# Chapter 1

# Genesis

**Chapter Outline:** Structural versus Computational Mathematics. Historical perspectives. Brief overview of complexity theory. Hilbert's Tenth Problem. Turing Machines. Undecidability. Cantor's method of diagonalization. Undecidability of the halting problem. Time and space bounded Turing machines. Hierarchy Theorems. Complexity Classes (L, NL, P, NP, PSPACE, E, EXP).

## 1.1 Structural versus Computational

There have always been two major strands of mathematical thought since the time of antiquity: Structural Theory and Computational Methods. For example, Euclid's *Elements* is a synthesis of much that was known in geometry up till that time, it is also largely structural in that it emphasizes (and establishes) theorems and deductive proofs; by contrast, the writings of Diophantus, as in *Arithmetica*, were primarily algorithmic, where some of the methods probably go back to Babylonian mathematics 2000 years before that. Of course these strands of mathematical thought are not in opposition to, but rather, complement each other. Even in Euclid's *Elements*, one finds algorithmic gems such as The Euclidean Algorithm which finds the greatest common divisor of two positive integers. It is a shining example of an early triumph in algorithm design, whose correctness and efficiency demands proof in a purely structural sense. Outside of the Greek tradition, other ancient civilizations also had various emphasis on either the *Structural Theory* which prizes the framing and proof of general theorems by deductive reasoning, or *Computation* which seeks efficient computational method to solve problems. For example, Chinese mathematicians of antiquity seem to concern themselves primarily with computation.

To a great majority of the classical masters throughout history, starting with Archimedes, Newton, Leibniz, Euler, Lagrange, Gauss, ..., computation is an inseparable aspect of mathematics as a whole. It is a rather recent phenomenon that the computation is somehow delegated as secondary to the Big Math edifice, perhaps helped along by the influential schools

such as Bourbaki. But in reality, much of the motivation for the big structural discoveries were computational originally. For example, Calculus was invented so as to facilitate computation of orbits of heavenly bodies as well as measuring surface areas and volumes; Galois Theory and finite group theory was a discovery to investigate the solvability of equations by radicals; the Prime Number Theorem was first conjectured by Gauss after much computational experiments. It is also true that much of the advances made in structural mathematics had also greatly influenced the advances in computational mathematics.

While it can be said that the subject of algorithms is as old as mathematics itself, the serious mathematical study of algorithms as such, rather than the use of them, is a relatively new development.

Perhaps one could trace this beginning to *Set Theory*, that most structural of all subjects. In his study of Fourier series (surely one of the most computational subjects in origin), Cantor gave birth to a set of ideas that we now call (naive) set theory. Cantor's ideas are revolutionary in many aspects. In its basic framework it is highly non-constructive. For example, Cantor gave a conceptually crisp and simple proof of the existence of transcendental numbers, whereby inventing his famous diagonalization method. This proof is remarkable in many ways:

Firstly, it is much simpler than the monumental achievement of Hermite and Lindemann on the transcendence of $e$ and $\pi$ respectively. Perhaps one can still make the case that the "real" transcendental number theory is more along the lines of Hermite, Lindemann and Liouville, and not the mere existence proof by the magic of diagonalization. But even the most dedicated practitioners of "hard analysis" today will not dismiss the elegance and efficiency of Cantor's method. On the other hand, today many interesting computational problems, such as basis reductions for lattices, simultaneous Diophantine approximations, and volume estimations of convex bodies, form very active research areas which can be traced directly to the work such as Dirichlet, Liouville, Hermite and Minkowski.

Secondly, as Kronecker was quick to point out, Cantor's method is inherently non-constructive, and in his view, borders on the "philosophical". In particular it did not conform to the strictly finitistic and constructive approach that Kronecker had been advocating. To the end of his day, Kronecker never accepted Cantor's idea. The finitists distrust it on philosophical ground, which is ironic because the finitists are particularly concerned with the soundness of mathematical foundation, which is to be demonstrated in coming years to be closely related to computational undecidability, in which Cantor's diagonalization method is a forerunner.

Thirdly, the diagonalization method was to find its great application in Turing's undecidability proof of the Halting Problem. It subsequently became one of the basic mathematical tools in recurcsion theory, and in the founding of complexity theory with the proof of the time and space hierarchy theorems.

Because of its fundamental importance we will review the diagonalization proof by Cantor.

An algebraic number is a root of a polynomial with integral coefficients. A non-algebraic number is called a transcendental number. A set is countable if it can be put into one-to-one correspondence with the integers. It is clear that the set of all algebraic numbers is countable, since we can count all integral polynomials, and each polynomial of degree $n$ has at most $n$ roots.

**Exercise:** Show that the following is a *pairing function* which gives a one-to-one correspondence between non-negative integers $\mathbf{Z}_+ = \{0, 1, 2, \ldots\}$ with the cartesian product $\mathbf{Z}_+ \times \mathbf{Z}_+$.

$$\langle i, j \rangle = \binom{i+j-1}{2} + j.$$

**Exercise:** Show that the set of rational numbers, and the set of algebraic numbers are countable.

THEOREM 1.1 *The set of real numbers is uncountable; in particular, there are non-algebraic real numbers.*

A curious historical note: In order not to offend Kronecker, who was powerful and somewhat petty at the same time and might block the publication of this work, Cantor had to phrase his main result strictly on the existence of non-algebraic numbers, and not mention anything of his emerging cardinality theory of the infinite.

*Proof.* Consider all binary infinite sequences $B = \{\beta\}$, where

$$\beta = b_1 b_2 \ldots b_n \ldots,$$

and $b_i \in \{0, 1\}$. We know that the real numbers in $[0, 1]$ can be put in 1-1 correspondence with this set $B$, via binary expansion.[1]

**Claim:** $B$ is uncountable.

Suppose otherwise. Let $\beta_1, \beta_2, \ldots, \beta_n, \ldots$, be an enumeration of all $B$, where each $\beta_i = b_{i,1} b_{i,2} \ldots b_{i,n} \ldots$. Write these sequences by the rows, and we obtain an infinite table as follows.

[1] There is a technical problem of some real numbers in $[0, 1]$ having two different infinite binary expansions. But it is easy to see that these real numbers are precisely those with a finite terminating binary expansion, and thus are rational numbers, and clearly countable. Therefore the claim of 1-1 correspondence with $B$ is still valid.

| | 1 | 2 | 3 | 4 | $\cdots$ | $i$ | $\cdots$ |
|---|---|---|---|---|---|---|---|
| $\beta_1$ | $\boxed{b_{1,1}}$ | $b_{1,2}$ | $b_{1,3}$ | $b_{1,4}$ | $\cdots$ | $\cdots$ | $\cdots$ |
| $\beta_2$ | $b_{2,1}$ | $\boxed{b_{2,2}}$ | $b_{2,3}$ | $b_{2,4}$ | $\cdots$ | $\cdots$ | $\cdots$ |
| $\beta_3$ | $b_{3,1}$ | $b_{3,2}$ | $\boxed{b_{3,3}}$ | $b_{3,4}$ | $\cdots$ | $\cdots$ | $\cdots$ |
| $\beta_4$ | $b_{4,1}$ | $b_{4,2}$ | $b_{4,3}$ | $\boxed{b_{4,4}}$ | $\cdots$ | $\cdots$ | $\cdots$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | | |
| $\beta_i$ | $b_{i,1}$ | $b_{i,2}$ | $b_{i,3}$ | $b_{i,4}$ | $\cdots$ | $\boxed{b_{i,i}}$ | $\cdots$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | | | $\ddots$ |

Now we define a particular $\beta = b_1 b_2 \dots b_n \dots \in B$, which by its definition can not be on this list: We go down the diagonal of this infinite table; for the $n$ th entry, if it is 0 we let our new $b_n = 1$, and if it is 1 we let $b_n = 0$. Formally, $\forall n \geq 1$,

$$b_n = \begin{cases} 0 & \text{if } b_{n,n} = 1 \\ 1 & \text{otherwise.} \end{cases}$$

Thus we "constructed" our $\beta \in B$ to be "disagreeable": It disagrees with the $n$ th item on the list in the $n$ th place. Hence this $\beta$ cannot be among those listed. (If it were the $n$ th sequence, what should its $n$ th entry be?) ♣

It is self evident why Cantor's method is called the diagonalization method.

Kronecker's objections not withstanding, Cantor's set theory opened up a mathematical "paradise", from which, Hilbert was said to have remarked that mankind will never be driven out again. Nevertheless, it did bring troubles with it. In particular, at the turn of the 20th century, a number of set theoretic paradoxes were found that pertain to the foundation of mathematics.

Here is the famous Russell's paradox.

"Surely" pure logic dictates that a set $A$ either satisfies $A \in A$ or $A \notin A$. Let's form $\mathcal{A} = \{A \mid A \notin A\}$. In Cantor's naive set theory this seems perfectly legitimate. *If*, Russell says, we can form this set $\mathcal{A}$, then is $\mathcal{A} \in \mathcal{A}$? If $\mathcal{A} \in \mathcal{A}$, then by its definition, $\mathcal{A} \notin \mathcal{A}$. However, if $\mathcal{A} \notin \mathcal{A}$, then again by definition, $\mathcal{A} \in \mathcal{A}$.

## 1.2 Turing Machines and Undecidability

At the turn of the last century, paradoxes such as this one stirred up a lot of uneasiness with the foundation of mathematics, which was to be formulated increasingly based on Cantor's

naive set theory. This created the strong desire to re-examine the foundation of mathematics, and try to formalize all mathematics on an axiomatic basis without contradiction. (In the modern axiomatic set theory, such paradoxes are avoided by being more careful as to what is admissible as a set construction.)

Hilbert was a leading advocate of the formalist school. In his famous address to the International Mathematicians Congress in 1900, he listed 23 problems which he thought to be most likely to excite the imagination of mathematicians worldwide in the new century. A number of Hilbert's problems are concerned with foundational issues, such as the Continuum Hypothesis. The 10th problem on the list asks the following: Find a systematic procedure to determine if a polynomial with integral coefficients in several variables has an integral solution. More broadly then, Hilbert initiated the study of *Decision Problems* where the aim is to find an algorithm to decide for each instance of a problem. The research in the next 40 years showed that the study of computability is intimately related to the foundation of mathematics. Several other Hilbert problems also had a profound impact on the future development of the foundation of mathematics and computability theory, such as the Continuum Hypothesis (#1) and Consistency Problem (#2), but it was the Decision Problem, a.k.a. *Entscheidungsproblem*, that was the primary impetus to Turing's seminal work which established the foundation of computability theory.

It turns out that the notion of a systematic procedure to compute is intimately related to the notion of a formal proof in axiomatic system. The work of Turing, Gödel, Church, Post, and others, established that the original program envisioned by Hilbert cannot be complete. While Hilbert only asked for an algorithm in the 10th problem, the possibility of the non-existence of such an algorithm probably never crossed his mind. This turned out to be the case in the end, as it was finally proved by Matiyasevich in 1970. But, it was Hilbert who raised the question, and focus the attention on the very notion of what constitutes an algorithm, what is computation. In answering Hilbert, computability theory was born.

It is to address Hilbert's Entscheidungsproblem, Turing defined his model of a general purpose computer—the Turing Machine. A Turing Machine (TM) has a finite state control and an infinite input/output tape with a reading/writing head. A deterministic Turing Machine moves in each step in a unique way determined by its current state, and the symbol it is currently scanning. A move consists of a possible change of state and scanned symbol, and moving the head left or right. A non-deterministic Turing Machine (NTM) may have several legal moves, which are still determined by its current state and the scanned symbol. (NTMs are not important for computability theory, but important for complexity theory later.) An input $x$ is accepted by a deterministic TM if the computation starting with initial configuration ends in an accepting state. For a NTM acceptance is defined by the existence of some sequence of legal moves that ends in an accepting state. Again for complexity theory purposes, we also define multitape Turing Machines. For space bounded Turing Machines with sublinear space bound we allow a separate read-only input tape, and a work tape (or several tapes [2]), and the space bound is counted only on the work tape(s). We assume the

[2]For space complexity it turns out a single work tape suffices.

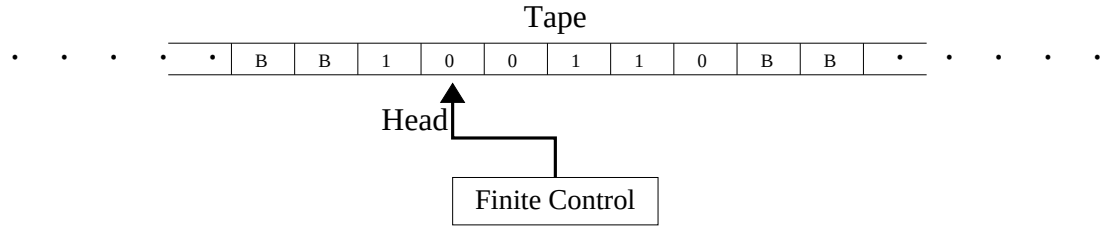



Figure 1.1: Turing Machine

readers are already familiar with these notions; for detailed definitions, please see [3, 4, 2].

TMs are not the only model of computation formulated. In the 1930s a number of different models were formulated; however every one of the general models were shown to be equivalent to TMs in due course. The universality of TMs is captured by The Church-Turing thesis which states that whatever can be computed can be computed by a Turing Machine.

Cantor's diagonalization method was technically the opening salvo by Alan Turing. Adapting Cantor's method, Turing showed that there are problems which can not be computed by any Turing Machine, and thus, by the Church-Turing thesis, uncomputable by any algorithm whatsoever. Such problems are called undecidable problems. In particular, the so-called Halting Problem for Turing Machines is one such problem. Furthermore, a reduciton theory is developed whereby a host of problems can be shown undecidable.

The proof of the undecidability of the Halting Problem goes as follows.

List all the Turing Machines $M_1, M_2, \ldots$ row by row, and index the columns by the inputs of all finite strings, which we will identify with integers $j = 1, 2, \ldots$. Mark the $(i, j)$ entry of this table by $M_i(j)$, the outcome of machine $M_i$ on input $j$. We will only consider the outcome of either "halt" or "not halt". We are not claiming this outcome can be computationally determined in general, and in fact the purpose of this proof is precisely to show that it is impossible to determine this outcome by TMs, and thus by the Church-Turing thesis, undecidable by any algorithm.

Thus we obtain an infinite table, much like that in Cantor's proof.

Now suppose there is a decision procedure in the form of a Turing Machine $M$ that, for any $\langle i, j\rangle$, can decide whether $M_i(j)$ halts or not in a finite number of steps. Let's say $M(\langle i, j\rangle) = 1$ if $M_i(j)$ halts and $M(\langle i, j\rangle) = 0$ otherwise. Then one can design another Turing Machine $M'$ as follows: On any $i$, $M'(i)$ simulates $M(\langle i, i\rangle)$. If $M(\langle i, i\rangle) = 1$, then $M'(i)$ enters a loop and never halts. If $M(\langle i, i\rangle) = 0$, then $M'(i)$ halts.

Now since all Turing Machines have been enumerated, there exists a $k$, such that our $M'$ is $M_k$. But what happens to $M_k(k)$?

| | 1 | 2 | 3 | $\cdots$ | $i$ | $\cdots$ |
|---|---|---|---|---|---|---|
| $M_1$ | $\boxed{M_1(1)}$ | $M_1(2)$ | $M_1(3)$ | $\cdots$ | $\cdots$ | $\cdots$ |
| $M_2$ | $M_2(1)$ | $\boxed{M_2(2)}$ | $M_2(3)$ | $\cdots$ | $\cdots$ | $\cdots$ |
| $M_3$ | $M_3(1)$ | $M_3(2)$ | $\boxed{M_3(3)}$ | $\cdots$ | $\cdots$ | $\cdots$ |
| $\vdots$ | $\vdots$ | $\vdots$ | | $\ddots$ | | |
| $M_i$ | $M_i(1)$ | $M_i(2)$ | $M_i(3)$ | $\cdots$ | $\boxed{M_i(i)}$ | $\cdots$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | | $\ddots$ |

If $M_k(k)$ eventually halts, then by the assumption of $M$, $M(\langle k,k\rangle) = 1$, and thus $M'(k)$, which is $M_k(k)$, never halts. If $M_k(k)$ does not halt, then again by assumption, $M(\langle k,k\rangle) = 0$, and $M'(k)$ halts. Either way we have a contradiction. ♣

Next we will define time and space bounded computation. This is the domain of computational complexity theory. We will see that diagonalization method reappears, to establish that more time or more space provably can compute more.

## 1.3 Time, Space and Non-determinism

The following notions are basic, and can be found in more details in any reference books [3, 4, 2, 1].

A DTM is formally $(Q, \Sigma, q_0, \delta, F)$, where $Q$ is a finite set of states. $\Sigma$ is a finite set of alphabet symbols, excluding a special symbol $\sqcup$. $q_0 \in Q$ is the starting state. $F \subseteq Q$ is a subset of accepting states. $\delta$ is a partial transition function that maps $(Q - F) \times (\Sigma \cup \{\sqcup\})$ to $Q \times \Sigma \times \{L, R\}$. The interpretation is as follows: when TM $M$ is in state $q \in Q - F$, and reading symbol $A \in \Sigma \cup \{\sqcup\}$, if $\delta$ is defined, say $\delta(q, A) = (q', A', \Delta)$, where $\Delta = L$ or $R$, then in one step the partial transition function $\delta$ provides the next state $q' \in Q$, and overwrite the symbol by $A' \in \Sigma$, and moves left ($\Delta = L$) or right ($\Delta = R$). Thus, a TM is defined by a finite set of quintuples of the form $(q, A, q', A', \Delta)$, which can be written as an integer in a standard encoding scheme, e.g., using repeated applications of our pairing function $\langle \cdot, \cdot \rangle$. We can further assume every integer encodes a TM, with the simple convention that an "invalid" encloding corresponds to a TM that has no legal moves. Furthermore, we will almost exclusively use the binary alphabet set $\{0, 1\}$.

We will use multitape TMs, which are similarly defined. The time complexity of deterministic TM $M$ is

$$\text{time}_M(n) = \max\{\#\text{ of steps in } M(x), \text{ for } |x| = n\}.$$

For any function $f(n)$,

$$\text{DTIME}[f] = \{L \mid \text{ for some } M \text{ }, L(M) = L, \text{ and time}_M(n) \leq f(n), \text{ for all large } n.\}.$$

In other words, DTIME[$f$] consists of problems that are computable by some TM with running time at most $f(n)$ asymptotically. By some silly tricks such as enlarging the alphabet set, and the set of finite states, one can show that any constant factor does not matter. Thus DTIME[$f$] = DTIME[$O(f)$]. For technical reasons we will only consider "nice" functions $f(n)$, called fully time-constructible functions. This means that there is some TM $M$, which for every $n$, and any input $x$ of size $n$, $M(x)$ runs in exactly $f(n)$ steps and then halts.[3] Almost any reasonable functions $\geq n$, such as $n$, $n^k$, $n^i(\log n)^j$, $2^{(\log)^k}$, $2^{n^k}$, and $2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}$. They are also closed under addition, multiplication, exponentiation, etc. We will always assume $f(n) \geq n$, the time needed to read the input.

The model of TM is chosen because it is relatively robust. One can show, for instance, that any $k$-tape TM running in time $f(n)$ can be simulated by a 2-tape TM in time $O(f(n)\log(f(n)))$. For our purposes we will only need the more trivial simulation in time $O(f(n)^2)$, even by 1-tape TM. This simulation can be seen easily as follows: Devide the single tape into $2k$ tracks, and use a large alphabet set with more than $2^{2k}$ symbols, say. Keep on the single tape the contents of all $k$ tapes, together with a mark for each head position. Then one step of the computation of the $k$-tape TM is simulated by the 1-tape TM with 2 sweeps. Note that each sweep of the tape area which has been used takes at most $O(f(n))$ steps. For time complexity, the default model is multitape TMs as in the following definitions.

Let poly denote the class of polynomials, or simply $n^i + i$, $i = 1, 2, \ldots$. Then the union

$$\text{P} = \bigcup_{f \in \text{poly}} \text{DTIME}[f]$$

is the class of deterministic polynomial time. Clearly this definition is invariant when restricted to 1-tape TMs. One can similarly define exponential time classes

$$\text{E} = \bigcup_{k>0} \text{DTIME}[2^{kn}]$$

$$\text{EXP} = \bigcup_{k>0} \text{DTIME}[2^{n^k}]$$

One can define space complexity similarly. In one aspect it is even simpler, since we can use $k$ tracks to mimic $k$ tapes and there is no additional space overhead in the simulation. So we will have just one work tape. However in another respect, there is a slight complicaiton. This happens when we wish to study sublinear space complexity, which contain important problems. In order to account for sublinear space, we use a separate read-only input tape, in addition to a read-write work tape.

[3]It is a fact in complexity theory that there exist functions which are not fully time-constructible, but we will not be concerned with that.

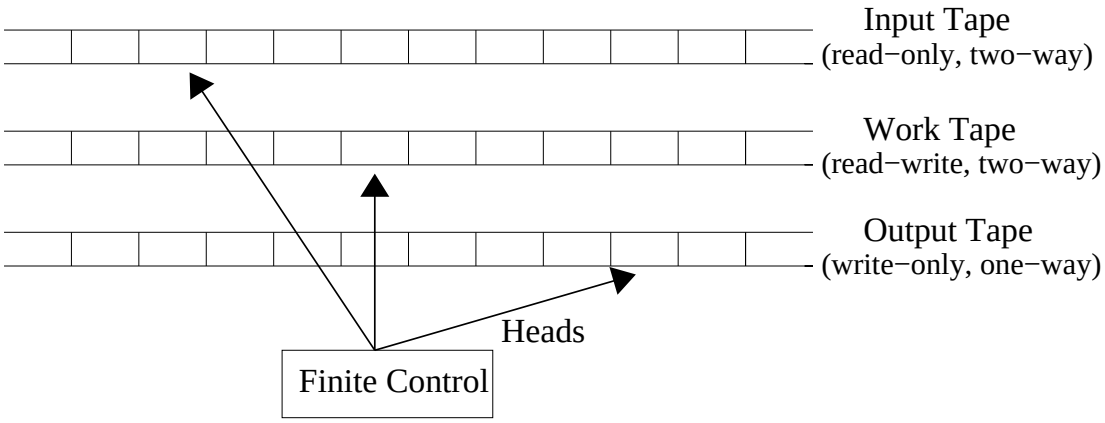



Figure 1.2: Multi-Tape Offline Turing Machine

On the read-only input tape, the input of length $n$ is written, but these $n$ cells do not count toward space complexity. We only count the number of tape cells used on the read-write work tape. Thus for space complexity, the standard model is what is known as an off-line TM, which has one read-only input tape, and one read-write work tape. Then one can define, in an obvious way,

$$\text{space}_M(n) = \max\{\#\text{ of cells on work tape used in } M(x), \text{ for } |x| = n\}.$$

Again we restrict to "nice" functions, called fully space-constructible functions.

**Exercise:** Define fully space-constructible functions.

For any such function $f(n)$, define

$$\text{DSPACE}[f] = \{L \mid \text{ for some } M \text{ , } L(M) = L, \text{and space}_M(n) \leq f(n), \text{ for all large } n.\}.$$

We define

$$\text{PSPACE} = \bigcup_{f \in \text{poly}} \text{DSPACE}[f],$$

the class of polynomial space. (There is a reason why we omit the word "deterministic", as we shall explain later.) We also have deterministic logarithmic space L = DSPACE[log]. Note again that constant factors do not matter, thus DSPACE[log] = $\bigcup_{c>0}$ DSPACE[$c \log$].

The notion called *non-determinism* is of central importance in complexity theory. This notion is related to the process of guess and verification. We will consider two examples. Boolean Satisfiability and Graph Accessibility.

Suppose we are given a Boolean formula $\varphi$ in propositional logic, i.e., it is a well-formed-formula made up from logical AND ($\wedge$), OR ($\vee$) and NOT ($\neg$), and Boolean variables $x_1, x_2, \ldots, x_n$. The problem is whether there is a truth assignment for the variables $x_1, x_2, \ldots, x_n$ such that $\varphi$ evaluates to True. This problem is called the (Boolean) Satisfiability Problem. The set of all satisfiable formulae is denoted as SAT.

No polynomial time algorithm is know for SAT. But whenever a formula $\varphi(x_1, x_2, \ldots, x_n)$ is satisfiable, i.e., $\varphi \in$ SAT, there is an assignment, if one is given that assignment, one can easily verify that $\varphi$ is satisfiable, in particular it evaluates to True under that assignment. We consider this process as consisting of two stages, first *guess* a witness, in this case a satisfying assignment, second *verification*, to check indeed it is a satisfying assignment. Note that $\varphi$ is satisfiable if and only if such a guess exists which will pass the verification stage. Of course, one can consider this guess-and-verify to go hand in hand. In this Satisfiability Problem for example, we can imagine one guesses one bit at a time for each variable, and evaluates the formula as each variable is assigned. We will formally define non-deterministic Turing Machines (NTM) as TMs where for each state and tape symbol being read $(q, A)$, the transition function $\delta$ provides a finite set of allowable next-step transitions consisting of a state $q'$, a tape symbol $A'$ and a left or a right movement, $\Delta = L$ or $R$, of the tape head. A NTM is defined by this set of quintuples of the form $(q, A, q', A', \Delta)$. For DTM, every $(q, A)$ has at most one quintuple of the form $(q, A, q', A', \Delta)$ signifying at most one legal move; with NTM, each $(q, A)$ can have a finite set of quintuples the first two entries being $q$ and $A$ (this finite set is part of the definition of $\delta$).

These two views of non-deterministic computations, namely, 2 stage guess and verify or multiple valid moves per step, are clearly equivalent for SAT. For problems such as SAT, or in general, for non-deterministic time classes (which is at least linear $cn$), the first approach, guess-and-verify, is more intuitive. However, for formal treatment of non-deterministic computations, it is easier with the second approach. This is especially so when one is dealing with possibly non-terminating computations. [4] Another place the second approach is more suitable is when we consider space bounded computation with sub-linear space bound as in the following example (where there is not enough space to guess and write down all the non-deterministic moves at the beginning.)

Consider the following Graph Accessibility Problem: Given a directed graph $G$, and two vertices $s$ and $t$, we ask whether there is a directed path from $s$ to $t$. We denote by GAP the set of all instances $(G, s, t)$ where a directed path exists in $G$ from $s$ to $t$. (This problem, when restricted to undirected graphs, is also very important. By replacing each undirected edge by a pair of directed edges in the opposite direction, it is clear that the directed GAP is a more general problem than the undirected version.)

For GAP, there is a simple polynomial time algorithm based on Depth-First-Search (DFS). However DFS uses at least linear space. By contrast, if the space bound is limited, we have the following non-deterministic algorithm: Start from $v_0 = s$, successively guess the next vertex $v_i$ and verify that $(v_{i-1}, v_i)$ is an edge. We can keep a counter to count up to $n$, and accept iff within $n - 1$ steps, some $v_k = t$. Note that we only need to keep at any time a pair of current vertices on the work tape; when we guess for $v_{i+1}$ we need only to remember $v_i$ (in order to verify that $(v_i, v_{i+1})$ is an edge), but we no longer need to keep

[4] We will not be concerned with non-terminating computations. For example, one can design a subroutine which acts as a time clock that runs exactly $n^i + i$ steps and shuts itself off, or a space bound which marks $f(n)$ (e.g. $f(n) = \log n$ or $n^i + i$) many tape squares.

$v_{i-1}$ and all previously guessed vertices. In addition we only need to keep a counter to count up to $n$. For a graph with $n$ vertices, to name a vertex takes only $\log n$ space. Thus GAP is in non-deterministic logspace.

We will only be concerned with time/space bounded NTM, and thus we can assume each computational path terminates within the specified time/space bound. Then for a NTM $N$ and a fully time constructible $f$ we define

$$\text{time}_N(x) = \max\{n+1, \min_p\{\#\text{ of steps along } p, \text{ where } N(x) \text{ accepts along computational path } p\}\},$$ [5]

$$\text{time}_N(n) = \max\{\text{time}_N(x), |x| = n\},$$

$$\text{NTIME}[f] = \{L \mid \text{ for some } N\ , L(N) = L, \text{ and } \text{time}_N(n) \leq f(n), \text{ for all large } n.\}.$$

Of course $L(N) = L$ means that $x \in L$ iff there is some computation path $p$, along which $N$ accepts $x$. Similarly we can define non-deterministic space classes.

$$\text{space}_N(x) = \max\{1, \min_p\{\#\text{ of worktape cells used in } N(x), \text{ where } N(x) \text{ accepts along } p\}\},$$ [6]

$$\text{space}_N(n) = \max\{\text{time}_N(x), |x| = n\},$$

$$\text{NSPACE}[f] = \{L \mid \text{ for some } N\ , L(N) = L, \text{ and } \text{space}_N(n) \leq f(n), \text{ for all large } n.\}.$$

$$\text{NP} = \bigcup_{f \in \text{poly}} \text{NTIME}[f],$$

$$\text{NPSPACE} = \bigcup_{f \in \text{poly}} \text{NSPACE}[f],$$

and also

$$\text{NL} = \bigcup_{c>0} \text{NSPACE}[c \log].$$

To recap, formally a non-deterministic TM (NTM) is simply a TM which has possibly more than one legal moves at any configuration. These next-moves are specified by the current state and the tape symbols it is currently reading. A NTM accepts an input $x$ iff there exists a legal computational path that ends in acceptance.

It is obvious that L $\subseteq$ NL $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE $\subseteq$ EXP.

We will show later that NSPACE$[f] \subseteq$ DSPACE$[f^2]$, thus NPSPACE = PSPACE and NPSPACE will not be used.

The relationship of P and NP is *the most outstanding* open problem in Computer Science.

[5] When $N$ does not accept $x$, the time complexity is $n + 1$.

[6] When $N$ does not accept $x$, the space complexity is 1.

## 1.4 Hierarchy Theorems

The complexity classes as defined attempt to classify *decidable* computational problems according to the *inherent* computational complexity of each problem. The first question one must ask is: Does such a stratification really exist, i.e., could this be merely a definitional mirage?

In order to truly establish the validity of the existence of computational complexity theory, one must prove that problems do indeed have different inherent computational complexity. This should not depend on merely the fact that, while we have fast algorithms for some problems, we do not *currently* have fast algorithms for some others. The proof must establish for a given time or space bound $f(n)$, the existence of *some* decidable computational problems which do not possess *any* algorithm within that bound $f(n)$.

The results establish these existence theorems are known as Hierarchy Theorems. One can show, for any two time complexity functions $T_1(n)$ and $T_2(n)$, if $T_2$ grows sufficiently fast compared to $T_1$, then there are indeed problems which can be solved in time $T_2$ but can not be solved in time $T_1$. (Here we will also require a technical condition of $T_2$ being time constructible.) In fact, if $\lim_{n\to\infty} \frac{T_1(n)\log T_1(n)}{T_2(n)} = 0$, then the above statement already holds. We will prove a weaker theorem

THEOREM 1.2 *Given any $T_1(n), T_2(n)$, if $T_2(n)$ is time constructible and $\lim_{n\to\infty} \frac{T_1(n)^2}{T_2(n)} = 0$, then there is a language $L \in \mathrm{DTIME}[T_2(n)] - \mathrm{DTIME}[T_1(n)]$.*

The proof also establishes in particular,

THEOREM 1.3 *Given any totally recursive time bound $T(n)$, there is a recursive language $L$ not in $\mathrm{DTIME}[T(n)]$.*

There are also nonderministic time hierarchy theorems.

The Hierarchy Theorem plays the same role as the existence of undecidable problems. Not only that, the proof also adapts the diagonalization method. (We have now seen it for the third time.) On the minus column, just as Cantor's slick proof establishing the existence of transcendental numbers, these Hierarchy Theorems do not give us the specificity that certain well known problems are hard. The present day research in complexity theory is much more dominated by the quest for such "real" problems.

We will only give a proof sketch:

*Proof.* The general idea is diagonalization. First we note that all TMs can be effectively enumerated as $M_1, M_2, \ldots$. For a time constructible bound $T(n)$, one can also design a "clock", a subroutine, which runs concurrently with any TM and shuts it off after $T(n)$ steps. We can effectively enumerate all "clocked" TMs, all of which runs within time $T(n)$, and every language in $\mathrm{DTIME}[T(n)]$ is accepted by one machine on this list.

We will enumerate all the Turing Machines $M_1, M_2, \ldots$, and consider each $M_i$ on longer and longer inputs. We simulate each of them on these inputs, for up to $T_2(n)$ steps for inputs of length $n$. To be more precise, we will allocate a disjoint segment $S_i = \{\langle i, y\rangle \mid i \geq 1, y \in \Sigma^*\}$, and simulate $M_i$ on every $x \in S_i$. If the simulation on $M_i(x)$ terminates in less than $T_2(|x|)$, then *we will do the opposite,* i.e., we accept $x$ if $M_i(x)$ rejects, and we reject $x$ if $M_i(x)$ accepts. If the simulation of $M_i(x)$ does not terminate within $T_2(|x|)$ steps, then we can decide arbitrarily, accept or reject.

To account for the possibility that the simulation of $M_i(x)$ does not terminate within $T_2(|x|)$ steps, we allocated an infinite subset of inputs for every $M_i$, Since one can simulate (easily) a multitape TM $M_i$ with running time $T_1(n)$ by a one tape TM in time $T_1(n)^2$, for sufficiently large $x \in S_i$, the simulation of $M_i(x)$ will terminate within $T_2(|x|)$ steps. Now suppose $L \in \text{DTIME}[T_1(n)]$. Then for some TM $M_i$ accepting $L$ with runing time $T_1(n)$, the simulation will terminate with a different outcome, for sufficiently large $n$. Thus the language $L'$ defined by this simulation does not belong to $\text{DTIME}[T_1(n)]$. However, by the construction it is in $\text{DTIME}[T_2(n)]$. ♣

Note that the simulation is carried out by a single TM, which must have a fixed number of tapes. It is known that any $k$-tape TM with running time $T(n)$ can be simulated by a 2-tape TM in time $O(T(n)\log(T(n)))$. This is the only modification needed in the above proof to get a tighter hierarchy theorem quoted earlier.

The method of this proof is sometimes called *delayed diagonalization.* This is because, the "diagonalizing action" which kills the TM $M_i$ may happen at a later stage (in our case, for an input in $S_i$, perhaps much longer than $i$). The set $S_i$ has infintely many strings, thus arbitrarily long strings. Thus the "diagonalizing action" will happen eventually, for every TM with running time $T_1(n)$.

There is also a similar version of the Hierarchy Theorem for space complexity. In fact since for space complexity we can simulate any TM in just one tape without any loss in asymptotic space complexity, the theorem reads even tighter: ($S_2(n) \geq \log n$ is needed to carry out some basic counting operations in simulations.)

THEOREM 1.4 *Given any $S_1(n)$ and $S_2(n) \geq \log n$, if $S_2(n)$ is fully space constructible and $\lim_{n\to\infty} \frac{S_1(n)}{S_2(n)} = 0$ (i.e, $S_1 = o(S_2)$), then there is a language $L \in \text{DSPACE}[S_2(n)]$ $-\text{DSPACE}[S_1(n)]$.*

The approach is the same.

**Exercise:** Prove the Space Hierarchy Theorem.