PHYS1007: Quantum Information for Everyone

Author: ThunderDora
Instructor: HUANG Shilin

Last Update: June 23, 2025

Contents

Chapter 1

Brief History of Computers

1.1 Computers before ENIAC

1.1.1 Chinese Abacus

Chinese Abacus is a counting tool used since Han Dynasty (2nd century BC).
However, there are some limitations:

- Speed and Robustness: Faster and more reliable than mental computation, but limited by the speed and correctness of the our hand movements.
- Not Automatic: It requires human intervention to operate.

1.1.2 Pascaline

Pascaline is the first mechanical calculator invented by Blaise Pascal in 1642.
It can perform addition and subtraction, but it is not automatic with less human errors.

1.1.3 Difference Engine

Difference Engine is a mechanical calculator designed by Charles Babbage, the FATHER of the Computer, in 1822.
It can be used to calculate polynomial functions: $\boxed{f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0}$.
Each column is a "register" that holds a number in decimal form.
$\Rightarrow$ Consists of a stack of gears, where each gear corresponds to a digit (0-9).

Difference Method
If the initial value of x is x_0, then the value of $f(x)$ can be calculated by:

$$f(x) = f(x_0) + (x - x_0) f'(x_0) + \frac{(x - x_0)^2}{2!} f''(x_0) + \cdots + \frac{(x - x_0)^n}{n!} f^{(n)}(x_0)$$

Example 1.1 **Computing $f(x)$ using Difference Method**

Suppose we want to compute $f(x) = 2x^2 + 3x + 1$ for some integer x.
We know $f(0) = 1$, $f(1) = 6$, $f(2) = 15$ and $f(3) = 28$, then we can compute the first differences and second differences.
First Differences $\triangle f(x)$:

$$\begin{aligned} \triangle f(0) &= f(1) - f(0) = 6 - 1 = 5 \\ \triangle f(1) &= f(2) - f(1) = 15 - 6 = 9 \\ \triangle f(2) &= f(3) - f(2) = 28 - 15 = 13 \end{aligned}$$

Second Differences $\triangle^2 f(x)$:

$$\begin{aligned} \triangle^2 f(0) &= \triangle f(1) - \triangle f(0) = 9 - 5 = 4 \\ \triangle^2 f(1) &= \triangle f(2) - \triangle f(1) = 13 - 9 = 4 = \triangle^2 f(0) \end{aligned}$$

Since the second differences are the same, we can conclude that the function is a quadratic function.
Therefore, we can write $f(x) = 2x^2 + 3x + 1$.

Example 1.2 **Computing $f(x)$ using Addition Only**

Suppose we want to compute $f(x) = 2x^2 + 3x + 1$ for some integer x.
Given $f(0) = 1$, $\triangle f(0) = 5$ and $\triangle^2 f(0) = 4$ as input, we can compute $f(x)$ for arbitrary positive integer x using addition (+) only.
For $f(6)$:

x	0	1	2	3	4	5	6
$f(x)$	1	$1+5=6$	$6+9=15$	$15+13=28$	$28+17=45$	$45+21=66$	$66+25=91$
$\triangle f(x)$	5	$5+4=9$	$9+4=13$	$13+4=17$	$17+4=21$	$21+4=25$	$25+4=29$
$\triangle^2 f(x)$	4	4	4	4	4	4	4

Therefore, we can compute $f(6) = 91$ using addition only.

1.1.4 Analytical Engine

Analytical Engine is the first general-purpose computer designed by Charles Babbage in 1837, which uses punched cards to input data and instructions.
However, it was never completed due to the lack of funding.
From 1880 to 1910, Babbage's son, Henry Babbage, completed a simplified version of the Analytical Engine called "The Mill" which was able to calculate a list of $\pi, 2\pi, 3\pi, \cdots$ to 29 decimal places.

1.1.5 Electromechanical Computers

Electromechanical Computers are computers that use both electrical and mechanical components.

- e.g.: Engima Machine used by the Germans during WWII to encrypt and decrypt messages, Bombes used by the British to break the Engima code and Z3 developed by Konrad Zuse in 1941.

However, there are some limitations for electromechanical computers:

- Low Speed: Rely on physical movements to operate which limits the speed.
- Unreliable: Moving parts experience friction and wear out over time (Components would degrade and fail).

1.1.6 ENIAC

ENIAC (Electronic Numerical Integrator and Computer) is the first general-purpose electronic digital computer designed by John Mauchly and J. Presper Eckert in 1943.
It was used to calculate artillery firing tables for the US Army during WWII.

- Size: Weighs 30 tons and occupies 167 square meters.
- Speed: 5000 additions per second.
- Reliability: It was reliable and could run for days without errors.

1.2 Binary Numbers

Claude Shannon, the father of modern digital circuit design theory, showed that the binary system is the most efficient way to represent numbers in electronic circuits in 1948.

- Binary System: Uses only two digits, 0 and 1, to represent numbers. (Mainly used in computers)
- Decimal System: Uses ten digits, 0-9, to represent numbers. (Mainly used by humans)

Decimal	0	1	2	3	4	5	6	7	8	9
Binary	0	1	10	11	100	101	110	111	1000	1001

$$\boxed{A = \sum_k a_k 2^k \quad \text{where} \quad a_k \in \{0,1\}}$$

Example 1.3 **Converting Decimal to Binary**

Convert 83 to binary.

$$83 = 64 + 16 + 2 + 1 = 2^6 + 2^4 + 2^1 + 2^0 = \underline{\underline{(1010011)_2}}$$

Example 1.4 **Converting Binary to Decimal**

Convert $(1101)_2$ to decimal.

$$(1101)_2 = 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0 = \underline{\underline{13}}$$

1.2.1 Binary Fraction

Binary Fraction is a number that can be expressed as a sum of negative powers of 2.

$$\boxed{A = \sum_k a_k 2^{-k} \quad \text{where} \quad a_k \in \{0,1\}}$$

Example 1.5 **Converting Decimal to Binary Fraction**

1. Convert $\frac{1}{2}$ to binary fraction.

$$\frac{1}{2} = 2^{-1} = \underline{\underline{(0.1)_2}}$$

2. Convert $\frac{1}{3}$ to binary fraction.

$$\frac{1}{3} = \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots = \underline{\underline{(0.0101\cdots)_2}}$$

1.2.2 Binary Addition

Binary Addition is the process of adding two binary numbers.

- $0+0=0$
- $0+1=1+0=1$
- $1+1=10$ (Carry 1 to the next column)

$$\begin{array}{rl}
11111 & \text{Carry Bits} \\
01101 & = (13)_{10} \\
+\ 10111 & = (23)_{10} \\
\hline
100100 & = (36)_{10}
\end{array}$$

1.2.3 Binary Subtraction

Binary Subtraction is the process of subtracting two binary numbers.

- $0-0=1-1=0$
- $1-0=1$
- $0-1=1$ (Borrow 1 from the next column)

$$\begin{array}{rl}
*\ \ *** & \text{Borrow Bits} \\
1101110 & = (110)_{10} \\
-\quad 10111 & = (23)_{10} \\
\hline
1010111 & = (87)_{10}
\end{array}$$

1.2.4 Binary Multiplication

Binary Multiplication is the process of multiplying two binary numbers.

- $0\times 0=0\times 1=1\times 0=0$
- $1\times 1=1$

$$\begin{array}{rl}
1011 & = (11)_{10} \\
\times\quad 1010 & = (10)_{10} \\
\hline
0000 & \\
+\ 1011 & \\
+\ 0000 & \\
+1011 & \\
\hline
1101110 & = (110)_{10}
\end{array}$$

1.2.5 Binary Division

Binary Division is the process of dividing two binary numbers.

- $0\div 1=0$
- $1\div 1=1$
- $1\div 0=0\div 0=\text{Undefined}$

```
              1 0 1
           ___________
1 0 1    ) 1 1 0 1 1
         - 1 0 1
           -----
               1 1 1
           -   1 0 1
               -----
               0 1 0
```

1.3 Base-b Representation and Horner's Method

We use $(a_{n-1}a_{n-2}\cdots a_1a_0)_b$ to represent a n-digit number in base b where $b>1$.
$\implies$ The value of the number is:

$$\boxed{A=\sum_{k=0}^{n-1} a_k b^k} \quad \text{where} \quad a_k \in \{0,1,\cdots,b-1\}$$

Example 1.6 **Representing a Number in Different Bases**

Represent 13 in base 10, base 8 and base 16.

- Base 10: $13 = 1\times 10^1 + 3\times 10^0 = \underline{\underline{(13)_{10}}}$
- Base 8: $13 = 1\times 8^1 + 5\times 8^0 = \underline{\underline{(15)_8}}$
- Base 16: $13 = 0\times 16^1 + 13\times 16^0 = \underline{\underline{(D)_{16}}}$

Remark: For hexadecimal numbers (base-16), we use the digits 0-9 and the letters A-F to represent the numbers 10-15.

Horner's Method

Theorem 1.1 **Horner's Method**

Horner's Method is a method to convert a number from base b to base 10.
For Integers:

$$\boxed{\sum_{k=0}^{n-1} a_k b^{k-1} = a_0 + b(a_1 + b(a_2 + b(a_3 + \cdots + b(a_{n-1})\cdots)))}$$

For Fractions:

$$\boxed{\sum_{k=1}^{n} a_{-k} b^{-k} = a_{-1} + b(a_{-2} + b(a_{-3} + \cdots + b(a_{-n})\cdots))}$$

Example 1.7 **Converting a Decimal Number to Binary using Horner's Method**

Convert $(35)_{10}$ to binary.

$$\begin{aligned}
35 &= 2\times 17 + 1\\
17 &= 2\times 8 + 1\\
8 &= 2\times 4 + 0\\
4 &= 2\times 2 + 0\\
2 &= 2\times 1 + 0\\
1 &= 2\times 0 + 1
\end{aligned}$$

$\therefore (35)_{10} = \underline{\underline{(100011)_2}}$

Example 1.8 **Converting a Decimal Fraction to Binary using Horner's Method**

Convert $\frac{1}{3}$ to binary.

$$\begin{aligned}
& & \tfrac{1}{3} < 1 &\implies \text{Real Number} = 0.\\
\tfrac{1}{3}\times 2 &= & \tfrac{2}{3} < 1 &\implies \text{Real Number} = 0.0\\
\tfrac{2}{3}\times 2 &= & \tfrac{4}{3} > 1 &\implies \text{Real Number} = 0.01\\
\tfrac{1}{3}\times 2 &= & \tfrac{2}{3} < 1 &\implies \text{Real Number} = 0.010\\
\tfrac{2}{3}\times 2 &= & \tfrac{4}{3} > 1 &\implies \text{Real Number} = 0.0101\\
& \cdots
\end{aligned}$$

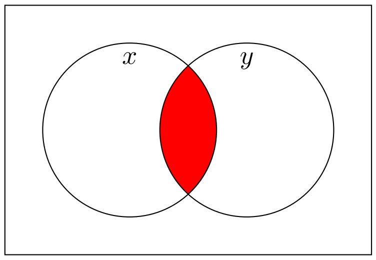
$\therefore \frac{1}{3} = \underline{\underline{(0.0101\cdots)_2}}$

1.4 Boolean Algebra

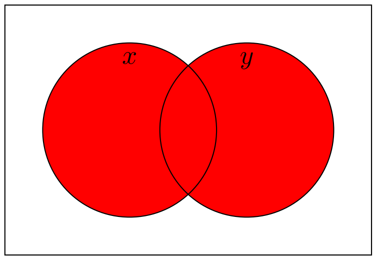
Boolean Algebra is a branch of algebra introduced by George Boole that deals with variables that can take on one of two values, 0 or 1 (True or False).

Three Basic Operations:

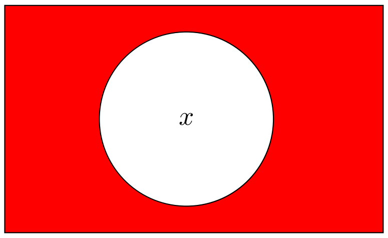
- Conjunction (AND): $\boxed{x \wedge y = x \cdot y}$
  If $x = 1$ and $y = 1$, then $x \wedge y = 1$. Otherwise, $x \wedge y = 0$.



- Disjunction (OR): $\boxed{x \vee y = x + y}$
  If $x = 0$ and $y = 0$, then $x \vee y = 0$. Otherwise, $x \vee y = 1$.



- Negation (NOT): $\boxed{\neg x = 1 - x}$ or $\boxed{\overline{x}}$
  If $x = 1$, then $\neg x = 0$. Otherwise, $\neg x = 1$.



Truth Table is a table that shows the output of a Boolean function for all possible input values.
Here are the truth tables for the basic operations:

x	y	$x \wedge y$	$x \vee y$	$\neg x$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Other Basic Operations Built from the Three Basic Operations:

- Implication (IMPL): $\boxed{x \to y = \neg x \vee y}$
- Exclusive OR (XOR): $\boxed{x \oplus y = (x \wedge \neg y) \vee (\neg x \wedge y)}$
- Equality (EQ): $\boxed{x \leftrightarrow y = (x \wedge y) \vee (\neg x \wedge \neg y)}$
- De Morgan's Laws: $\boxed{x \wedge y = \neg(\neg x \vee \neg y)}$ and $\boxed{x \vee y = \neg(\neg x \wedge \neg y)}$

x	y	$x \to y$	$x \oplus y$	$x \leftrightarrow y$
0	0	1	0	1
0	1	1	1	0
1	0	0	1	0
1	1	1	0	1

1.5 Boolean Functions

Definition 1.1 **Boolean Function**

A Boolean Function is a function that takes input values from a set of Boolean variables and returns a single Boolean value.

$$\boxed{f : \{0,1\}^k \to \{0,1\}}$$

Example 1.9 **Boolean Function of AND Gate**

For an AND Gate with $k = 2$ inputs and $m = 1$ output, the Boolean function is:

x_1	x_2	$f(x_1, x_2)$
0	0	0
0	1	0
1	0	0
1	1	1

Therefore, the Boolean function of an AND Gate is:

$$\boxed{f(x_1, x_2) = x_1 \wedge x_2}$$

Example 1.10 **Boolean Function of Parity Function**

Parity Function is a function that returns 1 if the number of 1's in the input is odd, and 0 otherwise.

For a Parity Function with $k = 3$ inputs and $m = 1$ output, the Boolean function is:

x_0	x_1	x_2	$f(x_0, x_1, x_2)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Therefore, the Boolean function of a Parity Function is:

$$\boxed{f(x_0, x_1, x_2) = x_0 \oplus x_1 \oplus x_2}$$

In Boolean Function, there are two types of adders:

1. Half Adder: Adds two bits and produces a sum and a carry bit.

$$\boxed{f(a, b) = (c, s) = (a \wedge b, a \oplus b)} \quad \text{where} \quad a, b \in \{0, 1\}$$

   - Sum Bit: $s = a \oplus b$
   - Carry Bit: $c = a \wedge b$

2. Full Adder: Adds three bits and produces a sum and a carry bit.

$$\boxed{f(a, b, c_{in}) = (c_{out}, s) = ((a \wedge b) \vee (b \wedge c_{in}) \vee (a \wedge c_{in}), a \oplus b \oplus c_{in})} \quad \text{where} \quad a, b, c_{in} \in \{0, 1\}$$

   - Sum Bit: $s = a \oplus b \oplus c_{in}$
   - Carry-in Bit generated by the Addition of Lower-order Bits: $c_{in} = a \wedge b$
   - Carry-out Bit passed to the Next Higher-order Bit: $c_{out} = (a \wedge b) \vee (b \wedge c_{in}) \vee (a \wedge c_{in})$

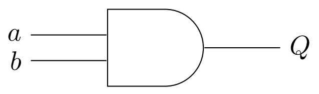
1.6 Logic Circuits

1.6.1 Logic Gates

There are 7 types of logic gates in Boolean Algebra:

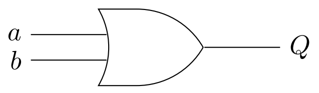
1. AND Gate: Outputs 1 if all inputs are 1.

$$\boxed{\text{AND}(a,b) = a \land b}$$



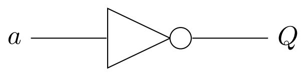
2. OR Gate: Outputs 1 if at least one input is 1.

$$\boxed{\text{OR}(a,b) = a \lor b}$$



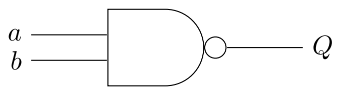
3. NOT Gate: Outputs 1 if the input is 0.

$$\boxed{\text{NOT}(a) = \neg a}$$



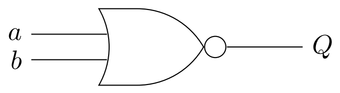
4. NAND Gate: Outputs 0 if all inputs are 1.

$$\boxed{\text{NAND}(a,b) = \text{NOT}(\text{AND}(a,b)) = \neg(a \land b)}$$



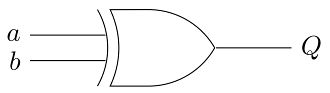
5. NOR Gate: Outputs 0 if at least one input is 1.

$$\boxed{\text{NOR}(a,b) = \text{NOT}(\text{OR}(a,b)) = \neg(a \lor b)}$$



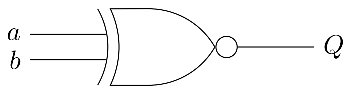
6. XOR Gate: Outputs 1 if the number of 1's in the input is odd.

$$\boxed{\text{XOR}(a,b) = a \oplus b}$$



7. XNOR Gate: Outputs 1 if the number of 1's in the input is even.

$$\boxed{\text{XNOR}(a,b) = \text{NOT}(\text{XOR}(a,b)) = \text{EQUAL}(a,b) = \neg(a \oplus b)}$$



The truth tables for the logic gates are as follows:

a	b	AND(a,b)	OR(a,b)	NOT(a)	NAND(a,b)	NOR(a,b)	XOR(a,b)	XNOR(a,b)
0	0	0	0	1	1	1	0	1
0	1	0	1	1	1	0	1	0
1	0	0	1	0	1	0	1	0
1	1	1	1	0	0	0	0	1

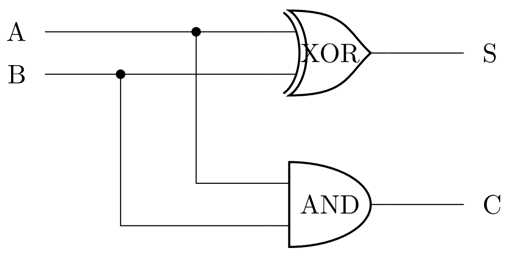
1.6.2 Logic Circuits: Half Adder and Full Adder

Half Adder:
The boolean function of a half adder is:

$$\boxed{\text{Half Adder}(A,B) = (C,S) = (A \wedge B, A \oplus B)} \quad \text{where} \quad A, B \in \{0,1\}$$

Carry Bit: $C = A \wedge B$

Sum Bit: $S = A \oplus B$



The truth table for a half adder is:

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half Adder is used to add two bits and produce a sum and a carry bit, we can illustrate the half adder as follows:

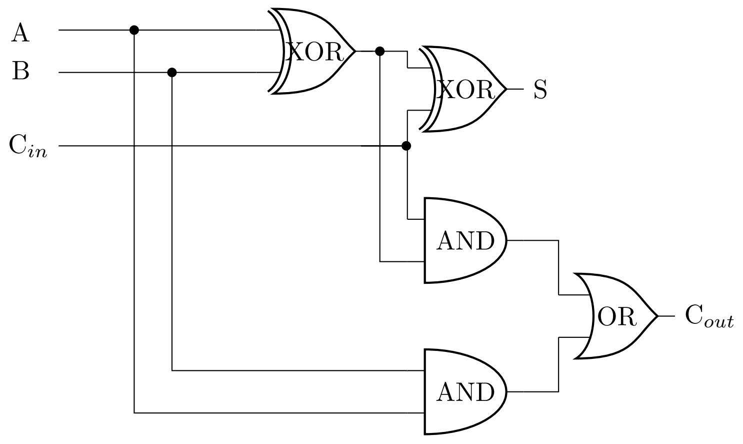
$$\begin{array}{rl} 11111^{C} & \text{Carry Bits} \\ 011\ 0\ 1^{A} & = (13)_{10} \\ +\ 101\ 1\ 1^{B} & = (23)_{10} \\ \hline 1001\ 0\ 0^{S} & = (36)_{10} \end{array}$$

Full Adder:
The boolean function of a full adder is:

$$\boxed{\text{Full Adder}(A,B,C_{in}) = (C_{out}, S) = ((A \wedge B) \vee (B \wedge C_{in}) \vee (A \wedge C_{in}), A \oplus B \oplus C_{in})} \quad \text{where} \quad A, B, C_{in} \in \{0,1\}$$

Carry-out Bit: $C_{out} = (A \wedge B) \vee (B \wedge C_{in}) \vee (A \wedge C_{in})$

Sum Bit: $S = A \oplus B \oplus C_{in}$



The truth table for a full adder is:

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

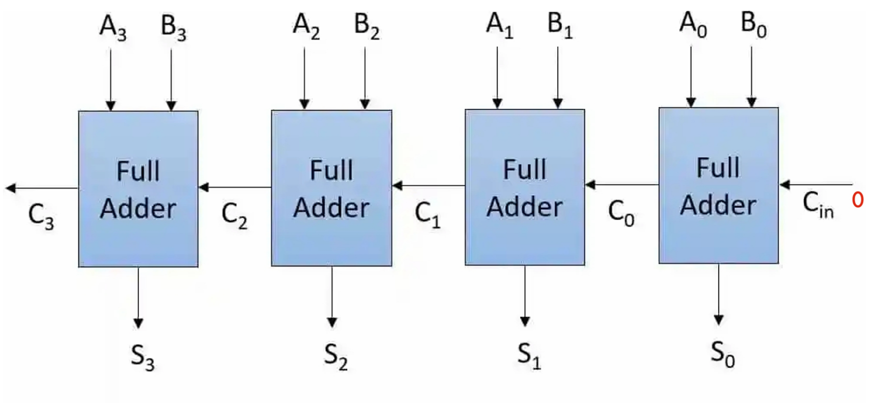
Full Adder is used to add three bits and produce a sum and a carry-out bit.

- Check S: If there are an odd number of 1's among the three bits, then $S = 1$. Otherwise, $S = 0$.
- Check C_{out}:
  - Case 1: If $A \wedge B = 1$, then $C_{out} = 1$.
  - Case 2: If $A \oplus B = 1$ and $C_{in} = 1$, then $C_{out} = 1$.
  - Case 3: If $A = 0$ and $B = 0$, then $C_{out} = 0$.

We can illustrate the full adder as follows:

$$\begin{array}{rcccl}
 & & \mathrm{C_{out}}\mathrm{C_{in}} & & \\
 11 & 1 & 1\ 1 & & \text{Carry Bits} \\
 0 & 1 & 1^{A}01 & & = (13)_{10} \\
 +\ 1 & 0 & 1^{B}11 & & = (23)_{10} \\
 \hline
 10 & 0 & 1^{S}00 & & = (36)_{10}
\end{array}$$

A multi-bit adder can be constructed by connecting multiple full adders together.



4-Bit Ripple Carry Adder

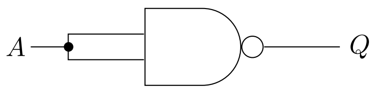
1.6.3 Universality of NAND Gate

Originally, we can use AND, OR and NOT gates to construct any logic circuit.

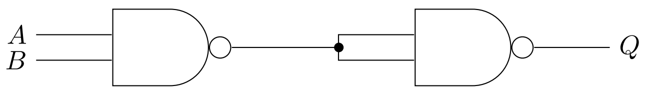
Example: $f(a,b,c) = (a \wedge b \wedge \neg c) \vee (\neg a \wedge \neg b \wedge c) \vee (a \wedge \neg b \wedge c)$ can be constructed using AND, OR and NOT gates. From the truth table, we can see that $f(a,b,c) = 1$ when $(a,b,c) = (1,1,0)$, $(0,0,1)$ and $(1,0,1)$.

However, we can also use NAND gates to construct any logic circuit.

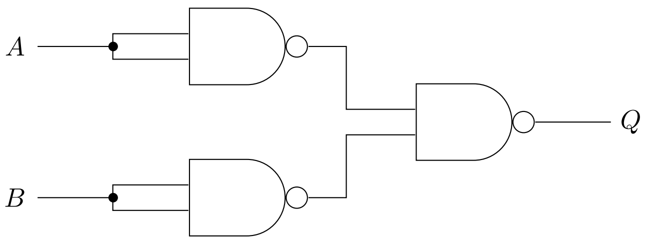
1. NOT Gate: $\boxed{\text{NOT}(A) = \text{NAND}(A, A)}$



2. AND Gate: $\boxed{\text{AND}(A, B) = \text{NOT}(\text{NAND}(A, B))}$



3. OR Gate: $\boxed{\text{OR}(A, B) = \text{NAND}(\text{NOT}(A), \text{NOT}(B))}$



Since {AND, OR, NOT} are universal, NAND gates are also universal as they can be used to construct any logic circuit.

1.7 Computer Science and Architecture

1.7.1 Turing Machine

In 1936, Alan Turing introduced the concept of a Turing Machine as a mathematical model of computation and proved that universal computation can be achieved with a simple machine.

Definition 1.2 **Turing Machine**

Turing Machine is a simplified mathematical model of a computer that can use for general-purpose/universal computation. It consists of an infinite tape divided into cells, a read/write head that can move left or right, and a finite set of states.

Components of a Turing Machine:

- Input: A sequence of symbols on the infinite tape.
- Rules: A set of instructions for the read/write head to move left or right, read or write a symbol, and change state.
- Output: The result of the computation on the tape.

Turing Machines nowadays can be applied in realizations of computers, games like Minecraft and Manufactoria (Puzzle Game) or Brainf**k Language Simulation.

1.7.2 Von Neumann Architecture

In 1945, John von Neumann proposed the concept of a stored-program computer, which is the basis of modern computer architecture.

Components of a Von Neumann Architecture:

- Central Processing Unit (CPU): Performs arithmetic and logical operations & Manages the data registers.
- Memory: Stores data and programs.
- Control Unit: Manages the instructions registers & Acts as the program counter.
- External Mass Storage: Stores large amounts of data.
- Input/Output (I/O) Devices: Interacts with the user.

1.7.3 Random Access Memory (RAM)

Definition 1.3 **Random Access Memory (RAM)**

Random Access Memory (RAM) is a type of computer memory that can be accessed at any memory location and quickly by the CPU.
$\implies$ Volatile and Temporary, meaning that the data is lost when the computer is turned off.

Since Turing Machines and Von Neumann Architecture read data sequentially, they are slow when accessing data stored at widely separated memory locations.
$\implies$ RAM is used to store data temporarily for quick access by the CPU.

1.8 Reversible Computing

1.8.1 Landauer's Principle

Definition 1.4 — **Landauer's Principle**

Landauer's Principle states that any logically irreversible operation like the erasure of 1 bit of information must result in an increase in entropy and heat dissipation in the physical system performing the computation.
$\implies$ Losing 1 bit of information dissipates $k_B T \ln 2$ of energy, where:

- k_B is the Boltzmann constant.
- T is the temperature (K).

- e.g.: If $T = 300$ K, then the energy dissipated is $k_B T \ln 2 \approx 2.9 \times 10^{-21}$ J.

Information Erasure:
There are two cases for information erasure:

- Case 1: If A bit has a value of 0/1 with probability of 50%, then we gain 1 bit of information when we measure the bit.
- Case 2: If A bit has a value of 0 with probability of 100%, then we gain 0 bit of information when we measure the bit.

Theorem 1.2 — **Average Information Gain**

Average Information Gain when measuring a bit if $P[0] = p$ and $P[1] = 1 - p$ is:

$$\boxed{H_b(p) = -p\log_2(p) - (1-p)\log_2(1-p)}$$

Example 1.11 — **Average Information Lost of Irreversible NAND Gate**

Naively, the 2 input bits of a NAND gate will produce 1 output bit. $\implies$ Probability of output bit being 0 is $\frac{1}{4}$ and 1 is $\frac{3}{4}$.
Information in Output Bit: $H_b\left(\frac{1}{4}\right) = -\frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{3}{4}\log_2\left(\frac{3}{4}\right) \approx 0.811$ bits.
$\therefore$ The average information lost for a NAND gate is:

$$\text{Average Information Lost} = 2 - 0.811 \approx \underline{\underline{1.189 \text{ bits}}}$$

AND, OR, NAND, NOR, XOR and XNOR gates are all irreversible, meaning these gates will lose information and dissipate heat when performing computations.
As a result, we need to design reversible circuits to have no heat dissipation due to information erasure.

1.8.2 Reversible Computing and Functions

Definition 1.5 — **Reversible Computing**

Reversible Computing is a form of unconventional computing that performs reversible computation in the entire process.
$\implies$ Each step of the computation implements a reversible function.

Definition 1.6 — **Reversible Function**

A reversible function is a function that is one-to-one/bijective (Given the output y, we can uniquely determine the input x).

NOT gate is an example of a reversible function as it is one-to-one.

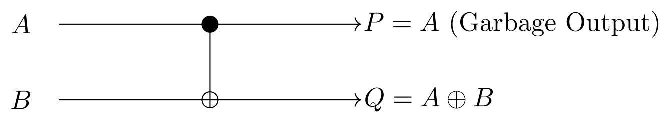
1.8.3 Reversible Gates

Definition 1.7 **Reversible Gates**

Reversible Gates are logic gates that implement reversible functions.
$\implies$ The number of input bits is equal to the number of output bits.

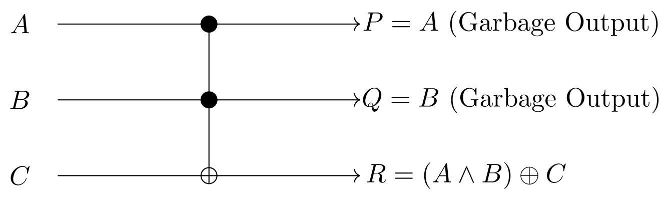
There are several reversible gates that can be used to construct reversible circuits:

1. Reversible XOR/Controlled NOT (CNOT) Gate: $\text{CNOT}(A, QB) = (A, A \oplus B) = (P, Q)$



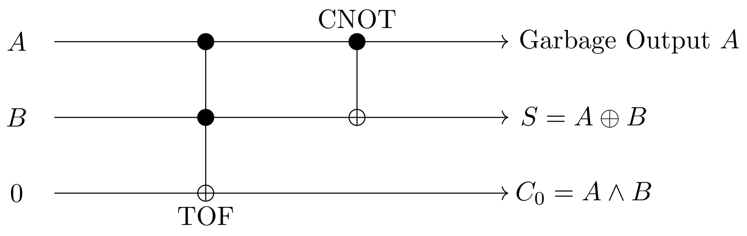
A	B	P	Q
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

2. Toffoli Gate: $\text{TOF}(A, B, C) = (A, B, (A \wedge B) \oplus C) = (P, Q, R)$
$\implies$ If $C = 1$, then $\text{TOF}(A, B, C) = (A, B, \text{NAND}(A, B))$ which is universal.



A	B	C	P	Q	R
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

3. Reversible Half Adder: $\text{Half Adder}(A, B) = (A \oplus B, A \wedge B) = (S, C_0)$



A	B	S	C_0
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Chapter 2

Brief Introduction to Quantum Computers

2.1 Quantum States

2.1.1 Quantization and Superposition

Definition 2.1 **Quantization**

Quantization is the process of discretizing a continuous variable into quantum states $|\psi\rangle$.

Differences between Classical Bits and Quantum Bits:

- Classical Bit: 0 or 1.
- Quantum Bit (Qubit): $|0\rangle$ or $|1\rangle$.

Definition 2.2 **Superposition**

Superposition is the ability of a quantum system to combine multiple quantum states $|\psi\rangle$ into a single quantum state.

Remark:
There is no $|0\rangle + |1\rangle = |1\rangle$ in quantum computing. $\implies$ Physical Meaning of Superposition: A combination of $|0\rangle$ and $|1\rangle$.

2.1.2 State of a Qubit

Definition 2.3 **State of a Qubit**

The state of a qubit is a superposition/combination of the basis states $|0\rangle$ and $|1\rangle$:

$$\boxed{|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle} \quad \text{where } \alpha \neq 0 \text{ or } \beta \neq 0$$

Examples of Qubit States with Different α and β:

- $\alpha = 1, \beta = 0$: $|\psi\rangle = 1\,|0\rangle + 0\,|1\rangle = |0\rangle$.
- $\alpha = 0, \beta = 1$: $|\psi\rangle = 0\,|0\rangle + 1\,|1\rangle = |1\rangle$.
- $\alpha = 1, \beta = 1$: $|\psi\rangle = 1\,|0\rangle + 1\,|1\rangle = |0\rangle + |1\rangle$.
- $\alpha = 1, \beta = -1$: $|\psi\rangle = 1\,|0\rangle - 1\,|1\rangle = |0\rangle - |1\rangle$.
- All real numbers $\implies$ $\alpha = \sqrt{\frac{1}{10}}, \beta = -\sqrt{\frac{9}{10}}$: $|\psi\rangle = \sqrt{\frac{1}{10}}\,|0\rangle + \sqrt{\frac{9}{10}}\,|1\rangle$.
- All complex numbers $\implies$ $\alpha = 1, \beta = -j$: $|\psi\rangle = 1\,|0\rangle - j\,|1\rangle$.

2.1.3 Algebra of Qubit States

Qubit states are vectors which can form a vector space.
$\implies$ Therefore, they can perform vector operations such as sclar multiplication and vector addition.

Scalar Multiplication:

Theorem 2.1 **Scalar Multiplication of Qubit States**

The scalar multiplication of a qubit state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ by a scalar γ is:

$$\boxed{\gamma|\psi\rangle = \gamma\alpha|0\rangle + \gamma\beta|1\rangle} \quad \text{where } \alpha, \beta, \gamma \in \mathbb{C}$$

- e.g. $\gamma = \frac{1}{\sqrt{2}}$: $\frac{1}{\sqrt{2}}|\psi\rangle = \frac{1}{\sqrt{2}}\alpha|0\rangle + \frac{1}{\sqrt{2}}\beta|1\rangle$.

Vector Addition:

Theorem 2.2 **Vector Addition of Qubit States**

The vector addition of two qubit states $|\psi_1\rangle = \alpha_1|0\rangle + \beta_1|1\rangle$ and $|\psi_2\rangle = \alpha_2|0\rangle + \beta_2|1\rangle$ is:

$$\boxed{|\psi_1\rangle + |\psi_2\rangle = (\alpha_1 + \alpha_2)|0\rangle + (\beta_1 + \beta_2)|1\rangle} \quad \text{where } \alpha_1, \alpha_2, \beta_1, \beta_2 \in \mathbb{C}$$

Combing both scalar multiplication and vector addition, we can perform the following operations:

$$\boxed{c_1(\alpha_1|0\rangle + \beta_1|1\rangle) + c_2(\alpha_2|0\rangle + \beta_2|1\rangle) = (\alpha_1 c_1 + \alpha_2 c_2)|0\rangle + (\beta_1 c_1 + \beta_2 c_2)|1\rangle}$$

- e.g. $\frac{1}{2}(|0\rangle + |1\rangle) + \frac{1}{2}(|0\rangle - |1\rangle) = \frac{1}{2}|0\rangle + \frac{1}{2}|1\rangle + \frac{1}{2}|0\rangle - \frac{1}{2}|1\rangle = |0\rangle$.

Remark: Do not write $\alpha|0\rangle + \beta|1\rangle$ in other courses as vector operations.

2.2 Single-Qubit Gates

2.2.1 Quantum Gates

Definition 2.4 **Quantum Gates**

Quantum Gates are unitary operators that operate on qubit states to perform quantum computations.
$\Longrightarrow$ Function $U(\cdot)$ takes a qubit state $|\psi\rangle$ and outputs a new qubit state $U(|\psi\rangle)$.

$$\begin{tikzcd} |\text{IN}\rangle \arrow[r] & \boxed{\text{U}} \arrow[r] & |\text{OUT}\rangle = U|\text{IN}\rangle \end{tikzcd}$$

Linearity of Quantum Gates: $U\left(U(\alpha|\psi_1\rangle + \beta|\psi_2\rangle)\right) = \alpha U(|\psi_1\rangle) + \beta U(|\psi_2\rangle)$.

$$\begin{tikzcd} |0\rangle \arrow[r] & \boxed{\text{U}} \arrow[r] & U|0\rangle & |1\rangle \arrow[r] & \boxed{\text{U}} \arrow[r] & U|1\rangle \end{tikzcd}$$

$$\begin{tikzcd} \alpha|0\rangle + \beta|1\rangle \arrow[r] & \boxed{\text{U}} \arrow[r] & \alpha U|0\rangle + \beta U|1\rangle \end{tikzcd}$$

Remark: Quantum Gates are reversible as they are unitary operators.

2.2.2 Types of Single-Qubit Gates

There are several types of single-qubit gates that can be used to perform quantum computations:

1. Identity Gates: $|0\rangle \to |0\rangle$ and $|1\rangle \to |1\rangle$.

$$\begin{tikzcd} |0\rangle \arrow[r] & |0\rangle & |1\rangle \arrow[r] & |1\rangle \end{tikzcd}$$

$$\begin{tikzcd} \alpha|0\rangle + \beta|1\rangle \arrow[r] & \alpha|0\rangle + \beta|1\rangle \end{tikzcd}$$

2. X Gate (Bit-Flip Gate): $|0\rangle \to |1\rangle$ and $|1\rangle \to |0\rangle$

$$\begin{tikzcd} {|0\rangle} \arrow[r] & \boxed{\text{X}} \arrow[r] & {|1\rangle} & {|1\rangle} \arrow[r] & \boxed{\text{X}} \arrow[r] & {|0\rangle} \end{tikzcd}$$

$$\begin{tikzcd} {\alpha\,|0\rangle + \beta\,|1\rangle} \arrow[r] & \boxed{\text{X}} \arrow[r] & {\alpha\,|1\rangle + \beta\,|0\rangle} \end{tikzcd}$$

- e.g. 1: $\alpha = \frac{1}{\sqrt{2}}, \beta = \frac{1}{\sqrt{2}}$: $|\psi\rangle = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle$.
- e.g. 2: $\alpha = \frac{1}{\sqrt{2}}, \beta = -\frac{1}{\sqrt{2}}$: $|\psi\rangle = \frac{1}{\sqrt{2}}\,|0\rangle - \frac{1}{\sqrt{2}}\,|1\rangle$.

3. Z Gate (Phase-Flip Gate): $|0\rangle \to |0\rangle$ and $|1\rangle \to -\,|1\rangle$

$$\begin{tikzcd} {|0\rangle} \arrow[r] & \boxed{\text{Z}} \arrow[r] & {|0\rangle} & {|1\rangle} \arrow[r] & \boxed{\text{Z}} \arrow[r] & {-\,|1\rangle} \end{tikzcd}$$

$$\begin{tikzcd} {\alpha\,|0\rangle + \beta\,|1\rangle} \arrow[r] & \boxed{\text{Z}} \arrow[r] & {\alpha\,|0\rangle - \beta\,|1\rangle} \end{tikzcd}$$

- e.g. 1: $\alpha = \frac{1}{\sqrt{2}}, \beta = \frac{1}{\sqrt{2}}$: $|\psi\rangle = \frac{1}{\sqrt{2}}\,|0\rangle - \frac{1}{\sqrt{2}}\,|1\rangle$.
- e.g. 2: $\alpha = \frac{1}{\sqrt{2}}, \beta = -\frac{1}{\sqrt{2}}$: $|\psi\rangle = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle$.

4. H Gate (Hadamard Gate): $|0\rangle \to \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle$ and $|1\rangle \to \frac{1}{\sqrt{2}}\,|0\rangle - \frac{1}{\sqrt{2}}\,|1\rangle$

$$\begin{tikzcd} {|0\rangle} \arrow[r] & \boxed{\text{H}} \arrow[r] & {\frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle} & {|1\rangle} \arrow[r] & \boxed{\text{H}} \arrow[r] & {\frac{1}{\sqrt{2}}\,|0\rangle - \frac{1}{\sqrt{2}}\,|1\rangle} \end{tikzcd}$$

$$\begin{tikzcd} {\alpha\,|0\rangle + \beta\,|1\rangle} \arrow[r] & \boxed{\text{H}} \arrow[r] & {\frac{\alpha+\beta}{\sqrt{2}}\,|0\rangle + \frac{\alpha-\beta}{\sqrt{2}}\,|1\rangle} \end{tikzcd}$$

- e.g. 1: $\alpha = \frac{1}{\sqrt{2}}, \beta = \frac{1}{\sqrt{2}}$: $|\psi\rangle = \frac{\frac{1}{\sqrt{2}}+\frac{1}{\sqrt{2}}}{\sqrt{2}}\,|0\rangle + \frac{\frac{1}{\sqrt{2}}-\frac{1}{\sqrt{2}}}{\sqrt{2}}\,|1\rangle = |0\rangle$.
- e.g. 2: $\alpha = \frac{1}{\sqrt{2}}, \beta = -\frac{1}{\sqrt{2}}$: $|\psi\rangle = \frac{\frac{1}{\sqrt{2}}-\frac{1}{\sqrt{2}}}{\sqrt{2}}\,|0\rangle + \frac{\frac{1}{\sqrt{2}}+\frac{1}{\sqrt{2}}}{\sqrt{2}}\,|1\rangle = |1\rangle$.

They are all reversible as they are unitary operators.

2.3 Multi-Qubit States and Gates

2.3.1 Two-Qubit States

Definition 2.5 **Two-Qubit States**

The state of two qubits is a superposition/combination of the basis states of 2 classical bits $|00\rangle, |01\rangle, |10\rangle, |11\rangle$:

$$\boxed{|\psi\rangle = c_{00}\,|00\rangle + c_{01}\,|01\rangle + c_{10}\,|10\rangle + c_{11}\,|11\rangle} \quad \text{where } c_{00}, c_{01}, c_{10}, c_{11} \text{ are complex numbers}$$

Joint State of Two Qubits

Theorem 2.3 **Joint State of Two Qubits**

The joint state of two qubits is the tensor product ($\otimes$) of the individual qubit states $|\psi_1\rangle = \alpha_1\,|0\rangle + \beta_1\,|1\rangle$ and $|\psi_2\rangle = \alpha_2\,|0\rangle + \beta_2\,|1\rangle$:

$$\boxed{\begin{aligned} |\psi\rangle &= |\psi_1\rangle \otimes |\psi_2\rangle \\ &= (\alpha_1\,|0\rangle + \beta_1\,|1\rangle) \otimes (\alpha_2\,|0\rangle + \beta_2\,|1\rangle) \\ &= \alpha_1\alpha_2\,|00\rangle + \alpha_1\beta_2\,|01\rangle + \beta_1\alpha_2\,|10\rangle + \beta_1\beta_2\,|11\rangle \end{aligned}}$$

Examples of Joint States of Two Qubits (Sometimes omit the tensor product symbol):

- $\alpha_1 = \alpha_2 = 1, \beta_1 = \beta_2 = 0$: $|\psi\rangle = |0\rangle \otimes |0\rangle = |00\rangle$.
- $\alpha_1 = \beta_2 = 1, \beta_1 = \alpha_2 = 0$: $|\psi\rangle = |0\rangle \otimes |1\rangle = |01\rangle$.
- $\beta_1 = \alpha_2 = 1, \alpha_1 = \beta_2 = 0$: $|\psi\rangle = |1\rangle \otimes |0\rangle = |10\rangle$.
- $\beta_1 = \beta_2 = 1, \alpha_1 = \alpha_2 = 0$: $|\psi\rangle = |1\rangle \otimes |1\rangle = |11\rangle$.

Properties of Two-Qubit States

Distribution Law of Tensor Product: $|A\rangle \otimes (|B\rangle + |C\rangle) = |A\rangle \otimes |B\rangle + |A\rangle \otimes |C\rangle$ or $(|A\rangle + |B\rangle) \otimes |C\rangle = |A\rangle \otimes |C\rangle + |B\rangle \otimes |C\rangle$.

- e.g. 1: $(|0\rangle + |1\rangle) \otimes (|0\rangle + |1\rangle) = |0\rangle \otimes |0\rangle + |0\rangle \otimes |1\rangle + |1\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle = |00\rangle + |01\rangle + |10\rangle + |11\rangle$.
- e.g. 2: $(|0\rangle + |1\rangle) \otimes (|0\rangle - |1\rangle) = |0\rangle \otimes |0\rangle - |0\rangle \otimes |1\rangle + |1\rangle \otimes |0\rangle - |1\rangle \otimes |1\rangle = |00\rangle - |01\rangle + |10\rangle - |11\rangle$.

Linearity of Tensor Product: $\alpha |A\rangle + \beta |B\rangle = |A\rangle \otimes (\alpha |B\rangle) = \alpha(|A\rangle \otimes |B\rangle)$.
Communative Property of Tensor Product: $|A\rangle \otimes |B\rangle = |B\rangle \otimes |A\rangle$.

2.3.2 N-Qubit States

Definition 2.6 **N-Qubit States**

The state of N qubits is a superposition/combination of the basis states of N classical bits $\{0,1\}^N$:

$$\boxed{|\psi\rangle = \sum_{s \in \{0,1\}^N} c_s |s\rangle} \quad \text{where } c_s \text{ are } 2^N \text{ complex numbers}$$

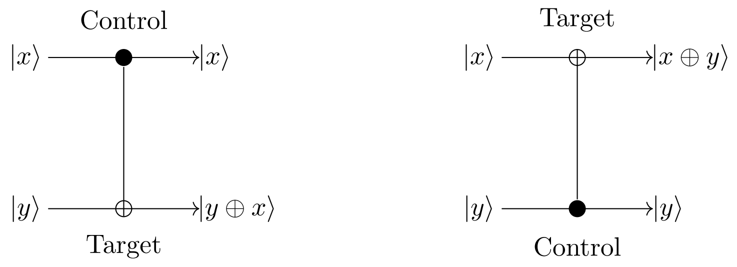
Example of N-Qubit States: $|\psi\rangle = |0...0\rangle + |0...01\rangle + \ldots + |1...1\rangle = (|0\rangle + |1\rangle)^{\otimes N}$.

2.3.3 Multi-Qubit Gates

There are several types of multi-qubit gates that can be used to perform quantum computations:

1. Controlled-X (CNOT) Gate: First qubit is the control qubit and the second qubit is the target qubit.
$\implies$ $\text{CNOT}|00\rangle = |00\rangle$, $\text{CNOT}|01\rangle = |01\rangle$, $\text{CNOT}|10\rangle = |11\rangle$, $\text{CNOT}|11\rangle = |10\rangle$.

$\boxed{\text{CNOT}|x\rangle|y\rangle = |x\rangle|y \oplus x\rangle}$ $\boxed{\text{CNOT}|y\rangle|x\rangle = |y\rangle|x \oplus y\rangle}$



   - e.g.:

$$\begin{aligned}\text{CNOT}(c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle) &= c_{00}\text{CNOT}|00\rangle + c_{01}\text{CNOT}|01\rangle + c_{10}\text{CNOT}|10\rangle + c_{11}\text{CNOT}|11\rangle \\ &= c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|11\rangle + c_{11}|10\rangle\end{aligned}$$

2. 1-Qubit Gates on Multi-Qubit States: Apply the 1-qubit gate to the target qubit.
   - e.g. 1: If we apply the U gate to the first bit of the 2-qubit state $|\psi\rangle = c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle$, we get:

$$\begin{aligned}&c_{00}(U|0\rangle)|0\rangle + c_{01}(U|0\rangle)|1\rangle + c_{10}(U|1\rangle)|0\rangle + c_{11}(U|1\rangle)|1\rangle \\ &= U|0\rangle \otimes (c_{00}|0\rangle + c_{01}|1\rangle) + U|1\rangle \otimes (c_{10}|0\rangle + c_{11}|1\rangle)\end{aligned}$$

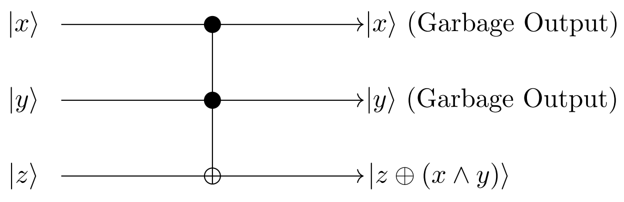
   - e.g. 2: If we apply the U gate to the second bit of the 2-qubit state $|\psi\rangle = c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle$, we get:

$$\begin{aligned}&c_{00}|0\rangle(U|0\rangle) + c_{01}|0\rangle(U|1\rangle) + c_{10}|1\rangle(U|0\rangle) + c_{11}|1\rangle(U|1\rangle) \\ &= (c_{00}|0\rangle + c_{10}|1\rangle) \otimes U|0\rangle + (c_{01}|0\rangle + c_{11}|1\rangle) \otimes U|1\rangle\end{aligned}$$

3. 3-Qubit Toffoli Gate: First two qubits are control qubits and the third qubit is the target qubit.

$$\boxed{\text{TOF}\,|x,y,z\rangle = |x,y,z\rangle} \quad \text{if } x, y \neq 1$$

$$\boxed{\text{TOF}\,|x,y,z\rangle = |x,y,z \oplus x \wedge y\rangle} \quad \text{if } x, y = 1$$


$|x\rangle$
$|y\rangle$
$|z\rangle$
$|x\rangle$ (Garbage Output)
$|y\rangle$ (Garbage Output)
$|z \oplus (x \wedge y)\rangle$

- e.g.: $\text{TOF}(|010\rangle + |110\rangle) = |010\rangle + |111\rangle$.

2.4 Measurement of Quantum Outputs

Definition 2.7 — **Measurement of Quantum Outputs**

The measurement of quantum outputs is the process of extracting classical information from a quantum state.
$\implies$ The quantum state collapses to one of the basis states $|0\rangle, |1\rangle$.

2.4.1 Single-Qubit Measurement

Theorem 2.4 — **Measurement of a Qubit State**

The measurement of a qubit state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ results in:

$$\text{Probability of Output Classical 0: } \boxed{P[0] = \frac{|\alpha|^2}{|\alpha|^2 + |\beta|^2}}$$

$$\text{Probability of Output Classical 1: } \boxed{P[1] = \frac{|\beta|^2}{|\alpha|^2 + |\beta|^2}}$$

- e.g.: For $|\psi\rangle = 4|0\rangle + 3|1\rangle$, the probability of measuring 0 is $\frac{16}{25}$ and the probability of measuring 1 is $\frac{9}{25}$.

There are different situations that can occur when measuring a qubit state:

1. For real numbers, $|\alpha|^2 = \alpha^2$ and $|\beta|^2 = \beta^2$.
2. For complex numbers, $|\alpha|^2 = \alpha \cdot \alpha^* = |\alpha|^2$ and $|\beta|^2 = \beta \cdot \beta^* = |\beta|^2$.
3. If $|\psi\rangle = |0\rangle$, then $P[0] = 1$ and $P[1] = 0$.
4. If $|\psi\rangle = |1\rangle$, then $P[0] = 0$ and $P[1] = 1$.
5. If $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ or $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, then $P[0] = P[1] = \frac{1}{2}$.

Qubit States cannot be remeasured as the qubit will be destroyed after the first measurement.

2.4.2 Two-Qubit Measurement

Theorem 2.5 — **Measurement of a Two-Qubit State**

The measurement of a two-qubit state $|\psi\rangle = c_{00}|00\rangle + c_{01}|01\rangle + c_{10}|10\rangle + c_{11}|11\rangle$ results in:

$$\text{Probability of Output } x_0x_1 : \boxed{P[x_0x_1] = \frac{|c_{x_0x_1}|^2}{\sum_{y_0y_1} |c_{y_0y_1}|^2}} \quad \text{where } x_0, x_1 \in \{0,1\}$$

- e.g.: For $|\psi\rangle = \sqrt{3}|00\rangle + \sqrt{2}|01\rangle + \sqrt{5}|10\rangle + \sqrt{7}|11\rangle$, the probability of measuring 00 is $\frac{3}{17}$, 01 is $\frac{2}{17}$, 10 is $\frac{5}{17}$, and 11 is $\frac{7}{17}$.

2.4.3 Difference between Classical and Quantum Measurements

	Classical Measurement	Quantum Measurement
Output	Deterministic	Probabilistic
Reusability	Yes	No
Information	Extracted	Destroyed
State	Definite	Superposition

For measurements of $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, they are not the same since measurements with Hadamard gates: $\mathrm{H}\,|+\rangle = |0\rangle$ and $\mathrm{H}\,|-\rangle = |1\rangle$ have different outputs even though they have the same probability of $\frac{1}{2}$.
For measurements of $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $\sqrt{2}\,|+\rangle = |0\rangle + |1\rangle$, they are the same since in general $|\psi\rangle$ and $\gamma\,|\psi\rangle$ are indistinguishable states with the same probability of $\frac{1}{2}$. $\implies$ Only $|A\rangle + |B\rangle$ and $|A\rangle + \gamma\,|B\rangle$ are distinguishable states.
Even though $|1\rangle$ and $-\,|1\rangle$ are the same states, for $Z\,|1\rangle = -\,|1\rangle$, we cannot write $Z\,|1\rangle = |1\rangle$ as it will lose the linearity of quantum mechanics.

2.5 Deutsch's Algorithm

Definition 2.8 — **Deutsch's Algorithm**

Deutsch's Algorithm is a quantum algorithm that determines whether a black box function is constant or balanced with only one query.

Theorem 2.6 — **Deutsch's Algorithm**

Given a Boolean function $f : \{0,1\} \to \{0,1\}$ as a black box.

RESET 0

$$\begin{tikzcd} 0 \arrow[r, mapsto] & 0 \\ 1 \arrow[ur, mapsto] & 1 \end{tikzcd}$$

WIRE

$$\begin{tikzcd} 0 \arrow[r, mapsto] & 0 \\ 1 \arrow[r, mapsto] & 1 \end{tikzcd}$$

NOT

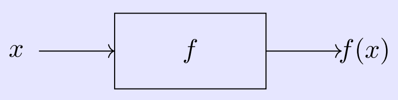
$$\begin{tikzcd} 0 \arrow[dr, mapsto] & 0 \\ 1 \arrow[ur, mapsto] & 1 \end{tikzcd}$$

RESET 1

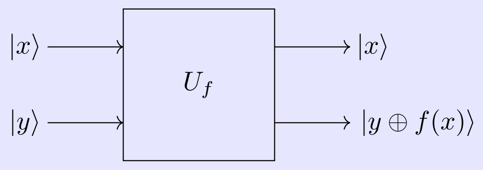
$$\begin{tikzcd} 0 \arrow[dr, mapsto] & 0 \\ 1 \arrow[r, mapsto] & 1 \end{tikzcd}$$

One can feed an input bit $x \in \{0,1\}$ to the black box and ask the black box to provide $f(x)$ as the output, but cannot "look inside" and see how the black box computes the output.

Classical Black Box



Quantum Black Box



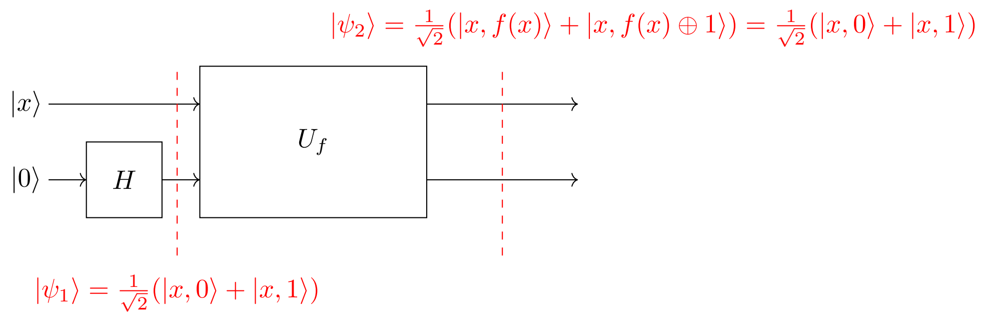
The goal is to determine if the function is balanced or constant:

- If $f(0) = f(1)$ or $f(0) \oplus f(1) = 0$, then the function is constant.
- If $f(0) \neq f(1)$ or $f(0) \oplus f(1) = 1$, then the function is balanced.

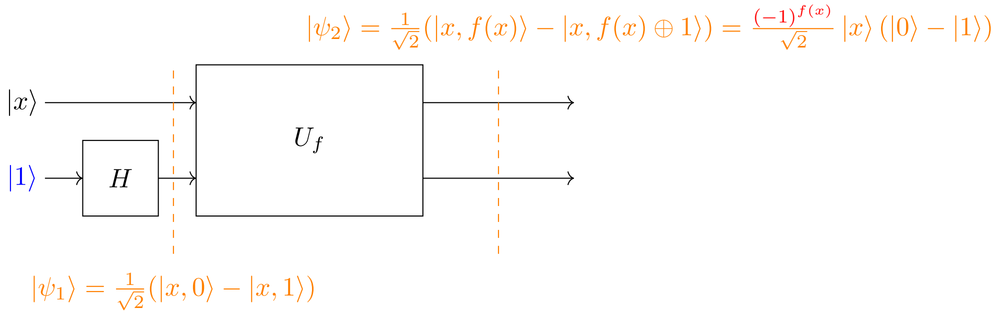
In the classical world, we need to ask the black box twice to get one bit of information $f(0) \oplus f(1)$, but in the quantum world, we only need to ask the black box once to get the same information.

Here are some examples of Deutsch's Algorithm:

1. For $|\psi_1\rangle = \frac{1}{\sqrt{2}}(|x,0\rangle + |x,1\rangle)$, the output is $|\psi_2\rangle = \frac{1}{\sqrt{2}}(|x,f(x)\rangle + |x,f(x)\oplus 1\rangle) = \frac{1}{\sqrt{2}}(|x,0\rangle + |x,1\rangle)$.

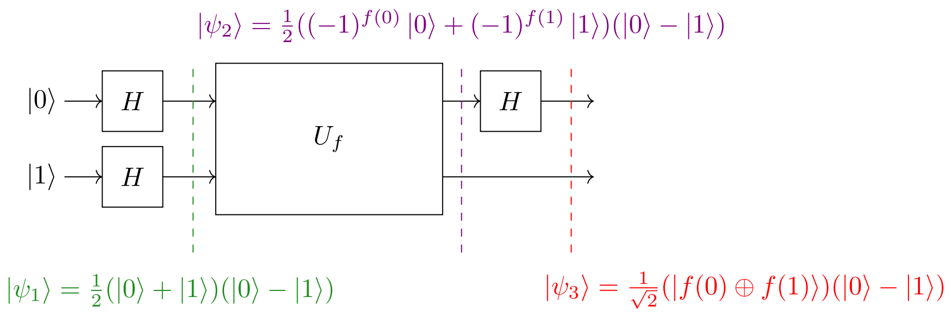

$|\psi_2\rangle = \frac{1}{\sqrt{2}}(|x,f(x)\rangle + |x,f(x)\oplus 1\rangle) = \frac{1}{\sqrt{2}}(|x,0\rangle + |x,1\rangle)$
$|x\rangle$
$|0\rangle$
H
U_f
$|\psi_1\rangle = \frac{1}{\sqrt{2}}(|x,0\rangle + |x,1\rangle)$

2. For $|\psi_1\rangle = \frac{1}{\sqrt{2}}(|x,0\rangle - |x,1\rangle)$, the output is $|\psi_2\rangle = \frac{1}{\sqrt{2}}(|x,f(x)\rangle - |x,f(x)\oplus 1\rangle) = \frac{(-1)^{f(x)}}{\sqrt{2}}|x\rangle(|0\rangle - |1\rangle)$ where $(-1)^{f(x)} = 1$ if $f(x) = 0$ and $(-1)^{f(x)} = -1$ if $f(x) = 1$.

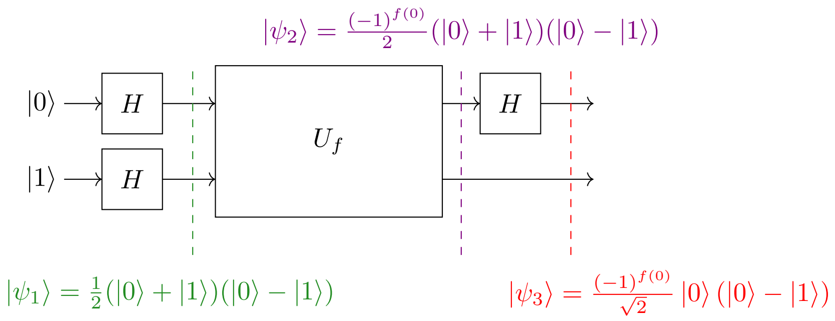

$|\psi_2\rangle = \frac{1}{\sqrt{2}}(|x,f(x)\rangle - |x,f(x)\oplus 1\rangle) = \frac{(-1)^{f(x)}}{\sqrt{2}}|x\rangle(|0\rangle - |1\rangle)$
$|x\rangle$
$|1\rangle$
H
U_f
$|\psi_1\rangle = \frac{1}{\sqrt{2}}(|x,0\rangle - |x,1\rangle)$

3. For $|\psi_1\rangle = \frac{1}{2}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle)$,
the outputs are $|\psi_2\rangle = \frac{1}{2}((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle)(|0\rangle - |1\rangle)$ and $|\psi_3\rangle = \frac{1}{\sqrt{2}}(|f(0)\oplus f(1)\rangle)(|0\rangle - |1\rangle)$.

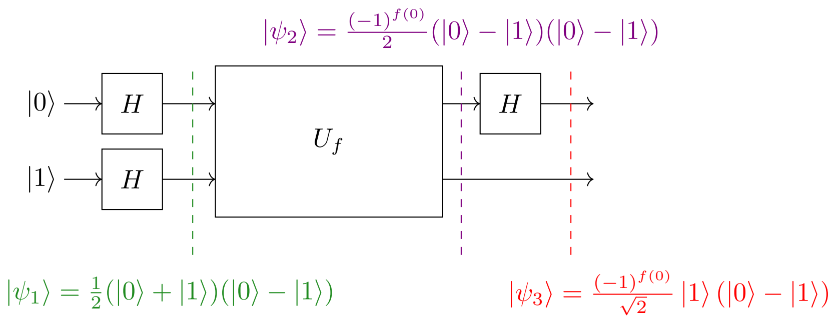
Remark: The measurement of the first qubit always outputs $|0\rangle \oplus |1\rangle$, but the measurement of the second qubit will determine if the function is constant or balanced.


$|\psi_2\rangle = \frac{1}{2}((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle)(|0\rangle - |1\rangle)$
$|0\rangle$
$|1\rangle$
H
H
U_f
H
$|\psi_1\rangle = \frac{1}{2}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle)$
$|\psi_3\rangle = \frac{1}{\sqrt{2}}(|f(0)\oplus f(1)\rangle)(|0\rangle - |1\rangle)$

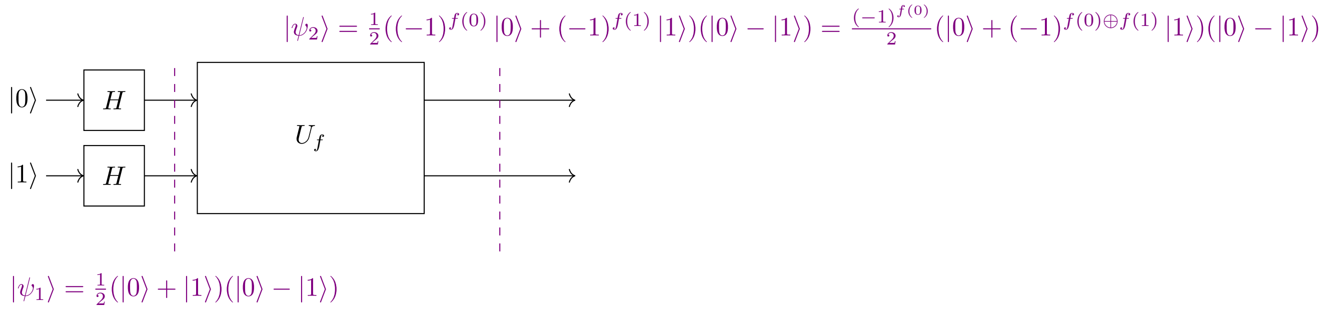
There are two cases for the outputs: $f(0) = f(1)$ and $f(0) \neq f(1)$.
Case 1: If $f(0) = f(1)$, then $|\psi_2\rangle = \frac{(-1)^{f(0)}}{2}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle)$ and $|\psi_3\rangle = \frac{(-1)^{f(0)}}{\sqrt{2}}|0\rangle(|0\rangle - |1\rangle)$.


$|\psi_2\rangle = \frac{(-1)^{f(0)}}{2}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle)$
$|0\rangle$
$|1\rangle$
H
H
U_f
H
$|\psi_1\rangle = \frac{1}{2}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle)$
$|\psi_3\rangle = \frac{(-1)^{f(0)}}{\sqrt{2}}|0\rangle(|0\rangle - |1\rangle)$

Case 2: If $f(0) \neq f(1)$, then $|\psi_2\rangle = \frac{(-1)^{f(0)}}{2}(|0\rangle - |1\rangle)(|0\rangle - |1\rangle)$ and $|\psi_3\rangle = \frac{(-1)^{f(0)}}{\sqrt{2}}|1\rangle(|0\rangle - |1\rangle)$.


$|\psi_2\rangle = \frac{(-1)^{f(0)}}{2}(|0\rangle - |1\rangle)(|0\rangle - |1\rangle)$
$|0\rangle$
$|1\rangle$
H
H
U_f
H
$|\psi_1\rangle = \frac{1}{2}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle)$
$|\psi_3\rangle = \frac{(-1)^{f(0)}}{\sqrt{2}}|1\rangle(|0\rangle - |1\rangle)$

4. For $|\psi_1\rangle = \frac{1}{2}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle)$,
the output is $|\psi_2\rangle = \frac{1}{2}((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle)(|0\rangle - |1\rangle) = \frac{(-1)^{f(0)}}{2}(|0\rangle + (-1)^{f(0)\oplus f(1)}|1\rangle)(|0\rangle - |1\rangle)$.



$|\psi_2\rangle = \frac{1}{2}((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle)(|0\rangle - |1\rangle) = \frac{(-1)^{f(0)}}{2}(|0\rangle + (-1)^{f(0)\oplus f(1)}|1\rangle)(|0\rangle - |1\rangle)$
$|0\rangle$
$|1\rangle$
H
H
U_f
$|\psi_1\rangle = \frac{1}{2}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle)$

Chapter 3

Matrix Representation of Quantum Gates and Circuits

3.1 Matrix Representation of Single-Qubit States and Gates

3.1.1 Matrix Representation and Multiplication

2×1 array representation of a single-qubit state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$:

$$\boxed{\mathrm{VEC}(|\alpha|0\rangle + \beta|1\rangle\rangle) = \begin{bmatrix}\alpha\\ \beta\end{bmatrix}}$$

Remark: We cannot write $|0\rangle = \begin{bmatrix}0\\1\end{bmatrix}$ and $|1\rangle = \begin{bmatrix}1\\0\end{bmatrix}$ as it will lose the linearity of quantum mechanics.

$\Longrightarrow$ Vector Addition of 2×1 arrays of single-qubit states:

$$\boxed{\mathrm{VEC}(\alpha|0\rangle + \beta|1\rangle) + \mathrm{VEC}(\gamma|0\rangle + \delta|1\rangle) = \begin{bmatrix}\alpha\\ \beta\end{bmatrix} + \begin{bmatrix}\gamma\\ \delta\end{bmatrix} = \begin{bmatrix}\alpha+\gamma\\ \beta+\delta\end{bmatrix}}$$

$\Longrightarrow$ Scalar Multiplication of 2×1 arrays of single-qubit states:

$$\boxed{c(\mathrm{VEC}(\alpha|0\rangle + \beta|1\rangle)) = c\begin{bmatrix}\alpha\\ \beta\end{bmatrix} = \begin{bmatrix}c\alpha\\ c\beta\end{bmatrix}}$$

2×2 matrix representation of single-qubit gates:

U Gate: $\begin{cases} U|0\rangle = u_{00}|0\rangle + u_{10}|1\rangle \\ U|1\rangle = u_{01}|0\rangle + u_{11}|1\rangle \end{cases}$

$\Longrightarrow$ 2×2 Matrix representation of the U gate: $\boxed{U = \begin{bmatrix}u_{00} & u_{01}\\ u_{10} & u_{11}\end{bmatrix}}$

X Gate: $\begin{cases} X|0\rangle = |1\rangle = 0|0\rangle + 1|1\rangle \\ X|1\rangle = |0\rangle = 1|0\rangle + 0|1\rangle \end{cases}$

$\Longrightarrow$ 2×2 Matrix representation of the X gate: $\boxed{X = \begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}}$

Z Gate: $\begin{cases} Z|0\rangle = |0\rangle = 1|0\rangle + 0|1\rangle \\ Z|1\rangle = -|1\rangle = 0|0\rangle - 1|1\rangle \end{cases}$

$\Longrightarrow$ 2×2 Matrix representation of the Z gate: $\boxed{Z = \begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}}$

Hadamard Gate: $\begin{cases} H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \\ H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle \end{cases}$

$\Longrightarrow$ 2×2 Matrix representation of the H gate: $\boxed{H = \begin{bmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}}$

Transformation of State Vectors by Quantum Gates:

U Gate: $U(\alpha|0\rangle+\beta|1\rangle)=\alpha U|0\rangle+\beta U|1\rangle=\alpha(u_{00}|0\rangle+u_{10}|1\rangle)+\beta(u_{01}|0\rangle+u_{11}|1\rangle)=(u_{00}\alpha+u_{01}\beta)|0\rangle+(u_{10}\alpha+u_{11}\beta)|1\rangle$

$\implies$ Transformation of state vectors by the U gate: $\boxed{\begin{bmatrix}u_{00} & u_{01}\\ u_{10} & u_{11}\end{bmatrix}\begin{bmatrix}\alpha\\ \beta\end{bmatrix}=\begin{bmatrix}u_{00}\alpha+u_{01}\beta\\ u_{10}\alpha+u_{11}\beta\end{bmatrix}}$

X Gate: $X(\alpha|0\rangle+\beta|1\rangle)=\alpha X|0\rangle+\beta X|1\rangle=\alpha|1\rangle+\beta|0\rangle=\beta|0\rangle+\alpha|1\rangle$

$\implies$ Transformation of state vectors by the X gate: $\boxed{\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\begin{bmatrix}\alpha\\ \beta\end{bmatrix}=\begin{bmatrix}\beta\\ \alpha\end{bmatrix}}$

Z Gate: $Z(\alpha|0\rangle+\beta|1\rangle)=\alpha Z|0\rangle+\beta Z|1\rangle=\alpha|0\rangle-\beta|1\rangle$

$\implies$ Transformation of state vectors by the Z gate: $\boxed{\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}\begin{bmatrix}\alpha\\ \beta\end{bmatrix}=\begin{bmatrix}\alpha\\ -\beta\end{bmatrix}}$

Hadamard Gate: $H(\alpha|0\rangle+\beta|1\rangle)=\alpha H|0\rangle+\beta H|1\rangle=\alpha\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)+\beta\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)=\frac{1}{\sqrt{2}}(\alpha+\beta)|0\rangle+\frac{1}{\sqrt{2}}(\alpha-\beta)|1\rangle$

$\implies$ Transformation of state vectors by the H gate: $\boxed{\begin{bmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\end{bmatrix}\begin{bmatrix}\alpha\\ \beta\end{bmatrix}=\begin{bmatrix}\frac{1}{\sqrt{2}}(\alpha+\beta)\\ \frac{1}{\sqrt{2}}(\alpha-\beta)\end{bmatrix}}$

3.1.2 Seqeuntial Gates

Definition 3.1 **Seqeuntial Gates**

Sequential Gates: If we apply two gates A and B in sequence, then the matrix representation of the combined gate is the matrix product of the individual gates, read from right to left.

$$\begin{tikzcd}|\psi\rangle \arrow[r] & \boxed{A} \arrow[r] & \boxed{B} \arrow[r] & \text{Function: } B(A(|\psi\rangle))\end{tikzcd}$$

$\implies$ Matrix representation is $\text{MAT}(B)\cdot\text{MAT}(A)\cdot\text{VEC}(|\psi\rangle)=B(A(|\psi\rangle))$.

Example 3.1 **Difference between XH and HX Circuits**

$$\begin{tikzcd}{} \arrow[r] & \boxed{H} \arrow[r] & \boxed{X} \arrow[r] & {}\end{tikzcd} \quad \text{and} \quad \begin{tikzcd}{} \arrow[r] & \boxed{X} \arrow[r] & \boxed{H} \arrow[r] & {}\end{tikzcd} \quad \text{are different.}$$

$$XH=\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\cdot\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}=\frac{1}{\sqrt{2}}\begin{bmatrix}1 & -1\\ 1 & 1\end{bmatrix}$$
$$HX=\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\cdot\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}=\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ -1 & 1\end{bmatrix}\neq XH$$

Useful Identities for Single-Qubit Gates:

1. $HZH=X$: $\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\cdot\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}\cdot\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}=\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$.
2. $HXH=Z$: $\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}\cdot\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\cdot\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}=\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}$.

3.2 Matrix Representation of Multi-Qubit States and Gates

3.2.1 Matrix Representation of Two-Qubit States and Gates

4×1 array representation of two-qubit states:

$\implies$ The two-qubit state $|\psi\rangle=a_{00}|00\rangle+a_{01}|01\rangle+a_{10}|10\rangle+a_{11}|11\rangle$ is represented as:

$$\boxed{\text{VEC}(|\psi\rangle)=a_{00}\begin{bmatrix}1\\0\\0\\0\end{bmatrix}+a_{01}\begin{bmatrix}0\\1\\0\\0\end{bmatrix}+a_{10}\begin{bmatrix}0\\0\\1\\0\end{bmatrix}+a_{11}\begin{bmatrix}0\\0\\0\\1\end{bmatrix}=\begin{bmatrix}a_{00}\\a_{01}\\a_{10}\\a_{11}\end{bmatrix}}$$

CNOT Gate 4×4 Matrix Representation:

- $|x, y\rangle$ is the input state and $|x, y \oplus x\rangle$ is the output state (First qubit is the control qubit).

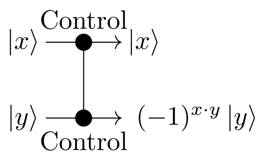
$$\begin{array}{c} \text{Output} \\ |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{array} \begin{array}{c} \begin{array}{cccc} |00\rangle & |01\rangle & |10\rangle & |11\rangle \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \begin{array}{c} \text{Input} \\ \\ \\ \\ \\ \end{array}$$

- $|x, y\rangle$ is the input state and $|x \oplus y, y\rangle$ is the output state (Second qubit is the control qubit).

$$\begin{array}{c} \text{Output} \\ |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{array} \begin{array}{c} \begin{array}{cccc} |00\rangle & |01\rangle & |10\rangle & |11\rangle \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{c} \text{Input} \\ \\ \\ \\ \\ \end{array}$$

Controlled-Z Gate 4×4 Matrix Representation:

- Given $|00\rangle \to |00\rangle$, $|01\rangle \to |01\rangle$, $|10\rangle \to |10\rangle$, $|11\rangle \to -|11\rangle$, the matrix representation is:



$$\begin{array}{c} \text{Output} \\ |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{array} \begin{array}{c} \begin{array}{cccc} |00\rangle & |01\rangle & |10\rangle & |11\rangle \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \end{array} \begin{array}{c} \text{Input} \\ \\ \\ \\ \\ \end{array}$$

3.2.2 Matrix Representation of 3-Qubit States and Gates

8×1 array representation of 3-qubit states:

$\Longrightarrow$ The 8 states of a 3-qubit system are $|000\rangle$, $|001\rangle$, $|010\rangle$, $|011\rangle$, $|100\rangle$, $|101\rangle$, $|110\rangle$, and $|111\rangle$.

$$|000\rangle \leftrightarrow \begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}, \quad |001\rangle \leftrightarrow \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}, \quad |010\rangle \leftrightarrow \begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\end{bmatrix}, \quad |011\rangle \leftrightarrow \begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\end{bmatrix}, \quad |100\rangle \leftrightarrow \begin{bmatrix}0\\0\\0\\0\\1\\0\\0\\0\end{bmatrix}, \quad |101\rangle \leftrightarrow \begin{bmatrix}0\\0\\0\\0\\0\\1\\0\\0\end{bmatrix}, \quad |110\rangle \leftrightarrow \begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\0\end{bmatrix}, \quad |111\rangle \leftrightarrow \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}$$

Toffoli Gate 8×8 Matrix Representation:

$$\begin{array}{c} \text{Output} \\ |000\rangle \\ |001\rangle \\ |010\rangle \\ |011\rangle \\ |100\rangle \\ |101\rangle \\ |110\rangle \\ |111\rangle \end{array} \begin{array}{c} \begin{array}{cccccccc} |000\rangle & |001\rangle & |010\rangle & |011\rangle & |100\rangle & |101\rangle & |110\rangle & |111\rangle \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \begin{array}{c} \text{Input} \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array}$$

3.2.3 Matrix Representation of n-Qubit States

Theorem 3.1 — **Matrix Representation of n-Qubit States**

The $2^n \times 1$ array representation of an n-qubit state $|\psi\rangle = \sum_{x=0}^{2^n-1} a_x |x\rangle$ is given by:

$$\boxed{|a_{n-1}a_{n-2}\dots a_1 a_0\rangle \leftrightarrow \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{2^n-1} \end{bmatrix}} \quad \text{where} \quad a_x \in \{0,1\}.$$

Remark: The entry indices of the $2^n \times 1$ array representation of an n-qubit state begins at 0 and ends at $2^n - 1$.

3.3 Tensor Product of Qubit States and Gates

Theorem 3.2 — **Tensor Product of Two Matrices**

Let $A = \begin{bmatrix} a_{0,0} & \dots & a_{0,n-1} \\ \vdots & \ddots & \vdots \\ a_{m-1,0} & \dots & a_{m-1,n-1} \end{bmatrix}$ be a $m \times n$ matrix and B be a $p \times q$ matrix.

$\Longrightarrow$ Tensor/Kronecker Product of A and B is a $(mp) \times (nq)$ matrix given by:

$$\boxed{A \otimes B = \begin{bmatrix} a_{0,0}B & \dots & a_{0,n-1}B \\ \vdots & \ddots & \vdots \\ a_{m-1,0}B & \dots & a_{m-1,n-1}B \end{bmatrix}}$$

3.3.1 Tensor Product of Qubit States

Theorem 3.3 — **Tensor Product of 2-Qubit States**

The two-qubit joint state (tensor product of two-qubits) $|\psi_1\rangle = a|0\rangle + b|1\rangle$ and $|\psi_2\rangle = c|0\rangle + d|1\rangle$ is given by:

$$\boxed{|\psi_1\rangle \otimes |\psi_2\rangle = (a|0\rangle + b|1\rangle) \otimes (c|0\rangle + d|1\rangle) = ac|00\rangle + ad|01\rangle + bc|10\rangle + bd|11\rangle}$$

$\Longrightarrow$ The 4×1 array representation of the two-qubit joint state is:

$$\begin{bmatrix} a \\ b \end{bmatrix} \otimes \begin{bmatrix} c \\ d \end{bmatrix} = \begin{bmatrix} a\begin{bmatrix} c \\ d \end{bmatrix} \\ b\begin{bmatrix} c \\ d \end{bmatrix} \end{bmatrix} = \begin{array}{cc} \begin{bmatrix} ac \\ ad \\ bc \\ bd \end{bmatrix} & \begin{matrix} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{matrix} \end{array} \quad \text{which is a separable state}$$

Examples of Tensor Product of 2-Qubit States:

- **e.g. 1**: $|0\rangle \otimes |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1\begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ 0\begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{array}{cc} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} & \begin{matrix} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{matrix} \end{array} = |00\rangle.$

- **e.g. 2**: $|1\rangle \otimes |0\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0\begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ 1\begin{bmatrix} 1 \\ 0 \end{bmatrix} \end{bmatrix} = \begin{array}{cc} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} & \begin{matrix} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{matrix} \end{array} = |10\rangle.$

However, for $|00\rangle + |11\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$, the state is **entangled**.

There are some useful properties of the tensor product of qubit states:

1. Non-commutative: $A \otimes B \neq B \otimes A$.
2. Associative: $A \otimes (B \otimes C) = (A \otimes B) \otimes C$.
3. Distributive: $(A \otimes B) \cdot (C \otimes D) = (A \cdot C) \otimes (B \cdot D)$ if A, C are $m \times n$ matrices and B, D are $p \times q$ matrices.

Example 3.2 **Non-Commutativeness of X and I**

Let the matrices X gate be $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ and the identity matrix be $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

$$\implies X \otimes I = \begin{bmatrix} 0\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & 1\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ 1\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & 0\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\implies I \otimes X = \begin{bmatrix} 1\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & 0\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \\ 0\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & 1\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$\therefore X \otimes I \neq I \otimes X$.

3.3.2 Tensor Product of a Single-Qubit Gate on Multi-Qubit Systems

Theorem 3.4 **Tensor Product of a Single-Qubit Gate on Multi-Qubit Systems**

If a circuit has n qubits and a single-qubit gate U acts on the k-th qubit
$\implies$ Tensor product of the n-qubit identity matrix and the single-qubit gate U is given by:

$$\boxed{U_k = I^{\otimes(k-1)} \otimes U \otimes I^{\otimes(n-k)}} \quad \text{where } I^{\otimes k-1} = I \otimes \ldots \otimes I \text{ for } k-1 \text{ times}$$

Example 3.3 **Tensor Product of a X Gate on 3-Qubit Systems**

Let the X gate be $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ and the identity matrix be $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.
$\implies$ The tensor product of the X gate on the 2nd qubit of a 3-qubit system is:

$$X_2 = I \otimes X \otimes I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

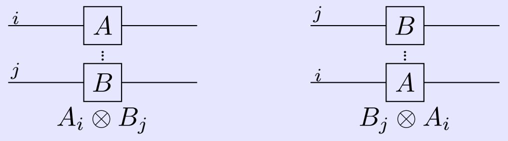
3.3.3 Parallel Gates on Multi-Qubit Systems

Theorem 3.5 **Parallel Gates on Multi-Qubit Systems**

If a circuit has a gate A on qubit i and a gate B on qubit j
$\implies$ Tensor product of the n-qubit identity matrix and the gates A and B is given by:

$$\boxed{A_i \otimes B_j = I^{\otimes(i-1)} \otimes A \otimes I^{\otimes(j-i-1)} \otimes B \otimes I^{\otimes(n-j)}} \quad \text{for } i < j$$

$$\boxed{B_j \otimes A_i = I^{\otimes(i-1)} \otimes B \otimes I^{\otimes(j-i-1)} \otimes A \otimes I^{\otimes(n-j)}} \quad \text{for } i > j$$



Example 3.6 **3 Parallel Hadamard Gates on 3-Qubit Systems**

Let the Hadamard gate be $H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$.

$\Longrightarrow$ The tensor product of the Hadamard gate on the 1st, 2nd, and 3rd qubits of a 3-qubit system is:

$$H_1 \otimes H_2 \otimes H_3 = H^{\otimes 3} = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} \otimes \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} \otimes \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix} = \frac{1}{\sqrt{8}}\begin{bmatrix}1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1\\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1\\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1\\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1\\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1\\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1\\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1\end{bmatrix}$$

For a circuit with n qubits, when we apply a CNOT gate on the i-th qubit with the j-th qubit as the control qubit:

$$\begin{aligned}\text{CNOT}_{i,j} &= |\ldots, x_i, \ldots, x_j \oplus x_i, \ldots\rangle \quad \text{for } i < j\\ \text{CNOT}_{i,j} &= |\ldots, x_j \oplus x_i, \ldots, x_i, \ldots\rangle \quad \text{for } i > j\end{aligned}$$

Example 3.7 **CNOT Gate on 3-Qubit Systems**

For CNOT gate with the 3rd qubit as the control qubit and the 1st qubit as the target qubit.

$\Longrightarrow$ The matrix representation is:

$$\text{CNOT}_{3,1} = \begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}$$

Chapter 4

Grover's Search Algorithm

4.1 Search Problem

Suppose we have an n-bit string x as input and output a function $f : \{0,1\}^n \to \{0,1\}$ such that $f(x) = 1$ if x is the solution s and $f(x) = 0$ otherwise. Our goal is to find the solution s such that $f(s) = 1$.

We access the function f with an oracle U_f which is a unitary operator that acts as:

$$U_f |x\rangle = \begin{cases} |x\rangle & \text{if } f(x) = 0 \\ -|x\rangle & \text{if } f(x) = 1 \end{cases} = (-1)^{f(x)} |x\rangle$$

Grover's Search Algorithm is a quantum algorithm that can search an unsorted database of N items in $O(\sqrt{N})$ time.

4.2 Grover's Search Algorithm

4.2.1 Steps of Grover's Search Algorithm

The steps of Grover's Search Algorithm are as follows:

1. Initialize the system to the uniform superposition of all possible states:

$$|\psi_0\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$

2. Apply the oracle U_f to the state $|\psi_0\rangle$:

$$|\psi_1\rangle = U_f |\psi_0\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} (-1)^{f(x)} |x\rangle$$

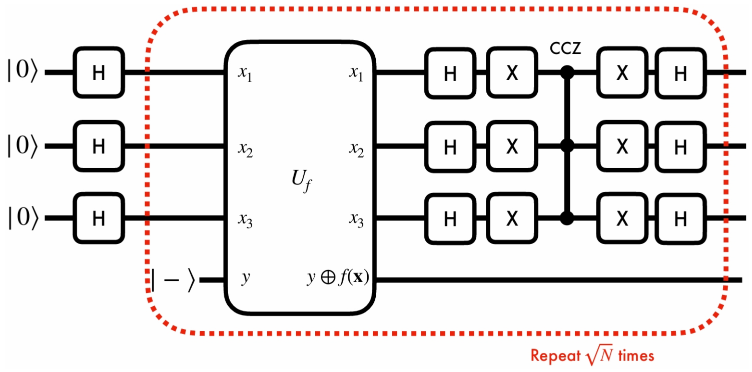
3. Apply the Grover diffusion operator G to the state $|\psi_1\rangle$:

$$G = 2 |\psi_0\rangle \langle\psi_0| - I$$

4. Repeat steps 2 and 3 for k iterations, where $k = \frac{\pi}{4}\sqrt{\frac{N}{M}}$ and M is the number of solutions to the search problem.

5. Measure the final state $|\psi_k\rangle$ to obtain the solution s.

$$|\psi_k\rangle = G^k U_f^k |\psi_0\rangle$$



4.2.2 Geometric Interpretation of Grover's Search Algorithm

First, we consider the quantum state space spanned by two vectors $|\alpha\rangle = \frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}|x\rangle$ and $|\beta\rangle = |x_0\rangle$, where $|x_0\rangle$ is the target state.
$\Longrightarrow$ The initial state $|\psi_0\rangle$ is represented as:

$$|\psi_0\rangle = \sqrt{\frac{N-1}{N}}\,|\alpha\rangle + \sqrt{\frac{1}{N}}\,|\beta\rangle = \cos(\theta)\,|\alpha\rangle + \sin(\theta)\,|\beta\rangle$$

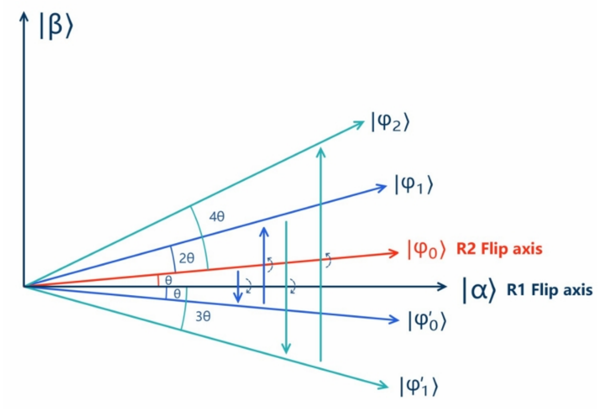
The angle between $|\psi_0\rangle$ and $|\alpha\rangle$ proves that:

$$\sin(\theta) = \sqrt{\frac{1}{N}} \quad \text{and} \quad \cos(\theta) = \sqrt{\frac{N-1}{N}} \quad \text{where } \theta = \arcsin\left(\frac{1}{\sqrt{N}}\right)$$

$\Longrightarrow$ This leads to $\sin(2\theta) = 2\sin(\theta)\cos(\theta) = \frac{2\sqrt{N-1}}{N}$.

Here is the operations of Grover's Search Algorithm with the geometric interpretation:

- After the first operation (from function f to operator U_f), the state $|\psi_0\rangle$ is transformed to $|\psi'\rangle$ by applying a phase flip on the target state $|\beta\rangle$:
$$|\psi'\rangle = U_f\,|\psi_0\rangle = \sqrt{\frac{N-1}{N}}\,|\alpha\rangle - \sqrt{\frac{1}{N}}\,|\beta\rangle = \cos(\theta)\,|\alpha\rangle - \sin(\theta)\,|\beta\rangle$$
- After the second operation (inversion about the mean), the state $|\psi'\rangle$ is transformed to $|\psi_1\rangle$ by reflecting through the average amplitude, flipping around the initial state $|\psi_0\rangle$:
$$|\psi_1\rangle = G\,|\psi'\rangle = \sqrt{\frac{N-1}{N}}\,|\alpha\rangle + \sqrt{\frac{1}{N}}\,|\beta\rangle = \cos(\theta)\,|\alpha\rangle + \sin(\theta)\,|\beta\rangle$$
- Each iteration of Grover's Search Algorithm consists of applying the oracle U_f and the Grover diffusion operator G to the state $|\psi_k\rangle$, which rotates the state vector by an angle of 2θ in the direction of $|\beta\rangle$:
$$|\psi_{k+1}\rangle = GU_f\,|\psi_k\rangle = \cos((k+1)\theta)\,|\alpha\rangle + \sin((k+1)\theta)\,|\beta\rangle$$



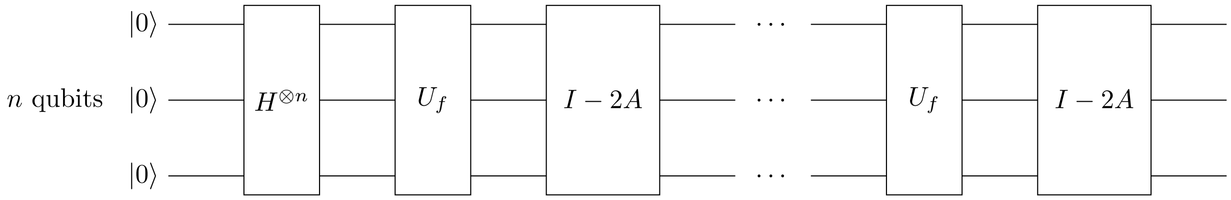
For large N, the angle θ is small, and the number of iterations k is approximately $\frac{\pi}{4}\sqrt{N}$ to rotate the state vector $|\beta\rangle$ with a probability of $\frac{1}{2}$ to the target state $|x_0\rangle$.

4.2.3 Circuit Representation of Grover's Search Algorithm

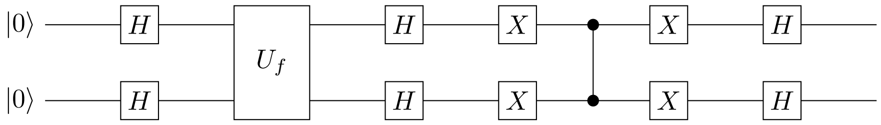
The circuit of Grover's Search Algorithm for N iterations contains the following components:

- Hadamard gates on all qubits to create the uniform superposition of all possible states.
- Oracle U_f to mark the target state.
- Gate $I - 2A$ to invert the amplitude of the target state.
- Repeat the oracle and inversion steps for $\sqrt{2^N}$ iterations.

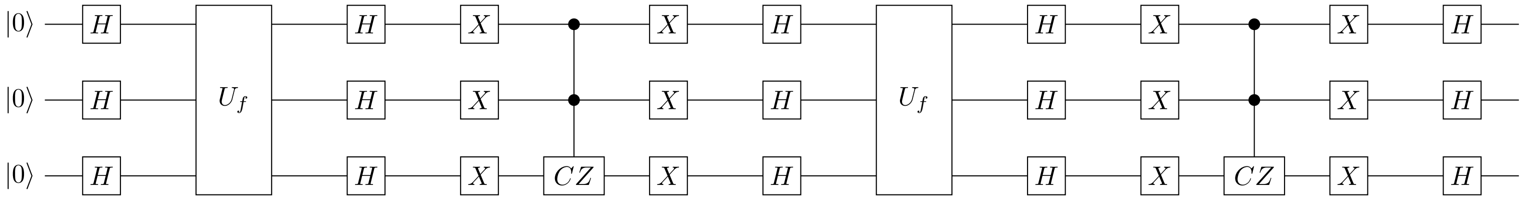
The circuit representation of Grover's Search Algorithm is as follows:



We mainly consider the cases for $N = 2$ and $N = 3$ qubits.
For $N = 2$ qubits, the circuit representation of Grover's Search Algorithm is as follows:



For $N = 3$ qubits, the circuit representation of Grover's Search Algorithm is as follows:



Chapter 5

Complex Numbers and Bloch Sphere

5.1 Complex Numbers

5.1.1 Definition and Operations of Complex Numbers

Definition 5.1 **Complex Numbers**

A complex number is a number of the form $\boxed{z = a + bi}$ where $a, b \in \mathbb{R}$ and $\boxed{i = \sqrt{-1}}$.
The real part of z is denoted by $\Re(z) = a$ and the imaginary part of z is denoted by $\Im(z) = b$.

The operations on complex numbers are as follows:

- Addition: $z_1 + z_2 = (a_1 + a_2) + (b_1 + b_2)i$
- Subtraction: $z_1 - z_2 = (a_1 - a_2) + (b_1 - b_2)i$
- Multiplication: $z_1 \cdot z_2 = (a_1a_2 - b_1b_2) + (a_1b_2 + a_2b_1)i$
- Division: $\frac{z_1}{z_2} = \frac{(a_1a_2+b_1b_2)+(b_1a_2-a_1b_2)i}{a^2+b^2}$
- Modulus: $|z| = \sqrt{a^2 + b^2}$
- Argument: $\arg(z) = \tan^{-1}\left(\frac{b}{a}\right)$, which is the angle between the positive real axis and the line connecting the origin to the point (a, b) in the complex plane.
- Complex Conjugate: $\bar{z} = a - bi$
- Exponential Form: $z = re^{i\theta}$ where $r = |z|$ and $\theta = \arg(z)$
- Polar Form: $z = r(\cos(\theta) + i\sin(\theta))$ where $r = |z|$ and $\theta = \arg(z)$

The 4 roots of unity are the complex numbers that satisfy the equation $z^4 = 1$. They are $1, i, -1, -i$ and can be represented in polar form as:

$$\boxed{z_0 = 1 = e^{i0}} \quad , \quad \boxed{z_1 = i = e^{i\frac{\pi}{2}}} \quad , \quad \boxed{z_2 = -1 = e^{i\pi}} \quad , \quad \boxed{z_3 = -i = e^{i\frac{3\pi}{2}}}$$

5.1.2 Euler's Formula

Theorem 5.1 **Euler's Formula**

Euler's formula states that for any real number θ:

$$\boxed{e^{i\theta} = \cos(\theta) + i\sin(\theta)}$$

$\Longrightarrow$ Relate the complex exponential function to the trigonometric functions cosine and sine.

The proof of Euler's formula can be derived from the Taylor series expansions of the exponential function, cosine, and sine:

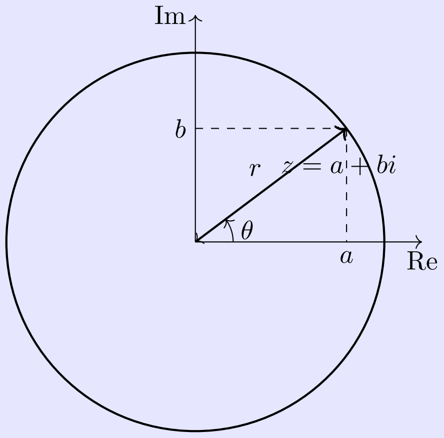
$$e^{i\theta} = \sum_{n=0}^{\infty} \frac{(i\theta)^n}{n!} = 1 + i\theta + \frac{(i\theta)^2}{2!} + \frac{(i\theta)^3}{3!} + \cdots = 1 + i\theta - \frac{\theta^2}{2!} - i\frac{\theta^3}{3!} + \cdots = \left(1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \cdots\right) + i\left(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \cdots\right)$$
$$= \cos(\theta) + i\sin(\theta)$$

5.1.3 Polar Form of Complex Numbers

Theorem 5.2 **Polar Form of Complex Numbers**

The polar form of a complex number $z = a + bi$ is given by:

$$\boxed{z = re^{i\theta}} \quad \text{where} \boxed{r = |z| = \sqrt{a^2 + b^2}} \text{ and } \boxed{\theta = \arg(z) = \tan^{-1}\left(\frac{b}{a}\right)}$$



The polar form of complex numbers allows us to express them in terms of their modulus and argument, which can simplify calculations involving multiplication and division.

- Multiplication: $z_1 \cdot z_2 = r_1 r_2 e^{i(\theta_1 + \theta_2)}$
- Division: $\frac{z_1}{z_2} = \frac{r_1}{r_2} e^{i(\theta_1 - \theta_2)}$
- Conjugate: $\bar{z} = re^{-i\theta}$
- Modulus: $|z| = r$
- Argument: $\arg(z) = \theta$

5.2 Bloch Sphere

5.2.1 Definition and Representation of Bloch Sphere

Definition 5.2 **Bloch Sphere**

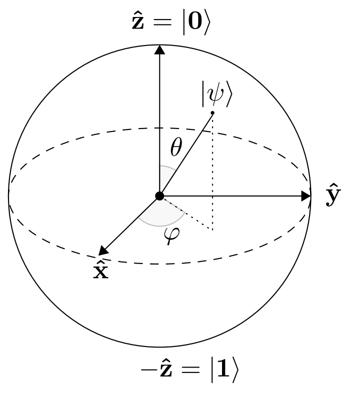
The Bloch sphere is a geometrical representation of a qubit state in a three-dimensional space.
A qubit state can be represented as:

$$\boxed{|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle}$$

where θ is the polar angle and ϕ is the azimuthal angle.

The Bloch sphere is a unit sphere in three-dimensional space, which has the following properties:

- The north pole of the Bloch sphere represents the state $|0\rangle$.
- The south pole of the Bloch sphere represents the state $|1\rangle$.
- The equator of the Bloch sphere represents the superposition states of the qubit.
- The points on the surface of the Bloch sphere represent all possible pure states of a qubit.
- The center of the Bloch sphere represents the mixed states of a qubit.
- The radius of the Bloch sphere is 1, which represents the normalization condition of the qubit state.
- The angles θ and ϕ are related to the probability amplitudes of the qubit state.



$\hat{\mathbf{z}} = |\mathbf{0}\rangle$
$|\psi\rangle$
θ
$\hat{\mathbf{y}}$
φ
$\hat{\mathbf{x}}$
$-\hat{\mathbf{z}} = |\mathbf{1}\rangle$

5.2.2 Operations on the Bloch Sphere

The sphereical coordinates of the Bloch sphere can be expressed in terms of the Cartesian coordinates as:

$$\boxed{x = \sin(\theta)\cos(\phi)} \quad , \quad \boxed{y = \sin(\theta)\sin(\phi)} \quad \text{and} \quad \boxed{z = \cos(\theta)}$$

$\implies$ The squared radius of the Bloch sphere is $\boxed{r^2 = x^2 + y^2 + z^2 = 1}$.

For X, Y, Z gates,

5.2.3 Rotations on the Bloch Sphere

Theorem 5.3 **Rotations on the Bloch Sphere**

The rotation of a qubit state on the Bloch sphere can be represented by the following unitary operators:

- Rotation around the z-axis:
$$R_z(\phi) = e^{-i\frac{\phi}{2}\sigma_z} = \cos\left(\frac{\phi}{2}\right) I - i\sin\left(\frac{\phi}{2}\right)\sigma_z$$
- Rotation around the x-axis:
$$R_x(\theta) = e^{-i\frac{\theta}{2}\sigma_x} = \cos\left(\frac{\theta}{2}\right) I - i\sin\left(\frac{\theta}{2}\right)\sigma_x$$
- Rotation around the y-axis:
$$R_y(\theta) = e^{-i\frac{\theta}{2}\sigma_y} = \cos\left(\frac{\theta}{2}\right) I - i\sin\left(\frac{\theta}{2}\right)\sigma_y$$

where σ_x, σ_y, and σ_z are the Pauli matrices.

Chapter 6

Quantum Gates as Unitary Operators

6.1 Unitary Matrices

Definition 6.1 **Unitary Matrices**

A matrix $U \in \mathbb{C}^{n\times n}$ is **unitary** if $\boxed{UU^\dagger = U^\dagger U = I_n}$ where $U^\dagger$ is the conjugate transpose (adjoint) of U.

Unitary matrices preserve inner products and correspond to reversible quantum evolutions.

There are two examples of unitary matrices:

- **Pauli Matrices** (all unitary and Hermitian): $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$.
- **Rotation Matrices** (unitary): $R_z(\theta) = \begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix}$.

There are some properties of any unitary matrix U:

- Invertibility: $\boxed{UU^{-1} = U^1 U = I_n}$, which means $U^{-1} = U^\dagger$.
- Transpose: $\boxed{U^T[j,k] = U[k,j]}$.
- Conjugate: $\boxed{\overline{U}[j,k] = \overline{U[j,k]}}$.
- Adjoint (Conjugate Transpose): $\boxed{U^\dagger[j,k] = \overline{U[k,j]}}$.

6.2 Quantum Controlled Gates

6.2.1 Controlled Gates' Matrices

There are some controlled gates that are used in quantum computing. The controlled gates are unitary operations that act on two qubits, where one qubit is the control qubit and the other is the target qubit. The controlled gates can be represented by their matrices.
The controlled gates are defined as follows:

- **Controlled-NOT (CNOT) / Controlled-X (CX)**: $\boxed{\text{CNOT}\,|x\rangle\,|y\rangle = |x\rangle\,|y \oplus x\rangle}$, where $\oplus$ is the XOR operation.
  $\implies$ The CNOT gate matrix is:
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

- **Controlled-Z (CZ)**: $\boxed{\text{CZ}\,|x\rangle\,|y\rangle = (-1)^{xy}\,|x\rangle\,|y\rangle}$.
  $\Longrightarrow$ The CZ gate matrix is:
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

- **Controlled-U (CU)**: $\boxed{\bar{0}\,|0\rangle \otimes I + \bar{1}\,|1\rangle \otimes U}$, where U is a unitary matrix.
  The CU gate matrix is:
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{bmatrix}$$

6.2.2 Decomposition of Controlled Gates

Any controlled-U can be constructed from CNOT and single-qubit gates. For any single-qubit unitary U, there exist single-qubit gates A, B, C and a phase α such that
$$U = e^{i\alpha} AXBXC, \quad \text{with } ABC = I$$
where X is the Pauli-X gate. The decomposition is:
$$\begin{aligned} A &= R_z(\beta)R_y(\gamma/2) \\ B &= R_y(-\gamma/2)R_z(-(\delta+\beta)/2) \\ C &= R_z((\delta-\beta)/2) \end{aligned}$$

6.3 Universality of Quantum Gates

6.3.1 Universal Gate Sets

Theorem 6.1 **Universality of Quantum Gates**

Any unitary operation on n qubits can be approximated to arbitrary accuracy using only:

- Single-qubit gates
- CNOT gates

6.3.2 Examples of Universal Gate Sets

- $\{H, T, \text{CNOT}\}$, where $T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$
- $\{\text{Toffoli}, H\}$

6.3.3 Toffoli Gate Construction

The Toffoli (CCNOT) gate can be constructed from controlled-Z and Hadamard gates:
$$\text{Toffoli} = (I \otimes H)\,\text{CCZ}\,(I \otimes H)$$
where CCZ is the doubly-controlled-Z gate.

Chapter 7

Quantum Fourier Transform

7.1 Definition of Quantum Fourier Transform

Definition 7.1 **Quantum Fourier Transform**

The Quantum Fourier Transform (QFT) is a quantum algorithm that linearly transforms a quantum state from the computational basis to the Fourier basis.
The QFT of an n-qubit state $|x\rangle$ is defined as:

$$\boxed{QFT(|x\rangle) = \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{2\pi i \frac{xy}{2^n}} |k\rangle}$$

where x is the input state and k is the output state.

The QFT can be represented by an unitary matrix U_{QFT}, which is an $N \times N$ matrix where $N = 2^n$:

$$\boxed{U_{QFT} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix}} \quad \text{where } \omega = e^{2\pi i/N}$$

The QFT can be implemented using a series of Hadamard gates and controlled phase gates. The QFT algorithm has a time complexity of $O(n^2)$, which is exponentially faster than the classical Fourier transform algorithm with a time complexity of $O(n2^n)$.

Chapter 8

Quantum Communication and Cryptography

8.1 Classical and Quantum Key Distribution

8.1.1 One-Time Pad Protocol (Classical)

A one-time pad is a classical encryption method that provides perfect secrecy if the key is truly random, used only once, and kept secret.

- Let T be the original message (a binary string of length n).
- Let K be a random key of length n shared by Alice and Bob.
- Alice computes the encrypted message $E = T \oplus K$ and sends E over a public channel.
- Bob decrypts by computing $T = E \oplus K$.

8.1.2 Limitations of Classical Key Distribution

In classical communication, an eavesdropper (Eve) can copy the transmitted data without detection and store it for later analysis. This makes secure key distribution challenging.

8.1.3 Quantum Key Distribution (QKD) Motivation

Quantum mechanics introduces two key properties:

- **No-Cloning Theorem**: It is impossible to make a perfect copy of an unknown quantum state.
- **Measurement Disturbs State**: Measuring a quantum state generally changes it, so eavesdropping can be detected.

8.2 Linear Algebra Concepts for Quantum Information

8.2.1 Vectors and Linear Combinations

A vector V is a linear combination of vectors $V_0, V_1, \ldots, V_{n-1}$ if

$$V = c_0V_0 + c_1V_1 + \cdots + c_{n-1}V_{n-1}$$

for some complex numbers $c_0, \ldots, c_{n-1}$.

8.2.2 Linear Independence and Basis

A set of vectors $[V_0, V_1, \ldots, V_{n-1}]$ is linearly independent if the only solution to

$$0 = c_0V_0 + c_1V_1 + \cdots + c_{n-1}V_{n-1}$$

is $c_0 = c_1 = \cdots = c_{n-1} = 0$.
A **basis** is a set of linearly independent vectors such that any vector in the space can be written as a linear combination of them.

8.2.3 Standard Basis

For $\mathbb{R}^n$ or $\mathbb{C}^n$, the standard basis is

$$E_0 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad E_1 = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \quad \ldots, \quad E_{n-1} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

Any vector $[c_0, c_1, \ldots, c_{n-1}]^T$ can be written as $\sum_{j=0}^{n-1} c_j E_j$.

8.2.4 Inner Product and Orthonormal Basis

The inner product of $V_1 = [r_0, \ldots, r_m]^T$ and $V_2 = [r'_0, \ldots, r'_m]^T$ is

$$\langle V_1, V_2 \rangle = \sum_{j=0}^{m} \bar{r}_j r'_j$$

Vectors are orthogonal if their inner product is zero. An orthonormal basis is a set of vectors that are both orthogonal and of unit length.

8.3 Quantum Measurement in Different Bases

Any quantum state $|\psi\rangle$ can be written in an orthonormal basis $\{|e_i\rangle\}$ as

$$|\psi\rangle = \sum_i c_i |e_i\rangle$$

Measuring $|\psi\rangle$ in this basis yields outcome $|e_i\rangle$ with probability

$$p_i = \frac{|c_i|^2}{\sum_j |c_j|^2}$$

8.4 The BB84 Quantum Key Distribution Protocol

8.4.1 Encoding with Two Bases

The BB84 protocol uses two bases for encoding qubits:

- **Z basis** $(+)$: $\{|0\rangle, |1\rangle\} = \{[1,0]^T, [0,1]^T\}$
- **X basis** $(\times)$: $\left\{\frac{1}{\sqrt{2}}([1,1]^T), \frac{1}{\sqrt{2}}([-1,1]^T)\right\}$

8.4.2 Protocol Steps

1. Alice randomly chooses a bit value and a basis for each qubit, prepares the qubit accordingly, and sends it to Bob.
2. Bob randomly chooses a basis for each qubit and measures.
3. Alice and Bob publicly compare which bases they used and keep only the bits where their bases matched.
4. To check for eavesdropping, Bob reveals a random subset of his bits; Alice checks if they match her original bits. If too many errors are found, they abort.

8.5 The Bell Basis and Quantum Entanglement

The Bell basis consists of four maximally entangled two-qubit states:

$$|\Psi^+\rangle = \frac{|01\rangle + |10\rangle}{\sqrt{2}}$$
$$|\Psi^-\rangle = \frac{|01\rangle - |10\rangle}{\sqrt{2}}$$
$$|\Phi^+\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}$$
$$|\Phi^-\rangle = \frac{|00\rangle - |11\rangle}{\sqrt{2}}$$

8.6 Quantum Teleportation Protocol

Quantum teleportation allows the transfer of an unknown quantum state using entanglement and classical communication.

1. Alice and Bob share an entangled pair in the $|\Phi^+\rangle$ Bell state.
2. Alice combines her unknown state $|\psi\rangle$ with her half of the entangled pair and performs a joint measurement (Bell measurement).
3. Alice sends the result (two classical bits) to Bob.
4. Bob applies a corresponding unitary operation to his qubit to reconstruct $|\psi\rangle$.

Bob's Correction Operations

Alice's bits	Bob applies
00	$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
01	$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
10	$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
11	$XZ = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$

Chapter 9

Building Quantum Computers

9.1 Physical Realization of Quantum Computers: The Circuit Model

DiVincenzo Criteria

To build a quantum computer, a physical system must satisfy the following criteria (DiVincenzo, 2000):

- **Scalable qubits**: A physical system with well-characterized, scalable qubits.
- **Initialization**: Ability to initialize qubits to a known fiducial state.
- **Long coherence times**: Decoherence times much longer than gate-operation times.
- **Universal gates**: Ability to implement a universal set of quantum gates.
- **Measurement**: Qubit-specific measurement capability.

9.2 Quantum Dynamics and Unitary Evolution

Postulate: Unitary Evolution

The evolution of a closed quantum system (excluding measurement) is described by a unitary operator U. If $|\psi(t)\rangle$ is the state at time t, then

$$|\psi(t+1)\rangle = U|\psi(t)\rangle.$$

Schrödinger's Equation

The time evolution of a quantum state is governed by Schrödinger's equation. For a small time increment Δt:

$$\begin{aligned}|\psi(t+\Delta t)\rangle - |\psi(t)\rangle &= -iH\Delta t\,|\psi(t)\rangle\\ |\psi(t+\Delta t)\rangle &= (I - iH\Delta t)\,|\psi(t)\rangle\\ &\approx e^{-iH\Delta t}|\psi(t)\rangle\end{aligned}$$

Thus, the unitary evolution operator is

$$U = e^{-iH\Delta t}$$

where H is the Hamiltonian (energy operator) of the system.

9.3 Hamiltonians and Quantum Gates

Hamiltonian: Definition and Examples

The Hamiltonian H describes the total energy of the system. Examples:

- **Classical:**
  - Kinetic: $H = \frac{p^2}{2m}$
  - Gravitational: $H = mgh$
  - Elastic: $H = \frac{1}{2}kx^2$
- **Quantum (spin in magnetic field B_0):**

$$H = -\frac{1}{2}\hbar\gamma B_0 Z = -\frac{1}{2}\hbar\omega_0 Z = \begin{bmatrix} -\frac{1}{2}\hbar\omega_0 & 0 \\ 0 & \frac{1}{2}\hbar\omega_0 \end{bmatrix}$$

9.4 Hermitian and Unitary Matrices

Hermitian Matrices

A matrix $A \in \mathbb{C}^{n\times n}$ is:

- **Symmetric** if $A^T = A$ ($A[j,k] = A[k,j]$).
- **Hermitian** if $A^\dagger = A$ ($A[j,k] = \overline{A[k,j]}$).

Examples:

$$\begin{bmatrix} 5 & 4+5i & 6-16i \\ 4-5i & 13 & 7 \\ 6+16i & 7 & -2.1 \end{bmatrix}$$

is Hermitian.
The Pauli matrices

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

are Hermitian.

Unitary Matrices

A matrix $U \in \mathbb{C}^{n\times n}$ is **unitary** if

$$UU^\dagger = U^\dagger U = I_n$$

Property: If H is Hermitian, then e^{iH} is unitary:

$$(e^{iH})^\dagger e^{iH} = e^{-iH}e^{iH} = I_n$$

Physical meaning: To realize a quantum gate $e^{-iH\Delta t}$, engineer the system Hamiltonian H for time Δt.

9.5 NMR Quantum Computing Example

NMR System and DiVincenzo Criteria

NMR (Nuclear Magnetic Resonance) systems satisfy the DiVincenzo criteria:

- ✓Qubits: nuclear spins in B_0 field ($\uparrow$ and $\downarrow$ as 0 and 1)
- ✓Quantum gates: RF pulses and delays
- (✓) Initialization: Boltzmann distribution at room temperature
- ✓Measurement: RF coil detection
- ✓Coherence times: several seconds

Single Spin System

Hamiltonian for a single spin in B_0:

$$H = -\frac{1}{2}\hbar\gamma B_0 Z = -\frac{1}{2}\hbar\omega_0 Z = \begin{bmatrix} -\frac{1}{2}\hbar\omega_0 & 0 \\ 0 & \frac{1}{2}\hbar\omega_0 \end{bmatrix}$$

Control Hamiltonian (RF Pulses)

In the rotating frame, the control Hamiltonian is:

$$H_c = -\frac{1}{2}\hbar\omega_1(\cos\phi\, X + \sin\phi\, Y)$$

where $\omega_1 = \gamma B_1$ (RF amplitude), and ϕ is the phase. The evolution for pulse width t_{pw}:

$$e^{-iH_c t_{pw}} = e^{-i\frac{1}{2}\hbar\omega_1(\cos\phi\, X+\sin\phi\, Y)t_{pw}}$$

9.6 Single-Qubit Gates as Rotations

$$R_x(\theta) = e^{-i\frac{\theta}{2}X} = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}X = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}$$

$$R_y(\theta) = e^{-i\frac{\theta}{2}Y} = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}Y = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}$$

$$R_z(\theta) = e^{-i\frac{\theta}{2}Z} = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}Z = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}$$

A general rotation about axis $\vec{D}$:

$$R_{\vec{D}}(\theta) = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}(D_xX + D_yY + D_zZ) = \exp\left(-i\frac{\theta}{2}\vec{D}\cdot\vec{\sigma}\right)$$

General Single-Qubit Gate Decomposition

Any single-qubit gate U can be written as:

$$U = e^{i\alpha}Z_\beta Y_\gamma Z_\delta$$

for real $\alpha, \beta, \gamma, \delta$, or similarly with X and Y rotations.

9.7 Two-Qubit Interactions and Gates

Coupling Hamiltonian

For two qubits, the coupling Hamiltonian is:

$$H_J = \hbar\frac{\pi}{2}JZ \otimes Z$$

The evolution operator:

$$U_J(t) = e^{-i\hbar\frac{\pi}{2}JtZ\otimes Z}$$

For $t = \frac{1}{2\hbar J}$:

$$U_J\left(\frac{1}{2\hbar J}\right) = e^{-i\frac{\pi}{4}Z\otimes Z}$$

Controlled-Z and Controlled-NOT Gates

Controlled-Z:

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$$

$$CZ = e^{i\frac{\pi}{4}(S^\dagger\otimes S^\dagger)U_J\left(\frac{1}{2\hbar J}\right)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

Controlled-NOT:

$$CNOT = e^{i\frac{\pi}{4}(I\otimes H)(S^\dagger\otimes S^\dagger)U_J\left(\frac{1}{2\hbar J}\right)(I\otimes H)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Chapter 10

Classical Coding Theory

Brief Introduction to Information Theory

Definition 10.1 **Information Theory**

Information theory is the mathematical framework for understanding the quantification, storage, and communication of information by Claude Shannon in 1948.

There are different forms of communication to exchange information more efficiently, such as:
$\Longrightarrow$ **e.g.**: Talking, Beacon Towers, Writing Letters, Mail with horses/pigeons, Telegraph, Telephone, Radio, Television, Email.

However, communication is not always perfect, which encounter these challenges:

1. Efficiency: Minimize the resources used to transmit information.
2. Correctness: Ensure that the information is transmitted accurately.
3. Security: Protect the information from unauthorized access or tampering.

We can use information theory to solve these challenges by Entropy, Coding Theory, and Cryptography.

10.1 Shannon Entropy

10.1.1 Definition of Shannon Entropy

Definition 10.2 **Shannon Entropy**

The Shannon entropy is a measure of the average uncertainty or information content associated with a random variable X that can take on N possible values $\{x_1, x_2, \ldots, x_N\}$ with probabilities $\{p(x_1), p(x_2), \ldots, p(x_N)\}$:

$$\boxed{H(X) = -\sum_{i=1}^{N} p(x_i)\log_2 p(x_i)}$$ where $H(X)$ is the Shannon entropy of the random variable X.

Since the entropy represents the average number of bits needed to encode the information, less predictable sources of information requires more bits to encode than more predictable sources.

Example 10.1 **Uniform and Non-Uniform Distribution**

Uniform Distribution:

- A fair coin toss has two possible outcomes: heads or tails, each with a probability of $\frac{1}{2}$.
  $\Longrightarrow$ The Shannon entropy is $H(X) = -\left(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}\right) = 1$.

Non-Uniform Distribution:

- A biased coin toss has a probability of $\frac{3}{4}$ for heads and $\frac{1}{4}$ for tails.
  $\Longrightarrow$ The Shannon entropy is $H(X) = -\left(\frac{3}{4}\log_2\frac{3}{4} + \frac{1}{4}\log_2\frac{1}{4}\right) = 0.81$.

$\therefore$ The uniform distribution has a higher Shannon entropy than the non-uniform distribution, indicating that it is less predictable and requires more bits to encode the information.

10.1.2 Shannon's Noiseless Coding Theorem

Theorem 10.1 **Shannon's Noiseless Coding Theorem**

Shannon's noiseless coding theorem states that for a discrete memoryless source with entropy $H(X)$, the average code length L of a prefix code satisfies:

$$\boxed{H(X) \leq L < H(X) + 1}$$

The Theorem implies that the average code length of a prefix code can be made arbitrarily close to the entropy of the source, but it cannot be less than the entropy.

- Lower Bound: The average code length cannot be less than the entropy of the source.
- Upper Bound: The average code length can be at most one bit longer than the entropy of the source.

If the entropy limit is approached, the average code length can be made arbitrarily close to the entropy of the source by these methods:

- Encode pairs, triples, or larger blocks of symbols instead of single symbols.
- Use variable-length codes instead of fixed-length codes.
- Use more efficient and modern compression algorithms like Huffman coding, Arithmetic coding, and Lempel-Ziv coding.

10.1.3 Coding Efficiency of Uniform and Non-Uniform Distribution

Theorem 10.2 **Coding Efficiency**

The coding efficiency is measured by its average code length L, which is defined as:

$$\boxed{L = \sum_{i=1}^{N} p(x_i) l(x_i)}$$

where $l(x_i)$ is the length of the codeword for the symbol x_i.

From find the average code length L of a prefix code, we can determine the coding scheme tailored to the distribution of the source symbols:

- Uniform Distribution: The average code length is equal to the entropy of the source, $L = H(X)$.
- Non-Uniform Distribution: The average code length is less than the entropy of the source, $L < H(X)$.

Example 10.2 **Coding Efficiency of Uniform and Non-Uniform Distribution**

Given two schemes of coding:

- Scheme 1: $x_1 = 00$, $x_2 = 01$, $x_3 = 10$, $x_4 = 11$ with probabilities $\frac{1}{4}$, $\frac{1}{4}$, $\frac{1}{4}$, and $\frac{1}{4}$ respectively.
- Scheme 2: $x_1 = 0$, $x_2 = 10$, $x_3 = 110$, $x_4 = 111$ with probabilities $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, and $\frac{1}{8}$ respectively.

Scheme 1:

- The average code length is $L = \frac{1}{4} \cdot 2 + \frac{1}{4} \cdot 2 + \frac{1}{4} \cdot 2 + \frac{1}{4} \cdot 2 = 2$.
- The entropy is $H(X) = -\left(\frac{1}{4}\log_2\frac{1}{4} + \frac{1}{4}\log_2\frac{1}{4} + \frac{1}{4}\log_2\frac{1}{4} + \frac{1}{4}\log_2\frac{1}{4}\right) = 2$.

Scheme 2:

- The average code length is $L = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 = 1.75$.
- The entropy is $H(X) = -\left(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{4}\log_2\frac{1}{4} + \frac{1}{8}\log_2\frac{1}{8} + \frac{1}{8}\log_2\frac{1}{8}\right) = 1.75$.

$\therefore$ The coding efficiency of Scheme 1 is equal to the entropy of the source, while the coding efficiency of Scheme 2 is less than the entropy of the source.

10.2 Huffman Coding

Definition 10.3 — **Huffman Coding**

Huffman coding is a lossless data compression algorithm generating optimal prefix codes for given symbols and their probabilities.
$\implies$ Prefix codes are codes in which no codeword is a prefix of any other codeword, ensuring that the codes can be uniquely decoded.

The Huffman coding algorithm works as follows:

1. Start with all symbols as leaf nodes in a binary tree with their corresponding probabilities.
2. Select the two nodes with the lowest probabilities and combine them into a new node with a probability equal to the sum of the two selected nodes.
3. Repeat the process until there is only one node left, which becomes the root of the tree.
4. Assign binary codes to each symbol by traversing the tree: assign 1 to the left branch and 0 to the right branch.
5. The resulting binary codes are the Huffman codes for the symbols.

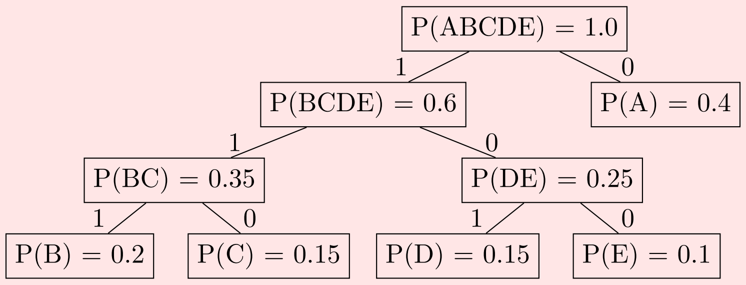
Example 10.3 — **Huffman Coding and its Performance**

Consider a source with the following symbols and their probabilities:

$$P(A) = 0.4, \quad P(B) = 0.2, \quad P(C) = 0.15, \quad P(D) = 0.15, \quad P(E) = 0.1$$

- **Step 1**: Combine the two symbols with the lowest probabilities iteratively:
  - Combine $P(D) = 0.15$ and $P(E) = 0.1$ into $P(DE) = 0.25$.
  - Combine $P(C) = 0.15$ and $P(B) = 0.2$ into $P(BC) = 0.35$.
  - Combine $P(DE) = 0.25$ and $P(BC) = 0.35$ into $P(BCDE) = 0.6$.
  - Combine $P(A) = 0.4$ and $P(BCDE) = 0.6$ into $P(ABCDE) = 1.0$.
- **Step 2**: Construct the final binary tree with $P(ABCDE) = 1.0$ as the root node.

The Huffman coding algorithm generates the following binary tree:



The resulting Huffman codes for the symbols are:

$$A = 0, \quad B = 111, \quad C = 110, \quad D = 101, \quad E = 100$$

The average code length is:

$$L = 0.4 \cdot 1 + 0.2 \cdot 3 + 0.15 \cdot 3 + 0.15 \cdot 3 + 0.1 \cdot 3 = 0.4 + 0.6 + 0.45 + 0.45 + 0.3 = 2.2 \text{ bits}$$

The entropy of the source is:

$$H(X) = -\left(0.4 \log_2 0.4 + 0.2 \log_2 0.2 + 0.15 \log_2 0.15 + 0.15 \log_2 0.15 + 0.1 \log_2 0.1\right) \approx 2.17 \text{ bits}$$

$\therefore$ The average code length of the Huffman code is close to the entropy of the source, indicating that Huffman coding is an efficient method for compressing data.

$\implies$ The Huffman coding algorithm is optimal for the given symbols and their probabilities, which satisfies Shannon's noiseless coding theorem.

10.3 Noisy Channels

10.3.1 Introduction to Noisy Channels

In practical communication systems, the received messages may not match the transmitted messages due to noise, interference, or other factors.

There are various sources of noise in communication systems, such as:
$\Longrightarrow$ **e.g.**: Thermal noise, Electromagnetic interference, Cosmic radiation, Physical obstacles, Component degradation and Cross-talk.

To analyze the communication over noisy channels, we can use the transition probability matrix, the definition is as follows:

Definition 10.4 **Transition Probability Matrix**

The transition probability matrix $P(y|x)$ is a matrix that describes the probabilities of transitioning from one state x to another state y in a communication system.
$\Longrightarrow$ The elements of the matrix are defined as:

$$\boxed{P(y|x) = P(Y = y|X = x) = \begin{bmatrix} P(Y=0|X=0) & P(Y=0|X=1) \\ P(Y=1|X=0) & P(Y=1|X=1) \end{bmatrix}}$$

where $P(Y = y|X = x)$ is the conditional probability of receiving y given that x was transmitted.

10.3.2 Channel Modeling

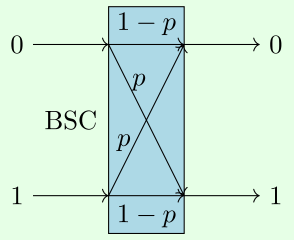
Binary Symmetric Channel (BSC)

Definition 10.5 **Binary Symmetric Channel**

The binary symmetric channel (BSC) is a communication channel that transmits binary symbols (0 or 1) with a certain probability of error.
$\Longrightarrow$ The transition probability matrix for a BSC is given by:

$$\boxed{P(y|x) = \begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix}}$$

where p is the probability of error in the channel.



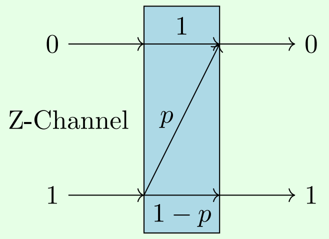
Z-Channel

Definition 10.6 **Z-Channel**

The Z-channel is an binary asymmetric communication channel that transmits binary symbols (0 or 1) with a certain probability of error, but only for the symbol 1.
$\Longrightarrow$ The transition probability matrix for a Z-channel is given by:

$$\boxed{P(y|x) = \begin{bmatrix} 1 & p \\ 0 & 1-p \end{bmatrix}}$$

where p is the probability of error in the channel.



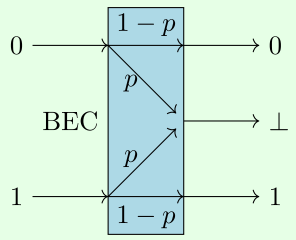
Binary Erasure Channel (BEC)

Definition 10.7 **Binary Erasure Channel**

The binary erasure channel (BEC) is a communication channel that transmits binary symbols (0 or 1) with a certain probability of erasure.
$\Longrightarrow$ The transition probability matrix for a BEC is given by:

$$\boxed{P(y|x) = \begin{bmatrix} 1-p & 0 & p \\ 0 & 1-p & p \end{bmatrix}}$$

where p is the probability of erasure in the channel.



10.4 Error Detection and Correction

To combat noise, we need to detect when errors have occurred and correct errors without retransmitting the data.

10.4.1 Repetition Codes

Definition 10.8 **Repetition Codes**

Repetition codes are a simple form of error-correcting codes that repeat the original message multiple times to detect and correct errors by using majority voting.

$\Longrightarrow$ The repetition code for a binary message x is defined as:

$$\boxed{C(x) = x^n = \underbrace{xx\ldots x}_{n \text{ times}}} \quad \text{where } x \in \{0,1\} \text{ and } n \text{ is the number of repetitions.}$$

The repetition code can correct up to $\left\lfloor \frac{n-1}{2} \right\rfloor$ errors in the received message.

Here are the steps to encode and decode a message using repetition codes:

1. When a sender transmits a bit, the bit is repeated n times (Note that n is odd).
2. During transmission, the bits may be flipped due to noise.
3. The receiver receives the repeated bits and counts the number of 0s and 1s.
4. The receiver uses majority voting to determine the most likely original bit.

Since the noise only affects each bit independently, the probability of error in the received message is as follows:

Theorem 10.3 **Probability of Error in Repetition Codes**

The probability of error in a repetition code with n repetitions is given by:

$$\boxed{P_e = \sum_{k=\left\lfloor \frac{n}{2} \right\rfloor + 1}^{n} \binom{n}{k} p^k (1-p)^{n-k}} \quad \text{where } p \text{ is the probability of error in each bit.}$$

As the number of repetitions n increases, the probability of error decreases exponentially, making repetition codes more reliable.

Example 10.4 **Repetition Codes**

Consider a binary message $x = 1$ that is transmitted using repetition codes with $n = 5$:

$$C(x) = 11111$$

If the received message is $y = 10111$, we can use majority voting to determine the original bit:

- Count the number of 0s and 1s: $y = 10111$ has 4 ones and 1 zero.
- Since the majority is 1, we decode the received message as $x' = 1$.

Assume the probability of error in each bit is $p = 0.1$. The probability of error in the received message is:

$$\begin{aligned} P_e = \sum_{k=3}^{5} \binom{5}{k} p^k (1-p)^{5-k} = \binom{5}{3} p^3 (1-p)^2 + \binom{5}{4} p^4 (1-p)^1 + \binom{5}{5} p^5 (1-p)^0 &= 10(0.1)^3(0.9)^2 + 5(0.1)^4(0.9)^1 + (0.1)^5 \\ &= 0.0081 + 0.00045 + 0.00001 = 0.00856 \end{aligned}$$

$\therefore$ The probability of error in the received message is approximately 0.00856, which is quite low compared to the original probability of error $p = 0.1$.

Even though the repetition codes are simple and effective, they are extremely inefficient in terms of bandwidth and storage, as they require n times the amount of data to be transmitted.

10.4.2 Hamming Codes

Definition 10.9 **Hamming Codes**

Hamming codes are a class of error-correcting codes created by Richard Hamming that can detect and correct single-bit errors in binary messages.
$\Longrightarrow$ Hamming codes are based on the concept of parity bits, which are added to the original message to create a codeword.

Hamming Distances

Definition 10.10 **Hamming Distance**

The Hamming distance between two binary strings of equal length is defined as the number of positions at which the corresponding bits are different.
$\Longrightarrow$ The Hamming distance between two binary strings x and y is denoted as:

$$\boxed{d_H(x, y) = \sum_{i=1}^{n} |x_i - y_i|} \quad \text{where } n \text{ is the length of the strings.}$$

The Hamming distance is used to measure the error-correcting capability of a code.

The minimum Hamming distance d_{min} of a code is the smallest Hamming distance between any two codewords in the code.
$\Longrightarrow$ The error detection capability t of a code is given by:

$$\boxed{t = d_{min} - 1} \quad \text{where } t \text{ is the number of errors that can be detected.}$$

$\Longrightarrow$ The error-correcting capability t of a code is given by:

$$\boxed{t = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor} \quad \text{where } t \text{ is the number of errors that can be corrected.}$$

(7, 4) Hamming Code

Definition 10.11 **(7, 4) Hamming Code**

The (7, 4) Hamming code is a linear error-correcting code that encodes 4 bits of data into a 7-bit codeword by adding 3 parity bits.

The syndrome-basedd decoding approach used by the (7, 4) Hamming code is as follows:

1. Calculate the parity bits for the received codeword.
2. Compare the calculated parity bits with the received parity bits.
3. If there is a mismatch, determine the error position using the syndrome vector.
4. Flip the bit at the error position to correct the error.
5. Decode the corrected codeword to retrieve the original message.

Chapter 11

Quantum Error Correction

11.1 Introduction to Quantum Error Correction

Quantum error correction is a set of techniques used to protect quantum information from errors due to decoherence and other noise, which is essential for building reliable quantum computers and quantum communication systems.

Quantum error correction codes are designed to detect and correct errors in quantum states without measuring the states directly, which would collapse the quantum state.

The main principles of quantum error correction are:

- Redundancy: Quantum information is encoded in a larger Hilbert space using multiple qubits.
- Error Detection: Errors are detected without measuring the quantum state directly.
- Error Correction: The original quantum state is recovered by applying appropriate correction operations.
- Fault Tolerance: The error correction process is designed to be fault-tolerant, meaning that it can correct errors even if some qubits are affected by noise.

To understand quantum error correction, we need to know the concept about quantum error correction codes.

Definition 11.1 **Quantum Error Correction Codes**

Quantum error correction codes are a set of techniques used to protect quantum information from errors due to decoherence and other noise.
$\Longrightarrow$ Quantum error correction codes are designed to detect and correct errors in quantum states without measuring the states directly, which would collapse the quantum state.

There are 2 examples of quantum error correction codes:

1. **Quantum Repetition Code**.
2. **Shor 9-Qubit Code**.

11.2 Quantum Repetition Code

Definition 11.2 **Quantum Repetition Code**

The quantum repetition code is a simple quantum error correction code that encodes a single qubit into multiple qubits by repeating the state.
$\Longrightarrow$ The quantum repetition code can correct errors in the encoded qubit by using majority voting.

11.3 Shor 9-Qubit Code

Definition 11.3 **Shor 9-Qubit Code**

The Shor 9-qubit code is a quantum error correction code that encodes a single qubit into 9 qubits and can correct arbitrary single-qubit errors.
$\implies$ The Shor 9-qubit code uses a combination of classical error correction techniques and quantum error correction techniques.

11.4 Encoding Procedure for the Shor 9-Qubit Code

The Shor code encodes a single logical qubit into 9 physical qubits, protecting against any single-qubit error (bit-flip or phase-flip).

11.4.1 Step 1: Express in the ±-basis

Given an arbitrary qubit
$$|\psi\rangle = a|0\rangle + b|1\rangle,$$
rewrite it in the $\{|+\rangle, |-\rangle\}$ basis, where
$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}, \quad |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}.$$
Thus,
$$|\psi\rangle = \alpha|+\rangle + \beta|-\rangle, \quad \text{where} \quad \alpha = \frac{a+b}{\sqrt{2}}, \quad \beta = \frac{a-b}{\sqrt{2}}.$$

11.4.2 Step 2: Phase-Flip Protection (Repetition in ±-basis)

Encode each $|\pm\rangle$ as a 3-qubit repetition code:
$$|+\rangle \mapsto |+++\rangle, \qquad |-\rangle \mapsto |---\rangle.$$
After this step,
$$|\psi\rangle \mapsto \alpha|+++\rangle + \beta|---\rangle.$$
This 3-qubit block can correct a single phase-flip (Z) error.

11.4.3 Step 3: Bit-Flip Protection (Repetition in Computational Basis)

Now, encode each $|\pm\rangle$ in the computational basis using the 3-qubit bit-flip code:
$$|+\rangle \mapsto \frac{|000\rangle + |111\rangle}{\sqrt{2}}, \qquad |-\rangle \mapsto \frac{|000\rangle - |111\rangle}{\sqrt{2}}.$$
The final encoded state is
$$|\psi_L\rangle = \alpha\left(\frac{|000\rangle + |111\rangle}{\sqrt{2}}\right)^{\otimes 3} + \beta\left(\frac{|000\rangle - |111\rangle}{\sqrt{2}}\right)^{\otimes 3}.$$
Now, the logical qubit is protected against both bit-flip and phase-flip errors.

11.5 Error Detection and Correction Procedure

The Shor code corrects both bit-flip (X) and phase-flip (Z) errors by using the structure of the encoding.

11.5.1 A. Bit-Flip (X) Error Correction

Each group of 3 qubits (blocks $\{q_1, q_2, q_3\}$, $\{q_4, q_5, q_6\}$, $\{q_7, q_8, q_9\}$) forms a classical repetition code. To detect and correct a single X error in a block:

1. Measure the parities $q_1 \oplus q_2$ and $q_2 \oplus q_3$ using ancilla qubits and CNOT gates.
2. The measurement outcomes identify which qubit (if any) was flipped.
3. Apply an X gate to correct the error if needed.

Parity check circuit for one block:
Correction rule:

- $m_{12} = m_{23} = 1$: flip q_2
- $m_{12} = 1$, $m_{23} = 0$: flip q_1
- $m_{12} = 0$, $m_{23} = 1$: flip q_3
- $m_{12} = m_{23} = 0$: no correction needed

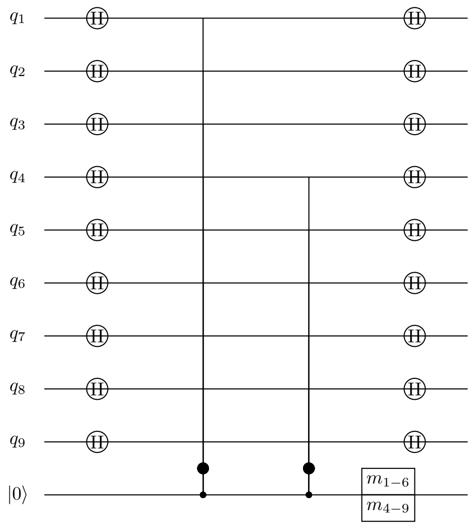
Repeat this procedure for all three blocks. In total, 6 ancilla qubits and 12 CNOTs are needed for all blocks.

11.5.2 B. Phase-Flip (Z) Error Correction

After correcting bit-flip errors, the three blocks together form a repetition code in the $\pm$-basis, which protects against phase-flip errors.

1. Apply Hadamard gates to all 9 qubits (converts Z errors to X errors).
2. Measure the parity between blocks: e.g., measure parity of blocks $\{1, 2, 3\}$ and $\{4, 5, 6\}$, and between $\{4, 5, 6\}$ and $\{7, 8, 9\}$, using ancilla qubits and CNOTs.
3. The measurement outcomes identify which block (if any) has a phase-flip error.
4. Apply a Z gate to correct the error in the identified block.
5. Apply Hadamard gates again to return to the computational basis.

Syndrome extraction circuit:



Summary:

- The Shor code corrects any single-qubit error by first correcting bit-flip errors within each block, then correcting phase-flip errors across blocks.
- Error syndromes are extracted using ancilla qubits and CNOTs, and corrections are applied accordingly.