CS 81 Assignment 9 for Wed., April 13 Solutions

1. **Show that the following functions are primitive-recursive:**

   a. **gt(x, y) = 1 if x > y ; 0 otherwise**
   b. **lte(x, y) = 1 if x ≤ y ; 0 otherwise**

   a. gt(x, y) = sgn(sub(x, y))
   b. lte(x, y) = not(gt(x, y))

   where **not**, **sgn**, and **sub** are primitive-recursive, as defined in the notes.

2. **Suppose that g is a primitive-recursive function of two arguments. Show that the following are also primitive-recursive:**

   a. **s(n, x) = g(0, x) + g(1, x) + …. + g(n, x)**
   b. **m(b, x) = the least y ≤ b such that g(y, x) = 0 if there is such a y; b+1 otherwise**

   a. s(0, x) = g(0, x)
      s(n+1, x) = add(s(n, x), g(s(n), x))

   b. m(0, x) = ifthenelse(eq(g(0, x), 0), 0, 1)
      m(n+1, x) = ifthenelse(lte(m(n, x), n),
                  m(n, x),
                  ifthenelse(eq(g(s(n), x), 0), s(n), s(s(n))))

   where **add**, **ifthenelse**, **eq** are primitive-recursive, as defined in the notes, and **lte** is defined in problem 1.b.

3. **Show that the following functions are primitive-recursive:**

   a. **isPrime(x) = 1 if x has no divisors other than 1 and x; 0 otherwise**
   b. **prime(n) = the n^{th} prime number (p(0) = 2, p(1) = 3, p(2) = 5, …).**

   a. First define functions *or* and *and* by:

      or(x, y) = sgn(x + y)
      and(x, y) = not(or(not(x), not(y)))

   Then in the definition of m in 2.b, use as g:

      g(y, x) = or(eq(y, 1), gt(mod(x, y), 0))

      where the r.h.s. is equivalent to not(and(not(eq(y, 1)), eq(mod(x, y), 0))

So g(y, x) = 0 when either y ≠ 1 and y divides x evenly.
Thus, m(n, x) in 2.b will be the smallest number y ≤ n such that y divides x evenly, or is equal to n if not.

Then define

isPrime(n) = and(gt(n, 1), eq(n, m(n, n)))

That is, isPrime(n) is true if n > 1 and n is the smallest number ≤ n that divides it evenly.

b. Define by primitive recursion (analogous to m above)

primeIn(0, L) = 0

```
primeIn(B+1, L) = ifthenelse(gt(primeIn(B, L),0),
                             primeIn(B, L),
                             ifthenelse(gt(B+1, L),
                                        ifthenelse(is_prime(B+1),
                                                   (B+1),
                                                   0),
                                        0)
                              )
```

So primeIn(B, L) = the smallest prime p such that L < p ≤ B or 0 if there is no such prime.

Next define the function that gives the smallest prime p such that n < p < 2n:

primeAfter(n) = primeIn(n+n, n)

Finally, define by primitive recursion

prime(0) = 2;
prime(n+1) = primeAfter(prime(n))

4. Suppose that L and M are recursive languages (meaning languages accepted by Turing machines). Classify whether the following are true or false, justifying each answer. (Note that if the answer is false, you only need to give a counterexample).

a. L ∪ M is recursive.
b. L ∩ M is recursive.
c. L - M is recursive.

All statements are true. In each case, there are Turing machines R and S that accept languages L and M respectively. Construct a new machine that first reserves a copy of the input tape, then runs as R on the input, recording the answer, then runs as S on the original input.

In case a, the new machine accepts if either R or S accepted.
In case b, the new machine accepts if both R and S accepted.
In case c, the new machine accepts if R accepted but S did not.

5. **Suppose that L and M are recursively-enumerable languages (meaning languages recognized by Turing machines). Classify whether the following are true or false, justifying each answer. (Note that if the answer is false, you only need to give a counterexample).**

   a. **$L \cup M$ is recursively-enumerable.**
   b. **$L \cap M$ is recursively-enumerable.**
   c. **$L - M$ is recursively-enumerable.**

Case a: True. There is a Turing machine R that recognizes L and a Turing machine that recognizes M. Thus we can build a Turing machine T that recognizes $L \cup M$ as follows: With input x, T simulates a step of R then a stop of S, then another step of R, then another step of S, etc. until one of the simulations accepts then input. In this case, T accepts. If neither accepts, then T does not either.

Case b: True. This is similar to case a, except that we don't have to run the machines in parallel; we can run them sequentially, as both must accept in order for T to accept.

Case c: False. Let $M = \{ x \mid x$ does not describe a Turing machine or x describes a Turing machine that halts on a blank tape$\}$. Let L = the language of all strings over the designated alphabet. Then

$L - M = \{x \mid x$ describes a Turing machine that diverges on a blank tape$\}$

We know that there is no Turing machine that recognizes $L - M$, so $L - M$ is not recursively enumerable.

6. **State whether or not there is a decision procedure for each of the following questions about a language $L \subseteq \{a, b\}^*$ as the language recognized by a Turing machine.**
   a. **$L = \varnothing$.**
   b. **$L = \{a, b\}^*$.**
   c. **$\{aba\} \subseteq L$.**
   d. **$L \subseteq \{aba\}$.**
   e. **$L \not\subset \{aba\}$.**

f. L is recursive.
g. L is finite.

All of the questions are functional properties; they depend only on the language, not the characteristics of a particular machine. All are non-trivial. Therefore Rice's theorem is applicable: There is no decision procedure for any of the properties.