CS 181D, Complexity Theory
Fall 2012
Homework 1
Due Wednesday, September 12, 11:59 PM

Please submit your assignment as a pdf document named `hw1.pdf` at the course submission website: `http://www.cs.hmc.edu/~submissions/submissionsFall12`.

1. **[20 Points] A Weird Language!** Describe a language L such that neither L nor $\overline{L}$ are recursively enumerable. (You must *prove* that the languages that you describe are not recursively enumerable.)

2. **[25 Points] To Infinity and Beyond!**

   (a) Consider a set S of strings $S = \{0^{i_1}, 0^{i_2}, \ldots\}$ representing the natural numbers $i_1, i_2, \ldots$. This set may be finite or infinite. Is every such set recursively enumerable? Explain carefully.

   (b) Consider a set $S = \{0^{i_1}, 0^{i_2}, \ldots\}$ as above and assume that it is known to be recursively enumerable (either because you have shown that all such sets are recursively enumerable or, if they aren't, then this one was guaranteed to be recursively enumerable). Consider the infinite union of languages $L_{\text{union}} = \bigcup_{i \in S} L(M_i)$. Is this language L_{union} necessarily recursively enumerable? Explain carefully.

   (c) Is the union of any infinite number of recursively enumerable languages always recursively enumerable? Explain carefully.

3. **[25 Points] Kleene Star, P, and NP** Recall that if L is a language then $L^0 = \{\epsilon\}$, $L^1 = L$, and $L^i = LL^{i-1}$ (the concatenation of L and L^{i-1}). The Kleene Closure (pronounce "Clean Knee")[1] of L is denoted L^* and is defined

[1] From the History of Mathematics Archives at University of St. Andrews, Scotland: "Stephen C. Kleene studied for his first degree at Amherst College. He went on to receive a doctorate from Princeton University in 1934, supervised by Church, for a thesis entitled "A Theory of Positive Integers in Formal Logic." Then Kleene taught at Princeton until he joined the University of Wisconsin at Madison in 1935. He became a full professor at the University of Wisconsin at Madison in 1948 and remained on the staff there until he retired in 1979. Kleene's research was on the theory of algorithms and recursive functions. He developed the field of recursion theory with Church, Godel, Turing and others. He contributed to mathematical Intuitionism which had been founded by Brouwer. His work on recursion theory helped to provide the foundations of theoretical computer science. By providing methods of determining which problems are soluble, Kleene's work led to the study of which functions can be computed."

to be

$$\bigcup_{i \geq 0} L^i$$

In other words, L^* is the language made up of all strings w where w can be written as the concatenation of 0 or more strings in L.

(a) (Warmup) Show that if L is recursive then L^* is recursive.

(b) (Warmup) Show that if L is recursively enumerable then L^* is recursively enumerable.

(c) (Moderate) A recursive language L is said to be in the class NP (non-deterministic polynomial time) if there exists some non-deterministic TM M and a polynomial $p(n)$ such that M accepts L and such that for every string w, all computation paths of M terminate within $p(|w|)$ steps. (Note that this says "for every string w", so w need not necessarily be in L.) Show that if L is in NP then L^* is in NP.

(d) (A bit more challenging) A recursive language L is said to be in the class P if there exists some deterministic TM M and a polynomial $p(n)$ such that M accepts L and such that for every string w, M terminates within $p(|w|)$ steps. (Note that this says "for every string w", so w need not necessarily be in L.) Show that if L is in P then L^* is in P.

4. **[30 Points] The Busy Beaver!** So far we have looked at Turing Machines as acceptors of languages; a TM accepts a word if it eventually halts in the accept state and it rejects a word if it eventually halts in the reject state or runs forever.

   Another view of a Turing Machine is as a device that computes a function. A **Function-Computing TM (FCTM)** is a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{done})$ where Q is the finite set of states, $\Sigma = \{0\}$ is the input alphabet, Γ is the tape alphabet containing 0, the blank symbol, and a finite number of additional symbols that are used in the process of the computation, q_0 is the start state, and q_{done} is the "done computing" state. We say that a TM M computes the function $f : \mathbb{N} \to \mathbb{N}$ if whenever M starts in the start state with its tape head at the left end ("beginning") of the tape with exactly k consecutive 0's on the tape, it eventually returns its tape head to the starting position and enters the state q_{done} (where it is forced to halt) with exactly $f(k)$ consecutive 0's on the tape. A function that is computable by a FCTM is said to be *Turing-computable*.

(a) First, give a simple proof that shows that there exists at least one function from $\mathbb{N}$ to $\mathbb{N}$ that is not Turing-computable. In the remainder of this problem, we will find an *actual* function that is not Turing-computable.

(b) Next, show that every Turing-computable function is computable using only the tape alphabet $\{0, 1, B\}$ where B is the blank symbol. In other words, we may restrict the tape alphabet with no loss of generality.

(c) Next, draw the state diagram for a FCTM that computes the function $f(n) = 2n$ and uses only the tape alphabet $\{0, 1, B\}$. *Alternatively, you can methodically explain in clear English how such a FCTM would operate. However, if you complete the state diagram, read on and earn a special little prize!* If you choose to draw the state diagram, please use the following notation on the arcs of your state diagram: Given tape alphabet Γ, an arc between two states may be labelled $r \to w, D$ where $r, w \in \Gamma$ and $D \in \{L, R\}$. This represents the transition in which on reading symbol r, the machine should write symbol w and move in direction D where D is either L (left) or R (right).

If you're careful, this state diagram won't be very large at all. For completing this task, you will receive the **IN**ternational **T**ur**I**ng **M**achine **ID**entific**AT**ion passp**OR**t (INTIMIDATOR). The INTIMIDATOR is recognized at participating institutions of higher learning, including some of the finest graduate programs, and exempts you from any subsequent Turing Machine construction in the future. "The INTIMIDATOR, don't leave college without it!"

Note: If you are familiar with JFLAP, you may use it and print out the TM.

(d) Now, let's consider the famous **busy beaver** function. We'll show that this function is not computable by any Turing Machine (also known simply as "uncomputable").

Imagine that we're told we can use only n states to build a TM, and we want to build a TM that will start with an empty tape and write down as many 0's as possible, return its tape head to the starting position, and then halt. The TMs tape alphabet is only $\{0, 1, B\}$. What's the largest number of consecutive 0's that our TM could write and then halt? Clearly, the more states you have, the more 0's you can write. The busy beaver function formalizes this idea. The busy beaver function $\beta : \mathbb{N} \to \mathbb{N}$ is defined as follows: For each $n \geq 0$, $\beta(n)$ is the largest number of 0's that a Turing Machine could write on its tape and halt if the TM has exactly n

states, has the tape alphabet $\{0, 1, B\}$, and starts in the start state with an entirely blank tape. We're going to show that there does not exist a TM that computes the busy beaver function!

i. First argue that for any n and m, $\beta(n) \geq \beta(m)$ if and only if $n \geq m$.

ii. Now show that if $f : \mathbb{N} \to \mathbb{N}$ is computed by a FCTM with k states and tape alphabet $\{0, 1, B\}$, then $\beta(n + k + 42) \geq f(n)$ for all n. In other words, there exists a TM with $n + k + 42$ states which writes at least $f(n)$ 0's and halts when started on a blank tape. (*Note:* The number 42 is overkill. A much smaller constant will work in its place, but the constants 0 or 1 might be too small.)

iii. Finally argue that the function β is not computed by any FCTM. Start the proof as follows: "Assume that β is computed by some FCTM. Then the function $f(n) = \beta(2n)$ would also be computed by some FCTM (explain why!)." Now use the earlier results from this problem to obtain the desired contradiction. Pretty Slick!!

5. **[20 Point OPTIONAL BONUS PROBLEM] Every TM has a corresponding unrestricted grammar.** In class we stated, without proof, that the languages generated by unrestricted grammars are exactly the recursively enumerable languages. Prove it!