Assignment 3: Numbering and Universal Programs: Solutions

1. The Fibonacci sequence is given by

   - $F(0) = 0$,
   - $F(1) = 1$,
   - $F(n+2) = F(n+1) + F(n)$.

   Show that F is primitive recursive. (Hint: use the pairing function).

   We cannot directly give recursion equations for F, since it has two base cases, and is recursively defined by two of its previous terms, rather than just one. Instead, we define a new sequence $K(n)$ by

   $$K(n) =< F(n), F(n+1) >,$$

   and show that K is primitive recursive. Since $F(n) = l(K(n))$, and l is a primitive recursive function, this will show that F is primitive recursive.

   We will show that K is primitive recursive by giving recursion equations for it:

   $$K(0) =< 0, 1 >$$

   and

   $$\begin{aligned} K(t+1) &= < F(t+1), F(t+2) > \\ &= < r(K(t)), F(t+1) + F(t) > \\ &= < r(K(t)), r(K(t)) + l(K(t)) > . \end{aligned}$$

   Since r, l, and $+$ are primitive recursive, this shows that K is primitive recursive. Then, since $F(n) = l(K(n))$, F is primitive recursive as well.

2. A function f is given by "unnested double recursion" if there are functions g_1, g_2, h such that

   - $f(0, y) = g_1(y)$,
   - $f(x+1, 0) = g_2(x)$,
   - $f(x+1, y+1) = h(x, y, f(x, y+1), f(x+1, y))$.

   Show that if g_1, g_2, and h are all in some PRC class $\mathbf{C}$, then so is f. (Hint: use the functions $[a_1, \cdots a_n]$).

   Again, since f is defined by "double recursion", we cannot show that it is primitive recursive directly. Instead, we will define a new function,

show that this is primitive recursive, and use this to prove f is primitive recusive. We define K by

$$K(n) = [l_0 \cdots l_n]$$

where

$$l_i = [f(i,0), f(i,1), \cdots f(i, n-i)]$$

Essentially, $K(n)$ contains all of the values $f(a,b)$ for which $a+b \leq n$. Note that $f(m,n) = (K(m+n)_m)_n$, so if we can prove that K belongs to the class $\mathbf{C}$, then so will f.

So, we need to show that K belongs to $\mathbf{C}$. We will show that it is given by recursion equations of functions in $\mathbf{C}$. We have:

$$K(0) = [[f(0,0)]] = [[g_1(0)]]$$

and:

$$K(t+1) = [l_0, \cdots l_{t+1}]$$

but each of the l_i's can be found in terms of $K(t)$ and the g_i's. Indeed,

$$l_0 = [g_1(0), g_1(1), \cdots g_1(n+1)]$$

and for $i > 0$,

$$l_i = [f(i,0), f(i,1), \cdots f(i, n-i)]$$

but $f(i,0) = g_2(i-1)$ and for $1 \leq k \leq n-i$,

$$f(i,k) = h(i-1, k-1, (K(n)_{i-1})_k, (K(n)_i)_{k-1})).$$

Thus, K has a recursion equation given by functions of g_1, g_2, and h. Thus, since each of these are in $\mathbf{C}$, so is K.

Thus, since $f(m,n) = (K(m+n)_m)_n$, K is in $\mathbf{C}$, and the subscript functions are primitive recursive, f is in $\mathbf{C}$, as required.

3. Suppose the number of the program P is $(2^{46})(3^0)(5^2)(7^{37})-1$. Write out the code for P, and determine what it returns if given the input X_1.

By the definition of the number of the program, the above number tells us that the program has four lines, with instructions corresponding to the numbers 46, 0, 2, and 37.

For 46, the largest number x such that 2^x divides $46+1$ is 0. The y such that $2y+1 = 47$ is 23. Thus, $46 =< 0, 23 >$. The largest x such that 2^x divides $23+1$ is 3 (8 divides 24). Then the y such that $2y+1 = 3$ is 1. Thus, $46 =< 0, < 3, 1 >>$. The 0 tells us the instruction is unlabelled, the

1 that the variable is X_1, and the 3 says that the instruction is `IF` $X_1 \neq 0$ `GOTO` L_1.

An instruction with number 0 is simply $Y \leftarrow Y$.

For 2, the largest x such that 2^x divides $2 + 1$ is 0. The y such that $2y + 1 = 3$ is 1. Thus, $2 =< 0, 1 >$. The largest x such that 2^x divides $1 + 1$ is 1, and the y such that $2y + 1 = 0$ is 0. Thus $2 =< 0, < 1, 0 >>$. The instruction is unlabelled, the variable is Y, and the instruction is an addition: $Y \leftarrow Y + 1$.

For 37, the largest x such that 2^x divides $37 + 1$ is 1. The y such that $2y + 1 = 38/2 = 19$ is 9. Thus $37 =< 1, 9 >$. The largest x such that 2^x divides $9 + 1$ is 1. The y such that $2y + 1 = 10/2 = 5$ is 2. Thus $37 =< 1, < 1, 2 >>$. Thus, the instruction is labelled with L_1, and the instruction itself is to increment the number 2 variable, which is Z_1. Thus, the instruction is $Z_1 \leftarrow Z_1 + 1$.

In total, the program is:
`IF` $X_1 \neq 0$ `GOTO` L_1
$Y \leftarrow Y$
$Y \leftarrow Y + 1$
`(`L_1`)` $Z_1 \leftarrow Z_1 + 1$

Thus, the program returns 1 if $X_1 = 0$, and 0 if $X_1 \neq 0$.

4. Suppose we define a predicate $H(x)$, which is true exactly when the program with number $r(x)$ halts on input $l(x)$. Show that H is not computable.

   Suppose for contradiction that H was computable. By definition of the HALT predicate, $\text{HALT}(x, y) = H(< x, y >)$. Since $< x, y >$ is primitive recursive and H is computable, this would imply that HALT is also computable. But we have shown in class that it is not computable. Contradiction: thus H is not computable.

5. Suppose that $f(x_1, \cdots x_n)$ is computable by some program P. Suppose we also know that there is some primitive recursive function $g(x_1, \cdots x_n)$ such that
   $$\text{STP}^{(n)}(x_1, \cdots x_n, \#(P), g(x_1, \cdots x_n))$$
   is always true. Show that f is primitive recursive. (In other words, if the amount of time the program takes to run is bounded by some primitive recursive function, then the original function is primitive recursive).

By the normal form theorem, there is a primitive recursive predicate R such that

$$f(x_1 \cdots x_n) = l(\min_z R(x_1 \cdots x_n, z))$$

However, if we know that the program always terminates by $g(x_1 \cdots x_n)$ steps, then we can replace the unbounded minimization in the above by the bounded minimization

$$f(x_1 \cdots x_n) = l(\min(z \leq g(x_1 \cdots x_n))R(x_1 \cdots x_n, z)$$

Since l, g, and R are all primitive recursive and the minimization is bounded, we thus have that f is primitive recursive, as required.