

Turing Degrees and Post's Problem

Tanmoy Chakraborty

June 22, 2006

1 Introduction

This report describes what I studied in summer 2006 on Turing degrees under Prof. David Madore at the Ecole Normale Supérieure. Most of my study on the topic was from the book titled “*Theory of Recursive Functions and Effective Computability*”, authored by Hartley Rogers, Jr.

2 Basic Definitions

An *alphabet* is a finite set of symbols. A *language* L is a subset of the set of all finite strings over a fixed *alphabet*. Since the set of all finite strings over a fixed finite alphabet is countable, we can equivalently define languages as functions $L : \mathbb{N} \rightarrow \{0, 1\}$, or as subsets of natural numbers.

We assume that the reader is familiar with the definition of Turing machines. To recall, a Turing machine is a computational model that takes a finite string over a fixed alphabet (equivalently, an integer), as input, and performs some mechanical, deterministic computation based on the input. Depending on the input, it can either *halt*, *i.e.* stop the computation, or *not halt in finite time*.

A Turing machine is a finite state machine, with a finite number of infinite tapes which it can read and write on, and which are used upto a finite length at any point of the computation, and a finite set of rules. Every Turing machine can be encoded as a finite string, and hence the set of Turing machines is countable. We shall assume a standard, natural and easy-to-compute bijection between Turing machines and the natural numbers. For example, the description of the Turing machine over an alphabet with ten characters, gives us a natural bijection. We say that a natural number *encodes* its corresponding Turing machine.

An input is provided to a Turing machine on one of its tapes. If the Turing machine halts, it can halt in an *accepting state* or a *rejecting state*. We denote the set of all strings, on which a Turing machine M halts in an accepting state, as $L(M)$, the language accepted by M . We say that a Turing machine is *total* if it halts on all inputs, and a total Turing machine is said to *decide* the language accepted by it.

An alternative model of Turing machines is capable of producing an output. One of its tapes is considered the output tape, and the Turing machine can write down a finite string, considered as output, on the tape before it halts. Such Turing machines compute functions $f : \mathbb{N} \rightarrow \mathbb{N}$. In this discussion, we shall, however, confine ourselves to the former model.

There are several other alternative models to capture the notion of computability. According to the Church-Turing thesis, all such models are equivalent with respect to the functions computable by them, and can compute all functions that are naturally considered as computable.

We call a language *recursively enumerable (r.e.)* if there exists a Turing machine accepting it. We call a language *recursive* if there exists a total Turing machine deciding it. Since the set of finite strings is infinite, so the set of all languages is uncountable. Since the set of all Turing machines is countable, so the set of all recursively enumerable languages is countable. Hence, most languages are not recursively enumerable.

We consider Turing machines with *oracles*. The oracle can be any language A . A Turing machine can call the oracle at any point in its computation, and as many times as it requires. It calls the oracle A with a string x , and the oracle promptly tells it whether $x \in A$. To state more precisely, a Turing machine M with an oracle A (denoted by M^A) has an additional tape on which it can write down a finite string, and call the oracle. The oracle writes down 1 or 0 on a specified tape of M , depending on whether the string written on the additional tape belongs to the language A or not. M changes its configuration depending on the output of the oracle. Thus the language accepted by M depends on A , and is denoted by $L(M^A)$. Henceforth, we shall discuss only Turing machines with oracles, and whenever we refer to a Turing machine without specifying any oracle for it, we shall assume that it is provided with the empty language \emptyset as oracle.

M^A can also be viewed as the Turing machine M provided with an additional read-only tape that has all the members of A written on it, in increasing order of length. The two models are equivalent, since the Turing machine can read the tape to decide membership of a string in A , instead of calling the oracle. The ordering of the strings on the read-only tape allows M to decide the membership of any string in A , in finite time.

We say that a language A is *Turing reducible* or *T-reducible* to another language B if there exists a total Turing machine M such that $A = L(M^B)$. We abbreviate it as $A \leq_T B$, and also say that A is recursive in B . If A can be accepted by a Turing machine(not necessarily total) with oracle B , we say that A is recursively enumerable in B .

We also abbreviate $A \leq_T B$ and $B \not\leq_T A$ as $A <_T B$.

3 Turing Degrees

It can be easily verified that the relation \leq_T is transitive, *i.e.* if $A \leq_T B$ and $B \leq_T C$, then $A \leq_T C$. Also, it is reflexive, *i.e.* $A \leq_T A$. However, it is not anti-symmetric, for, if \bar{A} denote the complement of A , then $A \leq_T \bar{A}$ and $\bar{A} \leq_T A$. We say that $A \equiv_T B$ if $A \leq_T B$ and $B \leq_T A$. Because of reflexivity and transitivity of \leq_T , \equiv_T is an equivalence relation. The equivalence classes of \equiv_T are called *Turing degrees* or *T-degrees*. We shall denote the T-degree of a language X by $deg(X)$. We can extend the notion of Turing reducibility to the Turing degrees in a natural way: If T_1 and T_2 are two Turing degrees, and $A \in T_1$, $B \in T_2$, then $T_1 \leq_T T_2$ if and only if $A \leq_T B$.

Turing reducibility(\leq_T) on T-degrees is anti-symmetric, in addition to being reflexive and transitive. It thus induces a partial order on the set of Turing degrees. The set of all recursive languages form the least element of the partial order, since any recursive language can be decided by a total Turing machine without making any call to the oracle. We denote this Turing degree by $0 = deg(\emptyset)$, where \emptyset is the empty language.

Turing degrees give a measure of the “uncomputability” of languages. $A \leq_T B$ indicates, in some sense, that A is no more difficult to compute than B , *i.e.* if B can be computed in some way, then A can be computed too. Thus, the higher the Turing degree of a language, the more uncomputable it is. The least element of the poset of T-degrees form the set of languages that was considered computable by the Church-Turing thesis.

The later sections of this report are devoted to the structure of this partially ordered set(poset) of T-degrees.

4 The Join Operation

Considering languages as sets of natural numbers, we define the *join* of two languages A and B as $A \oplus B = \{2n | n \in A\} \cup \{2n + 1 : n \in B\}$. Clearly, $A \leq_T A \oplus B$ and $B \leq_T A \oplus B$ (Given n as input, the Turing machine

checks whether $2n$ or $2n + 1$, respectively, is in $A \oplus B$). Further, suppose C is any Turing degree satisfying $A \leq_T C$ and $B \leq_T C$. Then, to decide $A \oplus B$, given n as input, a total Turing machine with oracle C can halve n and, depending on n was even or odd, check its membership in A or B , respectively. So it follows that $A \oplus B \leq_T C$. It is also easy to see that if $A_1 \equiv_T A_2$ and $B_1 \equiv_T B_2$, then $\text{deg}(A_1 \oplus B_1) = \text{deg}(A_2 \oplus B_2)$. Thus, the join operation is induced on the T-degrees, and if T_1 and T_2 are two Turing degrees such that $A \in T_1$, $B \in T_2$, then we can define the join operation on the degrees as $T_1 \vee T_2 = \text{deg}(A \oplus B)$.

Hence, for any two Turing degrees T_1 and T_2 , gives the least upper bound for $\text{deg}(A)$ and $\text{deg}(B)$. The existence of the least upper bound makes this partially ordered set an *upper semi-lattice*.

5 Turing Jump

As mentioned before, the set of T-degrees has a least element, namely, 0 , the set of recursive languages. However, it turns out that for each T-degree T , there exists a T-degree strictly greater than T .

Let $A \in T$. We define K_A as the following problem: Given $\langle x, y \rangle$ as input, decide whether the Turing machine encoded by x , supplied with A as an oracle, halts on y . We call K_A the *halting problem* for Turing machines with A as oracle.

Claim 1. $A <_T K_A$.

Proof. We can construct a Turing machine M which, given an input x , accepts x if x belongs to its oracle, and otherwise does not halt. Then M^A halts on an input x if and only if $x \in A$. In other words, $x \in A \Leftrightarrow \langle e(M), x \rangle \in K_A$, where $e(M)$ denotes the encoding of M . Hence, $A \leq_T K_A$.

Now, suppose that $K_A \leq_T A$. Then, there exists a total Turing machine M , which when given A as oracle, decides K_A . Using M , we can construct a Turing machine M' , with A as oracle, that, given x as input, accepts if $\langle x, x \rangle \in \overline{K_A}$, and does not halt otherwise. Suppose we give the input $e(M')$, the encoding of M' , to M' . Then, if M' accepts $e(M')$, it would imply that $\langle e(M'), e(M') \rangle \notin K_A$, i.e. M' does not halt on $e(M')$, a contradiction. If M' does not halt on $e(M')$, it would imply that $\langle e(M'), e(M') \rangle \in K_A$, i.e. M' halts on $e(M')$, again a contradiction. \square

We define $\text{deg}(K_A)$ to be the *Turing Jump* of T , and denote it by $TJ(T)$. From the above claim, it follows that $T <_T TJ(T)$. Again, this definition is unambiguous, since $A \equiv_T B \Leftrightarrow K_A \equiv_T K_B$.

Thus, there are infinite chains of distinct T-degrees. For any T-degree T_0 , we can define an infinite chain $T_0, T_1, T_2 \dots$ in which $T_n = TJ(T_{n-1})$, $\forall n > 0$. There even exists a T-degree that is an upper bound for the whole chain. Let A_n be a language in T_n , $\forall n \geq 0$. Consider the language $B = \{\langle n, x \rangle \mid x \in A_n\}$. Clearly, $A_n \leq_T B$, but $B \not\leq_T A_n$, $\forall n \geq 0$. Hence, $deg(B)$ is an upper bound for the chain.

The techniques discussed above, however, does not show the existence of two incomparable Turing degrees. Such pairs of Turing degrees exist, as we shall see later. Infact, there exist infinite anti-chains of Turing degrees under the partial order of \leq_T .

6 Post's Problem

Let $K = K_\emptyset$ be the halting problem for Turing machines with no oracle (or the empty language \emptyset as oracle). Clearly, K is recursively enumerable, since we can construct a Turing machine that, given $\langle x, y \rangle$ simulates the Turing machine encoded by x , fed with input y (Such a Turing machine is called a *Universal Turing machine*). It is an example of a problem that is recursively enumerable not recursive. Note that $deg(K) = TJ(0)$.

Claim 2. *If A is a recursively enumerable language, then $A \leq_T K$.*

Proof. If A is a recursively enumerable language, then there exists a Turing machine M accepting it. We can define a Turing machine M' that simulates M , accepts if M accepts, loops forever if M rejects, and does not halt if M does not halt. Thus, M' halts on x if and only if $x \in A$. We can find whether $x \in A$ by feeding the input $\langle \text{encoding of } M', x \rangle$ to the oracle K . \square

We say that A is *T-complete*, if $B \leq_T A$ for all recursively enumerable languages B . Hence, we have shown that K is T-complete. The above result easily relativises to yield the following:

Claim 3. *If B is recursively enumerable in A , then $B \leq_T K_A$.*

Definition. *A T-degree is called recursively enumerable if it contains at least one recursively enumerable language.*

Note that $deg(K) = TJ(0)$ is an upper bound for all recursively enumerable T-degrees, and 0 is a lower bound. For a long time, no other recursively enumerable T-degree was known. Post posed the following problem in 1944: Does there exist any recursively enumerable T-degree other than 0

and $\text{deg}(K)$? In other words, does there exist a language that is recursively enumerable, but neither recursive nor T-complete?

Post's problem remained unsolved for 12 years, until it was solved independently and almost simultaneously by Friedberg and Muchnik in 1956. They in fact proved a stronger statement, which is given below:

Theorem 1. *[The Friedberg-Muchnik Theorem] There exist recursively enumerable languages A and B such that $A \not\leq_T B$ and $B \not\leq_T A$.*

Theorem 1 implies the existence of two incomparable recursively enumerable Turing degrees. Since 0 and $\text{deg}(K)$ are comparable with all other recursively enumerable T-degrees, the theorem also implies that there are at least *two* Turing degrees between 0 and $\text{deg}(K)$.

We describe a modified form of Friedberg's proof for Theorem 1 in the next section.

7 Proof of Friedberg-Muchnik Theorem

Suppose M is the Turing machine encoded by the natural number x . Then, let $W_x = L(M)$, and $W_x^A = L(M^A)$.

Lemma 4. *Let A and B be any two languages. A is recursive in B if and only if both A and \bar{A} are recursively enumerable in B .*

Proof. If A is recursive in B , then \bar{A} is also recursive in B , hence both A and \bar{A} are recursively enumerable in B .

Let M_1 and M_2 be the Turing machines for A and \bar{A} respectively. We can construct a Turing machine M , with oracle B , that alternately simulates M_1 and M_2 on the same input on different sets of tapes: the odd steps simulate M_1 , while the even steps simulate M_2 on a set of steps disjoint from those on which M_1 is simulated. M accepts an input if the simulation of M_1 accepts it, and rejects an input if the simulation of M_2 accepts it. If $x \in A$, it gets accepted by M_1 , and hence accepted by M , in finite time. If $x \notin A$, it gets accepted by M_2 , and hence rejected by M , in finite time. Thus, M is a total Turing machine that decides A . \square

Lemma 5. *Let A be recursively enumerable in B . Then, $A \not\leq_T B$ if and only if there exists a function f such that $(\forall x)[f(x) \in A \Leftrightarrow f(x) \in W_x^B]$.*

Proof. Since A is recursively enumerable in B , lemma 4 implies that $A \not\leq_T B$ if and only if \bar{A} is not recursively enumerable in B . This is equivalent to saying that $\bar{A} \neq W_x^B \forall x$, which is again equivalent to saying that $(\forall x)(\exists y)[y \in A \Leftrightarrow y \in W_x^B]$. We define $f(x) = y$, to complete the proof. \square

If the function f in Lemma 5 is recursive, we say that A is *constructively non-recursive* in B .

Hence, to prove Theorem 1, it is sufficient to describe the instructions (that can be simulated by a Turing machine) for enumerating two languages A and B , and then show the existence of two functions f and g (need not be computable) such that $(\forall x)[f(x) \in A \Leftrightarrow f(x) \in W_x^B]$ and $(\forall x)[g(x) \in B \Leftrightarrow g(x) \in W_x^A]$.

For every Turing machine M with encoding z , there exists a Turing machine M' that, provided with an oracle D , enumerates all the members of W_z^D , as follows: it simulates 1 step of M on all possible inputs of length 1, then simulates 2 steps of M on all possible inputs of length 2, and so on. In the i^{th} stage, it simulates i steps of M on all possible inputs of length i . Whenever an input is accepted in the partial simulation of M , it is written on a separate step. In this way, all elements of W_z^D are enumerated on the tape. We refer to the set of members of W_z^D enumerated within the first n stages of operation of M' as *the partial n -listing of W_z^D* . Thus, computing the partial n -listing of W_z^D is recursive in D . We finally note that if D is finite, then the the partial n -listing of W_z^D can be computed by a total Turing machine, since the members of D can be provided as a part of the finite control, and any oracle call to D can then be answered by this modified machine itself. Infact, for each member of the partial listing, we can enumerate the finite list of calls made to the oracle.

Our construction of A and B involves two identical infinite lists, of the natural numbers in increasing order. We call the lists *A-list* and *B-list* respectively. For explanation's sake, we shall consider these lists to be vertical, *i.e.* we shall have to move *down* to find a larger number. We shall mark some integers on both lists with a *plus* (+) and some integers with a *minus* (−) in the course of the computation. The *plus* marked integers on the *A-list* will constitute A , while those on the *B-list* will constitute B . The *minus* marks will be used to keep temporary stalk of the fact that the marked integer does not belong to A or B , respectively. A *minus* mark can be erased and replaced by a *plus* mark in the course of computation, but our computation will ensure that a *plus* mark is never erased. Our computation proceeds in stages, and each stage alternately focuses on adding a member to A and adding a member to B , respectively.

Define $A_0 = B_0 = \emptyset$, the empty language. Define $A_n = \{x | x \text{ receives a plus mark in the } A\text{-list by the end of stage } n\}$, and $B_n = \{x | x \text{ receives a plus mark in the } B\text{-list by the end of stage } n\}$, for $n = 1, 2, \dots$. Then, $A_0 \subseteq A_1 \subseteq A_2 \subseteq \dots$. Define $A = \bigcup_{n=0}^{\infty} A_n$. Similarly,

$B_0 \subseteq B_1 \subseteq B_2 \subseteq \dots$. Define $B = \bigcup_{n=0}^{\infty} B_n$.

In addition, we shall also have two countably infinite sets of *movable markers*, one set for each list. The markers for the *A-list* are denoted by $0_A, 1_A, 2_A \dots$, and the markers for the *B-list* are denoted by $0_B, 1_B, 2_B$. These markers are introduced in the course of the computation and associated with some integer in their corresponding lists, and they can be associated with any one integer in the list at any point of the computation. They are movable in the sense that they can be moved downwards in the list in the course of computation and made to associate with some larger integer.

The technique that we are using to prove Theorem 1 is often called a *priority argument*. The markers have a specific priority order, which is $0_A, 0_B, 1_A, 1_B \dots$, in decreasing order of priority. This is called a *priority ordering*. The i^{th} marker in this priority ordering is introduced in the i^{th} stage of the algorithm. Also, at all stages of the computation, a marker with higher priority will have a higher position in the list than a marker of lower priority.

We shall show that every marker is moved only a finite number of times. Thus, every marker has a final position in the list to which it is associated. Let x_j be the final position of j_A in the *A-list*, and y_j be the final position of j_B in the *B-list*. We shall show that x_j has a *plus* mark $\Leftrightarrow x_j \in W_j^B$, and y_j has a *plus* mark $\Leftrightarrow y_j \in W_j^A$. Then, $f(j) = x_j$ and $g(j) = y_j$ gives the two required functions, thus proving Theorem 1.

We shall now describe the stages of our computation for enumerating *A* and *B*. At any point of the computation, we say that a number is *free* in a particular list if there is no mark or movable marker associated with it or any integer below it. We say that an integer is *vacant* in a particular list if it does not have a *plus* mark.

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

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

.....

Stage $2n+1$ ($n > 0$):

- Associate n_A with the first free integer in the *A-list*. Let $a_0, a_1 \dots a_n$ be the current positions of $0_A, 1_A \dots n_A$ in the *A-list*.

- Compute the partial n -listing of $W_0^{B_{2n}}, W_1^{B_{2n}} \dots W_n^{B_{2n}}$, and also the calls to B_{2n} made for all members of all these partial listings. This can be done by a Turing machine in finite time, since B_{2n} is finite. Let a_i be the smallest integer among $\{a_0, a_1 \dots a_n\}$ such that a_i is vacant in the A -list (*i.e.* not marked with a *plus*), and a_i is a member of the partial n -listing of $W_i^{B_{2n}}$. If there is no such a_i , skip the remaining of stage $2n + 1$, go on to the next stage $2n + 2$.
- Mark a *plus* on a_i in the A -list, and a *minus* on every integer in the B -list that is not a member of B_{2n} , and was queried to the oracle B_{2n} while checking for membership of a_i in the partial n -listing of $W_i^{B_{2n}}$. Note that, if B_{2n} were expanded to another language C by adding all or some integers to it, except those mentioned above, even then a_i would appear in the partial n -listing of W_i^C .
- Move all markers in the B -list that have a lower priority than i_A , *i.e.* all markers j_B , $i \leq j < n$, down to free integers in the B -list. The new positions of the markers should be such that a marker with higher priority is placed higher in the list.

Stage $2n+2$ ($n > 0$): This step is symmetric to stage $2n + 1$, with the roles of the two lists interchanged, and B_{2n} replaced by A_{2n+1} .

.....

It is easy to see from the description of the algorithm that no *plus* gets erased, but a *minus* can be overwritten by a *plus* mark in a later stage.

We need to show that every marker is moved a finite number of times, so that every marker has a final position. To see this, we observe when an arbitrary marker, say i_A , can be moved. Let $n(x)$ denote the number of distinct positions taken up by a marker x since it was introduced into the computation. Clearly, i_A gets moved every time $(i - 1)_A$ gets moved. If $(i - 1)_A$ doesn't get moved in a particular stage, i_A can still get moved only if $(i - 1)_B$ gets a *plus* mark in that stage. But, $(i - 1)_B$ can get marked as *plus* only once in one of its positions. So, i_A can be moved without moving $(i - 1)_A$ at most $n((i - 1)_B)$ times. Hence, $n(i_A) \leq n((i - 1)_A) + n((i - 1)_B)$. Similarly, we get that $n(i_B) \leq n((i - 1)_B) + n(i_A)$. Let $marker(i)$ denote the i^{th} marker in the priority ordering. Then, these two relations is equivalent to saying that $n(marker(i)) \leq n(marker(i - 1)) + n(marker(i - 2))$.

It is easy to see that 0_1 is never moved, and 0_2 can be moved at most once, when 0_1 gets a *plus* mark. So, $n(0_A) \leq 1$, $n(0_B) \leq 2$.

The Fibonacci sequence is defined inductively as $F_0 = 1$, $F_1 = 1$, and $F_i = F_{i-1} + F_{i-2}$, $\forall i > 1$.

Claim 6. $n(\text{marker}(i)) \leq F_i$, $\forall i > 0$.

Proof. This is easy to prove by induction on i . □

Thus, every marker has a final position on one of the lists. We define $f(x)$ to be the final position of x_A , and $g(x)$ to be the final position of x_B .

Note that our construction itself shows that A and B are recursively enumerable: given an integer x , a Turing machine can keep simulating the stages, and accept x whenever it receives a *plus* mark in the *A-list* and *B-list*, respectively. Our construction, however, does not show A or B to be recursive in any obvious manner, *i.e.* there is no obvious way to decide, for every integer, whether it will be marked with a *plus* in some later stage, if it has not yet received it. Also, $f(x)$ and $g(x)$ are not computable by a total Turing machine in any obvious fashion, since, again, there is no obvious way to decide, for every marker, whether it will be moved in some later stage.

A and B are indeed not recursive, which shall be implied by Theorem 1 (as and when we prove it). $f(x)$ and $g(x)$ will also turn out to be non-recursive, due to the following theorem that we state without proof:

Theorem 2. *Let A and B be recursively enumerable languages, such that A is constructively non-recursive in B . Then, B is recursive.*

Since B will turn out to be nonrecursive, A is not constructively non-recursive in B , and so f cannot be recursive. Similarly, g is also non-recursive.

The key element of the construction is that whenever a marker gets a *plus* on its position, all markers of lower priority *yields*, *i.e.* they are shifted down, so that they cannot affect the *minus* marks that are introduced in this stage. The other key fact is that every marker is introduced after a finite number of stages, and moved finite number of times, ensuring that $f(x)$ and $g(x)$ are well-defined. These two aspects of our construction help us to prove the following lemmas, which immediately imply Theorem 1.

Lemma 7. $f(x)$ has a *plus* mark if and only if $f(x) \in W_x^B$.

Proof. (\Rightarrow) Suppose $f(x)$ has a *plus* mark. Then, $f(x)$ received the *plus* mark at stage $2n + 1$, for some n , which implies that $f(x)$ occurred in the partial n -listing of $W_x^{B_{2n}}$. So, if none of the *minus* marked in this stage gets

overwritten by a *plus* at a later stage, then $f(x)$ will occur in the partial n -listing of W_x^B , and hence would be a member of W_x^B .

To show that none of these *minus* marks get overwritten, we argue that a *minus* mark can be overwritten by a *plus* mark only if a marker gets associated with it at some later stage of the computation. When $f(x)$ is marked with a *plus*, all the markers k_B , $k \geq x$, are moved to free positions, which, by definition, are below all the *minus* marks introduced so far, and so cannot cause any of them to be overwritten. If a *minus* mark introduced in stage $2n + 1$ gets overwritten by a *plus* mark in stage $2m$ for some marker k_B , $k < x$, then x_A is moved in stage $2m$, contradicting the fact that $f(x)$ is the final position of x_A .

(\Leftarrow) Suppose $f(x) \in W_x^B$. Then, $f(x)$ gets enumerated as a member of W_x^B at some finite point, say, in the partial m_1 -listing of W_x^B . There are finite number of integers $y \in B$, such that y was queried to the oracle B while checking for membership of $f(x)$ in W_x^B . So, there exists m_2 such that B_{2m_2} contains all such integers y . Let $m = \max(m_1, m_2)$. Then, $\forall n \geq m$, $f(x)$ occurs in the partial n -listing of W_x^{2n} .

We need to prove that $f(x)$ is marked *plus* at some stage. Since $f(x)$ is the final position of x_A . Let stage $2l + 1$ be such that $l > m$, and all markers k_A , $k \leq x$ have reached their final positions $f(k)$, respectively. Then, either $f(x)$ must receive a *plus* mark by the end of stage $2(l + x + 1) + 1$. This is because until $f(x)$ gets marked with *plus*, $f(x)$ satisfies the dual criteria of being vacant and appearing in the partial listing in every stage after stage $2l + 1$ (since $l > m$), and hence is not marked with a *plus* only if the position of some marker with a lower index satisfies the dual criteria. But there can be at most x such markers, and since each of them has reached their final positions, this can happen for at most x more stages, after which $f(x)$ will get a *plus* mark. \square

Lemma 8. $g(x)$ has a *plus* mark if and only if $g(x) \in W_x^A$.

Proof. Identical to the proof of Lemma 7. \square

Lemmas 7 and 8 are precisely the results that were required to complete the proof of the Friedberg-Muchnik Theorem, which solves Post's problem.

8 Stronger Results

After Post's problem was solved, several variations of the priority argument was used to prove the existence of numerous recursively enumerable

T-degrees.

It is easy to show, by a similar argument that there are three recursively enumerable T-degrees that are incomparable with each other. We use 3 infinite lists, one for each language A_1 , A_2 , and A_3 , and 6 collection of markers: n_{ij} , $1 \leq i \leq 3$, $1 \leq j \leq 3$, $i \neq j$, $n = 0, 1, 2, \dots$. n_{ij} is used to mark positions on list i . The markers have an appropriate priority order so that x_{ij} has a lower priority than y_{kl} if $x < y$. At stage $3n + i$ ($i = 1, 2, 3$), a *plus* is put on the position x of a marker k_{ij} on the i^{th} list, that is vacant and x appears in the partial n -listing of $W_k^{A_j^n}$, where A_j^n is the set of positions already marked *plus* in the j^{th} list. Of course, all markers in the other lists, that have a lower priority than k_{ij} are move down to free integers in their respective lists.

Again, it is easy to see that every marker moves a finite number of times. (The first marker of every set can move only a finite number of times, and a loose upper bound for the number of positions taken by any other marker x is the sum of the number of positions taken by markers with a higher priority than x .) The final positions of the markers x_{ij} give the function $f_{ij}(x)$, which satisfy the required fact: $f_{ij}(x)$ has a *plus* if and only if $f(x) \in W_x^{A_j}$. Hence, f_{ij} shows that $A_i \not\leq_T A_j$. Thus, A_1, A_2, A_3 are incomparable with each other.

The above argument can be easily modified to show that there exists n pairwise incomparable recursively enumerable T-degrees, for every n . Infact, a further modification allows us to show that there is a countably infinite set of pairwise incomparable, recursively enumerable T-degrees. We now have an infinite sequence of lists, and a set of markers for each pair of lists. The entire set of markers is still countable, and we can enumerate them to create a priority ordering, and introduce the markers into their assigned lists in that order. Every marker still gets introduced at a finite stage, and moved only a finite number of times.

Thus, we have the following result, which is much stronger than Post's problem:

Theorem 3. *There exists a countably infinite family of recursively enumerable Turing degrees such that any two members of the family are incomparable with respect to \leq_T . Thus, there exists \aleph_0 distinct recursively enumerable Turing degrees.*

Now that we know that the poset of recursively enumerable T-degrees is an infinite one, the natural question to ask is what type of structures can be found in the poset. Sacks proved the following result, again a strong one:

Theorem 4 (Sacks). *Let Π be a countable partially ordered set, and let T be a recursively enumerable T -degree that is not equal to 0 . Then there exists a collection of recursively enumerable T -degrees, all of which are T -reducible to T , and the partial order, induced by \leq_T , on which is isomorphic to Π .*

In other words, all possible countable structures are present in the poset of recursively enumerable T -degrees. In particular, there exists a set of recursively enumerable T -degrees which are isomorphic to the rationals, *i.e.* the degrees are linearly ordered by \leq_T , and the order type is that of the rationals.

The following theorem is also due to Sacks:

Theorem 5. *Let T_1 and T_2 be recursively enumerable T -degrees such that $T_1 <_T T_2$. Then, there exists a recursively enumerable T -degree T_3 , such that $T_1 <_T T_3 <_T T_2$.*

Theorem 5 implies that there is no maximal element in the poset $\{T|T \text{ is a recursively enumerable } T\text{-degree, and } T <_T TJ(0)\}$.