CS5311 Homework #2 (Spring 2001)

Due: Tuesday, February 19, 2002 **before class**

100 points

You can only discuss the meaning of a problem with each other. Any forms of collaboration are absolutely forbidden. I will forward any suspicious cases to the Dean of Students or Graduate School.

When you write your program, make sure you also supply an explanation of your program, preferably a general elaboration followed by an instruction-by-instruction discussion. Nonsense, or direct translations of instructions (*i.e.*, add 1 to X), is unacceptable.

1. Do the following problems:
   (a) [**5 points**] Show that function $f(x) = \lfloor \log_2(x+1) \rfloor$ is primitive recursive.
   (b) [**5 points**] Define predicate $R(x, y)$ as follows:

$$R(x,y) = \begin{cases} 1 & \text{if } x \text{ and } y \text{ are relatively prime} \\ 0 & \text{otherwise} \end{cases}$$

   Two numbers x and y are said to be *relatively prime* if they have no common factor except 1. Show that $R(x, y)$ is primitive recursive *without* using quotient, remainder, and the GCD() function.
   (c) [**5 points**] Show that the least common multiple function LCM(x, y) is primitive recursive. The LCM of x and y is the smallest common multiple of x and y.
   (d) [**5 points**] Show that the greatest common factor function GCD(x, y) is primitive recursive using the fact that LCM(x, y) is primitive recursive. The GCD of x and y is the largest common factor of x and y.
2. [**10 points**] Let $\sigma(x)$ be the sum of the divisors of x if $x \neq 0$, and $\sigma(x) = 0$ if $x = 0$. For example, $\sigma(6) = 1 + 2 + 3 + 6 = 12$ because 1, 2, 3 and 6 are divisors of 6. Show that $\sigma(x)$ is primitive recursive.
3. Do the following problems:
   (a) [**15 points**] The *history function* of $f(n)$, denoted as $f^*(n)$, is defined as follows:

$$\begin{aligned} f^*(0) &= 1 \\ f^*(n+1) &= [f(0), f(1), \ldots, f(n)] \\ &= 2^{f(0)} 3^{f(1)} \cdots p_n^{f(n)} \\ &= \prod_{i=0}^{n} p_i^{f(i)} \end{aligned}$$

   Here, for the sake of convenience, we use $p_0 = 2$, $p_1 = 3$ and so on. Let $g(n)$ be a primitive recursive function such that $f(n) = g(f^*(n))$ holds. Show that $f^*(n)$ is primitive recursive, and then show that $f(n)$ is also primitive recursive.

(b) **[5 points]** The Fibonacci sequence is defined as follows:

$$f(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ f(n-1) + f(n-2) & \text{if } n \geq 2 \end{cases}$$

Show that function $f(n)$ is primitive recursive. (**Hint:** use the history function of $f(n)$)

(c) **[5 points]** The following is a proof showing that the above Fibonacci function $f(n)$ is primitive recursive:

> We prove the proposition with induction on n.
> **Base Case:** $n = 0, 1$**:** Since $f(0) = f(1) = 1$, obviously, $f(0)$ and $f(1)$ are primitive recursive.
> **Induction Step:** Suppose that $f(i)$ is primitive recursive for all $i \leq n$. Since $f(n)$ and $f(n-1)$ are primitive recursive by the induction hypothesis, and since $+$ and $-$ are also primitive recursive, $f(n+1) = f(n) + f(n-1)$ is primitive recursive. Therefore, $f(n)$ is primitive recursive.

Is this proof completely correct? If it is not completely correct, correct the mistakes to make this proof correct. If it is completely incorrect, point out the problem and elaborate why it is a problem.

4. **[15 points] (Simultaneous Recursion)** Suppose functions f_1, f_2, ..., f_k are defined as follows:

$$\begin{aligned}
f_1(x_1, x_2, \ldots, x_m, 0) &= h_1(x_1, x_2, \ldots, x_m) \\
f_2(x_1, x_2, \ldots, x_m, 0) &= h_2(x_1, x_2, \ldots, x_m) \\
&\vdots \\
f_k(x_1, x_2, \ldots, x_m, 0) &= h_k(x_1, x_2, \ldots, x_m) \\
f_1(x_1, x_2, \ldots, x_m, t+1) &= g_1(t, f_1(x_1, x_2, \ldots, x_m, t), f_2(x_1, x_2, \ldots, x_m, t), \ldots, f_k(x_1, x_2, \ldots, x_m, t), x_1, \ldots, x_m) \\
f_2(x_1, x_2, \ldots, x_m, t+1) &= g_2(t, f_1(x_1, x_2, \ldots, x_m, t), f_2(x_1, x_2, \ldots, x_m, t), \ldots, f_k(x_1, x_2, \ldots, x_m, t), x_1, \ldots, x_m) \\
&= \\
f_k(x_1, x_2, \ldots, x_m, t+1) &= g_k(t, f_1(x_1, x_2, \ldots, x_m, t), f_2(x_1, x_2, \ldots, x_m, t), \ldots, f_k(x_1, x_2, \ldots, x_m, t), x_1, \ldots, x_m)
\end{aligned}$$

where h_i's and g_j's are primitive recursive functions. Show that f_1, f_2, ..., f_k are all primitive recursive. (**Hint:** find and use a history function of all f_i's)

5. **[15 points]** Consider the following LOOP-END instruction:

LOOP (X)	X is a variable
<*instructions*>	execute <*instructions*> X times
END	

It executes in the following way:

- Variable X is the *loop variable*. It can be any variable (*i.e.*, input, output or local)
- The *scope* of the LOOP-END instruction includes <*instructions*> and END.
- The value of the *loop variable* specifies the number of iterations the scope of a LOOP-END executes.
- However, in the course of executing a LOOP-END, the value of the *loop variable* is not changed. More precisely, the value of a loop variable can only be changed by the instructions in <*instructions*>. The following loop doubles the value of local variable Z_2:

  LOOP (Z_2)
  $\quad Z_2 \leftarrow Z_2 + 1$
  END

- The LOOP instruction can be used in a nested way. The following nested loop computes the product of X_1 and X_2:

```
LOOP (X1)
   LOOP (X2)
      Y ← Y + 1     // add 1 to Y X1 × X2 times
   END
END
```

A program is called a *loop* program if and only if it is a syntactically correct program that only uses the four instructions: $V \leftarrow V$, $V \leftarrow V+1$, $V \leftarrow V-1$ and LOOP-END.

Show that a function is primitive recursive if and only if it is computed by a loop program. Thus, this proposition completely characterizes what primitive recursion can do. (**Hint:** Somewhere in your proof, you may need to use the simultaneous recursion theorem above).

6. **[10 points]** We shall show a *perfect debugger*, a program that can guarantee a program is bug-free, never exists. Define predicate DEBUG() as follows:

$$\text{DEBUG}(p, x) = \begin{cases} 1 & \text{if program } p \text{ running on input } x \text{ has a bug} \\ 0 & \text{otherwise} \end{cases}$$

Show that DEBUG() is not computable, and, hence, no perfect debugger is possible.

7. **[5 points]** A function f is a constant c if and only if $f(x) = c$ for all $x \in \mathcal{N}$. The following is a proof showing that testing if a function is a constant function is not computable.

   - Define a predicate CONST(p) as follows:

   $$\text{CONST}(p) = \begin{cases} 1 & \text{program } p \text{ always returns } c, \text{ a constant} \\ 0 & \end{cases}$$

   - Construct a program as follows:

   $$\begin{array}{ll} & Y \leftarrow c+1 \\ [A] & \text{IF CONST}(X) \text{ GOTO } [E] \\ & Y \leftarrow c \end{array}$$

   Let the program number of the above program by y.

   - From the above definition of CONST(), we have

   $$\begin{array}{rcl} \text{CONST}(y) & \Leftrightarrow & y \text{ is a constant function} \\ & \Leftrightarrow & Y \leftarrow c \text{ always executes} \\ & \Leftrightarrow & \text{CONST}(x) \text{ is false for } \textit{all } x \end{array}$$

   Thus, substituting y for x, the following is established:

   $$\text{CONST}(y) \Leftrightarrow \neg\text{CONST}(y)$$

   - Therefore, we have a contradiction and CONST() is not computable.

Is this proof completely correct? If it is not completely correct, correct the mistakes to make this proof correct. If it is completely incorrect, point out the problem and elaborate why it is a problem.

Print this page and attach it as the last page of your submission

Grade Report

Problem		*Assigned*	*Received*
1	a	5	
	b	5	
	c	5	
	d	5	
2		10	
3	a	15	
	b	5	
	c	5	
4		15	
5		15	
6		10	
7		5	
Total		100	