

# Kleene's Second Recursion Theorem and Self-Knowing Programs

Last Updated November 1st, 2023

## Kleene's Second Recursion Theorem

*"Know Thyself"*

---

Socrates

Consider a computable function  $f(x, y)$ , where  $x$  is viewed as a Gödel number of some program and  $y$  is some other input. The following are some statements that could be made in an informal program that computes  $f$ .

- Print the instructions of  $P_x$ .
- Simulate the computation of  $P_x$  on input  $y$ .
- Count the number of Jump instructions that are executed in the computation of  $P_x$  on input  $y$ .
- Send program  $x$  and  $y$  to a server on the internet.
- Return the tuple of configurations that constitutes the computation of  $P_x$  on input  $y$ .

Now suppose we take  $f$ 's program statements and re-write them in a self-referencing way, to where we get statements like the following ones.

- Print  $my$  instructions.
- Simulate  $myself$  on input  $y$ .
- Count the number of Jump instructions that I execute when I'm computing input  $y$ .
- Send  $myself$  and  $y$  to a server on the internet.
- Return the tuple of configurations that constitutes  $my$  computation on input  $y$ .

A program that makes one or more references to its own Gödel number is said to be **self-knowing** or **self-referencing**.

## Catch-22 for a Program $P$ Attempting to Know Itself

1. For  $P$  to know its Gödel number, it must know each of its instructions.
2. Some instructions, such as “print myself”, requires  $P$  to know its Gödel number.

## Proposed Solution to Catch-22

1. Assume for the sake of argument that, after replacing statements about  $x$  with statements about itself, that there does in fact exist a program  $P_e$  with Gödel number  $e$  that computes the resulting function.
2. Then  $P_e$  is a function of the single variable  $y$  (since variable  $x$  has been assigned constant  $e$ ).
3. Therefore, we have, for all  $y$ ,

$$\phi_e(y) = f(e, y).$$

In other words, there is a program  $P_e$  that, on input  $y$  computes  $f(e, y)$ , and thus makes references (to  $e$  that has been substituted for  $x$ ) to its own Gödel number.

4. Thus, we have reduced the problem to that of finding a Gödel number  $e$  that satisfies the above equation.
5. Stephen Kleene’s second recursion theorem states that such an  $e$  does exist!

**Kleene’s Second Recursion Theorem.** Let  $f(x, y)$  be a computable function that takes as input a Gödel number  $x$ , and some additional input  $y$ . Then there is a Gödel number  $e$  for which  $\phi_e(y) = f(e, y)$ .

**Example 1.** Consider the URM computable function  $f(x, y)$  which, on inputs  $x$  and  $y$ , simulates the computation  $P_x(y)$ , and returns the number of times that a jump instruction is executed during the computation  $P_x(y)$ . Then by the 2nd recursion theorem, there is a program  $P_e$  for which  $P_e(y) = f(e, y)$ , and so, for input  $y$ ,  $P_e$  computes the number of times that its own self executes a jump instruction during its computation with input  $y$ .  $\square$

**Proof of Kleene's Second Recursion Theorem.** The idea behind the proof is to divide the construction of the desired program  $P = ABC$  into three parts:  $A$ ,  $B$ , and  $C$  which we now describe. Assume that  $y$  is the input to  $P$ .

- Part A.
- Move  $y$  to register  $R_2$ .
  - Place  $B$ 's Gödel number  $b$  in  $R_1$ .

- Part B.
- Use  $b$  in  $R_1$  to compute  $A$ 's Gödel number  $a$ .
  - Compute  $C$ 's Gödel number  $c$ .
  - Compute

$$e = \gamma(\gamma^{-1}(a), \gamma^{-1}(b)\gamma^{-1}(c)) = \gamma(ABC) = \gamma(P),$$

the Gödel number of the concatenation of  $A$ 's,  $B$ 's, and  $C$ 's instructions.

- Place  $e$  in  $R_1$ , with  $y$  remaining in  $R_2$ .

Part C. Compute  $f(e, y)$ .

### Notes.

1. The most straightforward of the three is part  $C$ , since its sole purpose is to compute function  $f$  which is assumed URM computable, and so  $C$ 's instructions consist of the instructions of the URM program used to compute  $f$ .
2. The clever part of the above program is understanding how  $A$  is able to compute  $B$ 's Gödel number and vice versa. This is actually made possible by an elementary use of the s-m-n theorem.
3. Consider the function  $g(x, y) = x$ . By the s-m-n theorem, there is a total computable function  $k(x)$  for which

$$\phi_{k(x)}(y) = g(x, y) = x.$$

4. Then define  $A$ 's Gödel number to be equal to  $k(b)$ . This works because, on input  $y$ , program  $A$  outputs

$$\phi_{k(b)}(y) = b$$

in register  $R_1$ , and has the side effect of placing  $y$  in  $R_2$  via an initial  $T(1, 2)$  statement. Therefore  $A$  works in exactly the way it was described above.

Given that  $a = k(b)$  we may now describe  $B$ 's program as follows.

**Program  $B$**

Input Gödel number  $z$ .

Compute Gödel number  $k(z)$ .

Compute  $c = \gamma(C)$ .

Return

$$\gamma(\gamma^{-1}(k(z)), \gamma^{-1}(z), c).$$

**Important:** notice that  $B$ 's program does *not* depend on knowing  $A$ 's Gödel number  $a$ . If it did, then it would create a circularity error, since  $a = k(b)$  already depends on  $B$ 's Gödel number. However,  $B$  is able to compute  $a$  once it has its own Gödel number  $z = b$  since step 2 of its algorithm yields  $a = k(b)$ .

Thus, we see that, after the execution of  $A$  on input  $y$ ,  $B$  receives input  $z = b$  which gives  $a = k(z) = k(b)$ , and so  $B$  outputs into  $R_1$  the value

$$e = \gamma(\gamma^{-1}(a), \gamma^{-1}(b), \gamma^{-1}(c)) = \gamma(ABC) = \gamma(P).$$

The following diagram shows the results of all three programs combined in sequence, where  $v \xrightarrow{X} w$  means that program  $X$  inputs  $v$  and outputs  $w$ . Then we have

$$y \xrightarrow{A} (b, y) \xrightarrow{B} (e = \gamma(ABC), y) \xrightarrow{C} f(e = \gamma(ABC), y).$$

Therefore,  $P = ABC = P_e$  computes

$$\phi_e(y) = f(e, y),$$

and the proof is complete. □

**Example 2.** Program  $P$  is called **totally introspective** iff, on input  $y$ ,  $P$  returns a number that encodes every configuration of the computation of  $P$  on input  $y$ . Letting  $\sigma(x, y, i)$  denote the encoding of the  $i$  th configuration of the computation  $P_x(y)$ , then we define the computable function

$$f(x, y) = \begin{cases} \prod_{i \leq t} p_i^{\sigma(x, y, i)} & \text{if } P_x(y) \downarrow \text{ in } t \text{ steps} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Now, by the 2nd recursion theorem, there exists a Gödel number  $e$  for which  $\phi_e(y) = f(e, y)$ , meaning that  $P_e$  is totally introspective, since, on input  $y$ , it outputs an encoding of all the configurations used in the computation  $P_e(y)$ .  $\square$



# The self Programming Statement

The Recursion theorem gives rise to a tool that may be used when writing a program  $P$ . Namely, we may make reference to  $P$ 's Gödel number, which is represented with the keyword `self`. This keyword is similar to the `this` keyword of Java, which refers to the object that a Java method is acting on. For example, the following are valid programming statements for program  $P$ :

```
int f(int y)
{
    int length = instructions(self).length;

    print("Hi! I have Godel number equal to ");
    print(self);
    print(".\nI have ");
    print(length);
    print(" instructions.\nI will return the square of the output that results ");
    print("from simulating myself on input y.\n");
    print("What are the possibilities for the value of output?\n"); //Hint: there are 3
    int output = simulate(self,y); //Note: output = f(y), since I compute f(y)
    return output*output;
}
```

To justify such a program, suppose  $y \in \mathcal{N}$  is the input to  $P$ , and the purpose of  $P$  is to implement the unary computable function  $f(y)$ . Then we may do the following.

1. Transform  $P$  by adding another input  $x$ , so that we are now implementing function  $f(x, y)$ .
2. Replace each occurrence of `self` with  $x$ .

```
int f(int x, int y)
{
    int length = instructions(x).length;
    print("Hi! I have Godel number equal to ");
    print(x);
    print(".\nI have ");
    print(length);
    print(" instructions.\nI will return the square of the output that results ");
    print("from simulating myself on input y.\n");
    print("What are the possibilities for the value of output?\n");
    int output = simulate(x,y); //Note: output = f(y), since I compute f(y)
    return output*output;
}
```

3. Use the method described in the proof of Kleene's 2nd Recursion Theorem to compute an  $e$  for which  $P_e$  computes

$$\phi_e(y) = f(e, y).$$

4. Thus,  $P_e$  computes  $f(y)$ , with  $e$  substituted for  $x$ .
5. Therefore,  $P_e$ 's references to **self** are justified, since **self** =  $e$ , the Gödel number of the program that computes  $f(y)$ .

# Kleene's 2nd Recursion Theorem and Undecidability

The `self` programming construct that is made possible by Kleene's 2nd Recursion theorem may be readily used to prove the undecidability of most program properties, including the properties `Self Accept`, `Halting Problem`, `Total`, and `Zero` from the Undecidability and the Diagonalization Method lecture. The idea is outlined as follows.

1. Let  $A$  be a program property that we want to prove is undecidable.
2. Let  $f_A(x)$  denote  $A$ 's characteristic function.
3. Assume  $A$  is decidable in which case  $f_A(x)$  is total computable.
4. Consider the following program  $P$ .

Input  $y \in \mathcal{N}$ .

If  $f_A(\mathbf{self}) = 1$ ,  $//P$  has property  $A$ .

Return a value that implies  $P$  does *not* have property  $A$ .

Else  $//f_A(\mathbf{self}) = 0$  and thus  $P$  does not have property  $A$ .

Return a value that implies  $P$  *does* have property  $A$ .

5. Regardless of whether or not  $P$  has property  $A$ , a contradiction arises. Therefore, the assumption that  $A$  is decidable must be false.

**Example 3.** We prove that Halting Problem is undecidable.

**Solution.** Suppose Halting Problem is decidable, i.e.

$$H(x, y) = \begin{cases} 1 & \text{if } y \in W_x \\ 0 & \text{otherwise} \end{cases}$$

is total computable. Now consider the following program  $P$ .

Input  $y \in \mathcal{N}$ .

If  $H(\mathbf{self}, y) = 1$ , loop forever.

Return 1.

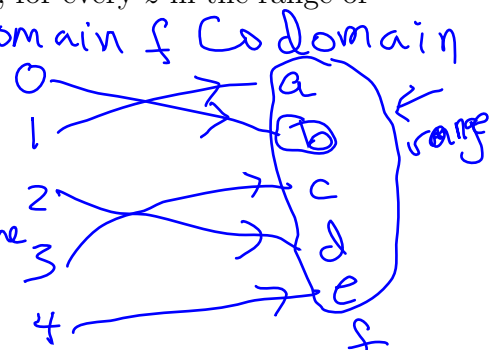
Let  $e = \mathbf{self}$  denote the Gödel number for  $P$ . Then  $P_e(e) = 1$  provided  $H(e, e) = 0$  iff  $P_e(e)$  does not halt, a contradiction. Similarly,  $P_e(e)$  does not halt provided  $H(e, e) = 1$  iff  $P_e(e)$  does halt, another contradiction. Therefore, the assumption that Halting Problem is decidable must be false.

**Example 4.** Prove that the Total decision problem is undecidable. Also, give examples of programs  $P_1$  and  $P_2$  for which  $g(\gamma(P_1)) = 1$  and  $g(\gamma(P_2)) = 0$ .

**Solution.**

**Example 4b.** An instance of the decision problem **One-to-One** is a Gödel number  $x$ , and the problem is to decide if function  $\phi_x$  is a one-to-one function, meaning that, for every  $z$  in the range of  $\phi_x$ , there is *exactly one*  $y$  for which  $\phi_x(y) = z$ . Consider the function

$$g(x) = \begin{cases} 1 & \text{if } \phi_x \text{ is one-to-one} \\ 0 & \text{otherwise} \end{cases}$$



Evaluate  $g(x)$  for each of the following Gödel number's  $x$ .

- $x = e_1$ , where  $e_1$  is the Gödel number of the program that computes the function  $\phi_{e_1}(y) = \text{sgn}(y)$ . Hint: recall that  $\text{sgn}(y)$  equals 1 if  $y > 0$ , and equals 0 otherwise.  $g(e_1) = 0$  since e.g.  $\text{sgn}(1) = 1$ ,  $\text{sgn}(2) = 1$
- $x = e_2$ , where  $e_2$  is the Gödel number of the program that computes the function  $\phi_{e_2}(y) = y^2$ .
- $x = e_3$ , where  $e_3$  is the Gödel number of the program that computes  $g(x)$  (assuming that  $g(x)$  is URM computable).

$$\phi_{e_2}(y) = y^2 \quad g(e_2) = 1$$

$$\text{range}(\phi_{e_2}) = \{0, 1, 4, 9, 16, \dots\}$$

$$g(e_3) = 0 \text{ since } \text{range}(g) = \{0, 1\} \text{ and there are an infinite number of one-to-one functions.}$$

Prove that  $g(x)$  is not URM computable. In other words, there is no URM program that, on input  $x$ , always halts and either outputs 1 or 0, depending on whether or not  $\phi_x$  is a one-to-one function. Do this by writing a program  $P$  that uses  $g$  and makes use of the **self** programming concept. Then show how  $P$  creates a contradiction.

Input  $y$

If  $g(\text{self}) = 1$ , return 1.

Else //  $g(\text{self}) = 0$

Return  $y$ .

Case 1:  $g(\text{self}) = 1$ . Then  $P(y) = 1$ , for all  $y$  inputs and  $P$  is not one-to-one a contradiction.  $P(0) = P(1) = 1 \Rightarrow$  not one-to-one

Case 2:  $g(\text{self}) = 0$ . Then  $P(y) = y$  is one-to-one since every output  $y$  has only  $y$  as input

## Other Applications of Kleene's 2nd Recursion Theorem

A subset  $A \subset \mathcal{N}$  of the natural numbers is said to be **recursively enumerable** iff there is a program that can print all the members of  $A$  in a (possibly infinite) list, in no particular order. Also, we say that decision problem  $A$  is recursively enumerable if the set of positive instances of  $A$  is recursively enumerable.

Note:  $A$  is recursively enumerable iff there is a total computable function  $f$  for which  $A = \text{range}(f)$ .

**Example.** Show that the set of even natural numbers is recursively enumerable.

**Theorem.** If decision problem  $A$  is decidable, then it is recursively enumerable.

**Proof.** Let  $f_A(x)$  denote  $A$ 's characteristic function. Since  $A$  is decidable there is a program  $P$  that halts on all inputs, and for which  $P(x) = f_A(x)$  for all  $x \in \mathcal{A}$ . Then the following program prints all the positive instances of  $A$ .

Input  $x \in \mathcal{N}$ .

For each  $i = 0, 1, \dots$

    Simulate  $P$  on input  $i$ .

    If  $P(i) = 1$ , then print  $i$ .



**Example.** Show that `Self Accept` is recursively enumerable.

**Theorem 3.** Consider the set  $M$ , where  $x \in M$  iff there is no  $y < x$  for which  $\phi_y = \phi_x$ . In other words,  $P_x$  is a minimal program for function  $\phi_x$ . Then  $M$  is not recursively enumerable.

**Proof of Theorem 3.** Suppose  $M$  is recursively enumerable. Then it is an exercise to show that there is a total computable unary function  $f$  whose range is equal to  $M$ . In other words  $M = \{f(i) | i \in \mathcal{N}\}$ . Consider the following program  $P$ .

Input  $x \in \mathcal{N}$ .

For each  $i = 0, 1, \dots$

    If  $f(i) > \mathbf{self}$ , then **break**.

    Simulate program  $P_{f(i)}$  on input  $x$ , and return  $y$  in case  $P_{f(i)}(x) \downarrow y$ .

Let  $e$  be the Gödel number of  $P$ . Then it follows that  $\phi_e = \phi_{f(i)}$ . But  $f(i) > e$  which contradicts the fact that  $f(i) \in M$ . Therefore, the assumption that  $M$  is r.e. must be false.

**Theorem 4.** Let  $f$  be a total computable unary function. Then there is a number  $n \in \mathcal{N}$  for which  $\phi_n = \phi_{f(n)}$ . We refer to  $n$  as a **fixed point** for  $f$ .

**Proof of Theorem 4.** Consider the following program  $P$ .

Input  $x \in \mathcal{N}$ .

Compute  $y = f(\mathbf{self})$ .

Simulate program  $P_y$  on input  $x$ , and return  $z$  in case  $P_y(x) \downarrow z$ .

Then

$$\phi_y = \phi_{f(\mathbf{self})} = \phi_{\mathbf{self}},$$

and so  $n = \mathbf{self}$  is a fixed point for  $f$ .

# Exercises

- With respect to Kleene's 2nd Recursion Theorem, prove that there are infinitely many values  $e$  for which  $\phi_e(y) = f(e, y)$ . Hint: consider program  $B$  in the proof of the theorem.
- Recall that a function  $f : \mathcal{N} \rightarrow \mathcal{N}$  is **onto** provided for every  $y \in \mathcal{N}$  there is an  $x \in \mathcal{N}$  for which  $f(x) = y$ . Consider the function

$$g(x) = \begin{cases} 1 & \text{if } \phi_x \text{ is onto} \\ 0 & \text{otherwise} \end{cases}$$

Domain  $f$  Codomain



Evaluate  $g(a)$ ,  $g(b)$ , and  $g(c)$ , where

(a)  $\phi_a(y) = y^2$

(b)  $\phi_b(y) = 1$

(c)  $\phi_c(y) = y$ .

$g(a) = 0$  since only perfect squares have a preimage  
 $\text{range}(\phi_b) = \{1\} \neq \mathbb{N}$   $g(b) = 0$   
 $g(c) = 1$  since  $\text{range}(\phi_c) = \mathbb{N} = \text{Domain of } \phi_c$

$f$  is onto since each codomain value has a preimage.

- Prove that the function

$$g(x) = \begin{cases} 1 & \text{if } \phi_x \text{ is onto} \\ 0 & \text{otherwise} \end{cases}$$

is not URM computable. In other words, there is no URM program that, on input  $x$ , always halts and either outputs 1 or 0 as output, depending on whether or not  $\phi_x$  is onto. Do this by writing a program  $P$  that uses  $g$  and makes use of the **self** programming concept.

- Recall that  $W_x$  denotes the domain of the function  $\phi_x(y)$ , i.e. the natural number inputs  $y$  to  $\phi_x$  for which  $\phi_x(y)$  is defined. Consider the function

$$g(x) = \begin{cases} 1 & \text{if } W_x = \emptyset \\ 0 & \text{otherwise} \end{cases}$$

Evaluate  $g(a)$ ,  $g(b)$ , and  $g(c)$ , where

(a)  $P_a = S(2), S(2), S(1), J(1, 2, 6), J(1, 1, 3)$

(b)  $P_b = S(2), J(2, 3, 3), J(1, 1, 1)$

(c)  $P_c = S(1), S(1), S(2), J(1, 2, 6), J(1, 1, 1)$

- Prove that the function

$$g(x) = \begin{cases} 1 & \text{if } W_x = \emptyset \\ 0 & \text{otherwise} \end{cases}$$

is not URM computable. In other words, there is no URM program that, on input  $x$ , always halts and either outputs 1 or 0 as output, depending on whether or not  $\phi_x$  has an empty domain. Do this by writing a program  $P$  that uses  $g$  and makes use of the **self** programming concept. Then show how  $P$  creates a contradiction.

- Consider the function

$$g(x) = \begin{cases} 1 & \text{if } |E_x| = \infty \\ 0 & \text{otherwise} \end{cases}$$

In other words  $g(x) = 1$  iff function  $\phi_x(y)$  has an infinite range, meaning that it outputs an infinite number of different values. Evaluate  $g(a)$ ,  $g(b)$ , and  $g(c)$ , where

- (a)  $\phi_a(y) = y^2$
- (b)  $\phi_b(y) = y$
- (c)  $\phi_c(y) = \text{sgn}(y)$ .

7. Prove that the function

$$g(x) = \begin{cases} 1 & \text{if } |E_x| = \infty \\ 0 & \text{otherwise} \end{cases}$$

is not URM computable. In other words, there is no URM program that, on input  $x$ , always halts and either outputs 1 or 0 as output, depending on whether or not  $\phi_x$  has an infinite range. Do this by writing a program  $P$  that uses  $g$  and makes use of the **self** programming concept. Then show how  $P$  creates a contradiction.

8. Rice's theorem states that if  $\mathcal{C}_1$  denotes the set of unary computable functions, and  $\mathcal{B}$  is a nonempty proper subset of  $\mathcal{C}_1$ , then the predicate function

$$B(x) = \begin{cases} 1 & \text{if } \phi_x \in \mathcal{B} \\ 0 & \text{otherwise} \end{cases}$$

is undecidable. Prove Rice's theorem by writing an informal program  $P$  that uses  $B(x)$  and makes use of the **self** programming concept. Then show how  $P$  creates a contradiction. Hint: assume  $B(x)$  is decidable, and take advantage of the fact that the set of functions  $\mathcal{B}$  is both nonempty and not all of  $\mathcal{C}_1$ .

- 9. For each constant  $n \geq 1$ , show that  $\lfloor x^{1/n} \rfloor$  is a primitive-recursive function of  $x$ .
- 10. Prove that there exists an  $n$  for which  $\phi_n(x) = \lfloor x^{1/n} \rfloor$ . Hint: use the s-m-n theorem and Theorem 4.
- 11. Recall that program  $P_x$  has the self-output property iff  $x \in E_x$ . By writing an informal program that makes use of the programming construct **self**, prove that the self-output property is undecidable.
- 12. Show that there is a number  $e$  for which  $\phi_e(x) = e^{10}$ , for all  $x \in \mathcal{N}$ .
- 13. Consider the following description of a function  $f(n)$ . On input  $n$ , return the Gödel number of the program  $P'$  that is the result of appending program  $P_n$  with a minimum number of successor instructions  $S(1), \dots, S(1)$  so that it is always guaranteed that, should  $P_n$  halt on an input, then the final instruction of  $P'$  will be one of these successor instructions. Then by the Church-Turing thesis,  $f$  is total computable. Moreover, prove that, if  $n$  is a fixed point for  $f(n)$ , i.e.  $\phi_n = \phi_{f(n)}$ , then necessarily  $\phi_n(x)$  is undefined for all  $x$ .

## Exercise Solutions

- 1. Since the proof of Kleene's 2nd Recursion Theorem constructs  $e$  as  $e = \gamma(ABC)$ , by changing the instructions of  $B$ , we get a new value for  $e$ , since  $B$  has changed. We only have to make sure that  $B$ 's instructions are changed in a trivial way that does not affect its functionality as described in the proof.

2. A function  $\phi_x(y)$  is onto iff  $E_x = \mathcal{N}$ , where  $E_x$  denotes the range of  $\phi_x$ . Thus,

(a)  $g(a) = 0$  since  $\phi_a(y) = y^2$  is not onto since  $E_a = \{1, 4, 9, 25, \dots\} \neq \mathcal{N}$ ,

(b)  $g(b) = 0$  since  $\phi_b(y) = 1$  is not onto since  $E_b = \{1\} \neq \mathcal{N}$ , and

(c)  $g(c) = 1$  since  $\phi_c(y) = y$  is onto since  $E_c = \mathcal{N}$ .

3. We have the following program  $P$ .

Input  $y \in \mathcal{N}$ .

If  $g(\mathbf{self}) = 1$ , loop forever.

Return  $y$ ;

If  $g(\mathbf{self}) = 1$ , then  $P$  has a range equal to  $\mathcal{N}$  which is impossible since it does not terminate on any input (loops forever). If  $g(\mathbf{self}) = 0$ , then  $P$  does not have a range equal to  $\mathcal{N}$ , which is contradicted by the fact that  $P$  returns  $y$  on input  $y$ , and so has the set of return values  $\{0, 1, \dots\} = \mathcal{N}$ .

4. We have the following answers.

(a)  $g(a) = 0$  since  $P_a$  terminates on input 1 (verify!) and thus  $W_a = \{1\} \neq \emptyset$ .

(b)  $g(b) = 1$  since  $P_b$  does not terminate on any input (why?) and thus  $W_b = \emptyset$ .

(c)  $g(c) = 1$  since  $P_c$  does not terminate on any input (why?) and thus  $W_c = \emptyset$ .

5. We have the following program  $P$ .

Input  $y \in \mathcal{N}$ .

If  $g(\mathbf{self}) = 1$ , Return 0.

Loop Forever.

If  $g(\mathbf{self}) = 1$ , then it means  $W_{\mathbf{self}} = \emptyset$ , but  $P$  returns 0 for each input  $y$ , which implies  $W_{\mathbf{self}} = \mathcal{N}$ , a contradiction.

If  $g(\mathbf{self}) = 0$ , then it means  $W_{\mathbf{self}} \neq \emptyset$ , but  $P$  loops forever on each input  $y$ , which implies  $W_{\mathbf{self}} = \emptyset$ , a contradiction.

6. We have the following answers.

(a)  $g(a) = 1$  since  $\phi_a(y) = y^2$  has an infinite range:  $E_a = \{1, 4, 9, 25, \dots\}$ ,

(b)  $g(b) = 1$  since  $\phi_b(y) = y$  has an infinite range  $E_b = \mathcal{N}$ , and

(c)  $g(c) = 0$  since  $\phi_c(y) = \text{sgn}(y)$  has finite range equal to  $\{0, 1\}$ .

7. Consider the following program  $P$ .

Input  $y \in \mathcal{N}$ .

If  $g(\mathbf{self}) = 1$ , Return 0.

Return  $y$ .

If  $g(\mathbf{self}) = 1$ , then it means  $|E_{\mathbf{self}}| = \infty$ , but the program returns 0 for each input  $y$ , which implies  $E_{\mathbf{self}} = \{0\}$  which is finite, a contradiction.

If  $g(\mathbf{self}) = 0$ , then it means  $|E_{\mathbf{self}}|$  is finite, but the program returns  $y$  on each input  $y$ , which implies  $E_{\mathbf{self}} = \mathcal{N}$ , a contradiction.

8. Assume  $B(x)$  is decidable. Since  $\mathcal{B}$  is nonempty there exists a unary computable function  $f \in \mathcal{B}$ . Similarly, since  $\mathcal{B}$  is not all of  $\mathcal{C}_1$ , there is a unary computable function  $g \notin \mathcal{B}$ . Now consider the following program  $P$ .

```

Input  $x \in \mathcal{N}$ .
If  $B(\mathbf{self}) = 1$ ,
    Simulate  $g$  on input  $x$ .
    Return  $g(x)$  if it is defined.
Simulate  $f$  on input  $x$ .
Return  $f(x)$  if it is defined.

```

Since  $f$  and  $g$  are computable, so is  $P$ . Let  $e$  denote the Gödel number of  $P$ . Assume  $B(e) = 1$ . By definition, this means that  $\phi_e \in \mathcal{B}$ . But in examining  $P$  we see that  $P$  simulates  $g$  so that  $\phi_e = g \notin \mathcal{B}$ , a contradiction. Similarly, if  $B(e) = 0$ , then  $\phi_e \notin \mathcal{B}$ . But in this case  $P$  simulates  $f$  so that  $\phi_e = f \in \mathcal{B}$ , a contradiction. Therefore,  $B$  cannot be decidable.

9. The function  $\lfloor x^{1/n} \rfloor$  may be computed as

$$\mu(z \leq x)(z^n > x) - 1.$$

10. Function  $f(n, x) = \lfloor x^{1/n} \rfloor$  is computable by the previous exercise. Therefore, by the s-m-n theorem, there exists a total computable function  $k(n)$  for which  $\phi_{k(n)}(x) = \lfloor x^{1/n} \rfloor$ . Finally, by Theorem 4, there is an integer  $n$  for which

$$\phi_n(x) = \phi_{k(n)}(x) = \lfloor x^{1/n} \rfloor.$$

11. Assume  $E(x)$  is decidable, where  $E(x) = 1$  iff  $x \in E_x$ . Now consider the following program  $P$ .

```

Input  $x \in \mathcal{N}$ .
If  $E(\mathbf{self}) = 1$ ,
    Loop forever.
Return  $\mathbf{self}$ .

```

Since  $E(x)$  is decidable,  $P$  is computable. Let  $e$  denote the Gödel number of  $P$ . Assume  $E(e) = 1$ . By definition, this means that  $e \in E_e$ , meaning that  $P$  returns  $e$  on some input  $x$ . However, since  $E(e) = 1$ ,  $P$  does not terminate on any input, meaning that  $E_e = \emptyset$ , a contradiction.

Similarly, if  $E(e) = 0$ , then  $e \notin E_e$ . But in this case  $P$  returns  $e$ , meaning that  $e \in E_e$ , a contradiction. Therefore,  $E(x)$ , i.e. the Self-Output property, is not decidable.

12. Function  $f(y, x) = y^{10}$  is primitive recursive, and hence computable. Therefore, by the s-m-n theorem, there exists a total computable function  $k(y)$  for which  $\phi_{k(y)}(x) = y^{10}$ . Finally, by Theorem 4, there is an integer  $e$  for which

$$\phi_e(x) = \phi_{k(e)}(x) = e^{10}$$

for all  $x \in \mathcal{N}$ .

13. Since  $f(n)$  is total computable, by Theorem 4 there is an integer  $n$  for which  $\phi_n(x) = \phi_{f(n)}(x)$  for all  $x \in \mathcal{N}$ . But the way in which Gödel number  $f(n)$  is constructed is such that, whenever  $\phi_n(x) = y$  is true, then  $P_n$  halts, which in turn implies that  $P_{f(n)}$  halts with  $\phi_{f(n)}(x) = y + 1$ , since  $P_{f(n)}$  is the same as  $P_n$ , except that in its final instruction it adds 1 to register  $R_1$ . Thus, if  $\phi_n(x)$  is defined, then we have  $\phi_n(x) = y \neq \phi_{f(n)}(x) = y + 1$ . Therefore, we must conclude that  $\phi_n(x)$  must always be undefined, meaning that  $W_n = \emptyset$ .