1 Background

We give a basic overview of the mathematical background required for this course.

1.1 Set Theory

We introduce some concepts from naive set theory (as opposed to axiomatic set theory). The first part covers the basic definitions of set theory, and is covered by any basic course in discrete mathematics. The second part covers some results regarding the cardinality of sets, and is necessary for the arguments that you will encounter in this course.

Set Theory - Basic definitions

A set is informally defined as a collection of objects. The concept of a set in this setting is not precisely defined. If x is a member of the set A, we say x belongs to A, and write $x \in A$. If, every element of a set A belongs to a set B, we say that A is a subset of B, written as $A \subseteq B$. If both $A \subseteq B$ and $B \subseteq A$, we say that A and B are equal, written as $A = B$.

The operations on sets that we typically consider are the following.

1. Union: If A and B are sets, their union, written $A \cup B$ is defined as
$$A \cup B = \{x \mid x \in A \text{ or } x \in B\}.$$
2. Intersection: If A and B are sets, their intersection, written as $A \cap B$, is defined as
$$A \cap B = \{x \mid x \in A \text{ and } x \in B\}.$$
3. Relative Complement: If A and B are sets, then the complement of B relative to A, written as $A \setminus B$, is defined as
$$A \setminus B = \{x : x \in A \text{ and } x \notin B\}.$$

If we have the notion of a universal set, denoted U, the superset of all sets in the domain of interest, then it is possible to define the following notion.

5. Absolute Complement: If A is a set, the absolute complement of A, written A^c is defined as $U \setminus A$.

Theorem 1.1.1 (De Morgan's Laws)**.** *The following hold for arbitrary sets A and B.*

$$(A \cap B)^c = A^c \cup B^c.$$

and

$$(A \cup B)^c = A^c \cap B^c.$$

We have the notion of a power set, which is the collection of all subsets of a set.

6. Power Set: If A is a set, then there is a set, called the power set of A, written $\mathcal{P}(A)$ (or, as 2^A), is the set of all subsets of A. In symbols,

$$\mathcal{P}(A) = \{B \mid B \subseteq A\}.$$

The next operation that we define is the Cartesian product of sets. For this, we have to first define what an ordered pair of two objects is. A sequence of elements

$$abcd$$

where we say that the first element is a, the second b, the third c and the fourth d, in the specified order is actually a function $f : \{0, 1, 2, 3\} \to \{a, b, c, d\}$ specified by

$$f(0) = a, f(1) = b, f(2) = c, f(3) = d.$$

Since we have not defined functions yet, we cannot use this approach to define an ordered pair as a collection (a, b) with the first element being a and the second being b. We proceed as follows.

Definition 1.1.2. The *ordered pair* of a_1 and a_2 with first coordinate a_1 and the second coordinate a_2, written (a_1, a_2) is defined by

$$(a_1, a_2) = \{\{a_1\}, \{a_1, a_2\}\}.$$

Though this might appear strange at first, we have succeeded in defining an ordered pair of arbitrary elements without appealing either to the concept of an index (or a number) or a mapping from an index to an element. The first coordinate can now be specified using only the set operations thus, as *the unique element of*[1]

$$\bigcap_{A \in (a,b)} A = \{a_1\} \cap \{a_1, a_2\}$$

[1]Thanks to Pervez Khan for correcting some errors in a previous version of the definition.

The second co-ordinate may be defined as *the unique element of*

$$\left[\bigcap_{A\in[(a,b)\setminus\{a_1\}]} A\right] - \{a_1\}.$$

This strategy may be extended to define the ordered tuples of arbitrarily many elements. With this definition, it is now possible to define the Cartesian Product of sets.

7. Cartesian Product: If A and B are two sets, their cartesian product of A with B, written $A \times B$, is defined as:

$$A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}.$$

A *relation between two sets A and B* is just a subset of the cartesian product $A \times B$. A is called the *domain* of the relation, and B its *range*.

Definition 1.1.3. A *function f from A to B*, written $f : A \to B$, is a relation from A to B with the restriction that for every element a in A, there is exactly one ordered pair $(a, b) \in f$. A is called the *domain* of f and B its *range*. For any ordered pair $(a, b) \in f$, we say that b is the *image of a under f*, written $f(a)$. The *preimage of* $b \in B$ *under f* is the set of all elements $a \in A$ such that $f(a) = b$.

Example 1.1.4. *If $A = \{a, b, c, d\}$ and $B = \{1, 2, 3, 4\}$, and $f = \{(a, 1), (b, 1), (c, 3), (d, 4)\}$, then $f(a) = 1$, $f(b) = 1$, and $f^{-1}(1) = \{a, b\}$.*

We can restate the restriction of f as saying that every element in A has exactly one image in B.

We specify the following classes of functions for later discussion.

A function $f : A \to B$ is said to be *injective, or one-to-one* if, the elements of A have distinct images in B. In symbols,

$$\forall a_1, a_2 \in A \quad f(a_1) = f(a_2) \text{ implies } a_1 = a_2.$$

The function f is said to be *surjective, or onto* if every element in B has a pre-image. In symbols,

$$\forall b \in B \exists a \in A \quad f(a) = b.$$

If f is both one-to-one and onto, then f is said to be *bijective*.

Cardinality

We now discuss questions of cardinality, which will be used to compare the sizes of sets. For any finite set, we can define the cardinality as the number of elements in it. Suppose A and B are finite sets. It is easy to show that if A and B have equal cardinality, then there is a bijection $f : A \to B$, . If the cardinality of A is at most that of B, then there is a one-to-one function $f : A \to B$. If the cardinality of B is at most that of A, then there is an onto function $f : A \to B$.

When we consider infinite sets, however, this approach falters. We have no way of distinguishing among the cardinalities of infinite sets if we use this concept. We also run into seeming paradoxes: Galileo posed the question, "Which set is bigger - the set of even numbers or the set of numbers?" Obviously, the set of even numbers is a strict subset of the set of numbers. But, if we multiply a natural number with 2, then we get a unique even number, thus there is a bijection between these two sets. This example shows the puzzling nature of infinite sets – they can be isomorphic to strict subsets of themselves. A way to resolve the questions of cardinality consistently is to take the reverse approach of our initial approach to the cardinality of finite sets: Say, for example, that cardinality of A is less than or equal to that of B if there is a one-to-one function $f : A \to B$. We elaborate this approach, and study the properties of cardinalities of infinite sets, and delve into some famous examples.

This treatment follows Halmos. If there is a injection $f : A \to B$, then we say that A is dominated by B, and write $A \lesssim B$.

This suggests the idea that A and B are equivalent (not equal sets), written $A \sim B$ if there is a bijection from $f : A \to B$. In the case of comparison operators between numbers m and n, we know that if $m \leq n$ and $n \leq m$, then $m = n$. Can we define a notion that if $A \lesssim B$ and $A \gtrsim B$, then $A \sim B$?

This is a very difficult theorem in set theory, known as the Cantor-Schröder-Bernstein Theorem.

Theorem 1.1.5 (Cantor-Schröder-Bernstein)**.** *Let A and B be two arbitrary sets. If there is a one-to-one function $f : A \to B$ and a one-to-one function $g : B \to A$, then there is a bijection $h : A \to B$.*

We omit the proof, and go into the discussion of countability of sets. An infinite set is one, where, even if we remove any finite number of elements, there are still elements left. Removing an element is a process of assigning a natural number index to some particular element of the set - the only thing we forbid is repetition, since no element can be removed multiple times. This gives us the following definition of an infinite set.

Definition 1.1.6. A set A is infinite if $\mathbb{N} \lesssim A$.

This gives us an idea of the "smallest" kind of infinite sets. If an infinite set $A \lesssim \mathbb{N}$, then $A \sim \mathbb{N}$. We call such sets countably infinite sets. Countably infinite sets have the same "cardinality" as $\mathbb{N}$. It is customary to call *countable* any set that is either finite, or is countably infinite.

Definition 1.1.7. A set A is said to be *countable* if there is an onto function $f : \mathbb{N} \to A$.

The set of natural numbers $\mathbb{N}$ is an obvious example of such a set. The set of even numbers is also a countably infinite set. We show that the set of rational numbers (usually written as $\mathbb{Q}$), is also countable, in the following sequence of theorems.

First, we show that if we take a finite union of countable sets, the resulting set is countable.

Theorem 1.1.8. *If $A_0, A_1, \dots, A_{n-1}$ are pairwise disjoint countable sets, then their union is also countable.*

Proof. Since each A_i, $i = 0$ to $n-1$, is countable, there are functions $f_i : \mathbb{N} \to A_i$ which are onto. We construct a function $g : \mathbb{N} \to \cup_{i=0}^{n-1} A_i$ to show that the union of these sets is countable.

Let $g(i) = f_{i \bmod n}(i/n)$ where / denotes integer division.

If each $f_i : \mathbb{N} \to A_i$ is onto, we can see that $g : \mathbb{N} \to \cup_{i=1}^{n} A_i$ is also onto. For, the k^{th} element in the m^{th} set will be $g(k * n + m)$ (We start counting from 0). $\square$

Intuitively, if we let all A_is be countably infinite, we can explain as follows: g enumerates the elements of the union in the form of a two-dimensional array: there are exactly n columns, the column i enumerating the elements of A_i. g goes row-by-row, enumerating the first elements of all the A_is first, then the second elements and so on. In this manner, since each A_i is countable, we can count all the elements in the union.

Theorem 1.1.9. *A union of countably many finite sets is countable.*

Intuitively, since each A_i is finite, we can enumerate the elements of A_0 until they are exhausted, and then start enumerating the elements of A_1

and so on. [2] In this way, there is a finite index at which any given element of a given set will be enumerated.

Proof. Consider

$$\mathcal{A} = \bigcup_{i=0}^{\infty} A_i$$

where each A_i is a finite set. Then, if we construct an onto function $g : \mathbb{N} \to \mathcal{A}$, it will prove that $\mathcal{A}$ is countable.

Let $|A_i|$ denote the number of elements in A_i. Since each A_i is finite, there is a bijection $f_i : A_i \to \{0, 1, \ldots, |A_i| - 1\}$. Since

$$|A_0|, \quad |A_0| + |A_1|, \quad |A_0| + |A_1| + |A_2|, \ldots$$

is an increasing sequence, for any j, there is a least index n such that

$$j < |A_0| + |A_1| + \cdots + |A_n|.$$

Then

$$g(j) = f_n\left(j - [|A_0| + |A_1| + \cdots + |A_{n-1}|]\right).$$

To show that g is onto, we have to show that for any arbitrary i, and m such that $0 \leq m < |A_i|$, there is some index j such that $g(j) = f_i(m)$. This follows routinely from the definition of g. □

The above theorem is one of the most useful tools in proving the countability of sets - when we have to prove that some class of sets is countable, what we have to do essentially is to rethink of the same class as a countable union of finite sets. A brilliant example is the following theorem.

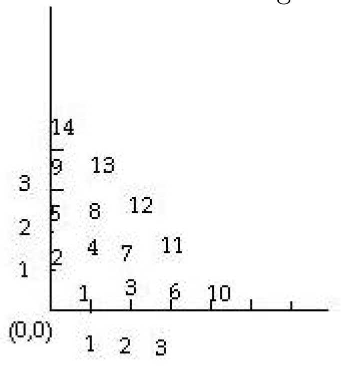
With this, we now can show that the set of non-negative rational numbers is countable. We will actually show something more, that $\mathbb{N} \times \mathbb{N}$ is countable. Since every non-negative rational number can be represented as a ratio $\frac{p}{q}$ where p is nonnegative and $q \neq 0$, what we prove amounts to proving that a bigger set is countable.

Theorem 1.1.10. *The cross-product of two countable sets is countable.*

The proof uses a technique which is sometimes picturesquely called "dovetailing". The idea can be paraphrased as follows. Consider a rectangular lattice where the "x-axis" corresponds to the elements of a countable set A and the "y-axis" corresponds to a countable set B. (Figure 1).

[2] This could not be done in the previous theorem because each A_i was infinite, so we would never have been done enumerating A_0 at a finite time.

Figure 1: Illustration of dovetailing



Proof. There are onto functions $g : \mathbb{N} \to A$ and $h : \mathbb{N} \to B$ which are given. We define an onto function $f : \mathbb{N} \to A \times B$ that enumerates ordered pairs in in $A \times B$ in the following manner. First, $(g(0), h(0))$ is enumerated. In the next pass, we enumerate $(g(1), h(0)), (g(0), h(1))$. In the following pass, we enumerate $(g(2), h(0)), (g(1), h(1))$ and $(g(0), h(2))$.

Thus in pass c, we enumerate pairs $(g(i), h(j))$ such that $i + j = c$. This enables us to distinguish among different passes. Within the same pass, we can distinguish among the pairs

$$(g(i+j), h(0)),\ (g(i+j-1), h(1)),\ \dots,\ (g(0), h(i+j))$$

by their second co-ordinates. Thus there are $i + j + 1$ elements in pass $i + j$.

Let a pair $(g(i), h(j))$ be given. Then it will be enumerated in pass $i + j$. Let us denote $i+j$ by c. The total number of elements in passes $0, 1, \dots, c-1$ is equal to

$$T[c-1] = \sum_{i=1}^{c-1} (i + 1) = \frac{c(c+1)}{2}.$$

Now, in pass c, the pair $(g(i), h(j))$ will be enumerated in position j.

Thus, the onto function $f : N \to A \times B$ we define is

$$f(T[i + j - 1] + j) = (g(i), h(j)).$$

□

However, there are sets of higher cardinalities. We prove a particular case of the general theorem. This proof follows what is usually called a "diagonalization" argument. (Please don't confuse this with "dovetailing"!)

Theorem 1.1.11. *There is no bijection $f : \mathbb{N} \to \mathcal{P}(\mathbb{N})$.*

Proof. Suppose there is a bijection $f : \mathbb{N} \to \mathcal{P}(\mathbb{N})$. Define the following set.

$$S = \{i : i \notin f(i)\}.$$

Then, we prove that there is no j such that $f(j) = S$, thus contradicting that every subset of $\mathbb{N}$ has been enumerated by f. For, if we consider whether j is present in S, we find that if j is in the set $f(j)$, then it is not in S. Conversely, if j is not in the set $f(j)$, then it is in S. Thus S differs from the set $f(j)$ in at least one element, hence cannot be the same as $f(j)$.

Since this is true for any j, S is not in the image of f. □

We briefly describe the characteristic function of a set of numbers. Let S be a set of numbers. The characteristic function $\chi_S : \mathbb{N} \to \{0, 1\}$ is the function defined by $\chi_S(i) = 1$ if $i \in S$ and $\chi_S(i) = 0$ otherwise. Intuitively, this can be seen as a "bit-map" of S. Every set of numbers has a unique characteristic function.

The cardinality of the power set of natural numbers is the same as the cardinality of the real numbers in $[0, 1)$ (the unit interval with all numbers r where $0 \leq r < 1$).

To see this, we use the binary representation of the real numbers. A set of numbers can be represented by its characteristic sequence $b_0 b_1 b_2 \dots$ where $b_i = 1$ if s_i is present in the set and $b_i = 0$ otherwise. Let us define a mapping from such a sequence to the number $\sum_{i=1}^{\infty} b_{i-1}/2^i$. This defines an onto function from $\mathcal{P}(\mathbb{N})$ to $[0, 1)$, the cardinality of $[0, 1)$ is at most as much as $\mathcal{P}(\mathbb{N})$.

To see that the cardinality of $[0, 1)$ is at least that of $\mathcal{P}(\mathbb{N})$, we define an onto function from $[0, 1)$ to $\mathcal{P}(\mathbb{N})$.

It is natural to consider the mapping defined by the binary representation of reals in the interval. However, wee cannot use the binary representation of real numbers in the unit interval for this purpose, since dyadic rationals, that is, rationals of the form $i/2^m$, have two binary representations. For example, 0.5 has the representations $0.1000\dots$ and $0.0111\dots$. If we pick one representation, and define $g(0.5) = 1000\dots$, where the binary sequence is the characterestic sequence of a set, then we do not have an onto function – The set defined by $0111\dots$ does not have an inverse image.

To solve this issue, we adopt the ternary notation of the reals in the interval. We define an onto function from $[0, 1)$ to $\mathcal{P}(\mathbb{N})$ as follows. If a ternary expansion of a number does not have the digit 2, that is, the number can be written in the form $\sum_{i=1}^{\infty} t_{i-1}/3^i$ where each t_{i-1} is 0 or 1, then its image is the set represented by the characteristic sequence $t_0t_1t_2\dots$. There are numbers in the interval with infinitely many 2s in their ternary expansion - the image of such numbers is defined to be $\mathbb{N}$, merely so that we have a total function from $[0,1)$ to $\mathcal{P}(\mathbb{N})$. Since this function is onto, we can conclude that the cardinality of $[0,1)$ is at least that of $\mathcal{P}(\mathbb{N})$.[3]

1.2 Computability Theory

In this part, we will briefly cover notions of computability. We will first define a Turing machine which accepts/decides a language, and then define the notion of a computable function.

Definition 1.2.1. A one-way infinite, 2-tape Turing Machine is an 8-tuple $(Q, \Sigma, \Gamma, B, \delta, q_0, q_a, q_r)$ where

- Q is the set of states of the Turing machine.
- Σ is the finite set of input alphabet symbols.
- Γ is the finite set of tape alphabet symbols, $\Sigma \subset \Gamma$. Here, Σ a strict subset.
- B is a reserved tape alphabet symbol called the blank, not part of the input alphabet. ($B \in \Gamma - \Sigma$.)
- $$\delta : (Q - \{q_a, q_r\}) \times \Gamma^2 \to Q \times \Gamma^2 \times \{L, R\}^2$$
  is the transition function of the Turing machine.
- $q_0 \in Q$ is the initial state of the Turing machine.
- $q_a \in Q$ is the accept state.
- $q_r \in Q$ is the reject state.

A *configuration* of the Turing machine consists of the state, the contents of the 2 tapes, and the position of the tape heads. An input string w is said to be *accepted* by a Turing machine if, the computation of M with initial configuration having w on the first tape and both heads at the left end of w, terminates in q_a. The machine is said to *reject* the string if the Turing machine terminates in q_r. (There is of course, the possibility that the Turing Machine may not terminate its execution.)

[3]Thanks to Arpita Korwar and Rohit Gurjar for pointing this out.

A language L is a set of strings. Recall that a Turing machine is said to *accept* a language L if every string x in the language is accepted by the Turing Machine in the above sense, and no other string is accepted. If a string is not in the language, then it might either be rejected by the Turing machine, or it may cause the Turing machine to run forever.

Definition 1.2.2. A language is said to be *acceptable* if there is a Turing machine which accepts it.

Definition 1.2.3. A language L is said to be *decidable* if both L and L^c are acceptable.

Equivalently, we may define a decidable language L as follows. There is a Turing machine which accepts every string in the language, and rejects every other string. (HW 1A)

It is possible to define a Turing machine that *computes* a function from strings to strings. Then we will proceed to define a computably enumerable language.

Definition 1.2.4. A one-way infinite, 2-tape Turing Machine is said to compute $f : \Sigma^* \to \Sigma^*$ if the machine, presented with the input string x in the initial configuration, terminates in state q_a with $f(x)B$ on the second tape whenever $f(x)$ is defined, and runs forever if $f(x) \uparrow$.

A function f is said to be *partial computable* if there is a Turing machine which computes f in the above sense. A partial computable function f is called *total computable* if there is a Turing machine computing f that halts on all inputs.

To define functions which operate on numbers, it is necessary to pick a representation of numbers. The standard enumeration of strings is the enumeration

$$\lambda, 0, 00, 01, 10, 11, 000, \ldots$$

- that is, the enumeration which lists all strings of a shorter length before any longer string, and within any length, lists strings in lexicographic order. For any number i, let s_i denote the i^{th} string in this sequence. Then $s : \mathbb{N} \to \Sigma^*$ is a bijection between numbers and strings, and can be used as a representation of numbers.

(There is an easy closed form formula for s, that is, given a number $n > 0$, find s_n. Convert $n+1$ to binary, and strip the leading 1 in the binary representation. This will give s_n. (Please verify!))

Definition 1.2.5. A function $f : \mathbb{N} \to \mathbb{N}$ is partial computable if there is a partial computable function $h : \Sigma^* \to \Sigma^*$ such that for every number $n \geq 0$

$$f(n) = s^{-1}h(s_n).$$

It is now routine to define unary partial computable functions which map any given countably infinite domain to any countably infinite range. We will use the following function to represent a pair of strings.

Definition 1.2.6. The bit-doubling function

$$\mathrm{bd} : \Sigma^* \to \Sigma^*$$

is defined by

$$\begin{aligned} \mathrm{bd}(\lambda) &= \lambda \\ \mathrm{bd}(x0) &= \mathrm{bd}(x)00 \\ \mathrm{bd}(x1) &= \mathrm{bd}(x)11 \end{aligned}$$

The bit doubling function of x is a string in which every bit in x is doubled. For example, the bd(0010) = 00 00 11 00.

Definition 1.2.7. The *pairing function*

$$\langle , \rangle : \Sigma^* \times \Sigma^* \to \Sigma^*$$

is defined by

$$\langle x, y \rangle = \mathrm{bd}(x)01\mathrm{bd}(y)01$$

where x and y are strings.

We observe that the number of times "01" occurs in any encoding, is exactly 2, and all valid encodings end in 01.

Lemma 1.2.8. *The pairing function is a prefix code - that is, the encoding of a pair cannot be the prefix of the encoding of another pair.*

Proof. Suppose

$$\mathrm{bd}(x)01\mathrm{bd}(y)01w$$

is the encoding of some pair. Since the number of times "01" can occur in the encoding is 2, "01" is forbidden from w. However, the suffix of the encoding $01w$ should end in "01". These conditions together imply that $w = \lambda$.

Thus, the encoding of a pair cannot be the prefix of another pair. □

For ease of description, it is preferrable to have a high-level description of Turing Machines so that we do not have to give detailed programs for doing elementary operations every time we specify a Turing machine. For this, the common approach is to use a high-level language, and prove that every elementary operation, as well as every programming construct like loops, can be simulated by a Turing machine. Once we do this, it follows that every program we write in the high level language corresponds to a Turing Machine. We will assume that programs written in an ALGOL-like language can be computed using Turing machines. We will describe our programs using this toy language.

Now, we can make the following analogy:

partial computable: total computable :: acceptable : decidable.

We will explore this analogy. For this, we need to now define Turing machines which *enumerates* a language. This is the computable analogue of an onto function $f : \mathbb{N} \to L$ with which we defined countability of L. Intuitively, Turing machine that enumerates L will "keep printing" the elements of L one after another. However, we would like every Turing machine outputting a string to halt in a finite time, so we model the machine such that for a given n, it will output the n^{th} element in the enumeration.

Definition 1.2.9. A language L is said to be *computably enumerable* if it is either finite, or there is a *total computable* bijection $f : \mathbb{N} \to L$.

Note that, L can be the empty language.

Theorem 1.2.10. *A language is computably enumerable if and only if it is acceptable.*

Proof. We first prove that if a language is computably enumerable, then it is acceptable. If a language is finite, then it is acceptable by a Turing Machine which hardcodes the strings in the language in a finite lookup table, and accepts exactly those strings. Suppose a computably enumerable language L is infinite. Then there is a computable bijection $f : \mathbb{N} \to L$. We describe a Turing machine that accepts L as follows. (This is not the formal description, but merely the pseudo-code description of the Turing Machine. There is a Turing Machine which corresponds to this, using the Church-Turing thesis.)

```
1. input x.
2. Initialize integer i to 0.
3. while true do
```

```
4.        if f(i) == x then
5.           accept x, halt.
6.        else increment i to i+1.
7.        endif
8. done
```

The above machine accepts and halts all x in L, and it does not accept any other string. Hence L is Turing acceptable.

Conversely, we assume that L is an acceptable language. If L is finite, then by definition, L is computably enumerable. Assume that L is infinite. There is a Turing machine M, that, given any input x accepts if and only if $x \in L$.

Given an input $i \geq 0$, we will dovetail the execution of M on $\lambda, 0, 1, \ldots$ until i strings have been accepted and output the last string that has been accepted.

We describe the pseudo-code as follows.

```
1. input number i.
2. Initialize Set and oldSet to $\emptyset$; and $m$ to 0.
3. while |Set| $\leq i$ do
4.        if $s_m$ is of the form $\langle x, s_n \rangle$ do
5.           Run $M(x)$ for $n$ steps.
6.           if $M(x)$ has accepted then
7.              oldSet := Set.
8.              Set := Set $\cup \{x\}$.
9.           endif
10.       endif
11.       Increment $m$.
12. end while
13. Output oldSet $-$ Set.
```

Since L is infinite, we know that this algorithm will enable us to output distinct elements in L for every number i, so M computes a one-to-one function. Also, every $x \in L$ is accepted in some finite number of steps N, so the dovetailing will output x when $m \leq \langle x, s_N \rangle$. □

There is also a characterization of decidable languages in this manner.

Theorem 1.2.11. *A language is decidable if and only if it is computably enumerable in increasing order. That is, a language L is decidable if and only if it is finite or there is a total computable bijection $f : \mathbb{N} \to L$ such that for all numbers n,*

$$f(n) < f(n+1).$$

Proof. If L is finite, then it is decidable. Suppose L is not finite, and there is a bijection $f : \mathbb{N} \to L$ and for all n,

$$f(n) < f(n+1).$$

Then, we can build a Turing Machine which decides L in the following manner.

```
1. input x.
2. Initialize $i$ to 0.
3. while$f(i) < x$ do
4.         Increment $i$ by 1.
5. end while
6. if $f(i) == x$ then
7.         Accept $x$ and halt.
8. else reject $x$ and halt.
```

Conversely, suppose L is decidable. If L is finite, then it is computably enumerable by definition. If L is infinite, then there is a total computable bijection $f : \mathbb{N} \to L$ that enumerates L in increasing order. In particular, there is a total computable bijection f that enumerates L, hence L is computably enumerable. □

It is now easy to prove the following theorem, which we use later in the course.

Theorem 1.2.12. *Every infinite computably enumerable set contains an infinite decidable subset.*

Proof. □

We use the following fact from computability theory.

Theorem 1.2.13. *There is a universal Turing machine.*

Theorem 1.2.14. *(Kleene's normal form theorem) There is a 3-ary partial computable function C and a 1-ary partial computable function U such that any 1-ary partial recursive function can be expressed as*

$$f_e(n) = U(\mu z[C(e, n, z) = 0]).$$

Here, μz denotes the least z such that.

Theorem 1.2.15. *There is a partial computable function that is not total computable.*

Proof. We define the following partial computable function $f : \mathbb{N} \to \mathbb{N}$ as

$$f_e(n) = U(\mu z[C(n, n, z) = 0]) + 1.$$

So $f(n) = f_n(n) + 1$ when the value of $f_n(n)$ is defined, and is undefined otherwise. $f(n)$ is a partial computable function by construction. But f is not a total computable function:

Suppose f is total computable. Then there is an index e in the computable enumeration of partial computable functions such that $f_e = f$. This would imply that $f(e) = f_e(e) + 1 > f(e)$, which is a contradiction. □