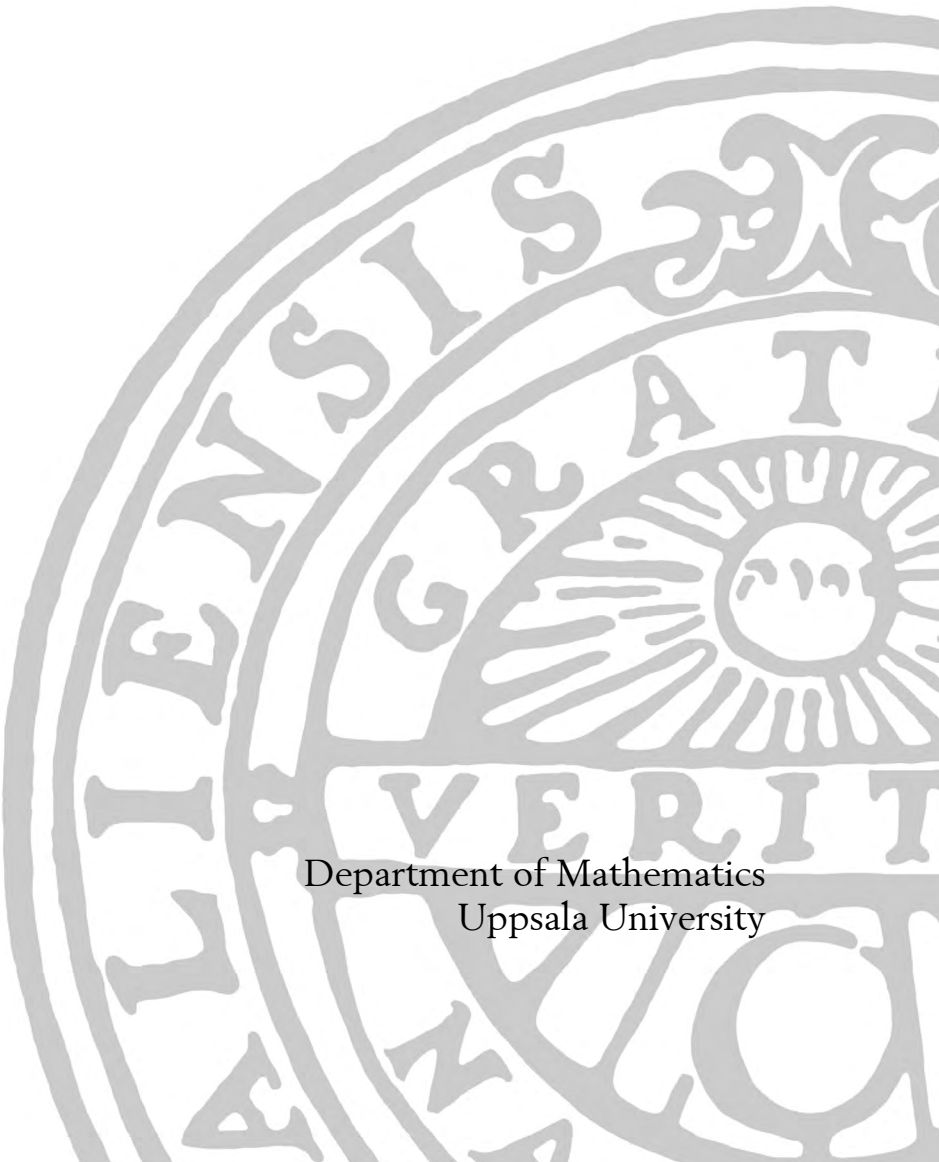
UPPSALA
UNIVERSITET

U.U.D.M. Project Report 2019:31

Post's problem in recursion theory and its solution

Anton Hallgren

Examensarbete i matematik, 15 hp
Handledare: Vera Koponen
Examinator: Martin Herschend
Juni 2019



Department of Mathematics
Uppsala University

Contents

1 Introduction

Computability theory is a field that studies algorithms and their complexity. The algorithms have natural numbers as their input and output, but we may consider more complex objects as input or output if we can find a way to encode them as numbers. Examples of such objects are tuples and finite sets of natural numbers, strings of symbols or other algorithms.

Problems in computability theory are therefore often about relationships between algorithms and sets of numbers, such as whether there is an algorithm that can distinguish between the numbers that are in a set and those that are not. One part of this is to study ways to compare the complexity of sets of natural numbers in the context of algorithms.

In this text we will study Turing reducibility which is one way to compare the complexity of sets, the related concept Turing degree and Post's problem which relates to these concepts. Informally speaking, a set of natural numbers B has a Turing degree which is at least as large as the Turing degree of a set A if there is some algorithm that computes the characteristic function of A if the algorithm has access to the characteristic function of B. The main result, informally stated, is that there is a Turing degree which lies strictly between the Turing degree of $\emptyset$ and the Turing degree of "the halting problem" which is the set denoted K in this text.

The term algorithm refers to a finite collection of instructions describing how to execute a procedure of computations (for example finding the greatest common divisor of two numbers). It is not precise enough to use in mathematical proof, but it is possible to define specific types of algorithms that are more precise, of which the most commonly used is Turing machines.

We will also use the term recursive functions (which is precisely defined, see Definition 1). It is a known theorem that a function is recursive if and only if it can be computed by a Turing machine.

It is conjectured that anything that we would recognize as an algorithm can be programmed as a Turing machine, which also means that the function computed by any algorithm is recursive. This is known as the Church-Turing thesis and cannot be proved due to the impreciseness of the term algorithm.

When proving results involving recursive functions or Turing machines we will often describe an algorithm informally without properly translating it into a Turing machine or recursive function.

This text is based on material from the books [1] and [2]. Some concepts are explained in more detail there.

2 Basic coumputability theory

This section will be about some of the basic concepts in computability theory, I will not go into too much details or give any proofs. See the references for more details and proofs.

We will frequently discuss subsets of $\mathbb{N}$ or $\mathbb{N}^k$, where $\mathbb{N}$ is the set of natural

numbers $\{0, 1, 2, \ldots\}$ and $\mathbb{N}^1$ is identified with $\mathbb{N}$. Unless otherwise stated the term "set" will therefore be used to refer to those subsets specifically.

For a set $A \subseteq \mathbb{N}^k$ the complement (with respect to $\mathbb{N}^k$) is denoted $\overline{A}$, i.e. $\overline{A} = \mathbb{N}^k \backslash A$.

Similarly we will use the term (partial) function specifically to refer to (partial) functions from $\mathbb{N}^k$ to $\mathbb{N}$, where partial means that it may be defined only on a subset of $\mathbb{N}^k$ rather than the entire set. The set that a function f is defined for is called its domain ($domain(f)$) and the set of values that it takes is called its range ($range(f)$).

2.1 Recursive functions

We will begin by looking at the functions that can be calculated by algorithms, which is the recursive functions defined as follows:

Definition 1. We define the set of *partial recursive functions* to be the smallest set $\mathcal{C}$ of partial functions such that:

1. The zero function $Z(x) = 0$ is in $\mathcal{C}$.
2. The successor function $s(x) = x + 1$ is in $\mathcal{C}$.
3. For any natural numbers k and i such that $1 \leq k$ and $1 \leq i \leq k$ the projection function $p_i^k(x_1, x_2, \ldots, x_k) = x_i$ is in $\mathcal{C}$.
4. Closure under composition: If g_1, g_2,...,g_m are k-ary functions in $\mathcal{C}$ and h is a m-ary function in $\mathcal{C}$ then

   $$f(x_1, x_2, \ldots, x_k) = h(g_1(x_1, x_2, \ldots, x_k), g_2(x_1, x_2, \ldots, x_k), \ldots, g_m(x_1, x_2, \ldots, x_k))$$

   is a k-ary function in $\mathcal{C}$.
5. Closure under primitive recursion: If h is a $(k-1)$-ary function in $\mathcal{C}$ and g is a $(k+1)$-ary function in $\mathcal{C}$ then the k-ary function f defined by

   $$f(0, x_2, \ldots, x_k) = h(x_2, \ldots, x_k)$$

   $$f(x_1 + 1, x_2, \ldots, x_k) = g(x_1, x_2, \ldots, x_k, f(x_1, x_2, \ldots, x_k))$$

   is a k-ary function in $\mathcal{C}$.
6. Closure under unbounded search: If $f(x_1, \ldots, x_k, y)$ is a $(k+1)$-ary function in $\mathcal{C}$ then the function $us_y f(x_1, \ldots, x_k, y)$ is in $\mathcal{C}$, where $us_y f(x_1, \ldots, x_k, y)$ is defined to be equal to z if $f(x_1, \ldots, x_k, z) = 0$ and $f(x_1, \ldots, x_k, y) > 0$ for $y < z$ and undefined if no such z exists.

If a partial recursive function is defined for all $x \in \mathbb{N}^n$ (for some n) we call it a *total recursive function*.

The set of functions that only satisfies condition 1 to 5 from the definition is called the set of *primitive recursive functions*. Primitive recursive functions are always total. We will not be interested in them in this text.

It can be shown that any partial recursive function can be computed by some Turing machine, and the function computed by any Turing machine is a recursive function. According to Church-Turing thesis this is true for any algorithm, not only Turing machines. This is not proven, due to the impreciseness of the term algorithm, but has been seen to be true for every kind of algorithm that has been tested. With this as motivation we will frequently show that a function is recursive by giving an informal description of an algorithm that computes it.

It is possible to encode a Turing machine, and thereby a recursive function, with an integer.

Definition 2. The Turing machine with code e is denoted by P_e. The k-ary function computed by it is denoted by ϕ_e^k. If it is 1-ary we usually just call it ϕ_e.

Any partial recursive function has an infinite number of codes (since the algorithm be can changed in ways that has no effect on the output).

Theorem 1. *The $(k+1)$-ary partial function defined by $U_k(e, x_1, \ldots, x_k) := \phi_e^k(x_1, \ldots, x_k)$ is recursive.*

Lemma 2 (s-m-n projection)**.** *For every m, n there is a total recursive function S_n^m such that if $S_n^m(e, x_1, \ldots, x_m) = z$ then $\phi_z^n(y_1, \ldots, y_n) \simeq \phi_e^{(n+m)}(x_1, \ldots, x_m, y_1, \ldots, y_n)$*

2.2 Recursive and recursively enumerable sets

After considering partial functions that can be computed by by algorithms we will look at for what subsets of $\mathbb{N}$ an algorithm can determine whether some element is in it or not.

Definition 3 (Characteristic function)**.** The *characteristic function* of a set A is

$$\chi_A(x) = \begin{cases} 1 & \text{for } x \in A \\ 0 & \text{for } x \notin A \end{cases}$$

We say that A is recursive if χ_A is a total recursive function.

Definition 4 (Recursive enumerable)**.** Let A be a set of natural numbers. We say that A is *recursively enumerable* if there exists a total recursive function f such that $A = \{f(0), f(1), f(3), \ldots\}$ or if $A = \emptyset$.

Theorem 3. *Let A be a set of non-negative integers. The following are equivalent:*

1. *A is recursively enumerable.*

2. *A is the domain of a partial recursive function*

3. *The function* $h(x) = \begin{cases} 1 & \text{for } x \in A \\ \text{undefined} & \text{for } x \notin A \end{cases}$ *is partial recursive.*

Definition 5. W_x is the domain of ϕ_x.

Definition 6. $K = \{x | x \in W_x\}$.

It is well known that K is recursively enumerable but that its complement $\overline{K} = \{x | x \notin W_x\}$ is not recursively enumerable. From the following theorem it follows that K is not recursive.

Theorem 4. *A set A is recursive if and only if both A and its complement* $\overline{A}$ *are recursively enumerable.*

This is a special case of Theorem 8 that I will get to and prove later.

3 Relative recursivity

Now we will look at the concept of relative recursivity. The idea is to use algorithms with an "oracle commando" that calculates $\chi_X(n)$ for any number n, without having a description of how $\chi_X(n)$ is calculated.

3.1 Recursive functions relative to a set

The easiest way of defining the notion of recursive functions relative to some set is in the following way.

Definition 7. The set of partial *recursive functions relative to a set* X is the smallest set of partial functions satisfying 1 to 6 of Definition 1 that also contains the function χ_X. We also call them *partial* X*-recursive functions.*

Just like how partial recursive functions are computed by algorithms, partial X-recursive functions are computed by special algorithms called oracle machines. An oracle machine is an algorithm like those previously mentioned but with the ability to acquire information of what elements are in the set X.

For our oracle machines we will use a modified Turing machine with an additional tape. The additional tape contains, in ascending order, an encoding of each element in X separated by blank-spaces (here encoding simply means a number of 1's equal to the encoded number plus one). The structure of the transition function is made so that it can read from the second tape but not write on it. The oracle machine can then determine that some x is in X by finding $(x + 1)$ 1's on the second tape or determine if x is not in X by finding a smaller number of 1's (or the start of the list) directly followed by a larger number of 1's (or the end of the list if it is finite).

An oracle machine can be encoded with an integer just like a normal Turing machine. It is important to notice that this code only contains information

of how the oracle machine works and handles information from X (i.e. the transition function) and not the actual information of X. The information of X is not a part of the oracle machine but rather a secondary input of unlimited size.

Definition 8. We denote by P'_e the oracle machine with code e and by ϕ_e^X the unary partial function computed by P'_e relative to the set X.

Theorem 5. *The functions that can be written on the form ϕ_e^X are exactly those that are partial recursive relative to X.*

This can be proven in a similar way to how it is proven that a function is recursive if and only if it is computed by a Turing machine.

Theorem 6. *If A is recursive then ϕ_z^A is a partial recursive function.*

Proof. If A is recursive then χ_A is in the set of partial recursive functions, so the set of partial A-recursive functions satisfy the conditions of Definition 1 without being larger than the set of partial recursive functions, so they are the same. Therefore ϕ_z^A is recursive.

Alternatively, since A is recursive some algorithm can tell whether $x \in A$ or not. Then ϕ_z^A can be computed in a recursive way by the oracle machine P'_z with the oracle replaced by that algorithm. □

3.2 Relatively recursive sets

The idea of relatively recursive or recursively enumerable sets is the same as the non-relative case, except that we use oracle machines instead of regular algorithms.

Definition 9. A is *recursive* in B if χ_A is B-recursive. A is *recursively enumerable* in B if $A = \emptyset$ or $A = range(f)$ for some total B-recursive function f.

One obvious example of this is that every set is recursive relative to itself. More generally if f is a recursive function then $\{f(x)|x \in X\}$ is recursive relative to X. Another example is that the complement $\overline{X}$ is recursive relative to X.

Theorem 7. *If A is recursive in B then A is recursively enumerable in B.*

This is proven the same way as it is proven that every recursive set is recursively enumerable.

Theorem 8. *A is recursive in B if and only if both A and $\overline{A}$ are recursively enumerable in B.*

Proof. If A is recursive in B then $\overline{A}$ is also recursive in B and then both A and $\overline{A}$ are recursively enumerable in B.

If either A or $\overline{A}$ is the empty set $\emptyset$ then both sets are recursive and therefore recursive in B.

If both A and $\overline{A}$ are recursively enumerable and none of them is $\emptyset$ then they are the ranges of two B-recursive functions, $A = range(f)$ and $B = range(g)$. Use f and g to generate the list $f(0), g(0), f(1), g(1), f(2), g(2), \ldots$. Since $A \cup \overline{A} = \mathbb{N}$ every number will eventually appear and we can determine whether a number x is in A or not by going through this list until we find x and seeing which function found it. □

Definition 10. W_x^X is the domain of ϕ_x^X.

Theorem 9. *A is recursively enumerable in $B \Leftrightarrow (\exists z)[A = W_z^B]$.*

Proof. $\Rightarrow$: Case (i): $A = \emptyset$. Since the function that is undefined for every input is partial recursive $A = W_z^B$ for some z.

Case (ii): $A \neq \emptyset$: $A = range(f)$ for some total B-recursive function f. We can construct an algorithm that halts exactly for inputs $x \in A$ by letting it compute $f(0)$, $f(1)$, ... until it finds an n such that $f(n) = x$ and then halt (the output is irrelevant, we can say that it gives n as its output). The function defined by this algorithm is partial B-recursive and has some code z. Then ϕ_z^B has A as its domain so $A = W_z^B$.

$\Leftarrow$: $A = W_z^B = domain(\phi_z^B)$ for some z. We define an algorithm that does as follows on input n: First calculate 1 machine step of $\phi_z^B(0)$, then two machine steps of $\phi_z^B(0)$ and $\phi_z^B(1)$, three machine steps of $\phi_z^B(0)$, $\phi_z^B(1)$ and $\phi_z^B(2)$ and so on until you have found $n+1$ such computations that halt within the given number of steps, then give the input to the $(n+1)$'th halting computation as output. The $(n+1)$ computations may include multiple instances with the same input, since it will try the same input with increasing upper bound on computation time (this is needed in cases where A is finite to guarantee that the algorithm halt for n larger than the number of elements in A).

If some x is in A then $\phi_z^B(x)$ will halt in a finite number of steps and then the algorithm above will give output x for some n. Likewise if x is not in A the algorithm will never give x as an output. Finally, unless $A = \emptyset$ (in which case A is recursively enumerable anyways) the algorithm will successfully find an output for any possible input. Therefore A is recursively enumerable in B. □

4 Turing degrees and Post's problem

4.1 Turing reducibility and Turing degrees

In recursion theory there are a number of different ways to reduce one set to another and classify them into different degrees. In this text we will only consider Turing reducibility and Turing degrees and in that context they can be seen as a simpler notation for how sets relate to each other in terms of relative recursivity.

Definition 11 (Turing reducibility)**.** A is *Turing reducible* to B (notation: $A \leq_T B$) if A is recursive in B.

Theorem 10. *$\leq_T$ is reflexive and transitive.*

Proof. Reflexive: It has already been mentioned that every set is recursive relative to itself from which it follows that $\leq_T$ is reflexive.

Transitive: If $A \leq_T B$ and $B \leq_T C$ then A is determined by an oracle machine P'_a using B and and B is determined by an oracle machine P'_b using C. We can then make an oracle machine that works like P'_a but whenever P'_a asks "is x in B" it instead runs P'_B on input x relative to C to get the answer. This oracle machine determines A using C so $A \leq_T C$. Therefore, $\leq_T$ is transitive.

Alternatively, we can see that if $B \leq_T C$ then χ_B is in the set of C-recursive functions which means that the set of B-recursive functions is a subset of the set of C-recursive functions. □

Definition 12 (Turing degrees)**.** $A \equiv_T B$ if $A \leq_T B$ and $B \leq_T A$. This is an equivalence relation (it follows from the previous theorem), we call the equivalence classes *Turing degrees* and denote the equivalence class of a set A with $d_T(A)$.

If A is recursive, then $A \leq_T B$ for any set B. If also $B \leq_T A$ then B is also recursive. Therefore there is one Turing degree that contains exactly the recursive sets. We usually use $\emptyset$ as a representative of the Turing degree of recursive sets, so we call this Turing degree $d_T(\emptyset)$. This is the minimum of all the Turing degrees.

For an example of a non recursive Turing degree we have $d_T(K)$, as we know that K is not recursive.

Definition 13 (Turing complete)**.** A is *Turing complete* if

1. A is recursively enumerable and
2. $(\forall B)[B$ is recursively enumerable $\Rightarrow B \leq_T A]$

Theorem 11. *K is Turing complete.*

Proof. We already know that K is recursively enumerable, we need to show that $W_z \leq_T K$ for any z. Let S^1_1 be the 2-ary recursive function from Lemma 2 (the s-m-n projection) and z_2 be the code of $\phi_z(p^2_1(x, y))$. Then the unary partial function with code $z_3 = S^1_1(z_2, x)$ is $\phi_{z_3}(y) = \phi_{z_2}(x, y) = \phi_z(p^2_1(x, y))$. This function is defined for every input y if $x \in W_z$ and for no input y if $x \notin W_z$. In particular it is defined for its own code z_3 if and only if $x \in W_z$. Since it is possible to compute z_3 from z_2 and x we only need to know if z_3 is in K to know if $x \in W_z$. Thus there is an oracle machine that determines whether x is in W_z or not with help from K, so W_z is recursive relative to K and then $W_z \leq_T K$. We conclude that K is Turing complete. □

It follows that $d_T(K)$ is a maximum of all recursively enumerable Turing degrees. There are also larger Turing degrees, but they are not recursively enumerable.

Definition 14. $K^A = \{x | x \in W^A_x\}$

It can be shown that $A \leq_T K^A$ and $K^A \not\leq_T A$. Therefore $\emptyset <_T K^\emptyset <_T K^{K^\emptyset} <_T \ldots$. While K and $K^\emptyset$ are not (necessarily) the same, they are in the same Turing degree.

4.2 Post's Problem

Post's problem was to prove (or disprove) the statement of the following theorem.

Theorem 12. *There are recursively enumerable T-degrees other than $d_T(\emptyset)$ and $d_T(K)$.*

In the next section this will be shown by proving another theorem by Friedberg and Muchnik. Then the solution to Post's problem follow.

Theorem 13. *If A and B are sets and there is a total function f such that $\forall x(f(x) \in A \leftrightarrow f(x) \in W_x^B)$ then A is not recursive in B.*

Proof. Let A and B be sets and f a function as described above. Then for any number x, either both or none of A and W_x^B will contain $f(x)$. Therefore it is impossible for $\overline{A}$ to be equal to W_x^B since $f(x) \in W_x^B \Leftrightarrow f(x) \in A \Leftrightarrow f(x) \notin \overline{A}$, so $\overline{A}$ is not recursively enumerable in B by theorem 9. Therefore A cannot be recursive in B by theorems 8 and 9. □

If the function f in the Theorem 13 is recursive we say that A is *constructively non-recursive in B*.

4.3 Friedberg-Muchnik theorem

Definition 15. Let z and a finite set D be given. We define $W_{z,n}^D = \{x | x < n$ and P_z' halts within n machine steps on input x relative to $D\}$.

Lemma 14. *For any sequence of finite sets A_0, A_1, ... such that $A_0 \subseteq A_1 \subseteq \ldots$, if $A = \bigcup_{n=0}^{\infty} A_n$ and if $a \in W_z^A$, then $\exists m \forall n (m \leq n \Rightarrow a \in W_{z,n}^{A_n})$.*

Proof. $a \in W_z^A$ means that the oracle machine P_z' with input a relative to A halts after a finite number of steps, say s steps. At that time it will have tested if a finite number elements is in A. Let b be an element that was tested. If $b \notin A$ then $b \notin A_i$ for all i since $A_i \subset A$. If $b \notin A_i$ for all i then $b \notin A$ since $A = \bigcup_{n=0}^{\infty} A_n$. Since only a finite number of elements were tested there is an A_i that contains every tested element in A.

Doing the computation P_z' with input a relative to A_i, or any A_j with $i \leq j$, will then give the exact same result as if it is done relative to A. By letting m be the maximum of i and s we get the result of the lemma. □

Theorem 15 (Friedberg-Muchnik theorem)**.** *There exist recursively enumerable sets A and B that are incomparable with respect to $\leq_T$.*

Proof. This will be proven by presenting an algorithm that defines the sets A and B so that they becomes recursively enumerable. We will then show that this definition ensures that none of A and B become recursive relative to the other.

The algorithm uses two lists of the natural numbers, the A-list and the B-list, which correspond to the sets. In each list an integer can be marked with a $(+)$ meaning it will be in the corresponding set or a $(-)$ which will be used later in the proof (a $(-)$ can later be replaced by a $(+)$).

We will also use two sets of movable markers a_0, a_1, a_2, ... and b_0, b_1, b_2, ... that can be associated with a number in their respective lists (a_i for the A-list, b_i for the B-list). They will first be introduced as associated with one element and can later change to be associated with another number later in the same list.

The algorithm takes place in stages. At each stage, only a finite number of numbers from each of the lists have been used (by being marked with $(+)$ or $(-)$ or being associated to a marker)and only finitely many markers are associated with some number in the list.

We define two lists of sets A_0, A_1, A_2, ... and B_0, B_1, B_2, ... by letting $A_0 = B_0 = \emptyset$, $A_n = \{x | x \text{ has got a } (+) \text{ in the } A\text{-list at the end of stage } n\}$ and $B_n = \{x | x \text{ has got a } (+) \text{ in the } B\text{-list at the end of stage } n\}$. Then we define A and B by $A = \bigcup_{n=1}^{\infty} A_n$ and $B = \bigcup_{n=1}^{\infty} B_n$.

We call an integer in a list *free* if no marker is associated to it or to any integer later in the same list. We call an integer in a list *vacant* if it does not have a $(+)$. Now we are ready to describe the algorithm itself.

Stage 1*:* Associate a_0 with 0 in the A-list.

Stage 2*:* Associate b_0 with 0 in the B-list.

Stage $2n+1$*:* Associate a_n with the first free integer in the A-list. Let $a_0^{(n)}, \ldots, a_n^{(n)}$ be the current positions of the markers $a_0, \ldots, a_n$ so $a_0^{(n)}$ is the number marked by a_0 currently, and so on. Compute the sets $W_{0,n}^{B_{2n}}, \ldots, W_{n,n}^{B_{2n}}$. If there is no number k such that $a_k^{(n)}$ is vacant and occurs in $W_{k,2n}^{B_{2n}}$ then go to the next stage $(2n+2)$. If there is at least one such number, let j be the smallest of them. Take the integers in $\overline{B}_{2n}$ whose membership was used to show $a_j^{(n)} \in W_{j,n}^{B_{2n}}$ and mark them with a $(-)$ in the B-list and mark $a_j^{(n)}$ with a $(+)$ in the A-list. Then for each i such that $j \leq i < n$ move b_i to the first free integer in the B-list.

Stage $2n+2$*:* Mostly the same as stage $2n+1$ but with A and B switching roles. Associate b_n with the first free integer in the B-list. Let $b_0^{(n)}, \ldots, b_n^{(n)}$ be the current positions of the markers $b_0, \ldots, b_n$. Compute the sets $W_{0,n}^{A_{2n+1}}, \ldots, W_{n,n}^{A_{2n+1}}$. If there is no number k such that $b_k^{(n)}$ is vacant and occurs in $W_{k,n}^{A_{2n+1}}$ then go to the next stage $(2n+3)$. If there is at least one such number, let j be the smallest of them. Take the integers

in $\overline{A}_{2n+1}$ whose membership was used to show $b_j^{(n)} \in W_{j,n}^{A_{2n+1}}$ and mark them with a $(-)$ in the A-list and mark $b_j^{(n)}$ with a $(+)$ in the B-list. Then for each i such that $j < i \leq n$ move a_i to the first free integer in the A-list.

There is no way to remove a $(+)$ mark from a number in the lists once it has got it, and from the description of how numbers can receive a $(-)$ it follows that this can only happen to numbers without a $(+)$. Therefore if a number ever gets a $(+)$ it will keep it forever. Then by the definition of A and B above A is the set of all numbers that ever gets a $(+)$ in the A-list and B is the set of all numbers that ever gets a $(+)$ in the B-list. Now we need to show that they have the desired properties.

First we need to show that each marker is only moved a finite number of times. A marker a_i will be moved only when the integer at b_j's position is marked with $(+)$ for $j < i$ and b_i will be moved only when the integer at a_j:s position is marked with $(+)$ for $j \leq i$. The marker a_0 is never moved, there is no marker b_j with $j < 0$, and b_0 is moved only if a_0 gets a $(+)$ so at most one time. For $j > 0$ the marker a_j may be moved once for each position that that b_{j-1} takes (since the number marked by b_{j-1} may get a $(+)$ on each of those positions) and whenever a_{j-1} is moved (since both are moved when earlier b markers get a $(+)$). A similar rule can be given for the b markers. From this it can be seen that the number of positions occupied by $a_0, b_0, a_1, b_1, a_2, \ldots$ is bounded by the Fibonacci sequence $0, 1, 1, 2, 3, 5, 8, \ldots$ and therefore each one of them will be moved only a finite number of times.

This means that each of them will eventually get to a final position and never be moved again. Let $f(x)$ be the final position of a_x and $g(x)$ the final position of b_x. We will show that $f(x) \in A \Leftrightarrow f(x) \in W_x^B$. The same argument can be used for $g(x) \in B \Leftrightarrow g(x) \in W_x^A$.

$\Rightarrow$: Let $f(x) \in A$. Then $f(x)$ is marked with a $(+)$ at some stage $2n+1$. Assume that at the same time $b_k^{(n)}$ is marked with a $(-)$. If $x \leq k$ then b_k is moved so the $(-)$ cannot be changed to a $(+)$. On the other hand, if $k < x$ and $b_k^{(n)}$ eventually receives a $(+)$ then a_x will be moved, contradicting that it was on its final position. We conclude that no integer that gets a $(-)$ in the same step as $f(x)$ gets its $(+)$ can be changed to a $(+)$. Therefore each number that was checked for membership in B_{2n} when concluding that $f(x) \in W_x^{B_{2n}}$ will give the same result if checked for membership in B, so the computation of $\phi_x^{B_{2n}}(f(x))$ is identical to that of $\phi_x^B(f(x))$. Hence $f(x) \in W_x^B$.

$\Leftarrow$: Let $f(x) \in W_x^B$. Then for n large enough a_x is at $f(x)$ and $f(x) \in W_{x,n}^{B_{2n}}$ by Lemma 14. Therefore $f(x)$ will eventually receive a $(+)$ so $f(x) \in A$.

Thus $f(x) \in A \Leftrightarrow f(x) \in W_x^B$ and similarly $g(x) \in B \Leftrightarrow g(x) \in W_x^A$. Hence $A \not\leq_T B$ and $B \not\leq_T A$ by Theorem 13. $\square$

Now we can easily derive:

Theorem 12. *There are recursively enumerable T-degrees other than $d_T(\emptyset)$ and $d_T(K)$.*

Proof. Let A and B be as in Theorem 15 . Since $\emptyset$ is recursive we have $\emptyset \leq_T A$ and $\emptyset \leq_T B$, therefore $d_T(\emptyset) \neq d_T(B)$ and $d_T(\emptyset) \neq d_T(A)$. Also since A and B are recursively enumerable and K is complete we have $A \leq_T K$ and $B \leq_T K$ so $d_T(B) \neq d_T(K)$ and $d_T(B) \neq d_T(K)$. This means that $d_T(A)$ and $d_T(B)$ are two new recursively enumerable Turing degrees which gives us the solution to Post's problem. □

Corollary 16. *There exist infinitely many recursively enumerable T-degrees such that none of them are comparable to each other with respect to* $\leq_T$.

See [1] for more details on this.

References

[1] Hartley Rogers. *Theory of recursive functions and effective computability.* McGraw-Hill, New York, 1967.

[2] Robert I. Soare. *Recursively enumerable sets and degrees: a study of computable functions and computably generated sets.* Springer-Vlg, New York, Berlin, 1987.