Problem Set 8 Solutions

Problem 1. A probabilistic TM (PTM) is like an ordinary TM except for having a distinguished state $q_8$ such that, when it enters this state, it will randomly print a 0 or 1 on its tape, each with probability 0.5. The machine is said to have “flipped a coin.” Let $\Pr[M \text{ accepts } x]$ be the probability, over all these coin flips, that $M$ accepts when we start it in configuration $(\varepsilon, q_0, x)$. We say that a PTM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R, q_8)$ decides $L$ with error probability $\varepsilon$ if $M$ always halts and $x \in L \Rightarrow \Pr[M \text{ accepts } x] \geq 1 - \varepsilon$ and $x \notin L \Rightarrow \Pr[M \text{ accepts } x] \leq \varepsilon$. We say that PTM $M$ decides $L$ if it decides $L$ with error probability 1/4.

1.1 Suppose a PTM $M$ decides $L$. Describe a (conventional) TM $M'$ that decides $L$. How much slower is $M'$ than $M$? (If $M$ takes at most $T$ steps to decide $x$, how long will $M'$ take to do so?)

On input $x$, let $M'$ simulate the running of $M$ on $x$ for all possible sequences of coin flips. Machine $M'$ will count how many sequences of coin flips cause $M(x)$ to accept and how many cause $M(x)$ to reject. When the first is bigger have (it will have at least three times as big), $M'$ accept. When the second is bigger, have $M'$ reject. If $M$ runs in time $T$ and uses at most $R$ coin flips, then the running time for the algorithm just described will be at least $T2^R \leq T2^T$. This is very slow—an exponential slowdown.

1.2 Suppose a PTM $M$ decides $L$. Describe a PTM $M'$ that decides $L$ with error probability $\varepsilon = 10^{-10}$. Make $M'$ run as fast as you can, analyzing its running time with a Chernoff bound, say the following one: If $X_1, \ldots, X_n$ are independent $[0, 1]$-valued variables and $X = \sum_{i=1}^n X_i$ is their sum and $\mu = E[X]$ is its expected value then $\Pr[X \geq (1 + \lambda)\mu] \leq e^{-\lambda^2\mu/(2(1+\lambda))}$ and for any $\lambda \geq 0$.

On input $x$, let $M'$ run $M$ on input $x$ for $n$ times, using random independent coins for each run. Let $X$ be the number of times that $M$ said to accept. If $X \geq n/2$ have $M'$ accept $x$. If $X < n/2$ have $M'$ reject $x$. Ignoring the overhead in the simulation and counting, the running time of $M'$ is $nT$, where $T$ is the running time of $M$. But what $n$ should we choose for this to work?

Suppose $x \notin L(M)$. Let $X_i \in \{0, 1\}$ be the event that $M$ accepts $x$ on the $i$th run. Let $X = \sum_{i=1}^n X_i$. So $M'$ accepts $x$ exactly when $X \geq n/2$. The $X_i$ values are independent because we use independent coins for each run. The expected value of $X_i$ is at most $1/4$ so the expected value of their sum is at most $n/4$. Machine $M'$ will (wrongly) accept $x$ only if $X$ exceeds its mean of at most $n/4$ by at least $n/4$, which is when $X$ is at least double its mean. Setting $\lambda = 1$ in the Chernoff bound we quoted, we conclude that $M'$ accepts $x$ with probability at most $e^{-n/3} = e^{-n/12}$.

Now suppose $x \in L(M)$ and let $X_i \in \{0, 1\}$ be the event that $M$ rejects $x$ on the $i$th run and let $X = \sum_{i=1}^n X_i$. So $M'$ will (wrongly) reject $x$ when $X > n/2$, which is when $X$ exceeds its expected value (that’s at most $n/4$) by a multiplicative factor of at least $2_\varepsilon$. As above, this happens with probability at most $e^{-n/3} = e^{-n/12}$.

We wish to ensure that $M'$ is wrong with probability at most $10^{-10}$, which will happen if $n$ is large enough that $e^{-n/12} \leq 10^{-10}$. Taking $\ln$ of both sides, we need $n/12 \geq 10 \ln 10$, so $n \geq 120 \ln 10$. As $n$ must be an integer, we select $n = \lceil 120 \ln 10 \rceil = 277$.

Concluding: just run $M(x)$ 277 times and take the majority vote to see if you will accept $x$. You’ll be wrong with probability less than 1 in 10 billion. The constant 277 can be improved as shown below. The meaning of this result is that the choice of error probability $\varepsilon$ doesn’t matter very much, since you can reduce it to whatever you want without paying an excessive price.

1.3* Extra credit. Make your algorithm of 1.2 more efficient by choosing a smaller constant—the smallest possible, as established by a computer-supported calculation. That is, identify the smallest constant that will “work” in your algorithm of 1.2 get the requested error probability. Finding his will require some
basic combinatorics (stuff you should have learned in ECS 20) plus writing a short computer program. Include your program and a run showing the constant you need.

The exact probability that \( M' \) above makes an error is

\[
f(n) = \sum_{k = \lceil n/2 \rceil}^{n} \binom{n}{k} \left( \frac{1}{4} \right)^k \left( \frac{3}{4} \right)^{n-k}
\]

where \( \binom{n}{k} = \frac{n!}{k!(n-k)!} \) is the “combinations” function. The following Maple program computes exact values. It shows that \( f(n) \) drops under \( 10^{-10} \) when \( n \) reaches 141; in other words, it is enough for \( M' \) to run \( M \) 141 times, accepting if there are 71 or more accepts. Programmed in Maple, the program does calculations exactly (until the evalf is called), so one can be sure that there are no numerical-accuracy problems that might otherwise be a concern.

```maple
> restart; with(combinat);
> f := proc (n)
local k, s;
    s := 0; for k from ceil((1/2)*n) to n do s := s+numbcomb(n, k)*(1/4)^k*(3/4)^(n-k) end do;
    return s
end proc;
> evalf(f(140)), evalf(f(141));
-10       -11
 1.789256097 10 , 8.825840527 10
> eps := 10^(-10); evalb(f(140) < eps); evalb(f(141) < eps);
false       true
```

**Problem 2.** Guess the classification for each of the following languages as either **recursive**, **r.e.** but not not co-r.e., **co-r.e.** but not r.e., or **neither** r.e. nor co-r.e. No justification is required.

While you didn’t have to include explanations, I will.

2.1 \( \{ \langle M, k \rangle : M \text{ is a TM that accepts at least one string of length } k \} \). **r.e.** An NTM can guess a string \( w \) of length \( k \) and verify that \( M \) accepts it.

2.2 \( \{ \langle M, k \rangle : M \text{ is a TM that accepts at most one string of length } k \} \). **co-r.e.** You can guess two distinct strings of length \( k \) and verify that \( M \) accepts both.

2.3 \( \{ \langle M \rangle : M \text{ is a TM and } M \text{ has 50 states} \} \). **recursive.** Just count the states

2.4 \( \{ \langle M \rangle : M \text{ is a TM and } L(M) \text{ contains a palindrome} \} \). **r.e.** Just guess the palindrome \( x \) that \( M \) accepts and verify that it’s a palindrome and that \( M \) accepts it.

2.5 \( \{ \langle M \rangle : M \text{ is a TM and } L(M) = \emptyset \} \). **co-r.e.** Just guess a string in \( L(M) \) and verify.

2.6 \( \{ \langle M \rangle : M \text{ is a TM and } L(M) \text{ is r.e.} \} \). **recursive.** Every valid TM encoding is in this language.

2.7 \( \{ \langle M \rangle : M \text{ is a TM and } L(M) \text{ is decidable} \} \). **neither.**

2.8 \( \{ \langle G \rangle : G \text{ is a CFG and } L(G) = \Sigma^* \} \). **co-r.e.** One can guess a string \( x \notin L(G) \) and then verify, using the CYK algorithm, that it’s not.

2.9 \( \{ \langle M \rangle : M \text{ is a C-program that halts on } \langle M \rangle \} \). **r.e.** Run \( M \) on its own description and see.

2.10 \( \{ \langle M \rangle : M \text{ is a TM that sometimes diverges} \} \). **neither**
3.2. **Prove:** 

Let \( w \in \text{ECS 120 ps8-soln: Problem Set 8 Solutions} \)

2.11 \( \{ \langle M, w \rangle : M \text{ is a TM and } M \text{ that uses at most 50 tape cells when run on } w \} \). **recursive.** If \( M \) has tape alphabet with \( \gamma = |\Gamma| \) then there are only \( \gamma^{50} \) strings that might be written on the tape. If it has \( q = |Q| \) states then, at any point in time, it can be in any of these \( q \) states and scanning any of the 50 squares. Thus there are at most \( 50q\gamma^{50} \) possible configurations. So run \( M \) on \( w \) for \( 50q\gamma^{50} + 1 \) steps. If \( M \) ever attempts to use more than 50 tape squares, **reject.** If \( M \) halts without using more than 50 tape squares, **accept.** If after the prescribed number of steps have been simulated the machine has still not halted, then \( M \) must be in an infinite loop, one that uses 50 or fewer tape squares and repeats configurations. So **accept.** Alternatively, just simulate \( M \), looking to see if it ever uses more than 50 squares, halt, or repeats a configuration. Accept or reject accordingly. The reasoning above ensures termination.

2.12 \( \{ \langle G \rangle : G \text{ is a CFG and } G \text{ accepts an odd-length string} \} \). **decidable.** Using the procedures seen in class, intersect with the regular language of all odd length strings and test foremptiness.

2.13 \( \{ \langle M \rangle : M \text{ is a smallest (fewest-state) NFA for } L(M) \} \). **decidable.** Try all smaller NFAs and use the procedure seen in class to test for equality. Highly inefficient, but still a decision procedure.

2.14 \( \{ \langle p \rangle : p \text{ is a multivariate polynomial and } p(x) = 0 \text{ for some } x \in \mathbb{Z}^n \} \). **r.e.**

2.15 \( \{ \langle p, q \rangle : p \text{ and } q \text{ are multivariate polynomials and } p(x) = q(x) \text{ for all } x \in \mathbb{Z}^n \} \). **decidable.** I claim that \( p \) and \( q \) agree on all points only if they are identical polynomials (whence one can list terms in a canonical order and compare). I will leave this to the top students in class to prove. You can email me a proof if you have an elegant one. 

**Problem 3.** Say that a language \( L = \{ x_1, x_2, \ldots \} \) is enumerable if there exists a two-tape TM \( M \) that outputs \( x_1 \sharp x_2 \sharp x_3 \sharp \cdots \) on a designated output tape. The other tape is a designated work tape. The output tape is write-only, with the head moving only from left-to-right. Say that \( L \) is enumerable in lexicographic order if \( L \) is enumerable as above but where, additionally, \( x_1 < x_2 < x_3 < \cdots \), where “<” denotes the lexicographic order.

3.1. **Prove:** \( L \) is r.e. iff it is enumerable.

Suppose that \( L \) is r.e., say \( L = L(M) \). Let \( w_i \) denote the \( i \)-th string in lexicographic order. For \( i \) going from 1, 2, 3, \ldots , run \( M \) on \( w_i \) for \( i \) steps. If \( M(w_i) \) has accepted by then, then output \( w_i \) on the output tape.

Conversely, suppose there exists a machine \( M \) that lists \( L \). Then a machine \( M' \) to accept \( L \) works as follows. On input \( x \), simulate the running of \( M \), watching what is produced by the output tape. If ever \( \sharp x \sharp \) appears on this tape, accept \( x \).

3.2. **Prove:** \( L \) is recursive iff it is enumerable in lexicographic order.

Let \( w_i \) denote the \( i \)-th string in lexicographic order. Suppose that \( L \) is decidable. Let \( M \) be a TM that decides \( L \). The the following machine \( M' \) lists the strings of \( L \) in lexicographic order:

Machine \( M' \) does the following:

\[
\text{for } i = 1 \text{ to } \infty \text{ do} \\
\quad \text{if } M(w_i) \text{ accepts then output } w_i \sharp \text{ on the output tape}
\]

For the other direction, suppose that TM \( M' \) lists \( L \) in lexicographic order. If \( L \) is finite then \( L \) is decidable; we know there is a machine \( M \) that decides \( L \), and we are done. Otherwise, when \( L \) is infinite, we construct a machine \( M \) to decide \( L \) as follows:

Machine \( M \) on input \( x \) does the following:

- Run \( M' \) until it outputs \( y \sharp \) on its output tape for some \( y \geq x \) (ordered lexicographically)
(before the $y$ is a $\uparrow$ or the left edge of the tape)

If $x = y$ then accept; if $x < y$ then reject.

Since $L(M')$ is infinite, $M$ always terminates, and clearly it decides $L$. (What goes wrong if we do the second part of the construction when $L$ is finite?)