The Friedberg-Muchnik Theorem

Mathematical Logic for Computer Science

Simone Bianco, 1986936

Sapienza Università di Roma, Italy

June 22, 2025

Contents

Notation **2**

1 Introduction **2**
1.1 Decidability and semi-decidability 2
1.2 Degrees of unsolvability 4

2 Post's problem **6**
2.1 The finite extension method 6
2.2 The finite injury priority method 7

References **11**

Notation

In this essay, we'll use the notation $\Phi_{i,s}(x)$ to denote the computation for $s \in \mathbb{N}$ steps on input $x \in \Sigma^*$ of the i-th Turing machine, where $\mathcal{M} = \{M_0, M_1, M_2, \ldots\}$ is the set of all Turing machines and Σ^* is the set of all strings on an alphabet Σ. When the computation $\Phi_{i,s}(x)$ halts, we write $\Phi_{i,s}(x)\downarrow$. We say that $\Phi_i(x)$ *terminates*, written as $\Phi_i(x)\downarrow$, when there is a value $s \in \mathbb{N}$ such that $\Phi_{i,s}(x)\downarrow$. Otherwise, we say that $\Phi_i(x)$ *diverges* (or *loops*), written as $\Phi_i(x)\uparrow$. If $\Phi_i(x)\downarrow$ holds, we denote the output of M_i on input x with $\phi_i(x)$, where ϕ is a (partial) function from Σ^* to $\{0,1\}$. Each notation naturally extends when using oracles machines. For instance, $\Phi_i^A(x)$ denotes the computation of the i-th Turing machine on input x while having access to an oracle of the set A.

1 Introduction

1.1 Decidability and semi-decidability

In 1936, Alan Turing's groundbreaking work introduced the concept of a function being computable by a **Turing machine** (TM), an abstract model of a computer that is capable of capturing the modern concept of computation with high precision while maintaining a simple interpretation. During his work, together with his mentor Alonzo Church who had already explored these concepts, Turing was able to use his notion of **computability** to prove that some problems are not algorithmically solvable, i.e. not computable by a Turing machine (and thus by a modern computer) [Tur37].

The most famous example of uncomputable problem is the Halting Problem, which asks to determine if a given program will halt or not for a given input. Formally, this problem can be described as the set $H = \{(i,x) \mid \Phi_i(x)\downarrow\}$. After Turing's work gave birth to the field of computability theory, researchers began to explore what is computable and what isn't. In particular, they were able to distinguish two types of computability, known as **decidability** and **semi-decidability**.

Definition 1 (Semi-decidability)**.** Given a subset $S \subseteq \Sigma^*$, we say that S is *semi-decidable* if $\exists i \in \mathbb{N}$ such that $\forall x \in S$ it holds that $\Phi_i(x)\downarrow$ and $\phi_i(x) = 1$.

We observe that this definition implies that for each string $x' \notin S$ it can either hold that $\Phi_i(x')\uparrow$ or that $\Phi_i(x')\downarrow$ but $\phi_i(x') = 0$. When only the

second option holds for every string outside of S, i.e. the computation never diverges, we say that the set is *decidable*.

Definition 2 (Decidability)**.** Given a subset $S \subseteq \Sigma^*$, we say that S is *decidable* if $\exists i \in \mathbb{N}$ such that $\forall x \in \Sigma^*$ it holds that $\Phi_i(x) \downarrow$ and $\phi_i(x) = 1$ if $x \in S$, otherwise $\phi_i(x) = 0$ if $x \notin S$.

In other words, semi-decidability implies that an algorithm is capable of recognizing a positive instance for a problem but cannot always recognize a negative instance, while decidability implies that an algorithm can distinguish between positive and negative instances. In fact, it's easy to see that a set $S \subseteq \Sigma^*$ is decidable if and only if both S and $\overline{S}$ are semi-decidable.

Turing showed that the set H describing the Halting problem is semi-decidable, but not decidable. In particular, both result were proven through the use of an Universal Turing Machine (UTM), i.e. a Turing machine that can take the description of a TM and simulate its execution for a given input. The semi-decidability of H can be simply proven through an UTM that takes the pair $\langle M, x \rangle$, simulates $M(x)$ and returns 1 if it halts, or going into an infinite loop otherwise. The undecidability of H, instead, can be proven through the use of the diagonalization technique: if we assume the existence of a TM M that decides H, then there must also be a machine $\overline{M}$ that returns the opposite result of M on any input, making the computation $M(\langle \overline{M} \rangle, \langle \overline{M} \rangle)$ halt if and only if it doesn't halt, raising a clear contradiction.

Semi-decidability can also be described with an equivalent concept, that being **recursive enumeration** (or r.e. for short). This latter concepts describes the natural property of a set of elements to be enumerated by an algorithm, meaning that there is a procedure that prints all the elements of the set. We observe that such procedure doesn't have to halt, as some sets are clearly infinite: the definition simply requires that an algorithm can achieve such task, even if it takes an infinite amount of time.

Definition 3 (Recursive enumerability)**.** Given a subset $S \subseteq \Sigma^*$, we say that S is *recursively enumerable* if there is an algorithmic procedure $\mathcal{A} : \mathbb{N} \to \Sigma^*$ that produces a list of all and only the elements inside it, meaning that $S = \{\mathcal{A}(0), \mathcal{A}(1), \ldots\}$.

From this very definition, it should seem natural that a set is semi-decidable if and only if it is recursively enumerable: if a set has a semi-deciding procedure then the latter can be used to test all inputs in parallel and print only those with a positive answer, while if a set has an enumerating procedure then all positive inputs will be eventually printed by such procedure.

1.2 Degrees of unsolvability

From Turing's work and the underlying separation between computable and uncomputable problems, a natural question arises: is there a good measure of how much a problem is uncomputable? It turns out that a good enough measure can be obtained by other concepts that were developed by Turing himself, in particular **Turing reductions**, which describes the idea of a problem being solvable through the use of a machine that knows how to solve another problem. In particular, Turing formulated such notion through the use of *oracle machines*, that being Turing machines that have access to a black-box oracle (which can be seen as a magic tool or simply as another TM whose internal workings aren't known).

Definition 4 (Turing reduction)**.** Given two sets $A, B \subseteq \Sigma^*$, we say that A is *Turing reducible* to B, denoted with $A \leq_T B$, if there is an oracle TM M^B that has access to an oracle for B and that decides A.

Turing reductions are a generalized version of more commonly used *many-one reductions*, denoted with $A \leq_m B$, where the oracle TM is restricted to one single final query to the oracle. It's easy to see that Turing reductions define a partial order on the set 2^{Σ^*}. We also observe that there are clearly some problems that are equivalent to each other, either due to their definition (e.g. pairs of problems that ask the same question in two different ways) or to their intrinsic nature (e.g pairs of problems whose instances can be transformed back and forth). In particular, two problems $A, B \subseteq \Sigma^*$ are said to be **Turing equivalent**, written as $A \equiv_T B$, when $A \leq_T B$ and $B \leq_T A$. Intuitively, Turing equivalence defines an equivalence relation $\equiv_T$ that partitions the set 2^{Σ^*} into various classes of problems. Each class describes the complexity of the problems lying inside of it: for a problem $X \subseteq \Sigma^*$, each problem $Y \in [X]$ (where $[X]$ denotes the class of X over $\equiv_T$) is X*-complete*, meaning that it is perfectly interchangeable with the problem X [Soa87; Soa16].

When two problems lie in two different classes, at least one of them cannot be reduced to the other, implying that there is some intrinsic property that differentiates them. This allows us to view all of these classes as a *reduction hierarchy*. This hierarchy can be described in a natural way by a partial order over the classes and Turing reductions among them. In particular, given two classes $[X], [Y]$, we say that $[X] \leq [Y]$ if and only if $X \leq_T Y$. Intuitively, if $[X] \leq [Y]$ holds then each problem in the class $[Y]$ is "at least as hard" as each problem in the class $[X]$. This raises a natural concept of *hardness degree*, known as the **Turing degree**, first introduced by Post [Pos44].

Definition 5 (Turing degree)**.** Given a subset $X \subseteq \Sigma^*$, the *Turing degree* of

X is the equivalence class $[X]$ over the relation $\equiv_T$. We denote with $\mathcal{D}$ the set of all Turing degrees, i.e. the quotient set $\mathcal{D} = 2^{\Sigma^*}/_{\equiv_T}$.

The Turing degree acts as a measure that perfectly encapsulates the concept of algorithmic unsolvability. In fact, it's easy to see that any decidable problem has the same degree since for any pair $A, B \subseteq \Sigma^*$ of decidable sets we can decide A through a TM provided with an oracle for B simply by ignoring the oracle. We refer to the unique class of decidable problems as the degree 0 (or the degree $[\varnothing]$). Clearly, any problem that can be decided through an oracle for any set X of degree 0 is also also of degree 0. This implies that any problem outside of 0 must be undecidable under every single decidable oracle. The largest class of problems that lie outside of 0 is known as the class $0'$, where $0'$ denotes the **Turing jump** of 0 [Soa87; Soa16].

Definition 6 (Turing jump)**.** Given a subset $X \subseteq \Sigma^*$, the *Turing jump* of X is a problem X' that cannot be Turing reduced to X, i.e. $X' \not\leq_T X$.

This definition of jump comes from the fact that the Turing degree strictly increases, meaning that $d < d'$ for any degree $d \in \mathcal{D}$. In particular, we observe that the famous Halting problem H is $0'$-complete, meaning that $0' = [H]$. To prove this, we give a stronger result.

Theorem 1. *A problem is recursively enumerable if and only if it can be Turing reduced to the Halting problem.*

Proof. Given $A \subseteq \Sigma^*$, suppose that $A \leq_T H$. Since H is recursively enumerable, its semi-decider can be used to semi-decide the instances of A also is. Vice versa, suppose that A is recursively enumerable. Then, there is a semi-decider M_i for A. We can build a new machine M_j that on input x simulates $M_j(x)$ and accepts if $M_i(x) = 1$, otherwise it loops. It's easy to see that $x \in A$ if and only if $\langle j, x\rangle \in H$, concluding that $A \leq_m H$ and thus that $A \leq_T H$. □

The above theorem concludes that $0'$ is exactly the class of semi-decidable and undecidable problems. In particular, we get that for every **r.e. degree** d, i.e. a degree that contains an r.e. set, it holds that $d \leq 0'$. However, the converse doesn't always hold: there are some degrees that are below $0'$ but do not contain r.e. sets.

The above result also expresses that the jump of a set X be described as the set $X' = \{i \mid \Phi_i^X(i) \downarrow\}$, i.e. the set of Turing machines that halt when having access to X and themselves as input. Hence, we get that $[X] \leq 0'$ if and only if $X' \equiv_T 0'$ since $0' \leq_T X'$ holds for every set X. When d is a degree such that $d' \leq 0'$, we say that d is **low**.

2 Post's problem

As we discussed in the previous section, the set of all Turing degrees has the structure of a poset, where the partial order is given by Turing reducibility. The first natural question one should ask is "is this order total?", i.e. is every pair of degrees comparable in some way? Kleene and Post [KP54] proved, among many other results, that the answer to this question is no: there are two incomparable Turing degrees. To prove this result, they used a very powerful method called the **finite extension method**. A more interesting question was asked by Post in his initial work on Turing degrees [Pos44]. He asked whether there is some r.e. degrees $[X]$ that is non-decidable and non-zero, i.e. such that $0 < [X] < 0'$. Post's problem was independently solved by Friedberg [Fri57] and Muchnik [Muc56]. Surprisingly, both authors solved the problem by extending Kleene and Post's method with the same idea (with some slight differences), which is now know as the **finite injury priority method**. In the following sections, we'll first discuss the finite extension method and then extend it to the finite injury priority method.

2.1 The finite extension method

The finite extension method is a general technique used to prove many results for uncomparability of two types of sets. In this section, we'll give an idea behind the method by proving Kleene and Post's result [KP54].

Given two sets A, B, we want to define a countable list of requirements on both sets that must be satisfied by a string. By their construction, these requirements will automatically guarantee that the two sets are uncomparable. In the finite extension method, we start with the empty string ε and in each step, we extend the string in a way that preserves the satisfaction of previous requirements and satisfies new ones, and previously. Formally, we define a list the requirements be $\{R_i\}_{i\in\mathbb{N}}$ and set the empty string $A_0 = \varepsilon$. On each step s, we construct a finite string A_{s+1} that extends A_s and satisfies $R_0, R_1, \ldots, R_{s+1}$. In the end, we set $A = \bigcup_i A_i$.

In this context, strings are to be considered as an infinite tape of cells. Each cell is marked by an index and either contains a 0 or a 1 or is undefined. In some sense, each string can be viewed as a partial function on three values: $0, 1$ and $*$. Hence, the string A_{s+1} extends the string A_s if for each position $x \in \mathbb{N}$ it holds that if $A_s(x) \neq *$ then $A_{s+1}(x) = A_s(x)$. We now give an example of application of the finite extension method by proving Karp and Post's result, which gives that $(\mathcal{D}, \leq)$ is not a total ordering as a corollary.

Theorem 2 (Karp-Post theorem)**.** *There are two sets A and B that are incomparable under $\leq_T$.*

Proof. For each $i \in \mathbb{N}$, we define the requirement R_{2i} as $\phi_i^A \neq B$ and R_{2i+1} as $\phi_i^B \neq A$ (when $\Phi_i^A(x) \uparrow$ or $\Phi_i^B(x) \uparrow$, the requirements are considered to be satisfied). Let $\{R_i\}_{i\in\mathbb{N}}$ be the list of requirements and let $A_0 = B_0 = \varepsilon$. Consider a generic step $s \in \mathbb{N}$.

We discuss only the case when $s = 2i$ for some $i \in \mathbb{N}$ since the case $s = 2i+1$ is symmetrical. Choose an index $x \in \mathbb{N}$ such that $B_s(x)$ is undefined (observe that such index always exists since the strings are an infinite tape). If there is a finite extension A' of A_s such that $\Phi_i^{A'}(x) \downarrow$, set $A_{s+1} = A'$ and set:

$$B_{s+1}(y) = \begin{cases} B_s(y) & \text{if } y \neq x \\ 1 - \phi_i^{A'}(y) & \text{otherwise} \end{cases}$$

If no such extension exists, i.e. $\Phi_i^{A'}(x) \uparrow$ for all A' finite extensions of A_s, set $A_{s+1} = A_s$ and set:

$$B_{s+1}(y) = \begin{cases} B_s(y) & \text{if } y \neq x \\ \text{rand}(\{0,1\}) & \text{otherwise} \end{cases}$$

By construction, each $R_0, R_1, \ldots, R_s$ are still satisfied since A_{s+1}, B_{s+1} extend A_s, B_s and $\phi_{s+1}^A(x) \neq \phi_{s+1}^B(x)$ holds in both cases. Inductively, all requirements are satisfied, thus $A = \bigcup_i A_i$ and $B = \bigcup_i B_i$ are such that $\Phi_s^A(x) \neq B$ and $\Phi_s^B(x) \neq A$ for all $s \in \mathbb{N}$. □

We observe that this general method can be easily modified and extended. For instance, the requirement R_{2i} for each $i \in \mathbb{N}$ can be answered by an oracle for the halting problem H, meaning that $[A], [B] \leq 0'$ holds. Similarly, the conditions imposed by the list of requirements used in the above proof can be made more strict to get more specific results. For instance, by modifying the requirements we can "force the jump" by making them about the jump sets A' and B', obtaining that A and B are low sets.

2.2 The finite injury priority method

We saw how the finite extension method can be modified to force additional conditions on the two underlying sets A and B. However, there is no way requirement modification that guarantee recursive enumerability of the two sets. This is due to the underlying nature of the method itself: the sets are

constructed in a non-recursively enumerable way. This is what stopped Post and Karp from using their own method to solve Post's problem.

The finite injury priority method independently developed by Friedberg and Muchnik fixes this problem by "breaking" a finite number of requirements in order to guarantee that the two sets are r.e. As always, we start with the empty string $A_0 = \varepsilon$. On each step s, we add again extend A_s to A_{s+1} to satisfy some new requirements, but we also allow some previously satisfied requirements to be unsatisfied by A_{s+1}. This unsatisfied requirements are called *injuries*. The key idea is to assign a priority level to each requirement and force two conditions during the construction of the set A:

1. Each requirement as a finitely many requirements with higher priority
2. Each requirement can be injured by requirements with higher priority

Together, these two conditions impose that each requirement can be injured a finite number of times by other requirements. To show how the method works, we define a **query function** that enables us to measure when an injury occurs.

Definition 7 (Query function)**.** Let A be a set and let $i, s, x \in \mathbb{N}$. We define the query function $\omega^A_{i,s}$ as the largest index that is queried on an oracle for A during the computation $\Phi^A_{i,s}$:

$$\omega^A_{i,s}(x) = \begin{cases} \max\{z \in \mathbb{N} \mid A(z) \text{ is queried in } \Phi^A_{i,s}\} & \text{if } \Phi^A_{i,s} \downarrow \\ -1 & \text{otherwise} \end{cases}$$

Theorem 3 (Friedberg-Muchnik theorem)**.** *There are two r.e. sets A and B that are incomparable under $\leq_T$.*

Proof. For each $i \in \mathbb{N}$, we define the requirement R_{2i} as $\phi^A_i \neq B$ and R_{2i+1} as $\phi^B_i \neq A$ (when $\Phi^A_i(x) \uparrow$ or $\Phi^B_i(x) \uparrow$, the requirements are considered to be satisfied). Let $\{R_i\}_{i\in\mathbb{N}}$ be the list of requirements. We say that an index x is a *witness* for the requirement R_{2i} at step s if $\Phi^{B_s}_{i,s}(x) \downarrow$ and $\phi^{B_s}_{i,s}(x) \neq A_s(x)$. Symmetrically, we say that an index x is a *witness* for the requirement R_{2i+1} at step s if $\Phi^{A_s}_{i,s}(x) \downarrow$ and $\phi^{A_s}_{i,s}(x) \neq B_s(x)$.

On each step s, we compute the index $x_{j,s}$ and the restriction index $r_{j,s}$ for each $j \in \mathbb{N}$, where $x_{j,s}$ is a witness for R_j at step s and $r_{j,s}$ is used to dictate the priority of the requirements. In general, we'll force that when $x_{j,s} = r_{j,s} = -1$ holds then no witness is known for R_j at step s. Clearly, this implies that at the start of step s for each $k \in \mathbb{N}$ such that $s \leq k$ it holds that $x_{k,s} = r_{k,s} = -1$.

Let $A_0 = B_0 = \varepsilon$ and let $x_{j,0} = r_{j,0} = -1$ for each $j \in \mathbb{N}$. Consider a generic step $s \in \mathbb{N}$ and consider the requirement R_{2i} for each $i \in \mathbb{N}$ (the requirements R_{2i+1} are symmetrical). We construct the values $x_{2i,s+1}$ and $r_{2i,s+1}$ in the following way. If $x_{2i,s} \neq -1$, we propagate the previous values, i.e. $A_{s+1} = A_s$, $B_{s+1} = B_s$, $x_{2i,s+1} = x_{2i,s}$ and $r_{2i,s+1} = r_{2i,s}$.

Assume now that $x_{2i,s} = -1$. Let x be the minimum index such that $A_s(x) = *$ and such that for all $k < 2i$ we have that $x > r_{k,s}$, that being the minimum undefined index of the string A_s that is larger than each restriction value of the previous requirements. If $\Phi_{i,s}^{B_s} \uparrow$, we preserve that $x_{2i,s+1} = r_{2i,s+1} = -1$ and propagate $A_{s+1} = A_s$, $B_{s+1} = B_s$. Otherwise, we set $x_{2i,s+1} = x$ and $r_{2i,s+1} = \max(x, \omega_{i,s}^{B_s}(x))$. Finally, we set $B_{s+1} = B_s$ and:

$$A_{s+1}(y) = \begin{cases} A_s(y) & \text{if } y \neq x \\ A_s(y) & \text{if } y = x \text{ and } \phi_{i,s}^{B_s} = 1 \\ x & \text{if } y = x \text{ and } \phi_{i,s}^{B_s} = 0 \end{cases}$$

We now prove the correctness of the construction through the following claims.

Claim 1: for each $j \in \mathbb{N}$ it holds that:

1. R_j is injured a finite number of times
2. There exists a step s_0 such that for all $s \geq s_0$ and for all $k < j$ no new index is added to A_s

Proof of Claim 1. We observe that the requirement R_j is injured at step s if some $x \leq r_{j,s}$ is added to A_s, which only happens when for some $k < j$ a new index is added to A_s, i.e. all of $x_{j,k} = -1, \Phi_{j,s}^{B_s} \downarrow$ and $\phi_{j,s}^{B_s} = 1$ hold.

Let $g_{j,s} = \sum_{k<j \text{ s.t.} x_{k,s} \neq -1} 2^{-(k+1)}$ the injure value of R_j at step s. When R_j is injured at step s, the smallest value $k < j$ such that $x_{k,s} \neq x_{k,s+1}$ must satisfy $x_{k,s} = -1$ by construction. Thus, we get that:

$$g_{j,s+1} - g_{j,s} \geq 2^{-(j+1)} - \sum_{k<h<j} 2^{-(h+1)} = 2^{-j}$$

Hence, for each step s we have that $0 \leq g_{j,s} \leq 1 - 2^{-j}$, concluding that R_j can be injured at most $2^j - 1$ times. □

Claim 2: for each $j \in \mathbb{N}$ it holds that $\lim_{s\to\infty} x_{j,s}$ and $\lim_{s\to+\infty} r_{j,s}$ exist and are finite.

Proof of Claim 2. Assume that $j = 2i$ for some $i \in \mathbb{N}$ (the case $j = 2i + 1$ is symmetrical). By point 1) of Claim 1, we know there is a step s_0 such that for all $s \geq s_0$ the requirement R_j is not injured at step s. Let s_0^* be the biggest step satisfying both the above condition and point 2) of Claim 1.

If for some step $s' \geq s_0^*$ it holds that $x_{j,s} \neq -1$, then all values are preserved from step s', which implies that for all $t \geq s'$ we have that:

- $x_{j,t} = x_{j,s'}$ and $r_{j,t} = r_{j,s'}$
- $A_t(x_{j,t}) = A_{s'}(x_{j,s'})$
- $\Phi_{i,s}^{B_s}(x_{j,t}) \downarrow$ and $\phi_{i,s}^{B_s}(x_{j,t}) \neq A_t(x_{j,t})$

Hence, the constraint R_j is satisfied. Suppose now that $s' \geq s_0^*$ it holds that $x_{j,s} \neq -1$. Then, for each step $s \geq s_0^*$ let y_s be the minimum value such that $A_s(y_s) = *$ and such that $\forall k < j$ it holds that $y_s > r_{j,s}$. Since $x_{j,s} \neq -1$ for all $s' \geq s_0^*$, by construction we have that $\Phi_{i,s}^{B_s}(y_s) \uparrow$ for all $s' \geq s_0^*$, thus R_j is automatically satisfied. □

By Claim 2, we get that eventually each requirement R_j will be satisfied by the construction. Moreover, the use of the restricting values $r_{j,s}$ allows us to recursively enumerate the sets A and B since each requirement will eventually be satisfied in a finite number of steps.

□

The finite injury priority method can also be modified or extended to prove various results. For instance, Sacks [Sac64] proved that the above construction can be iterated an infinite amount of times, which is known as the *infinite injury priority argument*. This means that for each pair of r.e. sets A, B there is another r.e. set such that $A <_T C <_T B$. In other words, the partial ordering $\leq$ on $\mathcal{D}$ is actually dense, giving a way stronger answer to Post's problem.

References

[Fri57] Richard M. Friedberg. "Two recursively enumerable sets of incomparable degrees of unsolvability (solution of Post's problem, 1944)". In: *Proceedings of the National Academy of Sciences* (1957).

[KP54] Stephen C. Kleene and Emil L. Post. "The upper semi-lattice of degrees of recursive unsolvability". In: *Annals of mathematics* (1954).

[Muc56] Albert A. Muchnik. "On the unsolvability of the problem of reducibility in the theory of algorithms". In: *Dokl. Akad. Nauk SSSR* (1956).

[Pos44] Emil L. Post. "Recursively enumerable sets of positive integers and their decision problems". In: *Bull. Amer. Math. Soc.* (1944).

[Sac64] Gerald E. Sacks. "The Recursively Enumerable Degrees are Dense". In: *Annals of Mathematics* (1964).

[Soa16] Robert I. Soare. *Turing Computability: Theory and Applications*. Springer, 2016.

[Soa87] Robert I. Soare. *Recursively Enumerable Sets and Degrees*. Springer, 1987.

[Tur37] Alan M. Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem". In: *Proceedings of the London Mathematical Society* (1937).