ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 345
Logic & Computability

99% AI Generated Notes

Ali Emre Kaya

FALL 2024

Contents

1 1 - Introduction to Logic and Computability

1.1 Compass and Straightedge

The compass and straightedge construction method, first systematically explored by ancient Greek mathematicians, particularly in Euclid's "Elements" around 300 BCE, represents one of the foundational tools in geometric construction. A compass is used to draw circles and arcs, while a straightedge (unlike a ruler, without measurements) is used to draw straight lines between points. These simple tools became fundamental to Greek mathematics and geometry because they allowed mathematicians to construct perfect geometric shapes and solve complex mathematical problems without measurement. The Greeks considered these constructions to be "pure" since they relied solely on logic and precision rather than measurement. Their importance extends beyond practical applications - they helped establish the foundations of abstract mathematical thinking and led to significant discoveries about the nature of numbers. For instance, the impossibility of certain compass and straightedge constructions (like trisecting an angle or squaring a circle) eventually contributed to our understanding of transcendental numbers and helped bridge the gap between geometry and algebra.

The compass and straightedge method, while elegant and foundational, has its limitations. Ancient Greek mathematicians discovered that certain geometric problems could not be solved using these tools alone. For instance, problems such as trisecting an arbitrary angle, doubling the cube, or squaring the circle were proven impossible to achieve using only a compass and straightedge. These limitations, though initially frustrating, played a critical role in advancing mathematical understanding. They highlighted the need for new approaches and inspired future mathematicians to explore the boundaries of geometry and algebra.

In the 17th century, René Descartes significantly expanded the scope of geometry with the introduction of Cartesian coordinates, which merged algebra and geometry into a unified framework. Descartes' method allowed for geometric problems to be expressed and solved algebraically, overcoming many of the constraints of classical constructions. By using equations to represent curves and shapes, Descartes laid the groundwork for analytic geometry, which in turn opened the door to calculus and modern mathematics. This advancement showed that while the compass and straightedge were powerful tools, the integration of algebra and geometry provided an even more comprehensive and versatile mathematical toolkit.

1.2 Euclids's GCD Algorithm

Euclid's algorithm finds the GCD of two integers a and b by repeatedly applying the division algorithm:

$$a = bq + r$$

Then replace $a \leftarrow b$ and $b \leftarrow r$, repeating until $r = 0$. The GCD is the last non-zero remainder.

Time complexity: $O(\log \min(a, b))$

2 2, 3, 4 - Ordered Logic, Proof, Deduction, Induction

- **Soundness**: A system is sound if all derived theorems are true, meaning no false results are produced.
- **Completeness**: A system is complete if it can prove all true statements expressible within its framework.
- **Satisfiability**: A set of statements is satisfiable if there exists at least one interpretation where all are true.

2.1 Finite Automata

A **finite automaton** is a theoretical machine used to recognize patterns within input. It consists of a finite set of states, an initial state, a set of accepting states, and a transition function that dictates how the automaton moves between states based on input symbols. There are two main types:

- **Deterministic Finite Automaton (DFA)**: For each state and input symbol, there is exactly one transition.
- **Nondeterministic Finite Automaton (NFA)**: For a state and input symbol, there can be multiple possible transitions.

Finite automata are used to recognize regular languages and can be represented graphically with states as nodes and transitions as edges.

2.2 Validity and Expressiveness

- **Interpretation**: An assignment of meanings to symbols in a formal language.
- **Model**: An interpretation where the formula is true.
- **Countermodel**: An interpretation where the formula is false.
- **Valid - Satisfiable**: A valid formula is true in all interpretations, thus satisfiable.
- **Invalid - Unsatisfiable**: An invalid formula is false in all interpretations, thus unsatisfiable.
- **Invalid - Satisfiable**: An invalid formula is false in some interpretations but true in others.

2.2.1 Gentzen's Natural Deduction

Gentzen's Natural Deduction, introduced by Gerhard Gentzen in 1935, is a formal logical system that attempts to mimic how humans naturally reason when constructing mathematical proofs. Unlike earlier formal systems, Natural Deduction allows assumptions to be made temporarily during a proof and then discharged, similar to how mathematicians might say "let's assume x..." and later complete that line of reasoning. The system is built around introduction and elimination rules for logical connectives (like AND, OR, IMPLIES), making it particularly intuitive for writing formal proofs. This approach was revolutionary because it aligned formal logic more closely with actual mathematical practice, making it easier to understand and use than previous systems like Hilbert's axiomatic approach. Gentzen's Natural Deduction remains influential in logic, computer science, and the foundations of mathematics, particularly in type theory and programming language theory.

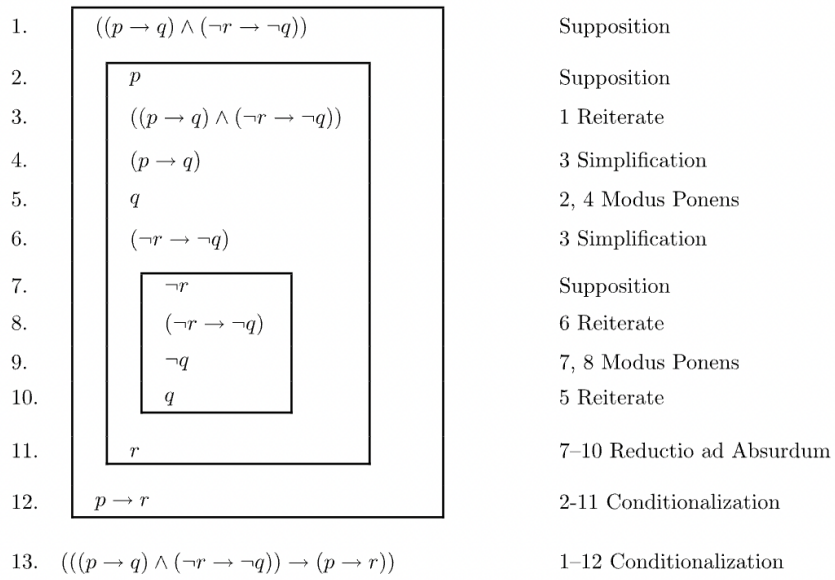


Figure 1: Example natural deduction [1]

2.2.2 Robinson's Resolution

Robinson's Resolution, introduced by J.A. Robinson in 1965, is a powerful method for automated theorem proving in first-order logic. The technique is based on a single inference rule called resolution, which combines and simplifies logical clauses. The key innovation of Resolution is that it operates on clauses in a standardized form (clausal form), and uses unification to match and combine terms. What makes Resolution particularly significant is its completeness - if a contradiction exists in a set of clauses, Resolution will always be able to find it. This made it especially valuable for computer-based theorem proving and laid the groundwork for logic programming languages like Prolog. The beauty of Resolution lies in its simplicity: despite having just one inference rule, it's powerful enough to derive any valid logical conclusion. This efficiency and mechanistic nature made it a cornerstone of automated reasoning and artificial intelligence.

2.3 Propositional (Zeroth Order) Logic

Zeroth-order propositional logic, also known as propositional logic, is a formal system that deals with propositions and their logical relationships. It consists of:

- **Propositions**: Statements that are either true or false.
- **Connectives**: Logical operators such as $\wedge$ (and), $\vee$ (or), $\neg$ (not), $\implies$ (implies), and $\iff$ (if and only if).
- **Syntax**: Defines how propositions and connectives are combined to form valid formulas.
- **Semantics**: Assigns truth values (true or false) to propositions and formulas.

Propositional logic is the foundation of more complex logical systems, such as first-order logic, and is widely used in mathematics, computer science, and philosophy.

2.4 Predicate (First Order) Logic

First-order predicate logic extends propositional logic by introducing quantifiers and predicates, allowing more expressive representations of statements. It consists of:

- **Predicates**: Functions that express properties or relationships between objects (e.g., $P(x)$, $R(x, y)$).
- **Quantifiers**:
  - $\forall$ (Universal quantifier): Indicates a statement applies to all elements (e.g., $\forall x\, P(x)$).
  - $\exists$ (Existential quantifier): Indicates a statement applies to at least one element (e.g., $\exists x\, P(x)$).
- **Domain of Discourse**: The set of all possible objects being considered.
- **Syntax and Semantics**: Extends propositional logic with rules for predicates and quantifiers.

First-order logic is more powerful than propositional logic and is commonly used in mathematics, computer science, and artificial intelligence to express and reason about complex statements involving objects and their relationships.

2.5 Second (High) Order Logic

Second-order logic and **higher-order logic** extend first-order logic by allowing quantification over predicates and functions, not just individual variables. These logics enable reasoning about properties of properties or functions.

- **Second-Order Logic**: Extends first-order logic by allowing quantifiers to range over predicates and relations. For example:

$$\forall P \, (\exists x \, P(x) \implies \exists y \, P(y)).$$

  Here, P is a predicate variable, not an individual variable.

- **Higher-Order Logic**: Extends second-order logic by allowing quantification over higher-level constructs like sets of predicates, functions, or even sets of sets. It generalizes logical reasoning to any level of abstraction.

- **Expressiveness**: These logics are more expressive than first-order logic, allowing statements about properties of predicates and functions, but they are generally not complete or decidable.

- **Applications**: Used in areas like formal semantics, type theory, and higher-level reasoning in mathematics and computer science.

2.6 Order Calculation

- **Zeroth-Order Logic**: In zeroth-order logic, there are no quantifiers. The logic involves only propositional variables or constants, and reasoning is based on truth values (true or false). It can be seen as the most basic form of logic, dealing only with atomic statements.

$$p \vee q$$

  Here, p and q are propositional variables, and no quantifiers are involved. This is zeroth-order logic because it doesn't involve any variables or structures that are quantified over.

- **First-Order Logic**: In first-order logic, quantifiers like $\forall$ (for all) and $\exists$ (there exists) range over individual variables. These variables represent objects in the domain of discourse.

$$\forall x \, (P(x) \implies Q(x))$$

  Here, x is a variable ranging over objects in the domain. This is first-order logic because quantifiers range over individual objects.

- **Second-Order Logic**: Second-order logic extends first-order logic by allowing quantification over predicates or relations, not just individual variables. In second-order logic, we can quantify over properties, relations, or functions that hold between objects.

$$\forall P \, (\exists x \, P(x) \implies \exists y \, P(y))$$

Here, P is a predicate (a relation), and the quantifier ranges over predicates, not individual objects. This is second-order logic because quantifiers range over predicates or relations.

- **Higher-Order Logic**: Higher-order logic generalizes second-order logic by allowing quantification over even more complex constructs such as sets of predicates, functions, or relations between relations. It extends the expressive power to any level of abstraction.

$$\forall S \, \exists P \, \forall x \, (P(x) \in S)$$

Here, S is a set of predicates, and P is a predicate quantified over a set of predicates. This is higher-order logic because it involves quantification over sets or higher-level constructs.

2.7 Linear Time Temporal Logic

Linear Time Temporal Logic (LTL) is a formalism used to reason about the behavior of systems over time. It extends propositional logic by introducing temporal operators to specify how properties evolve along a single linear timeline. Key components include:

- **Propositions**: Statements about the system's state at a given point in time.
- **Temporal Operators**:
  - **X** (Next): Specifies that a property holds in the next state.
  - **G** (Globally): Specifies that a property holds in all future states.
  - **F** (Finally): Specifies that a property will eventually hold.
  - **U** (Until): Specifies that one property holds until another becomes true.

LTL is widely used in the verification of reactive systems, such as software and hardware, to ensure they meet desired temporal properties like safety and liveness.

Consider a system where students take lessons, and if they study, they pass. This behavior can be expressed using LTL as follows:

- Let:
  - T: The student takes a lesson.
  - S: The student studies.
  - P: The student passes.
- Temporal formula:

$$\mathbf{G}(T \implies (\mathbf{F}S \implies \mathbf{F}P)).$$

This means: - Globally ($\mathbf{G}$), if a student takes a lesson (T), then eventually ($\mathbf{F}$) they study (S), and if they study, they will eventually pass (P).

2.8 Computation Tree Logic

Computation Tree Logic (CTL) is a branching-time temporal logic used to reason about the behavior of systems across multiple possible execution paths. Unlike Linear Time Temporal Logic (LTL), CTL allows branching structures, representing different possible futures.

- **Temporal Operators**: CTL combines path quantifiers with temporal operators:
  - Path quantifiers:
    - **A** (For All): Specifies that a property holds on all possible paths.
    - **E** (There Exists): Specifies that a property holds on at least one path.
  - Temporal operators:
    - **X** (Next): A property holds in the next state.
    - **G** (Globally): A property holds in all future states.
    - **F** (Finally): A property eventually holds.
    - **U** (Until): One property holds until another becomes true.
- **Syntax Example**: A CTL formula combines these operators, e.g.,

$$\mathbf{E}(\mathbf{F}P) \quad \text{(There exists a path where } P \text{ eventually holds).}$$

CTL is widely used in model checking to verify properties of systems, such as ensuring that safety or liveness conditions are met across all execution paths.

Consider a system where a student either studies or procrastinates, and if they study, they pass the exam. This behavior can be expressed in CTL as follows:

- Let:
  - S: The student studies.
  - P: The student passes.
  - R: The student procrastinates.
- CTL formulas:
  - $\mathbf{A}(\mathbf{G}(S \vee R))$: On all paths, it is always true that the student either studies or procrastinates.
  - $\mathbf{A}(S \implies \mathbf{F}P)$: On all paths, if the student studies, they will eventually pass the exam.
  - $\mathbf{E}(\mathbf{F}P)$: There exists at least one path where the student eventually passes.

These formulas illustrate how CTL captures branching behaviors, ensuring that certain properties hold across all or some possible execution paths of the system.

3 6 - Verification with Model Checking

3.1 Consensus Theorem

$$AB + \overline{A}C + BC = AB + \overline{A}C$$

Explanation: The term BC is redundant (or unnecessary) in the presence of AB and $\overline{A}C$. The theorem effectively simplifies a Boolean expression by eliminating this redundant term.

Intuition: If AB is true, the value of BC does not change the output because AB already makes the expression true. Similarly, if $\overline{A}C$ is true, BC also has no effect since $\overline{A}C$ already makes the expression true.

This theorem is useful in simplifying logic circuits and optimizing Boolean expressions.

3.2 Skolemization

Skolemization is a process in first-order logic used to eliminate existential quantifiers by introducing Skolem functions. It simplifies formulas into an equisatisfiable form, often used in automated theorem proving.

3.2.1 Skolem Rule

The **Skolem Rule** states that:

- Replace an existentially quantified variable $\exists x$ in a formula with a Skolem function $f(y_1, y_2, \ldots, y_n)$.
- The arguments $y_1, y_2, \ldots, y_n$ of the Skolem function are all universally quantified variables in the scope of $\exists x$.

Example:

$$\forall y\, \exists x\, P(x, y) \quad \text{is transformed to} \quad \forall y\, P(f(y), y).$$

3.2.2 Skolem Algorithm

The **Skolem Algorithm** is a step-by-step procedure to apply Skolemization to a first-order formula:

1. Convert the formula to **prenex normal form** (all quantifiers at the front).
2. Remove existential quantifiers by introducing Skolem functions, following the Skolem Rule.
3. Simplify the formula to a quantifier-free form or keep universal quantifiers if needed.

Skolemization preserves satisfiability but not equivalence, as the transformed formula may no longer be logically equivalent to the original but remains satisfiable if the original is satisfiable.

3.3 Program Verification

Program verification is the process of proving that a program satisfies its specified properties or correctness requirements. It ensures that a program behaves as intended for all possible inputs and executions, using formal methods like logic and mathematics.

3.3.1 Hoare Triples

Hoare triples are a formalism used in program verification to specify and reason about the behavior of programs. A Hoare triple has the form:

$$\{P\}\, S\, \{Q\}$$

where:

- P: The **precondition**, a logical assertion about the state of the program before executing S.
- S: The **program statement** or sequence of statements to be executed.
- Q: The **postcondition**, a logical assertion about the state of the program after executing S.

The meaning of a Hoare triple is: "If the precondition P holds before executing S, then the postcondition Q will hold after S executes, provided S terminates."

Example:

$$\{x > 0\}\, x := x + 1\, \{x > 1\}$$

This asserts that if $x > 0$ before the statement, then $x > 1$ after incrementing x.

Hoare triples are a key component of Hoare logic, a systematic approach for reasoning about program correctness.

4 7 - Computation with Boolean Circuits and Automata

4.1 BDD (Binary Decision Diagram)

Binary Decision Diagrams (BDD) are data structures used to represent Boolean functions in a compact and canonical form. They provide an efficient way to perform operations like function evaluation, equivalence checking, and logical operations.

4.1.1 Reduction Methods

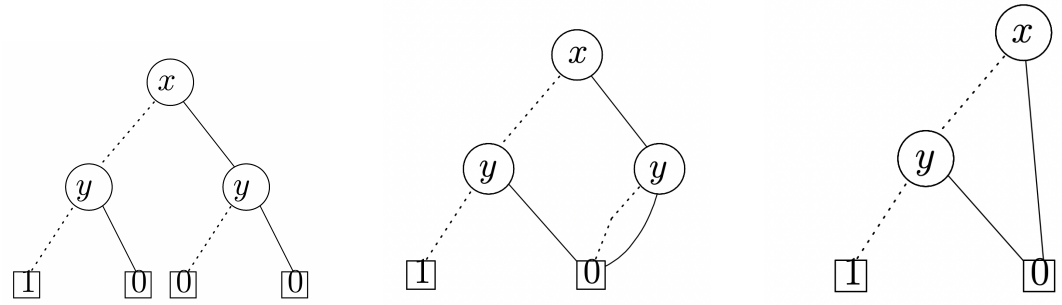
Reduction methods are techniques used to minimize the size of a BDD while preserving the Boolean function it represents. Key reduction methods include:

- **Elimination of Redundant Nodes**: Remove nodes where both edges point to the same child, replacing the node with the child.

- **Merging of Isomorphic Subgraphs**: Combine nodes with identical variables and children into a single node.
- **Variable Ordering**: Optimize the ordering of variables, as the size of a BDD depends heavily on the chosen order.

Applying these reduction methods results in a **Reduced Ordered Binary Decision Diagram (ROBDD)**, which is unique for a given Boolean function and variable ordering.

4.1.2 Reducting BDD



(a) First Form (b) Medium Form (c) Reduced Form

Figure 2: Example from M.T.Sandıkkaya's Slides [2]

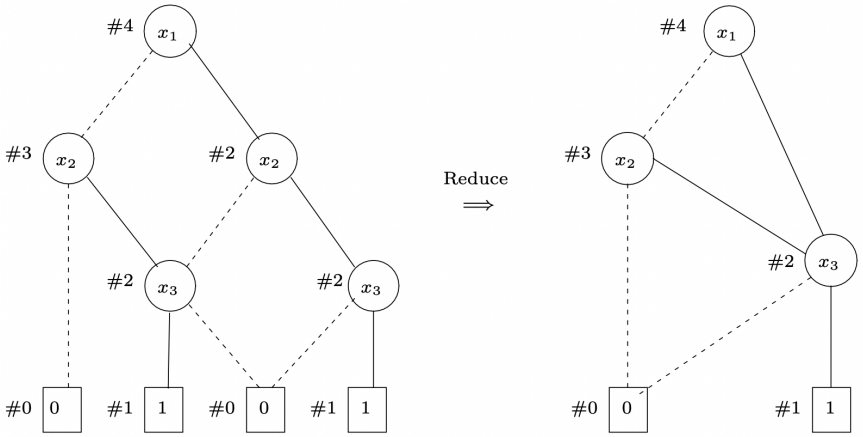


Figure 3: Example from M.T.Sandıkkaya's Slides [2]

https://www.sciencedirect.com/topics/computer-science/binary-decision-diagram [3]

4.1.3 Constructing BDD

https://www.benmandrew.com/articles/bdd [4]

4.2 Finite State Machine

A Finite State Machine (FSM) is a computational model consisting of a finite number of states, transitions between these states, and actions. It is used to design and analyze systems that exhibit sequential behavior. An FSM can be represented as:

- **States:** A finite set of conditions or configurations.
- **Transitions:** Rules for moving from one state to another based on inputs.
- **Input Alphabet:** A set of symbols triggering transitions.
- **Initial State:** The starting state of the FSM.
- **Final States (optional):** One or more states representing acceptance or completion.

FSMs are widely used in digital circuits, software design, and control systems to model predictable and sequential behavior.

5 8 - Turing Machines Limit, Oracles, Reducability

5.1 Limits of Finite Automata

Finite automata are powerful tools for modeling and analyzing regular languages, but they have several limitations:

- **Memory Limitations:** Finite automata have no memory beyond their finite states, making them unsuitable for problems requiring arbitrary storage or recursion.
- **Non-Regular Languages:** They cannot recognize non-regular languages, such as those requiring balanced parentheses or palindrome detection.
- **Complexity of Representation:** As the complexity of the language grows, the number of states required can increase exponentially, making the automaton cumbersome.
- **Inability to Count:** Finite automata cannot perform tasks requiring precise counting beyond a fixed number, as they lack memory to track arbitrary counts.

These limitations highlight the need for more powerful computational models, such as pushdown automata or Turing machines, to handle more complex languages and problems.

5.1.1 Is NDFA more powerful computer than DFA?

Theorem: NFAs (Nondeterministic Finite Automata) and DFAs (Deterministic Finite Automata) are computationally equivalent.

Key Properties:

1. Every DFA is an NFA by definition
2. Every NFA can be converted to an equivalent DFA using the subset construction method
3. If an NFA has n states, the equivalent DFA may have up to 2^n states

Conclusion: While NFAs and DFAs differ in their structure and implementation:

$$L_{DFA} = L_{NFA} = \text{Regular Languages}$$

Therefore, NFAs and DFAs have identical computational power, both being capable of recognizing exactly the class of regular languages in the Chomsky hierarchy.

5.2 Church - Turing Article

The Church-Turing Thesis [5] is a foundational principle in computer science and mathematics, asserting that any function that can be effectively computed by an algorithm can be computed by a Turing machine. It is named after Alonzo Church and Alan Turing, who independently proposed equivalent formalizations of computation in the 1930s.

Key points of the thesis include:

- **Equivalence of Models:** Various models of computation, such as the lambda calculus and Turing machines, are equivalent in their computational power.
- **Limits of Computability:** The thesis establishes a boundary between what is computable and what is not, defining the class of effectively computable functions.
- **Implications:** It serves as the theoretical foundation for modern computer science, influencing areas like algorithm design, complexity theory, and artificial intelligence.

While the Church-Turing Thesis is not a formal theorem, it is widely accepted as a guiding principle in understanding computation.

5.2.1 Universal Turing Machine

Definition: A Universal Turing Machine is a Turing Machine U that can simulate any other Turing Machine M on any input w.

Formal Definition:

$$U(\langle M, w \rangle) = M(w)$$

where:

- U is the Universal Turing Machine
- M is any Turing Machine
- w is an input string
- $\langle M, w \rangle$ represents an encoding of both M and w

Key Properties:

1. **Universality:** Can simulate any Turing Machine
2. **Input Format:** Takes two inputs:

$$\text{Input} = (\text{Description of TM } M, \text{Input string } w)$$

3. **Simulation:** Executes the computation of M on w

Significance:

- Foundation of stored-program computers
- Proves program-data equivalence: Programs $\equiv$ Data
- Theoretical basis for general-purpose computation

Modern Analogy:

$$\text{UTM} \equiv \text{CPU}$$

$$\text{M} \equiv \text{Program Code}$$

$$\text{w} \equiv \text{Program Input}$$

This concept, introduced by Alan Turing (1936), established the theoretical foundation for modern computer architecture.

5.2.2 Limitations of Universal Turing Machine

1. Halting Problem:

$$\nexists \text{ UTM that can decide if arbitrary TM M halts on input w}$$

This is undecidable, proved by Turing himself.

2. Resource Limitations:

- **Time Complexity:** Simulation usually runs slower than direct computation

$$T_{UTM}(n) = O(T_M(n) \log T_M(n))$$

- **Space Overhead:** Requires additional memory for simulation

$$S_{UTM}(n) = O(S_M(n))$$

3. Computational Impossibilities: Cannot solve:

- Non-computable functions (e.g., Busy Beaver function)

- Problems requiring infinite precision real arithmetic
- True randomness generation

4. Physical Limitations:

- Requires infinite tape (theoretical construct)
- Unlimited time assumption
- Perfect reliability assumption

5.2.3 Goldbach's Conjecture

Goldbach's Conjecture is a famous unproven statement in mathematics. It says that every even number greater than 2 can be written as the sum of two prime numbers. For example, $4 = 2 + 2$, $6 = 3 + 3$, and $8 = 3 + 5$.

Computational Aspects: Finding solutions for a single number is relatively easy - computers can do it quickly. Given an even number n, we can find two primes that sum to it in approximately $O(n \log \log n)$ time. This puts the problem in complexity class P (polynomial time).

Complexity Classification: Since the problem is in P, it's automatically in NP too. It's definitely not NP-complete. Verifying a solution (checking if two numbers are prime and sum to n) is even faster, taking only $O(\sqrt{n})$ time.

Despite being computationally simple to check individual cases, proving this conjecture for all numbers remains one of mathematics' great unsolved problems.

5.3 Cantor's Theorem

Cantor's Theorem states that the power set of any set (the set of all its subsets) has a strictly greater cardinality than the set itself. This implies that there are infinitely many sizes of infinity.

5.3.1 Cardinality

Cardinality is a measure of the "size" of a set, describing the number of elements in it. For finite sets, the cardinality is the count of elements. For infinite sets, cardinality is used to compare the sizes of different infinities. Key concepts include:

- **Countable Infinity:** Sets like the natural numbers ($\mathbb{N}$) that can be listed in a sequence.
- **Uncountable Infinity:** Sets like the real numbers ($\mathbb{R}$) that cannot be listed, having a greater cardinality than countable sets.

5.3.2 Diagonalization

Diagonalization is a proof technique introduced by Cantor to show that the real numbers are uncountable. The method involves constructing an element that cannot belong to any given list of elements, proving the set is larger than countable. For example:

- Assume all real numbers in $[0, 1)$ are listed as decimal expansions.
- Construct a new number by changing the n-th digit of the n-th number in the list.
- This new number differs from every number in the list, showing the list cannot include all real numbers.

Cantor's Theorem, cardinality, and diagonalization highlight the surprising and rich structure of infinite sets, revolutionizing mathematics.

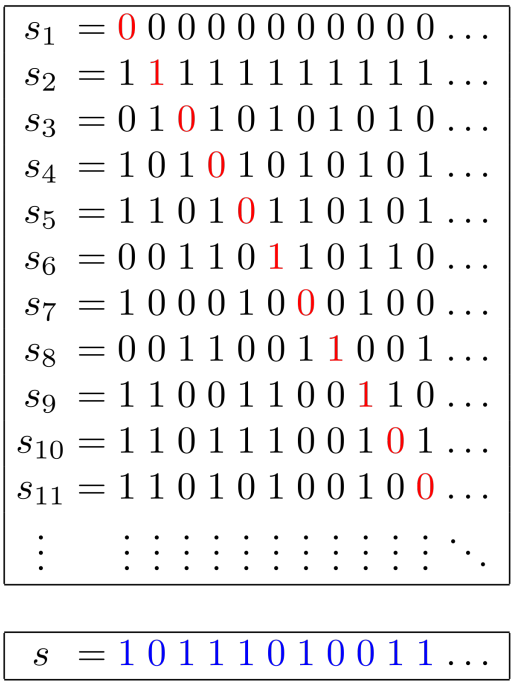


Figure 4: Cantor's Diagonal

5.4 Oracles

In theoretical computer science, an oracle is a hypothetical black-box mechanism that can instantly solve specific decision problems or computational tasks. Oracles are often used to study the relative computational power of different models and complexity classes.

5.4.1 Key Concepts of Oracles

- **Oracle Machines:** A computational model, such as a Turing machine, that can query an oracle to obtain solutions to specific problems as part of its computation.
- **Complexity Classes with Oracles:** Oracles help define relative complexity classes, such as P^A (problems solvable in polynomial time with access to oracle A).
- **Applications:** Oracles are used in theoretical studies, such as examining the relationships between complexity classes like P, NP, and $PSPACE$.

5.4.2 Oracle's Oracle

The concept of "Oracle's Oracle" extends the idea of oracles to a meta-level. It refers to an oracle machine that queries another oracle for solutions to even harder problems. This recursive idea is used to explore hierarchies of complexity, such as:

- **Second-Order Oracles:** Machines that query oracles for solutions, where the oracle itself might depend on another oracle.
- **Relativized Complexity:** Studies how the computational landscape changes when different oracles are assumed to exist.
- **Example:** A P^{NP}-oracle machine uses an NP-oracle to solve P-time problems, but a P^{NP^A}-machine uses an NP^A-oracle, creating a more powerful framework.

Oracles and their recursive extensions are crucial tools for understanding theoretical limits of computation and exploring problems beyond traditional models.

5.5 Diophantine Equations

Diophantine equations are polynomial equations where only integer solutions are sought. These equations are named after the ancient mathematician Diophantus and play a significant role in number theory.

5.5.1 Pythagorean Triples

Pythagorean triples are specific integer solutions to the Diophantine equation:

$$a^2 + b^2 = c^2$$

where a, b, and c are positive integers. These represent the lengths of the sides of right-angled triangles. Examples include:

- $(3, 4, 5)$
- $(5, 12, 13)$

A general formula for generating Pythagorean triples is:

$$a = m^2 - n^2, \quad b = 2mn, \quad c = m^2 + n^2$$

where $m > n > 0$ are integers.

5.5.2 Fermat's Last Theorem

Fermat's Last Theorem states that there are no three positive integers a, b, and c that satisfy the equation:

$$a^n + b^n = c^n$$

for any integer $n > 2$. The theorem, conjectured by Pierre de Fermat in 1637, remained unproven for over 350 years. It was finally proven by Andrew Wiles in 1994 using advanced techniques in algebraic geometry and modular forms.

These topics illustrate the deep connections between integers, geometry, and the fundamental structures of mathematics.

5.6 Gödel's Theorems

Kurt Gödel's Incompleteness Theorems [6] are landmark results in mathematical logic, demonstrating inherent limitations in formal systems capable of arithmetic.

5.6.1 First Incompleteness Theorem

The First Incompleteness Theorem states that in any consistent formal system that is sufficiently expressive to describe basic arithmetic:

- There exist true statements that cannot be proven within the system.
- This reveals the limits of formal systems in capturing all mathematical truths.

Gödel achieved this by constructing a self-referential statement equivalent to "This statement is not provable," showing its truth but unprovability.

5.6.2 Second Incompleteness Theorem

The Second Incompleteness Theorem states that no consistent formal system capable of expressing basic arithmetic can prove its own consistency.

- In other words, a system cannot establish its reliability solely from within itself.
- This highlights that external reasoning or assumptions are needed to assert consistency.

5.6.3 Consistency vs Soundness

- **Consistency:** A formal system is consistent if it does not derive a contradiction (i.e., it cannot prove both a statement and its negation).
- **Soundness:** A formal system is sound if every statement it proves is true in its intended interpretation.
- **Key Difference:** Consistency ensures no contradictions, but soundness additionally guarantees correctness of all proofs. A system can be consistent without being sound if it proves false statements.

Gödel's Theorems demonstrate the profound limitations of formal mathematical systems, shaping modern views on the foundations of mathematics and logic.

5.7 Super Halting Problem

The Super Halting Problem is a theoretical extension of the classic Halting Problem, exploring the limits of computation and decision-making. It involves determining whether a machine capable of solving the Halting Problem (a "super Turing machine") can itself be halted or analyzed.

5.7.1 The Classic Halting Problem

The Halting Problem, proven undecidable by Alan Turing, asks whether an algorithm can determine if another algorithm halts on a given input. Turing demonstrated that no general solution exists for this problem.

5.7.2 Post's Super Halting Problem

Post's formulation of the Super Halting Problem extends this idea:

- Suppose there exists a super Turing machine capable of solving the Halting Problem.
- The question is whether a further machine could decide if the super Turing machine halts on its own inputs.

5.7.3 Implications and Paradoxes

The Super Halting Problem highlights recursive undecidability:

- It reveals that even more powerful computational models cannot escape the limitations of self-referential analysis.
- This reinforces the foundational principles of undecidability in computation.

The Super Halting Problem is a theoretical construct that emphasizes the boundaries of computation and the inherent limits of algorithmic reasoning.

6 9, 10, 11 - Complexities, Pseudorandomness

6.1 Complexity Classes : P, NP, Co-NP, NP-complete, NP-hard, NP-intermediate

Complexity classes categorize computational problems based on their difficulty and the resources required to solve them. Key classes include:

- **P (Polynomial Time):** Problems that can be solved in polynomial time by a deterministic Turing machine. Example: Finding the shortest path in a graph.
- **NP (Nondeterministic Polynomial Time):** Problems for which a solution can be verified in polynomial time by a deterministic Turing machine. Example: The Boolean satisfiability problem (SAT).
- **Co-NP:** The class of problems where the complement of the problem is in NP. Example: Proving that no satisfying assignment exists for a given Boolean formula.
- **NP-complete:** Problems in NP that are as hard as any other problem in NP, meaning that if any NP-complete problem can be solved in polynomial time, all NP problems can be solved in polynomial time. Example: SAT is the first proven NP-complete problem.

- **NP-hard:** Problems that are at least as hard as the hardest problems in NP, but not necessarily in NP themselves. These may include optimization problems. Example: The Traveling Salesman Problem (TSP).
- **NP-intermediate:** Problems believed to lie between P and NP-complete, meaning they are in NP but are neither in P nor NP-complete. Example: Graph isomorphism is conjectured to be NP-intermediate.

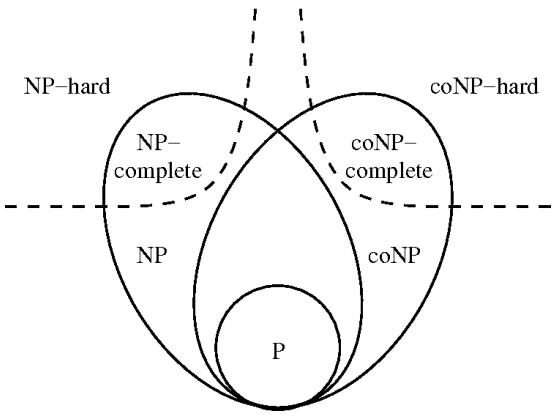


Figure 5: Complexity Classes [7]

Understanding these classes helps in identifying the computational complexity of various problems and guides algorithm design.

6.2 Complexity Spaces Related with Times: L, NL, P, NP, PSPACE, EXPTIME, NEXPTIME, EXPSPACE

Complexity spaces categorize problems based on the resources (time and space) required for their solution, such as memory (space) and computation time.

- **L (Logarithmic Space):** Problems that can be solved using logarithmic space on a deterministic Turing machine. Example: Checking if a graph is connected.
- **NL (Nondeterministic Logarithmic Space):** Problems that can be solved using logarithmic space on a nondeterministic Turing machine. Example: Reachability in a directed graph.
- **P (Polynomial Time):** Problems solvable in polynomial time by a deterministic Turing machine. Example: Sorting a list of numbers.
- **NP (Nondeterministic Polynomial Time):** Problems for which a solution can be verified in polynomial time by a deterministic Turing machine. Example: Boolean satisfiability (SAT).
- **PSPACE (Polynomial Space):** Problems that can be solved using polynomial space, regardless of the time complexity. Example: Solving a generalized game of chess.

- **EXPTIME (Exponential Time):** Problems that require exponential time on a deterministic Turing machine. Example: Solving certain puzzles or games with an exponential number of possibilities.
- **NEXPTIME (Nondeterministic Exponential Time):** Problems that can be solved in exponential time using a nondeterministic Turing machine. Example: Some game theory problems.
- **EXPSPACE (Exponential Space):** Problems that require exponential space to solve. These problems can be very space-intensive but may be solvable in polynomial or exponential time. Example: Certain database query problems.

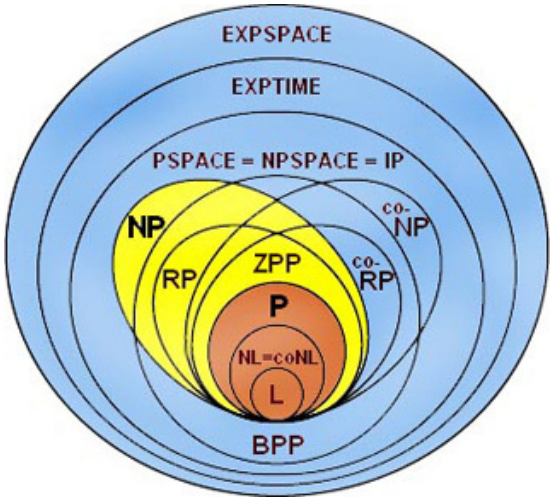


Figure 6: Complexity Classes [8]

These complexity spaces help in classifying problems based on their time and space requirements, contributing to the study of efficient algorithms and computational limits.

Class	Example and Explanation
P	*Sorting numbers using Merge Sort*: Given an unsorted array, sorting it can be done in polynomial time $O(n \log n)$.
NP	*Sudoku solving*: Given a partially filled grid, checking if it can be solved is non-deterministic polynomial. Verifying a solution is polynomial.
coNP	*Checking unsatisfiability of a Boolean formula*: Proving that a formula is not satisfiable involves verifying that no assignment makes it true.
NP-complete	*Traveling Salesman Problem (decision version)*: Determining if there is a route visiting all cities with a total distance less than a given value is NP-complete.
NP-hard	*Halting problem*: Determining if an arbitrary program halts or loops forever. No algorithm can solve this problem for all inputs.
PSPACE	*Quantified Boolean formula (QBF)*: Checking if a QBF with nested quantifiers is true can be solved with polynomial space.
EXPTIME	*Solving chess on an $n \times n$ board*: Determining if the first player has a winning strategy in exponential time based on the board size.
BPP	*Primality testing*: Determining if a number is prime using probabilistic algorithms with high accuracy.
ZPP	*Las Vegas primality testing*: Similar to BPP but guarantees correctness when it terminates, though the runtime may vary.

Table 1: Complexity Classes with Examples

6.3 Savitch's Theorem

Savitch's Theorem provides a relationship between space complexity classes. It states that for any nondeterministic Turing machine using $S(n)$ space to solve a problem, there exists a deterministic Turing machine that can solve the same problem using at most $S(n)^2$ space. Formally:

$$\text{NSPACE}(S(n)) \subseteq \text{DSPACE}(S(n)^2)$$

This implies that if a problem can be solved by a nondeterministic machine in some space $S(n)$, it can also be solved by a deterministic machine in space $S(n)^2$.

6.3.1 Implications

Savitch's Theorem shows that for many problems, nondeterministic space complexity is at most squared in the deterministic space complexity. It also implies that NL = co-NL, where NL is Nondeterministic Logarithmic Space and co-NL is the class of problems whose complement is in NL.

This theorem is important in computational complexity theory as it helps relate nondeterministic and deterministic space complexities, shedding light on the capabilities of different models of computation.

6.4 PCP Theorem

The PCP (Probabilistically Checkable Proofs) Theorem is a fundamental result in computational complexity theory, particularly in the study of NP-hard problems. It characterizes NP as the class of problems for which a solution can be verified probabilistically with very limited access to the proof.

6.4.1 Statement of the PCP Theorem

The PCP Theorem states that every problem in NP has a proof system where:

- A solution (proof) can be verified by reading only a small, constant number of bits of the proof.
- The verifier uses randomness to check the proof, meaning that the verifier makes probabilistic decisions about which parts of the proof to check.
- If the solution is correct, the verifier always accepts, and if the solution is incorrect, the verifier rejects with high probability.

6.4.2 Implications of the PCP Theorem

The PCP Theorem has profound implications:

- It establishes that NP-complete problems can be approximated within certain bounds, leading to the development of approximation algorithms.
- The theorem also plays a key role in the hardness of approximation, showing that for many NP-complete problems, it is difficult to approximate the optimal solution within any reasonable factor.
- It connects probabilistic verification and the hardness of approximation in NP problems.

The PCP Theorem is a cornerstone in the study of computational complexity, particularly in the context of approximation algorithms and hardness results.

6.5 SAT, 3-SAT, MAX-3-SAT Problems

6.5.1 SAT (Satisfiability Problem)

The SAT (Satisfiability) problem asks whether there exists an assignment of truth values (true or false) to the variables in a Boolean formula such that the formula evaluates to true. A Boolean formula is typically represented in Conjunctive Normal Form (CNF), where it is a conjunction of clauses, and each clause is a disjunction of literals.

6.5.2 3-SAT Problem

The 3-SAT problem is a special case of SAT where each clause in the CNF has exactly three literals. Formally, the problem is to determine whether a 3-CNF formula can be satisfied by some assignment of truth values. The 3-SAT problem is NP-complete, meaning that it is computationally intractable to solve it efficiently for large inputs unless $P = NP$.

6.5.3 MAX-3-SAT Problem

The MAX-3-SAT problem is a generalization of 3-SAT where the objective is to find an assignment of truth values to the variables that maximizes the number of satisfied clauses in a 3-CNF formula. Unlike 3-SAT, which asks whether a solution exists, MAX-3-SAT seeks an optimal solution that satisfies as many clauses as possible. This problem is NP-hard because it is at least as hard as 3-SAT.

6.6 MAX-3-SAT's 7/8 Approximation with the PCP Theorem

The MAX-3-SAT problem has a well-known approximation algorithm that guarantees a solution that satisfies at least $\frac{7}{8}$ of the clauses. This result is based on the PCP (Probabilistically Checkable Proofs) Theorem, a fundamental result in complexity theory.

6.6.1 7/8 Approximation Algorithm for MAX-3-SAT

The MAX-3-SAT problem can be approximated using a randomized algorithm that leverages the PCP theorem. The key idea is:

- A random assignment is made to the variables of the 3-CNF formula.
- The algorithm checks only a few bits of the assignment using a probabilistic verifier, as guaranteed by the PCP theorem.
- By using randomness, the algorithm ensures that with high probability, the number of satisfied clauses is at least $\frac{7}{8}$ of the total clauses.

The core of this result is that for a large portion of the clauses, the algorithm can either make them true or identify the clauses that will remain false. By probabilistically checking random bits of the assignment, the verifier can efficiently approximate the maximum number of satisfied clauses.

6.6.2 Connection with the PCP Theorem

The PCP theorem plays a key role in the approximation guarantee for MAX-3-SAT. It provides a framework where the correctness of an assignment can be verified probabilistically with limited access to the proof (the variable assignments). The 7/8 approximation arises from the fact that the verifier in the PCP model has a high probability of accepting valid assignments while rejecting incorrect ones, ensuring that at least 7 out of 8 clauses can be satisfied in the approximation.

6.6.3 Implications of the 7/8 Approximation

The result shows that while solving MAX-3-SAT exactly is NP-hard, a simple probabilistic algorithm can efficiently approximate the solution with a guarantee of at least 7/8 of the clauses being satisfied. This is a significant result in the study of approximation algorithms and hardness of approximation, demonstrating that there is a limit to how well we can approximate certain NP-hard problems like MAX-3-SAT.

6.7 Probabilistic Complexity Classes: BPP, RB, ZPP

Probabilistic complexity classes involve algorithms that make random decisions during computation. These algorithms allow for probabilistic behavior in determining results.

- **BPP (Bounded-Error Probabilistic Polynomial Time):** This class contains decision problems that can be solved in polynomial time by a probabilistic Turing machine, with an error probability of at most $\frac{1}{3}$. BPP algorithms have a high probability of correctness. Example: Randomized primality testing.
- **RB (Randomized Bounded Error):** Similar to BPP, but with specific bounds on the error probability for both correct and incorrect results. It may refer to a broader category of problems within the probabilistic complexity classes, often used interchangeably with BPP in some contexts.
- **ZPP (Zero-Error Probabilistic Polynomial Time):** This class consists of problems solvable in expected polynomial time, where the algorithm has no error in the output (i.e., it always gives the correct result). ZPP algorithms are allowed to use randomization but guarantee a correct answer with no error. Example: Finding the median of a set of numbers using a randomized algorithm.

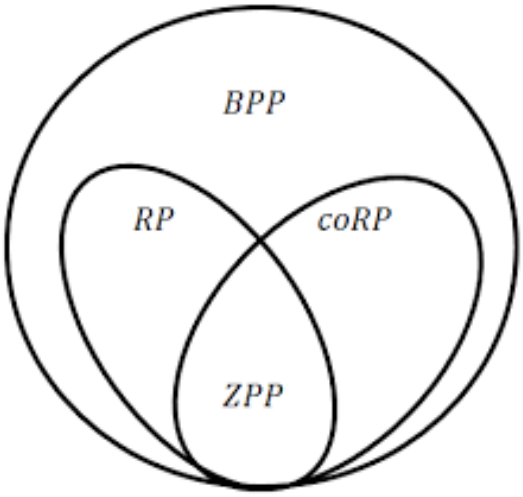


Figure 7: Probabilistic Complexty Classes Relations [9]

BQP algorithm (1 run)		
Correct answer \ Answer produced	Yes	No
Yes	≥ 2/3	≤ 1/3
No	≤ 1/3	≥ 2/3

Figure 8: BQP Algorithm Ratios [10]

These classes help in understanding the power of algorithms that utilize randomness, where BPP algorithms allow bounded error, ZPP guarantees zero error, and RB emphasizes specific error bounds.

The notation $M(x, r)$ generally represents a probabilistic computational model, where:

- x: This denotes the *input* to the machine M. It is the data or problem instance that the machine processes.
- r: This represents the *random bits* used by the machine. These random bits enable the probabilistic behavior of M, allowing it to make randomized decisions during computation.

In this context, $M(x, r)$ models a computation where the output depends on both the input x and the random choices r made by the machine. The randomness r is bounded, meaning the number of random bits used is limited, and the machine's behavior is analyzed with respect to this constraint. This notation is commonly used in studies of probabilistic algorithms, complexity theory, and cryptography.

In binary classification, the confusion matrix evaluates the performance of a model using the following metrics:

- **True Positive (TP):** Cases where the model correctly predicts the positive class.

$$TP = \text{Number of samples correctly classified as positive.}$$

- **True Negative (TN):** Cases where the model correctly predicts the negative class.

$$TN = \text{Number of samples correctly classified as negative.}$$

- **False Positive (FP):** Cases where the model incorrectly predicts the positive class when the true label is negative (Type I error).

$$FP = \text{Number of negative samples misclassified as positive.}$$

- **False Negative (FN):** Cases where the model incorrectly predicts the negative class when the true label is positive (Type II error).

$$FN = \text{Number of positive samples misclassified as negative.}$$

The confusion matrix is typically structured as follows:

	Predicted Positive	**Predicted Negative**
Actual Positive	TP	FN
Actual Negative	FP	TN

Using these values, additional evaluation metrics can be calculated:

- **Accuracy:**
$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$
- **Precision:** (Positive Predictive Value)
$$\text{Precision} = \frac{TP}{TP + FP}$$
- **Recall:** (Sensitivity or True Positive Rate)
$$\text{Recall} = \frac{TP}{TP + FN}$$
- **F1 Score:** Harmonic mean of Precision and Recall
$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

7 12 - Derandomization and Cryptologic Models

7.1 Primality Testing

Primality testing is the process of determining whether a given number is prime or not. Various algorithms exist to test for primality, including deterministic and probabilistic methods.

7.1.1 Fermat's Little Theorem

Fermat's Little Theorem states that if p is a prime number and a is any integer such that $\gcd(a, p) = 1$, then $a^{p-1} \equiv 1 \mod p$. This theorem is often used in primality testing, but it can be fooled by Carmichael numbers, which are composite yet satisfy this congruence for all a.

7.1.2 Pseudoprime

A **pseudoprime** is a composite number that satisfies a particular primality test, giving the impression of being prime, even though it is not. For example, in the case of Fermat's Little Theorem, a number n is considered prime if, for any integer a, the following holds:

$$a^{n-1} \equiv 1 \pmod{n}$$

However, some composite numbers satisfy this equation for certain values of a. These numbers are called pseudoprimes. In other words, pseudoprimes are composite numbers that pass a specific primality test, but they are not actually prime.

Example of a Pseudoprime

A well-known example of a pseudoprime is 341, which is a pseudoprime to base 2. To verify this:

$$2^{340} \equiv 1 \pmod{341}$$

Even though this condition holds, 341 is not prime because it factors as $341 = 11 \times 31$. Thus, it is a composite number that behaves like a prime number under the Fermat test for base 2.

Primes vs. Pseudoprimes

The key distinction between primes and pseudoprimes lies in the behavior across all possible primality tests. While primes satisfy primality tests for all bases a that are coprime to n, pseudoprimes pass the test only for specific bases.

Fermat's Little Theorem and Primes

For a prime number p, Fermat's Little Theorem states that for any integer a that is coprime to p:

$$a^{p-1} \equiv 1 \pmod{p}$$

This holds true for all integers a that are coprime with p.

Pseudoprimes and Their Limitations

A pseudoprime, on the other hand, may satisfy Fermat's Little Theorem for one or more bases but is still a composite number. Therefore, pseudoprimes give a false positive for primality under specific tests, making them unreliable for confirming whether a number is prime.

7.1.3 Carmichael Numbers

Carmichael numbers are composite numbers that pass certain tests for primality, such as Fermat's Little Theorem, which is often used in primality testing. Specifically, Carmichael numbers are composite numbers that satisfy the condition $a^{n-1} \equiv 1 \mod n$ for all integers a coprime to n, even though they are not prime. This makes them "pseudoprimes" that fool certain primality tests. Example: $341 = 31 \times 11$ is a Carmichael number.

A Carmichael number is a special type of pseudoprime that satisfies Fermat's Little Theorem for all integers a that are coprime to the number. In other words, Carmichael numbers are absolute pseudoprimes.

7.2 Derandomization

Derandomization is a crucial concept in computational complexity and algorithm design. It involves converting a probabilistic algorithm (which uses randomness) into a deterministic algorithm that does not require random choices but still achieves similar results. The importance of derandomization can be understood through the following points:

- **Efficiency**: Randomized algorithms often offer elegant solutions to complex problems, but derandomization can result in more efficient algorithms by eliminating the need for randomness, which can be computationally expensive in practice.
- **Predictability**: Derandomized algorithms provide more predictable and stable behavior since they do not rely on random choices, which can vary with each execution.
- **Practical Applicability**: Many real-world systems (e.g., cryptography, hardware systems, distributed computing) require deterministic algorithms to ensure reliability and security. Derandomization makes probabilistic methods applicable in such settings.
- **Improved Understanding**: Derandomization helps to improve our theoretical understanding of the computational limits of various classes of problems and provides insights into the complexity of randomized algorithms.
- **Complexity Class Separation**: Derandomization contributes to the understanding of complexity class relationships (e.g., whether P = BPP), providing key insights into the boundaries between deterministic and probabilistic complexity classes.

In summary, derandomization plays an essential role in improving algorithm efficiency, ensuring reliability in practical systems, and deepening our understanding of computational complexity.

7.2.1 Random Path Finding vs Reingold's Logspace Path Finding

Random path finding refers to algorithms that use random decisions to explore paths or search spaces, often applied in graph traversal or related problems. These algorithms may rely on randomness to make decisions efficiently in large or complex search spaces.

Reingold's logspace pathfinding algorithm is a breakthrough in derandomization. It shows that finding a path between two vertices in an undirected graph can be done in logarithmic space, and importantly, it is a deterministic process, removing the need for randomness. This result demonstrates that certain problems, previously thought to require randomness, can be solved efficiently using only a small amount of memory (logarithmic space), thus achieving derandomization for pathfinding problems.

7.3 Cryptography

Cryptography is the study of securing communication and information through mathematical techniques, ensuring confidentiality, integrity, authenticity, and non-repudiation.

7.3.1 Shannon's One-Time Pad Proof

Shannon's One-Time Pad is a cryptographic system that provides perfect secrecy, meaning the ciphertext gives no information about the plaintext without the key. The one-time pad uses a key that is as long as the message and is used only once. The key is combined with the plaintext using bitwise XOR. Shannon proved that if the key is truly random, at least as long as the message, and used only once, then the encryption is unbreakable, and the ciphertext is statistically independent of the plaintext. This guarantees perfect secrecy but requires a secure method of key distribution.

7.3.2 CPRG & OWF

CPRG (Cryptographically Pseudorandom Generator): A CPRG is an algorithm that generates pseudorandom sequences that appear indistinguishable from truly random sequences to any efficient observer. It is used in cryptography to generate keys or random values needed for encryption, ensuring that the output is unpredictable and secure.

OWF (One-Way Function): An OWF is a function that is easy to compute in one direction but hard to invert. Given an input x, it is computationally easy to compute $f(x)$, but given $f(x)$, it is hard to find x. OWFs are foundational to many cryptographic protocols, such as digital signatures and public-key encryption schemes.

7.3.3 DH & RSA

DH (Diffie-Hellman): The Diffie-Hellman key exchange protocol allows two parties to securely exchange cryptographic keys over a public channel. It is based on the mathematical problem of computing discrete logarithms in a finite group. The security of DH relies on the difficulty of the discrete logarithm problem, making it hard for an adversary to deduce the shared secret key.

RSA (Rivest-Shamir-Adleman): RSA is a widely used public-key encryption algorithm that relies on the difficulty of factoring large composite numbers. It involves generating a public-private key pair where the public key is used to encrypt messages and the private key is used to decrypt them. The security of RSA is based on the hardness of factoring the product of two large primes, which is a computationally difficult problem.

7.3.4 TDOWF

TDOWF (Trapdoor One-Way Function): A TDOWF is a special type of one-way function that is easy to compute in one direction but difficult to invert, except when certain "trapdoor" information is known. The trapdoor allows for the efficient inversion of the function, making it useful for public-key cryptosystems. The most well-known example of a TDOWF is the function used in RSA encryption, where factoring the product of two primes is hard without the trapdoor information (the private key).

8 13 - Computational Learning Theory

Computational Learning Theory is a field that explores the theoretical aspects of machine learning, including the conditions under which learning algorithms can effectively learn from data.

8.1 Learning

Learning in computational theory refers to the process through which algorithms or models improve their performance over time by being exposed to more data. This learning can be supervised, unsupervised, or reinforcement-based, with each type having specific methods and challenges.

8.1.1 Hume's Problem of Induction

(Algorithm can not predict)

Hume's problem of induction refers to the philosophical dilemma regarding the justification of inductive reasoning. Inductive reasoning involves making generalizations from specific observations (e.g., concluding that all swans are white after observing many white swans). Hume argued that we cannot rationally justify the inference that future unobserved instances will be like past observed instances because this reasoning itself is based on past experiences, creating a circular argument. In machine learning, this problem is analogous to the challenge of generalizing from limited data to make predictions about unseen examples.

8.1.2 Occam's Razor

(Give simpliest data, Algorithm should not overfit)

Occam's Razor is a principle that suggests that, when faced with competing hypotheses or models, the one with the fewest assumptions or simplest explanation is preferred. In computational learning, this principle is applied to model selection, encouraging the choice of simpler models that are less likely to overfit the data. Occam's Razor provides a heuristic for finding efficient and generalizable models that balance simplicity and accuracy.

8.1.3 Valiant's PAC (Probably Approximately Correct) Learning

(Algorithm can predict a result with an error)

Valiant's PAC learning framework provides a formal model for understanding machine learning. In PAC learning, the goal is for a learning algorithm to produce a hypothesis that, with high probability, approximates the true function within an error bound, given a sufficient amount of training data. The key components of PAC learning are:

- **Probably**: The algorithm should perform well with high probability.
- **Approximately Correct**: The hypothesis should be close to the true function (within a specified error).

- **Efficient Learning**: The algorithm should learn in polynomial time with respect to the size of the input and the desired accuracy.

PAC learning provides a foundation for understanding the theoretical limits of learning algorithms and the conditions under which they can be expected to perform well.

8.2 Complexity of Learning

The complexity of learning refers to the resources, such as time and number of samples, required for an algorithm to learn a concept from data.

8.2.1 How Many Samples Are Required to Learn an Infinite Class?

In machine learning, especially when dealing with infinite hypothesis classes (such as a class of functions with infinitely many possible hypotheses), it is not necessary to see every possible example or data point to learn an accurate model. Instead, learning can often be achieved with a finite, representative sample.

The number of samples required to learn a concept depends on several factors:

- **VC-dimension (Vapnik-Chervonenkis dimension)**: This is a measure of the capacity of a hypothesis class. The higher the VC-dimension, the more samples are needed to learn the class accurately. A hypothesis class with a high VC-dimension can fit a wide variety of data patterns, but it also requires more data to ensure generalization.

- **Generalization vs. Overfitting**: A learning algorithm must balance between fitting the training data well (which may lead to overfitting) and generalizing to unseen data. With a sufficiently large number of diverse samples, a learning algorithm can generalize well without needing to see all possible data.

- **Sample Size and Confidence**: For infinite hypothesis classes, the number of samples needed depends on the desired level of confidence and the acceptable error. As the sample size increases, the algorithm can make more reliable inferences about the true function, even without seeing every possible example.

Thus, learning from a data set does not require reading all the data; a sufficiently large, representative sample can often provide the necessary information for accurate learning. This idea is a central theme in the PAC (Probably Approximately Correct) learning model, where the goal is to approximate the target concept well with high probability based on a finite sample of data.

9 14 - Quantum Computing (BQP), Quantum XOR-gate, Shor's A2G Algorithm

Quantum computing leverages the principles of quantum mechanics to perform calculations in ways that classical computers cannot. Quantum computing uses quantum bits (qubits) and quantum gates to perform operations on data.

9.1 ⟨bra|-|ket⟩ Notation

The bra|ket notation is a standard way of representing quantum states in quantum mechanics and quantum computing. It consists of two parts:

- **Ket notation ($|\psi\rangle$)**: Represents a quantum state vector, where ψ is the quantum state.
- **Bra notation ($\langle\psi|$)**: Represents the conjugate transpose of a quantum state vector, used in inner products and other operations.

This notation is essential for describing the state of qubits, quantum operations, and measurements in quantum algorithms.

9.2 Pauli Matrix

Pauli matrices are a set of 2x2 matrices that are used to represent quantum gates and operations on qubits. They are named after the physicist Wolfgang Pauli and are represented as:

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

The Pauli matrices are fundamental in quantum mechanics and are used to describe rotations and measurements in quantum systems. They are also key components of quantum gates like the NOT gate.

9.3 Hadamard Transform

The Hadamard transform (or Hadamard gate) is a quantum gate that operates on a single qubit and creates superpositions of quantum states. The Hadamard gate is represented by the matrix:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

When applied to a qubit, it transforms the state into an equal superposition of the $|0\rangle$ and $|1\rangle$ states. The Hadamard transform is often used in quantum algorithms to create superpositions, which are a key resource for quantum parallelism.

9.4 Quantum Gates

Quantum gates are the fundamental operations in quantum computing that manipulate qubits. These gates are the quantum analogs of classical logic gates but operate based on the principles of quantum mechanics. Some common quantum gates include:

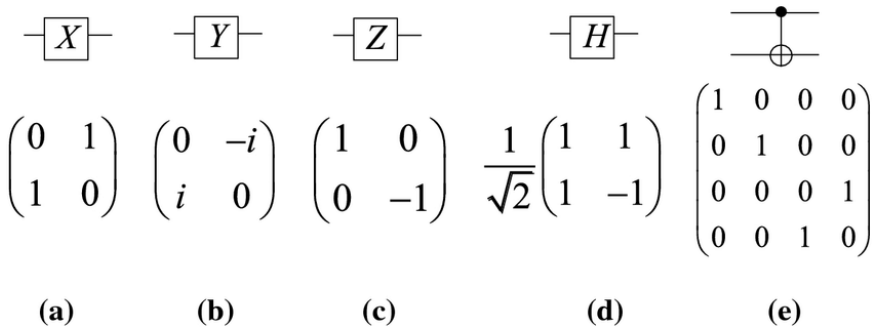


Figure 9: X Y Z pauili, hadamart, cnot matricies

a tensor product of two spaces produces another space consisting all linear combinations [2]

$$|X\rangle \oplus |Y\rangle = |XY\rangle$$

– – – – –

$$|X\rangle|Y\rangle = |XY\rangle$$

9.4.1 Hadamard Gate (H)

Creates superpositions, as explained above.

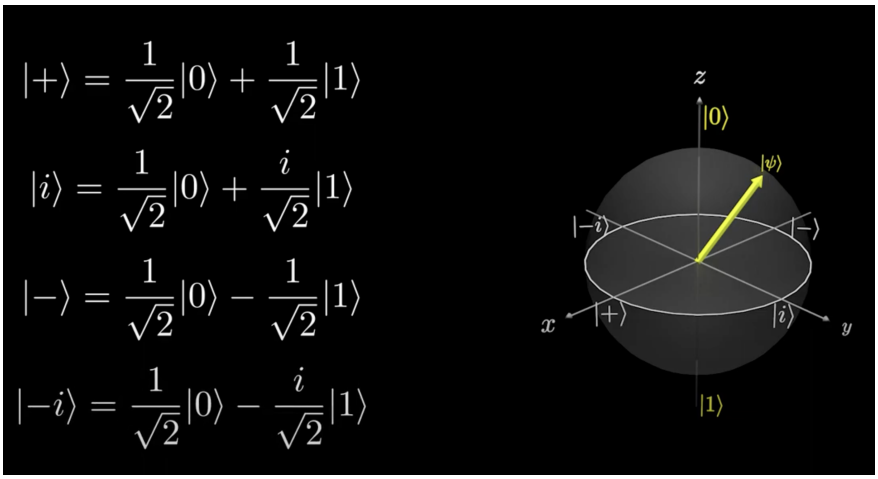


Figure 10: Hadamard Symbols

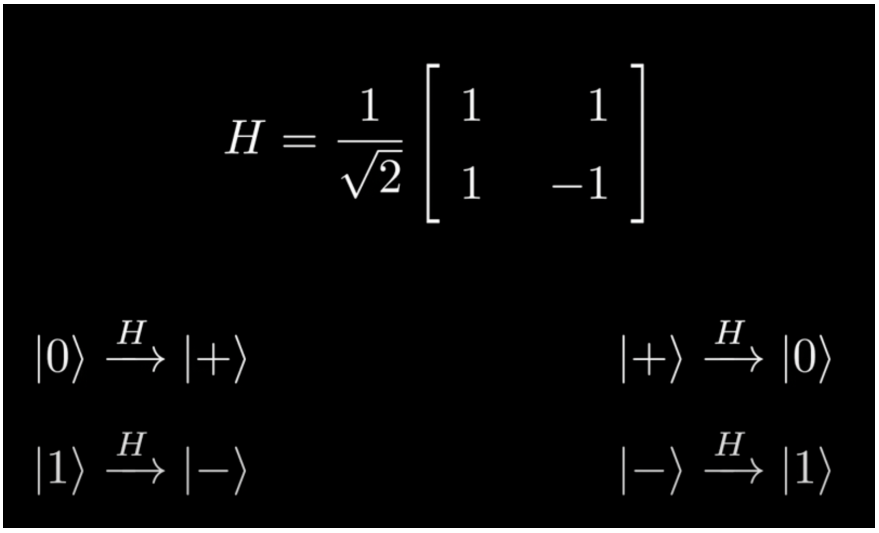


Figure 11: Hadamard Symbols

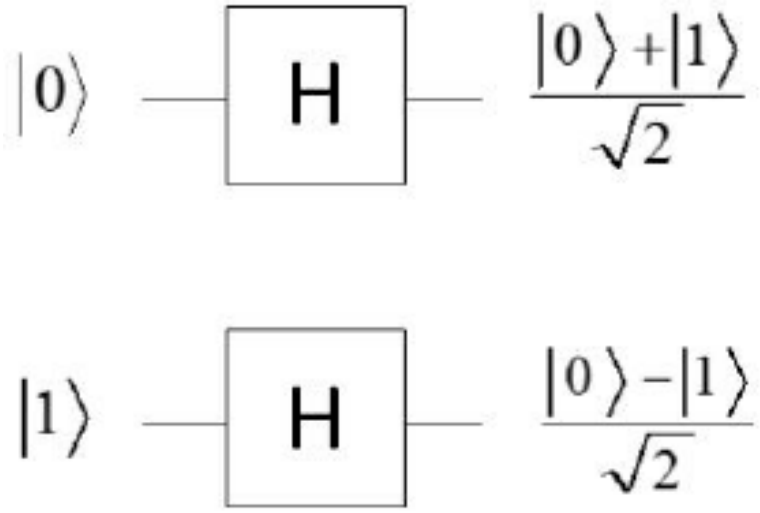


Figure 12: Hadamard Symbols

9.4.2 Entanglement

Entanglement is crucial in quantum computing because it enables qubits to be in a state where their information is deeply interconnected, even when separated by large distances. This phenomenon allows quantum computers to perform complex calculations much faster than classical computers. Through entanglement, quantum algorithms can process multiple possibilities simultaneously, harnessing parallelism and exponentially increasing computational power. It plays a key role in algorithms like Shor's algorithm for factoring large numbers and Grover's algorithm for searching unsorted databases. Additionally, entanglement is central to quantum error correction, quantum teleportation, and quantum cryptography, making it foundational for the development of secure and efficient quantum technologies.

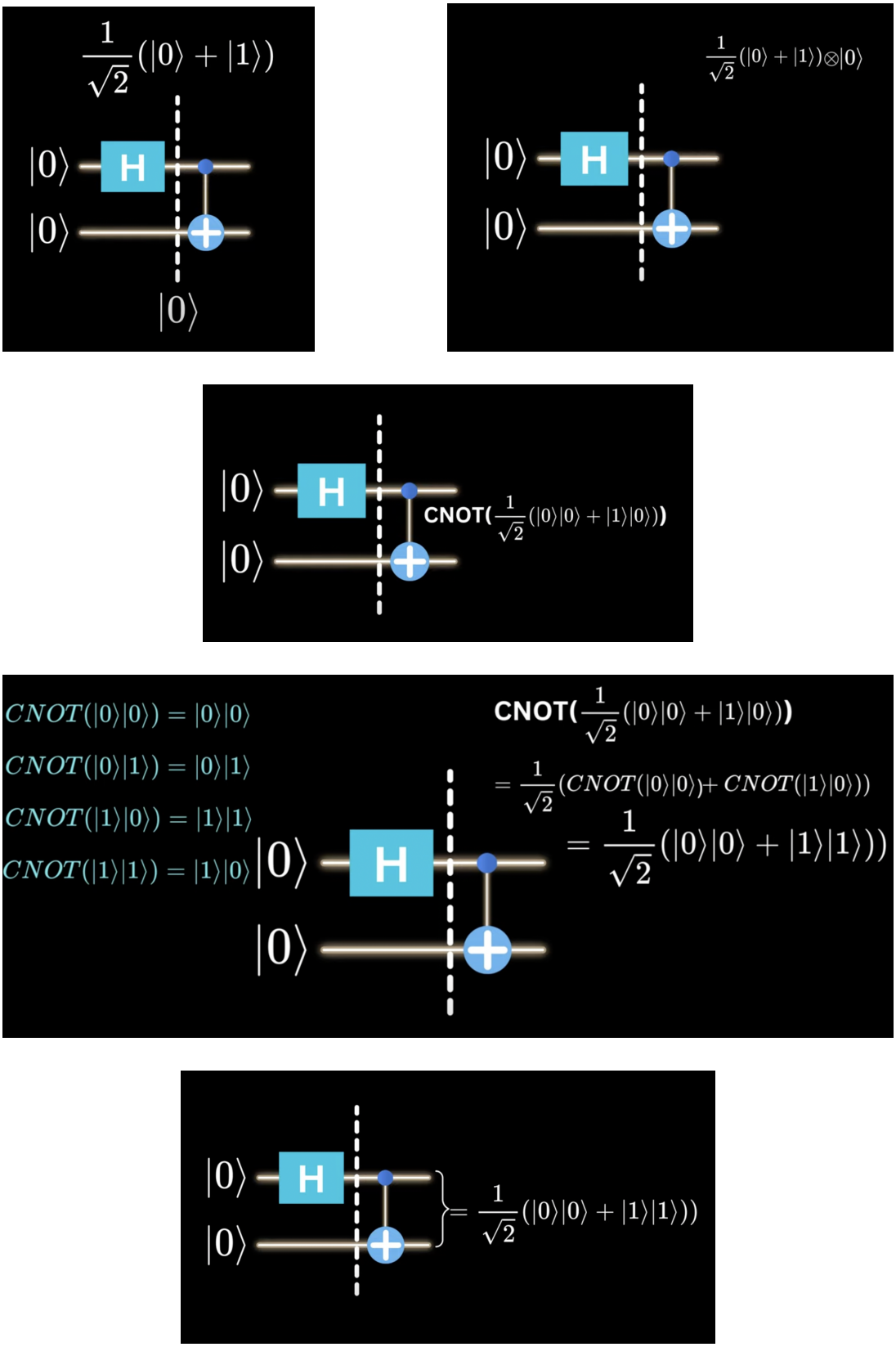


Figure 13: Entanglement, Hadamard Gate, Bell State [11]

9.4.3 Pauli Gates (X) (Y) (Z)

X = Equivalent to the classical NOT gate, flipping the state of a qubit.
Z = Applies a phase flip to the state of a qubit.

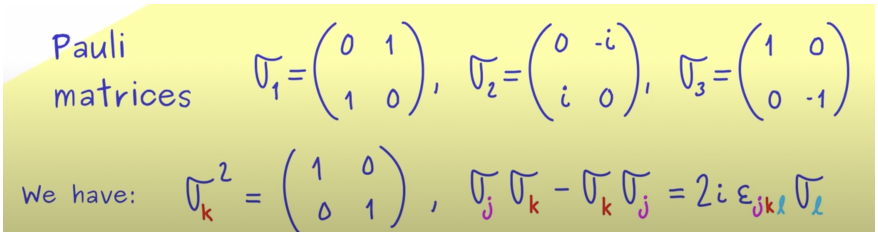


Figure 14: Pauli Matricies [12]

9.4.4 CNOT Gate (CX)

A two-qubit gate that flips the state of the target qubit if the control qubit is in state $|1\rangle$.

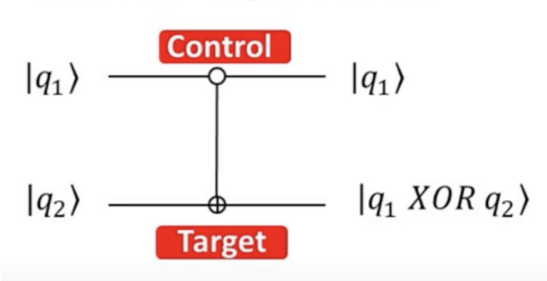


Figure 15: CNOT Gate [13]

$$|00\rangle \rightarrow |00\rangle$$
$$|01\rangle \rightarrow |01\rangle$$
$$|10\rangle \rightarrow |11\rangle$$
$$|11\rangle \rightarrow |10\rangle$$

Figure 16: CNOT Table on 2 qubit basis [13]

$$U_{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$|01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \qquad |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

Figure 17: CNOT Matrix Representation [13]

2-Qubit Swap

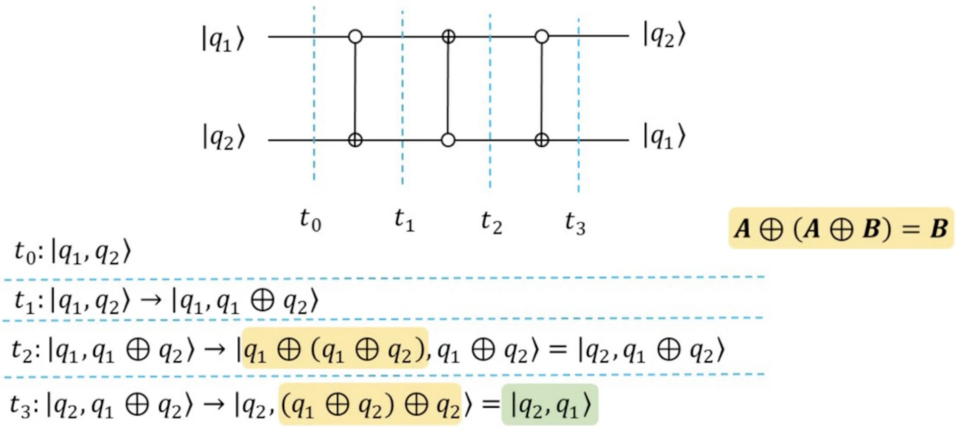


Figure 18: 2-qubit swap

3-Qubit System : Toffoli Gate Controled by CNOT gate

$$(q_1, q_2, q_3) \rightarrow (q_1, q_2, q_3 \oplus q_1 q_2)$$

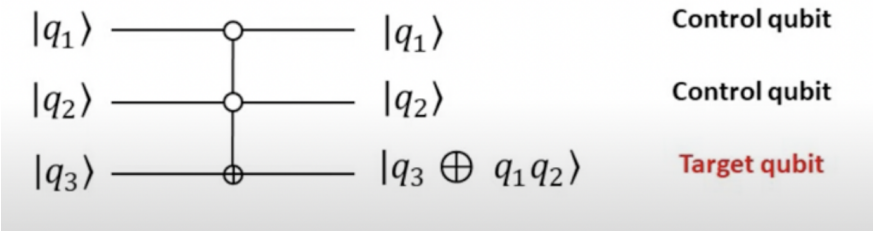


Figure 19: Toffoli Gate [13]

Inputs			Outputs		
q_1	q_1	q_3	q_1'	q_2'	q_3'
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

Figure 20: Toffoli Table [13]

Quantum gates are used to create quantum circuits that solve problems that are hard or impossible for classical computers to address.

9.5 Shor's Algorithm

Shor's algorithm is a quantum algorithm that efficiently solves the integer factorization problem, which is a computationally hard problem for classical computers. Shor's algorithm has the potential to break many classical cryptographic systems (e.g., RSA encryption) by factoring large numbers exponentially faster than the best-known classical algorithms. The algorithm relies on quantum Fourier transform and the properties of quantum parallelism to find the period of a function, which can be used to factorize large integers in polynomial time. This is a key example of how quantum computing can outperform classical computing for certain tasks.

9.5.1 Quantum Parallelism

Quantum parallelism is a phenomenon in quantum computing where a quantum computer can evaluate multiple possibilities simultaneously due to the superposition of states. Unlike classical bits, which can only represent a single state at a time (either 0 or 1), quantum bits (qubits) can exist in a superposition of both states at once, represented as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex coefficients.

Because of superposition, a quantum computer can process many inputs simultaneously. For example, in a quantum algorithm, a quantum system can evaluate a function for all possible inputs in parallel, providing an exponential speedup for certain types of problems.

Quantum parallelism is the foundation of many quantum algorithms, including Grover's search algorithm and Shor's factoring algorithm. However, the results from quantum parallelism often need to be collapsed or measured, which leads to the need for clever algorithms that extract useful information from this parallelism.

It's important to note that quantum parallelism doesn't allow direct observation of all possible outcomes at once; rather, the quantum state can encode multiple solutions that can be extracted through appropriate quantum operations, typically involving interference.

9.5.2 Fourier Transform

The Fourier transform is a mathematical technique used to convert a signal from its original domain (usually time or space) into the frequency domain. It breaks down a complex signal into a sum of simpler sinusoidal components, each with its own frequency, amplitude, and phase. This process is fundamental in fields such as signal processing, physics, and quantum mechanics.

The Fourier transform of a continuous-time signal $f(t)$ is defined as:

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt$$

where $F(\omega)$ represents the frequency components of the signal $f(t)$, and ω is the angular frequency.

The inverse Fourier transform reconstructs the original signal from its frequency components:

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{i\omega t} d\omega$$

In quantum mechanics and quantum computing, the Fourier transform plays a critical role in algorithms such as Shor's algorithm, which leverages the quantum Fourier transform to solve problems more efficiently than classical algorithms.

The **Quantum Fourier Transform (QFT)** is the quantum equivalent of the discrete Fourier transform (DFT), applied to quantum states. It transforms the amplitudes of quantum states into a new basis that reveals frequency or periodic information, essential for many quantum algorithms. Efficiently implemented using a sequence of Hadamard and controlled phase gates, the QFT operates in $O(n^2)$ or $O(n \log n)$ steps for n qubits. It is a cornerstone of quantum algorithms like Shor's algorithm for factoring integers and quantum phase estimation, enabling exponential speedups in problems involving periodicity or eigenvalue estimation.

9.6 Bounded-Error Quantum Polynomial Time (BQP)

In computational complexity theory, bounded-error quantum polynomial time (BQP) is the class of decision problems solvable by a quantum computer in polynomial time, with an error probability of at most 1/3 for all instances. It is the quantum analogue to the complexity class BPP.

A decision problem is a member of BQP if there exists a quantum algorithm (an algorithm that runs on a quantum computer) that solves the decision problem with high probability and is guaranteed to run in polynomial time. A run of the algorithm will correctly solve the decision problem with a probability of at least 2/3.

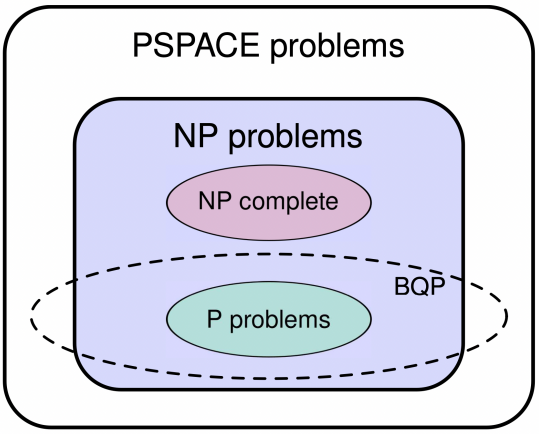


Figure 21: Compexity Space [2]

REFERENCES

[1] Stanford Encyclopedia of Philosophy. Natural deduction. https://plato.stanford.edu/entries/natural-deduction/. Accessed: 2025-01-18.

[2] Mehmet Tahir SANDIKKAYA. Logic & computability, 2024. Fall 2024/25, Istanbul Technical University, Computer Engineering Department.

[3] Elisa Bertino. Binary decision diagram. In: Handbook on Securing Cyber-Physical Critical Infrastructure, 2012. Accessed: 2023-01-17.

[4] Ben Andrew. Constructing binary decision diagrams (bdds), 2023. Accessed: 2023-01-17.

[5] B. Jack Copeland. The Church-Turing Thesis. In Edward N. Zalta and Uri Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2024 edition, 2024.

[6] Panu Raatikainen. Gödel's Incompleteness Theorems. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Spring 2022 edition, 2022.

[7] Andreas Roos and Jörg Rothe. Introduction to computational complexity —. https://www.semanticscholar.org/paper/Introduction-to-Computational-Complexity-%E2%88%97-%E2%80%94-%E2%80%94-Roos-Rothe/73baf84348b5918494bf02b1ade1950592e2449d. Accessed: 2025-01-18.

[8] Karleigh Moore and Christopher Williams. Complexity theory. https://brilliant.org/wiki/complexity-theory/. Contributed by Karleigh Moore and Christopher Williams. Accessed: 2025-01-18.

[9] Ragesh Jaiswal. Csl853: Complexity theory. Indian Institute of Technology Delhi, 2023. Course offered in the Department of Computer Science and Engineering.

[10] Wikipedia contributors. Bqp, 2025. Accessed: 2025-01-19.

[11] Mehrdad's Quantum AI. Quantum computing — ep. 7: Quantum circuit design: the power of cnot gate and entangled qubits, 2021. Accessed: January 18, 2025.

[12] The Bright Side of Mathematics. Pauli matrices - the bright side of mathematics. https://www.youtube.com/watch?v=7rE9gcj5I2U&ab_channel=TheBrightSideofMathematics, 2023. Accessed: 2025-01-18.

[13] Professor Nano. Rsa encryption - professor nano. https://www.youtube.com/watch?v=iMjpZwISIlA&ab_channel=ProfessorNano, 2023. Accessed: 2025-01-18.

[14] Wikipedia contributors. Straightedge and compass construction. https://en.wikipedia.org/w/index.php?title=Straightedge_and_compass_construction&oldid=1267401417, 2025. Accessed: 2025-01-17.