Equivalence of Countable and Computable

Hantao Zhang
Department of Computer Science
The University of Iowa
Iowa City, Iowa, USA
hantao-zhang@uiowa.edu

Abstract

The concept of "countable set" is attributed to Georg Cantor, who set the boundary between countable and uncountable sets in 1874. The concept of "computable set" arose in the study of computing models in the 1930s by the founders of computer science, including Gödel, Church, and Turing. However, the connection between countability and computability was not thoroughly studied in the past eight decades. A *counting bijection* of a set S is a bijection from the set of natural numbers to S. We say S is *enumerable* if either S is finite or S has a computable counting bijection. S is countable if S is enumerable. We prove that S is enumerable if and only if S is computable. This equivalence provides insights to the study of set theory and computability theory and reveals that countability is related to space and time complexity. We show that the set of total computable functions (or deciders, algorithms) is unenumerable. We also provide a sufficient and necessary condition for a set to be enumerable and use the concept of *counting order* to explain why some sets are unenumerable. We show that some popular statements about countable sets, such as "every subset of a countable set is countable" or "every formal language is countable," lack a valid proof. These unproven statements are widely taught in college courses on discrete mathematics, set theory, and computability theory.

1 Introduction

In 1874, Georg Cantor [2] published the first proof that there is no one-to-one correspondence between the set of all real numbers, which is uncountable, and the set of all natural numbers, which is countable. In 1891, Cantor simplified his proof using the well-known diagonal method, which has important applications in set theory and computability theory [7, 14].

In the 1930s, several independent attempts were made to formalize the notion of computability [4]:

- In 1934, Kurt Gödel [8] defined formally a class of functions called *general recursive functions* (also called *partial recursive functions*).
- In 1932, Alonzo Church [3] created a method for defining functions called *lambda calculus*, often written as λ-calculus. Later, Church defined an encoding of the natural

numbers called *Church numerals*. A function on the natural numbers is called λ-computable if the corresponding function on the Church numerals can be represented by a term of the λ-calculus.

- In 1936, Alan Turing [17] created a theoretical model for computing, now called *Turing machines*, which could carry out calculations from inputs by manipulating symbols on a tape. Given a suitable encoding of the natural numbers as strings of symbols, a function on the natural numbers is called *Turing computable* if some Turing machine starts with the encoded natural numbers as input and stops with the result of the function on the tape as output.

Church and Turing proved that these three formally defined classes of computable functions coincide and proposed the well-known conjecture called Church–Turing thesis: A function is computable if it is Turing computable. Other formal attempts to characterize computability, including Kleene's recursion theory and von Neumann's random-access stored-programs (RASP) models, have subsequently strengthened this belief. Since we cannot exhaust all computing models, Church–Turing thesis, although it has near-universal acceptance, cannot be formally proved.

Nearly nine decades have passed, it looks puzzling that the connection between countability and computability has not been investigated until this day. We say a set S has a *counting bijection* if there exists a bijection $f : \mathcal{N} \mapsto S$, where $\mathcal{N}$ denotes the set of natural numbers. We say S is *enumerable* if either S is finite or S has a *computable counting bijection*. We show that a formal language is enumerable if and only if it is recognizable. This result comes from the relation between enumerable sets and enumerators, a variant of Turing machines, which print strings on a printer as an output device. Our equivalence result has several interesting consequences:

1. The concept of "enumerable" is not only related to space complexity, but also to time complexity.

2. We have alternative tools to show either a formal language is (un)recognizable or a set is (un)enumerable. An enumerable set is always countable.

3. If every countable set has a computable counting bijection, the concept of "countable" is equivalent to "enumerable" as well as to "computable".

In addition to the above contribution, we show that both the set of deciders and the set of algorithms are unenumerable, thus unrecognizable or uncomputable. We also provide a sufficient and necessary condition for a set to be enumerable and use the concept of *counting order* to explain why some sets are unenumerable.

We extend the notion of "computable" by allowing Turing machines to accept input strings of infinite length. This extended notion allows us to talk about computability of all counting bijections. We believe that Cantor used computable bijections in his study of countable sets. We propose a conjecture that "every countable set has a computable counting bijection." Assuming this conjecture is true, "countable" is equivalent to "enumerable" as well as to "computable" or "recognizable", and any unrecognizable language serves as a counterexample to the claim that "every subset of a countable set is countable."

2 Equivalence of Recognizable and Enumerable

We will assume the basic concepts from theory of computation [14, 18]. Given a Turing machine M, $L(M)$, the language *recognized* by M, is said to be *recognizable*. If M halts on every input, $L(M)$ is *decidable* and M is a *decider*. A function $f : \mathcal{N} \mapsto \mathcal{N}$, where $\mathcal{N}$ denotes the set of natural numbers, is *computable* if f can be computed by a Turing machine M. This M is said to be an *algorithm* if M halts on every input, and in this case, f is *total computable* (total and computable). We also call $f : \mathcal{N} \mapsto \{0, 1\}$ (or $f : \Sigma^* \mapsto \{0, 1\}$) a *decision function*, where Σ^* is the set of finite-length strings built on alphabet Σ. When a set S is defined by a decision function $p : \mathcal{N} \mapsto \{0, 1\}$, i.e., $S = \{x \in \mathcal{N} \mid p(x) = 1\}$, the complexity of p governs that of S. That is, we say S is *computable* if p is computable. If S is a formal language, then S is *computable* iff (if and only if) S is recognizable; S is decidable iff p is total computable. There is a bijection $\gamma : \Sigma^* \mapsto \mathcal{N}$ and we use γ to define a *canonical order* (also called *standard string order*), denoted by $<$, over Σ^*: For any $x, y \in \Sigma^*$, $x < y$ iff $\gamma(x) < \gamma(y)$ (see Definition A.2 in the appendix). $<$ is a well order of Σ^*.

We will need a computing model called *enumerator*, which is a variant of Turing machine. An enumerator is a Turing machine M with an attached printer [14]. M can use that printer as an output device to print strings. The set $E(M)$ of printed strings is the language *enumerated* by M. The following result is known for enumerators [14, 18]:

Theorem 2.1 (*a*) *If $L = E(M)$, where M is an enumerator, then L is recognizable.*
(*b*) *If L is a recognizable formal language, there exists an enumerator M such that $L = E(M)$ and every printed string by M is unique.*

For decidable languages, the following result is known [14, 18].

Theorem 2.2 *A formal language L is decidable if and only if there exists an enumerator M such that $L = E(M)$ and the printed strings are increasing in the canonical order.*

In combinatorics, mathematicians use a technique for proving that two sets X and Y of combinatorial objects have equally many elements by finding a bijection $f : X \mapsto Y$. This technique, called *bijective proof*, is old and useful, and is mostly applied to finite sets as a way of finding a formula for $|X|$ if $|Y|$ is easier to count [12]. Cantor used this popular technique to define the countability of an infinite set.

Every bijection must be total, injective, and surjective. If it is not total, its inverse is not surjective. If $f : X \mapsto Y$ is a bijection, the following properties of f are desired in countability study:

- **Counting**: $X = \mathcal{N}$ and we say f is a *counting bijection* of Y.
- **Explicit**: The definition of f does not depend on $|X|$ or $|Y|$.
- **Computable**: f can be computed by an effective method to obtain $f(x)$ for any $x \in X$.

Counting bijections occur in countability proofs. If you want to show an infinite set is countable, you must find a counting bijection for this set.

People in combinatorics often call the bijection used in a bijective proof an *explicit bijection* [12]. Some people insist that all bijections in bijective proofs must be explicit. The notion of *explicit bijection* is well understood and used in practice, but it does not seem to have a formal definition. Here we define *explicit bijection* as a bijection that satisfies the explicit property.

Since the computable property of bijections is much more important than the explicit property, our discussion will focus on computable counting bijections.

Definition 2.3 *A set S is said to be* enumerable *if either S is finite or S has a computable counting bijection $f : \mathcal{N} \mapsto S$.*

By this definition, S is enumerable iff S is countable and either S is finite or S has a computable counting bijection.

Proposition 2.4 *If S is countable but unenumerable, then there exists an uncomputable bijection and every counting bijection of S is uncomputable.*

Proof. Since S is unenumerable, S must be infinite. Since S is countable, there exists a bijection $f : \mathcal{N} \mapsto S$ and f must be uncomputable. For any bijection $g : \mathcal{N} \mapsto S$, g must be uncomputable, otherwise, S becomes enumerable. □

The existence of an uncomputable counting bijection is an open problem. If every countable set has a computable counting bijection, then "enumerable" is equivalent to "countable".

The following two theorems show that a formal language is enumerable iff it is recognizable.

Theorem 2.5 *Every recognizable formal language is enumerable.*

Proof. Suppose L is recognizable. The case when L is finite is trivial. If L is infinite, by Theorem 2.1(b), we have an enumerator M such that $L = E(M)$ and every printed string by M is unique. $E(M)$ is countable because the order in which the strings are printed by M defines a computable bijection $f : \mathcal{N} \mapsto E(M)$. That is, for $n \in \mathcal{N}$, we compute $f(n)$ by algorithm $A(n)$ as follows:

> **Algorithm** $A(n)$: Let $c := 0$ and simulate M. When a string x is printed by M, check if $c = n$. If yes, return x; otherwise $c := c+1$ and continue the simulation.

Algorithm $A(n)$ will terminate because $E(M)$ is infinite and c, which records the number of printed strings by M, will reach n eventually. $A(n)$ can be modified as $B(w)$ to compute $f^{-1}(w)$ for $w \in E(M)$: Instead of checking $c = n$, check if $x = w$ and, if yes, return c. Since both f and f^{-1} are total functions, f is a bijection. Algorithm $A(n)$ is the evidence that f is computable. □

Theorem 2.6 *Every enumerable formal language is recognizable.*

Proof. The case when formal language L is finite is trivial. If L is enumerable, then there exists a computable bijection $f : \mathcal{N} \mapsto L$. We use f to design an enumerator M that prints strings $f(0)$, $f(1)$, $f(2)$, and so on. It is evident that $E(M) = L$. By Theorem 2.1(a), L is recognizable. □

The following theorem combines the above two theorems into one and extends the result from formal languages to general sets.

Theorem 2.7 *A set is enumerable iff it is computable.*

Proof. Let S be any set. The case when S is finite is trivial and we assume S is infinite in the proof.

If S is enumerable, then there exists a computable counting bijection $f : \mathcal{N} \mapsto S$. Since f is total computable, there exists an algorithm to compute f. Hence, S can be represented by the formal language $\{\langle f(0)\rangle, \langle f(1)\rangle, ...\}$, where $\langle f(i)\rangle$ is a finite-length string representing $f(i)$. By Theorem 2.6, S is recognizable, thus, computable.

When S is unenumerable, there are two cases to consider: S can be represented by a formal language or not. If yes, then S is unrecognizable by Theorem 2.5, thus uncomputable. If not, then S cannot be recognized by any Turing machine, thus uncomputable. □

The above theorem provides an alternative way to show if a set is enumerable or not. It is known that the collection of recognizable sets is closed under intersection (i.e., the intersection of any two recognizable sets is recognizable). It implies that the intersection of two enumerable sets is enumerable, thus countable. Consider the following encoding of some decision problems:

- $G_{TM} = \{\langle M\rangle \mid \text{TM } M \text{ is well-defined }\}$
- $E_{TM} = \{\langle M\rangle \mid L(M) = \emptyset \text{ for TM } M\}$
- $N_{TM} = \{\langle M\rangle \mid L(M) \neq \emptyset \text{ for TM } M\}$
- $All_{TM} = \{\langle M\rangle \mid L(M) = \Sigma^* \text{ for TM } M\}$

They are formal languages since $\langle M\rangle \in \Sigma^*$ for some alphabet Σ. G_{TM} is the encoding of all well-defined Turing machines. G_{TM} is decidable by the assumption that we have an efficient decoder to check if $\langle M\rangle$ comes from a well-defined M. By Theorem 2.5, Σ^* and G_{TM} are enumerable, hence countable.

E_{TM}, N_{TM}, and All_{TM} are the encoding of all Turing machines that accept, respectively, nothing, something, and everything. N_{TM} is the complement of E_{TM}. It is known that N_{TM} is recognizable and E_{TM} is unrecognizable [14]. By Theorem 2.5, N_{TM} is countable. By Theorem 2.6, E_{TM} is unenumerable. Both All_{TM} and its complement, $\overline{All_{TM}}$, are unrecognizable, hence, unenumerable.

We say a counting bijection $f : \mathcal{N} \mapsto X$ is *increasing* if $f(n) < f(n+1)$ for every $n \in \mathcal{N}$, where $<$ is a well order of X. The following proposition is important about countable sets.

Proposition 2.8 *If X is countably infinite, then*

1. *X is decidable iff there exists a computable and increasing bijection $f : \mathcal{N} \mapsto X$.*

2. *X is computable iff there exists a computable bijection $f : \mathcal{N} \mapsto X$.*

3. *X is uncomputable iff every bijection $f : \mathcal{N} \mapsto X$ is uncomputable.*

Proof. By Theorem 2.7, X is computable iff X is enumerable. (3) comes from Proposition 2.4, although the existence of an uncomputable counting bijection is unknown. (2) is logically equivalent to (3). For (1), by Theorem 2.2, X is decidable iff the elements of X can be enumerated in increasing order. As in the proof of Theorem 2.5, this enumeration defines an increasing and computable bijection. On the other hand, as in the proof of Theorem 2.6, an increasing and computable bijection will produce an enumerator that enumerates elements of X in increasing order. □

The above result shows that the properties of counting bijection $f : \mathcal{N} \mapsto X$ is crucial to dictate if X is decidable, computable, or uncomputable. In particular, if X is computable but undecidable, then f must be computable but not increasing.

From Theorem 2.7, we also have the following result.

Corollary 2.9 *Let $S = \{x \in \mathcal{N} \mid p(x) = 1\}$.*

1. *S is decidable iff $p(x)$ is total computable.*

2. *S is enumerable iff $p(x)$ is computable.*

3. *S is unenumerable iff $p(x)$ is uncomputable.*

The above result shows that, when $p(x)$ becomes hard to compute, S may become unenumerable. Some people take for granted that $p(x)$ is computable and arrive at the conclusion that S is countable. Many formal languages are unrecognizable. By Theorem 2.6, any unrecognizable language is unenumerable and defines an uncomputable decision function (i.e., the characteristic function of this unenumerable set).

3 Some Unenumerable Sets

The assumption that a function f is computable is the same as saying "f can be computed by a Turing machine" in the spirit of Church-Turing thesis. Turing *et al.*'s notion of "computable" limits the scope of computable functions to the equivalent of integer functions. When considering all bijections, we need to extend the notion of "computable" to functions $f : A \mapsto B$, where A or B cannot be represented by Σ^* for any alphabet Σ. We do so by allowing Turing machines to accept input strings of infinite length. To facilitate the discussion, we introduce the following definition.

Definition 3.1 *An object x is* finitely presentable *if x can be represented by $w \in \Sigma^*$ for a chosen alphabet Σ.*

For instance, every natural number $i \in \mathcal{N}$ is finitely presentable by a^i for $\Sigma = \{a\}$. A Turing machine M is finitely presentable by $\langle M \rangle$. A recognizable language L is finitely presentable also by $\langle M \rangle$ for some Turing machine M. Σ^* is finitely presentable because

Σ^* is recognizable. $\mathcal{N}$ is finitely presentable because $\mathcal{N}$ can be represented by Σ^*. On the other hand, neither an arbitrary binary string s of infinite length nor an arbitrary subset $X \subset \mathcal{N}$ is finitely presentable. If each member of a set is finitely presentable, the set itself can be represented by a string which is the concatenation of its members separated by a delimiter symbol. Thus, a formal language can be represented by a string whose length may be infinite. Any function $f : \mathcal{N} \mapsto \mathcal{N}$ can be represented by string $f(0)\#f(1)\#f(2)\#\cdots$, where $\#$ is a delimiter symbol.

Proposition 3.2 *If $f : X \mapsto Y$ is a computable bijection and each member of Y is finitely presentable, then each member of X is finitely presentable.*

Proof. Use $f(x)$ to represent $x \in X$. □

The new notion of "computable" will cover $f(x) = y$ when x is not finitely presentable but can be represented by a string of infinite length. The computing device is a variant of Turing machine. Let us call this variant of Turing machine a *3-tape Turing machine* that has three tapes: the input tape (read-only), the working tape (read and write), and the output tape (write-only). A 3-tape Turing machine is equivalent to a standard Turing machine because it is a multiple tape Turing machine [14].

Definition 3.3 *A function $f : X \mapsto Y$ is said to be* computable *with $f(x) = y$ for $x \in X$ and $y \in Y$ if there exists a 3-tape Turing machine M such that when the input tape of M contains x, M produces y on the output tape.*

In the above definition, if $X = Y = \Sigma^*$, then the new notion of "computable" coincides with the old notion, where x and y are finitely presentable for $f(x) = y$. If $x \in X$ is not finitely presentable, then x can be represented by a string of infinite length and computed by the same machine. In this case, the machine will not terminate. For instance, if s is a binary string of infinite length, then the machine for computing $f(s) = 0.s$ (a real number between 0 and 1) will copy s from the input tape to the output tape and never terminate. In general, a 3-tape machine will work like an online algorithm which can process its input piece-by-piece in a serial fashion to produce (partial) output, without having to read the entire input. To reason about inputs of infinite length, instead of an algorithmic approach, we use formal tools such as mathematical induction.

Example 3.4 We can show that two uncountable sets $\mathcal{P}(\mathcal{P}(\Sigma^*))$ and $\mathcal{P}(\mathcal{P}(\mathcal{N}))$ have the same size by using the bijection $\gamma : \Sigma^* \mapsto \mathcal{N}$ (Definition A.2). That is, we define $g : \mathcal{P}(\Sigma^*) \mapsto \mathcal{P}(\mathcal{N})$ by $g(X) = \{\gamma(x) \mid x \in X\}$ for any $X \in \mathcal{P}(\Sigma^*)$, and $h : \mathcal{P}(\mathcal{P}(\Sigma^*)) \mapsto \mathcal{P}(\mathcal{P}(\mathcal{N}))$ by $h(S) = \{g(s) \mid s \in S\}$ for any $S \in \mathcal{P}(\mathcal{P}(\Sigma^*))$. Since γ is a bijection, g and h are bijections by an induction on the structure of sets. While X and S are not finitely presentable in general, g and h are all computable, because X and S can be represented by strings of infinite length and we can implement g and h by Turing machines. □

From now on, when we say a bijection is computable, we use the extended notion of "computable". Once a set S is known to be enumerable, each member of S is finitely presentable (Proposition 3.2), and we can compute S by a standard Turing machine. That is, the 3-tape Turing machine is purely a theoretic analysis tool for enumerable sets.

In the following we will consider a set of integer functions and a set of formal languages. The remaining of this section is devoted to the following result.

Theorem 3.5 *The following sets are unenumerable, hence uncomputable.*

- $F_T = \{f \mid f : \mathcal{N} \mapsto \mathcal{N} \text{ is total }\}$
- $D_T = \{f \mid f : \mathcal{N} \mapsto \{0,1\} \text{ is total }\}$
- $F_{TC} = \{f \mid f : \mathcal{N} \mapsto \mathcal{N} \text{ is total computable }\}$
- $D_{TC} = \{f \mid f : \mathcal{N} \mapsto \{0,1\} \text{ is total computable }\}$
- $D_L = \{L \mid L \subseteq \Sigma^* \text{ is decidable }\}$
- $D_{TM} = \{\langle M \rangle \mid \text{Turing machine } M \text{ is a decider }\}$
- $Al_{TM} = \{\langle M \rangle \mid \text{Turing machine } M \text{ is an algorithm }\}$

By definition, $D_{TC} \subset D_T \subset F_T$ and $D_{TC} \subset F_{TC} \subset F_T$. D_L is the collection of decidable languages. D_{TM} and Al_{TM} are, respectively, the encoding of deciders and algorithms. There is a surjective function $f : D_{TM} \mapsto D_L$ by $f(\langle M \rangle) = L(M)$ and f is not injective. The proof of the above theorem is divided into several propositions and corollaries, and the major proof tool is Cantor's diagonal method.

Proposition 3.6 *The set of all total computable decision functions, D_{TC}, is unenumerable.*

Proof. If D_{TC} is enumerable, then there exists a computable bijection $h : \mathcal{N} \mapsto D_{TC}$, and we can compute $f_n = h(n)$. The function $g(n) = 1 - f_n(n)$ is total computable because h and f_n are total computable. Define $g(n) = 1 - f_n(n)$ for every $n \in \mathcal{N}$, then $g \in D_{TC}$ because g is a total decision function. If $g = f_k$ for some $f_k \in D_{TC}$, then $g(k) \neq f_k(k)$, a contradiction coming from the assumption that D_{TC} is enumerable. Hence, D_{TC} must be unenumerable. □

The method used in the above proof is called *the diagonal method*, coming originally from Cantor [2]. If we list the values of $f_i(j)$ in a table for $i, j \in \mathcal{N}$, then g is defined using the values on the main diagonal of the table, i.e., $f_n(n)$.

Corollary 3.7 *The set of all decidable languages, D_L, is unenumerable.*

Proof. Because each $L \in D_L$ has its characteristic function in D_{TC} and vice versa, D_{TC} and D_L have the same size. That is, for every $L \in D_L$, there exists a bijection $f : D_L \mapsto D_{TC}$ with $f(L) = L$'s characteristic function. D_L cannot be enumerable if D_{TC} is unenumerable. □

Corollary 3.8 *The set of all total decision functions, D_T, is unenumerable.*

Proof. The proof is analogous to that of Proposition 3.6. □

In literature, D_T is proved to be uncountable by the diagonal method. The proof goes as follows: If D_T is countable, then there exists a bijection $f : \mathcal{N} \mapsto D_T$ and $D_T = \{f_0, f_1, f_2, ...\}$, where $f_i = f(i)$ for $i \in \mathcal{N}$. The rest of the proof is the same as that of

Proposition 3.6 by defining $g(n) = 1 - f_n(n)$ for every $n \in \mathcal{N}$. There is a problem in this proof: To get $f_i = f(i)$, we need a computable $f : \mathcal{N} \mapsto D_T$; if f is uncomputable, we cannot get f_i. If f is computable, then the conclusion should be "D_T is unenumerable," not "D_T is uncountable."

An alternative proof of Corollary 3.8 is to show that D_T and $\mathcal{P}(\mathcal{N})$ (the power set of $\mathcal{N}$) have the same size. That is, let $g : \mathcal{P}(\mathcal{N}) \mapsto D_T$ be $g(S) =$ "the characteristic function of $S \in \mathcal{P}(\mathcal{N})$", then g is a bijection. Since $\mathcal{P}(\mathcal{N})$ is uncountable, so is D_T. However, to prove that $\mathcal{P}(\mathcal{N})$ is uncountable by the diagonal method, we still need a computable bijection and $\mathcal{P}(\mathcal{N})$ should be unenumerable. Since g is computable, we still arrive at the conclusion that D_T is unenumerable. This problem exists in every proof using the diagonal method. In other words, Cantor's diagonal method can show only a set is unenumerable, not uncountable. It enforces our belief that Cantor would have used computable bijections in his diagonal method.

The above result on decision functions can be generalized to all total functions.

Corollary 3.9 *The set of total integer functions, F_T, and the set of total computable integer functions, F_{TC}, are unenumerable.*

Proof. The proof is analogous to that of Proposition 3.6, using $g(n) = f_n(n) + 1$ instead of $g(n) = 1 - f_n(n)$. □

Let $f : D_{TM} \mapsto D_L$ be $f(\langle M \rangle) = L(M)$, then f is surjective, but not injective. From the unenumerability of D_L, we cannot conclude that D_{TM} is unenumerable. Using a proof like that of Proposition 3.6, we have the following result.

Proposition 3.10 *The set of all deciders is unenumerable, hence, D_{TM} is unrecognizable.*

Proof. Assume D_{TM} is enumerable with a computable bijection $h : \mathcal{N} \mapsto D_{TM}$ such that $h(i) = \langle M_i \rangle$ for each $i \in \mathcal{N}$. Using h and γ (Definition A.2), we construct the following Turing machine X:

$X =$ "On input $w \in \Sigma^*$,
1. Compute $i = \gamma(w)$ and $h(i) = \langle M_i \rangle \in D_{TM}$.
2. Simulate M_i on w until M_i halts. // M_i is a decider
3. **return** $1 - M_i(w)$." // i.e., $X(w) = \neg M_i(w)$

Since every step of X stops, $\langle X \rangle \in D_{TM}$. Let $X = M_k$ for some $k \in \mathcal{N}$. However, $X(w_k) \neq M_k(w_k)$, a contradiction to $X = M_k$. So, the assumption is wrong and D_{TM} is unenumerable. By Theorem 2.5, D_{TM} is unrecognizable. □

The concept of countability has been studied from the viewpoint of space complexity (from $\mathcal{N}$ to $\mathcal{P}(\mathcal{N})$). It is also actually related to the time complexity of functions. From the unenumerability of D_{TM}, either any counting bijection of D_{TM} is uncomputable (a time complexity issue) or D_{TM} is uncountable.

Assuming every countable set has a computable counting bijection, from Corollary 2.9, D_{TM} becomes uncountable because of the difficulty of deciding whether a Turing machine is a decider or not. Indeed, D_{TM} is the encoding of this very decision problem and is unrecognizable. This view of countability will help us to understand why a subset of a countable set can be uncountable. The uncountability of D_{TM} is counter-intuitive because people believe that decision functions outnumber Turing machines. A conventional counting argument goes as follows: D_T is uncountable but G_{TM} (the set of Turing recognizers) is countable. Hence, there are many decision functions of D_T that cannot be computed by Turing machines. Now, assuming every countable set has a computable counting bijection, we have an uncountable set D_{TM} and $D_{TM} \subset G_{TM}$. We may wonder if there exists a bijection from D_T to D_{TM} and there is a decider for every decision function — we know this is false because there are no Turing machines for uncomputable decision functions. Nonetheless, the logic behind the conventional counting argument is weakened, and how to compare $|D_{TM}|$ and $|D_T|$ is an open problem.

Corollary 3.11 *The set of all algorithms is unenumerable, hence, Al_{TM} is unrecognizable.*

Proof. The proof is analogous to that of Proposition 3.10, using $g(n) = f_n(n) + 1$ instead of $g(n) = 1 - f_n(n)$ for the function computed by an algorithm. □

This result is also counter-intuitive because we know that the set of procedures is countable (each procedure can be stored in a text file, a binary string) and Al_{TM} is a subset of procedures. However, we do not have a Turing machine to tell if an arbitrary procedure is an algorithm or not – this is the so-called *uniform halting problem* that asks if a Turing machine halts on every input and is known uncomputable.

In summary, since every set in Theorem 3.5 is unenumerable, by Theorem 2.7, they are uncomputable.

4 A Criterion for Being Enumerable

In mathematics, a *poset* (partial order set) is a pair $(S, \succeq)$, where S is a set and $\succeq$ is a partial order of S, i.e., $\succeq$ is an antisymmetric, transitive, and reflexive relation on S. $\succeq$ is *total* if $a \succeq b$ or $b \succeq a$ for any $a, b \in S$. $\succeq$ is *well-founded* if every subset of S has a minimal element by $\succeq$. $\succeq$ is a *well order* if $\succeq$ is total and well-founded. $\succeq$ is *dense* if for any $a, b \in S$, if $a \succeq b$, then there exists $c \in S$ such that $a \succeq c \succeq b$.

Definition 4.1 (*a*) *Given a poset $(S, \succeq)$, $\succeq$ is said* gap-finite *if for any two elements $a, b \in S$, any sequence of distinct elements ordered by $\succeq$ from a to b, i.e., $a \succeq c_1 \succeq c_2 \cdots \succeq b$, is finite.*

(*b*) *$\succeq$ on S is a* counting order *of S if $\succeq$ is a gap-finite well order.*

For example, $\geq$ is a counting order for $\mathcal{N}$; $\geq$ is not a counting order for $\mathcal{Z}$, the set of integers, because $\geq$ is not well-founded on $\mathcal{Z}$.

If we list $\mathcal{N}$ by listing all even numbers before any odd number, then the order of this listing is not gap-finite, as there are infinite many even numbers between 2 and 1.

Let $\geq$ be the order on $\mathcal{N} \times \mathcal{N}$ such that $(x_1, y_1) \geq (x_2, y_2)$ if $x_1 \geq x_2$ and $x_1 \neq x_2$ or $x_1 = x_2$ and $y_1 \geq y_2$ (that is, $\geq$ is a lexigographic order of $\mathcal{N} \times \mathcal{N}$). $\geq$ is a well order on $\mathcal{N} \times \mathcal{N}$. However, $\geq$ is not a counting order because it is not gap-finite. There exists an infinite sequence between $(0, 0)$ and $(1, 0)$: $(1, 0) \geq \cdots \geq (0, 2) \geq (0, 1) \geq (0, 0)$. Note that a dense order is never gap-finite, and $\geq$ on $\mathcal{N} \times \mathcal{N}$ shows that the concepts of "gap-finite" and "non-dense" are different because $\geq$ on $\mathcal{N} \times \mathcal{N}$ is neither dense nor gap-finite.

Let $\succeq$ denote the subset relation on $\mathcal{P}(\mathcal{N})$: $X \succeq Y$ if and only if $Y \subseteq X$. It is known that $\succeq$ is a well-founded partial order on $\mathcal{P}(\mathcal{N})$. However, $\succeq$ is neither total nor gap-finite.

Proposition 4.2 *If S has a counting order $\succeq$, then every element of S can be listed one by one in the increasing order of $\succeq$.*

Proof. If $b \in S$ is never listed by $\succeq$ from $a = min(S)$, the minimal element of S, then there are an infinite number of elements between b and a ordered by $\succeq$, a contradiction to $\succeq$ being gap-finite. □

Cantor's diagonal method is to show that the assumed counting order is not gap-finite because some elements are left out of the order.

Proposition 4.3 *If S is enumerable, then S has a computable counting order.*

Proof. We consider the case when S is infinite as the finite case is trivial. If S is enumerable, then there exists a computable bijection $f : S \mapsto \mathcal{N}$ and $S = \{a_0, a_1, a_2, ...\}$ with $f(a_i) = i$. We use f to define $\succeq$: For any $a, b \in S$, $a \succeq b$ if $f(a) \geq f(b)$. Since $\geq$ is a gap-finite well order on $\mathcal{N}$, $\succeq$ is a gap-finite well order on S. Since f is computable, $\succeq$ is computable. □

Given a poset $(S, \succeq)$, where $\succeq$ is a counting order, let us define operation *deleteMin* as follows:

- *deleteMin*$(S, \succeq)$: delete the minimal element of S from S and return this element.

We assume that there are no other operations that can change S.

Theorem 4.4 *A set S is enumerable if and only if* deleteMin$(S, \succeq)$ *is an algorithm.*

Proof. If S is enumerable, let $S = \{a_0, a_1, a_2, ...\}$, where $a_i = f(i)$ and $f : \mathcal{N} \mapsto S$ is a computable bijection of S. By Proposition 4.3, S has a computable counting order, say $\succeq$. Since no operations can change S other than *deleteMin*, the first call of *deleteMin* will return a_0 and S becomes $\{a_1, a_2, ...\}$; the second call of *deleteMin* will return a_1 and S becomes $\{a_2, ...\}$, and so on. Thus, operation *deleteMin*$(S, \succeq)$ can be implemented easily as follows: Use an integer variable c to remember how many times *deleteMin* has been called. Initially $c := 0$. *deleteMin* returns $a_c = f(c)$ and then increases c by one. We have just described how *deleteMin* was implemented by a simple algorithm.

On the other hand, if *deleteMin*$(S, \succeq)$ is an algorithm, then S can be enumerated by repeatedly calling *deleteMin*. Since $\succeq$ is a counting order, every element of S will be returned by *deleteMin*. Define $g : \mathcal{N} \mapsto S$ by $g(i) =$ "the element returned by the $(i+1)^{th}$ call of *deleteMin*", then g is a computable bijection. So, S is enumerable. □

The above theorem gives us a necessary and sufficient condition for a set S to be enumerable: Having an algorithm *deleteMin* with a counting order $\succeq$. Every element of S can be enumerated one by one from the smallest by the counting order $\succeq$. Note that S can be any enumerable set, not just a set of natural numbers. The enumerated elements are not necessarily in increasing order. For instance, the set N_{TM} (the complement of E_{TM}) is recognizable but undecidable, then N_{TM} cannot be enumerated in increasing order (see Theorem 2.2). The above theorem can also explain why uncomputable sets like E_{TM} (the emptiness problem of Turing machines) are unenumerable, because these sets lack algorithm *deleteMin* or a counting order. In $S = \{x \in \mathcal{N} \mid p(x) = 1\}$, the high complexity of $p(x)$ may destroy the counting order of $\mathcal{N}$ and make no counting orders available for S, so S may lose any counting order and become unenumerable.

Since an enumerable set is also countable, Theorem 4.4 provides a sufficient condition for countable sets. Can we have a necessary and sufficient criterion for countable sets? It looks unlikely, because an uncomputable bijection prohibits the existence of any procedure like algorithm *deleteMin*.

5 Unproven Claims about Countable Sets

The claim that "every subset of $\mathcal{N}$ is countable" appears in many textbooks on set theory or theory of computation [9, 10]. In a textbook on set theory [5] published in 1977, the author simply wrote without proof that "Obviously every subset of a countable set is countable." In a textbook on theory of computation [13], the claim appears as Theorem 8.25. It also appears in an influential textbook by Terence Tao [16] as shown below.

Proposition 8.1.5 of [16] *Let X be an infinite subset of the natural numbers $\mathcal{N}$. Then there exists a unique bijection $f : \mathcal{N} \mapsto X$ which is increasing, in the sense that $f(n+1) > f(n)$ for all $n \in \mathcal{N}$. In particular, X has equal cardinality with $\mathcal{N}$ and is hence countable.*

A proof sketch of the above proposition goes as follows [16]: Since $<$ is a well order of X, X has a minimal element under $<$, say $a_0 = min(X)$. Let $X_0 = X$ and, for $i > 0$, $X_i = X_{i-1} - \{a_{i-1}\}$, where $a_i = min(X_i)$, we obtain an increasing sequence

$$a_0 < a_1 < a_2 < ...$$

along with the sequence $X_0, X_1, X_2, ...$. Since $X \subseteq \mathcal{N}$, $X = \{a_0, a_1, a_2, ...\}$. Define $f : \mathcal{N} \mapsto X$ by $f(n) = a_n$, it is evident that f is a bijection and $f(n+1) > f(n)$ (f is increasing).

Today computability theory is a mature field of mathematics, countability should be placed in the context of computability. The proof of *Proposition* 8.1.5 does not address the property of f, which is vital to decide if X is decidable, computable, or uncomputable (Proposition 2.8).

Proposition 5.1 *Proposition 8.1.5 is invalid.*

Proof. The proof of *Proposition* 8.1.5 does not address the computability of f, which is vital to decide if X is decidable or not as stated in Proposition 2.8. If X is decidable, then f should be computable and increasing. Hence, there exists a Turing machine M that computes f. However, when X is undecidable, we can still use M to compute f and arrive at a contradiction that X is decidable. Moreover, when X is computable but undecidable, the bijection f computed by M is still increasing. By Proposition 2.8, we need an non-increasing f. Hence, *Proposition* 8.1.5 is invalid. □

The claim that "every subset of $\mathcal{N}$ is countable" has several equivalent statements.

Proposition 5.2 *The following statements are logically equivalent:*

1. *Any subset of a countable set is countable.*
2. *Any subset of $\mathcal{N}$ is countable.*
3. *If there is an injective function from set S to $\mathcal{N}$, then S is countable.*
4. *If there is a surjective function from $\mathcal{N}$ to S, then S is countable.*

Proof. (1) $\Rightarrow$ (2): $\mathcal{N}$ is countable.

(2) $\Rightarrow$ (3): Let $f : S \mapsto \mathcal{N}$ be injective and $X = \{f(x) \mid x \in S\}$. Since $X \subseteq \mathcal{N}$, X is countable by (2). Note that $f : S \mapsto X$ is a bijection from S to X, that is, S and X have the same size. So S must be countable.

(3) $\Rightarrow$ (4): Let $g : \mathcal{N} \mapsto S$ be surjective, then for every $x \in S$, the set $g^{-1}(x) = \{y \mid g(y) = x\}$ is not empty. Define $f : S \mapsto \mathcal{N}$ by $f(x) = min(g^{-1}(x))$, then f is injective. By (3), S is countable.

(4) $\Rightarrow$ (1): Let $X \subseteq S$ and S be countable. If X is finite, then X is countable. If X is infinite, then S must be infinite and there exists a bijection $h : \mathcal{N} \mapsto S$. Let a be an element of X. Define $g : \mathcal{N} \mapsto X$ by $g(x) = h(x)$ if $h(x) \in X$, otherwise $g(x) = a$. Then f is surjective. By (4), S is countable. □

If every countable set has a computable counting bijections, the four statements in the above proposition are all false. Many textbooks claim that they are true statements, but their proofs are erroneous. If they are true, we may have an alternative definition of "countable": A set S is *countable* if there is an injective function from S to $\mathcal{N}$. Some textbooks use this definition [6]. This definition is much simpler than Cantor's definition because it does not care if S is finite or not and uses only injection, not bijection. This definition would make the countability proof simpler for some sets. Why didn't Cantor use this simpler definition? Our guess is that Cantor did not think that the new definition was equivalent to his.

When some subsets of a countable set are not countable, it is false to claim that $S \cup T$ and $S - T$ are uncountable if S is uncountable and T is countable. For instance, let S be the set of natural numbers representing E_{TM} (the emptiness problem of Turing machines) and $T = \mathcal{N}$, then S is uncountable and $S \subset T$. However, both $S \cup T = \mathcal{N}$ and $S - T = \emptyset$ are countable.

Several textbooks also contain the following statement (see Theorem B of Chapter 1 [11]; also [15] p. 14): "Every infinite set contains an infinite countable subset." Its proof has the same error as that of *Proposition* 8.1.5. It is a mission impossible to find an infinite enumerable subset of E_{TM}.

Since "computable" is a desirable property of a counting bijection and we believe that every countable set has a computable counting bijection, we propose the following conjecture for all countable sets.

Counting Bijection Thesis: *Every countable set has a computable counting bijection.*

To refute it, one must find a countable set that does not have any computable counting bijection. If this thesis is true, the word "enumerable" in Theorems 2.5, 2.6, 2.7, 3.5, and 4.4 can all be replaced by "countable".

We believe that Cantor would have accepted this thesis as a true statement. In the days of Cantor, the notion of "computable" was not formally defined. It is reasonable to guess that Cantor used counting bijections as others, assuming these bijections are supported by effective methods. Cantor used the verb "count" to indicate an operation is performed. If a bijection is uncomputable, by Church-Turing thesis, no methods exist to compute this bijection and no counting is possible. It is unimaginable to think that Cantor would have wanted an inoperable counting method.

If the thesis is true, then any unrecognizable language L, e.g., E_{TM}, is uncountable. This result is counter-intuitive because $L \subset \Sigma^*$. It is hard to believe that L is uncountable, because by Cantor's criterion, if L is uncountable, L is larger in size than Σ^*. The common belief is that a set is larger or equal in size than any of its subset. For finite sets A and B, if $A \subset B$, then $|A| < |B|$. We known $|\mathcal{N}| = |\mathcal{E}|$, where $\mathcal{E}$ is the set of all even natural numbers and $\mathcal{E} \subset \mathcal{N}$. Now we must accept $|L| > |\Sigma^*| = |\mathcal{N}|$ albeit $L \subset \Sigma^*$. In naïve set theory, allowing a set to be its own member will lead to a contradiction. Does $|L| > |\Sigma^*|$ lead to a contradiction in modern set theory? This is a topic for further research. We pose below some open questions in set theory and computability theory. Assume S is uncountable and T is countable.

1. Under what conditions $S \cup T$ is countable or not?
2. Under what conditions $S - T$ is countable or not?
3. Under what conditions $T - S$ is countable or not?
4. If $f : T \mapsto S$ is surjective, is f computable or not?
5. If $f : S \mapsto T$ is surjective, is f computable or not?
6. If $T = S \cup X$, under what conditions X is countable or not?

More open problems are given below:

1. Is the set of countable subsets of $\mathcal{N}$ countable or not?
2. Is the set of uncountable subsets of $\mathcal{N}$ countable or not?
3. Is there a countable set which has no computable counting bijections?
4. Is the set of computable functions countable or not?
5. Is the set of recognizable languages countable or not?

One of the sets in the first two questions must be uncountable because $\mathcal{P}(\mathcal{N})$ is uncountable. The third question concerns about the validity of the counting bijection thesis.

6 Conclusion

We say a set S is enumerable if either S is finite or S has a computable counting bijection. We showed that a formal language is recognizable iff it is enumerable. This equivalence provides useful insights to set theory and computability theory. We introduced the concept of *counting order* and procedure *deleteMin* to work with it. We showed that the existence of *deleteMin* as an algorithm is a sufficient and necessary condition for an enumerable set. We believe that every countable set has a computable counting bijection and took it as the *counting bijection thesis*. We strongly believe that Cantor would have accepted this thesis as a true statement. Assuming the thesis is true, a formal language is recognizable iff it is countable. We proved that the set of deciders (or algorithms) is unenumerable, hence uncountable if the counting bijection thesis is true. We showed that several statements taught in colleges, such as "every subset of a countable set is countable", lack valid proof. These statements are false if the counting bijection thesis is true. We listed some open questions for further research in set theory and computability theory.

The boundary between countable sets and uncountable sets was drawn by Georg Cantor in 1891. The boundary between decidable sets and undecidable sets was drawn by Kurt Gödel in 1931. The boundary between computable sets and uncomputable sets was drawn by Alan Turing *et al.* in the 1930s. Assuming the counting bijection thesis is true, then Cantor's boundary of countable sets coincides with Turing *et al.*'s boundary of computable sets. If we require a formal language to have an increasing and computable counting bijection, then Cantor's boundary coincides with Gödel's boundary of decidable sets (Proposition 2.8). Unlike Gödel, Church, and Turing, Cantor's definition of "countable" provides only a necessary property of a computable set, and Cantor did not provide a formal computing model to support his definition. However, we should not ask too much from a man who lived six decades earlier than the founding fathers of computer science. The findings of this paper support that Cantor is not only a founding father of set theory, but also a founding father of computer science.

Acknowledgment: The author thanks Jian Zhang and Tian Liu for providing helpful comments on a preliminary version of this article.

References

[1] Bauer, Andrej, "What is an explicit bijection?" (FPSAC 2019 slides), `math.andrej.com/all/`, retrieved Nov. 11, 2023

[2] Cantor, Georg, "Ein Beitrag zur Mannigfaltigkeitslehre", *Journal für die Reine und Angewandte Mathematik*, (1878) vol. 84: pp. 242–248.

[3] Church, Alonzo, "An Unsolvable Problem of Elementary Number Theory", *American Journal of Mathematics*, (1936) 58(2):345–363

[4] Copeland, B. Jack, "The Church-Turing Thesis", *The Stanford Encyclopedia of Philosophy*, Edward N. Zalta (ed.),

`plato.stanford.edu/archives/sum2020/entries/church-turing/`, retrieved Nov. 11, 2023

[5] Enderton, Herbert, "Elements of Set Theory", Academic Press, 1977

[6] Enderton, Herbert, "A Mathematical Introduction to Logic", (2nd Ed.) Academic Press, 2001

[7] Ferreiros, José, *Labyrinth of Thought: A History of Set Theory and Its Role in Mathematical Thought* (2nd ed.), Birkhäuser, 2007

[8] Gödel, Kurt, "On Undecidable Propositions of Formal Mathematical Systems", Princeton Lectures, 1934. Reprinted in *Collected Works I: Publications 1929–1936*, S. Feferman *et al.* (eds.), Oxford University Press, 1986. pp 338-371

[9] Goldrei, Derek, "Classic Set Theory for Guided Independent Study", Chapman & Hall/CRC, 1996

[10] Hrbacek, Karel, and Jech, Thomas, "Introduction to Set Theory", (3rd Ed.), Marcel Dekker, 1999

[11] Kleene, Stephen, *Introduction to Metamathematics*, Wolters Noordhoff Publishing, 1971

[12] Loehr, Nicholas A., *Bijective Combinatorics*, CRC Press (2011)

[13] Martin, John, "Introduction to Languages and the Theory of Computation", (4th Ed.) McGraw Hill, 2009

[14] Sipser, Michael, *Introduction to the Theory of Computation*, (3rd ed.), Cengage Learning (2014)

[15] Sridharan, Sriraman, and Balakrishnan, R., "Foundations of Discrete Mathematics with Algorithms and Programming", CRC Press, 2019

[16] Tao, Terence, "Infinite sets", *Analysis I.* Texts and Readings in Mathematics. Vol. 37 (Third ed.), Springer (2016) pp. 181–210

[17] Turing, Alan, "On Computable Numbers, with an Application to the Entscheidungsproblem", *Proceedings of the London Mathematical Society*, (1937) s2-42(1): 230–265

[18] Zhang, Hantao, and Zhang, Jian, *Logic in Computer Science: Problem Solving by Logic Tools*, Springer, *submitted.*

A Appendix: Basic Concepts

Definition A.1 *Let $\mathcal{N}$ denotes the set of all natural numbers. A set S is* countable *if either S is finite or there exists a bijection $r : S \mapsto \mathcal{N}$, where r is a total injective and surjective function. We will say r is the* rank *function of S and $r(x)$ is the* rank *of $x \in S$. S is* countably infinite *if S is both countable and infinite, or equivalently, S and $\mathcal{N}$ have the* same size. *S is* uncountable *if S is not countable.*

Given a set $\Sigma = \{a_0, a_1, ..., a_{k-1}\}$, $k > 0$, of symbols, called *alphabet*, let Σ^* denote the set of all strings of finite length built on the symbols from Σ. We show below that Σ^* is countable by defining a bijection $\gamma : \Sigma^* \mapsto \mathcal{N}$.

Let $\theta : \Sigma \mapsto \mathcal{N}$ be $\theta(a_i) = i$ for $0 \leq i \leq k-1$. The decimal value of $s \in \Sigma^*$, $v(s)$, is computed as follows: $v(\epsilon) = 0$ and $v(xa) = v(x) * k + \theta(a)$, where $x \in \Sigma^*$ and $a \in \Sigma$. If $|s| = n$, there are $(k^n - 1)/(k-1)$ strings of Σ^* whose length are less than n, and there are $v(s)$ strings of length n whose decimal values are less than $v(s)$. Define $\gamma : \Sigma^* \mapsto \mathcal{N}$ by $\gamma(s) = (k^{|s|} - 1)/(k-1) + v(s)$, then it is easy to check that γ is a bijection and the inverse of γ, γ^{-1}, is well-defined: For any $x \in \mathcal{N}$, $\gamma^{-1}(x) = s$, where $|s| = n = \lfloor log_k(x+1) \rfloor$ and $v(s) = x - (k^n - 1)/(k-1)$.

If we order the strings of Σ^* by γ, i.e., listing them as $\gamma^{-1}(0)$, $\gamma^{-1}(1)$, $\gamma^{-1}(2)$, and so on, this order will choose (a) shorter length first; (b) smaller decimal value first for strings of the same length. In this order, each $s \in \Sigma^*$ is preceded by $(k^{|s|} - 1)/(k-1)$ strings whose length is less than $|s|$, plus $v(s)$ strings of length $|s|$ and whose value is less than $v(s)$. Hence, the rank of s in this order is exactly $\gamma(s) = (k^{|s|} - 1)/(k-1) + v(s)$.

Definition A.2 *Let $\gamma : \Sigma^* \mapsto \mathcal{N}$ be the rank function in the above discussion. The order induced by γ is called the* canonical order *of Σ^* and is denoted by $>$.*

Example A.3 Let $\Sigma = \{a, b\}$, then the first few strings of Σ^* are given below with γ, where a^i denotes i copies of a:

$x \in \Sigma^*$	ϵ	a	b	a^2	ab	ba	b^2	a^3	a^2b	aba	ab^2	ba^2	bab	b^2a	b^3	a^4	a^3b	...
$\gamma(x)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...

Σ^* becomes binary strings if we replace a by 0 and b by 1, □

Σ^* plays a key role in theory of computation, and a *formal language* is any subset of Σ^*. The canonical order of Σ^* is uniquely defined by the rank function γ. Because of γ, we will speak of strings of Σ^* and numbers of $\mathcal{N}$ indifferently. The power set of Σ^*, $\mathcal{P}(\Sigma^*)$, and the power set of $\mathcal{N}$, $\mathcal{P}(\mathcal{N})$, have the same size, and they are among the first known uncountable sets.

For a formal definition of Turing machines and their encoding (denoted by $\langle M \rangle \in \Sigma^*$ for a Turing machine M), please see [14, 18]. Each Turing machine M as a *language recognizer* can also be viewed as a *decision function* $M : \Sigma^* \mapsto \{0, 1\}$: For any $w \in \Sigma^*$, $M(w) = 1$ means "M accepts w"; $M(w) = 0$ means "M rejects w"; if $M(w)$ is undefined, it means "M loops on w". If $M(w)$ is a total function, that is, M does not loop on any w, M is called a *decider*.

Definition A.4 *A formal language L is* recognizable *if there exists a Turing machine M such that $L = L(M) = \{w \in \Sigma^* \mid M(w) = 1\}$. $L(M)$ is the language* recognized *by M. Moreover, L is* decidable *if L is recognized by a decider.*

So, all formal languages are divided into three disjoint groups: (a) decidable, (b) recognizable but undecidable, and (c) unrecognizable. In literature, a recognizable language is also called *recursively enumerable*, *partially decidable*, *semi-decidable*, *computable*, or *solvable*. A decidable language is also called *recursive*, *total recognizable*, or *total computable.*[1]

Every set S has a *characteristic function* $p : S \mapsto \{0, 1\}$ such that $x \in S$ iff (if and only if) $p(x) = 1$. When S is a subset of $\mathcal{N}$ or Σ^*, p is said to be a *decision function*. Decision functions are a subset of general functions on $\mathcal{N}$ or Σ^*. We will accept *Church–Turing thesis* (also known as *computability thesis*, or *Church–Turing conjecture*), which says that if a function can be computed by any computing model, it can be computed by a Turing machine. That is, a function f is *computable* if it can be computed by a Turing machine M called *procedure*; f is *total computable* if M is an *algorithm*, which is a Turing machine and halts on every input.

Let $S = \{x \mid p(x) = 1\}$. We say S is *computable* if $p(x)$ is computable. So, if S is a formal language, then S is recognizable iff S is computable. S and $p(x)$ are *total computable* or *decidable* if S has a decider or an algorithm. S and $p(x)$ are *uncomputable* if they are not computable.

domain	**distinction**	**name of Turing machines**
formal languages	decidable	decider
	recognizable	recognizer
decision problems	decidable	decider
	computable	procedure
integer functions	total computable	algorithm
	computable	procedure

[1]Some researchers use "computable"[14] or "solvable" for "decidable"; and "partially computable" or "partially solvable" for "recognizable". We use the word "enumerable" differently.