Problem 1:

a)

We use a variation of diagonalization. Let

\[ g(x) = \begin{cases} 
    x, & \text{if } f_x(x) \text{ is undefined} \\
    \text{undefined}, & \text{if } f_x(x) \text{ is defined}
\end{cases} \]

where \( a \) is some constant.

By construction, \( g(x) \) is defined whenever \( f_x(x) \) is undefined, and \( g(x) \) is undefined whenever \( f_x(x) \) is defined. Thus, for any \( i \in \mathbb{N} \), we have that either \( i \notin \text{Dom}(f_i) \) and \( i \notin \text{Dom}(g) \), or \( i \notin \text{Dom}(f_i) \) and \( i \in \text{Dom}(g) \). Therefore, \( \text{Dom}(g) \neq \text{Dom}(f_i) \) for each \( i \), as desired.
b) We can define \( g(x) \) as follows:

\[
g(x) = \begin{cases} 
  f(x), & \text{if } x \leq m \\
  \phi(x) + 1, & \text{if } x > m \text{ and } \phi(x) \text{ is defined} \\
  a, & \text{if } x > m \text{ and } \phi(x) \text{ is undefined}
\end{cases}
\]

\( a \) is some constant.

First, we note that it satisfies the condition that

\[
g(x) \approx f(x) \quad \text{for } x \leq m.
\]

This is the first part of the piecewise definition. The rest of the definition ensures that \( g \) is non-computable. In particular, \( g \) is constructed so as to differ from every computable function \( \phi_k \) at some \( x \) value greater than \( m+1 \) (diagonalization starting from \( x = m+1 \)). Thus,

\[
\begin{align*}
g(m+1) &\neq \phi_0(m+1) \\
g(m+2) &\neq \phi_1(m+2) \\
g(m+3) &\neq \phi_2(m+3)
\end{align*}
\]
and so on for all $\phi_k$. In the event that $\phi_k$ is undefined at the corresponding $x$ value, $g$ takes the value of the constant $c$, so $g$ is defined at that $x$ value and therefore differs again from $\phi_k$ as desired. Since $g$ differs from all computable functions, it is non-computable, which satisfies the necessary requirement.
a) Define the function

\[ f(e, y) = 1 - \Phi_e(y) = 1 - \Phi_{\mu}(e, y) \]

Then \( f \) is computable by composition and the fact that \( \Phi_{\mu} \) is computable. Thus, by the \( S-M-N \) theorem, there exists a total computable function \( k(e) \) such that

\[ g(e, y) = \Phi_{k(e)}(y) \]

Note that if \( \Phi_e \) is the characteristic function of a decidable predicate \( M \), then

\[ \Phi_{k(e)} = 1 - \Phi_e \] is the characteristic function of "Not \( M \)" as desired.

QED

b) Define the function

\[ f(x, y, t) = \begin{cases} \Phi_x(t/2) & \text{if } t \text{ is even} \\ \Phi_y((t-1)/2) & \text{if } t \text{ is odd} \end{cases} \]

Since "even" and "odd" are decidable predicates and \( \Phi_x, \Phi_y \) and the basic arithmetic operators are computable we have that \( f \) is computable as well.

Thus, by the \( S-M-N \) theorem, there is a total computable function \( s(x, y) \) such that

\[ \Rightarrow \]
\[ \Phi_s(x, y)(t) = \begin{cases} \Phi_x(t \cdot \frac{1}{2}) & \text{if even} \\ \Phi_y(t \cdot \frac{1}{2}) & \text{if odd} \end{cases} \]

Since \( t \cdot \frac{1}{2} \) takes on all values in \( \mathbb{N} \) for even \( t \in \mathbb{N} \) and \( t \cdot \frac{1}{2} \) takes on all values in \( \mathbb{N} \) for odd \( t \in \mathbb{N} \), we have that

\[ E_s(x, y) = E_x \cup E_y. \]

Clearly \( E_s(x, y) \) only outputs values in the ranges of \( \Phi_x \) and \( \Phi_y \), so \( E_s(x, y) \subseteq E_x \cup E_y \)

as well. Thus

\[ E_s(x, y) = E_x \cup E_y, \quad \checkmark \]

as desired. \quad \text{QED}
3 (a) Suppose "\( W_x = W_y \)" were decidable. This means that its characteristic function \( f(x, y) \) is computable. Let \( n \) be the index of \( Z(1) \) which give output zero on all output (and is thus total). By the \( S = M - n \) theorem \( f_n(x) = f(x, n) \) is computable. But \( f(x, n) \) is the characteristic function for "\( W_x = \emptyset \)" or "\( \Phi_x \) is total." Because "\( \Phi_x \) is total" is not decidable (Thm 5.2.1 in Cutland) we have a contradiction and conclude "\( W_x = W_y \)" is not decidable.

(b) We can use Rice's Theorem for this problem. Let \( f_\emptyset \) be undefined for all inputs and \( B = \{ f_\emptyset \} \). \( B \) satisfies the hypotheses for Rice's theorem because it is not empty and \( f_\emptyset \) is computable. We can thus conclude that "\( \Phi_x \in B \)" i.e. "\( \Phi_x = f_\emptyset \)" is not decidable. "\( W_x = \emptyset \)" is the same problem as "\( \Phi_x = f_\emptyset \)" since only one function, \( f_\emptyset \), is undefined for all input, so we have proven "\( W_x = \emptyset \)" is undecidable.
Define

\[ B(n) = \text{maximum output of any URM program with at most } n \text{ many instructions, on input } 0. \]

Note that \( B \) is a total function. The proof of this will follow from the solution to question (4). Note that the purported “proof” included in the pset assignment sheet is wrong. In particular, there are \textbf{infinitely} many program with at most \( n \) many instructions since each instruction is determined by its type (one of four possible) \textit{and} the registers it mentions (infinitely many possible).

4. Prove that if the halting problem were solvable then \( B \) would be computable.

---

**Claim:** To compute \( B(n) \), we need only consider the output of finitely many URM programs of length at most \( n \) on input 0.

Proof: Notice that any program with at most \( n \) many instructions can mention at most \( 2n \) many registers. Why? Instructions of the form \( S(m) \), \( Z(m) \) mention exactly one register and instructions of the form \( T(m, m'), J(m, m', q) \) mention at most two registers. Moreover, registers may be duplicated across instructions. So, with at most \( n \) instructions, we can mention at most \( 2n \) registers.

However, notice that many such programs compute the same function. For example, the programs

\[
\begin{align*}
P : & \text{ Z(3) J(3, 6, 5) S(3) S(1)} \\
Q : & \text{ Z(42) J(42, 2, 5) S(42) S(1)}
\end{align*}
\]

are equivalent up to a relabelling of the “counter” registers (i.e. all \( R_m \) with \( m > 1 \)). Using our first observation, we see that for each URM program of length \( i \) there is an equivalent URM program of length \( k \) which mentions only registers labeled \( 1, \ldots, 2i \).

Recall from our early discussion of URM program that any program can be put into standard form. Hence, the label of the destination instruction in any jump instruction in a program of length \( i \) can be assumed to be at most \( i + 1 \).

Thus, we have reduced the problem of considering the outputs of all URM programs of length at most \( n \) on input 0 to the problem of considering only the outputs of URM programs

- of length at most \( n \),
- in standard form,
- and mentioning only registers in \([1, 2i]\) (where \( i \) is the length of the program).

There are finitely many such URM programs, so the claim is proved.
We now use the claim to prove that if the halting problem were solvable then \( B \) would be computable. Assume that the halting problem is solvable. We give an algorithm that, for each \( n \), computes \( B(n) \). A counting argument based on the claim above shows that there are

\[
C_n = \sum_{i=0}^{n} (2i + 2i + (2i)^2 + (2i)^2(i + 1))^i = \sum_{i=0}^{n} (4i^3 + 8i^2 + 4i)^i
\]

programs in standard form mentioning only the appropriate registers. We start with \( P_0 = \gamma^{-1}(0) \) and continue on to successive URM programs (ordered by their Gödel numbers), proceeding as follows at stage \( k \):

- Determine if \( P_k \) is in standard form and mentions only registers in \([1,2i_k]\) where \( i_k \) is the length of \( P_k \). Note that these are decidable questions.
- If not, skip to \( P_{k+1} \).
- If yes, increment a counter counting the number of programs fitting the criteria which have been examined.
- Use the decision procedure for the halting problem to ask whether \( P_k(0) \downarrow \).
- If \( P_k(0) \uparrow \), skip to \( P_{k+1} \).
- If \( P_k(0) \downarrow \), compute \( P_k(0) \) and record its output in a list of outputs under consideration.
- If the counter for programs meeting the criteria has not yet reached \( C_n \), loop back to first step, now looking at \( P_{k+1} \).
- If the counter has reached \( C_n \), return the maximum value in the list of outputs. This is \( B(n) \).

By Church’s thesis, this informal description of an algorithm computing \( B(n) \) is sufficient to prove that the function \( B \) is computable.
(a) Prove that $B$ is a strictly increasing function (for all $n, B(n) < B(n + 1)$).

Proof. Let $P$ be a URM program with $n$ or less instructions such that $P(0) \downarrow B(n)$. Then if $P$ is not in standard form, convert $P$ into a URM program $P_s$ in standard form that is equivalent to $P$. This process does not increase the number of instructions, since only the destinations of the Jump instructions are changed (cf. p.26 of Cutland). Then, construct a URM program $P'$ by adding $S(1)$ at the end of $P_s$ so that $P'$ gives $B(n) + 1$ on 0. Thus, $B(n + 1) \geq B(n) + 1 > B(n)$. \hfill \square

(b) Prove that for all $n \geq 1, B(n + 5) \geq 2n$. (Hint: How many instructions does it take to double the contexts of a register?)

Proof. The following program $P$ doubles the input:

$I_1 : T(1, 2) \quad I_2 : J(2, 3, 6) \quad I_3 : S(1) \quad I_4 : S(3) \quad I_5 : J(1, 1, 2)$

For any $n \geq 1$, there is a program $P'$ with $n + 5$ instructions that outputs $2n$, namely, $P'$ performs $S(1)$ $n$ times and then runs $P$. Specifically, $P'$ consists of the following instructions: for $i \leq n, I_i : S(1)$, and $I_{n+1} : T(1, 2), I_{n+2} : J(2, 3, n + 6), I_{n+3} : S(1), I_{n+4} : S(3), I_{n+5} : J(1, 1, n + 2)$. Then, $B(n + 5)$ is guaranteed to be at least as big as $2n$.

Therefore, for all $n \geq 1, B(n + 5) \geq 2n$. \hfill \square
(c) Recall the definition of one function dominating another from PS 2. Also, recall the result from that problem set: any URM computable function \( f \) is dominated by a strictly increasing URM computable function \( g \).

- Note that if \( g(n) \) is defined but \( f(n) \) is not, we say that \( f(n) \preceq g(n) \).

Therefore, prove that for any URM computable function \( f \), \( B \) dominates \( f(n) + 1 \).

(Hint: given an increasing URM computable function \( g \), find a program that witnesses that \( B(n + k_0) > g(B(n)) \) for some number \( k_0 \).)

Proof. Let \( f(x) \) be a computable function. \( g(x) \) that dominates \( f(x) \) is defined by primitive recursion as follows:

\[
g(x) = \begin{cases} 
  f(x) + 1 & \text{if } x = 0 \\
  \max(g(y), f(x)) + 1 & \text{if } x = y + 1
\end{cases}
\]

It is clear that \( g(x) \) is computable.

Let \( P_g \) be a URM program in standard form that computes \( g \). Construct \( P'_g \) by adding \( S(1) \) at the end.

Then take a program \( P_n \) in standard form with at most \( n \) instructions and computes \( B(n) \) on 0. Construct \( P'_n \) by putting \( Z(2), \ldots, Z(\rho(P_n)) \) at the end.

Now construct the program \( Q = P'_n P'_g \). Let \( k_0 \) be the sum of the length of \( P'_g \) and \( \rho(P_n) - 1 \) (= the sum of the length of \( P_g \) and \( \rho(P_n) \)). Then, \( Q \) has at most \( n + k_0 \) instructions, and when run on 0, it outputs \( g(B(n)) + 1 \). Thus, \( B(n + k_0) > g(B(n)) \).

Because \( B \) and \( g \) are both strictly increasing,

\[
B(n + k_0 + 6) > B(n + k_0 + 5) \geq g(B(n + 5)) \geq g(2n) \geq g(n + k_0 + 6)
\]

Thus, for \( m > k_0 + 6 \), \( B(m) > g(m) \).

Therefore, for any computable function \( f \), there is \( k_0 \) such that for \( m > k_0 + 6 \), \( B(m) > g(m) > f(m) \), and hence \( B(n) \) strictly dominates \( f(n) \).

(d) Finally, prove that \( B \) is not computable.

Proof. Since \( B \) strictly dominates all computable functions, it differs from all of them. Therefore, \( B \) is not computable.