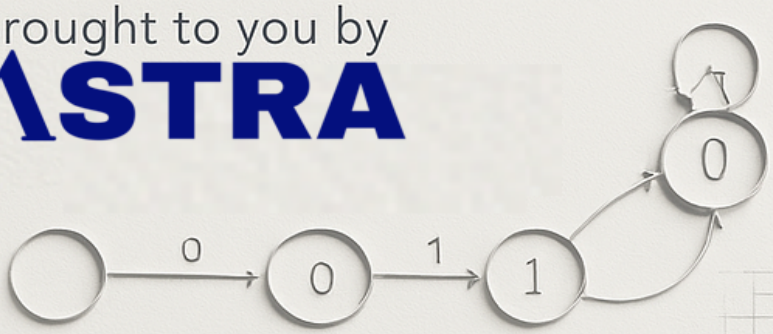
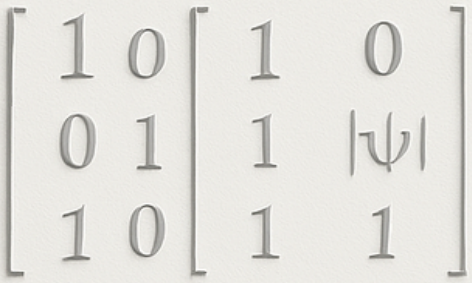
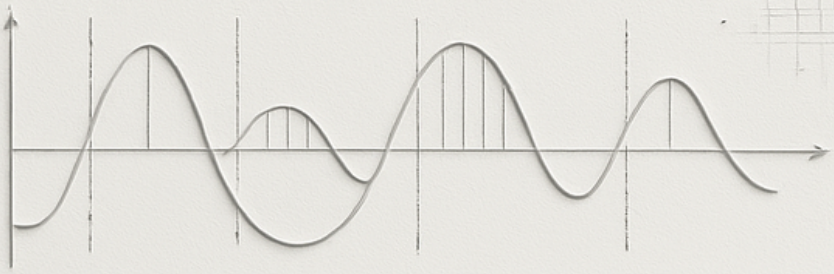
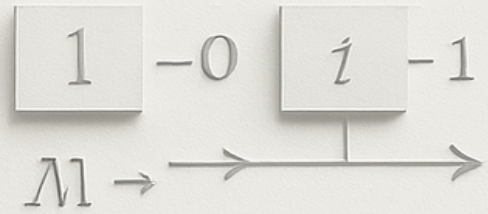
Brought to you by
ASTRA



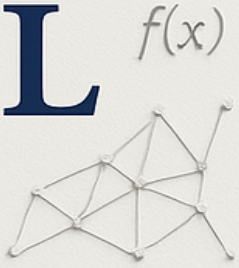


QUANTUM COMPUTING

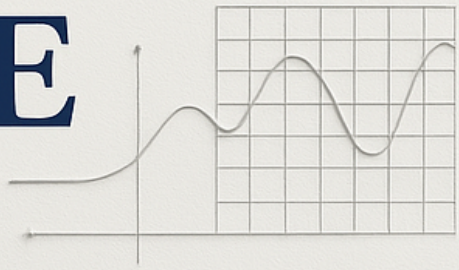




THEORETICAL COMPUTER SCIENCE

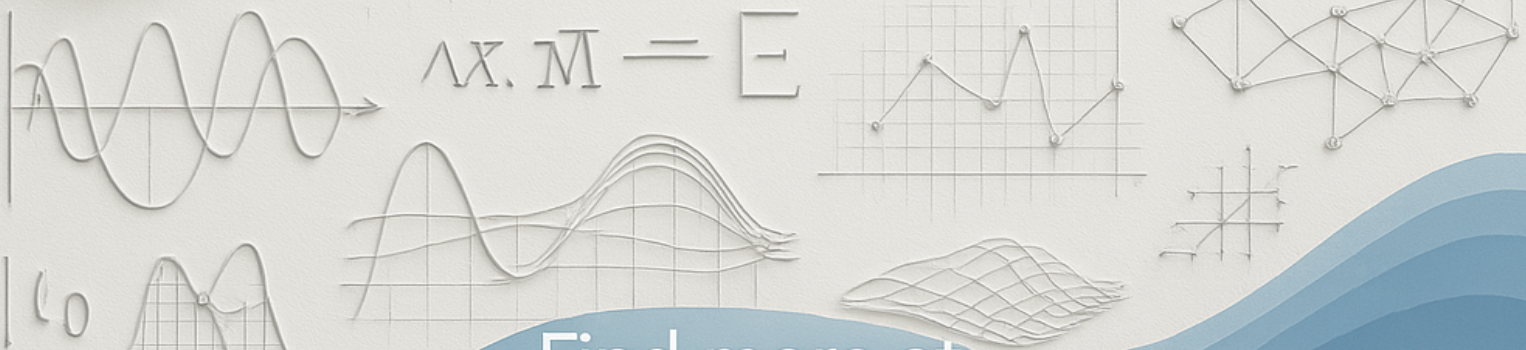


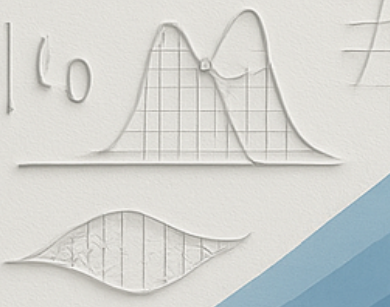
ELECTIVE



Written by
Giorgio Micaletto

Turing Machine





Find more at
astrabocconi.it

Contents

Lecture 1: Foundations of Computation

The study of theoretical computer science begins with a fundamental observation: the manner in which a problem is represented significantly influences our understanding and ability to solve it. This insight serves as a foundational pillar upon which much of computation theory rests.

1.1 Problem Complexity

Consider a *9-stone game*, where two players alternately select from nine stones numbered 1 through 9. A player wins by holding any subset of three stones whose sum is exactly 15. While seemingly challenging, this game can be elegantly represented as a variant of *tic-tac-toe* played on a magic square. In this transformed representation, the complexity dissolves due to the familiarity and simplicity of tic-tac-toe.
This transformation underscores an essential concept:

> *The complexity of problem-solving often hinges upon how we represent and interpret the problem.*

Another illustrative example emerges from graph theory. Given a graph $\mathcal{G} = (V, E)$ and two vertices $s, t \in V$, we ask:

- **Eulerian Path:** Is there a path from s to t traversing each edge exactly once?
- **Hamiltonian Path:** Is there a path from s to t traversing each vertex exactly once?

Remarkably, while Eulerian paths admit efficient algorithms (via Fleury's algorithm or Hierholzer's algorithm), Hamiltonian paths currently lack known efficient solutions. Despite their apparent similarity, their computational complexities diverge drastically, prompting fundamental inquiries into the nature of *computational hardness*.
A further motivating example arises in number theory:

1. **Multiplication:** Given primes P, Q, compute $N = PQ$ (computationally trivial).
2. **Primality Testing:** Given N, determine whether it is prime (efficient via polynomial-time algorithms like AKS).
3. **Factoring:** Given $N = PQ$, find primes P, Q (no known polynomial-time algorithm exists).

Notably, verifying correctness ($N = PQ$ given P, Q) is computationally straightforward, yet finding the solution (P, Q given N) is not. This dichotomy encapsulates the celebrated P vs. NP problem and raises fundamental questions:

Is verification inherently easier than discovery?
Is decision inherently simpler than search?

1.2 Classification of Problems and the Turing Machine

Problems broadly divide into three classes:

- **Undecidable Problems:** No algorithm exists (e.g., the Halting Problem, Tiling Problem).
- **Decidable in Unreasonable Time:** Solutions exist but only with impractically large computational resources (e.g., Hamiltonian Path, Factoring).
- **Decidable in Reasonable Time:** Efficient solutions exist (e.g., Eulerian Path).

The precise boundary between reasonable and unreasonable remains unresolved ($P = NP$ conjecture).

1.3 Models of Computation

Before formalizing computation via Turing machines, we first introduce several simpler automata models that correspond to increasingly powerful language classes. These models capture distinct computational capabilities and serve as the foundation for the Chomsky hierarchy.

Definition 1.1 (Deterministic Finite Automaton)**.** A *deterministic finite automaton* (DFA) is a 5-tuple

$$M = (Q, \Sigma, \delta, q_0, F),$$

where:

- Q is a finite set of states,
- Σ is a finite input alphabet,
- $\delta : Q \times \Sigma \to Q$ is the transition function,
- $q_0 \in Q$ is the start state,

- $F \subseteq Q$ is the set of accept (final) states.

A string $w \in \Sigma^*$ is accepted by M if the extended transition function δ^* satisfies $\delta^*(q_0, w) \in F$.

Definition 1.2 (Nondeterministic Finite Automaton)**.** A *nondeterministic finite automaton* (NFA) is a 5-tuple

$$M = (Q, \Sigma, \delta, q_0, F),$$

where:

- Q, Σ, q_0, F are as in the DFA,
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ is the transition function, mapping to subsets of states.

M accepts a string $w \in \Sigma^*$ if there exists a sequence of transitions starting at q_0 that processes w and ends in a state $q \in F$.

Definition 1.3 (Nondeterministic Pushdown Automaton)**.** A *nondeterministic pushdown automaton* (NPDA) is a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F),$$

where:

- Q is a finite set of states,
- Σ is the input alphabet,
- Γ is the stack alphabet,
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^*)$ is the transition function,
- $q_0 \in Q$ is the initial state,
- $Z_0 \in \Gamma$ is the initial stack symbol,
- $F \subseteq Q$ is the set of accept states.

An NPDA accepts a string if it can process the entire input and reach a state in F, possibly leaving any content on the stack.

Definition 1.4 (Linear Bounded Automaton)**.** A *linear bounded automaton* (LBA) is a non-deterministic Turing machine where the tape is restricted to a finite portion—specifically, the tape head is confined to the portion containing the input and at most a constant factor more space. Formally, it is a Turing machine that uses space $\mathcal{O}(n)$ for input of length n. LBAs recognize exactly the class of *context-sensitive languages*.

However, to rigorously analyze computational problems, we formalize computation using the concept of the *Turing Machine* (TM), introduced by Alan Turing (1936). Formally, we have the following definition:

Definition 1.5 (Turing Machine)**.** A Turing Machine is defined as a 7-tuple:

$$\text{TM} = \big(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}\big)$$

with components:

1. Q: Finite set of states.
2. Σ: Finite input alphabet (not containing the blank symbol _).
3. Γ: Finite tape alphabet, containing blank symbol _, with $\Sigma \subset \Gamma$.
4. δ: Transition function, $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\text{L}, \text{R}\}$.
5. q_0: Initial state.
6. q_{accept}: Accepting state.
7. q_{reject}: Rejecting state.

A Turing Machine configuration includes:

1. Tape contents (uqv, with q indicating the head position).
2. Current machine state.
3. Head position on the tape.

A configuration C_1 *yields* another configuration C_2 if C_2 is reachable from C_1 through a single application of the transition function δ. Computation is defined as a finite sequence of configurations:

$$C_1 \to C_2 \to \cdots \to C_n$$

A Turing Machine M **accepts** an input w if:

- C_1 is the initial configuration on input w.
- C_n is an accepting configuration (state q_{accept}).

Conversely, M **rejects** w if the final configuration reaches q_{reject}.

Definition 1.6 (Nondeterministic Turing Machine)**.** A *nondeterministic Turing machine* (NTM) is a tuple

$$M = \big(Q, \Sigma, \Gamma, \delta, q_0, q_{\texttt{accept}}, q_{\texttt{reject}}\big)$$

where:

- Q is a finite set of states.
- Σ is a finite input alphabet that does not include the blank symbol $\sqcup$.
- Γ is a finite tape alphabet, with $\Sigma \subseteq \Gamma$ and including the blank symbol $\sqcup$.
- $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$ is the transition function, which maps a state and a tape symbol to a *set* of possible moves. Each move consists of a triple (q, a, D) where $q \in Q$ is a next state, $a \in \Gamma$ is the symbol to be written on the tape, and $D \in \{L, R\}$ indicates the direction in which the tape head moves.
- $q_0 \in Q$ is the start state.
- $q_{\texttt{accept}} \in Q$ is the accept state.
- $q_{\texttt{reject}} \in Q$ is the reject state, with $q_{\texttt{reject}} \neq q_{\texttt{accept}}$.

An NTM operates nondeterministically by, at each step, choosing one of the possible moves specified by δ for the current state and tape symbol. The machine accepts an input if there exists at least one sequence of moves (i.e., a computation branch) that eventually leads to the accept state $q_{\texttt{accept}}$. If no branch reaches $q_{\texttt{accept}}$, the input is rejected (either by reaching $q_{\texttt{reject}}$ in every branch or by some branches not halting).

Proposition 1.1 (Automata Hierarchy)**.** *There exists a strict hierarchy among classes of automata based on their computational power, as follows:*

$$\text{DFA} \subsetneq \text{NFA} \subsetneq \text{NPDA} \subsetneq \text{LBA} \subsetneq \text{TM} \subsetneq \text{NTM}$$

The Church–Turing thesis asserts that Turing Machines adequately capture the intuitive notion of computation. More precisely:

Proposition 1.2. *Any computation achievable by a physically realizable computational device can be simulated by a Turing Machine.*

This thesis, though empirically supported, remains non-provable, yet it serves as the cornerstone for theoretical computation.

1.4 Computational Problems

Computational problems are generally categorized into:

- **Search Problems:** Defined by a function $f : \Sigma^* \to \Sigma^*$. Example: Given N, find (P, Q) such that $N = PQ$.
- **Decision Problems:** Defined by a language $\mathcal{L} \subseteq \Sigma^*$. Decision problems output binary answers (`YES`/`NO`). A decision problem is a special case of a search problem.

Inputs to these problems are strings from a finite alphabet Σ (commonly $\{0, 1\}$), and problems vary by their computational complexity and decidability.

Lecture 2: Decidability and Recognizability

A Turing Machine (TM) computation on an input w has three distinct possible outcomes:

1. Halting and accepting,
2. Halting and rejecting,
3. Never halting (the machine runs indefinitely without necessarily repeating states).

Definition 2.1 (Decider Turing Machines)**.** A Turing Machine M that halts for every input is called a *decider*.

If M accepts all $w \in \mathcal{L}$ and rejects all $w \notin \mathcal{L}$, we say M **decides** the language $\mathcal{L}$.

Definition 2.2 (Recognizer Turing Machine)**.** If M accepts all inputs $w \in \mathcal{L}$ and either rejects or never halts for inputs $w \notin \mathcal{L}$, we say M *recognizes* the language $\mathcal{L}$.

We denote $\mathcal{L}(M)$ as the set of strings that M accepts. This formulation motivates critical questions:

- Are all languages decidable?
- Are all functions computable?
- Are all languages recognizable?

2.1 Language Classification

Languages can be categorized based on their computability properties.

Definition 2.3 (Regular Languages)**.** A language $\mathcal{L} \subseteq \Sigma^*$ is called *regular* if there exists a deterministic finite automaton (DFA) $M = (Q, \Sigma, \delta, q_0, F)$ such that

$$\mathcal{L} = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\},$$

where δ^* is the extended transition function.

Definition 2.4 (Context-Free Languages (CF))**.** A language $\mathcal{L} \subseteq \Sigma^*$ is called *context-free* if there exists a context-free grammar $G = (V, \Sigma, R, S)$ such that $\mathcal{L} = \mathcal{L}(G)$, where:

- V is a finite set of variables (nonterminals),

- Σ is the finite input alphabet,
- $R \subseteq V \times (V \cup \Sigma)^*$ is a finite set of production rules of the form $A \to \gamma$ with $A \in V$, $\gamma \in (V \cup \Sigma)^*$,
- $S \in V$ is the start symbol.

Equivalently, $\mathcal{L}$ is accepted by a nondeterministic pushdown automaton (NPDA).

Definition 2.5 (Turing-decidable Languages)**.** A language $\mathcal{L} \subseteq \Sigma^*$ is called *Turing-decidable* (or *recursive*) if there exists a Turing machine M such that for all $w \in \Sigma^*$, M halts on input w and:

$$M(w) = \begin{cases} \text{accept}, & \text{if } w \in \mathcal{L}, \\ \text{reject}, & \text{if } w \notin \mathcal{L}. \end{cases}$$

Definition 2.6 (Turing-recognizable Languages)**.** A language $\mathcal{L} \subseteq \Sigma^*$ is called *Turing-recognizable* (or *recursively enumerable*) if there exists a Turing machine M such that for all $w \in \Sigma^*$:

$$w \in \mathcal{L} \iff M(w) \text{ accepts.}$$

If $w \notin \mathcal{L}$, then $M(w)$ either rejects or loops forever (i.e., may not halt).

Clearly,

$$\mathrm{R} \subseteq \mathrm{RE}$$

A stricter condition, $\mathrm{R} \subset \mathrm{RE}$, holds, and this strictness shall be demonstrated later.

Proposition 2.1 (Canonical inclusion chain)**.**

$$\textit{Regular} \subset \textit{Context-Free (CF)} \subset \textit{Turing-decidable} \subset \textit{Turing-recognizable}$$

Theorem 2.1 (Closure under complement)**.** *If* $\mathcal{L} \in \mathrm{R}$*, then* $\overline{\mathcal{L}} \in \mathrm{R}$*.*

Proof. Let M be a Turing Machine that decides $\mathcal{L}$. This means that on any input w, M halts and either accepts (if $w \in \mathcal{L}$) or rejects (if $w \notin \mathcal{L}$). We construct a new Turing Machine M' to decide $\overline{\mathcal{L}}$ as follows:

1. On input w, simulate M on w.
2. If M accepts, then M' rejects.
3. If M rejects, then M' accepts.

Since M halts on all inputs, M' also halts on all inputs. Thus, M' decides $\overline{\mathcal{L}}$. □

Theorem 2.2 (Intersection Property)**.** *If* $\mathcal{L} \in \text{RE}$ *and* $\overline{\mathcal{L}} \in \text{RE}$*, then* $\mathcal{L} \in \text{R}$*.*

Proof. Let M_1 be a Turing Machine that recognizes $\mathcal{L}$ and M_2 a Turing Machine that recognizes $\overline{\mathcal{L}}$. We construct a new machine M that decides $\mathcal{L}$ using *dovetailing*, a technique where both M_1 and M_2 are simulated in parallel.

1. On input w, start simulating M_1 and M_2 on w in parallel. This can be done by alternating one step of M_1 with one step of M_2, repeating indefinitely until one of them halts.
2. If M_1 accepts w, then M accepts w.
3. If M_2 accepts w, then M rejects w.

Since $w \in \mathcal{L}$ or $w \in \overline{\mathcal{L}}$ (but not both), one of the two machines must eventually accept. Thus, M always halts and correctly decides whether $w \in \mathcal{L}$. Hence, $\mathcal{L} \in \text{R}$. □

Corollary 2.1. *If* $\text{RE} \setminus \text{R} \neq \varnothing$*, then there exists a language* $\mathcal{L} \in \text{RE}$ *such that* $\overline{\mathcal{L}} \notin \text{RE}$*.*

Proof. Assume for contradiction that for every $\mathcal{L} \in \text{RE}$, we also have $\overline{\mathcal{L}} \in \text{RE}$. Then by the Intersection Theorem, every $\mathcal{L} \in \text{RE}$ would also be in R, because both $\mathcal{L}$ and $\overline{\mathcal{L}}$ are recognizable implies $\mathcal{L}$ is decidable. This would imply that $\text{RE} = \text{R}$, contradicting the assumption that $\text{RE} \setminus \text{R} \neq \varnothing$. Therefore, there must exist some language $\mathcal{L} \in \text{RE}$ such that $\overline{\mathcal{L}} \notin \text{RE}$. □

2.2 Variants of Turing Machines

We often consider variants of the basic Turing Machine model:

1. **Stay-transition Machine**: Transition function allows head to remain stationary:
$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{\text{L}, \text{R}, \text{S}\}$$
2. **Multi-tape Turing Machine**: Uses k tapes, each with an independent head.
3. **Non-deterministic Turing Machine**: Allows multiple computational branches.

These variants are all equivalent computationally:

Theorem 2.3 (Equivalence of Turing Machine Models)**.** *Every* k*-tape or non-deterministic Turing Machine has an equivalent single-tape deterministic Turing Machine.*

Proof. Let M be a k-tape Turing Machine. We aim to construct a single-tape Turing Machine S that simulates M.
Tape Encoding: On the single tape of S, we interleave the k tapes of M using a delimiter symbol (e.g., $\#$). A configuration of M is encoded as:

$$\#u_1a_1v_1\#u_2a_2v_2\#\ldots\#u_ka_kv_k\#$$

where each $u_ia_iv_i$ represents the content of tape i with the head scanning symbol a_i, and u_i, v_i are the portions to the left and right of the head, respectively.
Head Tracking: The position of each tape head is tracked by a marked symbol (e.g., overlining or tagging a_i) indicating the current scan position.
Simulation Procedure:

1. The machine S simulates one step of M by scanning the entire tape to extract the symbols under each virtual head $a_1, \ldots, a_k$.

2. Based on the current state and tuple $(a_1, \ldots, a_k)$, S computes the transition of M: write symbols, move directions, and new state.

3. S performs a second pass over the tape to update each a_i to its new symbol and move the virtual head one cell in the indicated direction (left or right).

Each step of M is thus simulated in a finite number of steps by S using bounded passes over the encoded tape. Though this simulation incurs a polynomial overhead, it is still effective. Hence, the computational power of a multi-tape Turing Machine is equivalent to that of a single-tape Turing Machine. □

Formally, we have the following definition:

Definition 2.7 (k-tape Turing Machine)**.** A k-tape Turing Machine is described by:

$$\text{TM} = \big(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}\big)$$

with the following modifications:

1. $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{\text{L}, \text{R}\}^k$

2. Each tape has an independent head.

2.3 Algorithms and Computability: Hilbert's 10th Problem

The notion of an algorithm precedes formalization in computer science. David Hilbert, in 1900, posed his 10th problem:

> *Given a polynomial with integer coefficients, devise an algorithm deciding if the polynomial has an integer root.*

We now represent Hilbert's problem formally as:

$$D = \{p \mid p \text{ polynomial with integer root}\}$$

In 1970, Matiyasevich proved the undecidability of this set, concluding a significant open question:

Theorem 2.4 (Matiyasevich, 1970)**.** *The set D is undecidable.*

Without a rigorous definition of algorithms however, problems like Hilbert's remained elusive. In 1936, Alonzo Church and Alan Turing independently introduced formal models of computation that aimed to capture the intuitive notion of an algorithm.

Church proposed the *lambda calculus* (λ-calculus), a minimalist formal system grounded in function abstraction and application. Computation in this model proceeds via symbolic reduction of expressions, using transformation rules such as α-conversion, β-reduction, and η-conversion. Despite its syntactic simplicity, lambda calculus is capable of expressing any computable function.

Turing introduced the *Turing Machine*, a mechanistic abstraction that models computation through a finite set of states, an infinite tape for memory, and a read/write head governed by a transition function. It simulates computation by altering tape symbols and moving left or right, effectively representing algorithmic procedures in a stepwise fashion.

Though distinct in structure, the lambda calculus and Turing Machines were later shown to be equivalent in computational power. This foundational equivalence led to the **Church–Turing Thesis**:

Proposition 2.2 (Church–Turing Thesis)**.** *Every effectively computable function (intuitively computable by an algorithm) is computable by a Turing Machine.*

Lecture 3: Unsolvability and the Halting Problem

Consider the language:

$$D = \{p \mid p \text{ is a polynomial with an integer root}\}.$$

Consider a simpler case:

$$D_1 = \{p \mid p \text{ is a polynomial over one variable } x \text{ with an integer root}\}.$$

We can construct a Turing Machine M_1 that *recognizes* D_1:

1. Iterate over integer values: $x = 0, 1, -1, 2, -2, \ldots$
2. Evaluate $p(x)$ for each value.
3. If $p(x) = 0$ for some x, accept.

This approach works because we can enumerate all integers and halt if a root is found. However, for multivariate polynomials $p(x_1, \ldots, x_k)$, no such recognizer suffices to yield a decider. The search space $\mathbb{Z}^k$ cannot be exhaustively covered by a finite-time procedure.

3.1 Encoding of Inputs and Meta-Computational Reasoning

To describe a Turing Machine precisely, we must specify:

- The set of states Q
- The input and tape alphabets Σ and Γ
- The transition function δ
- The initial, accept, and reject states

From now on, inputs to a Turing Machine will include formal mathematical objects such as graphs, vectors, polynomials, grammars, and automata. These must be encoded as strings over a finite alphabet:

- The encoding of an object O is denoted $\langle O \rangle$
- The encoding of a tuple $O_1, \ldots, O_k$ is $\langle O_1, \ldots, O_k \rangle$

This allows us to reason about machines that operate on descriptions of machines and other structured data.

3.2 Problems involving Turing Machine

There are problems that can't be computed, and understanding what *cannot* be computed sheds light on the fundamental capabilities and limitations of mechanized reasoning. Furthermore:

- It clarifies the boundaries of decidability and encourages problem simplification.
- It informs us about the inherent limitations of formal systems.
- It probes the adequacy of our computational models.

There are, however, algorithmically interesting computable problems:

- Does a Turing Machine M accept a given input w?
- Is the language of a Turing Machine empty?
- Are two Turing Machines equivalent (i.e., accept the same language)?

Turing Machines can be represented as strings, and so a Turing Machine can itself take another machine as input. This enables a meta-theoretic investigation of machines that simulate others.
For instance, define:

Example 3.1 (Acceptance Problem)**.**

$$A_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a Turing machine and } M \text{ accepts } w\},$$

Theorem 3.1. *A_{TM} is Turing-recognizable.*

Proof. We construct a Turing Machine U (called the Universal Turing Machine) that simulates any Turing Machine M on input w. Given an input $\langle M, w\rangle$, U proceeds as follows:

1. Decode the string $\langle M\rangle$ to recover the description of M: its states, transition function, alphabets, and special states.
2. Simulate the computation of M on input w step by step.
3. If M accepts w, then U accepts $\langle M, w\rangle$.
4. If M rejects w, U rejects $\langle M, w\rangle$.
5. If M loops indefinitely, so does U.

Thus, U recognizes A_{TM}, meaning it accepts precisely those strings of the form $\langle M, w \rangle$ such that M accepts w. □

Theorem 3.2. *A_{TM} is undecidable.*

Proof. Assume, for contradiction, that there exists a Turing Machine H that decides A_{TM}. Then, we construct a Turing Machine D as follows: On input $\langle M \rangle$, D runs H on input $\langle M, \langle M \rangle \rangle$. If H accepts, then D rejects. If H rejects, then D accepts. That is,

$$D(\langle M \rangle) = \begin{cases} \text{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ \text{reject} & \text{if } M \text{ accepts } \langle M \rangle \end{cases}$$

Now consider what happens when we run D on its own description, $\langle D \rangle$:

$$D(\langle D \rangle) = \begin{cases} \text{accept} & \text{if } D \text{ does not accept } \langle D \rangle \\ \text{reject} & \text{if } D \text{ accepts } \langle D \rangle \end{cases}$$

This is a contradiction. Therefore, no such decider H can exist. Hence, A_{TM} is undecidable. □

We now show that not all languages are Turing-recognizable.

Definition 3.1. A set A is *countable* if it is finite or if there exists a bijection $f : \mathbb{N} \to A$. Otherwise, it is *uncountable*.

Theorem 3.3. *The set of Turing Machines is countable.*

Proof. Turing Machines are described by finite strings over a finite alphabet (e.g., $\{0, 1\}$). The set of all such strings is countably infinite because strings over a finite alphabet can be enumerated lexicographically. Therefore, the set of Turing Machines is countable. □

Theorem 3.4. *The set of languages over Σ^* is uncountable.*

Proof. Each language is a subset of Σ^*, and the set of all subsets of a countably infinite set is uncountable (Cantor's Theorem). Therefore, the power set $\mathcal{P}(\Sigma^*)$ is uncountable. □

Corollary 3.1. *There exist languages that are not Turing-recognizable.*

Proof. Since there are only countably many Turing Machines (and hence countably many Turing-recognizable languages), but uncountably many languages over Σ^*, it follows that some languages are not recognized by any Turing Machine. □

Theorem 3.5. $\overline{A}_{\text{TM}} \notin \text{RE}$.

Proof. By Theorem 2.2, we have that $A_{\text{TM}} \in RE$ and $A_{\text{TM}} \notin R$, thus it follows that $\overline{A}_{\text{TM}} \notin RE$. Another proof is the following:
Assume, for contradiction, that $\overline{A}_{\text{TM}} \in \text{RE}$. Then both A_{TM} and $\overline{A}_{\text{TM}}$ would be recognizable. We can construct a decider as follows:

1. Given input $\langle M, w \rangle$, simulate both recognizers in parallel (e.g., via dovetailing).
2. If the recognizer for A_{TM} accepts, then accept.
3. If the recognizer for $\overline{A}_{\text{TM}}$ accepts, then reject.

Because either $\langle M, w \rangle$ is in A_{TM} or its complement, one of these recognizers must eventually accept. Thus, the simulation halts, and we have constructed a decider for A_{TM}, contradicting its undecidability. □

This result illustrates a fundamental limitation in computation: not only are some languages undecidable, but some cannot even be *recognized* by any machine.

Lecture 4: Undecidability and Mapping Reducibility

Many languages are non-recognizable; one classical example is Example 3.1 which is known to be non-decidable.

Another such language is the Halting Problem.

Theorem 4.1. HALT_{TM}, *where*

$$\text{HALT}_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a Turing machine and } M \text{ halts on input } w\},$$

is undecidable.

Proof. Assume, toward a contradiction, that HALT_{TM} is decidable. That is, suppose there exists a Turing machine H that decides HALT_{TM}.
We now show how to decide A_{TM} with the help of H, contradicting the known undecidability of A_{TM}.
Define a computable function f that transforms an instance $\langle M, w\rangle$ into an instance $\langle M', w'\rangle$ as follows. Construct a Turing machine M' that, on any fixed input w' (the specific form of w' is irrelevant), works as follows:

(a) Simulate M on input w.

(b) If M accepts w, then M' halts (for example, by entering an accepting state).

(c) If M rejects w, then M' enters an infinite loop.

Thus, by construction, we have:

$$\langle M, w\rangle \in A_{\text{TM}} \iff \langle M', w'\rangle \in \text{HALT}_{\text{TM}}.$$

Now, using the decider H for HALT_{TM}, we can decide whether $\langle M', w'\rangle$ is in HALT_{TM} and, therefore, decide whether M accepts w. This yields a decider for A_{TM}, a contradiction. Hence, HALT_{TM} is undecidable. □

Another undecidable problem is defining the emptiness of $\mathcal{L}(M)$, where $\mathcal{L}(M)$ denotes the set of strings accepted by a Turing machine M.
Formally,

$$E_{\text{TM}} = \{\langle M\rangle \mid M \text{ is a Turing machine and } \mathcal{L}(M) = \varnothing\}.$$

Theorem 4.2. E_{TM} *is undecidable.*

Proof. We prove undecidability by reducing A_{TM} to E_{TM}. Assume there exists a decider D for E_{TM}.
Given an instance $\langle M, w\rangle$ for A_{TM}, we construct a Turing machine M' that operates as follows on any input x:

(a) Ignore the input x and simulate M on input w.

(b) If M accepts w, then M' accepts (for instance, on any input).

(c) If M does not accept w (i.e., it rejects or loops), then M' never accepts any input.

Thus, we have:

$$\langle M, w\rangle \in A_{\text{TM}} \iff \mathcal{L}(M') \neq \varnothing \iff \langle M'\rangle \notin E_{\text{TM}}.$$

Now, if D could decide E_{TM}, one could decide A_{TM} by computing M' from $\langle M, w\rangle$ and then running D on $\langle M'\rangle$. This contradicts the undecidability of A_{TM}. Hence, E_{TM} is undecidable. □

4.1 Reduction Techniques: Mapping Reducibility and Computable Functions

A reduction is a conversion of a problem A into a problem B such that a solution to B can be used to solve A.

Definition 4.1 (Computable Functions)**.** A function $f : \Sigma^* \to \Sigma^*$ is *computable* if there exists a Turing machine M such that on every input w, M halts with the output $f(w)$ written on its tape.

Definition 4.2 (Mapping Reducibility)**.** A language A is mapping reducible to a language B, denoted $A \leq_m B$, if there exists a computable function $f : \Sigma^* \to \Sigma^*$ such that for all $w \in \Sigma^*$,

$$w \in A \iff f(w) \in B.$$

Theorem 4.3. *If $A \leq_m B$ and B is decidable, then A is decidable.*

Proof. Let f be a computable function reducing A to B and suppose there exists a decider M_B for B. Then we construct a Turing machine N that, on input w, performs the following steps:

(i) Compute $f(w)$.

(ii) Run M_B on $f(w)$.

(iii) If M_B accepts, then accept w; otherwise, reject w.

Since both the computation of f and the decider M_B always halt, N always halts and decides A. Hence, A is decidable. □

Corollary 4.1. *If $A \leq_m B$ and A is undecidable, then B is undecidable.*

Returning to the Halting Problem, we now provide a reduction from A_{TM} to HALT_{TM}. Recall that

$$\text{HALT}_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a Turing machine and } M \text{ halts on input } w\}.$$

We define a computable function f such that for any instance $\langle M, w\rangle$ we construct a Turing machine M' and a fixed input w' satisfying

$$\langle M, w\rangle \in A_{\text{TM}} \iff \langle M', w'\rangle \in \text{HALT}_{\text{TM}}.$$

The construction of M' is as follows:

(a) On input w', simulate M on input w.

(b) If M accepts w, then M' halts (for example, by accepting).

(c) If M rejects w or does not halt, then M' does not halt (i.e., it loops indefinitely).

Thus, if there were a decider for HALT_{TM}, one could decide A_{TM} by computing $f(\langle M, w\rangle)$ and testing for halting, contradicting the undecidability of A_{TM}.

4.2 TM-Recognizability and Mapping Reductions

Define the language

$$\overline{E}_{\text{TM}} = \{\langle M\rangle \mid M \text{ is a Turing machine and } \mathcal{L}(M) \neq \varnothing\}.$$

Theorem 4.4. *$\overline{E}_{\text{TM}}$ is undecidable.*

Proof. We reduce A_{TM} to $\overline{E}_{\text{TM}}$. Given an instance $\langle M, w\rangle$ for A_{TM}, construct a Turing machine M_w as follows:

(a) On any input x, if $x \neq w$ then immediately reject.

(b) If $x = w$, simulate M on w.

(c) If M accepts w, then accept; if M rejects w (or loops), then do not accept.

Notice that

$$\mathcal{L}(M_w) \neq \varnothing \iff M \text{ accepts } w.$$

That is,

$$\langle M, w\rangle \in A_{\mathrm{TM}} \iff \langle M_w\rangle \in \overline{E}_{\mathrm{TM}}.$$

If $\overline{E}_{\mathrm{TM}}$ were decidable, then by computing M_w from $\langle M, w\rangle$ and applying a decider for $\overline{E}_{\mathrm{TM}}$, we could decide A_{TM}. Since A_{TM} is undecidable, it follows that $\overline{E}_{\mathrm{TM}}$ is undecidable. □

Theorem 4.5. *If $A \leq_m B$ and B is Turing-recognizable (i.e., recursively enumerable, or $B \in RE$), then A is Turing-recognizable (or $A \in RE$).*

Proof. Let f be a computable reduction from A to B and let R_B be a Turing machine that recognizes B. We build a Turing machine R_A that recognizes A as follows:

(i) On input w, compute $f(w)$.

(ii) Run R_B on $f(w)$.

(iii) If R_B accepts, then accept w; if R_B does not halt or rejects, then do not accept.

Since f is computable and R_B recognizes B, it follows that R_A recognizes A. Hence, A is Turing-recognizable. □

Corollary 4.2. *If $A \leq_m B$ and A is not Turing-recognizable ($A \notin RE$), then B is not Turing-recognizable ($B \notin RE$).*

Corollary 4.3. *If $A \leq_m B$, then $\overline{A} \leq_m \overline{B}$.*

Proof. Let f be a computable reduction from A to B, so that for every $w \in \Sigma^*$,

$$w \in A \iff f(w) \in B.$$

Taking complements, we have

$$w \notin A \iff f(w) \notin B,$$

which establishes that the same function f serves as a computable reduction from $\overline{A}$ to $\overline{B}$. □

4.3 Classes Defined via Complements

Definition 4.3. Let C be a class of languages. The class co-C is defined by

$$\mathcal{L} \in \text{co-}C \iff \overline{\mathcal{L}} \in C.$$

Example 4.1. The class co-RE is defined by

$$\text{co-}RE = \{\mathcal{L} \mid \overline{\mathcal{L}} \in RE\},$$

where RE denotes the recursively enumerable languages (i.e., Turing-recognizable languages).

Theorem 4.6. *EQ_{TM} is neither Turing-recognizable nor co-Turing-recognizable; in other words, $EQ_{\text{TM}} \notin RE$ and $\overline{EQ}_{\text{TM}} \notin RE$.*

Proof. We prove this result by contradiction using Rice's theorem and the method of mapping reducibility.
First, note that the property of two Turing machines having equal languages is non-trivial because there exist machines M_1 and M_2 such that $\mathcal{L}(M_1)$ is different from $\mathcal{L}(M_2)$, while there are also machines for which the property holds. By Rice's theorem, any non-trivial property of the language recognized by a Turing machine is undecidable. In particular, the language EQ_{TM} is undecidable.
Now suppose, for the sake of contradiction, that EQ_{TM} were Turing-recognizable. Then there would exist a Turing machine R that recognizes EQ_{TM}. Using a similar diagonalization or reduction argument, one can construct a decider for a known undecidable language (for instance, A_{TM}), thus contradicting its undecidability. A similar argument applies if one assumes that $\overline{EQ}_{\text{TM}}$ is Turing-recognizable. Hence, neither EQ_{TM} nor its complement is Turing-recognizable. □

Wrapping up undecidable problems, we have observed that the following languages

$$\begin{aligned}
A_{\text{TM}} &= \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\} \\
\text{HALT}_{\text{TM}} &= \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on } w\} \\
E_{\text{TM}} &= \{\langle M\rangle \mid M \text{ is a TM and } \mathcal{L}(M) = \varnothing\} \\
EQ_{\text{TM}} &= \{\langle M_1, M_2\rangle \mid M_1, M_2 \text{ are TM and } \mathcal{L}(M_1) = \mathcal{L}(M_2)\}
\end{aligned}$$

are unrecognizable and also $A_{\text{TM}} \leq_m \text{HALT}_{\text{TM}}$, $A_{\text{TM}} \leq_m E_{\text{TM}}$ and $E_{\text{TM}} \leq_m \overline{EQ}_{\text{TM}}$, which means A_{TM} unrecognizable $\implies \text{HALT}_{\text{TM}}, E_{\text{TM}}, EQ_{\text{TM}}$ unrecognizable.

Remark 4.1. $\mathcal{L}$ is recognizable $\iff \overline{\mathcal{L}}$ is recognizable

Lecture 5: Recognizability and Rice's Theorem

We now ask the question of which of the following languages are recognizable:

$$\mathcal{L}_1 = \{\langle M\rangle \mid M \text{ is a Turing machine and } \mathcal{L}(M) \text{ is recognizable}\},$$
$$\mathcal{L}_2 = \{\langle M\rangle \mid M \text{ is a Turing machine and } \mathcal{L}(M) \text{ is finite}\},$$
$$\mathcal{L}_{\text{odd}} = \Big\{\langle M\rangle \mid \mathcal{L}(M) \subseteq \{0,1\}^* \text{ and } \forall w \in \mathcal{L}(M),\ |w| = 2n+1,\ n \in \mathbb{N}\Big\}.$$

5.1 Rice's Theorem

Theorem 5.1 (Rice's Theorem)**.** *Let $C \subseteq RE$ be a nontrivial subset of RE (i.e., $C \neq \varnothing$ and $C \neq RE$). Define*

$$\mathcal{L}_C = \{\langle M\rangle \mid M \text{ is a Turing machine and } \mathcal{L}(M) \in C\}.$$

Then $\mathcal{L}_C$ is undecidable.

Proof. Assume without loss of generality that $\varnothing \notin C$ (if $\varnothing \in C$, one may consider the complement $\overline{\mathcal{L}}_C$). Since C is nontrivial, there exists a language $\mathcal{L}_0 \in C$ and a Turing machine M_0 such that $\mathcal{L}(M_0) = \mathcal{L}_0$.
We now define a computable reduction f from HALT_{TM} to $\mathcal{L}_C$. Given an arbitrary instance $\langle M, w\rangle$ of the halting problem, we construct a new Turing machine $M_{\langle M,w\rangle}$ such that:

$$\mathcal{L}(M_{\langle M,w\rangle}) = \begin{cases} \mathcal{L}_0, & \text{if } M \text{ accepts } w, \\ \varnothing, & \text{if } M \text{ does not accept } w. \end{cases}$$

The construction of $M_{\langle M,w\rangle}$ is as follows:

(a) On any input x, simulate M on input w.

(b) If the simulation of M on w eventually accepts, then run M_0 on x and output its result.

(c) If the simulation of M on w does not accept (i.e., it rejects or does not halt), then reject x unconditionally.

Since we have assumed that $\varnothing \notin C$, it follows that:

$$\langle M, w\rangle \in \text{HALT}_{\text{TM}} \iff \mathcal{L}(M_{\langle M,w\rangle}) = \mathcal{L}_0 \iff \langle M_{\langle M,w\rangle}\rangle \in \mathcal{L}_C.$$

Thus, if there were a decider for $\mathcal{L}_C$, one could decide HALT_{TM} by computing $f(\langle M, w\rangle) = \langle M_{\langle M,w\rangle}\rangle$ and testing its membership in $\mathcal{L}_C$, which contradicts the known undecidability of HALT_{TM}. Hence, $\mathcal{L}_C$ is undecidable. □

Example: Consider the set

$$C_0 = \Big\{ \mathcal{L} \mid \mathcal{L} \in RE,\ \mathcal{L} \subseteq \{0,1\}^*,\ \forall w \in \mathcal{L},\ w = 0u \text{ for some } u \in \{0,1\}^* \Big\}.$$

Then define

$$\mathcal{L}_0 = \{\langle M \rangle \mid M \text{ is a Turing machine and } \mathcal{L}(M) \in C_0\}.$$

To prove that $\mathcal{L}_0$ is undecidable, note that:

(i) $C_0 \subset RE$ because the property "all strings in $\mathcal{L}(M)$ start with 0" is a semantic property of M's language.

(ii) C_0 is nontrivial. For example, consider the Turing machine M_0 that accepts the language $0\{0,1\}^*$; clearly, $\mathcal{L}(M_0) \in C_0$ and $\varnothing \notin C_0$ (by our assumption or by choosing a nonempty language).

By Rice's Theorem, since C_0 is nontrivial, the language $\mathcal{L}_0$ is undecidable.

5.2 Function Version of Rice's Theorem

Theorem 5.2 (Function Version of Rice's Theorem)**.** *Let F_1 and F_2 be a nontrivial partition of the set of computable functions, meaning that F_1 and F_2 are nonempty, disjoint, and every computable function belongs to either F_1 or F_2. Then it is impossible to decide, given a Turing machine M (or an index for a computable function), whether the function computed by M belongs to F_1 or to F_2.*

Proof. Assume, for the sake of contradiction, that there exists a decision procedure D that takes as input a description $\langle M \rangle$ of a Turing machine and decides whether the function f_M computed by M is in F_1 or in F_2.
Since F_1 and F_2 form a nontrivial partition, there exists at least one computable function $g \in F_1$ and at least one computable function $h \in F_2$. We now show how to decide the Halting Problem using D.
Given an instance $\langle M, w \rangle$ of the Halting Problem, we construct a new Turing machine $M_{\langle M,w \rangle}$ that operates as follows on any input x:

(1) Simulate M on input w.

(2) If the simulation halts (i.e., M accepts or rejects w), then compute $g(x)$.

(3) If the simulation does not halt, then compute $h(x)$.

Thus, the function computed by $M_{\langle M,w\rangle}$ is exactly g if M halts on w and exactly h if M does not halt on w.
Now, by applying decision procedure D to $\langle M_{\langle M,w\rangle}\rangle$, we determine whether the computed function is in F_1 or F_2. Since $g \in F_1$ and $h \in F_2$, it follows that:

$$M \text{ halts on } w \iff f_{M_{\langle M,w\rangle}} = g \iff D(\langle M_{\langle M,w\rangle}\rangle) \text{ accepts.}$$

This procedure would then decide the Halting Problem, which is impossible. Therefore, no such decision procedure D exists. □

5.3 Modified Post Correspondence Problem (MPCP)

In the Modified Post Correspondence Problem (MPCP), one is given a collection of dominos (each domino is a pair of strings), with one domino marked to be the first. The goal is to determine whether there exists a sequence (starting with the designated domino) such that the concatenation of the top strings equals the concatenation of the bottom strings.

Example 5.1. Consider the set

$$A = \left\{ \frac{a}{ab}, \frac{ca}{a}, \frac{ac}{ab}, \frac{b}{ca}, \frac{abc}{c} \right\}.$$

A match for A exists if one can arrange the dominos (starting with the marked domino) so that the top and bottom concatenations are identical.

Theorem 5.3 (Undecidability of MPCP)**.** *The language*

$$MPCP = \{\langle P\rangle \mid P \textit{ is an instance of MPCP with a match}\}$$

is undecidable.

Proof. We prove this theorem by a reduction from A_{TM}. Given an arbitrary Turing machine M and input w, we construct a domino set P such that there is a match for P if and only if M accepts w.
The construction of P is done by encoding the computation of M on input w as a sequence of configurations. Each domino in P corresponds to a transition between consecutive configurations of M. The marked domino corresponds to the initial configuration. The design ensures that any valid match (i.e., a sequence of dominos whose top and bottom strings are equal) describes a valid computation history of M starting from the initial configuration and ending in an accepting configuration.
More concretely, the domino set P is constructed so that:

- The first domino encodes the initial configuration of M on w.
- Subsequent dominos encode the transition rules of M, allowing only valid moves between configurations.
- A domino (or a set of dominos) encodes the acceptance condition.

Thus, there exists a match (i.e., a sequence of dominos forming equal top and bottom strings) if and only if there exists a valid sequence of configurations of M on w that leads to acceptance.
Therefore, if one could decide $MPCP$, then one could decide whether M accepts w. This provides a decision procedure for A_{TM}, which is known to be undecidable. Hence, $MPCP$ is undecidable. □

Remark 5.1. Since MPCP (which requires the first domino to be fixed) is undecidable, it follows that the general Post Correspondence Problem (PCP), where the first domino is not specified, is also undecidable.

5.4 Summary of Computability

Problems can be categorized by both the nature of their output (search problems versus decision problems) and by whether they are solvable by an algorithm. In particular, we distinguish:

- **Search Problems (Functions):** Problems whose goal is to compute a function.
- **Decision Problems (Languages):** Problems whose goal is to decide membership of an input in a language.

Moreover, problems can be divided into:

- **Solvable Problems**
  - Decidable (languages)
  - Computable (functions)
- **Unsolvable Problems**
  - Non-decidable (languages)
  - Non-recognizable (languages)

The Church-Turing Thesis asserts that a problem is solvable or unsolvable regardless of the computational model used, provided the model is sufficiently powerful (i.e., Turing-complete).

Key Concepts:

- Church-Turing Thesis
- Universal Turing Machine
- Halting Problem
- Reduction

Tools:

- Diagonalization
- Direct Reduction
- Mapping Reducibility
- Rice's Theorem

Lecture 6: Time Complexity and Efficiency of Algorithms

A *Turing machine*, defined as in Definition 1.5, is as a machine with an infinite tape, a tape head, and a finite set of states, and is said that it decides a language $\mathcal{L}$ following Definition 2.1.
Now we turn to ask the question:

> "Can every solvable problem be solved efficiently?"

A problem that is solvable in principle may not be solvable in practice because the amount of time or space required might be unreasonable.

6.1 Types of Measured Complexity

There are two main types of complexity analysis:

- **Worst-case analysis:** The largest running time over all inputs of length n.
- **Average-case analysis:** The average running time over all inputs of length n.

Definition 6.1 (Running Time)**.** Let M be a Turing machine that halts on all inputs. The *running time* (or *time complexity*) of M is a function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is defined as the maximum number of steps that M takes on any input of length n.

Because the exact running time of an algorithm is often complex, we use *asymptotic analysis* (or *big-$\mathcal{O}$ analysis*) to approximate its behavior.

Definition 6.2 (Big-$\mathcal{O}$ Notation)**.** Let $f, g : \mathbb{N} \to \mathbb{R}^+$. We say that

$$f(n) = \mathcal{O}\big(g(n)\big)$$

if there exist constants $c > 0$ and $n_0 \in \mathbb{N}$ such that

$$\forall n \geq n_0, \quad f(n) \leq c\, g(n).$$

We say that $g(n)$ is an *asymptotic upper bound* for $f(n)$.

Remark 6.1. In Big-$\mathcal{O}$ notation, the bigger term in a function dominates, for instance,

$$f(n) = \mathcal{O}(n^2) + \mathcal{O}(n) \equiv \mathcal{O}(n^2),$$

or

$$f(n) = \mathcal{O}(2^n) + \mathcal{O}(n^3) \equiv \mathcal{O}(2^n).$$

Remark 6.2. Since $\log_b n = \frac{\log_2 n}{\log_2 b}$, when writing in Big-$\mathcal{O}$ notation, $f(n) = \mathcal{O}(\log n)$ does not require specifying a logarithm base.

Definition 6.3 (Types of Bounds)**.** A function $f : \mathbb{N} \to \mathbb{R}^+$ is said to be:

(a) *Linearly bounded* if $f(n) = \mathcal{O}(n)$.

(b) *Polynomially bounded* if $f(n) = \mathcal{O}(n^c)$ for some constant $c > 1$.

(c) *Quasi-polynomially bounded* if $f(n) = \mathcal{O}(n^{\log^k n})$ for some constant $k > 0$.

(d) *Sub-exponentially bounded* if $f(n) = \mathcal{O}(2^{n^\varepsilon})$ for some constant $0 < \varepsilon < 1$.

(e) *Exponentially bounded* if $f(n) = \mathcal{O}(2^{n^c})$ for some constant $c > 0$.

(f) *Super-exponentially bounded* if $f(n) = \mathcal{O}(n^n)$ (or more generally, if $f(n)$ grows faster than any exponential 2^{n^c} for all $c > 0$, but slower than any function of the form n^{n^d} for all $d > 0$).

(g) *Factorially bounded* if $f(n) = \mathcal{O}(n!)$, with the asymptotic formula $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$.

(h) *Double-exponentially bounded* if $f(n) = \mathcal{O}(2^{2^{n^c}})$ for some constant $c > 0$.

(i) *Non-elementarily bounded* if $f(n)$ grows faster than any finite tower of exponentials of n (for example, $f(n) = \mathcal{O}(A(n, n))$, where A denotes the Ackermann function).

Definition 6.4 (Little-o Notation)**.** Let $f, g : \mathbb{N} \to \mathbb{R}^+$. We say that

$$f(n) = o\big(g(n)\big)$$

if

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

Remark 6.3. In particular, $f(n)$ is never $o\big(f(n)\big)$.

Definition 6.5 (Time Complexity Classes)**.** Let $t : \mathbb{N} \to \mathbb{R}^+$ be a function. The time complexity class

$$\mathrm{TIME}\big(t(n)\big)$$

is defined as the collection of all languages that are decidable by a Turing machine that runs in time $\mathcal{O}(t(n))$.

Now, take the language

Example 6.1. $A = \{0^k 1^k \mid k \geq 0\}$.

Consider the Turing machine M_1 that, given an input string w, performs the following:

(1) Scan the tape and *reject* if a 0 is found to the right of a 1.

(2) Scan across the tape, crossing off one 0 and one 1.

(3) Repeat step (2) until either all 0s or all 1s are exhausted.

(4) *Reject* if there remain unmatched symbols, and *accept* otherwise.

Let n be the length of the input string w. The running time of M_1 can be analyzed as follows:

- Step (1): A full scan of the tape requires $\mathcal{O}(n)$ steps to go through the tape and an additional $\mathcal{O}(n)$ steps to return back, for a total of $\mathcal{O}(n)$ steps.
- Steps (2) and (3): In each round, one 0 and one 1 are crossed off. In the worst case, there are $\frac{n}{2}$ rounds, and each round requires scanning $\mathcal{O}(n)$ steps, for a total of $\mathcal{O}(n) \cdot \frac{n}{2} = \mathcal{O}(n^2)$.
- Step (4): A final scan of the tape takes $\mathcal{O}(n)$ steps.

Thus, the overall runtime of M_1 is:

$$\mathcal{O}(n) + \mathcal{O}(n^2) + \mathcal{O}(n) = \mathcal{O}(n^2).$$

This implies that $A \in \text{TIME}(n^2)$.

Remark 6.4. It is possible to improve the time complexity for deciding A. In fact, one may show that $A \in \text{TIME}(n \log n)$.

However, it turns out that this is asymptotically optimal for a single-tape Turing machine in the following sense:

Theorem 6.1. *Any language that is decidable in time $o(n \log n)$ on a single-tape Turing machine is regular.*

Proof. Let M be a deterministic single-tape Turing machine that decides a language $\mathcal{L}$ in time $T(n) = o(n \log n)$; that is, for any input w of length n, M halts in at most $T(n)$ moves, where

$$\lim_{n \to \infty} \frac{T(n)}{n \log n} = 0.$$

Let Q denote the finite set of states of M.

When M runs on an input w of length n, consider the tape as divided into cells indexed by $1, 2, \ldots, n$. In addition, consider the *boundaries* between the cells, indexed $0, 1, \ldots, n$ (where boundary 0 lies just before the first cell and boundary n just after the last cell). Each time M moves its head from one cell to an adjacent cell, it *crosses* one of these boundaries. Define the *crossing sequence* at a given boundary as the ordered sequence of states in which M enters immediately after crossing that boundary (one may record a state for each crossing in either direction, though the precise convention does not affect the argument).

Since M takes at most $T(n)$ steps on input w, each crossing sequence has length at most $T(n)$ and each element of the sequence is one of the finitely many states in Q. Hence, the total number of possible distinct crossing sequences is at most

$$|Q|^{T(n)}.$$

Now, because $T(n) = o(n \log n)$, for any fixed constant $\varepsilon > 0$ there exists an n_0 such that for all $n \geq n_0$

$$T(n) < \varepsilon \log n.$$

Thus, the number of distinct crossing sequences is bounded by

$$|Q|^{\varepsilon \log n} = n^{\varepsilon \log |Q|}.$$

Since $\varepsilon > 0$ can be chosen arbitrarily small, for sufficiently large n the number of distinct crossing sequences is $o(n)$.

On any input of length n there are $n + 1$ boundaries. By the pigeonhole principle, if $n + 1$ is larger than the number of possible crossing sequences, then there exist two distinct boundaries, say at positions i and j (with $0 \leq i < j \leq n$), that have the identical crossing sequence.

The significance of two identical crossing sequences is as follows. Intuitively, the crossing sequence at a given boundary summarizes the *interaction* of M with the portion of the tape to one side of that boundary. If two boundaries have identical crossing sequences, then the manner in which M "communicates" between the regions on either side of these boundaries is identical. In particular, if one were to *pump* (i.e., repeat or remove) the segment of the tape between these two boundaries, M's behavior could be simulated unchanged.

One formalizes this argument by showing that the existence of two boundaries with the same crossing sequence implies a *pumping lemma* for $\mathcal{L}$. In fact, one can construct a finite automaton whose states represent the finitely many possible crossing sequences. This automaton can simulate M's behavior on the regions of the tape delimited by boundaries, showing that the behavior of M on long inputs is essentially that of a finite automaton. Hence, the language $\mathcal{L}$ decided by M is regular.

Thus, any language decided in time $T(n) = o(n \log n)$ on a single-tape Turing machine must be regular. □

Using alternative models of computation can affect running times. For instance, a k-tape Turing machine may decide certain problems more efficiently than a single-tape Turing machine, for example a 2-tape TM can decide A in $\mathcal{O}(n)$ time.

Theorem 6.2. *Let $t(n)$ be any function with $t(n) \geq n$. Then every language decided in time $t(n)$ by a k-tape Turing machine has an equivalent single-tape Turing machine deciding the language in time $\mathcal{O}\big(t(n)^k\big)$.*

Proof. Let M be a k-tape Turing machine that decides a language $\mathcal{L}$ in time $t(n)$ on inputs of length n (with the assumption that $t(n) \geq n$). We will construct a single-tape Turing machine S that simulates M and decides $\mathcal{L}$ in time $\mathcal{O}\big(t(n)^k\big)$.
The simulation is based on encoding the entire configuration of M on a single tape. Recall that the configuration of a k-tape Turing machine consists of the contents of each tape, the positions of the tape heads, and the current state of the machine. We encode this information on a single tape as follows:

$$\# \, w_1 \, \# \, w_2 \, \# \, \cdots \, \# \, w_k \, \#$$

where each w_i represents the contents of tape i (including a special mark that indicates the head position on that tape). The symbol $\#$ is used as a delimiter to separate the tapes. Since M runs in time $t(n)$, each tape can have at most $\mathcal{O}(t(n))$ nonblank symbols (beyond what is initially given), so the total length of the encoded configuration is $\mathcal{O}(k \cdot t(n))$; with k being fixed, this length is $\mathcal{O}(t(n))$.
To simulate one move of M, the single-tape machine S must perform the following tasks:

(a) *Reading the Current Configuration:* S scans its tape from left to right to read the encoded configuration. In doing so, it locates the delimiters and identifies the symbols that are under each tape head (by checking for the special markers embedded in each w_i).

(b) *Determining the Next Move:* Using the current state of M and the k symbols read from the respective tape head positions, S consults M's transition function (which is fixed and hard-coded in S) to determine the new state, the new symbols to write, and the directions in which each head should move.

(c) *Updating the Configuration:* S then makes another pass over the tape to update the encoding. This involves modifying the symbols and shifting the head markers as prescribed by the transition function. In a straightforward simulation, S might update the parts corresponding to each tape one at a time.

Each complete simulation step (steps (a) through (c)) involves scanning over the entire encoded configuration, which has length $\mathcal{O}(t(n))$. In a simulation procedure

where the update for each of the k tapes is handled in a separate pass, the time for one simulated move is $\mathcal{O}(t(n))$ for reading plus $\mathcal{O}(t(n))$ per tape for updating—that is, roughly $\mathcal{O}(k \cdot t(n))$. In the worst-case analysis for this simulation strategy, one can bound the time for one move by $\mathcal{O}(t(n))$ multiplied by a factor that depends on k, which, when compounded over the $t(n)$ moves, yields a total runtime in $\mathcal{O}\big(t(n)^k\big)$. Since M runs in time $t(n)$, it makes at most $t(n)$ moves. Simulating each move costs at most $\mathcal{O}(t(n))$ time for each of the k tapes (using a separate full-scan per tape update, if done sequentially). Thus, the overall simulation time is bounded by

$$t(n) \cdot \mathcal{O}\big(t(n)^{k-1}\big) = \mathcal{O}\big(t(n)^k\big).$$

Thus, we have constructed a single-tape Turing machine S that simulates the k-tape machine M in polynomial time with respect to $t(n)$, specifically $\mathcal{O}\big(t(n)^k\big)$. □

Proposition 6.1. *All reasonable deterministic computational models are* polynomially equivalent *(they differ only by a polynomial factor in running time).*

Thus, from now on, we focus on complexity measures that are invariant under polynomial-time reductions.

6.2 The Class P

Definition 6.6 (Class P (or PTIME))**.** The class P is defined as

$$P = \bigcup_{k \in \mathbb{N}} \mathrm{TIME}(n^k),$$

i.e. the set of languages that are decidable in polynomial time on a deterministic single-tape Turing machine.

The class P is considered important because:

- It is invariant for all models of computation that are polynomially equivalent.
- It roughly corresponds to the class of problems that are realistically solvable on a computer.

6.3 Input Representation and Its Effect on Complexity

The manner in which input is represented can also significantly affect the running time of an algorithm. For instance,

Example 6.2. Consider the problem of factoring an integer m. In the natural representation, one might expect $\sqrt{m}$ steps for trial division. However, if m is represented in binary, its length is $n = \log m$, and the algorithm then takes exponential time in n.

or

Example 6.3. Given a directed graph $\mathcal{G}$ and two nodes s and t, the problem is to decide whether there is a directed path from s to t. Formally, we define:

$$\text{PATH} = \{\langle \mathcal{G}, s, t \rangle \mid \mathcal{G} \text{ is a directed graph with a directed path from } s \text{ to } t\}.$$

If $\mathcal{G}$ has m nodes, a brute-force search may require examining as many as m^m potential paths, resulting in super-exponential complexity.

Proposition 6.2. PATH $\in P$.

Proof. The *Breadth-First Search* (BFS) algorithm decides the PATH problem in time that is linear in the size of the graph (i.e., $\mathcal{O}(m + n)$, where m is the number of edges and n the number of nodes). Therefore, the PATH problem can be decided in polynomial time. □

Remark 6.5. From now on, the class P will only be denoted as PTIME.

Lecture 7: Nondeterministic Computation and Complexity Classes

Recall from Definition 1.6 that a nondeterministic Turing machine (NTM) *accepts* an input if there is at least one branch of computation that leads to the *accept* state. The transition function for an NTM is defined as

$$\delta : Q \times \Gamma \to \mathcal{P}\Big(Q \times \Gamma \times \{L, R\}\Big),$$

so that for each state and tape symbol, the machine may move to a set of possible next states, writing a symbol and moving left or right.
To organize these computations, we view them as a *computation tree* where:

- Each *node* represents a configuration of the Turing machine.
- An *edge* between two nodes represents a single move (or transition) that the machine makes according to the transition function δ.
- A *branch* represents one possible sequence of moves (or a computation path) beginning at the initial configuration.

Below is a minimal visual example illustrating a computation tree.

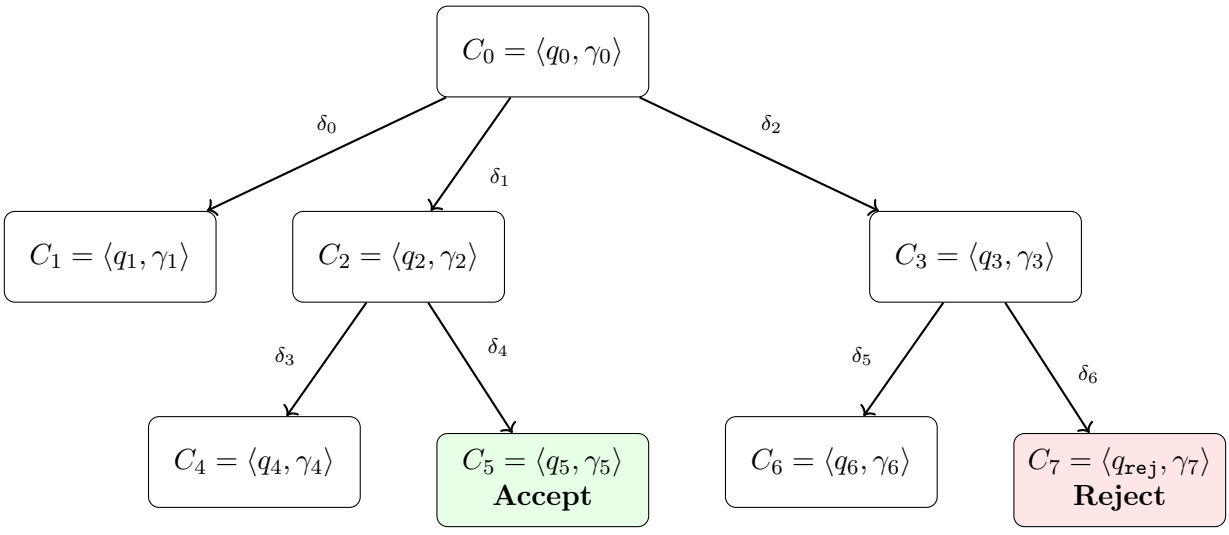


Figure 1: Computation tree of a nondeterministic Turing machine on input γ_0 starting from the initial state q_0. Each node represents a configuration $C_i = \langle q_i, \gamma_i \rangle$.

In a computation tree for an NTM, the overall result is determined by whether there exists at least one accepting branch. Even if some branches result in rejection or fail to halt, the NTM accepts the input if any computation branch reaches an `accept` state.

7.1 Equivalence and Time Complexity of Nondeterministic Turing Machines

Theorem 7.1. *Every nondeterministic Turing machine (NTM) has an equivalent deterministic Turing machine (TM).*

Proof. We prove the equivalence in two parts.

($\Leftarrow$ Any deterministic Turing machine (DTM) is a special case of an NTM, where the transition function always maps to a singleton set.

To simulate an NTM N with a deterministic Turing machine D, we construct D so that it systematically explores all branches of N's computation tree. A standard approach is to perform a Breadth-First Search (BFS) on the computation tree. Although each node has finitely many children, some branches may be infinitely long; BFS ensures that every node at finite depth is eventually reached.

A common simulation uses a multi-tape Turing machine. In our case, we design a 3-tape TM D with the following tapes:

(i) *Input tape*: Contains the input string w.

(ii) *Simulation tape*: Maintains a copy of N's tape for the branch currently being simulated.

(iii) *Address tape*: Encodes the lexicographic "address" of a branch in N's computation tree, which uniquely identifies the nondeterministic choices made.

The simulation proceeds as follows:

(i) Initialize tape 1 with w; tapes 2 and 3 are initially empty.

(ii) Copy the input from tape 1 to tape 2 to begin simulating N on w.

(iii) Use the string on tape 3 to guide the simulation along one branch in the computation tree. In each step, consult the next symbol on tape 3 to decide which nondeterministic move to simulate.

(iv) If tape 3 is exhausted, or if the nondeterministic choice indicated by tape 3 is invalid, or if the current simulation leads to a reject configuration, then update tape 3 to the lexicographically next string (thus exploring the next branch) and restart the simulation from the initial configuration using the new branch identifier.

(v) If any branch reaches an `accept` configuration, then D accepts.

This simulation ensures that if N has any accepting branch, D will eventually find it. Hence, D is an equivalent deterministic simulation of N.

We have thus proven both directions of the statement, completing the proof. □

Definition 7.1 (Time Complexity of an NTM)**.** Let N be an NTM. The *running time* of N is given by a function $f : \mathbb{N} \to \mathbb{N}$ where $f(n)$ is the maximum number of steps that N performs on any computation branch for any input of length n.

Theorem 7.2. *Let $t(n)$ be a function with $t(n) \geq n$. Then every single-tape NTM that runs in time $t(n)$ has an equivalent deterministic single-tape TM that runs in time $2^{\mathcal{O}(t(n))}$.*

Proof. Let N be a single-tape nondeterministic Turing machine (NTM) that, on any input x of length n, runs in time at most $t(n)$ (with $t(n) \geq n$). This means that every computation path on input x comprises at most $t(n)$ moves.
A configuration of N is determined by:

(i) The current state (which is one of the finitely many states in the state set Q).

(ii) The contents of the tape. In a single-tape machine, although the tape is infinite, the machine can only access and modify cells that are within a distance proportional to the number of moves. Since N runs in at most $t(n)$ steps, it can only access at most $\mathcal{O}(t(n))$ tape cells.

(iii) The position of the head, which can be anywhere within the portion of the tape that has been used; this is at most on the order of $t(n)$.

Since the tape alphabet Γ is finite and the number of states $|Q|$ is finite, the number of possible distinct configurations that N can enter in a computation of length at most $t(n)$ is bounded by

$$|Q| \cdot |\Gamma|^{\mathcal{O}(t(n))} \cdot \mathcal{O}(t(n)) \leq 2^{\mathcal{O}(t(n))}.$$

Thus, although N is nondeterministic (and can explore many branches), the total number of distinct configurations is at most exponential in $t(n)$.

We construct a deterministic Turing machine D that simulates N by performing a breadth-first search (BFS) of the computation tree of N. The idea is to systematically explore all possible computation branches of N up to $t(n)$ steps.
The simulation proceeds as follows:

(i) **Initialization.** Construct the initial configuration C_0 of N on input x. Place C_0 in a work area (or list) that will hold configurations to be explored.

(ii) **Breadth-First Exploration.** For each level from 0 to $t(n)$:

  (a) For every configuration C at the current level, use the transition function δ to compute all configurations C' that are reachable from C in one move.

  (b) Add each newly generated configuration C' to a list (or queue) of configurations for the next level, but do not add duplicates.

(iii) **Termination.** During the simulation, if any configuration is an `accept` configuration, then D accepts x. Otherwise, if all configurations at level $t(n)$ have been explored without encountering an accepting configuration, D rejects x.

Because the total number of configurations is bounded by $2^{\mathcal{O}(t(n))}$ and each configuration is generated in at most polynomial time in its description, the overall time taken by D is bounded by $2^{\mathcal{O}(t(n))}$. In particular, at each level the number of configurations is at most exponential in $t(n)$, and the cost to update each configuration is polynomial. Therefore, the total simulation time is $2^{\mathcal{O}(t(n))}$. □

Remark 7.1 (Is $P = NP$?)**.** This result shows an exponential overhead in the worst-case simulation of nondeterminism by determinism on a single tape. A natural question arises: can we simulate an NTM on a deterministic TM with only a polynomial slowdown? Despite extensive research, no general polynomial-time simulation is known, and this gap underpins the famous P versus NP problem.

7.2 Decision Problems, Verifiers and the Class NP

Given two nodes s and t in a directed graph G, a basic decision problem asks: *Is there a directed path from s to t?* This problem is denoted as PATH and is known to be decidable in polynomial time.
An important extension is the Hamiltonian Path problem:

$$\text{HAMPATH} = \{\langle G, s, t\rangle \mid G \text{ is a directed graph that has a Hamiltonian path from } s \text{ to } t\},$$

where a *Hamiltonian path* is a path that visits every node in G exactly once.

When G has m nodes, a brute-force search for a Hamiltonian path has a worst-case time complexity of $\Theta(m^m)$, making it super-exponential in the size of G. While finding a Hamiltonian path is computationally hard, if a candidate Hamiltonian path is provided, one can verify its correctness in polynomial time.
On the other hand, not all problems are polynomially verifiable. For example, the verification complexity of $\overline{\text{HAMPATH}}$ (the complement of HAMPATH, asking whether there is no Hamiltonian path from s to t) is an open problem.

Definition 7.2 (Verifier for a Language)**.** A *verifier* for a language A is an algorithm V such that

$$A = \{w \mid \exists c \text{ a certificate (or proof) such that } V \text{ accepts } \langle w, c\rangle\},$$

with the running time of V measured in terms of $|w|$. The certificate c is a string that serves as evidence for the membership of w in A. A language is said to be *polynomially verifiable* if it has a verifier V that runs in polynomial time.

Definition 7.3 (NP Class)**.** NP is the class of languages that have a polynomial-time verifier. Equivalently,

$$NP = \Big\{A \mid \exists V \text{ a polynomial-time verifier} : A = \{w \mid \exists c, V(\langle w, c\rangle) \text{ accepts}\}\Big\}.$$

NP is central not only because it contains many practical problems but also because it encapsulates the notion of efficiently verifiable proofs. It is also directly related to the long-standing open question: *Is $P = NP$?*

7.3 Equivalence of NP and NTM Polynomial-Time Computability

Theorem 7.3 (NP Class Membership)**.** *A language A is in NP if and only if it can be decided by some nondeterministic Turing machine (NTM) in polynomial time.*

Proof. We prove the theorem in two directions.

($\Rightarrow$) Suppose $A \in NP$ and let V be a polynomial-time verifier for A. We construct an NTM N that decides A in polynomial time as follows:

(i) On input w of length n, nondeterministically guess a certificate c of length at most n^k for some constant k.

(ii) Run the verifier V on the input $\langle w, c\rangle$.

(iii) Accept if V accepts; otherwise, reject.

Since V runs in polynomial time, the nondeterministic machine N also runs in polynomial time, and N accepts w if and only if $w \in A$.

($\Leftarrow$) Conversely, suppose that A is decided by an NTM N in polynomial time. We now construct a polynomial-time verifier V for A. The idea is that the certificate c will encode the sequence of nondeterministic choices that lead N to an *accept* state. On input $\langle w, c \rangle$, the verifier V simulates N on w using the choices indicated by c:

(i) Interpret c as a description of the nondeterministic choices in N's computation path.

(ii) Simulate N on input w following the branch specified by c.

(iii) Accept if the simulated computation reaches an *accept* state; otherwise, reject.

Because N is a polynomial-time machine, the simulation by V is also polynomial-time. Thus, V is a polynomial-time verifier for A.

These two directions complete the proof. □

Corollary 7.1. *Any language in NP can be decided by a deterministic Turing machine in exponential time.*

This follows from the fact that an NTM running in polynomial time can be simulated deterministically with an exponential overhead in the worst case.

Definition 7.4 (NTIME)**.** Let $t : \mathbb{N} \to \mathbb{R}^+$ be a function. The complexity class $\text{NTIME}(t(n))$ is defined as

$$\text{NTIME}(t(n)) = \Big\{ \mathcal{L} \;\Big|\; \mathcal{L} \text{ is decidable by an NTM in } \mathcal{O}\big(t(n)\big) \text{ time} \Big\}.$$

Definition 7.5 (Alternative Definition of NP)**.** An alternative definition of NP is given by

$$NP = \bigcup_{k \geq 1} \text{NTIME}(n^k),$$

which is equivalent to the class of languages decidable by a nondeterministic Turing machine in polynomial time.

Definition 7.6 (co-NP)**.** A language $\mathcal{L}$ is in co-NP if and only if its complement $\overline{\mathcal{L}}$ is in NP. That is,

$$\mathcal{L} \in \text{co-}NP \iff \overline{\mathcal{L}} \in NP.$$

It is currently an open problem whether $NP = \text{co-}NP$ or if one of the inclusions is strict.
Several scenarios are conceivable regarding the relationships among P, NP, and co-NP:

- If $P = NP$, then it follows that $P = NP = \text{co-}NP$.
- It is possible that $NP = \text{co-}NP$ while $P \neq NP$.
- Alternatively, one might have $NP \neq \text{co-}NP$ with the strict inclusions $P \subseteq NP$ and $P \subseteq \text{co-}NP$.
- Another possibility is that $NP \neq \text{co-}NP$, and further $P \neq NP$ and $P \neq \text{co-}NP$.

Lecture 8–9: NP-Completeness and Polynomial-Time Reductions

Recall that we gave two definitions for the NP class, namely Definition 7.3 and Definiton 7.5.

Theorem 8.1. *Definitions 7.3 and 7.5 are equivalent.*

Proof. We demonstrate both directions of the theorem.

($\Rightarrow$) Given a verifier V and a certificate length bound $p(n)$ from Definition 7.3, build an NTM N that at input w non-deterministically guesses a string c of length $\leq p(|w|)$ in $p(|w|)$ steps, then deterministically runs V on $\langle w, c\rangle$ in $\mathcal{O}(p(|w|))$, accepting exactly when V does. Thus, N runs in polynomial time and accepts w iff V accepts some certificate.

($\Leftarrow$) Given an NTM N from Definition 7.5, with a time bound $q(n)$, every computation that accepts input w can be described by the sequence of nondeterministic choices (a string c of length at most $q(|w|)$). Define a verifier V that on input $\langle w, c\rangle$ simulates N on w, following exactly the choices encoded by c, and accepts if and only if the branch halts in *accept*. This simulation takes polynomial time. Hence V is a polynomial-time verifier for A.

□

In their result, Cook and Levin showed that some problems in NP are *at least as hard* as all other problems in NP. These are the $NP-$complete problems.
Many natural problems in NP are either in P or $NP-$complete; a few remain of unknown status.

8.1 The Cook-Levin Theorem

Definition 8.1 (SAT)**.**

$$\text{SAT} = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula over } n \text{ variables}\}.$$

Proposition 8.1. $\text{SAT} \in NP$.

Proof. Given ϕ on n variables, a certificate is an assignment $a \in \{0,1\}^n$. The verifier evaluates $\phi(a)$ in time $\mathcal{O}(n + |\phi|)$ and accepts iff $\phi(a) = \text{TRUE}$. Hence SAT has a poly-time verifier. □

Theorem 8.2 (Cook–Levin)**.** SAT *is* $NP-$complete. *In particular,*

$$P = NP \iff \text{SAT} \in P.$$

Proof. Let $N = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ be a nondeterministic Turing machine and let $w \in \Sigma^*$ be an input of length n. Suppose N runs in time at most $T = p(n)$ for some polynomial p. We will in polynomial time construct a Boolean formula

$$\phi_{N,w}$$

that is satisfiable if and only if N has an accepting computation on w within T steps. The construction is as follows.
We introduce two kinds of Boolean variables:

- $X_{t,i,a}$ for $t = 0, 1, \ldots, T$, $i = 1, 2, \ldots, T$, and $a \in \Gamma$. Intuitively, $X_{t,i,a} = 1$ means "at time t, the tape cell i contains the symbol a."
- $H_{t,i,q}$ for $t = 0, 1, \ldots, T$, $i = 1, 2, \ldots, T$, and $q \in Q$. Intuitively, $H_{t,i,q} = 1$ means "at time t, the head is scanning cell i and the machine is in state q."

There are $\mathcal{O}\big(T^2\,(|\Gamma| + |Q|)\big)$ variables overall, which is polynomial in T.
For each t and i:

$$\big(X_{t,i,a_1} \vee X_{t,i,a_2} \vee \cdots \vee X_{t,i,a_{|\Gamma|}}\big) \quad \text{and} \quad \big(\neg X_{t,i,a} \vee \neg X_{t,i,b}\big) \quad \text{for every } a \neq b \in \Gamma.$$

This ensures that at each time and cell exactly one tape symbol holds.
For each t:

$$\big(H_{t,1,q_1} \vee H_{t,1,q_2} \vee \cdots \vee H_{t,T,q_{|Q|}}\big) \quad \text{and} \quad \big(\neg H_{t,i,q} \vee \neg H_{t,j,r}\big) \quad \text{for every distinct pairs } (i,q) \neq (j,r).$$

Thus at each time step, the head is in exactly one cell and one state.
At time $t = 0$, we assert:

- For $i = 1, \ldots, n$, X_{0,i,w_i} holds, and for $i = n+1, \ldots, T$, $X_{0,i,}$ holds.
- $H_{0,1,q_0}$ holds.

These are simple unit clauses (or small conjunctions) setting up the tape to contain w and positioning the head in the start state at cell 1.
For each time $t = 0, \ldots, T-1$, each cell $i = 1, \ldots, T$, each state $q \in Q$, and each tape symbol $a \in \Gamma$, consider all transitions

$$(q, a) \xrightarrow{\delta} (q', a', D),$$

where $D \in \{L, R\}$. We enforce that if at time t the head is at cell i in state q and cell i contains a, then at time $t+1$:

- Cell i contains a'.
- The head is at cell $i-1$ in state q' if $D = L$, or at cell $i+1$ in state q' if $D = R$.
- All other cells $j \neq i$ keep their symbol: for each $b \in \Gamma$,

$$\big(X_{t,i,b} \implies X_{t+1,i,b}\big) \quad \text{if } j \neq i.$$

Each such implication can be written in CNF by rewriting

$$(A \wedge B) \implies C \quad \text{as} \quad \neg A \vee \neg B \vee C,$$

and similarly for the "all other cells unchanged" conditions. Since there are $\mathcal{O}(T \cdot |\Gamma| \cdot |Q|)$ transitions and each yields a constant number of clauses over t steps, this set of clauses is polynomial in T.
Finally, we assert that *some* time step is accepting:

$$\big(H_{0,1,q_{\text{acc}}} \vee H_{1,1,q_{\text{acc}}} \vee \cdots \vee H_{T,T,q_{\text{acc}}}\big).$$

(We allow the head to be in any cell when it enters the accept state.)
Let $\phi_{N,w}$ be the conjunction of all the above clauses. By construction:

$$\phi_{N,w} \text{ is satisfiable} \iff \text{there exists an assignment to the variables}$$

encoding a valid tableau of N's computation on w of length at most T that reaches q_{acc}.
That, in turn, holds if and only if N has an accepting computation on w within T steps.
Finally, the number of variables and clauses is $\mathcal{O}(T^2)$, and they can all be generated in time polynomial in T. Since $T = p(n)$ is polynomial in $n = |w|$, the reduction

$$\langle N, w\rangle \mapsto \phi_{N,w}$$

is computable in polynomial time. Hence SAT is NP-hard. Combining with SAT $\in NP$ completes the proof that SAT is NP-complete, and establishes SAT $\in P \iff P = NP$. □

8.2 Polynomial-Time Reducibility

Definition 8.2 (Polynomial-Time (Mapping) Reducibility)**.** A language A is *polynomial-time mapping reducible* to a language B, written

$$A \leq_p B,$$

if there exists a function $f : \Sigma^* \to \Sigma^*$ computable in time polynomial in $|w|$ such that

$$\forall\, w, \quad w \in A \iff f(w) \in B.$$

Theorem 8.3. *If $A \leq_p B$ and $B \in P$, then $A \in P$.*

Proof. Let f be the poly-time reduction and let M_B decide B in time $\mathcal{O}(n^k)$. To decide A on input w, compute $f(w)$ in polynomial time, then run M_B on $f(w)$. The total time remains polynomial, so $A \in P$. □

Corollary 8.1. *If $A \leq_p B$ and $A \notin P$, then $B \notin P$.*

Definition 8.3 (CLIQUE)**.**

$$\text{CLIQUE} = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a clique of size } k\}.$$

Theorem 8.4. 3SAT $\leq_p$ CLIQUE.

Proof. Let $F = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be a 3-CNF formula with clauses $C_i = (\ell_{i1} \vee \ell_{i2} \vee \ell_{i3})$. Construct graph G as follows:

(i) For each clause C_i, create three vertices v_{i1}, v_{i2}, v_{i3} labeled by the literals $\ell_{i1}, \ell_{i2}, \ell_{i3}$.

(ii) For any two vertices v_{ij} and $v_{k\ell}$ with $i \neq k$, add an edge *iff* their labels are not complementary literals (i.e. not x vs. $\neg x$).

(iii) Set $k = m$.

Correctness:

- If F is satisfiable, pick one true literal ℓ_{ij_i} from each clause C_i. The corresponding vertices $\{v_{ij_i}\}_{i=1}^{m}$ form a clique of size m because no pair is contradictory.
- Conversely, if G has a clique of size m, it must select exactly one vertex in each clause-gadget (otherwise two vertices from the same clause are non-adjacent). Assign each chosen literal to be true; this yields a satisfying assignment for F.

The construction takes time $\mathcal{O}(m^2)$, hence is a polynomial-time reduction. □

8.3 NP-Completeness and NP-Hardness

Definition 8.4 (NP-Completeness)**.** A language B is $NP-$complete if

$$B \in NP \quad \text{and} \quad \forall A \in NP,\ A \leq_p B.$$

Definition 8.5 (NP-Hardness)**.** A language H is NP-hard if

$$\forall A \in NP,\ A \leq_p H.$$

Equivalently, H is NP-hard iff some NP-complete problem reduces to H in polynomial time.

Theorem 8.5. *If B is NP-complete and $B \in P$, then $P = NP$.*

Proof. Since every $A \in NP$ reduces to B in polynomial time and B is decidable in polynomial time, by transitivity of poly-time reducibility every $A \in NP$ is in P. Hence $P = NP$. □

Theorem 8.6. *If B is NP-complete and $B \leq_p C$ for some $C \in NP$, then C is NP-complete.*

Proof. We have $C \in NP$ by assumption. For any $A \in NP$, since $A \leq_p B$ and $B \leq_p C$, by composing the reductions we get $A \leq_p C$. Thus C is NP-complete. □

Definition 8.6 (Bounded-Halting)**.**

$$\text{BH} = \left\{ \langle M, x, 1^t \rangle \mid M \text{ is a TM that halts and accepts } x \text{ within } t \text{ steps} \right\}.$$

Often abbreviated as $\langle M, x \rangle$ with the understanding that $t = \text{poly}(|x|)$.

Theorem 8.7. BH *is NP-complete.*

Proof. Given $\langle M, x, 1^t \rangle$, a certificate is the accepting computation history of M on x, encoded as a sequence of configurations of length $\leq t$. A verifier can check in time polynomial in $|x| + t$ that each configuration legally follows from the previous one and that the final configuration is accepting.
Let $A \in NP$ with verifier V and polynomial $p(n)$. On input w, build the instance

$$f(w) \;=\; \left\langle V,\; w,\; 1^{p(|w|)} \right\rangle.$$

Clearly f is computable in polynomial time. Moreover,

$$w \in A \iff \exists c : V \text{ accepts } \langle w, c \rangle \text{ in } p(|w|) \text{ steps} \iff \left\langle V, w, 1^{p(|w|)} \right\rangle \in \text{BH}.$$

Thus $A \leq_p$ BH, proving NP-hardness. □

Theorem 8.8 (CLIQUE is NP-Complete)**.** CLIQUE $\in NP$, *and since* 3SAT $\leq_p$ CLIQUE*, it is NP-hard. Hence* CLIQUE *is NP-complete.*

Theorem 8.9 (Vertex Cover is NP-Complete)**.** *Define*

$$\text{VC} = \{ \langle G, k \rangle \mid G \text{ is an undirected graph with a vertex-cover of size } k \}.$$

Then VC $\in NP$*, and one shows NP-hardness by reducing* CLIQUE *to* VC *via graph complementation:*

$$G \text{ has a clique of size } k \iff \overline{G} \text{ has a vertex-cover of size } |V| - k.$$

Hence VC *is NP-complete.*

Remark 8.1. To show a new problem B is NP-complete, it suffices to give a polynomial-time reduction from any known NP-complete problem to B.

Remark 8.2. If *any* NP-complete problem lies in P, then $P = NP$, and every problem in NP can be solved in polynomial time.

Theorem 8.10. HAMPATH *is* NP*-complete.*

Proof. We must show:

(a) HAMPATH $\in NP$.

(b) HAMPATH is NP-hard.

A nondeterministic polynomial-time verifier V for HAMPATH works as follows on input $\langle G, s, t\rangle$ and certificate c: c is a purported listing of all vertices of G in the order they are visited. V checks in $\mathcal{O}(|V| + |E|)$ time that:

- c is a sequence of length $|V|$ containing each vertex exactly once,
- the first entry is s, the last is t,
- for each consecutive pair (u, v) in c, there is a directed edge $u \to v$ in G.

If all checks pass, V accepts; otherwise it rejects. Thus HAMPATH $\in NP$.
We reduce from **3SAT**, the canonical NP-complete problem:

$$\text{3SAT} = \{\varphi \mid \varphi \text{ is a satisfiable Boolean formula in 3-CNF}\}.$$

Given a formula $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ over variables $x_1, \ldots, x_n$, we construct in polynomial time a directed graph G_φ with distinguished vertices s and t such that φ is satisfiable $\iff \langle G_\varphi, s, t\rangle \in$ HAMPATH.
Construction outline:

- **Variable gadgets:** For each variable x_i we build a little "diamond" forcing the Hamiltonian path to choose exactly one of two routes, corresponding to $x_i =$ True or $x_i =$ False. Concretely, introduce vertices

  $$a_i,\ T_i,\ F_i,\ b_i,$$

  and edges

  $$a_i \to T_i \to b_i, \quad a_i \to F_i \to b_i.$$

  Link the gadgets in sequence:

  $$s = a_1, \quad b_i \to a_{i+1}\ (\forall i = 1, \ldots, n-1), \quad b_n \to c_1.$$

- **Clause gadgets:** After the variable gadgets, create clause vertices

$$c_1, c_2, \ldots, c_m, \quad c_m \to t.$$

For each clause $C_j = (\ell_{j1} \vee \ell_{j2} \vee \ell_{j3})$, add edges from the *appropriate* literal-route vertex into c_j:

$$\text{if } \ell_{jk} = x_i, \text{ add } T_i \to c_j; \quad \text{if } \ell_{jk} = \neg x_i, \text{ add } F_i \to c_j.$$

Then add edges $c_j \to c_{j+1}$ for $j = 1, \ldots, m-1$, and finally $c_m \to t$.

Correctness. A Hamiltonian path from s to t must:

(i) traverse each variable gadget by choosing exactly one of the two routes $a_i \to T_i \to b_i$ or $a_i \to F_i \to b_i$, thereby fixing a truth assignment,

(ii) then proceed from b_n into the clause gadgets in order,

(iii) visit each c_j exactly once, which is possible *only* if at least one incoming edge from the chosen literal-route is present, i.e. that clause is satisfied.

Thus the path exists if and only if there is a satisfying assignment for φ. This completes the reduction and shows NP-hardness. □

Definition 8.7 (Max-Cut)**.** Let

$$\text{MAXCUT} = \big\{ \langle G = (V, E, w), K \rangle \mid \exists\, (S, V \setminus S) \text{ cut of total weight } \geq K \big\},$$

where $w : E \to \mathbb{Z}^+$ assigns positive integer weights.

Theorem 8.11. MAXCUT *is NP-complete.*

Proof. A certificate is a partition $S \subseteq V$. In polynomial time we sum the weights $w(e)$ of all edges e crossing between S and $V \setminus S$ and check if the total is $\geq K$.
We reduce from the NP-complete *Partition* problem:

$$\text{PART} = \big\{ a_1, \ldots, a_n \mid \sum_i a_i \text{ is even and there is } S \subseteq [n] \text{ with } \sum_{i \in S} a_i = \tfrac{1}{2} \sum_i a_i \big\}.$$

Given $(a_1, \ldots, a_n)$ with total $2B$, construct a complete graph G on n vertices, labeling vertex i. Assign each (undirected) edge $\{i, j\}$ weight $a_i \cdot a_j$. Let $K = B^2$.
Claim: There is a partition S with $\sum_{i \in S} a_i = B$ if and only if G has a cut of weight $\geq B^2$.

Proof of Claim. If S sums to B, then the cut $(S, \bar{S})$ has total weight

$$\sum_{i \in S,\, j \notin S} a_i\, a_j = \Big(\sum_{i \in S} a_i\Big)\Big(\sum_{j \notin S} a_j\Big) = B \cdot B = B^2.$$

Conversely, if some cut $(S, \bar{S})$ has weight $\geq B^2$, then

$$\sum_{i \in S} a_i \;=\; X, \quad \sum_{j \notin S} a_j \;=\; 2B - X,$$

and the cut weight is $X(2B - X)$. The quadratic $X(2B - X)$ achieves its maximum B^2 only at $X = B$. Hence $X = B$, giving a valid partition.
Thus PART $\leq_p$ MAXCUT, completing the NP-hardness proof □

8.4 General Recipe for Proving NP-Completeness

To show a language $\mathcal{L}$ is NP-complete:

(i) *Show* $\mathcal{L} \in NP$. Exhibit a nondeterministic polynomial-time verifier or an NTM that decides L in polytime.

(ii) *Choose a known NP-complete problem A.* Common choices: 3SAT, CLIQUE, HAMPATH, ...

(iii) *Construct a polynomial-time reduction* $f : A \to \mathcal{L}$. Given an instance x of A, build an instance $f(x)$ of $\mathcal{L}$ in time poly($|x|$).

(iv) *Prove correctness* of the reduction:

$$x \in A \iff f(x) \in \mathcal{L}.$$

Once these steps are complete, L is NP-complete.

Remark 8.3. Some common pitfalls in NP-Completeness proofs are the following:

- Reducing $\mathcal{L} \leq_p A$ with $A \in NP$-complete *does not* show $\mathcal{L}$ is NP-complete. One must reduce $A \leq_p \mathcal{L}$, not the other way around.
- Do not let the reduction "solve" the problem $\mathcal{L}$; it must simply transform instances of A into instances of $\mathcal{L}$.
- In the correctness proof, show how a solution (certificate, path, cut, etc.) for $x \in A$ converts to one for $f(x) \in \mathcal{L}$, and vice versa.
- Never forget the first step: prove $\mathcal{L} \in NP$.

Lecture 10: Space Complexity and PSPACE-Completeness

An alternative measure of the complexity of a problem is the maximal amount of *space* a Turing machine uses when solving it.

Definition 10.1. Let M be a Turing machine that halts on *all* inputs. The *space complexity* of M is the function

$$f : \mathbb{N} \to \mathbb{N}, \quad f(n) = \max\{\text{number of tape cells scanned by } M \text{ on any input of length } n\}.$$

Claim 10.1. If M is a decider using space $f(n) \geq n$ on inputs of length n, then its time complexity is at most

$$2^{\mathcal{O}(f(n))}.$$

Proof. On each input of length n, M can only visit at most $f(n)$ distinct tape cells. A *configuration* of M consists of:

- the current state (one of $|Q|$ possibilities),
- the contents of the $f(n)$ scanned cells (each of $|\Gamma|$ symbols),
- the head position (one of $f(n)$ cells).

Hence the total number of possible configurations is

$$|Q| \cdot |\Gamma|^{f(n)} \cdot f(n) = 2^{\mathcal{O}(f(n))}.$$

Since M always halts and never repeats a configuration (otherwise it would loop), it can make at most as many steps as there are configurations. Therefore its running time is bounded by $2^{\mathcal{O}(f(n))}$. □

Definition 10.2. For any function $f : \mathbb{N} \to \mathbb{R}^+$, define

$$\text{SPACE}(f(n)) = \big\{L \mid L \text{ is decidable by a deterministic TM using } \mathcal{O}(f(n)) \text{ space}\big\},$$
$$\text{NSPACE}(f(n)) = \big\{L \mid L \text{ is decidable by a nondeterministic TM using } \mathcal{O}(f(n)) \text{ space}\big\}.$$

Claim 10.2. SAT is decidable in *linear* space.

Proof. Let ϕ be a Boolean formula of length n with $m \leq n$ variables $x_1, \ldots, x_m$. We describe a deterministic TM M_1 that uses $\mathcal{O}(n)$ space and decides satisfiability of ϕ:

(1) On a work tape, maintain a binary counter of length m to enumerate all 2^m truth assignments.

(2) On another work tape, evaluate ϕ under the current assignment in a single left-to-right pass, using $\mathcal{O}(n)$ space to keep track of the subformula values.

(3) If any evaluation yields `true`, accept; otherwise, increment the counter and repeat.

(4) If all assignments are tried without acceptance, reject.

The counter uses $m \le n$ cells, the evaluation uses $\mathcal{O}(n)$ cells, and the input tape holds ϕ. Thus the total space is $\mathcal{O}(n)$. □

Theorem 10.1 (Savitch's Theorem)**.** *For any $f(n) \ge n$,*

$$\mathrm{NSPACE}\big(f(n)\big) \subseteq \mathrm{SPACE}\big(f(n)^2\big).$$

Proof. Let N be an NTM that uses $f(n)$ space on inputs of length n. On input w, consider the directed *configuration graph* $G_{N,w}$, whose vertices are the at most $2^{\mathcal{O}(f(n))}$ configurations of N, and with an edge from c_1 to c_2 if N can move in one step from c_1 to c_2. Then N accepts w if and only if there is a path from the start configuration c_{start} to the unique accept configuration c_{accept}.
We define a deterministic recursive procedure

$$\mathrm{CANYIELD}(c_1, c_2, t)$$

that decides whether there exists a path of length $\le t$ from c_1 to c_2 in $G_{N,w}$. We take $t = 2^{d\,f(n)}$ for a suitable constant d.

Algorithm CANYIELD(c_1, c_2, t)**:**

(a) If $t = 1$, accept if c_2 is reachable from c_1 in one step (i.e. there is an edge), otherwise reject.

(b) Otherwise, non-deterministically *guess* an intermediate configuration c_m, and recursively check

$$\mathrm{CANYIELD}(c_1, c_m, \lfloor t/2 \rfloor) \quad \text{and} \quad \mathrm{CANYIELD}(c_m, c_2, \lfloor t/2 \rfloor).$$

Converting this to a deterministic algorithm amounts to *trying all* possible c_m one by one and only accepting if some c_m makes both recursive calls accept.

Each configuration c_i can be stored in $\mathcal{O}(f(n))$ space. The recursion depth is $\mathcal{O}(\log t) = \mathcal{O}(f(n))$, and at each level we store two configurations plus the counter for t. Hence total space is

$$\mathcal{O}\big(f(n)\big) \times \mathcal{O}\big(f(n)\big) \;=\; \mathcal{O}\big(f(n)^2\big).$$

Thus a single deterministic TM using $\mathcal{O}(f(n)^2)$ space can decide whether c_{accept} is reachable, i.e. simulate N. This shows $\mathrm{NSPACE}(f(n)) \subseteq \mathrm{SPACE}(f(n)^2)$. □

Corollary 10.1. PSPACE = NPSPACE.

Proof. Clearly PSPACE $\subseteq$ NPSPACE. By Savitch's Theorem, NPSPACE $\subseteq$ SPACE(poly(n)) = PSPACE. $\square$

Remark.

$$P \subseteq NP \subseteq \text{NPSPACE} = \text{PSPACE} \subseteq \text{EXPTIME}.$$

It is conjectured that all these inclusions are strict except the equality NPSPACE = PSPACE.

Definition 10.3 (PSPACE-Completeness)**.** A language B is *PSPACE-complete* if:

(i) $B \in$ PSPACE.

(ii) For every $A \in$ PSPACE, $A \leq_p B$ (polynomial-time many-one reduction).

If only (ii) holds, B is *PSPACE-hard*.

Next we introduce the canonical PSPACE-complete problem.

Definition 10.4 (TQBF)**.**

$$\text{TQBF} = \{\phi \mid \phi \text{ is a true fully-quantified Boolean formula}\}.$$

A *fully-quantified* formula has the form

$$Q_1 x_1 \, Q_2 x_2 \cdots Q_n x_n \; F(x_1, \ldots, x_n),$$

where each Q_i is $\forall$ or $\exists$, and F is a propositional formula.

Theorem 10.2. TQBF *is PSPACE-complete.*

Proof. Let

$$\phi \;=\; Q_1 x_1 \, Q_2 x_2 \cdots Q_n x_n \; F(x_1, \ldots, x_n)$$

be a fully-quantified Boolean formula of total length m. We describe a deterministic algorithm *Eval*(ϕ) that uses $\mathcal{O}(m)$ space:

(1) If there are no quantifiers (i.e. $n = 0$), evaluate the propositional formula F directly in $\mathcal{O}(m)$ time and $\mathcal{O}(m)$ space and return its truth value.

(2) Otherwise, let $Q_1 x_1$ be the first quantifier:

- If $Q_1 = \exists$, then

$$Eval(\phi) \;=\; Eval\big(F[x_1 := 0]\big) \;\vee\; Eval\big(F[x_1 := 1]\big).$$

- If $Q_1 = \forall$, then

$$Eval(\phi) \;=\; Eval\big(F[x_1 := 0]\big) \;\wedge\; Eval\big(F[x_1 := 1]\big).$$

Space analysis:

- We store the remaining suffix of the quantifier prefix and the matrix F in-place on the input tape (read-only).
- We use $\mathcal{O}(1)$ additional work space to record which branch we are exploring ($x_1 = 0$ vs. $x_1 = 1$) and to hold recursive return values.
- The recursion depth is $n \leq m$, and at each level we allocate only $\mathcal{O}(1)$ workspace beyond the space holding the current subformula.

Hence the total space is $\mathcal{O}(m)$. This shows TQBF $\in$ PSPACE.

Let $A \in$ PSPACE. Then there is a deterministic TM M that decides A in space $p(n)$ on inputs of length n. Fix an input w of length n. We will in polynomial time construct a fully-quantified formula $\Phi_{M,w}$ such that

$$w \in A \iff \Phi_{M,w} \text{ is true.}$$

Because M uses at most $p(n)$ tape cells, each configuration can be encoded as a bitstring of length

$$s = \mathcal{O}(p(n)),$$

describing the tape contents, head position, and finite state.
Let $\mathcal{C}$ be the set of all configurations of M on w. Its size is $|\mathcal{C}| \leq 2^{\mathcal{O}(p(n))}$. Define a directed graph $G = (\mathcal{C}, E)$ where $(c_1, c_2) \in E$ iff M can move from configuration c_1 to c_2 in one step.
Then M accepts w exactly if there is a path in G from the start configuration c_{start} to the unique accepting configuration c_{accept}.
Define a family of predicates

$$\text{Reach}_t(c_1, c_2) \;=\; \big(\text{there is a path of length} \leq t \text{ from } c_1 \text{ to } c_2 \text{ in } G\big).$$

We will construct a quantified Boolean formula for $\text{Reach}_T(c_{\text{start}}, c_{\text{accept}})$ with

$$T \;=\; 2^{d\,p(n)}$$

for a suitable constant d, which bounds the maximum number of steps M can take without revisiting a configuration.
We build $\Phi_{M,w}$ by expressing $\mathrm{Reach}_T(c_{\text{start}}, c_{\text{accept}})$ with a standard "divide and conquer" QBF:

$$\mathrm{Reach}_t(c_1, c_2) \;=\; \begin{cases} (c_1 = c_2) \;\vee\; E(c_1, c_2), & \text{if } t = 1, \\ \exists\, c_m \,\big[\mathrm{Reach}_{t/2}(c_1, c_m) \;\wedge\; \mathrm{Reach}_{t/2}(c_m, c_2)\big], & \text{if } t > 1, \end{cases}$$

where $E(c_1, c_2)$ is a propositional formula (of size $\mathcal{O}(s)$) that is true exactly when $(c_1, c_2) \in E$.
Unfolding this definition for $t = T$ yields a fully-quantified Boolean formula $\Phi_{M,w}$ whose:

- *Quantifier depth* is $\mathcal{O}(\log T) = \mathcal{O}(p(n))$.
- *Total size* is polynomial in T times $\mathcal{O}(s)$, which is $2^{\mathcal{O}(p(n))} \cdot \mathcal{O}(p(n))$. However, by sharing subformulas (re-using the same subformula names) one can achieve a final formula of size polynomial in n.
- *Construction time* is polynomial in $|w|$, since each level adds $\mathcal{O}(p(n))$ new variables and clauses and there are $\mathcal{O}(p(n))$ levels.

Therefore $w \in A \iff \Phi_{M,w}$ is true, establishing a polynomial-time reduction $A \leq_p$ TQBF. This completes the PSPACE-hardness proof. □

Lecture 11: Foundations of Quantum Computation

Computing with n-bit integers on a classical Turing machine yields the following:

- **Multiplication:** The product of two n-bit numbers can be computed in time $\mathcal{O}(n \log n)$ using advanced fast Fourier transform (FFT) methods.
- **Factoring:** The best classical algorithm (General Number Field Sieve) factors an n-bit integer in time roughly

$$\exp\left(\mathcal{O}(\sqrt{n}\,)\right) = \mathcal{O}(2^{\sqrt{n}}).$$

Remark 11.1. These classical complexities set baselines: multiplication is near-linear, while factoring remains super-polynomial and intractable for large n.

11.1 Probabilistic Computation

Augmenting classical machines with randomness leads to probabilistic algorithms, often offering polynomial speedups:

- **Primality Testing (PRP vs AKS):** A probabilistic *probable prime test* (PRP, e.g., Miller–Rabin) runs in $\mathcal{O}(n^2)$ time to decide if an n-bit number is prime with controllable error probability, while the deterministic AKS test runs in $\mathcal{O}(n^6)$ time, hence a *polynomial* speed-up.
- **Polynomial Identity (Zero-) Testing:** Given a polynomial $P(x_1, \ldots, x_n)$ represented by an arithmetic circuit, the Schwartz–Zippel lemma allows a randomized test in time proportional to the circuit size to decide if $P \equiv 0$ with high confidence.

Definition 11.1. A *probabilistic Turing machine* is a classical model equipped with a random coin-flip oracle, enabling transitions with specified probabilities.

Proposition 11.1. *The state of an n-bit probabilistic register is a probability distribution $p : \{0,1\}^n \to [0,1]$, evolving under a* stochastic matrix $S \in \mathbb{R}^{2^n \times 2^n}$ *with non-negative entries and each column summing to 1.*

Remark 11.2. Key features of probabilistic computing:

- Access to unbiased coin flips at unit cost.
- Simulation of stochastic processes and randomized heuristics.
- Polynomial (but conjecturally not *exponential*) speedups for certain problems.

11.2 Qubits and Bra-Ket Notation

Passing from real probabilities to complex amplitudes leads us to Hilbert spaces.

Definition 11.2. A *Hilbert space* is a complete vector space over $\mathbb{C}$ with an inner product $\langle\psi|\phi\rangle = \sum_i \overline{\psi_i}\,\phi_i$.

Let $z = a + ib$, $i^2 = -1$. Its magnitude is $|z| = \sqrt{a^2 + b^2} = \sqrt{\bar{z}\,z}$.

Remark 11.3. Operators on $\mathbb{C}^d$ include:

- *Hermitian* $H = H^\dagger$, representing observables with real eigenvalues.
- *Unitary* U, satisfying $U^\dagger U = UU^\dagger = I$, representing reversible quantum evolution.

Definition 11.3. A *qubit* is a unit vector in $\mathbb{C}^2$:

$$|\varphi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \quad |\alpha|^2 + |\beta|^2 = 1.$$

Compare to a classical random bit, whose state $(p, q)^T$ with $p + q = 1$ lives in the ℓ_1 simplex (a line segment). Pure qubit states form the Bloch sphere in ℓ_2 norm:

$$|\varphi\rangle = \cos(\tfrac{\theta}{2})\,|0\rangle + e^{i\phi}\sin(\tfrac{\theta}{2})\,|1\rangle\,,$$

parametrized by angles (θ, ϕ).

Example 11.1. The unit circle in the complex plane (ignoring global phase) illustrates the continuum of qubit states beyond discrete classical bits.

Column vectors are *kets* $|\psi\rangle \in \mathbb{C}^d$, rows are *bras* $\langle\psi| = |\psi\rangle^\dagger$. We have:

$$\langle\psi|\phi\rangle = \sum_i \overline{\psi_i}\,\phi_i, \quad \langle\psi|\psi\rangle = \|\psi\|^2.$$

The computational basis states are $|0\rangle = (1, 0)^T$, $|1\rangle = (0, 1)^T$, orthonormal since $\langle 0|1\rangle = 0$.

Definition 11.4. Define the states:

$$|+\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \quad |-\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

Claim 11.1. These form an orthonormal basis: $\langle +|-\rangle = 0$, with the Hadamard transform

$$H = \tfrac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad H^2 = I.$$

Lecture 12: Quantum Measurements and Dynamics

Definition 12.1. A *qubit state* is a unit vector $|\psi\rangle \in \mathbb{C}^2$, with respect to the standard inner product:

$$\| |\psi\rangle \|^2 = \left|\langle 0|\psi\rangle\right|^2 + \left|\langle 1|\psi\rangle\right|^2 = 1.$$

Theorem 12.1 (Born Rule)**.** *Let $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. Measuring in the computational basis $\{|0\rangle, |1\rangle\}$ yields outcome $i \in \{0,1\}$ with probability*

$$p(i) = \left|\langle i|\psi\rangle\right|^2 = \begin{cases} |\alpha|^2 & i = 0, \\ |\beta|^2 & i = 1, \end{cases}$$

and the post-measurement state becomes

$$\frac{|i\rangle\langle i|\,\psi}{\sqrt{p(i)}} = |i\rangle\,.$$

Proof. Measurement is modeled by the set of projectors $\{P_0 = |0\rangle\langle 0|, P_1 = |1\rangle\langle 1|\}$ satisfying $P_0 + P_1 = I$. The probability rule follows from $p(i) = \langle\psi|P_i|\psi\rangle$, and collapse by projection postulate. □

Remark 12.1. Generalized measurements (POVMs) extend this framework, but projective measurements suffice for most introductory algorithms.

12.1 Unitary Evolution and the Schrödinger Equation

Proposition 12.1. *The time evolution of a closed quantum state $|\psi_0\rangle$ under a time-independent Hamiltonian $H = H^\dagger$ is given by the unitary operator*

$$U(t) = e^{-iHt/\hbar},$$

so that $|\psi(t)\rangle = U(t)|\psi_0\rangle$ and $U(t)^\dagger U(t) = I$.

Sketch. From the Schrödinger equation $i\hbar\frac{d}{dt}|\psi(t)\rangle = H|\psi(t)\rangle$, integration yields $U(t) = \exp(-iHt/\hbar)$. Unitarity follows since H is Hermitian. □

Definition 12.2. A *unitary* matrix $U \in \mathbb{C}^{2\times 2}$ satisfies $U^\dagger U = UU^\dagger = I$, preserving inner products and norms.

Example 12.1 (Single-Qubit Gates)**.** The following unitaries generate most single-qubit operations:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix},$$

$$H = \tfrac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}.$$

Each corresponds to rotations about Bloch-sphere axes.

12.2 Measurements on Quantum Circuits: Measurement and Feedback

In measurement-based quantum circuits, outcomes of intermediate measurements can dictate subsequent operations. This feedback loop is crucial in protocols like quantum teleportation and error correction.

Example 12.2 (Two-Stage Circuit with Feedback)**.** Consider the circuit:

$$|0\rangle \xrightarrow{H} |+\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \xrightarrow{\text{measure}} \begin{cases} |0\rangle & \text{with } p = 1/2, \\ |1\rangle & \text{with } p = 1/2. \end{cases}$$

The classical bit outcome $m \in \{0, 1\}$ is then used to apply a conditional gate:

$$|m\rangle \xrightarrow{H} |\phi_m\rangle\,,$$

where $|\phi_0\rangle = |+\rangle$ and $|\phi_1\rangle = |-\rangle$. Finally, measuring again yields a second random bit independent of the first.
Explicitly:

- If first measurement $m = 0$, state collapses to $|0\rangle$. Applying H gives $|+\rangle$, which measured yields 0 or 1 equally.
- If $m = 1$, collapse to $|1\rangle$; $H|1\rangle = |-\rangle$, again yielding 0 or 1 equally but with a phase difference irrelevant to measurement.

This illustrates how measurement results can inform later operations, a building block of adaptive quantum algorithms.

The bomb tester employs an interferometer to detect a live bomb without direct interaction.

Example 12.3 (Elitzur–Vaidman Bomb Tester)**.** Setup: a Mach–Zehnder interferometer splits a photon into paths $|a\rangle$ and $|b\rangle$ via a beam splitter BS_1:

$$|0\rangle \xrightarrow{BS_1} \tfrac{1}{\sqrt{2}}(|a\rangle + |b\rangle).$$

If path b contains a live bomb detector, any photon there triggers an explosion (interaction), collapsing the superposition to $|a\rangle$. Recombining at BS_2 yields:

$$\tfrac{1}{\sqrt{2}}(|a\rangle + |b\rangle) \to |0\rangle \quad \text{(if no bomb)}; \quad |a\rangle \to \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \quad \text{(if bomb present)}.$$

Detectors D_0, D_1 at outputs click with probability:

- No bomb: D_0 always clicks (constructive interference).
- Live bomb: D_1 clicks with 1/4 probability, revealing bomb without detonation; with 1/2 the photon is absorbed (bomb explodes).

Thus, one can detect a bomb "interaction-free" with non-zero probability.

Remark 12.2. This thought experiment illustrates counterfactual reasoning in quantum physics: information gained even when the interaction did not occur.

Another important discovery is that rapid, repeated measurements can freeze evolution of a quantum state.

Theorem 12.2 (Quantum Zeno Effect)**.** *Let $|\psi\rangle$ evolve under H. Performing N projective measurements onto $|\psi\rangle$ at intervals T/N yields survival probability*

$$P_N = \left|\langle\psi|\left(e^{-iHT/(N\hbar)}\right)|\psi\rangle\right|^{2N} \approx \left(1 - \tfrac{(\Delta E)^2 T^2}{2N^2\hbar^2}\right)^N \to 1$$

as $N \to \infty$, where ΔE is energy uncertainty in $|\psi\rangle$.

Outline. Expand $e^{-iH\Delta t/\hbar} \approx I - iH\Delta t/\hbar - (H\Delta t)^2/2\hbar^2$. Each measurement projects back onto $|\psi\rangle$, suppressing off-diagonal transitions. □

12.3 Multi-Qubit States and Entanglement

Definition 12.3. The joint Hilbert space of n qubits is $(\mathbb{C}^2)^{\otimes n}$, with computational basis $\{|x\rangle\}_{x\in\{0,1\}^n}$.

Definition 12.4. Measuring $|\psi\rangle \in (\mathbb{C}^2)^{\otimes n}$ in the computational basis yields outcome x with probability $|\langle x|\psi\rangle|^2$, collapsing to $|x\rangle$.

Definition 12.5 (Entanglement)**.** A two-qubit state $|\psi\rangle$ is *product* if $|\psi\rangle = |\alpha\rangle \otimes |\beta\rangle$; otherwise it is *entangled*.

Example 12.4 (Bell Pair)**.** The maximally entangled state

$$|\Phi^+\rangle = \tfrac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

is non-separable, as no single-qubit states $|\alpha\rangle, |\beta\rangle$ satisfy $|\Phi^+\rangle = |\alpha\rangle \otimes |\beta\rangle$.

Lecture 13: Multi-Qubit Gates, Universality, and Algorithms

The tensor product formalism allows one to build joint operations on composite quantum systems.

Definition 13.1. Given vector spaces $\mathcal{H}_1$, $\mathcal{H}_2$ and linear operators $A : \mathcal{H}_1 \to \mathcal{H}_1$, $B : \mathcal{H}_2 \to \mathcal{H}_2$, the *tensor product operator* $A \otimes B$ acts on the joint space $\mathcal{H}_1 \otimes \mathcal{H}_2$ by

$$(A \otimes B)(|\alpha\rangle \otimes |\beta\rangle) = (A\,|\alpha\rangle) \otimes (B\,|\beta\rangle)$$

for all product states $|\alpha\rangle \in \mathcal{H}_1$, $|\beta\rangle \in \mathcal{H}_2$, and extends by linearity.

Sketch of Proof. Expanding

$$A = \sum_{ij} a_{ij}\,|i\rangle\langle j|$$

and

$$B = \sum_{kl} b_{kl}\,|k\rangle\langle l|\,,$$

$$A \otimes B = \sum_{i,j,k,l} a_{ij} b_{kl} (|i\rangle\langle j| \otimes |k\rangle\langle l|),$$

so its action on $\sum_{j,l} \alpha_j \beta_l\,|j\rangle\,|l\rangle$ yields the stated separable form. □

Remark 13.1. Tensor products preserve linearity: although state collapse is nonlinear, unitary actions on product states remain linear maps on the composite space.

13.1 Entanglement Generation

The CNOT gate is the simplest two-qubit entangling operation.

Definition 13.2. CNOT is the unitary on $\mathbb{C}^2 \otimes \mathbb{C}^2$ defined by

$$\mathrm{CNOT}\,|\alpha, \beta\rangle = |\alpha, \beta \oplus \alpha\rangle \quad (\alpha, \beta \in \{0, 1\}),$$

where $\oplus$ denotes addition modulo 2.

Example 13.1 (Matrix Representation)**.** In the computational basis $\{|00\rangle\,, |01\rangle\,, |10\rangle\,, |11\rangle\}$,

$$\mathrm{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

Proposition 13.1. *CNOT together with arbitrary single-qubit unitaries generates any two-qubit unitary up to arbitrary precision.*

Applied to a superposition, CNOT creates entanglement.

Example 13.2 (Bell State Preparation)**.** Start from $|0\rangle\,|0\rangle$.

$$H_1 :\ |0\rangle\,|0\rangle \to \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\,|0\rangle\,,$$
$$\text{CNOT} :\ \tfrac{1}{\sqrt{2}}(|0\rangle\,|0\rangle + |1\rangle\,|0\rangle) \to \tfrac{1}{\sqrt{2}}(|00\rangle + |11\rangle) = |\Phi^+\rangle\,.$$

This *Bell state* is maximally entangled: its reduced density matrices are maximally mixed.

13.2 Universality of Gate Sets and Multi-Control Gates

A central result is that a small gate set can approximate any unitary.

Definition 13.3. A finite set of gates $\mathcal{G}$ is *universal* if for every n-qubit unitary $U \in U(2^n)$ and $\varepsilon > 0$, there exists a finite sequence $G_1, G_2, \ldots, G_k \in \mathcal{G}$ such that

$$\left\| U - G_k \cdots G_2 G_1 \right\| \leq \varepsilon.$$

Theorem 13.1 (Lloyd Universality)**.** *Almost any two-qubit entangling gate (e.g. CNOT) together with generic single-qubit rotations is universal for quantum computation.*

Theorem 13.2 (Solovay–Kitaev)**.** *Given a finite gate set generating a dense subgroup of $SU(2)$, any target single-qubit gate U can be approximated within accuracy ε using $\mathcal{O}\big(\log^c(1/\varepsilon)\big)$ gates, with $c \approx 4$.*

Proof Sketch. Use recursive commutator constructions to refine approximations: if A and B approximate desired rotations, their group commutator yields higher-order error cancellation, enabling polylogarithmic scaling. □

The Toffoli gate extends control to two qubits.

Definition 13.4. The Toffoli (CCNOT) gate acts on three qubits:

$$\text{CCNOT}\,|a, b, c\rangle = |a, b, c \oplus (a \cdot b)\rangle \quad (a, b, c \in \{0, 1\}).$$

Proposition 13.2. *Toffoli plus Hadamard generate all reversible Boolean functions; with arbitrary single-qubit gates, they achieve full quantum universality.*

Remark 13.2. In fault-tolerant architectures, Toffoli is often synthesized from CNOT and T gates via ancilla-mediated constructions.

13.3 Deutsch's Algorithm

Deutsch's problem illustrates quantum parallelism and interference.

Definition 13.5 (Oracle Model)**.** A Boolean function $f : \{0,1\} \to \{0,1\}$ is accessed via a unitary oracle

$$U_f \left|x, y\right\rangle = \left|x, y \oplus f(x)\right\rangle .$$

Theorem 13.3 (Deutsch's Algorithm)**.** *For a promised constant or balanced f, one query to U_f suffices to decide its type with certainty, compared to two queries classically.*

Proof. We start by preparing $\left|\psi_0\right\rangle = \left|0\right\rangle \otimes \left|1\right\rangle$.
We apply hadamards

$$H^{\otimes 2} \left|\psi_0\right\rangle = \frac{1}{2} \sum_{x,y} (-1)^y \left|x, y\right\rangle ,$$

and we query the oracle, with U_f mapping $\left|x, y\right\rangle \to \left|x, y \oplus f(x)\right\rangle$, giving

$$\frac{1}{2} \sum_{x,y} (-1)^y \left|x, y \oplus f(x)\right\rangle = \frac{1}{2} \sum_{x,y} (-1)^{y \oplus f(x)} \left|x, y\right\rangle .$$

We then uncompute second Hadamard: apply H on the second qubit leaves it in $\left|-\right\rangle = \frac{1}{\sqrt{2}}(\left|0\right\rangle - \left|1\right\rangle)$ and transforms the first qubit to

$$\tfrac{1}{2} \sum_{x} (-1)^{f(x)} \big(\left|0\right\rangle + \left|1\right\rangle\big) = \tfrac{(-1)^{f(0)} + (-1)^{f(1)}}{2} \left|0\right\rangle + \tfrac{(-1)^{f(0)} - (-1)^{f(1)}}{2} \left|1\right\rangle .$$

And lastly, measurement of the first qubit yields 0 if $f(0) = f(1)$ (constant) and 1 otherwise (balanced), distinguishing with certainty. □

Remark 13.3. Deutsch's algorithm generalizes to Deutsch–Jozsa on n-bit functions, detecting promised global properties with exponentially fewer queries than classical.

Lecture 14: Quantum Oracles and Algorithms

14.1 Quantum Oracle Construction and Complexity Class

Definition 14.1 (Quantum Oracle Q_f)**.** For a classical function $f : \{0,1\}^n \to \{0,1\}^m$, its *quantum oracle* is the unitary operator

$$Q_f : |x\rangle_{\text{in}} |y\rangle_{\text{out}} \mapsto |x\rangle_{\text{in}} |y \oplus f(x)\rangle_{\text{out}} ,$$

where $x \in \{0,1\}^n$ is the query register and $y \in \{0,1\}^m$ is the target register initialized to a known state (often 0^m).

This construction ensures reversibility: even if f is irreversible, Q_f is a bijection on the computational basis and thus extends linearly to a unitary on the full Hilbert space.

Remark 14.1 (Implementation Complexity)**.** If f requires T_f classical steps (gates) to evaluate, then Q_f can be implemented with $\mathcal{O}(T_f)$ reversible gates plus overhead for uncomputing ancillae, yielding a circuit of size $\text{poly}(n, m, T_f)$.

Definition 14.2 (BQP)**.** BQP (Bounded-error Quantum Polynomial time) is the class of promise decision problems solvable by a polynomial-size, uniform quantum circuit family $\{C_n\}$ such that for each input x of length n:

- If x is a YES instance, $C_n(x)$ accepts with probability $\geq 2/3$.
- If x is a NO instance, $C_n(x)$ accepts with probability $\leq 1/3$.

Error amplification via repetition can reduce the error to exponentially small in polynomial time.

14.2 Deutsch's Problem: One-Query Quantum Solution

Definition 14.3 (Deutsch's Problem)**.** Given an oracle O_g for $g : \{0,1\} \to \{0,1\}$, determine whether g is *constant* ($g(0) = g(1)$) or *balanced* ($g(0) \neq g(1)$) using as few queries as possible.

Theorem 14.1. *Deutsch's algorithm solves the problem with a single quantum query and zero error.*

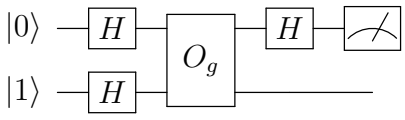
Proof. The Step-by-Step Procedure is

1. **Initialization:** Prepare two qubits in state $|\psi_0\rangle = |0\rangle_1 |1\rangle_2$.

2. **Hadamard Transform:** Apply H to both qubits: $|\psi_1\rangle = (H \otimes H)\,|0,1\rangle = \frac{1}{2}\sum_{x,y}(-1)^y\,|x,y\rangle$.
3. **Oracle Query:** Apply O_g to get $|\psi_2\rangle = O_g\,|\psi_1\rangle = \frac{1}{2}\sum_{x,y}(-1)^y\,|x, y \oplus g(x)\rangle$.
4. **Interference:** Apply H to the first qubit: $|\psi_3\rangle = (H \otimes I)\,|\psi_2\rangle$. One finds that the first-qubit amplitude for $|0\rangle$ is nonzero only when g is constant.
5. **Measurement:** Measure the first qubit. Outcome `0` implies g is constant; `1` implies balanced.

□

Example 14.1 (Quantum Circuit)**.**



14.3 Simon's Problem and Exponential Speedup

Definition 14.4 (Simon's Problem)**.** Let $n \in \mathbb{N}$ and let

$$f : \{0,1\}^n \longrightarrow \{0,1\}^n$$

be an oracle (black-box) function that satisfies the two-to-one promise

$$\forall x, y \in \{0,1\}^n : \quad f(x) = f(y) \iff y = x \oplus s$$

for a single, unknown, non-zero "secret string" $s \in \{0,1\}^n \setminus \{0^n\}$. The task is to determine s using queries to the oracle unitary[1]

$$O_f : \; |x\rangle\,|y\rangle \;\longmapsto\; |x\rangle\,|y \oplus f(x)\rangle\,.$$

Theorem 14.2 (Simon's Algorithm)**.** *There exists a quantum algorithm that, with probability $\geq 1 - \varepsilon$ (for arbitrary constant $0 < \varepsilon < 1$),*

- *queries the oracle O_f at most $N = \mathcal{O}(n)$ times,*
- *performs* $\mathrm{poly}(n)$ *additional (classical and quantum) operations,*
- *and outputs the secret string s.*

Any bounded-error classical algorithm needs $\Omega\big(2^{n/2}\big)$ oracle queries.

[1]Throughout, registers are assumed to be n-qubit and $\oplus$ is bit-wise XOR.

Algorithm 1 One iteration of Simon's quantum procedure

Require: Oracle O_f as in Definition 14.4

1: **procedure** SIMONITER(O_f)

2: **initialize** two n-qubit registers in $|0^n\rangle\,|0^n\rangle$

3: Apply $H^{\otimes n}$ to the first register

$$|\psi_1\rangle = 2^{-n/2} \sum_{x\in\{0,1\}^n} |x\rangle\,|0^n\rangle$$

4: Query the oracle O_f

$$|\psi_2\rangle = 2^{-n/2} \sum_{x} |x\rangle\,|f(x)\rangle$$

5: Measure the second register $\to f(x_0)$

$$|\psi_3\rangle = 2^{-1/2}\big(|x_0\rangle + |x_0 \oplus s\rangle\big)$$

6: Apply $H^{\otimes n}$ to the first register

$$|\psi_4\rangle = 2^{-(n+1)/2} \sum_{y\in\{0,1\}^n} (-1)^{x_0 y}\big(1 + (-1)^{s\cdot y}\big)\,|y\rangle$$

7: Measure the first register; **return** y ▷ only y with $s \cdot y = 0$ have non-zero amplitude

State-evolution summary. Writing just the first register after the measurement in line 4 of Algorithm 1:

$$|\psi_{\text{post}}\rangle \;=\; \frac{1}{\sqrt{2}}\big(|x_0\rangle + |x_0 \oplus s\rangle\big) \;\xrightarrow{H^{\otimes n}}\; \frac{1}{\sqrt{2^{n+1}}} \sum_{y} (-1)^{x_0\cdot y}\left(1 + (-1)^{s\cdot y}\right)|y\rangle\,.$$

The global phase $(-1)^{x_0\cdot y}$ is irrelevant; the crucial factor $1+(-1)^{s\cdot y}$ *filters out* exactly the 2^{n-1} bit-strings orthogonal to s.

Sketch of Proof. Repeat Algorithm 1 until you have obtained $m \geq n-1$ linearly independent vectors $y^{(1)}, \ldots, y^{(m)}$ satisfying $y^{(i)} \cdot s = 0$. A coupon-collector argument shows that $m = n + \mathcal{O}(1)$ suffices with probability $1-\varepsilon$. Collect the m parity equations in matrix form $Y\,s = 0$ over $\mathbb{F}_2$ and solve by Gaussian elimination in $\mathcal{O}(n^3)$ classical time. The secret $s \neq 0^n$ is the unique non-zero vector in the null-space of Y.

Simon's original collision argument shows that a classical algorithm must evaluate f on $\Omega(2^{n/2})$ inputs on average before seeing *any* collision $f(x) = f(y)$, which is necessary to learn $s = x \oplus y$. □

Example 14.2 (A hand-worked case for $n = 3$)**.** Let the secret be $s = 110_2$. Suppose two iterations of Algorithm 1 yield

$$y^{(1)} = 010, \quad y^{(2)} = 001.$$

The resulting linear system is

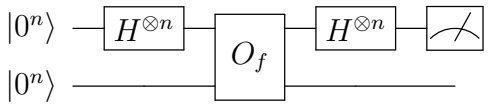
$$\begin{aligned} 0 \cdot s_1 \oplus 1 \cdot s_2 \oplus 0 \cdot s_3 &= 0, \\ 0 \cdot s_1 \oplus 0 \cdot s_2 \oplus 1 \cdot s_3 &= 0, \end{aligned} \qquad \Longrightarrow \qquad s = 110.$$

Indeed two independent equations suffice because the space orthogonal to s has dimension $n - 1 = 2$.

Remark 14.2 (Complexity table)**.**

Resource	Quantum (Simon)	Classical (best known)
Oracle queries	$\mathcal{O}(n)$	$\Omega(2^{n/2})$
Gate depth	$\mathcal{O}(n)$	n/a
Classical post-processing	$\mathcal{O}(n^3)$	$\mathcal{O}(n \cdot 2^{n/2})$

Example 14.3 (Quantum Circuit)**.**



Lecture 15: Grover's Search Algorithm

Grover's algorithm finds a marked element in an unstructured database of size $N = 2^n$ using only $\mathcal{O}\big(\sqrt{N/k}\big)$ uses of the oracle, where k is the (unknown) number of solutions. This is a quadratic improvement over the classical $\Theta(N/k)$ bound and is optimal in the quantum query model.

15.1 Problem set-up

Definition 15.1 (Search Oracle)**.** Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean indicator function whose 1-inputs form the *marked set* $M = \{x : f(x) = 1\}$ with $|M| = k \geq 1$. The phase-oracle unitary is

$$O_f : |x\rangle \longmapsto (-1)^{f(x)} |x\rangle \quad \Big(= I - 2\sum_{x \in M} |x\rangle\langle x| \Big).$$

Definition 15.2 (Unstructured Search)**.** Given black-box access to O_f, output *any* element of M. We assume no promise on the structure of M beyond its size k.

Remark 15.1 (Classical baseline)**.** Sampling inputs uniformly until a hit is seen requires $\Theta(N/k)$ oracle calls on average. No classical algorithm can do better in the worst case.

15.2 Geometry of amplitude amplification

Let

$$|\alpha\rangle = \frac{1}{\sqrt{k}} \sum_{x \in M} |x\rangle\,, \qquad |\beta\rangle = \frac{1}{\sqrt{N-k}} \sum_{x \notin M} |x\rangle\,,$$

so that $\{|\alpha\rangle, |\beta\rangle\}$ is an orthonormal basis for the two-dimensional subspace that matters. Write the initial uniform superposition as

$$|\psi_0\rangle \;=\; \frac{1}{\sqrt{N}} \sum_{x} |x\rangle \;=\; \cos\theta\, |\beta\rangle + \sin\theta\, |\alpha\rangle\,, \qquad \sin^2\theta = \frac{k}{N}.$$

Two reflections:

(1) **Oracle reflection** O_f: multiplies $|\alpha\rangle$ by -1 and leaves $|\beta\rangle$ unchanged.

(2) *Diffusion operator*

$$D \;=\; 2\,|\psi_0\rangle\langle\psi_0| - I \;=\; H^{\otimes n}\big(2\,|0\rangle\langle 0| - I\big)H^{\otimes n}$$

reflects about $|\psi_0\rangle$.

Their product

$$G \;=\; D\,O_f$$

is a rotation by angle 2θ in the plane $\operatorname{span}\{|\beta\rangle, |\alpha\rangle\}$:

$$G\begin{pmatrix}|\beta\rangle\\|\alpha\rangle\end{pmatrix} = \begin{pmatrix}\cos(2\theta) & -\sin(2\theta)\\ \sin(2\theta) & \cos(2\theta)\end{pmatrix}\begin{pmatrix}|\beta\rangle\\|\alpha\rangle\end{pmatrix}.$$

15.3 Grover's algorithm

Algorithm 2 Grover's quantum search (outputs a marked element with high probability)

Require: Oracle O_f; unknown number of marked items $k \geq 1$

1: **Prepare** $|\psi_0\rangle = H^{\otimes n}|0^n\rangle$

2: Estimate k (if not given) or replace the loop count in line 4 with the canonical BBHT iterative–deepening schedule

3: Compute $r = \left\lfloor \dfrac{\pi}{4\theta} \right\rfloor \;=\; \Theta\!\left(\sqrt{N/k}\right)$

4: **for** $i \leftarrow 1$ **to** r **do**

5: Apply O_f followed by D ▷ one Grover step

6: **Measure** in the computational basis and output the observed n-bit string

15.4 Performance guarantee

Theorem 15.1 (Success probability)**.** *After r Grover iterations,*

$$\Pr[\textit{output} \in M] \;=\; \sin^2\!\big((2r+1)\theta\big) \;\geq\; 1 - \mathcal{O}\big(k/N\big).$$

Choosing $r \approx \frac{\pi}{4}\sqrt{N/k}$ makes the right-hand side $1 - \mathcal{O}\big(1/N\big)$.

Sketch of Proof. Express the state after r rounds as $|\psi_r\rangle = \cos((2r+1)\theta)\,|\beta\rangle + \sin((2r+1)\theta)\,|\alpha\rangle$. Measuring in the computational basis lands in M exactly when the projection onto $|\alpha\rangle$ is observed, yielding the asserted probability. □

Remark 15.2 (Optimality)**.** Bennett, Bernstein, Brassard and Vazirani showed that any quantum algorithm needs $\Omega(\sqrt{N/k})$ oracle calls, so Grover's query complexity is optimal up to constant factors.

15.5 Worked example ($n = 4$, $k = 1$)

With $N = 16$ and a single marked string, $\sin\theta = 1/\sqrt{16} = 1/4$ and $\theta \approx 14.48°$. Hence $r = \lfloor \pi/(4\theta) \rfloor = 3$. A textbook simulation yields

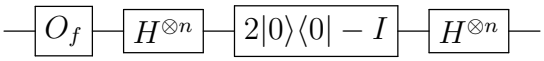
iteration	Pr[marked]
0	1/16
1	9/16
2	15/16
3	99/100

after which measuring gives the unique solution with probability ≥ 0.99.

15.6 Resource summary

Resource	Grover (quantum)	Classical random sampling
Oracle queries	$\mathcal{O}\big(\sqrt{N/k}\big)$	$\Theta(N/k)$
Gate depth	$\mathcal{O}\big(\sqrt{N/k} \cdot n\big)$	n/a
Extra qubits	$n + \Theta(1)$	n

Example 15.1 (Single Grover iteration as a circuit)**.**



The middle gate can be decomposed into one multi-controlled Z surrounded by layer-wise Hadamards.

Remark 15.3 (Known-k vs. unknown-k)**.** When k is unknown, the canonical solution is the *BBHT variable-iteration algorithm*: repeat Grover with geometrically increasing iteration counts $r = 1, 2, 4, \ldots$, measuring after each block and aborting on success. This keeps the overall expected queries in $\Theta\big(\sqrt{N/k}\big)$.

Lecture 16: The Discrete Logarithm Problem

Throughout let $G = \langle g \rangle$ be a cyclic group of order $N > 1$ written multiplicatively. (For concreteness think of $G = \mathbb{Z}_p^\times$ or an elliptic-curve group $E(\mathbb{F}_q)$.)

16.1 Problem statement and classical background

Definition 16.1 (Cyclic group)**.** A finite group G is *cyclic* if $G = \{g^0, g^1, \dots, g^{N-1}\}$ for some generator g.

Definition 16.2 (Discrete logarithm)**.** Given $h \in G$ there is a unique $x \in \{0, \dots, N-1\}$ such that $g^x = h$. We write $x = \log_g h$.

Definition 16.3 (Discrete Logarithm Problem (DLP))**.** Input (G, g, h) with known $N = |G|$; output $x = \log_g h$.

Remark 16.1 (Best classical algorithms)**.** Generic algorithms such as baby-step/giant-step and Pollard's ρ require $\Theta(\sqrt{N})$ group operations—exponential in the input size $n = \lceil \log_2 N \rceil$. Special-purpose index-calculus methods exist for $\mathbb{Z}_p^\times$ but not for elliptic curves; in either case, no known *polynomial* classical algorithm is available.

16.2 From DLP to a hidden period

Fix the unknown $x = \log_g h$. Define the two-variable function

$$f : \mathbb{Z}_N \times \mathbb{Z}_N \longrightarrow G, \qquad f(a, b) = g^a h^{-b}. \tag{1}$$

Proposition 16.1 (Period lattice)**.** *f is constant along the one-dimensional lattice $L = \{\, t\,(x, 1) \mid t \in \mathbb{Z}_N \} \subset \mathbb{Z}_N^2$:*

$$f(a, b) = f(a', b') \quad \Longleftrightarrow \quad (a', b') - (a, b) \in L.$$

Proof. Compute $f(a + xt, b + t) = g^{a+xt} h^{-(b+t)} = g^a h^{-b}\,(g^x h^{-1})^t = f(a, b)$ because $g^x = h$. The converse follows by rearranging the equality $g^{a-a'} = h^{b-b'} = g^{x(b-b')}$ in the exponent ring $\mathbb{Z}_N$. □

Thus DLP is a *hidden-subgroup / hidden-period* instance over the Abelian group $\mathbb{Z}_N^2$ with hidden shift $(x, 1)$.

16.3 Shor's quantum algorithm

Algorithm 3 Shor's discrete-log algorithm (outputs $x = \log_g h$ with high probability)

Require: Cyclic group $\langle g \rangle$ of order N; element $h = g^x$ (unknown x) ▷ $n = \lceil \log_2 N \rceil$ qubits per index register

1: **Prepare** three registers: two n-qubit index registers in $|0^n, 0^n\rangle$ and one group register in $|1_G\rangle$

2: Apply $H^{\otimes 2n}$ to the index registers: $|\psi_1\rangle = \frac{1}{N} \sum_{a,b \in \mathbb{Z}_N} |a, b\rangle \, |1_G\rangle$

3: **Oracle** U_f: compute $f(a, b) = g^a h^b$ into the group register ▷ one group multiplication

State becomes $|\psi_2\rangle = \frac{1}{N} \sum_{a,b} |a, b\rangle \, |f(a, b)\rangle$

4: Measure the group register, leaving a uniform superposition of one coset of $L = \{(t, -xt)\}$: $|\psi_3(a_0, b_0)\rangle = \frac{1}{\sqrt{N}} \sum_{t=0}^{N-1} |a_0 + xt, \; b_0 + t\rangle$

5: Apply QFT_N to each index register: $|u, v\rangle \mapsto \frac{1}{N} \sum_{k,\ell \in \mathbb{Z}_N} e^{2\pi i (uk + v\ell)/N} \, |k, \ell\rangle$

Resulting state $\frac{1}{\sqrt{N}} \sum_{k,\ell} \underbrace{\left(\frac{1}{N} \sum_{t=0}^{N-1} e^{2\pi i (k + \ell x) t / N} \right)}_{\delta_{\, k + \ell x \equiv 0}} e^{2\pi i (a_0 k + b_0 \ell)/N} \, |k, \ell\rangle$

6: Measure the index registers; outcome (k, ℓ) obeys

$$k + \ell x \equiv 0 \pmod{N}. \tag{2}$$

7: **Classical post-processing**: if $\gcd(\ell, N) = 1$ then $x \equiv -k \, \ell^{-1} \pmod{N}$; else repeat from line 3 until an invertible ℓ is obtained

16.4 State-evolution table

Step	State of the first two registers
0	$\lvert 0^n, 0^n\rangle$
1	$\frac{1}{N}\sum_{a,b} \lvert a, b\rangle$
3	$\frac{1}{\sqrt{N}}\sum_{t} \lvert a_0 + xt,\ b_0 + t\rangle$
5	$\frac{1}{N}\sum_{k,\ell:\, k+\ell x\equiv 0} e^{2\pi i(a_0 k + b_0 \ell)/N} \lvert k, \ell\rangle$
6	Classical pair (k, ℓ) s.t. $k + \ell x = 0$

16.5 Correctness and success probability

Theorem 16.1. *Algorithm 3 returns $x = \log_g h$ with probability at least $1 - \mathcal{O}(1/N)$ after $\mathcal{O}(\log N)$ repetitions.*

Sketch. Each run produces a random solution of Eq. (2). The probability that the sampled ℓ is coprime to N is $\varphi(N)/N \geq \Omega(1/\log\log N)$ by Euler's theorem; hence $\mathcal{O}(\log N)$ independent runs suffice to obtain at least one invertible ℓ with overwhelming probability. The modular inversion and multiplication recover x uniquely. □

Gate complexity. The two QFTs dominate the circuit cost: QFT_N can be implemented with $\frac{1}{2}n(n-1) = \mathcal{O}(n^2)$ controlled-phase gates, so the whole algorithm uses $\mathcal{O}(n^3)$ elementary gates and $\mathcal{O}(n)$ qubits.

16.6 Worked example ($N = 11$)

Let $G = \mathbb{Z}_{11}^{\times} = \langle 2\rangle$ and let $h = 9$. A quick calculation shows $2^7 \equiv 9 \pmod{11}$, so $x = 7$.

1. Choose random (a_0, b_0); suppose the oracle measurement returns $f = 2$.
2. The collapsed superposition is $\frac{1}{\sqrt{11}}\sum_{t=0}^{10} \lvert a_0 + 7t,\ b_0 + t\rangle$.
3. After the double QFT a measurement yields e.g. $(k, \ell) = (3, 5)$ because $3 + 7\cdot 5 \equiv 0 \pmod{11}$.
4. Since $\gcd(5, 11) = 1$, $x \equiv -3 \cdot 5^{-1} \equiv -3 \cdot 9 \equiv -27 \equiv 7 \pmod{11}$, the desired answer.

16.7 Resource comparison

Resource	Shor (quantum)	Baby-step & Pollard ρ (classical)
Group operations	$\mathcal{O}\big(\log^3 N\big)$	$\Theta\big(\sqrt{N}\big)$
Oracle queries	n (group mults.)	$\sqrt{N}$
Gate count	$\mathcal{O}\big(n^3\big)$	n/a
Qubits / space	$3n + \Theta(1)$	$\sqrt{N}$

Finite-precision implementations of QFT_N can be truncated to $\mathcal{O}(n^2)$ controlled rotations without affecting the success probability by more than 2^{-n}. Rounding errors in the classical modular arithmetic are negligible if n-bit integers are used.

Lecture 17: Quantum Phase Estimation

Quantum Phase Estimation (QPE) extracts the eigenphase $\phi \in [0, 1)$ of a known unitary U given access to an eigenstate $|\psi\rangle$ and controlled powers U^{2^j}. It is the engine behind Shor's order-finding, quantum chemistry eigenvalue algorithms, and the modern family of amplitude- estimation routines.

17.1 Problem definition and basic notation

Definition 17.1 (Eigenphase)**.** Let U be an m-qubit unitary and $|\psi\rangle$ an eigenstate:

$$U|\psi\rangle = e^{2\pi i\phi}|\psi\rangle, \qquad \phi \in [0, 1).$$

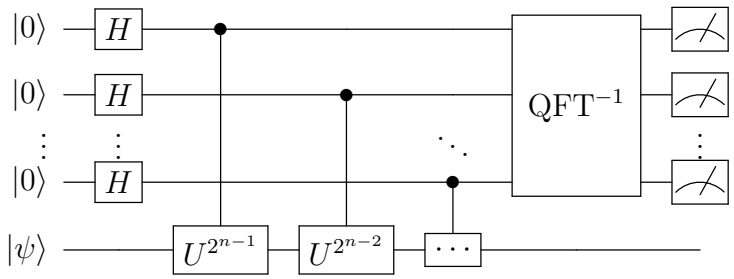
Given $n \in \mathbb{N}$ (desired precision), the *phase-estimation task* is to output $\tilde{\phi} \in \{0, \ldots, 2^n - 1\}$ such that[2] $\left|\phi - \tilde{\phi}/2^n\right| < 2^{-(n+1)}$ with probability $\geq 1 - \varepsilon$ for some target error ε.

The canonical QPE circuit uses

- an n-qubit *phase register* initialised to $|0^n\rangle$,
- an m-qubit *target register* initialised to the eigenstate $|\psi\rangle$.

17.2 Circuit diagram

Example 17.1 (Textbook QPE circuit)**.**



The j-th control (counting MSB $= 0$) applies the power U^{2^j}.

17.3 Step-by-step state evolution

For clarity denote the phase register basis as $|k\rangle$ where $k \in \{0, \ldots, 2^n - 1\}$.

[2] Equivalently, $\tilde{\phi}$ is the nearest integer to $2^n\phi$.

Step	Joint state
(0)	$\lvert 0^n\rangle \lvert\psi\rangle$
(1) $H^{\otimes n}$	$\frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} \lvert k\rangle \lvert\psi\rangle$
(2) controlled U^{2^j}	$\frac{1}{2^{n/2}} \sum_{k} e^{2\pi i k\phi} \lvert k\rangle \lvert\psi\rangle \quad (\because k = \sum_j k_j 2^j)$
(3) QFT^{-1}	$\frac{1}{2^n} \sum_{y,k} e^{2\pi i (k\phi - yk/2^n)} \lvert y\rangle \lvert\psi\rangle$
(4) measure	classical n-bit y with $y \approx 2^n\phi$

The amplitude on $\lvert y\rangle$ is

$$\frac{1}{2^n} \sum_k e^{2\pi i k(\phi - y/2^n)} = \frac{\sin\big(\pi(2^n\phi - y)\big)}{2^n \sin\big(\pi(\phi - y/2^n)\big)} \, e^{i\pi(2^n\phi - y)}$$

—the discrete Dirichlet kernel—yielding the familiar $\geq 4/\pi^2 \approx 0.405$ success probability for rounding to the nearest integer.

17.4 Algorithm description

Algorithm 4 Standard quantum–phase-estimation (QPE) algorithm — outputs an n-bit estimate $\tilde{\phi}$ of the phase ϕ

Require: Controlled powers U^{2^j} for $j = 0, \ldots, n-1$; eigenstate preparation $\lvert\psi\rangle$

1: Initialise the *phase* register in $\lvert 0^n\rangle$ and the *target* register in $\lvert\psi\rangle$
2: Apply $H^{\otimes n}$ to the phase register
3: **for** $j \leftarrow 0$ **to** $n-1$ **do**
4: Apply controlled-U^{2^j} with qubit j as the control
5: Apply inverse QFT, $\mathrm{QFT}_{2^n}^{-1}$, to the phase register
6: Measure the phase register and output $\tilde{\phi} \leftarrow \mathrm{binary}(y)$

17.5 Success probability and precision

Theorem 17.1 (QPE accuracy)**.** *Let $\delta = \lvert\phi - \tilde{\phi}/2^n\rvert$. After one run of Algorithm 4:*

(a) If ϕ has an exact n-bit binary expansion, the outcome equals that expansion with probability 1*.*

(b) Otherwise $\Pr\big[\delta < 2^{-(n+1)}\big] \geq \frac{4}{\pi^2}$*.*

(c) *Repeating the algorithm $t = \lceil \log(1/\varepsilon) \rceil$ times and taking the median raises the confidence to $1 - \varepsilon$.*

Idea. Items (a)–(b) follow by evaluating the Dirichlet kernel squared and summing over the two closest integers to $2^n\phi$. Item (c) is standard Chernoff boosting. □

17.6 Resource analysis

- **Qubits.** n (phase) + m (target) + $\Theta(1)$ work qubits if needed.
- **Gate count.** QFT^{-1} uses $\frac{1}{2}n(n-1) = \mathcal{O}(n^2)$ controlled-phase gates; line 4 costs are problem-dependent.
- **Circuit depth.** The QFT can be parallelised to $\mathcal{O}(n \log n)$; power-of-two controlled unitaries often dominate depth.
- **Fault-tolerance.** Approximating small controlled rotations to $O(n + \log(1/\varepsilon))$ bits suffices to keep the overall algorithmic error below ε.

17.7 Variants and optimisations

Semiclassical (Kitaev) QPE. Replace the full n-qubit QFT with a one-qubit inverse QFT performed iteratively: after measuring the most-significant qubit, classically rotate away its phase shift before measuring the next. This reduces phase-register qubits from n to 1 at the cost of feedback latency.

Iterative QPE. The iterative algorithm of Dobšíček–Johansson–Andersson–Nori requires only a single ancilla qubit and re-preparation of $|\psi\rangle$ each round, useful on shallow hardware.

Prony-style adaptive QPE. Adaptive strategies (e.g. robust phase estimation, Fourier Prony) vary the controlled-unitary exponent dynamically and require fewer long-range controlled rotations, trading circuitry for classical post-processing.

17.8 Worked example ($\phi = 0.101(2)$)

Take $n = 3$ and $\phi = \frac{5}{8} = 0.101_2$.

$$
\begin{aligned}
|\Phi_1\rangle &= \tfrac{1}{2^{3/2}} \sum_{k=0}^{7} |k\rangle\,|\psi\rangle \\
|\Phi_2\rangle &= \tfrac{1}{2^{3/2}} \sum_{k} e^{2\pi i k(5/8)}\,|k\rangle\,|\psi\rangle \\
|\Phi_3\rangle &= |101\rangle\,|\psi\rangle \quad \text{(exact bin. phase)} \\
\text{Measurement} &\;\Rightarrow\; 101\ (=5)
\end{aligned}
$$

Because $2^3\phi = 5$ is an integer, the estimate is perfect.

17.9 Applications

(1) *Order finding*: Estimating the phase of the modular-multiplication operator recovers the order r such that $a^r \equiv 1 \pmod{N}$ (Shor's factoring algorithm).
(2) *Hamiltonian eigenvalue estimation*: With $U = e^{-iHt}$ and $|\psi\rangle$ an eigenstate of H, $\phi = E\,t/2\pi$ yields the energy E.
(3) *Amplitude estimation*: The QFT^{-1} on a two-dimensional rotation operator gives quadratic speed-ups in Monte-Carlo style algorithms.

Lecture 18: Order Finding and Integer Factoring

18.1 Preliminaries

Definition 18.1 (Multiplicative order)**.** For coprime a, N ($\gcd(a, N) = 1$) the *order of a modulo N* is the least $r > 0$ with $a^r \equiv 1 \pmod{N}$.

Definition 18.2 (Order-Finding Problem)**.** Given integers $N > 1$ and a coprime to N, output $\mathrm{ord}_N(a)$.

Remark 18.1 (Classical hardness)**.** No classical algorithm is known that solves OrderFinding or Factoring in $\mathrm{poly}(\log N)$ time; the best general methods run in sub-exponential $\exp\big(\tilde{\mathcal{O}}((\log N)^{1/3})\big)$ time.

18.2 Order and factoring

Proposition 18.1 (Order $\Rightarrow$ factor)**.** *If r is even and $a^{r/2} \not\equiv -1 \pmod{N}$, then $\gcd\big(a^{r/2} \pm 1, N\big)$ is a non-trivial divisor of N.*

Proof. Because $a^r \equiv 1$ we have $\big(a^{r/2} - 1\big)\big(a^{r/2} + 1\big) \equiv 0 \pmod{N}$. Neither factor is $0 \bmod N$ by hypothesis, yet their product is; therefore each shares a non-empty proper divisor with N. □

Example 18.1 (Toy instance $N = 15, a = 2$)**.** $2^4 \equiv 1 \pmod{15} \Rightarrow r = 4$. Then $2^2 = 4$ and $\gcd(4 - 1, 15) = 3,\ \gcd(4 + 1, 15) = 5$ factor 15.

18.3 Quantum order-finding via phase estimation

Let $m = \lceil \log_2 N \rceil$ so that $N < 2^m$. Define the modular-multiplication unitary

$$U_a : |x\rangle \longmapsto |a\,x \bmod N\rangle\,, \qquad x \in \{0, \ldots, N - 1\}.$$

Because a is coprime to N, U_a acts as a permutation and hence is unitary on an m-qubit Hilbert space.

18.3.1 Eigen-decomposition.

For each $s \in \{0, \ldots, r - 1\}$ the "Fourier" state $|\Psi_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-2\pi i s k / r}\, |a^k \bmod N\rangle$ satisfies $U_a\,|\Psi_s\rangle = e^{2\pi i s/r}\,|\Psi_s\rangle$. Thus estimating the phase of $|\Psi_s\rangle$ amounts to estimating s/r.

Algorithm 5 Quantum order-finding — outputs the order r of a mod N with high probability

Require: Coprime integers a, N; precision $n \geq 2m$ such that $2^n \geq N^2$

1: Initialise the *phase* register in $|0^n\rangle$ and the *target* register in $|1\rangle$ (i.e. $|a^0\rangle$)
2: Apply $H^{\otimes n}$ to the phase register, obtaining $\frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} |k\rangle\,|1\rangle$
3: **for** $j \leftarrow 0$ **to** $n-1$ **do**
4: Apply controlled-$U_a^{2^j}$ with qubit j as the control ▷ costs one modular exponentiation $a^{2^j} \bmod N$
5: Apply inverse QFT, $\mathrm{QFT}_{2^n}^{-1}$, to the phase register
6: Measure the phase register, obtaining $y \in \{0, \ldots, 2^n - 1\}$ and set $\tilde{\theta} \leftarrow y/2^n$
7: Use the continued-fraction algorithm to recover the closest fraction s/r with $0 < r < N$

18.3.2 State evolution detail

After the controlled-unitaries the joint state is

$$\tfrac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} |k\rangle\,|a^k \bmod N\rangle = \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} \big(\tfrac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e^{2\pi i k s/r}\,|k\rangle\big)\,|\Psi_s\rangle\,.$$

Measuring $|\Psi_s\rangle$ collapses the phase register to a Dirichlet kernel centred at s/r, exactly as in QPE (cf. Lecture 17). The choice $n \geq 2m$ guarantees that rounding $\tilde{\theta}$ with a continued-fraction expansion finds the correct r with probability $\geq 1 - \mathcal{O}(1/N)$.

18.4 From order to factors: Shor's algorithm

Algorithm 6 Shor's factoring algorithm — factors N in expected polylogarithmic time

1: Select a random $a \in \{2, \ldots, N-1\}$
2: $g \leftarrow \gcd(a, N)$ ▷ trivial factor check
3: **if** $g > 1$ **then return** g
4: Run Algorithm 5 to obtain $r = \mathrm{ord}_N(a)$
5: **if** r is odd **or** $a^{r/2} \equiv -1 \pmod N$ **then**
goto Step 1 ▷ restart
6: **else**
7: $p \leftarrow \gcd\big(a^{r/2} - 1, N\big),\ \ q \leftarrow \gcd\big(a^{r/2} + 1, N\big)$
8: **return** (p, q)

Theorem 18.1 (Complexity)**.** *Algorithm 6 factors N using $\mathcal{O}(\log^2 N)$ qubits, $\mathcal{O}(\log^3 N)$ primitive gates, and succeeds with probability bounded below by a constant $> 1/2$.*

Idea. $\lceil 2\log_2 N \rceil$ phase qubits suffice ($n = 2m$). Each controlled modular-exponentiation costs $\mathcal{O}(\log^2 N)$ gates, repeated n times, and the QFT takes $\mathcal{O}(n^2)$. The probability that a random a yields an even r with $a^{r/2} \not\equiv -1$ is $\geq \frac{1}{2}$ for composite N; thus a constant expected number of iterations suffices. □

18.5 Worked run: $N = 15$

(1) Pick $a = 7$ (coprime to 15). Classically $7^4 \equiv 1 \pmod{15} \Rightarrow r = 4$; the quantum routine would return $\frac{1}{4}$ from the continued-fraction step.
(2) r even; $7^2 = 49 \equiv 4 \not\equiv -1 \pmod{15}$.
(3) Factors: $\gcd(4-1, 15) = \underline{\mathbf{3}}$, $\gcd(4+1, 15) = \underline{\mathbf{5}}$.

18.6 Resource summary

Resource	Shor (quantum)	Best classical (GNFS)
Time	$\mathcal{O}(\log^3 N)$	$\exp\big(\tilde{\mathcal{O}}((\log N)^{1/3})\big)$
Qubits / memory	$\mathcal{O}(\log^2 N)$	$\mathrm{poly}(\log N)$
Oracle calls	n controlled $U_a^{2^j}$	n/a
Bottleneck	modular exponentiation	sieving / linear algebra

Lecture 19: Hidden Variables and Tsirelson's Bound

19.1 Local hidden-variable models

Definition 19.1 (Local hidden-variable (LHV) model)**.** For bipartite measurement settings $a \in \mathcal{A},\ b \in \mathcal{B}$ (held by Alice and Bob), an *LHV model* assumes

(i) a shared random variable $\lambda \in \Lambda$ with distribution $\rho(\lambda) \geq 0,\ \int_\Lambda \rho = 1$;

(ii) deterministic response functions $A : \mathcal{A} \times \Lambda \to \{\pm 1\}$ and $B : \mathcal{B} \times \Lambda \to \{\pm 1\}$;

giving joint expectation

$$E(a,b) \;=\; \int_\Lambda d\lambda\ \rho(\lambda)\, A(a,\lambda)\, B(b,\lambda).$$

Locality means A (resp. B) depends only on a (resp. b), not on the distant choice.

Remark 19.1 (Bell's theorem)**.** Bell (1964) proved that the set of correlations attainable by LHV models is a strict subset of quantum-mechanical correlations.

19.2 The CHSH game

Definition 19.2 (CHSH game)**.** Referee samples independent inputs $x, y \in \{0,1\}$ uniformly; Alice outputs $a \in \{0,1\}$, Bob $b \in \{0,1\}$. They win if $a \oplus b = x \wedge y$.

Proposition 19.1 (Classical (LHV) bound)**.** *Any classical—or LHV—strategy wins CHSH with probability* $\leq \frac{3}{4}$.

Proof. It suffices to check deterministic strategies $a = a(x),\ b = b(y)$. Enumerating the four input pairs shows that at most three constraints can be satisfied simultaneously. Shared randomness only convex-combines deterministic payouts, so the bound persists. □

Example 19.1 (Optimal classical strategy)**.** Output $a = 0,\ b = 0$ always; wins unless $(x,y) = (1,1)$, hence success probability $3/4$.

19.3 Quantum strategy and Tsirelson's bound

Alice and Bob share the Bell state $|\Phi^+\rangle = \frac{1}{\sqrt{2}}\big(|00\rangle + |11\rangle\big)$ and apply Pauli-operator observables

$$A_0 = \sigma_z, \quad A_1 = \sigma_x, \quad B_0 = \tfrac{\sigma_z + \sigma_x}{\sqrt{2}}, \quad B_1 = \tfrac{\sigma_z - \sigma_x}{\sqrt{2}},$$

mapping eigenvalue $+1$ to output bit 0 and -1 to 1.

Definition 19.3 (CHSH operator)**.**

$$\mathcal{M} = A_0 \otimes B_0 + A_0 \otimes B_1 + A_1 \otimes B_0 - A_1 \otimes B_1.$$

Theorem 19.1 (Tsirelson's bound)**.** *For any state ρ on any Hilbert space and any dichotomic observables A_i, B_j with eigenvalues ± 1,*

$$\langle \mathcal{M} \rangle_\rho \;\leq\; 2\sqrt{2}.$$

Equivalently, the maximal CHSH winning probability is

$$p_{\max} = \tfrac{1}{2} + \tfrac{\sqrt{2}}{4} \approx 0.8536.$$

Sketch. Compute $\mathcal{M}^2 = 4\,\mathbb{1} + [A_0, A_1] \otimes [B_0, B_1]$. Because $\|[A_0, A_1]\| \leq 2 \cdot 2$ and likewise for Bob, $\|\mathcal{M}^2\| \leq 8\mathbb{1} \Rightarrow \|\mathcal{M}\| \leq 2\sqrt{2}$. Hence $|\langle \mathcal{M} \rangle| \leq 2\sqrt{2}$. The observables above and $|\Phi^+\rangle$ saturate the bound. □

Corollary 19.1 (Separation of classical, quantum, supra-quantum)**.** $\frac{3}{4} < \frac{1}{2} + \frac{\sqrt{2}}{4} < 1$*, so quantum mechanics violates the classical limit yet does not reach the algebraic maximum* 1 *allowed by no-signalling alone (cf. Popescu–Rohrlich boxes).*

19.4 State-evolution view of the optimal quantum strategy

Using the Jordan-frame decomposition $\{|\Phi^+\rangle, |\Psi^-\rangle\}$ (Bell basis), one verifies

$$\mathcal{M}\ |\Phi^+\rangle = 2\sqrt{2}\ |\Phi^+\rangle, \quad \mathcal{M}\ |\Psi^-\rangle = -2\sqrt{2}\ |\Psi^-\rangle,$$

so performing the CHSH measurements projects onto $|\Phi^+\rangle$ with eigenvalue $+2\sqrt{2}$ with probability 1, giving the optimal expectation.

Lecture 20: The No-Cloning Theorem and Quantum Money

This lecture covers three pillars:

(1) **No-cloning, no-deleting, and no-broadcast** — why quantum information cannot be copied or destroyed unitarily.

(2) **Wiesner's private-key quantum money** — the first protocol (1970) and its security proof.

(3) **Public-key quantum money** — modern proposals, challenges, and open questions.

Throughout, Hilbert spaces are finite-dimensional; $\mathrm{D}(\mathcal{H})$ denotes density operators on $\mathcal{H}$.

20.1 The No-Cloning Theorem

Theorem 20.1 (No-cloning)**.** *There is no unitary U and fixed "blank" state $|0\rangle$ such that $U\big(|\psi\rangle\,|0\rangle\big) = |\psi\rangle\,|\psi\rangle$ for* every *pure state $|\psi\rangle$.*

Proof via inner products. Assume U clones both $|\psi\rangle, |\phi\rangle$:

$$\langle\psi\rangle\,\phi = \langle\psi, 0|\,U^\dagger U\,|\phi, 0\rangle = \langle\psi, \psi|\,|\phi, \phi\rangle = \big|\langle\psi\rangle\,\phi\big|^2.$$

Let $c = \langle\psi\rangle\,\phi$. Then $c = c^2 \Rightarrow c \in \{0, 1\}$. Most pairs of states satisfy $0 < |c| < 1$, contradiction. □

Proof via linearity. Define U that clones both $|0\rangle, |1\rangle$: $U\big(|b\rangle\,|0\rangle\big) = |b\rangle\,|b\rangle\,,\ b \in \{0, 1\}$. Linearity on $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ implies

$$U\big(|+\rangle\,|0\rangle\big) = \tfrac{1}{\sqrt{2}}\big(|0, 0\rangle + |1, 1\rangle\big) \;\neq\; |+\rangle\,|+\rangle = \tfrac{1}{2}\big(|0, 0\rangle + |0, 1\rangle + |1, 0\rangle + |1, 1\rangle\big).$$

So U fails on $|+\rangle$. □

Corollary 20.1 (No-deleting)**.** *No unitary deletes an arbitrary copy: $U\big(|\psi\rangle\,|\psi\rangle\big) = |\psi\rangle\,|0\rangle$ for all $|\psi\rangle$.*

Theorem 20.2 (No-broadcast)**.** *A channel $\mathcal{B} : \rho \mapsto \rho_{AB}$ that marginally preserves ρ on* each *subsystem exists for* all *$\rho \implies$ all states commute. Hence non-commuting quantum information cannot even be* approximately *cloned onto* mixed *outputs.*

Remark 20.1 (Implications)**.** No-cloning underlies BB84 key distribution, quantum authentication, quantum software copy-protection, and quantum money.

20.2 Wiesner's Private-Key Quantum Money (1970)

Definition 20.1 (Banknote)**.** Parameters: security length n. A note is a classical serial s and a quantum register $|\psi_s\rangle = \bigotimes_{i=1}^{n} |r_i\rangle_{b_i}$, where each basis $b_i \in \{Z, X\}$ and bit $r_i \in \{0, 1\}$ are chosen uniformly; $|r\rangle_Z \equiv |r\rangle$, $|r\rangle_X \equiv H|r\rangle$. The bank keeps $\big(s, \{b_i, r_i\}_{i=1}^{n}\big)$ secret.

Algorithm 7 Bank-side verification procedure

Require: Serial number s and purported n-qubit state ρ

Retrieve $(b_i, r_i)_{i=1}^{n}$ from the private database
for $i \leftarrow 1$ **to** n **do** ▷ can be executed in parallel
    Measure qubit i in basis b_i, obtaining r'_i
    if $r'_i \neq r_i$ **then**
        return Reject
return Accept

Proposition 20.1 (Completeness)**.** *Honest notes pass Algorithm 7 with probability* 1*.*

Proposition 20.2 (One-note counterfeiting bound)**.** *Let an adversary start with a single valid note* (s, ρ)*. The maximum probability of producing two states that both pass* independent *verification is at most* $2^{-n}(1 + 2^{-\frac{n}{2}})$*.*

Sketch. Optimal attack: measure each qubit in the Breidbart basis, gaining $\theta \approx 15^\circ$ of basis information, then contrive two guesses. The attack is equivalent to cloning n random BB84 states, whose single-qubit optimal fidelity is $F^\star = \frac{1}{2} + \frac{1}{2\sqrt{2}}$. Chernoff bounds on the Hamming-weight deviation yield the exponent above. □

Remark 20.2 (Exponential security)**.** For $n = 256$, the forging success is $\leq 2^{-256}$—astronomically small.

	Classical serial+signature	Wiesner
Has public verification?	yes	no
Copyable by insider?	trivial	no (quantum)
Needs online bank?	no	yes
Security based on	math assumption	physics (no-cloning)

20.3 Public-Key Quantum Money

Definition 20.2 (Public-key quantum money)**.** A scheme consists of $(\mathsf{Gen}, \mathsf{Ver})$ such that

- $\mathsf{Gen}(1^\kappa) \to (s, |\psi_s\rangle)$ (serial and state) is poly-time quantum.
- Ver_s is a *public* circuit; $\mathsf{Ver}_s |\psi_s\rangle |0\rangle = |\text{Accept}\rangle$.
- For any QPT $\mathcal{A}^{\mathsf{Ver}_s}$ given $(s, |\psi_s\rangle)$, $\Pr[\text{two accepts}] \leq \text{negl}(\kappa)$.

Candidate families:

(a) **Hidden-subspace money** (Aaronson–Christiano). Secret $S \leq \mathbb{F}_2^m$, note $|\psi_S\rangle = \frac{1}{\sqrt{|S|}} \sum_{x \in S} |x\rangle$, public verification uses membership oracles $\chi_S, \chi_{S^\perp}$. *Security*: reduces to list-decoding random low-degree polynomials (unproved).

(b) **Quantum lightning** (Zhandry 2017). Produces states with "unique serials"; collision resistance relies on multi-collision hash assumptions or LWE-type structure.

(c) **Stabiliser money** (Farhi et al.). Notes are random stabiliser states authenticated by Ver_s measuring a commuting Hamiltonian. Broken if too random; secure variant remains open.

Scheme	Verification	Assumption	Status
Wiesner (1970)	private	none	proven
Hidden subspace	public	polynomial-hiding	unbroken*
Quantum lightning	public	hash/LWE	unbroken*
Stabiliser money	public	none	*broken*

Table 1: Snapshot of quantum-money landscape. *no general attack is known.

Lecture 21: Quantum Teleportation

Teleportation transmits an *unknown* quantum state without physically sending the particle. It converts one ebit of entanglement $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ plus two classical bits into an *identity quantum channel*. The protocol is fundamental in quantum networking, fault-tolerant gates, and measurement-based quantum computing.

21.1 Preliminaries

Bell basis:

$$|\Phi^{\pm}\rangle = \tfrac{1}{\sqrt{2}}(|00\rangle \pm |11\rangle), \quad |\Psi^{\pm}\rangle = \tfrac{1}{\sqrt{2}}(|01\rangle \pm |10\rangle).$$

We denote them by

$$|\Phi_{m_1 m_2}\rangle \ (m_1, m_2 \in \{0,1\})$$

with ordering

$$|\Phi_{00}\rangle = |\Phi^+\rangle\,, \quad |\Phi_{01}\rangle = |\Psi^+\rangle\,, \quad |\Phi_{10}\rangle = |\Phi^-\rangle\,, \quad |\Phi_{11}\rangle = |\Psi^-\rangle\,.$$

Pauli corrections:

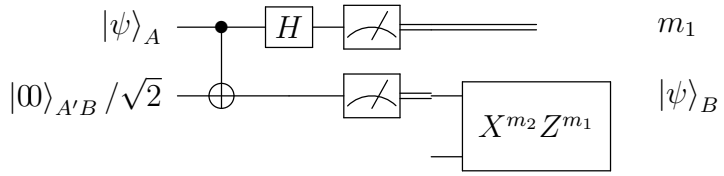
$$Z \equiv \sigma_z,\ X \equiv \sigma_x,\ X^{m_2} Z^{m_1} \in \{I, Z, X, XZ\}$$

indexed by the two classical bits.

21.2 Teleportation task

Definition 21.1 (Teleportation)**.** Let Alice hold an unknown qubit $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$. Alice and Bob share an entangled pair $|\Phi^+\rangle_{A'B}$. Using *only* local operations and two classical bits from Alice to Bob, the goal is for Bob to end up with $|\psi\rangle$ while *no copy* remains with Alice.

21.3 Protocol and state evolution



Algorithm 8 LOCC procedure (Alice $\to$ Bob)

Require: Alice: qubits A (unknown), A' (half EPR); Bob: qubit B.

1: Alice applies CNOT ($A \to A'$) then Hadamard on A.
2: Alice measures AA' in the computational basis, obtaining bits (m_1, m_2).
3: Alice sends (m_1, m_2) to Bob (2 cbits).
4: Bob applies $Z^{m_1} X^{m_2}$ to B.
5: Bob now holds $|\psi\rangle$; Alice's qubits are classical.

Step	Joint state
Init	$\|\psi\rangle_A \|\Phi^+\rangle_{A'B} = \frac{1}{\sqrt{2}}(\alpha \|0\rangle \|00\rangle + \beta \|1\rangle \|11\rangle)$
Bell rewrite	$= \frac{1}{2}\sum_{m_1,m_2\in\{0,1\}} \|\Phi_{m_1 m_2}\rangle_{AA'} \ (X^{m_2} Z^{m_1} \|\psi\rangle)_B$
Measure AA'	Outcome (m_1, m_2) collapses remainder to $X^{m_2} Z^{m_1} \|\psi\rangle$
Bob's correction	$Z^{m_1} X^{m_2} X^{m_2} Z^{m_1} \|\psi\rangle = \|\psi\rangle$

Because the post-measurement state depends only on (m_1, m_2), two classical bits suffice.

21.4 Proofs of optimality and correctness

Proposition 21.1 (Correctness)**.** *In the absence of noise, Algorithm 21.3 outputs Bob's state $\rho_B = |\psi\rangle\langle\psi|$.*

Proposition 21.2 (Two cbits are necessary)**.** *Perfect teleportation sends one arbitrary qubit, i.e. carries one qubit of quantum information $Q = 1$. Holevo's bound limits the quantum capacity of c classical bits to $Q \le c/2$. Hence $c \ge 2$.*

Idea. Replace entanglement with a maximally mixed $\rho_{A'B}$; any LOCC protocol then realises a *classical* channel. Holevo information transmitted in teleportation is one bit per classical bit; saturating requires two bits. □

21.5 Fidelity under noisy entanglement

Assume the shared Bell pair is subjected to a depolarising channel $\mathcal{D}_p(\rho) = (1-p)\rho + \frac{p}{3}\sum_{i=1}^{3} \sigma_i \rho \sigma_i$. Let $|\psi\rangle\langle\psi|$ be the target state. Teleportation fidelity

$$F(p) = \langle\psi|\ \mathcal{E}_{\text{tel}}\big(|\psi\rangle\langle\psi|\big)\ |\psi\rangle = 1 - \tfrac{2p}{3}.$$

A fully mixed pair ($p = 3/4$) gives $F_{\text{classical}} = 2/3$, matching the optimal measure-and-prepare classical limit.

21.6 Generalisations

Replace Bell basis with $|\Phi_{mn}\rangle = \frac{1}{\sqrt{d}} \sum_{k=0}^{d-1} \omega^{km} |k\rangle |k+n \bmod d\rangle$, $\omega = e^{2\pi i/d}$, and Pauli-like corrections $X|k\rangle = |k+1\rangle$, $Z|k\rangle = \omega^k |k\rangle$. Teleport requires $\log_2 d^2$ classical bits and one maximally entangled qudit pair.

For optical modes, EPR is approximated by two-mode squeezed states; homodyne detection plus displacement operations achieve fidelity $F = \frac{1}{1+e^{-2r}}$ where r is the squeezing parameter.

21.7 Applications

- **Entanglement swapping.** Teleporting one half of an EPR pair produces a new EPR pair between distant parties—crucial for quantum repeaters.
- **Measurement-based QC.** Teleportation is the primitive that propagates logical qubits across a cluster state.
- **Fault-tolerant gates.** Magic-state injection and gate teleportation realise non-Clifford gates via Pauli corrections conditioned on measurement outcomes.

21.8 Resource summary

Resource	Cost per qubit teleported
Entanglement	1 ebit
Classical communication	2 cbits ($A \to B$)
Quantum operation depth (ideal)	1 CNOT + 1H + 1 Pauli
Minimum fidelity (noisy p)	$1 - \frac{2p}{3}$
Optimal classical fidelity	$2/3$

Lecture 22: Quantum Complexity Classes and Their Classical Counterparts

Quantum algorithms define complexity classes that refine, rather than replace, the classical landscape. We review the core families, the best known inclusions, and the main open problems.

*AMScd*P@ >>> BPP@ >>> BQP@ >>> AWPP@ >>> PP@ >>> PSPACE@ >>> EXP
@VVV @. @VVV @. @. @| @|
@. @. EQP @. @. QMA @»> QIP=PSPACE
@. @. @. @. @VVV
@. @. @. @. QRG=EXP

22.1 Classical baselines

Definition 22.1 (Deterministic & probabilistic poly-time)**.** P and BPP are the sets of languages decidable by a deterministic, respectively probabilistic, polynomial-time Turing machine with bounded two-sided error $\varepsilon < \frac{1}{2}$.

Amplification reduces ε exponentially at linear cost.

22.2 Exact and bounded-error quantum poly-time

Uniform circuit families. A language $L \subseteq \{0,1\}^*$ is in BQP if there is a log-space classical Turing machine that on input 1^n outputs a description of an $n^{\mathcal{O}(1)}$-gate quantum circuit C_n such that $\Pr[C_n(x) \text{ accepts}] \geq 2/3$ ($x \in L$), $\leq 1/3$ ($x \notin L$).

Definition 22.2 (EQP, BQP)**.** *Exact* quantum poly-time (EQP) sets the error to 0. BQP is defined above with error $< 1/2$.

Remark 22.1 (Containments)**.**

$$\mathrm{P} \subseteq \mathrm{BPP} \subseteq \mathrm{BQP} \subseteq \mathrm{AWPP} \subseteq \mathrm{PP} \subseteq \mathrm{PSPACE}.$$

The steps:
(a) BPP $\subseteq$ BQP: simulate randomness.
(b) BQP $\subseteq$ AWPP: using postselection and gap-amplifier polynomials (Fortnow–Rogers 1999).
(c) AWPP $\subseteq$ PP $\subseteq$ PSPACE are classical containments.

22.3 Quantum certificates

Definition 22.3 (QMA & QCMA)**.** A language L is in QMA (Quantum Merlin–Arthur) if there exists a poly-time uniform verification circuit V_n such that for some poly $p(n)$:

$$x \in L \implies \exists\, |\psi\rangle \in (\mathbb{C}^2)^{\otimes p(n)} : \Pr[V_n(x, |\psi\rangle) \text{ accepts}] \geq 2/3,$$
$$x \notin L \implies \forall\, |\psi\rangle : \Pr[V_n(x, |\psi\rangle) \text{ accepts}] \leq 1/3.$$

If the witness ψ is restricted to be *classical* (computational basis), the class is QCMA.

Theorem 22.1 (Local-Hamiltonian, Kitaev 2002)**.** *The k-Local Hamiltonian problem is* QMA*-complete for $k \geq 2$. Hence* QMA *is a quantum analogue of* NP*.*

Hierarchy.

$$\text{NP} \subseteq \text{MA} \subseteq \text{QCMA} \subseteq \text{QMA} \subseteq \text{PP}.$$

QCMA $\subseteq$ QMA is trivial; QMA $\subseteq$ PP follows from the Feynman–Kitayev history construction summed by a PP machine.

22.4 Interactive proofs

Definition 22.4 (QIP)**.** QIP is the set of languages having a polynomial-round quantum interactive proof with completeness c and soundness s, $c - s \geq 1/\text{poly}$.

Theorem 22.2 (Jain–Ji–Upadhyay–Watrous 2010)**.** QIP = PSPACE.

Remark 22.2. Constant-round subclasses satisfy QIP(3) = QIP (Kitaev–Watrous); QIP(2) = QAM $\subseteq$ PP.

Multiple provers. MIP* = RE (2020) shows entangled provers are *more* powerful; nevertheless single-prover equality with PSPACE persists.

22.5 Zero-knowledge, refereed games, and beyond

Class	Quantum analogue of	Status
QSZK	SZK	QSZK $\subseteq$ QIP(2)
QAM	AM	QAM $\subseteq$ PP
QRG	RG	QRG = EXP

22.6 Oracle and relativised separations

- **BQP vs NP.** Bernstein–Vazirani (1997) built an oracle A with $\mathrm{BQP}^A \not\subseteq \mathrm{NP}^A$. Later, there are oracles with the reverse separation.
- **QCMA vs QMA.** Aaronson–Kuperberg (2007) give an oracle where $\mathrm{QCMA}^A \neq \mathrm{QMA}^A$.
- **BQP vs PH.** Relative to a random oracle, BQP is *likely* not in the polynomial hierarchy.

22.7 Summary table

Class	Definition type	Complete problem	Best upper bound
EQP	exact quantum P-time	parity of a permutation	BQP
BQP	bounded-error quantum P-time	order-finding	PP
QCMA	quantum verify / classical proof	group non-membership	QMA
QMA	quantum proof	k-Local Hamiltonian	PP
QIP	poly-round QIP	quantum circuit distinguish.	PSPACE
QSZK	zero-knowledge	quantum state distinguish.	QIP(2)
QRG	quantum refereed games	non-empty gap games	EXP

ASTRA
BOCCONI