Notes on Computability

Anagha

Introduction

This report was done as a part of the Twoples Reading Program in Mathematics, under the guidance of Dr. Waseet Kazmi. During the course of this project, we managed to covers the basics of **Computability Theory**: partial and total computable functions, diagonalization, recursion theorems, and Turing Reduction. The notes closely follow the book on Computability Theory by Rebecca Ward. The notes thus far are short and succinct; I hope to furnish further details in the next revisions.

Defining Computability

First, we focus on computable functions.

On functions

We focus particularly on these aspects of functions:

1. Limits: These functions are whole number valued, and their limits are convergent.
2. We deal with partial functions: A partial function f from a set X to a set Y is a function from a subset S of X (might be proper or improper) to Y. Then S is called the domain of definition or Dom(f). One can state the following for any partial function:

Let $f : X \rightharpoonup Y$ be a partial function. $\forall x \in X$,

- $\exists y \in Y$ such that $f(x) = y$
- otherwise, $f(x)$ is undefined.

If the subset S is improper, the function is said to be total.

Halting Criteria:

If $x \in$ Dom(f), then we say the computation halts (or converges), or $f(x) \downarrow$. If $x \notin$ Dom(f), we say the computation diverges and is denoted by $f(x) \uparrow$.

Equality of functions

Two total functions f and g are said to be equal ($f = g$) if $(\forall x)(f(x) = g(x))$. For partial functions,

$$f = g \rightarrow (\forall x)(f(x) \downarrow \Leftrightarrow g(x) \downarrow) \& (f(x) \downarrow = y \rightarrow g(x))$$

Characteristic function

Let A be a set. The characteristic function of A, denoted by χ_A is defined as:

$$\chi_{A(n)} = \begin{cases} 1 \text{ if } n \in A \\ 0 \text{ if } n \notin A \end{cases}$$

Turing Machines

A Turing Machine is a mathematical model that abstracts computation. More precisely, any Turing machine (TM) can be specified as a 4 tuple $\langle q_0, a, b, q_1 \rangle$. Here, q_0 is the start state of the TM, a is the current symbol being read, b is the instruction of the head as to which direction to move to (L or R), and q_1 is the final state.

Any computation that halts (and gives an output) uses only finitely many squares of the tape.

Exercises

1. **Write a Turing machine to flip the bits of an input string, so ∗011010∗ becomes ∗100101∗.**

Flip the current symbol being read, and move right. Go to the final state when a blank is read.

2. **Write a Turing machine that adds 1 to an input in tally notation**

- Let the TM be in state q_0 with the head at the first 1.
- Move right until you hit the first blank.
- Write 1, then halt.

Mathematically, these are the rules:

- $\langle q_0, 1, R, q_0 \rangle$
- $\langle q_0, B, 1, q_1$

Here q_1 is the final state.

3. **Binary addition**

- $\langle q_0, 0, 1, q_1 \rangle$
- $\langle q_0, 1, 0, q_2 \rangle$
- $\langle q_2, 0, R, q_0 \rangle$
- $\langle q_0, B, 1, q_1 \rangle$

Partial Recursive Functions

Primitive Recursive Functions

The primitive recursive functions form the smallest possible class C such that the following hold:

1. $\forall x \in \text{Dom(f)}$, $S(x) = x + 1 \in C$ (Successor Function)
2. All constant functions are in C: $M_m^n(x_1, x_2, ..., x_n) = m \ \forall m, n \in \mathbb{N}$
3. The projection functions are all in C: $P_i^n(x_1, x_2, ..., x_n) = x_i \ \forall n \geq 1, 1 \leq i \leq n$

The following two rules account for the closure:

4. Composition
5. Recursion

Examples

1. $g(x, y) = x \cdot y$ is primitive recursive

Soln:

$$g(x, 0) = 0$$

$$g(x, y + 1) = g(x, y) + x$$

2. $g(x, y) = x^y$ is primitive recursive.

Soln: We already know that $g(x, y) = x \cdot y$ is primitive recursive. $g(x, 0) = 1$

$$g(x, y + 1) = x \cdot g(x, y)$$

3. $n! = n \cdot (n - 1) \cdots 2 \cdot 1$ is primitive recursive.

Soln We already know that $g(x, y) = x \cdot y$ is primitive recursive.

- $0! = 1$
- $n! = n \cdot (n - 1)!$

4. **Consider a grid of streets, n east-west streets crossed by m north-south streets to make a rectangular map with nm intersections; each street reaches all the way across or up and down. If a pedestrian is to walk along streets from the northwest corner of this rectangle to the southeast corner, walking only east and south and changing direction only at corners, let $r(n, m)$ be the number of possible routes. Prove r is primitive recursive.**

Soln:

$$r(n, 0) = 1$$

$$r(n, m) = r(n - 1, m - 1) + r(n - 1, m)$$

Logic is very similar to construction of Pascal's triangle. $r(n, m) = \binom{n}{m}$

5. It is routine to check the following are also primitive recursive:
- min(x, y)
- max(x, y)

Non primitive recursive function

We now encounter our first total recursive function that is not primitive recursive: the Ackermann function. We give the following definition:

$$A(m,n) = \begin{cases} n{+}1 \text{ if } m{=}0 \\ A(m{-}1,1) \text{ if } m{>}0 \wedge n{=}0 \\ A(m{-}1,A(m,n{-}1)) \text{ if } m{>}0 \wedge n{>}0 \end{cases}$$

Proposition: Let $\theta(n)$ be any primitive recursive function. Then, the following holds:

$$(\exists N)(\forall m)(m > N \Rightarrow A(m,m) > \theta(m))$$

This means that the Ackermann function grows faster than any primitive recursive function.

Partial Recursive Functions: Unbounded Search

We shall add one more closure property to accommodate functions like the Ackermann function. This set is clearly larger than that of primitive recursive functions. First, a definition:

"The class of partially recursive functions is the smallest class of functions satisfying the 5 properties primitive recursive functions satisfy, along with the additional closure property of unbounded search."

Unbounded search, Minimization or μ recursion Suppose $\overline{x} = (x_1, x_2, ..., x_n)$. Let $\theta(\overline{x}, y)$ be a primitive recursive function in n+1 variables. Now, define $\psi(\overline{x})$ to be the smallest y such that $\theta(\overline{x}, y) = 0$, and $\forall z < y\ \theta(\overline{x}, z)$ is (well) defined.

- This closure property adds the sense of partiality; everything we had seen earlier were total recursive functions.
- In simple terms, the μ operator searches for the least possible natural number with some given property. Hence it is called the minimization operator.

Examples

1. $\mu(x > 5) = 6$

It returns the least natural number greater than 5.

2. Suppose $\theta(x,y) = \begin{cases} 0 \text{ if } x \text{ even} \\ 1 \text{ if } x \text{ odd} \end{cases}$

Then

$$\mu(\theta(x,y)) = \begin{cases} 0 \text{ if } x \text{ even} \\ \infty \text{ if } x \text{ odd} \end{cases}$$

On Coding and Countability

While working with domains that aren't explicit $\mathbb{N}$, it is sufficient to encode the elements of that domain as elements as $\mathbb{N}$. Suppose the set is S. Effectively, this is the same as setting up a bijection between S and $\mathbb{N}$.

Effective Countability

- The sets for which such bijections exist are called **effectively countable**
- $\forall s \in S$, im(s) is called its **code**
- It is important to worry about the finiteness of such sets.

- Subsets of effectively countable sets are not necessarily effectively countable.

A TM can:
- Decode the input, perform computation, encode the output
- Perform computation on encoded input and get encoded output

Examples:
1. Give an encoding of $\mathbb{Z}$ into $\mathbb{N}$

Soln: An intuitive choice would be

$$f(k) = \begin{cases} 2k \text{ if } k \geq 0 \\ -(2k+1) \text{ if k<0} \end{cases}$$

Pairing Function

A pairing function is a bijection

$$\pi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$

$$\pi(x+y) \coloneqq \frac{1}{2}(x+y)(x+y+1) + x$$

For more variables, we can iterate as

$$\pi(x, y, z) = \pi(\pi(x, y), z)$$

This construction is a generalization of Cantor's proof that $|\mathbb{Q}| = |\mathbb{N}|$

This tells us that $\forall k, \mathbb{N}^k$ is always effectively countable.

Examples: Effectively Countable Sets
1. The set of finite tuples of arbitrary length $\cup_{k \geq 0} \mathbb{N}^k \to N$ is effectively countable.

Consider the following map:

$$\tau : \cup_{k \geq 0} \mathbb{N}^k \to N$$

$\tau(\emptyset) = 0$ $\tau(a_1, a_2, ..., a_k) = 2^{a_1} + 2^{a_1+a_2+1} + 2^{a_1+a_2+a_3+2} + ... + 2^{a_1+a_2+...+a_k+k-1}$ For example, $\tau(0,0) = 1 + 2 = 3 = 11_2$ $\tau(2,1,0) = 4 + 16 + 32 = 52 = 110100_2$ An n-tuple is mapped to a number whose binary representation uses exactly n 1s. We now show that τ is a bijection:
2.
3. Let A, B be effectively countable, but infinite sets.

(i) If A and B are disjoint, then

$$A \cup B$$

is effectively countable. **Proof:** Given that

$$A \cap B = \emptyset$$

. A and B are disjoint $\to \exists$ bijections between $\mathbb{N}$ and A, and $\mathbb{N}$ and B. Now

$$A \cup B = \{x : x \in A \vee x \in B\}$$

Let a_i be the ith element of A, and b_i be the ith element of B. We enumerate elements of A as $a_1, a_2, ...$ and elements of B as $b_1, b_2, ...$ Enumerate the elements of

$$A \cup B$$

as $\{a_1, a_2, ..., b_1, b_2, ...\}$. We can always do this because both A and B are effectively countable.

(ii). $A \cap B$ is countable

Proof: Always $A \cap B \subseteq A$. But A is countable. So is $A \cap B$.

(iii) If A and B are not necessarily disjoint, prove that A ∪ B is effectively countable.

Proof: We can assume A and B are disjoint and proceed as above. Because if not, $A \cup B = (A \setminus B) \cup (B \setminus A) \cup (A \cap B)$. These are 3 disjoint sets, whose union we have shown to be countable.

4. Show that if a class A of objects is constructed recursively using a finite set of basic objects and a finite collection of computable closure operators (see §2.5), A is effectively countable.

Partially Recursive functions

- The code of a TM is called its index.
- Any particular code of a TM is called an enumeration of the TM,or equivalently, its partial recursive function
- Notation: φ_e is the e^{th} function in the enumeration of a machine with an index e.

Padding Lemma:
Given any index e of a Turing Machine M, there ∃ a larger index e' which codes a machine that computes the same function as M This can also be stated as

Proof: Add redundant states in between. Can we use a Rice's Theorem type argument?

Universal Turing Machines

We are now in a position to construct a TM that can emulate every other TM, using the enumeration of TMs and the pairing function.

$$U(\langle e, x \rangle) = \varphi_e(x)$$

On countability and effective countability

- Do countability and effective countability mean the same thing? No!
- Effective countability is the same thing as recursively enumerable. i.e., a set is effectively countable if there exists an enumeration procedure for it.
- We know that recursively enumerable → countable. However, the converse need not hold.
- To see this, consider the following example. We know that ℕ is both countable and effectively countable.
- Consider any subset of ℕ. That subset is countable.
- But it need not be effectively countable. For this, consider the indices of total computable functions.

Theorem:
There is no computable indexing of total computable functions.

The Church Turing Thesis

Theorem:
The class of Turing Computable and partial recursive functions are exactly the same.
Proof:
⇒ Given a Turing machine, simply emulate the behavior of the function on every step of the computation.

Other Definitions of Computability

So far, we have encountered TMs and UTMs. There are several other "redefinitions" of turing machines that also retain the power of TMs. I will prove one of them, and restate several (refer to Sipser TOC for more)

Example:
Prove that a two-symbol Turing machine can simulate an n-symbol Turing machine, for any n.

Soln

Develop a fixed binary length encoding for all symbols in the alphabet. Then proceed as usual.

More examples:
1. TM with semi-infinite tape
2. TM with one(or more) work tapes
3. TM with a grid of symbol squares instead of a tape, and a read/write head that can move up or down as well as left or right.
4. TM where read/write head can move more than one square in either direction
5. TM with multiple read/write heads with single tape/multiple tapes

etc.

Non determinism

An NDTM works very similar to that of a DTM, except the transition function. In this case, the TM has the option to go to several states from one particular state. This TM then has the power to make that choice *non deterministically*. It can be showed that both NDTMs and DTMs have the same computational power.

Lambda Calculus

Lambda Calculus forms the very basis of functional programming.

- It generally consists of a single transformation rule - a variable substitution and a single function definition scheme. A typical expression would look like

$$(\lambda x \mid E)A$$

- This means "replace every instance of x in E by A"
- These expressions are then built recursively.
- An expression can be an identifier (variables whose values don't change over time), function, several expressions, etc.
- Function evaluation happens from left to right. More precisely,

$$(\lambda xy|E)AB = (((\lambda x \mid)\lambda y \mid E)A)B$$

- Variables can be free or bound:

Working with Computable Functions

Halting Problem

Consider the halting function:

$$f(e) = \begin{cases} 1 \text{ if } \varphi_e(e) \downarrow \\ 0 \text{ if } \varphi_e(e) \uparrow \end{cases}$$

Define another function g:

$$g(e) = \begin{cases} \varphi_e(e) + 1 \text{ if } f(e) = 1 \\ 1 \text{ if } f(e) = 0 \end{cases}$$

We can say the following:

- If f is computable, so is g
- g is not computable, so f is not computable. The halting problem is not computable.

The following set is the **halting set:** $K = \left\{e : \varphi_{e(e)} \downarrow\right\}$

The contradictions

Suppose we have a collection of functions that can be indexed: $\psi_n, n \in \mathbb{N}$. Then define $g(n) = \psi_{n(n)} + 1$. Then, the following three can't hold simultaneously:

1. $\{\psi_n\}_{n\in\mathbb{N}}$ is an indexing
2. $g = \psi_i$ for some i
3. g is total

Parametrization

Parametrization refers to the ability to push input parameters into the index of a function. We now state the Parametrization Theorem:

Theorem:

There exists a total computable function s_1^1 such that $\forall i, x, y, \varphi_{i(x,y)} = \varphi_{s_1^1(i,x)}(y)$

We try to understand this more informally: This theorem basically says that, for a given programming language, and given $m, n \in \mathbb{Z}^+$, there exists an algorithm that accepts the source code of a program with $m + n$ free variables as input, along with m *values*. It then substitutes the values for the first m free variables, leaving the remaining n variables free.

A bit more formally: Consider a function $f(x, y)$ that is computable. Then $\exists$ a total, computable function φ such that $\varphi_{s(x)}(y) = f(x, y)$

smn Theorem (strong) Fix m and n. Let $\overline{x}$ and $\overline{y}$ be n and m tuples respectively. Then, there $\exists$ a primitive recursive *bijective* function s_n^m such that $\forall i, \overline{x}, \overline{y}$,

Parametrization and UTMs help us convert operations on sets and functions to operations on indices.

Fixed Point Theorem

Let f be a total computable function. Then $\exists n \in \mathbb{N}$ such that $\varphi_n = \varphi_{f(n)}$. Moreover, n can be computed from an index of f.

Proof: The proof uses the smn theorem.

Cor 4.4.4 Let f(x, y) be a partially computable function. Then there is an index e such that $\varphi_e(y) = f(e, y)$

Proof: A function is partially computable if

Rice's Theorem

Let A be a non trivial index set. Then χ_A is not computable.

In less mathematical terms, any non-trivial semantic property of a language is undecidable

Unsolvability

- By decidable, solvable and computable, we mean the same thing.
- The Halting Problem k is undecidable.
- It is useful to restate it as: "Is there an algorithm that decides, for any (index) e, does the eth TM halt on input e?"
- Similarly we have other candidates; in each case, we try to encode each problem (uniformly) using a TM and try to find out if it halts.

Relatives of the Halting Problem

We can extend the Halting Set as follows:

$$K_0 = \left\{\langle x, y\rangle : \varphi_{x(y)} \downarrow\right\}$$

This is a more general case. If we have χ_{K_0}, we can build χ_K.

Index Sets

From Rice's Theorem, the following index sets are non-computable.

1. Fin = $\{e : W_e$ is finite$\}$
2. Inf = $\mathbb{N}$ − Fin
3. Tot = $\{e : W_e = \mathbb{N}\}$= $\{e : \varphi_e$ is total$\}$
4. Rec = $\{e : \chi_{W_e}$ is computable$\}$

In short, determining whether a given set has any non trivial property of any partially computable function is uncomputable.

Production Systems:

These work very similar to Grammars.

Semi-Thue Productions: Let $g, \hat{g}$ be finite non-empty words. A production of the form $PgQ \to P\hat{g}Q$ is said to be Semi-thue.

- A semi-thue system is a (possibly infinite) collection of such productions.
- A single non-empty word a is called the *axiom* of the production.
- If a word w can be derived from a by finitely many applications of semi-thue productions, w is said to be a *theorem* of the system.

We have the following:

- Construct a Semi-Thue system with infinitely many productions
- The problem of determining whether a word is a theorem in a Semi-Thue system is undecidable.
- Any TM M can be represented using a Semi-Thue System

Exercises

4.1.2: Define T (n) as the maximum value of s such that some Turing machine with states contained in $\{q_0, ..., q_n\}$ halts after exactly s steps of computation.

(i) Show that T is not computable.

(ii) Show that there is no computable function B such that $B(n) \geq T(n)$ for all n

Proof: Suppose there is a TM M that computes T(n). Then, $\forall n$ M can determine the maximum number of steps that a TM with n states can take before halting. Construct M' as follows: On input n,

- Simulate all the TMs with states in $\{q_0, ..., q_n\}$ for s steps, where s is the output of M
- If any of these halt within m steps, then M' enters an infinite loop. Otherwise, it halts.

Now run M' on its own description. If M does compute T(n), M(m) gives the maximum number of steps a TM with m states can take before halting. But M' behaves differently: If M' halts, it enters an infinite loop and vice versa. Hence M does not compute T(n).

(ii). We already know that T(n) is not computable. If B(n) is indeed computable, then we can indirectly compute T(n) via B(n) leading to a contradiction.

4.3.2: Prove there is a computable function f such that $\varphi_{f(x)}(y) = 2\varphi_x(y) \forall y$

We invoke smn theorem. $\exists g(x, y)$ such that $\varphi_{g(x,y)}(z) = 2\varphi_x(y) \forall x, y, z$.

Computing and Enumerating sets

This section focuses on extending computability to sets, from functions.

Dovetailing Arguments

Given an arbitrary partial function $f : \mathbb{N} \to \mathbb{N}$, we don't know at what point we might not get an input. **Dovetailing** is a technique that aims to resolve that, by *interweaving* computations.

Some motivation:

Suppose you have an (potentially infinite) tree- which is saying its height is very long, and you perform a DFS on it. This might turn out to be very tedious because you might be searching an infinite path. However, consider performing BFS: you visit each node in a branching manner and never hit the above issue.

In Turing Machines

In a TM, first run step $f(0)$. If it halts, great. If not, run $f(0)$ and $f(1)$ together. At the nth step, run the computations of $f(0)$ through $f(n-1)$ each for n steps (don't consider the ones that have halted already). Each computation halts in finitely many steps. So we are good.

Computing and Enumerating

Definition: A function is computable if its characteristic function is computable.

A computable characteristic function is simply a membership procedure.

Facts

- Complement of a computable set is countable
- Any finite set is countable

Recursively enumerable

A set is said to be **recursively enumerable** if there exists a computational procedure to enumerate its elements. This gives us a notion of *computable approximability.*

Given a set A, TFAE:

1. A is recursively enumerable.
2. A is the domain of some partially computable function.
3. A is the range of some partially computable function.
4. Either $A = \emptyset$ or A is the range of a total computable function.
5. $\exists$ a total computable function $f(x, s)$ such that $\forall x, f(x, 0) = 0$, $\exists$ no more than one s such that $f(x, s+1) \neq f(x, s)$ and $\lim_s f(x, s) = \chi_A(x)$.
6. There is a sequence of finite sets $A_s, s \in \mathbb{N}$ such that $\forall s, A_s \subseteq A_{s+1}$ and $A = \cup_s A_s$

Note: Every computable set is computably enumerable. The converse is not true. For example, the halting problem is r.e but not computable.

Proposition: An infinite set is computable if and only if it can be computably enumerated in an increasing order (it is the range of a *monotone* total computable function)

Proof:

$\to$ Let A be an infinite computable set. Then we need to show that it can be computably enumerated in an increasing order. As A is computable, it has a computable characteristic function f(x). We want to show that A can be enumerated in increasing order. $\forall n$, check if $f(n) = 1$. If

so, output n. Moreover f(x) is computable and monotonically increasing. It will enumerate in an increasing order, and halt at some point.

← Suppose we have an infinite ser A that can be recursively enumerated in increasing order by a function say g(x). To show A is computable, we nee to compute a characteristic function f(x) for A. For any input x, check whether x appears in the enumeration of g(x). If so, set f(x) to 1, otherwise f(x)=0.

Now, g is computable. We can effectivey determine if x appears in the enumeration by running g(x) for every $x \in \mathbb{N}$. We do this until we find x, or we find a number larger than x, so in either case we terminate because it is an increasing enumeration.

Exercises:

5.2.9: If A is r.e., A is computable iff $\overline{A}$ is r.e.

Proof: We restate it for clarity. A is decidable iff both A and $\overline{A}$ are recursively enumerable. → Let A be decidable. Then A is r.e. Also, $\overline{A}$ is also decidable, and hence $\overline{A}$ is also r.e.

→ Given that both A and $\overline{A}$ are r.e. Let M_1 and M_2 be the recognizers for them respectively. We construct a decider for A: On input w:

- Run both M_1 and M_2 on w in parallel
- If M_1 accepts, accept. If M_2 accepts, reject.

This procedure continues until one of them accepts, which is bound to happen.

Enumeration and Incompleteness

A **language** L is a collection of symbols that represent constants, relations and functions. Formulas in L can use variables and quantifiers.

Languages consist of sentences. We can be interested in the syntactic or semantic aspects of these sentences. An L structure is a set of elements in L along with (semantic) interpretations of the constants and function symbols in L.

Soundness Theorem

Completeness plus Soundness: A formula is valid iff provable

Theorem: The set of valid formulas is computably enumerable, but not computable.

Let T be a set of sentences in L. We define a **model** M to be an L structure if all the sentences of T are true when interpreted in M. An L sentence φ such that neither φ nor $\neg\varphi$ is provable is said to be *independent* of T. In other words, φ is said to be independent of T if $\exists M, N \vDash T$, $M \vDash \varphi, N \nvDash \varphi$

Robinson Arithmetic

The language (0, S, +, ·, =, <) constitutes this arithmetic: 0 is the constant, S is the unary function symol, + and · are binary function symbols, and the others are binary relations.

A few points

- The Robinson Arithmetic (RA) has independent sentences.
- The axioms of RA are computable. Hence the set of provable sentences in RA is recursively enumerable.
- Enumerate the list of refutable sentences in RA (call this R) by listing $\neg\varphi$ whenever $\varphi \in P$.
- $P \cap R = \emptyset$.

Enumerating Noncomputable sets

- A set A is non-computable iff its characteristic function is nonequal to all (possible) totally computable functions.
- Again, A can either be re or co-re.
- We bother with the first case, i.e., when A is recursively enuemrable.

Definition: Let A be r.e. A is *simple* if $\overline{A}$ is infinite, but contains no infinite c.e. subsets. Then $\overline{A}$ is said to be immune.

A few points:

- A set is simple if it is RE and its complement is simple.
- All simple sets aren't necessarily infinite, but they turn out to be.
- Suppose a simple set A is finite. Then it is computable. Then $\overline{A}$ is cofinite, and also computable.
- But $\overline{A}$ is now an infinite ce set, and cannot be immune, a contradiction.

Problem: Prove that if A is simple, it is not computable

Proof: Let A be simple, and A be computable on the contrary. Then $\overline{A}$ would be computable too. From the discussion above, it would follow that A cannot be computable.

Similarly, we can prove the following:

- A coinfinite re set is simple iff it is not contained in any coinfinite *computable* set.
- Suppose A and B are simple. Then
  - $A \cap B$ is also simple
  - $A \cup B$ is either simple or cofinite

Turing Reductions and Post's Problem

Reducibility of Sets

Definition: An *oracle Turing Machine* with an oracle A is a TM that can ask finitely many questions about membership in A.

A few points:
- Finitely many questions corresponds to it being computable
- Number of questions asked is both input and output sensitive
- Uniformity: Every index codes a well defined function irrespective of the oracle
- Denote an oracle TM by M^A and an oracle function by φ^A
- Let σ be an oracle. Any oracle query with $\geq |\sigma|$ elements diverges.

Exercise: Let σ range over all finite binary strings. Let e, s, x, y range over $\mathbb{N}$. Prove that the set $\{< \sigma, e, x, s, y >: \varphi^{\sigma}_{e,s}(x) \downarrow = y\}$ is computable.

Proof: We need to construct a TM that can decide whether a given input belongs to this set (S) on not.

Let S be a set. For any input $< \sigma, e, x, s, y >$, decide whether $\varphi_e(\sigma, s)(x) \downarrow = y$.

Construct a TM M that simulates the computation of $\varphi_e(\sigma, s)(x)$ on input x, with σ as an oracle (use dovetailing here). Accept if computation halts and produces y. Else reject.

Definition: If $\varphi^A_{e,s}(x) \downarrow$, then define the *use* of the computation as

$$u(A; e, x, s) = 1 + \max\{n : \text{n in A}\}$$

asked during computation.

- The use of divergent computations is defined to be 0 for stage bounded version.
- For the unbounded version, it is defined.

Notation: $A \mid_n = A \cap \{0, 1, ..., n-1\}$

Use Principle
1. $\varphi^A_e(x) = y \rightarrow (\exists s)(\exists n)\big(\varphi^{A \mid_n}_{e,s}(x) = y\big)$
2. We can extrapolate the principle appropriately to finite and infinite binary strings as follows:

Let σ, τ be finite binary strings, and A be an infinite binary sequence. The following hold:

$$\varphi^{\sigma}_{e,s}(x) = y \rightarrow (\forall t \geq s)(\forall \tau \supseteq \sigma)\big(\varphi^{\tau}_{e,t}(x) = y\big)$$

$$\varphi^{\sigma}_{e,s}(x) = y \rightarrow (\forall t \geq s)(\forall A \supset \sigma)\big(\varphi^{A}_{e,t}(x) = y\big)$$

Turing Reduction

A set A is *Turing reducible* to a set B, denoted by $A \leq B$, if for some e, $\varphi^B_e = \chi_A$. Then A is said to be computable with oracle B.

Turing Equivalence:
Sets A and B are said to be Turing Equivalent, denoted by $A \equiv {}_T B$ if $A \leq {}_T B, B \leq {}_T A$.

Two sets can also be Turing incomparable: $A \perp {}_T B$.

We can show the following:
1. $\leq {}_T$ is a reflexive and transitive relation on $P(\mathbb{N})$.

2. There is a function k such that $\forall i, e, A, B, C$, if $\chi_C = \varphi_e^B$ and $\chi_B = \varphi_e^A$ then $\chi_C = \varphi_{k(e,i)}^A$
3. Turing equivalence is an equivalence relation.We will learn about the equivalence classes induced by this relation, the *turing degree* in the next section.
4. A computable set is Turing Reducible to any arbitrary set.
5. If A is computable, and $B \leq_T A$, then B is also computable.

Turing Complete Sets

Sets that are c.e and can compute all other c.e sets are said to be *turing complete*.

Theorems: We shall state the following theorems without proof here.

1. The halting set, K is Turing Complete.

Define the weak Jump $H = \{e : W_e \neq \emptyset\}$

1. The Weak Jump is c.e, and is Turing Complete.

Finite Injury Priority Arguments

The main goal of this section is to construct a simple set. A c.e set A is said to be simple if $\overline{A}$ is infinite but does not contain any infinite c.e subset.

To construct an arbitrary set A, we can have an infinite collection of requirements $\{R_e\}_{e \in I}$, I is an index set in bijection with $\mathbb{N}$. It is possible that these requirements can interact with each other and *injure* each other to the extent that none of them ever get satsified. To solve this, we introduce *priority*. We order the requirements in such a way that a requirement R_i can injure R_j if and only if $j > 1$. Moreover, R_i can injure R_j only finitely many times. We call these **finite injury priority arguments**.

Theorem: There exists a simple set

Post's Problem

Does there exist a c.e set A such that A is non-computable and incomplete?

We give an answer in the affirmative.

To show this, we will construct a set A that is simple, and a set B that is c.e but cannot be computed by A. And hence, A is incomplete.

The requirements are follows:

$$R_e : (|W_e| = \infty) \Rightarrow (A \cap W_e \cap \{x \in \mathbb{N} : x > 2e\} \neq \emptyset)$$

$$Q_e : \varphi_e^A \neq \chi_B$$

Hierarchies of Sets

In the previous section, we introduced the notion of Turing reducibility and Turing equivalence. Turing Equivalence can be seen as an equivalence relation on $P(\mathbb{N})$ using an s-m-n theorem type argument.

Given a set A and an equivalence relation R on A, the quotient of A by R A/R is the set whose elements are the equivalence classes of A under R.

Turing Degrees

- The quotient of $P(\mathbb{N})$ by Turing Equivalence is called **Turing Degree** or **Degree of Unsolvability**.
- Let $A \subseteq \mathbb{N}$ be any set. The **degree** of A is the equivalence class of A under Turing Equivalence, and is denoted by deg(A).

Results

1. The least turing degree is deg($\emptyset$). It is the degree of all computable sets.

Proof:

We want to show $\forall A \subset \mathbb{N}, \deg(\emptyset) \leq \deg(A)$. Let X be any computable set, and $A \subset \mathbb{N}$. Then $X \leq A$. Then $X \in \deg(\emptyset), \deg(\emptyset) \leq \deg(A) \forall A \subset \mathbb{N}$

2. Every pair of degrees deg(A) and deg(B) has a least upper bound. LUB = $\deg(A \oplus B)$

Proof:

3. For all sets, $\deg(A) = \deg\left(\overline{A}\right)$

Proof: We have seen that for any arbitrary set A, $A \equiv_T \overline{A}$. They belong to the same equivalence class.

4. Every infinite c.e set contains subsets of every Turing Degree.

Proof: Let D be the set of all Turing Degrees. First use Cantor's Diagonalization argument to show there can be uncountably many Turing Degree. So D is uncountable. Set up a bijection between D and $\mathbb{N}$. Then $D = \{d_n\}_{n \in \mathbb{N}}$. And let $A = \{a_n\} (n \in \mathbb{N})$.

Constructs subsets B_n of A as follows:
1. $\forall d_n$ let M_n be a TM such that A_{M_n} has turing degree d_n
2. Then $A_{M_n} = \left\{a_{n,k}\right\}_{k \in \mathbb{N}}$
3. Let B_n be the set having the first k elements of A such that $a_{n,k} \in A_{M_n}$.

Thus each B_n has Turing degree d_n. But D is an infinite set.

Defn: A degree is called c.e if it contains a c.e. set.

Recall that these Turing degrees are equivalence classes themselves.

Note: Also, K is Turing Complete. Therefore the maximum c.e degree is deg(K).

Theorems:

1. Every Turing Degree contains countably many sets

Proof: Let A be a set. Any other set $B = \varphi_e^A$, (e is countable) and so there are atleast countably many B such that $B \equiv_T A$. There should be *at most* countably many such B using a symmetric difference argument.

2. We have shown that there are infinitely many *indistinct* Turing degrees. There are $2^{\aleph_0}$ many distinct Turing degrees, where $\aleph_0 = |\mathbb{N}|$.

Proof: There are $2^{\aleph_0}$ sets, and every turing degree contains countably many sets.

3. There is no maximal Turing degree.

Relativization

Fix some set A and work with A as your oracle "i.e work relative to A".

Relativized s-m-n theorem

$\forall m, n \geq 1$ there exists a one to one computable function s_n^m of m+1 variables such that $\forall A \subseteq \mathbb{N}$, $\forall i, \overline{x}, \overline{y}$, where the last two are n-tuples

$$\varphi_{s_n^m}^A(i, \overline{x}) = \varphi_i^A(\overline{x}, \overline{y})$$

- This doesn't add much help, because s_n^m is already computable even without an oracle.

Defn: A set B is *computably enumerable in the set A* if, $\exists e, B = W_e^A$.

- Think of A as a parameter, and not in terms of inclusion.
- Essentially this means using A as an oracle, we can computably enumerate A

Notes:

1. This is not transitive. Let A be c.e in B, and B be c.e wrt C. We need to show A is not c.e wrt C. For this, Let C be any recursive set, and A is r.e wrt B but not r.e wrt all domains. More specifically, this should work if B is r.e but not co-r.e.
2.

Theorem:

The set $A' = \left\{e : \varphi_e^A(e) \downarrow\right\}$ is c.e in A but not computable in A.

- This is the halting set relativized to A
- Also called the Turing Jump of A

Turing Jump

Jump is an operator that assigns to a decision problem X a "successively harder decision problem X" that is minimal in the following sense: it cannot be decided by an oracle for X.

- Thus the jump of X can be thought of as an oracle for halting problem K for oracle machines with an oracle for X

Arithmetical Hierarchy

This is a way of categorizing sets according to how complicated the logical predicate representing them has to be:

Definitions

1. Let B a computable set. Then B is Σ_0 and Π_0
2. If $x \in B \Leftrightarrow (\exists y)(R(x, y))$, where R(x, y) is a computable relation, then $B \in \Sigma_1$
3. If $x \in B \Leftrightarrow (\forall y)(R(x, y))$, where R(x, y) is a computable relation, then $B \in \Pi_1$
4. If φ is logically equivalent to a formula $(\exists m_1)(\exists m_2)...(\exists m_k)\psi$, where psi is Π_n^0, then φ is Σ_{n+1}^0
5. If φ is logically equivalent to a formula $(\forall m_1)(\forall m_2)...(\forall m_k)\psi$, where psi is Σ_n^0, then φ is Π_{n+1}^0

6. If B is both Σ_n and Π_n, it is Δ_n
7. B is *arithmetical* if for some n $B \in \Sigma_n \cup \Pi_n$

We can relativize these appropriately.

Exercises:
1. **If** $B \in \Pi_n$ **or** $B \in \Sigma_n$**, then** $B \in \Pi_n$ **and** $\Sigma_n \forall m > n$
2. $B \in \Sigma_n \Leftrightarrow \overline{B} \in \Pi_n$
3. **B is c.e iff** $B \in \Sigma_1$

Proposition: $\forall n \geq 1 \exists$ **a** Σ_n **set that is not** Π_n
- The Σ_1 sets are effectively countable: they correspond exactly to c.e sets
- Thus they correspond to Π_1 sets as well
- We can have enumerations for both of those
- Using them, inductively construct enumerations for Π_n and $\Sigma_n \forall n \geq 1$
- In particular, we have a Σ_n set, call it S. S corresponds to the UTM. By this, it also corresoinds to Σ_n itself
- $\langle e, x \rangle \in S$ iff the e^{th} Σ_n set contains x
- Define $P \coloneqq \{x : \langle x, x \rangle \in S\}$
- **Claim:** P is Σ_n but P is not Π_n. $P \in \Sigma_n$ by construction. If $P \in \Pi_n$, then $\overline{P} \in \Sigma_n$. Then $\overline{P}$ is the eth Σ_n set for some n. This leads to a contradiction
- A similar argument shows that $\overline{P}$ is Π_n but not Σ_n

Definitions:
1. A set A is Σ_n complete if $A \in \Sigma_n$ and $\forall B \in \Sigma_n \exists$ a total computable 1-1 function f such that $x \in B \Leftrightarrow f(x) \in A$. Then B is 1-reducible to A.
2. A set A is Π_n complete if $A \in \Pi_n$ and $\forall B \in \Sigma_n \exists$ a total computable 1-1 function f such that $x \in B \Leftrightarrow f(x) \in A$. Then B is 1-reducible to A.

Observations:
- X is Σ_n complete $\Leftrightarrow \overline{X}$ is Π_n complete
- $\forall n > 0, \emptyset^{(n)}$ is Σ_n complete and $\overline{\emptyset^{(n)}}$ is Π_n complete

Post's Theorem

TFAE.
1. $B \in \Sigma_{n+1}$
2. B is c.e in some Π_n set.
3. B is c.e in some Σ_n set.

We also have:
- $B \in \Sigma_{n+1} \Leftrightarrow$ B is c.e in $\emptyset^{(n)}$
- $B \in \Delta_{n+1} \Leftrightarrow$ B is Turing-reducible to $\emptyset^{(n)}$

Index Sets and Arithmetical Completeness

We can define the following sets.
1. Fin $= \{e : |W_e| < \infty\}$
2. Inf $= \{e : |W_e| = \infty\}$
3. Tot $= \{e : \varphi_e\}$ is total.
4. Con $= \{e : \varphi_e\}$ is total and constant
5. rec $= \{e : W_e\}$ is computable

We summarise the findings of this section below:

- Fin $\in \Sigma_2$. Moreover, it is Σ_2 complete
- Inf $\in \Pi_2$. Moreover, it is Π_2 complete
- Rec $\in \Sigma_3$
- Tot $\in \Pi_2$. Moreover, it is Π_2 complete.
- Con in Π_2. Moreover, it is Π_2 complete.