PRIORITY METHODS IN COMPUTABILITY THEORY

KEYER THYME

Abstract. This paper introduces the use of priority methods in computability theory, with a focus on their application to Post's Problem of the existence of intermediate c.e. degrees. We will first see priority methods as applied to the proofs of Friedberg's Splitting Theorem and Post's construction of a simple set. We will then see how finite injury priority methods are used in the proofs for the existence of incomplete simple sets as well as the proof for the Friedberg-Muchnik Theorem, which are solutions to Post's Problem.

Contents

1. Introduction 1
1.1. Basic Definitions 2
1.2. The Halting Problem 3
2. Turing Degrees 4
3. Friedberg's Splitting Theorem 5
4. Simple Sets 7
5. Introduction to Finite Injury 9
6. Incomplete Simple Sets 10
7. The Friedberg-Muchnik Theorem 13
Acknowledgements 15
References 15

1. Introduction

In a 1944 paper studying the computably enumerable (c.e.) sets, Emil Post was able to conclude the existence of two types of c.e. sets: those which are *computable*, and those which are *complete*. More precise definitions of these concepts will be given later, so we provide an intuitive explanation for now. The *computable* sets are "decidable," meaning that one can tell unaided whether an element belongs to the set or not. These sets are also considered to carry no information regarding other c.e. sets. On the other hand, the *complete* sets are "undecidable," and we shall see in Section 2 that information regarding any c.e. set can be derived from a complete set. Post termed this way of classifying sets "degrees of unsolvability," or Turing degrees. These two degrees of unsolvability represent the floor and ceiling of how much information a c.e. set may contain. In the same paper, Post posed the question of whether there exist c.e. sets that are more complex than the computable sets, but not as complex as the complete sets. The original *"Post's Problem"* was posed as follows (note that "recursive" is equivalent to "computable.") [1]:

"We are left completely on the fence as to whether there exists a recursively enumerable set of positive integers of absolutely lower degree of unsolvability than the complete set K, or whether, indeed, all recursively enumerable sets of positive integers with recursively unsolvable decision problems are absolutely of the same degree of unsolvability."

Attempts to solve the problem motivated the development of the *priority methods*. In this paper, we shall see how priority methods are employed as a means of constructing sets in stages, in accordance with a sequence of requirements. These requirements are ordered by priority, where we let the fulfillment of some requirements take precedence over that of others. Post himself first employed a priority method in constructing a simple set (and also hypersimple sets and hyper-hypersimple sets), and later along with Stephen Kleene used a priority method in constructing an incomplete and noncomputable set. However, the simple set was not necessarily incomplete, and the set in the Kleene-Post construction was not specifically c.e., and so Post's Problem remained unsolved.

A few years later, Friedberg and Muchnik improved upon the work of Kleene and Post and produced a solution to Post's Problem. The Friedberg-Muchnik Theorem introduced the *finite injury priority methods*. After a requirement is satisfied, it may again become unsatisfied–which we call *injured*–due to an action later taken by a higher-priority requirement. The injured requirement needs to be satisfied again, and can possibly become injured again, which repeats for a finite number of times. Eventually, the strategy undertaken to fulfill the requirement is completed. On the other hand, the *infinite injury priority methods* present a more complicated situation. This type of method is briefly described in Section 5, but otherwise lies outside of the scope of this paper.

Finally, it should be noted that this paper does not demonstrate the full versatility of priority methods. Priority methods are variously applied to problems in computability which are not related to the attempts at solving Post's Problem, and we shall see just one example of that in the proof for Friedberg's Splitting Theorem in Section 3. Readers who are interested may consult [2] or [3] for more examples of how priority methods may be applied.

The rest of this section remarks briefly upon the c.e. sets and the halting problem. Readers seeking a more complete introduction should consult texts such as [2] and [4]. We then introduce the oracle Turing machines, which are used to characterize Turing degrees. Friedberg's Splitting Theorem is used as an example of a priority method without injury. The existence of simple sets is then proven, and we see how they fall just short of solving Post's Problem. Finally, three methods of solving Post's problem using the finite injury priority method are presented: the existence of two types of incomplete simple sets (Theorem 6.2 and Theorem 6.4), and the Friedberg-Muchnik Theorem (Theorem 7.1).

1.1. **Basic Definitions.** By the *Church-Turing Thesis*, an "effective procedure" is one which can be carried out by a Turing machine. A Turing program is a finite set of instructions for a Turing machine, which reads and writes numbers on a tape, and moves along said tape. A machine's input refers to numbers written on its tape at the beginning of a computation, and output refers to numbers produced at the end of the computation. Although a Turing machine is a mathematical concept, the thesis equates its capability with anything intuitively calculable by a human. Often, "algorithm," "program" and "machine" are used interchangeably.

With a given input written on its tape, a machine either halts with some output after a finite number of steps, or it never halts. A *partial computable* (or just *computable*) function is identified with a program that halts on a given input *iff* the function is defined on said input. The *total computable* functions are a subset of the partial computable functions, and they are defined everywhere on $\mathbb{N}$. Effective listing refers to the existence of a computable function which enumerates the elements in a list. Since each program is made of finite instructions, they may be effectively listed through Gödel numbering. Similarly, the computable functions can be found in a list–$\{\psi_e\}_{e\in\mathbb{N}}$. If the e^{th} program on input x halts, we write $\psi_e(x)\downarrow$. To indicate that the output is y, we may write $\psi_e(x) = y$. If the program does not halt, we write $\psi_e(x)\uparrow$.

A set $A \in \mathbb{N}$ is *computably enumerable (c.e.)* if there exists a partial computable function ψ_e such that $\psi_e(n)\downarrow$ *iff* $n \in A$. The c.e. sets may be effectively listed, and the e^{th} c.e. set is denoted $W_e = \{n \mid \psi_e(n)\downarrow\}$. In Section 3, we will begin to think of A *dynamically* by saying that it is *listable*–elements can be one-by-one enumerated into A by a total computable function. A set $A \in \mathbb{N}$ is *computable* (or "decidable") if there exists an algorithm for deciding whether an element is in the set or not. This can be formalized by saying that a set is computable if its characteristic function is (total) computable. The characteristic function of A is a total function defined as follows:

$$A(x) = \begin{cases} 1 & if\, x \in A \\ 0 & if\, x \notin A \end{cases}$$

Equivalently, A is computable *iff* both A and $\overline{A}$ (the complement of A) are c.e.

1.2. **The Halting Problem.** We now introduce a theorem which characterizes an inherent limitation of Turing machines.

Theorem 1.1. *There is no algorithm to determine whether a given machine halts on a given input. In other words, the halting problem is undecidable.*

Proof. Suppose in contrary that there exists a total computable function h which corresponds to the algorithm. We define h as follows:

$$h(e,x) = \begin{cases} 1 & if\, \psi_e(x)\downarrow \\ 0 & if\, \psi_e(x)\uparrow \end{cases}$$

Now, use h to define a partial computable function g as follows:

$$g(n) = \begin{cases} 1 & if\, h(n,n) = 0 \\ \text{undefined} & if\, h(n,n) = 1 \end{cases}$$

Let y be the Gödel number for g, so that $g(y) = \psi_y(y)$. By definition of g, if $h(y,y) = 0$, then $\psi_y(y) = 1$. By definition of h, we have instead that $h(y,y) = 0$ implies $\psi_y(y)\uparrow$. Similarly, if $h(y,y) = 1$, g gives that $\psi_y(y)\uparrow$, whereas h implies that $\psi_y(y)\downarrow$. This generates a contradiction, and h is thus not well-defined. □

Definition 1.2. The set $\mathcal{K} = \{\langle e,x\rangle | \psi_e(x)\downarrow\}$ is called the halting set.

Here, $\langle x,y\rangle$ denotes the standard pairing function, which is a 1-1 map taking an ordered pair of natural numbers (x,y) to a natural number.

Theorem 1.3. $\mathcal{K}$ is c.e. and noncomputable.

Proof. Suppose $\mathcal{K}$ is computable. Then the function $h(e, x)$ in Theorem 1.1 is computable. Therefore, $\mathcal{K}$ is noncomputable. To show that $\mathcal{K}$ is c.e., define the following partial computable function:

$$d(\langle e, x\rangle) = \begin{cases} 1 & if\ \psi_e(x)\downarrow \\ \text{undefined} & if\ \psi_e(x)\uparrow \end{cases}$$

$\mathcal{K}$ may be enumerated as the domain of the function d. □

2. Turing Degrees

A Turing degree is an equivalence class of sets which helps to characterize the information contained in a set relative to other sets. Before understanding Turing degrees, we must first define the oracle machine. An *oracle machine* is a Turing machine with two tapes: the original "work tape" and an additional "oracle tape" whose cells are imprinted with the characteristic function of some set A, which is referred to as the *oracle*. The oracle machine reads and writes on the "work tape," while the contents of the oracle tape are read but never modified. We may think of reading the oracle tape as the machine querying whether a certain number is an element of A. Like a regular Turing machine, the oracle machine produces an output on the worktape, but its computation process is aided by the knowledge of what numbers are in the oracle.

The e^{th} *Turing functional* is denoted $\Phi_e^A(x)$, corresponding to an oracle machine with oracle A and input x on its work tape. If the oracle machine halts and outputs y, we write $\Phi_e^A(x) = y$. The *use function* is denoted by $\psi_e^A(x)$. If the oracle machine halts, $\psi_e^A(x)$ is the least n such that all the numbers queried for membership in A in the computation with input x are less than n. If no numbers are queried, we let $\psi_e^A(x) = 0$. If the oracle machine halts in s steps, we may write $\Phi_{e,s}^A(x)\downarrow$ and let $\psi_{e,s}^A(x)$ denote the use. It is assumed that $\psi_{e,s}^A(x) \leq s$.

With the help of an oracle machine, we may derive information about a set from another set (i.e. the oracle set). This is the basic idea behind Turing reducibility and Turing degrees, which we begin to define more formally below.

Definition 2.1. A partial function f is *computable in* A (written $f \leq_T A$) if there is an e such that $\Phi_e^A(x) = y$ *iff* $f(x) = y$. A set B is *computable in* A, or *Turing reducible* to A, *iff* the characteristic function of B is computable in A. Symbolically, we write $B \leq_T A$.

Definition 2.2. If $B \leq_T A$ and $A \leq_T B$, then $A \equiv_T B$. The *Turing degree* of A is the equivalence class defined as $\{B | B \equiv_T A\}$.

Definition 2.3. The *Turing jump* of the set A is $A' = \{\langle e, x\rangle | \Phi_e^A(x)\downarrow\}$.

Notice that this is very similar to the set $\mathcal{K}$ in Definition 1.2, just with the addition of an oracle. We can say that $\mathcal{K}$ is the jump of $\emptyset$, defined using $\emptyset$ as the oracle ($\mathcal{K} \equiv \emptyset'$). Using the above definitions, we may rephrase the noncomputability of $\mathcal{K}$ as $\emptyset' \not\leq_T \emptyset$. In fact, the Turing jump allows us to define the *relativized* halting problem using an oracle A:

Theorem 2.4. *For any set A, $A' \not\leq_T A$.*

Proof. The proof of the theorem proceeds entirely analogous to the proof of Theorem 1.1, where every $\psi_e(x)$ is replaced by $\Phi_e^A(x)$. □

We now return our attention to the sets $\emptyset'$ and $\emptyset$. As we shall see, they are very relevant to our discussion of c.e. sets.

Theorem 2.5. *All c.e. sets are Turing reducible to $\emptyset'$.*

Proof. If we have $\emptyset'$ as the oracle, then we can ask it whether any program halts on a given input. Given any c.e. set A, we may ask the oracle whether the corresponding program of A halts on input x. If the answer is yes ($\Phi^{\emptyset'}(x) = 1$), then we know $A(x) = 1$. If the answer is no ($\Phi^{\emptyset'}(x) = 0$), then we know $A(x) = 0$. The characteristic function of A is thus computable in $\emptyset'$. Therefore, A is computable in $\emptyset'$ by Definition 2.1. □

Definition 2.6. A c.e. set A is *complete* if every c.e. set is Turing reducible to it.

Theorem 2.7. *If a c.e. set A is complete, then $A \equiv_T \emptyset'$.*

Proof. Since $\emptyset'$ is c.e., we have by Definition 2.6 that $\emptyset' \leq_T A$. Since A is c.e., we have by Theorem 2.5 that $A \leq_T \emptyset'$. By Definition 2.2, $A \equiv_T \emptyset'$. □

Theorem 2.8. *If a c.e. set A is computable, then $A \equiv_T \emptyset$.*

Proof. To say that a set is computable is equivalent to saying that it is computable without the need for an oracle; that is, the set is computable in $\emptyset$. Therefore, we have $A \leq_T \emptyset$. A computable set is in fact computable in any other set, where the use of the computation is simply 0. Since the empty set is defined to be computable, $\emptyset \leq_T A$. By Definition 2.2, $A \equiv_T \emptyset$. □

From Theorems 2.7 and 2.8, we may see that $\emptyset$ and $\emptyset'$ represent two Turing degrees for c.e. sets. Post's Problem then follows naturally: is there an intermediate Turing degree for c.e. sets between $\emptyset$ and $\emptyset'$? That is, does there exist a set which is c.e., not computable, and not complete? Three solutions to this problem are presented in Sections 6 and 7.

3. Friedberg's Splitting Theorem

So far, we have considered c.e. sets as "static" objects, without touching on the significance of their listability as foreshadowed in their definition in Section 1.1. We now begin looking at c.e. sets *dynamically*, by considering the real-time process of how the sets are listed. That is, we can construct A in stages, where A_s denotes A at the end of stage s. We may define a *computable enumeration* $\{A_s\}_{s\in\mathbb{N}}$, and let $A = \cup_s A_s$ at the end of the construction. The enumeration guarantees that A is c.e., and allows us to induce certain dynamic properties for the set. In this section, we will construct two sets stage by stage. At each stage, at most one set gains at most one element. We let $W_{e,s}$ denote the c.e. set W_e at the end of stage s.

An example of a priority method construction is used in the proof for Friedberg's Splitting Theorem. We first need to define a set of *requirements*, $\{R_e\}_{e\in\mathbb{N}}$, which are rules that stipulate the properties of the sets at the end of the construction. These requirements are ordered by priority, where R_i has *higher priority* than R_j if $i < j$. We need to then define strategies for satisfying the requirements. These strategies determine if and when an element enters a set. At the beginning of each stage, we look at all the requirements that need to be satisfied, and choose the highest-priority one to satisfy.

A *split* of a c.e. set A is the separation of A into a pair of c.e. sets whose disjoint union is the whole set. The Friedberg split is a particular type of split that has many applications, but the theorem is introduced in this paper mainly for the study of its proof. For more in-depth discussions of splitting theorems including the Friedberg split, please consult [5] or [6].

Definition 3.1. The disjoint union $A \sqcup B$ denotes $(A \cup B) - (A \cap B)$. A split $S_0 \sqcup S_1 = A$ is a *Friedberg split iff* for all c.e. W, if $W - A$ is not c.e., then both $W - S_0$ and $W - S_1$ are not c.e.

The following definitions concern the *dynamic properties* of sets. The proof for Friedberg's Splitting Theorem is an example of utilizing dynamic properties that arise *during* the process of enumeration in order achieve the desired properties at the end of the enumeration.

Definition 3.2. $A\backslash B = \{x|\exists s[x \in (A_s - B_s)]\}$. This is the set of numbers which are enumerated in A before possibly being enumerated in B. Note that this depends upon the particular enumerations chosen for the sets.

Notice that if both A and B are c.e., $A\backslash B$ is also c.e. via the enumeration process described in the definition above. We find an x and wait for it to appear in A, if ever. If this happens and x has not appeared in B yet, we know that $x \in A\backslash B$.

Definition 3.3. $A \searrow B = (A\backslash B) \cap B$. This is the set of numbers which are enumerated in A and then enumerated in B.

Lemma 3.4. $A\backslash B = (A \searrow B) \cup (A - B)$.

Proof. We first interpret this intuitively. We may classify the numbers in $A\backslash B$ into two groups: the numbers that eventually enter B, and the numbers that never enter B. Notice that these two groups are precisely $A \searrow B$ and $(A - B)$.

We may also attempt a proof symbolically, as follows:

$$A\backslash B = ((A\backslash B) \cap B) \cup ((A\backslash B) - B)$$
$$A\backslash B = (A \searrow B) \cup ((A - (B\backslash A)) - B)$$
$$A\backslash B = (A \searrow B) \cup (A - B)$$

□

Theorem 3.5 (Friedberg's Splitting Theorem)**.** *Every non-computable c.e. set A has a Friedberg split.*

Proof. When an element enters A at stage $s + 1$, it is added to either S_0 or S_1, as determined by the index of the associated requirement. The requirements are defined as follows:

$$R_{\langle e,i\rangle} : \text{If } W_e \searrow A \text{ is infinite, then } W_e \cap S_i \neq \emptyset.$$

Let us show that the requirements sufficiently guarantee a Friedberg split. Let W be some c.e. set where $W - S_i$ *is* c.e. Suppose that $R_{\langle e,i\rangle}$ is met for $W_e = W - S_i$. By Lemma 3.4, we have that $(W - S_i)\backslash A = [(W - S_i) \searrow A] \cup [(W - S_i) - A]$. Since both $W - S_i$ and A are c.e., we have that $(W - S_i)\backslash A$ is c.e., as explained after Definition 3.2. Since $(W - S_i) \cap S_i = \emptyset$, the requirement gives that $(W - S_i) \searrow A$ is finite. This implies that $(W - S_i) - A = W - A$ is also c.e.

We now describe how S_0 and S_1 are constructed.

Construction.
Stage 0: Put the first element of A into S_0.
Stage $s+1$: Search for the least $\langle e,i\rangle < s$ such that $W_{e,s} \cap S_{i,s} = \emptyset$, and $(\exists x)[x \in W_{e,s} \cap (A_{s+1} - A_s)]$. If such $\langle e,i\rangle$ exists, enumerate the corresponding x into S_i. We say that $R_{\langle e,i\rangle}$ *acts* at stage $s+1$. Otherwise, enumerate x where $x \in A_{s+1} - A_s$ into S_0. Notice that S_0 and S_1 will be a split of A.

Let us show that $\{R_{\langle e,i\rangle}\}$ are satisfied. Notice that the condition $W_{e,s} \cap S_{i,s} = \emptyset$ will not be met again after $R_{\langle e,i\rangle}$ acts. We therefore say each requirement acts at most once. Let s be a stage by which all requirements $\{R_{\langle f,j\rangle}\}$ where $\langle f,j\rangle < \langle e,i\rangle$ have acted. If $W_e \searrow A$ is infinite, there will be a stage $t > s$ such that $W_{e,t} \cap (A_{t+1} - A_t)$ contains an $x \notin S_{i,t}$. $R_{\langle e,i\rangle}$ then acts by enumerating x. □

This is an example of a priority method *without injury*. There is only one type of requirement, and each requirement remains satisfied after it acts. We continue with examples of priority methods without injury in the following section.

4. Simple Sets

Definition 4.1. A set A is *simple iff* A is c.e., $\overline{A}$ is infinite, and $\overline{A}$ does not contain an infinite c.e. set.

The simple sets point to a possible solution to Post's Problem. Since their complements cannot be c.e., the simple sets are noncomputable. The simple sets thus satisfy two criteria–they are c.e. and noncomputable. However, as we shall see in Theorem 4.4, the simple sets are not necessarily incomplete.

The first proof of the existence of a simple set was proposed by Emil Post, and the proof makes use of a priority method. For an alternative proof called the "canonical construction" which makes use of a different priority method, please see Section 5.2.2 of [2].

Theorem 4.2 (Post)**.** *There exists a simple set.*

Proof. We first define the requirements $\{R_e\}_{e\in\mathbb{N}}$.

$$R_e : \text{If } W_e \text{ is infinite, then } W_e \cap A \neq \emptyset.$$

Construction.
Stage 0: Let $A_0 = \emptyset$.
Stage $s+1$: Search for the least $e < s$, if it exists, such that $W_{e,s} \cap A_s = \emptyset$, and $(\exists x)[x \in W_{e,s} \quad \text{and} \quad x > 2e]$. Enumerate the least such x into A_{s+1}. If this happens, then we say R_e acts at stage $s+1$. Otherwise, let $A_{s+1} = A_s$ and proceed to stage $s+2$.

Now, we can show that $A = \cup_s A_s$ satisfies Definition 4.1 for simplicity. We first define a notation that will simplify the explanations to follow.

Definition 4.3. Let $A \upharpoonright x$ denote all the elements in A up to and including x.

We know A is c.e. by the construction described above, since elements are enumerated into A. Suppose that all requirements R_i where $i < e$ have acted by stage s, and R_e is still unsatisfied ($W_{e,s} \cap A_s = \emptyset$). R_e is now the highest-priority requirement that has yet to act. If W_e is infinite, it will contain a number $x > 2e$. When x enters W_e at stage t, R_e acts by enumerating x into A_{t+1}, and we have that $W_e \cap A \neq \emptyset$. Therefore, R_e is satisfied and remains satisfied. When all $\{R_e\}$ are

satisfied, we have that $W_e \cap A \neq \emptyset$ for all e where W_e is infinite. Therefore, $\overline{A}$ cannot contain any infinite c.e. set W_e. Finally, only requirements R_i where $i < e$ may enumerate numbers $x \leq 2e$ into A. Thus, each set $\{0, 1, ..., 2e\}$ of $2e + 1$ elements will contain at most e elements which are also in A. For every e, $|\overline{A} \upharpoonright 2e| \geq e + 1$. Therefore, $\overline{A}$ is infinite. □

In the following theorem, we shall see that the simplicity of a set does not provide any more information on its degree other than that it is noncomputable and c.e.

Theorem 4.4. *For any noncomputable c.e. set C, we can construct a simple set $A \leq_T C$.*

Proof. First, we need to prove a lemma that will make clear the goal of our strategy.

Definition 4.5. The function $m_A(x) = (\mu s)[A_{s+1} \upharpoonright x = A_s \upharpoonright x]$ is called the *least modulus* (where the μ operator searches for the least such s). For a c.e. set A, $m_A(x)$ is the least stage at which all numbers less than or equal to x have entered the set.

Lemma 4.6. *For c.e. sets A and C, if $m_A(x) \leq m_C(x)$, then $A \leq_T C$.*

Proof. By Definition 4.5, $A \upharpoonright x = A_{m_A(x)} \upharpoonright x$. Since $m_C(x) \geq m_A(x)$, we also have $A \upharpoonright x = A_{m_C(x)} \upharpoonright x$. This implies that $A(x) = A_{m_C(x)}(x)$. Given C, we can compute $m_C(x)$, which can then be used to compute $A(x)$. □

We can now begin the proof of Theorem 4.4. We fix a noncomputable c.e. set C with a computable enumeration $\{C_s\}_{s\in\mathbb{N}}$. We will try to satisfy the antecedent of Lemma 4.6 by constructing A simultaneously with C. We define requirements that are slightly modified from those found in the proof for Theorem 4.2.

$$R_e : \text{If } W_e \text{ is infinite, then } W_e \cap A \neq \emptyset \text{ or } C \equiv_T \emptyset.$$

Suppose that all requirements are satisfied. Since we have stipulated as a premise that C is noncomputable, then it must be the case that $W_e \cap A \neq \emptyset$ for all e where W_e is infinite, thus satisfying the simplicity requirement for A. These requirements are therefore sufficient.

Construction.
Stage 0: $A_0 = \emptyset$
Stage $s + 1$: Search for the least $e < s$, if it exists, such that $W_{e,s} \cap A_s = \emptyset$, and $(\exists x)[x \in W_{e,s}$ and $x > 2e$ and $C_{s+1} \upharpoonright x \neq C_s \upharpoonright x]$. Enumerate the least such x into A_{s+1}. Otherwise, let $A_{s+1} = A_s$ and proceed to stage $s + 2$.

This construction contains all the conditions for constructing a simple set in Theorem 4.2. The additional condition stipulates that if every number less than or equal to x has already entered C by stage s, then x cannot enter A at stage $s + 1$. This guarantees that in case $m_C(x) = s$, we have that $m_A(x) \leq s$. Since $m_A(x) \leq m_C(x)$, we invoke Lemma 4.7 to conclude that $A \leq_T C$.

Suppose that W_e is infinite but R_e never enumerates anything. For any x, we can wait for a $y > x$ to enter W_e at stage s such that $y > 2e$. However, y cannot be enumerated at stage $s + 1$ since $C \upharpoonright y = C_s \upharpoonright y$. Therefore, x is in C *iff* x is in C_s, which means that C is computable. Otherwise, R_e is able to act, and so $W_e \cap A \neq \emptyset$. The requirements are therefore satisfied. □

5. Introduction to Finite Injury

The priority methods shown so far have been without injury, since the requirements cannot be undone by one another. In *finite injury* priority methods such as the ones employed in Section 6, we will see two types of requirements: *positive* and *negative*. If a negative requirement which has already acted is later contradicted by a higher-priority requirement, we allow the lower-priority requirement to be *injured* and reset it. We shall see in Section 7 that making the distinction between "positive" and "negative" is not in fact necessary. Nevertheless, we begin with these definitions to ease the introduction.

The positive requirements $\{P_e\}_{e\in\mathbb{N}}$ of a finite injury method are similar to the requirements seen in the priority methods without injury. Each positive requirement attempts to enumerate a single element into a set at a given stage, and cannot be injured after it acts.

To satisfy a negative requirement N_e at stage s of the construction, we need to define a *restraint function* $r(e, s)$. The restraint attempts to prohibit numbers x where $x \leq r(e, s)$ from entering the set A at stage $s + 1$. If such an x is nevertheless enumerated into A_{s+1}, we say that N_e is injured. We define the *injury set* $I_e = \{x|(\exists s)[x \leq r(e, s) \quad \text{and} \quad x \in A_{s+1} - A_s]\}$. A negative requirement may or may not also attempt to enumerate elements into a set.

On the other hand, the *infinite injury* priority methods are more complicated. Some requirements of an infinite injury priority method may attempt to enumerate all but finitely many elements of W_e into a set, which means that these requirements act infinitely often. In this case, we need to define restraints for the lower-priority requirements so that they avoid being injured infinitely often. However, some requirements do not act infinitely often, and thus demand strategies from the lower-priority requirements different from those of the infinitary case. Therefore, the strategy for fulfilling each lower-priority requirement may "branch" into several (possibly infinitely many) strategies at each higher-priority requirement, depending upon its behavior. For a full treatment of how to employ "priority trees" in infinite injury priority methods, please refer to Section 2.14 of [3].

In the following sections, we will see three theorems (Theorems 6.2, 6.4, 7.1) that solve Post's Problem using finite injury priority methods. We will proceed with constructions that differ somewhat in format from those of the no-injury methods. We first define at the start of a given stage *witnesses* for requirements that need witnesses but do not currently have any. (Note that Theorem 6.2 will not have requirements that need witnesses.) The witness is a number which has not yet been used in the construction, and also "large enough" to bypass restraints set so far by higher-priority requirements, if applicable. We say a requirement *requires attention* when a certain set of conditions is met, if ever. For a requirement with a witness, the conditions pertain to the witness. Then, the highest-priority requirement that needs attention is chosen to *act*. Action may involve enumerating a number into a set or defining a restraint, cancelling the witnesses of lower-priority requirements, and injuring lower-priority (negative) requirements by cancelling their restraints. If a requirement is injured, it needs to later repeat the process. Since any requirement can only be injured a finite number of times, it stops acting and remains satisfied after a certain stage.

6. Incomplete Simple Sets

We begin our study of incomplete simple sets with a special case of simple sets: the *low simple sets*.

Definition 6.1. A set A is *low* if $A' \equiv_T \emptyset'$.

Intuitively, a set A is low if the halting problem with the aid of an oracle A is of the same difficulty as the halting problem without an oracle. A low set has important properties of its own, but this paper is primarily concerned with the existence of a set that is both low and simple.

By its simplicity, a low simple set is c.e. and noncomputable. For any set A, we know that $A \leq_T A'$ because A' encodes the set of all machines that halt given A, and the algorithm that enumerates A itself must be among them. Further, since $A' \not\leq_T A$ by Theorem 2.4, we conclude that $A <_T A'$. Lowness of the set implies that $A' \leq_T \emptyset'$. Therefore, $A <_T A' \leq_T \emptyset'$. This means that A is incomplete. The existence of a low simple set therefore provides a solution to Post's Problem.

To show that A is low, we need to show that both $\emptyset' \leq_T A'$ and $A' \leq_T \emptyset'$. The former direction follows from the fact that $\emptyset \leq_T A$, and the fact that the Turing jump preserves Turing reducibility by the Jump Theorem (see [2] Section 3.4.2). Since this direction does not provide the necessary property for incompleteness, we do not further elaborate on it. The latter direction will be satisfied by the requirements that guide the construction of A.

Theorem 6.2. *There exists a low simple set.*

The following proof builds upon Post's simple set construction in Theorem 4.2, with the addition of negative requirements.

Proof. We define two types of requirements, $\{P_e\}_{e\in\mathbb{N}}$ and $\{N_e\}_{e\in\mathbb{N}}$. For the same e, let N_e take precedence over P_e.

$$P_e : \text{If } W_e \text{ is infinite, then } W_e \cap A \neq \emptyset.$$

$$N_e : \text{If there exist infinitely many } s \text{ such that } \Phi_{e,s}^{A_s}(e)\downarrow, \text{ then } \Phi_e^A(e)\downarrow.$$

The requirements $\{P_e\}$ are the familiar ones which guarantee the simplicity of A in Theorem 4.2. They are positive and attempt to enumerate elements into A. The requirements $\{N_e\}$ are negative and guarantee the lowness of A. Their significance will be further explained after the construction.

Construction.

Stage 0: Let $A_0 = \emptyset$.

Stage $s+1$. *Attention.* We say N_e requires attention if $\Phi_{e,s}^{A_s}(e)\downarrow$. We say P_e requires attention if $W_{e,s} \cap A_s = \emptyset$, and there exists an $x \in W_{e,s}$ such that x is "large enough." This means that $x > 2e$ and $(\forall i \leq e)[x > r(i,s)]$.

Action. Find the least $e < s$ such that the requirement (either P_e or N_e) requires attention.

To satisfy P_e, enumerate the least applicable x into A_{s+1}. Undefine $r(i, s+1)$ for $i > e$. Maintain $r(j, s+1) = r(j, s)$ for $j \leq e$. To satisfy N_e, define $r(e, s+1) = \psi_{e,s}^{A_s}(e)$.

We now explain how each type of requirement achieves its goal, and how it is satisfied by the construction.

In order to further discuss the sufficiency of $\{N_e\}$, we first define Shoenfield's Limit Lemma.

Lemma 6.3 (Limit Lemma)**.** *For any set C, $C \leq_T B'$ iff there exists a function $C(x,n) \leq_T B$ such that $C(x) = \lim_{n\to\infty} C(x,n)$.*

Proof. Please consult Chapter 6 of [7]. □

We want to show that $A' \leq_T \emptyset'$. Comparing this to Lemma 6.3, we need to define a function $A(x,s) \leq_T \emptyset$ (i.e. a computable function) such that $A'(x) = \lim_{s\to\infty} A(x,s)$. We define the computable function as follows:

$$A(e,s) = \begin{cases} 1 & \Phi_{e,s}^{A_s}(e)\downarrow \\ 0 & \text{otherwise} \end{cases}$$

The function outputs 1 if e appears to be in A' by stage s, and 0 otherwise. This should look similar to a characteristic function. Indeed, if all $\{N_e\}$ are satisfied, then $\lim_{s\to\infty} A(e,s) = A'(e)$ for all e. By Lemma 6.3, $A' \leq_T \emptyset'$.

Now, let us show that the requirements $\{N_e\}$ are satisfied.

Suppose as an inductive hypothesis that all higher-priority requirements have been satisfied by the end of stage s, and N_e is now the highest priority requirement that has yet to act. We wait for a stage $t > s$ where $\Phi_{e,t}^{A_t}(e)\downarrow$. If there exist infinitely many stages s such that $\Phi_{e,s}^{A_s}(e)\downarrow$, then t will arrive. Otherwise, N_e is automatically satisfied. Given that $\Phi_{e,t}^{A_t}(e)\downarrow$, we need to make sure that the beginning of A up to the use at stage t remains unchanged. At stage $t+1$, we define the restraint function $r(e,t+1) = \psi_{e,t}^{A_t}(e)$. In order to circumvent this restraint, any still unsatisfied P_i where $i \geq e$ will only possibly enumerate $x > \psi_{e,t}^{A_t}(e)$. Since no requirement which has yet to act will enumerate below the use, we have that $A_t \upharpoonright \psi_{e,t}^{A_t}(e) = A \upharpoonright \psi_{e,t}^{A_t}(e)$. Therefore, $\Phi_{e,t}^{A_t}(e) = \Phi_e^A(e)$, and $\Phi_e^A(e)\downarrow$.

The positive requirements $\{P_i\}$ cannot be injured, so each P_i acts at most once. It can therefore injure other requirements at most once. The requirement N_e can only be injured by P_i if $i < e$. Therefore, $|I_e| \leq e$. In conclusion, $\{N_e\}$ are satisfied for all e and the injury is finite.

The requirements $\{P_e\}$ and their construction guarantee the simplicity of A for the same reasons as in Theorem 4.2.

Let us show that $\{P_e\}$ are satisfied. Suppose as an inductive hypothesis that $\{P_i\}_{i<e}$ and $\{N_i\}_{i\leq e}$ have all finished acting by the end of some stage s, and that P_e is still not satisfied ($W_{e,s} \cap A_s = \emptyset$). Thus, P_e becomes the highest priority requirement that still needs to act. Let $r_{max} = max\{2e, \{r(k,s)\}_{k\leq e}\}$. Since $2e$ and $\{r(k,s)\}_{k\leq e}$ are all finite, r_{max} is defined. If W_e is infinite, then it must contain an $x > r_{max}$. After the first stage t where this x enters W_e, x is enumerated into A. We now have that $W_{e,t} \cap A_{t+1} \neq \emptyset$. Therefore, we say P_e receives attention at stage $t+1$ and remains satisfied.

□

For an alternative but similar proof which uses the canonical simple set construction, please see Section 7.2 of [2].

In our next solution to Post's Problem, we continue to use the property of simplicity to guarantee that the set A is c.e. and noncomputable. To guarantee incompleteness, instead of the lowness of A, we now require that not all c.e. sets are reducible to A. We do this by concurrently building up another c.e. set B.

Theorem 6.4. *There exist a simple set A and a c.e. set B such that $B \not\leq_T A$.*

Proof. We define two types of requirements, $\{P_e\}$ and $\{N_e\}$. For the same e, let N_e take precedent over P_e.

$$P_e : \text{If } W_e \text{ is infinite, then } W_e \cap A \neq \emptyset.$$

$$N_e : B \neq \Phi_e^A.$$

The positive requirements $\{P_e\}$ attempt to enumerate members into A, and are analogous to the $\{P_e\}$ in the proof for Theorem 6.2. The negative requirements $\{N_e\}$ ensure that B is not computable in A, i.e., $B \not\leq_T A$. Unlike Theorem 6.2, $\{N_e\}$ also attempt to enumerate numbers into a set–the set B. We now define an extra measure to control which elements a certain N_e is responsible for.

Definition 6.5. Let $\mathbb{N}^{[e]} = \{\langle x, e\rangle | x \in \mathbb{N}\}$.

This means that we fix a number "e" and create a set of countably infinite size using this "e." This set is thus disjoint from any other set that uses a different "e." In the following construction, we want each N_e to choose its witness from a different pool of candidates, so any number can only be possibly enumerated by one requirement. If a requirement does not act, its witness will not be enumerated.

Construction.

Stage 0: Let $A_0 = B_0 = \emptyset$.

Stage $s+1$: We define witnesses for $\{N_e\}$ which do not currently have witnesses. Define a witness n for N_e where $n \in \mathbb{N}^{[e]} - B_s$.

Attention. We say N_e requires attention if $\Phi_{e,s}^{A_s}(n) = 0$. As before, we say P_e requires attention if $W_{e,s} \cap A_s = \emptyset$, and there exists an $x \in W_{e,s}$ such that $x > 2e$, and $(\forall i \leq e)[x > r(i,s)]$.

Action. Find the least $e < s$ such that the requirement (either P_e or N_e) requires attention.

To satisfy N_e, enumerate n into B_{s+1}, and define $r(e, s+1) = \psi_{e,s}^{A_s}(n)$.

To satisfy P_e, enumerate the least applicable x into A_{s+1}. Cancel witnesses for N_i and undefine $r(i, s+1)$ for $i > e$. Maintain $r(j, s+1) = r(j, s)$ for $j \leq e$.

First, let us show how the requirements $\{N_e\}$ are satisfied.

As an inductive hypothesis, suppose that all requirements $\{N_i\}_{i<e}$ have been satisfied. Since requirements $\{P_j\}$ cannot be injured (by merit of their positiveness), each P_j where $j < e$ acts at most once. Thus, N_e can be injured at most e times. After a certain stage s, N_e will not be injured again and the value of its witness n is finalized. There are now two cases that will determine how to satisfy N_e.

Case 1: There will never be a stage $t > s$ where $\Phi_{e,t}^{A_t}(n) = 0$. This implies that N_e will not require attention nor act from now on, and that either $\Phi_e^A(n) \uparrow$ or $\Phi_e^A(n) \neq 0$. If $\Phi_e^A(n) \uparrow$, we know that Φ_e^A is not a characteristic function, so it is also not equal to B. Since N_e does not act, n is kept out of B. If $\Phi_e^A(n) \neq 0$, we have that $B(n) = 0 \neq \Phi_e^A(n)$. The requirement is therefore satisfied.

Case 2: There will eventually be a stage $t > s$ where $\Phi_{e,t}^{A_t}(n) = 0$. We then enumerate n into B_{t+1} so that $B(n) = 1$. Given that $\Phi_e^A(n)$ outputs 0 while using only the numbers less than the use by the end of stage t, we need to make sure that the beginning of A remains unchanged. We therefore define $r(e, t+1) = \psi_{e,t}^{A_t}(n)$ to restrain any elements $x \leq \psi_{e,t}^{A_t}(n)$ from entering A after $t+1$. Since all higher-priority requirements have acted, this restraint cannot be injured. We then have that $\Phi_e^A(n) = \Phi_{e,t}^{A_t}(n) = 0$. Therefore, $\Phi_e^A(n) = 0 \neq 1 = B(n)$, and the requirement is satisfied.

In conclusion, $\{N_e\}$ are satisfied for all e, and $|I_e| \leq e$.

Since each restraint is some finite number, we may define r_{max} as we did in the proof for Theorem 6.2. The positive requirements $\{P_e\}$ are thus satisfied for the same reasons as those described for Theorem 6.2. □

7. The Friedberg-Muchnik Theorem

Theorem 7.1 (Friedberg-Muchnik Theorem)**.** *There exist c.e. sets A and B of incomparable degree, i.e., $A \not\leq_T B$ and $B \not\leq_T A$.*

The Friedberg-Muchnik Theorem was the first solution to Post's Problem. If a set is computable, then it is computable in any other set. Since A cannot be computed in B and B cannot be computed in A, neither A nor B are computable. If a set is complete, then every c.e. set is Turing reducible to it. Since A is not reducible to B and B is not reducible to A, neither B nor A are complete. Therefore, A and B represent two in-between degrees for c.e. sets.

Proof. We begin by defining the requirements.

$$R_{2e} : A \neq \Phi_e^B.$$

$$R_{2e+1} : B \neq \Phi_e^A.$$

These requirements resemble the negative requirements used in the proof of Theorem 6.4. However, since there are no positive requirements, we do not necessarily need to refer to these as "negative requirements." We may think of them as just "requirements"–they may injure lower-priority requirements, and they may be injured by higher-priority requirements.

The requirements $\{R_{2e}\}$ ensure that A is not computable in B, i.e., $A \not\leq_T B$. The requirements $\{R_{2e+1}\}$ ensure that B is not computable in A, i.e., $B \not\leq_T A$. The sets A and B are to be built concurrently, so we attempt to intersperse the requirements by stipulating that the index of the requirement be either even or odd.

Construction.

Stage 0: Let $A_0 = B_0 = \emptyset$.

Stage $s+1$: We define witnesses for requirements which do not currently have witnesses.

Define a witness x for R_{2e} where $x \in \mathbb{N}^{[e]} - A_s$ and $(\forall i < e)(x > r(i, s))$.

Define a witness n for R_{2e+1} where $n \in \mathbb{N}^{[e]} - B_s$ and $(\forall i < e)(n > r(i, s))$.

Attention. Let x and n be the witnesses of R_{2e} and R_{2e+1} respectively. We say R_{2e} requires attention if $\Phi_{e,s}^{B_s}(x) = 0$. We say R_{2e+1} requires attention if $\Phi_{e,s}^{A_s}(n) = 0$.

Action. Find the least $j < s$ such that R_j requires attention.

If $j = 2e$, enumerate x into A_{s+1}. If $j = 2e + 1$, enumerate n into B_{s+1}.

For both types of requirements, define restraint $r(e, s + 1) = s$. (We may also use $r(e, s+1) = \psi_{e,s}^{B_s}(x)$ for R_{2e} and $r(e, s+1) = \psi_{e,s}^{A_s}(n)$ for R_{2e+1}, similar to what we have done before. What we have now is fine since s is greater than or equal to the use at stage s.) Cancel witnesses for R_k and undefine restraints $r(k, s + 1)$ where $k > j$. Maintain restraints $r(i, s + 1) = r(i, s)$, where $i < j$.

Let us now examine how the requirements are satisfied.

Suppose that R_{2e+1} has not been satisfied. Further, suppose as an inductive hypothesis that all requirements $\{R_i\}_{i<2e+1}$ have finished acting by the end of stage s. This means that R_{2e+1} will not be injured again and the value of its witness n is finalized. We consider two cases for the satisfaction of R_{2e+1}, which are very similar to the arguments given for the satisfaction of N_e in the proof of Theorem 6.4. (The argument for R_{2e} is entirely analogous, where A and B are switched and n and x are switched whenever applicable.)

Case 1: There will never be a stage $t > s$ where $\Phi_{e,t}^{A_t}(n) = 0$. This implies that R_{2e+1} will not require attention nor act, and that either $\Phi_e^A(n) \uparrow$ or $\Phi_e^A(n) \neq 0$. The first possibility means that Φ_e^A is not any characteristic function, so it is also not equal to B. The second possibility means that we want $B(n) = 0 \neq \Phi_e^A(n)$. Since R_{2e+1} does not act, n is kept out of B. The requirement is therefore satisfied. In this case, no lower-priority requirements are injured as a result.

Case 2: There will eventually exist a stage $t > s$ where $\Phi_{e,t}^{A_t}(n) = 0$. Then R_{2e+1} requires attention, and we enumerate n into B_{t+1} so that $B(n) = 1$. We then define $r(e, t+1) = t$ in order to prevent $x \leq t$ from entering A after stage $t+1$. It ensures that $A_t \upharpoonright t = A \upharpoonright t$, so that we have $\Phi_e^A(n) = \Phi_{e,t}^{A_t}(n) = 0$. We cancel the restraints and witnesses of lower-priority requirements. This restraint cannot be injured by any requirements that have yet to act. Therefore, $\Phi_e^A(n) = 0 \neq 1 = B(n)$.

We may now show that the injury set is of finite size. This injury set is different from those of the previous two sections where $|I_e| \leq e$. Here, since we do not have strictly "positive" requirements that are never injured, we cannot say that every higher-priority requirement capable of injuring a certain requirement acts at most once. Nevertheless, we shall see that each requirement still acts only finitely often.

To illustrate as an example, only consider the requirements up to R_2. In the table below, we list out a possible scenario for these requirements at different stages s. The "$-$" sign indicates that the requirement is not currently satisfied, and "$+$" indicates that the requirement is currently satisfied. This scenario is the "worst case" scenario, where we try to maximize the amount of injury by always choosing the lowest priority requirement to act first. Also, we assume that each time a requirement is satisfied, it is due to action and thus causes injury to any lower-priority requirements (i.e. "Case 2," as described above).

	$s=0$	$s=1$	$s=2$	$s=3$	$s=4$	$s=5$	$s=6$	$s=7$
R_0	$-$	$-$	$-$	$-$	$+$	$+$	$+$	$+$
R_1	$-$	$-$	$+$	$+$	$-$	$-$	$+$	$+$
R_2	$-$	$+$	$-$	$+$	$-$	$+$	$-$	$+$

The highest priority requirement R_0 cannot be injured. It is unsatisfied until stage $s = 4$, and remains satisfied for the remainder of stages $s > 4$. The requirement R_1 can be injured at most once–when R_0 acts at $s = 4$. Eventually, R_1 acts again at $s = 6$, and it cannot be injured again thereafter. The requirement R_2 can be injured at most three times–once by R_0 ($s = 4$) and twice by R_1 ($s = 2; s = 6$). If we are to continue this construction, we discover that there is a pattern to the amount of times a given requirement can be injured. The requirement R_j can be injured at most $2^j - 1$ times. Therefore, we find that $|I_j| \leq 2^j - 1$.

In conclusion, $\{R_j\}$ are satisfied for all j, and injury is finite. □

Finally, notice that in all the finite injury methods presented herein, we are able to *a priori* quantify the size of the injury sets using some computable function. That is, we can find a computable upper bound on the number of times a given requirement can be injured before having to carry out the specifics of the construction. We thus call these *bounded injury* constructions. There exist also *unbounded injury* methods where $|I_e| < \infty$, but we cannot determine a computable upper bound. These priority methods produce very useful and important results, and the reader may consult [3] and [2] for examples.

Acknowledgements

It is a pleasure to thank my mentor, Gabriela Pinto, for her tireless support and her perceptive observations, from introducing the topic to suggesting corrections throughout each draft. This paper would have been impossible without our emails and Zoom calls. I would also like to thank Professor Denis Hirschfeldt for offering invaluable corrections and explanations both intuitive and technical, for answering any questions at all related to my topic, and for making me excited about computability. Finally, I would like to thank Professor Peter May for providing feedback regarding this paper, and for organizing this REU and allowing me to be part of it.

References

[1] Emil Post. "Recursively enumerable sets of positive integers and their decision problems." Bull. Amer. Math. Soc. 50 (1944), no. 5, 284–316. https://projecteuclid.org/euclid.bams/1183505800

[2] Robert Soare. Turing Computability: Theory and Applications. Springer. 2016.

[3] Rodney Downey and Denis Hirschfeldt. Algorithmic Randomness and Complexity. Springer. 2010.

[4] Hartley Rogers. Theory of Recursive Functions and Effective Computability. McGraw-Hill. 1967.

[5] Rodney Downey and Michael Stob. "Splitting theorems in recursion theory." Annals of Pure and Applied Logic 65 (1993), no. 1, 1–106. https://doi.org/10.1016/0168-0072(93)90234-5

[6] Peter Cholak. "On splits of computably enumerable sets." arXiv:1605.03034

[7] Joseph Shoenfield. Degrees of Unsolvability. North-Holland Publishing Company. 1971.