 100% positive what is wrong with this proof before you attempt any other problems on this problem set, because the mistake made here is extremely common!

Theorem: $L_D \in \mathbf{RE}$.

Proof: We exhibit a mapping reduction from L_D to A_{TM}; since A_{TM} is **RE**, this proves that L_D is **RE** as well. Given a TM $\langle M \rangle$, let the function $f(\langle M \rangle) = \langle N, \langle M \rangle \rangle$, where N is the following TM:

> N = "On input $\langle R \rangle$:
> Run R on $\langle R \rangle$.
> If R accepts $\langle R \rangle$, then N rejects $\langle R \rangle$.
> If R does not accept $\langle R \rangle$, then N accepts $\langle R \rangle$."

We claim that this function is computable and omit the details from this proof. We claim, moreover, that $\langle M \rangle \in L_D$ iff $f(\langle M \rangle) \in A_{TM}$. To see this, note that $f(\langle M \rangle) = \langle N, \langle M \rangle \rangle \in A_{TM}$ iff N accepts $\langle M \rangle$. By construction, N accepts $\langle M \rangle$ iff M does not accept $\langle M \rangle$. Finally, M does not accept $\langle M \rangle$ iff $\langle M \rangle \in L_D$. Thus $\langle M \rangle \in L_D$ iff $f(\langle M \rangle) \in A_{TM}$. Therefore, f is a mapping reduction from L_D to A_{TM}, so $L_D \leq_M A_{TM}$. Consequently, $L_D \in \mathbf{RE}$. ■

We've used mapping reductions to relate the difficulty of **R**, **RE**, and co-**RE** languages to one another. Could we use them to relate problems from classes of languages as well? Usually, the answer is no.

iii. Find an pair of languages L_1 and L_2 where $L_1 \leq_M L_2$ and L_2 is regular, but L_1 is not regular. Briefly explain why L_1 is not regular and why L_2 is regular, then prove that $L_1 \leq_M L_2$ by exhibiting a mapping reduction from L_1 to L_2. This means that we cannot establish that a language is regular by finding a mapping reduction from it to a known regular language.

To motivate why it is that we've defined reductions as we have, suppose that we change the definition of a mapping reduction as follows: For languages A and B, we say that $A \leq_M B$ iff there exists a computable function f such that for any $w \in A$, we have $f(w) \in B$.

iv. Prove that under this modified definition, *every* language A is mapping reducible to *every* language $B \neq \emptyset$. This (hopefully!) explains why we didn't define reductions this way.

Problem Two: Disjoint Unions (16 Points)

Given two languages L_1 and L_2, their union $L_1 \cup L_2$ is the set of all strings in at least one of L_1 and L_2. However, there is usually no connection between the "hardness" of L_1 and L_2 and the "hardness" of $L_1 \cup L_2$. For example, L_D is unrecognizable and $\overline{L}_D$ is co-unrecognizable, but $L_D \cup \overline{L}_D = \Sigma^*$ is decidable. To decide whether $w \in L_D \cup \overline{L}_D$, we don't actually need to check whether $w \in L_D$ or $w \in \overline{L}_D$. Since every string must belong to exactly one of these languages, we can always claim a string is in their union without having to inspect the string at all.

A more interesting construction is the *disjoint union* of two languages, a way of combining together strings from two different languages that tags each string with information about which language it came from. In what follows, let's assume all languages are over the alphabet $\Sigma = \{ \texttt{0}, \texttt{1} \}$. Formally, the disjoint union of two languages L_1 and L_2 is the language

$$L_1 \uplus L_2 = \{ \texttt{0}w \mid w \in L_1 \} \cup \{ \texttt{1}w \mid w \in L_2 \}$$

For example, if $L_1 = \{ \texttt{1}, \texttt{10}, \texttt{100}, \texttt{1000} \}$ and $L_2 = \{ \varepsilon, \texttt{0}, \texttt{1}, \texttt{00}, \texttt{01}, \texttt{10}, \texttt{11} \}$, then $L_1 \uplus L_2$ is the set

$$L_1 \uplus L_2 = \{ \texttt{01}, \texttt{010}, \texttt{0100}, \texttt{01000}, \texttt{1}, \texttt{10}, \texttt{11}, \texttt{100}, \texttt{101}, \texttt{110}, \texttt{111} \}$$

Notice how each string in $L_1 \uplus L_2$ is tagged with which language it originated in. Any string that starts with `0` came from L_1, and any string that starts with `1` came from L_2. Because of this tagging, the disjoint union of two languages produces a new language that is at least as "hard" as either of the input languages. In this problem, you will prove various important properties about the disjoint union.

i. Prove that if L_1 and L_2 are any languages, then $L_1 \leq_M L_1 \uplus L_2$. A similar proof can be used to show that $L_2 \leq_M L_1 \uplus L_2$, but you don't need to do that here.

ii. Prove that if L is recognizable but undecidable, then $L \uplus \overline{L}$ is neither **RE** nor co-**RE**.

Your result from (ii) shows how to construct extraordinarily hard problems out of problems that are already known to be hard. For example, $A_{TM} \uplus \overline{A}_{TM}$ is neither **RE** nor co-**RE**, nor is $L_D \uplus \overline{L}_D$.

Problem Three: A_{TM} and L_D (12 Points)

When we first saw the language L_D, we described it as a problem that was "harder" than A_{TM} because A_{TM} is **RE** while L_D is not. However, this might not have been a fair characterization. Both L_D and A_{TM} are hard problems, but neither one is "harder" than the other in the sense that neither one is mapping reducible to the other.

Prove that $A_{TM} \not\leq_M L_D$ and that $L_D \not\leq_M A_{TM}$.

Problem Four: Accept all the Strings! (20 Points)*

Consider the language

$$A_{ALL} = \{ \langle M \rangle \mid \mathscr{L}(M) = \Sigma^* \}$$

This language is neither **RE** nor co-**RE**, and in this problem you will see why.

i. Prove that $A_{TM} \leq_M A_{ALL}$. Since $A_{TM} \notin$ co-**RE**, this proves that $A_{ALL} \notin$ co-**RE** either.

The trickier part of the proof is proving that A_{ALL} is not **RE**. To do this, we will reduce $\overline{A}_{TM} \leq_M A_{ALL}$. Since $\overline{A}_{TM} \notin$ **RE**, this proves that $A_{ALL} \notin$ **RE** either.

Suppose that you are given a Turing machine M and a string w. We can construct a new TM N as follows:

> N = "On input x:
> Run M on w for $|x|$ steps.
> If M accepted within $|x|$ steps, reject.
> Otherwise, accept."

For example, on input `000`, N would run M on w for 3 steps, rejecting if M accepted w within that time and accepting otherwise. Similarly, if N were run on `010101`, N would accept if M did not accept w within 6 steps and would reject otherwise.

ii. Prove that M does not accept w iff $\mathscr{L}(N) = \Sigma^*$.

iii. Using your answer from (ii), prove that $\overline{A}_{TM} \leq_M A_{ALL}$. Since $\overline{A}_{TM}$ is not **RE**, this proves that A_{ALL} is not **RE** either.

Problem Five: Complementary Turing Machines (16 Points)

Consider the language *COMPLEMENT* defined as follows:

$$COMPLEMENT = \{ \langle M, N \rangle \mid M \text{ and } N \text{ are TMs and } \mathscr{L}(M) = \overline{\mathscr{L}(N)} \}$$

Prove that *COMPLEMENT* is neither **RE** nor co-**RE** by showing that $A_{ALL} \leq_M COMPLEMENT$. Make sure you justify how this reduction proves that *COMPLEMENT* is neither **RE** nor co-**RE**.

---

* Original image by Allie Brosh. This image courtesy of quickmeme.com.

Problem Six: Why All This Matters (32 Points)

Why are we studying A_{TM}, L_D, and these other "unsolvable" problems? It turns out that the fact that these problems are "unsolvable" has enormous practical implications for real computing systems.

Since their memory is finite, computers are not as powerful as Turing machines. However, as computers get more and more memory, we can think of them as getting progressively closer and closer in power to Turing machines. For the purposes of this question, we'll assume that a standard computer is equivalent in power to a Turing machine.

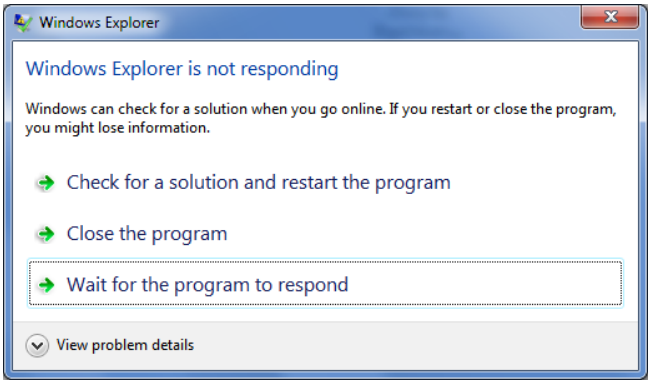
Because Turing machines and equivalently powerful models of computation can simulate one another, it is possible in any reasonable programming language to write a function like this one:

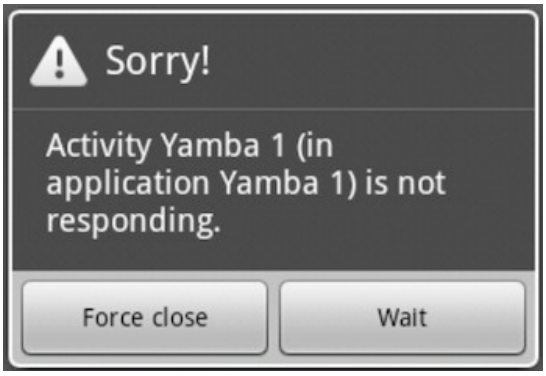
```
boolean simulateTuringMachine(TM M, string w)
```

This function takes in a suitably-encoded Turing machine M and a string w, then runs M on w. If M accepts w, then this function returns `true`. If M rejects w, then this function returns `false`. If M loops on w, then this function loops forever and never returns.

Using the existence of the above function, and the fact that a TM can simulate a computer, answer the following questions. In each proof, you will likely want to embed programs within TMs and TMs within programs.

i. 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.

Show that no OS can detect all programs that have gone into an infinite loop. Specifically, prove the following: if there were an OS that could always detect programs that have entered an infinite loop, then $\overline{HALT} \in$ **RE**. *(Hint: you will need to use the fact that computers can simulate TMs and TMs can simulate computers. You will probably need to embed a TM inside a computer program and a computer program inside a TM.)*

ii. Suppose that you want to create a website that teaches people how to program. On this site, you give a set of programming problems and invite users to submit programs that solve those problems. For each programming problem, you write a small set of test cases that submitted programs should be able to pass. Each test cases consists of a set of inputs to the user's program, along with the expected outputs. You can assume that all that matters is whether the program eventually outputs the right answer, not how long it takes to do so.

Show that it is impossible to automatically decide whether an arbitrary submitted program passes these tests. Specifically, prove the following: if a program could take in an input program and an arbitrary set of test cases and could decide whether the program passes the tests, then $A_{TM} \in \mathbf{R}$. *(Hint: Again, use the fact that TMs can simulate computers and computers simulate TMs. Try embedding a TM within a program and specifying a very simple set of test cases.)*

iii. Suppose that you want to build a *superoptimizing compiler* that takes as input a program and produces as output the smallest possible program equivalent to it. For example, given a program like this one:

```
int main() {
    int sum = 0;
    for (int i = 0; i < 10; i++) {
        sum += i;
    }
    cout << sum << endl;
}
```

The superoptimizer might output a program like this one:

```
int main() {
    cout << 45 << endl;
}
```

Similarly, given a program like this one that loops infinitely:

```
int main() {
    int x = 1;
    while (x != 0) {
        x--;
        x++;
    }
}
```

The superoptimizer might output

```
int main() {
    while(true) {}
}
```

Show that it is impossible to write a program that superoptimizes any input program. Specifically, prove the following: if a superoptimizer exists, then *HALT* $\in \mathbf{R}$. *(As before – you will probably want to embed programs in TMs and TMs in programs.)*

Problem Seven: Course Feedback (5 Points)

We want this course to be as good as it can be, and we'd really appreciate your feedback on how we're doing. For a free five points, please answer the following questions. We'll give you full credit no matter what you write (as long as you write something!), but we'd appreciate it if you're honest about how we're doing.

i. How hard did you find this problem set? How long did it take you to finish? Does that seem unreasonably difficult or time-consuming for a five-unit class?

ii. How is the pace of this course so far? Too slow? Too fast? Just right?

iii. Is there anything in particular we could do better? Is there anything in particular that you think we're doing well?

iv. **We will be holding a final exam review session and want it to be as useful as possible.** Are there any topics we should specifically focus on? Any topics that you think we can skip?

Submission instructions

There are three ways to submit this assignment:

1. Hand in a physical copy of your answers at the start of class. This is probably the easiest way to submit if you are on campus.
2. Submit a physical copy of your answers in the filing cabinet in the open space near the handout hangout in the Gates building. If you haven't been there before, it's right inside the entrance labeled "Stanford Engineering Venture Fund Laboratories." There will be a clearly-labeled filing cabinet where you can submit your solutions.
3. Send an email with an electronic copy of your answers to the submission mailing list (cs103-win1213-submissions@lists.stanford.edu) with the string "[PS8]" somewhere in the subject line. If you do submit electronically, please submit your assignment as a single PDF if at all possible. Sending multiple files makes it harder to print out and grade your submission.

Extra Credit Problem: A Smaller Version of A_{TM} (5 Points)

For any natural number k, define the language $A_{TM}^{(k)}$ as

$$A_{TM}^{(k)} = \{ \langle M, w \rangle \mid M \text{ is a TM with at most } k \text{ states and } M \text{ accepts } w \}$$

Prove that there exists a constant $k \in \mathbb{N}$ such that $A_{TM}^{(k)}$ is undecidable.