Assignment 2:
More Programs and Primitive Recursive Functions: Solutions

1. Let $P(x)$ be a computable predicate. If $f$ is defined by

$$f(x_1, x_2) = \begin{cases} 
  x_1 + x_2 & \text{if } P(x_1 + x_2); \\
  \uparrow & \text{otherwise.}
\end{cases}$$

show that $f$ is partially computable.

The following program computes $f$:

\begin{verbatim}
Y ← X_1 + X_2
(A) IF P(Y) GOTO E
GOTO A
(E)
\end{verbatim}

Note that since $P$ is computable, the second line is a valid macro. The program first sets $Y$ to $X_1 + X_2$. If $P$ sends this value to 1, then the program exits. Otherwise, it gets into an infinite loop, and is thus undefined otherwise. Thus, the program partially computes $f$.

2. For any isomorphism $f : X \rightarrow Y$, one can define an inverse function $f^{-1} : X \rightarrow Y$, where $f^{-1}(x)$ is the unique number $y$ such that $f(y) = x$. Suppose that $f : \mathbb{N} \rightarrow \mathbb{N}$ is an isomorphism and computable. Prove that $f^{-1}$ is also computable.

To compute $f^{-1}$, we do the following. We begin with $y = 0$, and check if $f(y) = x$. If this is the case, we exit, with $y = 0$. Otherwise, we increment $y$, and check again if $f(y) = x$. We continue to do this until we find a value $y$ such that $f(y) = x$. Such a value must exist (and be unique) since $f$ is an isomorphism. We make use of the computable predicate $Z_1 = X$, and the computable function $f$ in the program:

\begin{verbatim}
(A) Z_1 ← f(Y)
IF Z_1 = X_1 GOTO E
Y ← Y + 1
GOTO A
(E)
\end{verbatim}

3. The language $\mathcal{P}$ has only three instructions: increment, decrement, and loop if a variable is non-zero. But there are other reasonable choices for a simple programming language. Another choice is a language $\mathcal{P}'$ which has variables and labels just like $\mathcal{P}$, but has these three instructions:
\[ V \leftarrow V' \]
\[ V \leftarrow V + 1 \]
\[ \text{IF } V \neq V' \text{ GOTO L} \]

(Where \( V, V' \) are variables, and \( L \) is a label). Show that \( P' \) is equivalent to \( P \), in the sense that a function \( f \) is partially computable in \( P \) if and only if it is partially computable in \( P' \).

First, we show that every function partially computable by \( P' \) is partially computable by \( P \). To do this, we must give macros in \( P \) for each of the three instructions in the language \( P' \). Then, if \( f \) is partially computable by some program \( P' \) in \( P' \), we simply use the same program in \( P \), replacing each basic instruction of \( P' \) with a macro in \( P \). We already have macros in \( P \) for the first two instructions in \( P' \). For the third instruction, we can make such a macro provided we can show that the predicate \( X_1 \neq X_2 \) is computable. But this predicate is simply \( \alpha(X_1 = X_2) \). Both these predicates are PR; thus \( X_1 \neq X_2 \) is as well, and thus also computable. Thus, any instruction in \( P' \) corresponds to a macro in \( P \), and so any function partially computable by \( P' \) is partially computable by \( P \).

To complete the proof, we must show the converse, that any function partially computable in \( P \) is partially computable in \( P' \). Thus, we must make macros in \( P' \) for each of the three instructions in \( P \). The first basic instruction in \( P \) (\( V \leftarrow V + 1 \)) is the same as the second basic instruction in \( P' \), so that is done.

The second basic instruction in \( P \) is \( V \leftarrow V - 1 \). To get a macro for this in \( P' \), we must increment \( Y \) until it is 1 less than \( X_1 \). To do this, however, it will be helpful to have the GOTO macro in \( P' \). The following code in \( P' \) has the effect of GOTO L:
\[ Z_m \leftarrow Z_m + 1 \]
\[ \text{IF } Z_m \neq Z_{m+1} \text{ GOTO L} \]

(Where \( Z_m \) and \( Z_{m+1} \) are variables that do not exist in the main program).

The following code decreases \( Y \) until it is one less than \( X_1 \); with the variable \( Z_1 \) keeping track of when \( Y + 1 \) is equal to \( X_1 \):
\[ \text{IF } Y \neq X_1 \text{ GOTO A} \]
\[ \text{GOTO E} \]
\[ (A) \ Z_1 \leftarrow Z_1 + 1 \]
\[ \text{IF } Z_1 \neq X_1 \text{ GOTO B} \]
\[ \text{GOTO E} \]
\[ (B) \ Y \leftarrow Y + 1 \]
\[ \text{GOTO A} \]
\[ (E) \]

Thus, this gives a macro for \( V \leftarrow V - 1 \).
The third basic instruction, \textbf{IF} \( V \neq 0 \) \textbf{GOTO} \( L \) in \( \mathcal{P} \) is given by the following macro in \( \mathcal{P}' \): \textbf{IF} \( V \neq Z_{m} \) \textbf{GOTO} \( L \), where \( Z_{m} \) is a variable not in use in the rest of the program. Since it is not in use, it is initialized to 0, and so the macro does compute \textbf{IF} \( V \neq 0 \) \textbf{GOTO} \( L \) as required.

Thus, any function partially computable in \( \mathcal{P} \) is partially computable in \( \mathcal{P}' \), and so the two languages are equivalent, as required.

4. (a) For any \( n \), prove that the function \( f(x) = x^{n} \) is primitive recursive.

(b) Using part (a) and induction, prove that any polynomial function

\[
f(x) = a_{n}x^{n} + a_{n-1}x^{n-1} + \cdots + a_{1}x + a_{0}
\]

is primitive recursive (each \( a_{i} \) is a natural number).

(a) First, for any number \( x \), the function \( y \mapsto x \cdot y \) is primitive recursive. The function \( x^{n} \) is given by composing this function with itself \( n \) times, and is thus itself primitive recursive.

(b) The proof is by induction on the degree of the polynomial, \( n \). In the base case, \( f(x) = a_{0} \) for some natural number \( a_{0} \). We saw in class that this function is PR.

Now, assume the induction hypothesis: any polynomial \( g(x) = a_{n}x^{n} + a_{n-1}x^{n-1} + \cdots + a_{1}x + a_{0} \) is PR. Let \( f(x) = a_{n+1}x^{n+1} + a_{n}x^{n} + \cdots + a_{1}x + a_{0} \) be a polynomial of degree \( n+1 \). We must show it is PR as well. But \( f(x) = a_{n+1}x^{n+1} + g(x) \). We know that multiplication, addition, and \( g(x) \) are all PR; thus, \( f \) is as well. Thus, by induction, any polynomial is PR.

5. Let \( \pi(x) \) be the number of primes less than or equal to \( x \). Show that \( \pi(x) \) is computable.

First, we will show that \( \pi(x) \) is a primitive recursive function. The number of primes less than or equal to \( x + 1 \) is given by the number of primes less than or equal \( x \), plus 1 if \( x + 1 \) is a prime. Thus, \( \pi(x) \) is given by the recursion equations

\[
\pi(0) = 0, \quad \pi(t + 1) = \pi(t) + \text{prime}(t + 1)
\]

Since + and prime are primitive recursive, so is \( \pi \). Finally, since any primitive recursive function is computable, \( \pi \) is computable.