Problem Set 7

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

Start this problem set early. It contains six problems (plus one survey question and one extra credit problems), some of which require a fair amount of thought. I would suggest reading through this problem set at least once as soon as you get it to get a sense of what it covers.

As much as you possibly can, please try to work on this problem set individually. That said, if you do work with others, please be sure to cite who you are working with and on what problems. For more details, see the section on the honor code in the course information handout.

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.

If you are asked to prove something by induction, you may use weak induction, strong induction, the well-ordering principle, structural induction, or well-founded induction. In any case, you should state your base case before you prove it, and should state what the inductive hypothesis is before you prove the inductive step.

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 6% of your total grade. The earlier questions serve as a warm-up for the later problems, so do be aware that the difficulty of the problems does increase over the course of this problem set.

Good luck, and have fun!

Due Friday, May 25th at 2:15 PM

Problem One: Programming Turing Machines (20 Points)

Consider the alphabet $\Sigma = \{\texttt{0}, \texttt{1}, \texttt{≠}\}$ and the language

$$NE = \{ w\texttt{≠}x \mid w, x \in \{\texttt{0}, \texttt{1}\}^* \text{ and } w \neq x \}.$$

For example, **0110≠011** $\in NE$, **≠01** $\in NE$, but **01≠01** $\notin NE$, **≠** $\notin NE$, and **0≠1≠** $\notin NE$.

Write a program in **WB3** for this language. Recall that **WB3** supports the following commands:

- **Move left**;
- **Move right**;
- **Move left until { s_1, …, s_n }**, where s_1, …, s_n are tape symbols or variables;
- **Move right until { s_1, …, s_n }**, where s_1, …, s_n are tape symbols or variables;
- **Write** ***s***, for any tape symbol or variable ***s***;
- **Go to** ***L***, where ***L*** is a line number or a label;
- **If reading** ***s*****, go to** ***L***, where ***s*** is a symbol or variable and ***L*** is a line number or label;
- **Load current into** ***X***, where ***X*** is a variable;
- **Load** ***s*** **into** ***X***, where ***s*** is a tape symbol or variable and ***X*** is a variable;
- **If** ***X*** **=** ***Y*****, go to** ***L***, where ***X*** and ***Y*** are variables/constants and ***L*** is a line number/label;
- **Accept**; and
- **Reject**.

As a courtesy to your TAs, please comment your code and include a one- or two-paragraph summary of how the program works.

Problem Two: Nondeterministic Algorithms (24 points)

Nondeterministic Turing machines make it possible to solve many problems that might otherwise seem unrecognizably or undecidably hard.

i. Prove that **RE** is closed under union. That is, if $L_1 \in$ **RE** and $L_2 \in$ **RE**, then $L_1 \cup L_2 \in$ **RE** as well. Your proof can proceed along the lines of the ones we covered in lecture, so feel free to use high-level descriptions of Turing machines rather than formally writing out transition tables.

ii. Prove that the **RE** languages are closed under concatenation. That is, if $L_1 \in$ **RE** and $L_2 \in$ **RE**, then $L_1L_2 \in$ **RE** as well. *(Hint: If you are given a string w in L_1L_2 and knew how to cut it into strings x and y such that $x \in L_1$ and $y \in L_2$, could you check that you had guessed correctly?)*

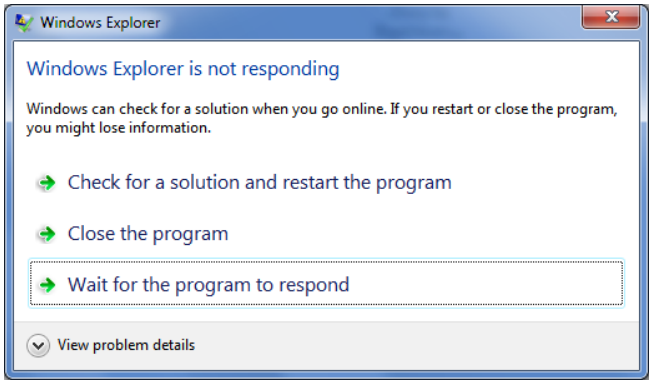
iii. Consider the language

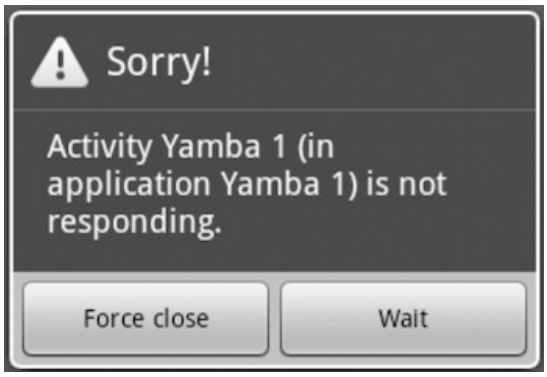
$$L_{ne} = \{ \langle M \rangle \mid M \text{ is a TM and } \mathscr{L}(M) \neq \varnothing \}$$

It is possible to show that this language is undecidable, but you don't need to prove this. Instead, show that $L_{ne} \in$ **RE** by giving a high-level description of an NTM that recognizes L_{ne}, then proving that your NTM is correct.

Problem Three: This Program is Not Responding (12 Points)

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.

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. Since it makes more sense for the OS to automatically detect programs that have gone into an infinite loop, why does the OS have to ask the user whether to terminate the program or let it keep running?

Problem Four: Unrecognizable and Undecidable Languages (20 Points)

i. Prove or disprove: If L_1 is unrecognizable and $L_1 \subseteq L_2$, then L_2 is unrecognizable.

ii. Prove or disprove: If L_2 is unrecognizable and $L_1 \subseteq L_2$, then L_1 is unrecognizable.

iii. Do your answers still hold when "unrecognizable" is replaced with "undecidable?" What about "not context-free?" What about "not regular?"

Problem Five: A_{TM} is Unrecognizable? (16 Points)

In Friday's lecture, we proved that $A_{TM} \in \mathbf{RE}$ but that $A_{TM} \notin \mathbf{R}$. Our proof worked as follows:

- Assume, for the sake of contradiction, that A_{TM} is decidable.
- Using a decider for A_{TM} as a subroutine, construct a recognizer for L_D.
- Arrive at a contradiction, since we know that L_D is unrecognizable.
- Conclude, therefore, that A_{TM} is undecidable.

Initially, it might seem like we could easily modify this proof to show that $A_{TM} \notin \mathbf{RE}$ by simply changing our assumptions as follows:

- Assume, for the sake of contradiction, that A_{TM} is recognizable.
- Using a recognizer for A_{TM} as a subroutine, construct a recognizer for L_D.
- Arrive at a contradiction, since we know that L_D is unrecognizable.
- Conclude, therefore, that A_{TM} is unrecognizable.

Below is an incorrect proof along these lines. This proof contains an error that renders the proof invalid. Give the **exact line** of the proof that contains the error and explain why it is incorrect.

Theorem: A_{TM} is unrecognizable.

Proof: By contradiction; assume that A_{TM} is recognizable and let H be a recognizer for it. Then consider this machine D:

> D = "On input $\langle M \rangle$:
> Construct $\langle M, \langle M \rangle \rangle$.
> Run H on $\langle M, \langle M \rangle \rangle$.
> If H accepts $\langle M, \langle M \rangle \rangle$, reject.
> If H rejects $\langle M, \langle M \rangle \rangle$, accept."

We claim that $\mathscr{L}(D) = L_D$. To see this, note that D accepts $\langle M \rangle$ iff H rejects $\langle M, \langle M \rangle \rangle$. Since H is a recognizer for A_{TM}, H rejects $\langle M, \langle M \rangle \rangle$ iff $\langle M, \langle M \rangle \rangle \notin A_{TM}$. Since $\langle M, \langle M \rangle \rangle \notin A_{TM}$ and $\langle M, \langle M \rangle \rangle$ is a valid TM/string pair, this means that $\langle M \rangle \notin \mathscr{L}(M)$. Consequently, we have that D accepts $\langle M \rangle$ iff $\langle M \rangle \notin \mathscr{L}(M)$. Therefore, $\mathscr{L}(D) = L_D$.

Since $\mathscr{L}(D) = L_D$, we know that $L_D \in \mathbf{RE}$. But this is impossible, since we know that $L_D \notin \mathbf{RE}$. We have reached a contradiction, so our assumption must have been wrong. Thus A_{TM} is unrecognizable. ■

Problem Six: Why Decidability and Recognizability? (28 Points)

There are two classes of languages associated with Turing machines – the **RE** languages, which can be recognized by a Turing machine, and the **R** languages, which can be decided by a Turing machine. But why are there two different complexity classes? Why didn't we talk about a model of computation that accepted just the **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 recursive languages, it would be much easier to see what problems are and are not decidable.

It turns out, interestingly, that there is no class of automata with this property, and in fact the only way to build automata that can decide all recursive languages is to also have those automata also accept some languages that are **RE** but not **R**. This problem explores why.

Suppose, for the sake of contradiction, that there is an automaton called a **deciding machine** (or DM for short) that has the computational power to decide precisely the **R** languages. That is, $L \in \mathbf{R}$ iff there is a DM that decides L.

We will make the following (reasonable) assumptions about deciding machines:

- Any recursive language is accepted by some DM, and each DM accepts a recursive language.
- Since DMs accept precisely the recursive languages, all DMs halt on all inputs. That is, all DMs are deciders.
- Since deciding machines are a type of automaton, each DM is finite and can be encoded as a string. For any DM D, we will let the encoding of D be represented by $\langle D \rangle$.
- Since DMs are an effective model of computation, the Church-Turing thesis says that the Turing machine is at least as powerful as a DM. Thus there is some Turing machine U_D that takes as input a description of a DM D and some string w, then accepts if D accepts w and rejects if D rejects w. Note that U_D can never loop infinitely, because D is a deciding machine and always eventually accepts or rejects. More specifically, U_D is the decider "On input $\langle D, w \rangle$, simulate the execution of D on w. If D accepts w, accept. If D rejects w, reject."

Unfortunately, these four properties are impossible to satisfy simultaneously.

i. Consider the language $REJECT_{DM} = \{ \langle D \rangle \mid D \text{ is a DM that rejects } \langle D \rangle \}$. Prove that $REJECT_{DM}$ is decidable.

ii. Prove that there is no DM that decides $REJECT_{DM}$.

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

- DMs do not accept precisely the **R** languages, or
- DMs are not deciders, or
- DMs cannot be encoded as strings (meaning they lack finite descriptions), or
- DMs cannot be simulated by a TM (they are not effective models of computation)

Thus there is no effective model of computation that decides just the recursive languages.

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?

ii. Does that seem unreasonably difficult or time-consuming for a five-unit class?

iii. Did you attend Monday's problem session? If so, did you find it useful?

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

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

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-spr1112-submissions@lists.stanford.edu) with the string "[PS7]" somewhere in the subject line. If you do submit electronically, please submit your assignment as a single PDF if at all possible. Sending multiple image files makes it much harder to print out and grade your submission.

If you are an SCPD student, we would strongly prefer that you submit solutions via email. Please contact us if this will be a problem.

Extra Credit Problem: Non-Closure under Homomorphism (5 Points Extra Credit)

Unlike the regular languages and context-free languages, recursive languages are *not* closed under homomorphism. In particular, there exists a recursive language L over some alphabet Σ_1 and a homomorphism $h^* : \Sigma_1^* \to \Sigma_2^*$ such that $h^*(L)$ is undecidable. Find a language L and homomorphism h^* such that that L is recursive, h^* is a homomorphism, and $h^*(L)$ is undecidable. Then prove that each of these properties holds for your choice of L and h^*.