Lecture Notes on Quantum Computing

Francesco Arzani, Sacha Cerf, Ulysse Chabaud, Sharon David, Jack Davis

Winter semester 2025

Contents

Preface

These notes were compiled during the Quantum Computing course taught by Francesco Arzani and Jack Davis for the Quantum Engineering master at ENS-PSL during the 2025 winter semester. They were based on many different sources and we will do our best to mention all of them at the end of the respective chapters.

They are meant as an aid to the course but there is no guarantee of completeness.

Use of AI tools. The 2026 revision of these notes has been reviewed, and in part written, with the help of large language models (LLMs), namely Claude (Anthropic) and ChatGPT (OpenAI). They were used to check the text for errors and inconsistencies, to draft some passages, examples and exercises, and to format the bibliography. All such contributions have been checked and edited by the authors, who remain responsible for the content. Any remaining errors are ours; please report them to us.

Part I

Quantum algorithms in the circuit model

Chapter 1

The circuit model

In the most general sense, the phrase *quantum computing* can be taken to mean using any quantum resources to solve computational tasks. We will define computational tasks more precisely later, but assuming for now that it just means anything that can be solved on an ordinary computer, there are many ways to encode and process information using quantum systems. There is, however, one model that has become prominent in the literature that most people's minds would immediately go to when hearing the words *quantum computer*, and that is the *circuit model*. This is a generalization of the representation of Boolean functions as sequences of logic gates, such as AND ($\wedge$) and NOT ($\neg$). The circuit model turns out to be a remarkable tool for reasoning about not just quantum algorithms but quantum experiments in general. It also gives an intuitive meaning to *computational complexity*, that will be the basis for discussions on quantum computational advantage in later chapters. It is therefore the best place to start our journey.

1.1 Classical computation

We begin by reviewing classical computation. In classical information theory, the elementary carriers of information are **bits**, which are variables that can only assume two values: 0 and 1. A computation process is therefore a map that acts on input bits and returns output bits. The **circuit model** is a graphical representation of computational operations on bits.

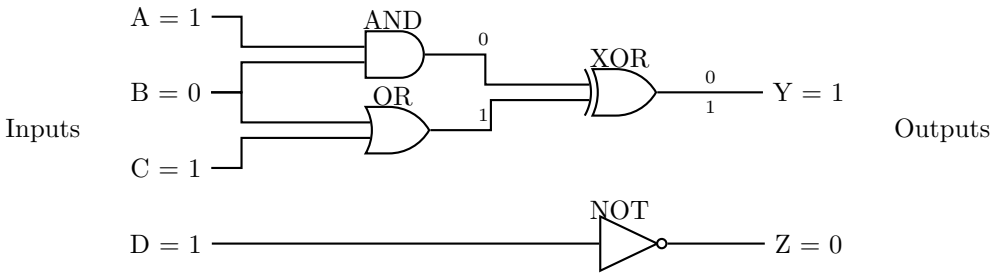


Figure 1.1: A classical logic circuit acting on four input bits.

The wires of the logic circuit represent the bits, and the blocks are the operations performed on the bits which, when combined, constitute the computation. These operations are represented by blocks that cause the wires to interact with each other.

1.1.1 Elementary Logic Gates

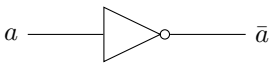
A computation process is therefore a map on a binary string of length M that returns another binary string of length m as a result:

$$f : \{0,1\}^M \to \{0,1\}^m$$

In general, $M \neq m$ (it could be that $M > m$ or $M < m$). The notation $\{0,1\}^k$ indicates a string of length k composed of binary digits (0 or 1).

It can be shown that all logical functions of this type can be decomposed into elementary operations, or **logic gates**. These are typically operations on $M = 1$ or $M = 2$ input bits that produce $m = 1$ output bit.

- **Case: 1-bit gates ($M = 1, m = 1$)**
  There are exactly four maps from one bit to one bit: the identity (leaving the bit unchanged), the inversion of the bit ($1 \to 0, 0 \to 1$), and the two **constant** maps $a \to 0$ and $a \to 1$. The two constant maps discard the value of the input entirely, so they are not invertible; we will see in Section 1.1.3 that this erasure carries an unavoidable thermodynamic cost. Only the identity and the inversion preserve the information carried by the bit, and the inversion is therefore the only non-trivial invertible operation available on a single bit. It is called a **NOT** gate (or negation) and is represented in the circuit model by the following symbol:

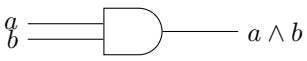
a	$\bar{a}$
0	1
1	0

  This operation is often indicated algebraically as:

  $$\boxed{\bar{a} = 1 - a}$$

- **Case: 2-bit gates ($M = 2, m = 1$)**
  These are logic gates that take two bits as input and produce one bit as output. There are many such operations that can be performed.

  The AND Gate The AND gate outputs 1 only if both of its inputs are 1.

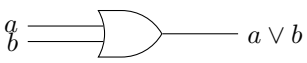
  

a	**b**	$a \wedge b$
0	0	0
0	1	0
1	0	0
1	1	1

  This is analogous to the multiplication of bits!

  $$\boxed{a \wedge b = a \cdot b}$$

  The OR Gate The OR gate outputs 1 if at least one of its inputs is 1.

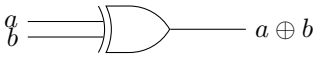
  

a	**b**	$a \vee b$
0	0	0
0	1	1
1	0	1
1	1	1

  The algebraic expression for the OR operation is:

  $$\boxed{a \vee b = a + b - a \cdot b}$$

The XOR Gate The XOR (Exclusive OR) gate outputs 1 only if the inputs are different from each other.

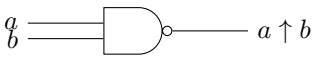


a	**b**	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

The algebraic expression for XOR is addition modulo 2:

$$\boxed{a \oplus b = (a + b) \pmod 2}$$

The NAND Gate The NAND gate is the negation of the AND gate. It outputs 0 only if both inputs are 1.

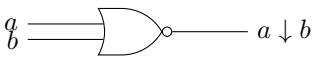


a	**b**	$a \uparrow b$
0	0	1
0	1	1
1	0	1
1	1	0

The NAND operation is algebraically:

$$\boxed{a \uparrow b = \overline{a \cdot b}}$$

The NOR Gate The NOR gate is the negation of the OR gate. It outputs 1 only if both inputs are 0.

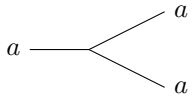


a	**b**	$a \downarrow b$
0	0	1
0	1	0
1	0	0
1	1	0

The algebraic expression for the NOR operation is:

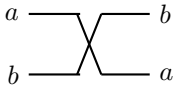
$$\boxed{a \downarrow b = \overline{a \vee b} = \overline{a + b - ab} = 1 - (a + b - ab) = 1 - a - b + ab}$$

- **Case:** $M = 1, m = 2$
  Several maps of this kind exist (for instance $a \to (a, 0)$ or $a \to (a, \bar{a})$), but one is of particular interest: the **COPY** or **FANOUT** operation.



  The map is $a \to (a, a)$. In essence, it makes a copy of the input bit and returns a string of two bits identical to the input.

- **Case:** $M = 2, m = 2$
  Here too, there is one fundamental elementary operation that is often useful: the **SWAP** operation, which exchanges the position of two bits.



  The map is $(a, b) \to (b, a)$.

1.1.2 Universal Gate Sets

With these "elementary" operations, it is possible to realize any other type of operation on bit strings. But are all the elementary operations we have seen really indispensable? In other words, is it impossible to derive some of them from the others? The answer is no; some of them are linked by relations known as **De Morgan's identities**.

Theorem 1.1 (De Morgan's identities and gate equivalences)**.**

$$\begin{aligned} \textit{NAND:} \quad & a \uparrow b = \overline{a \wedge b} = \bar{a} \vee \bar{b} \\ \textit{NOR:} \quad & a \downarrow b = \overline{a \vee b} = \bar{a} \wedge \bar{b} \\ \textit{XOR:} \quad & a \oplus b = (a \vee b) \wedge (\bar{a} \vee \bar{b}) = (a \vee b) \wedge \overline{(a \wedge b)} \end{aligned}$$

The last identity shows how an XOR gate can be constructed using only AND, OR, and NOT gates. This implies that we don't need all the gates to build any possible circuit. A set of gates that is sufficient to create any other logical operation is called a **functionally complete** set.

Theorem 1.2 (Functional Completeness)**.** *Any logic map $f : \{0,1\}^M \to \{0,1\}^m$ can be realized with a sequence of AND, OR, NOT, and COPY gates.*

This set is however not minimal: it can be shown that the NAND gate along with FANOUT/COPY are functionally complete. The same is true for the NOR+FANOUT/COPY set. This can be proven by writing each gate {AND,OR,NOT,COPY} in terms of {NAND, COPY}. Then circuits composed of the former set of gates can be compiled using only the latter.

Theorem 1.3. *The set* ***{NAND, COPY}*** *forms a minimal and universal (functionally complete) set of logic gates.*

1.1.3 Computational Reversibility

In the classical circuits above we assumed we can increase or decrease the number of bits at will. This process is not always reversible. We can define a computational process as **reversible** if, from the output, it is possible to recover the input. Consider a general process described by a function $f : \{0,1\}^M \to \{0,1\}^m$. If the number of output bits is less than the number of input bits ($m < M$), it is impossible, in general, to recover the input, as multiple different inputs could map to the same output. Such a process is fundamentally **irreversible**. In cases where $m \geq M$, the reversibility depends on the specific form of the function f. Irreversibility is fundamentally linked to a **loss of information** during the computation, effectively an erasure of bits. Physically, the loss of information corresponds to a loss of energy.

There is a deep connection between computational irreversibility and the second law of thermodynamics. When a bit is erased (for example, by an AND gate where inputs 1 and 0 result in 0), the system loses information about its prior state. This corresponds to a decrease in the system's entropy. To avoid violating the second law of thermodynamics, this decrease must be compensated by an equal or greater increase in the entropy of the surroundings, which occurs through the dissipation of energy as heat.

Landauer's Principle

Every time a single bit of information is erased, a minimum quantity of energy E is dissipated into the environment as heat:

$$E \geq k_B T \ln(2)$$

where k_B is the Boltzmann constant and T is the absolute temperature of the environment.

Constructing Reversible Circuits

It is possible to transform any irreversible computation into a reversible one by working with a larger number of bits, specifically $M + m$ bits. This construction will come in handy when working with

quantum systems, governed by unitary, reversible, evolution.
We can define a new function $\tilde{f}$ that operates on this larger space and is fully reversible:

$$\tilde{f} : \{0,1\}^{M+m} \to \{0,1\}^{M+m}$$

where, in particular, the mapping is defined as:

$$\tilde{f}(x, y) = (x, y \oplus f(x))$$

In this construction:

- x is the M-bit input string of the original function f.
- y is an m-bit auxiliary input string, which is the same size as the output of f.
- The $\oplus$ symbol represents a bitwise XOR operation (addition modulo 2).

Notice that in the output, the input string x is preserved without modification or erasure, and the result of the original computation $f(x)$ is stored in the auxiliary register y. To recover the original computation, we can simply set the auxiliary input to a string of zeros ($y = 0^m$):

$$\tilde{f}(x, y = 0) = (x, 0 \oplus f(x)) = (x, f(x))$$

That $\tilde{f}$ is reversible can be checked directly: applying it twice gives $\tilde{f}(\tilde{f}(x, y)) = (x, y \oplus f(x) \oplus f(x)) = (x, y)$, so $\tilde{f}$ is its own inverse.

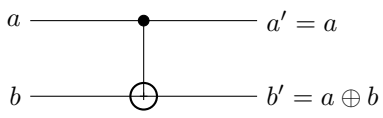
This pattern is not a classical curiosity: it is exactly how a function is handed to a quantum algorithm. Since quantum evolution is unitary, and therefore reversible, a function f can only enter a quantum circuit through the unitary

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

which is the same construction written in Dirac notation. Every algorithm we will meet that queries a function — Deutsch's problem and the Deutsch–Jozsa algorithm, Simon's problem, Grover's search, the period finding at the heart of Shor's algorithm — accesses it through an oracle of precisely this form; see Chapter 2. The reason the construction is worth the extra m bits is that it is the only way to make an arbitrary f compatible with reversible, and hence with unitary, dynamics.

Examples of Reversible Gates

- A clear example of a reversible gate we have already seen is the **SWAP** gate.
- The **CNOT**, or "Controlled-NOT", gate is a fundamental 2-bit reversible gate. It maps two input bits (a, b) to two output bits (a', b').



Input		Output	
a	**b**	**a'**	**b'**
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

  The CNOT gate is called "controlled-NOT" because the first bit (a) acts as a **control** bit, and the second bit (b) is the **target** bit. The value of the control bit determines whether a NOT operation is applied to the target bit.

  - If $a = 0$, nothing happens to b.
  - If $a = 1$, the target bit b is flipped (a NOT operation is performed).

  The reversibility is evident from the fact that the input state can always be uniquely determined from the output state.

- Using the CNOT gate, it is possible to realize other important reversible operations:

  1. **Identity:** The CNOT gate is its own inverse. Applying it twice returns the original state.

  $$(\text{CNOT})^2 = I$$

  2. **Reversible Copy (FANOUT):** By setting the target bit b to 0, the CNOT gate acts as a copying mechanism, mapping $(a, 0)$ to (a, a). This is a reversible way to create a copy of a bit.

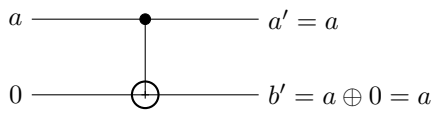


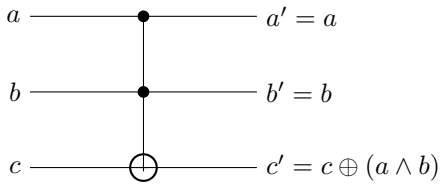
Figure 1.2: A reversible COPY operation constructed from a CNOT gate.

Reversible gates are those that implement a permutation on the set of possible input strings. That is, for a map $f : \{0,1\}^M \to \{0,1\}^M$, every unique input string maps to a unique output string. Despite the existence of 2-bit reversible gates, it can be proven that it is **not possible** to construct a universal set for reversible computation using only 2-bit gates. Universal reversible computation requires at least one gate that acts on 3 or more bits.

Remark 1.4. The COPY gate deserves a comment. No information is destroyed by COPY: the map is injective, and the input can be read off from either of the two outputs. Nevertheless it is not a reversible *gate* in the sense of this section, because it is not a permutation of a fixed set of bit strings and therefore cannot be inverted in general. It sends the two one-bit strings to only two of the four two-bit strings, so the strings $(0, 1)$ and $(1, 0)$ have no preimage and the inverse is simply undefined on them. This is the reason to introduce the reversible version above, which promotes it to a genuine two-bit gate by fixing the second input to 0.

Universal 3-Bit Reversible Gates

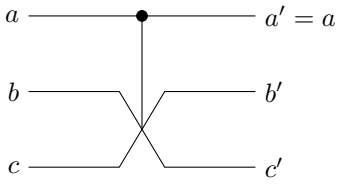
The Toffoli Gate The Toffoli gate is a 3-bit reversible logic operation, mapping $\{0,1\}^3 \to \{0,1\}^3$. It is also known as a **CCNOT** gate, as it is a CNOT gate with two control bits.



Input			Output		
a	**b**	**c**	**a'**	**b'**	**c'**
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

The Toffoli gate leaves the two control bits (a, b) unchanged. The target bit (c) is flipped if and only if both control bits are 1. Otherwise, it acts as the identity.
It can be shown that the Toffoli gate is **universal** for reversible computation. This means that any reversible operation can be realized using only Toffoli gates (and auxiliary bits prepared in fixed states).

The Fredkin Gate The Fredkin gate is another universal 3-bit reversible operation, also known as the **Controlled-SWAP** (CSWAP) gate.



Input			Output		
a	**b**	**c**	**a'**	**b'**	**c'**
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	1	0
1	1	0	1	0	1
1	1	1	1	1	1

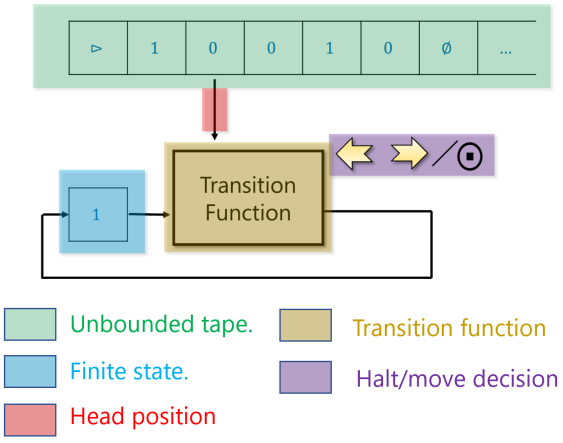
The Fredkin gate leaves the control bit a unchanged. It swaps the target bits b and c if and only if the control bit $a = 1$. The Fredkin gate is also **universal**.

1.1.4 Computational Complexity

The resources required during a computation process are primarily of two types: **time** ($\sim$ number of gates) and **space** ($\sim$ required memory, i.e. number of bits). The complexity of a calculation is therefore measured in terms of the resources required to execute it. For example, the time complexity measures how the execution time scales *as a function of the number of input bits*, n: assuming a constant execution time per gate, the time t required to add two n-bit numbers scales linearly with n, so $t \propto n$, whereas for schoolbook multiplication $t \sim n^2$.

The standard way to introduce the formalism necessary to rigorously analyze computational complexity is through Turing machines. A **Turing machine** (TM) is a simple mathematical model of computation introduced by Alan Turing in 1936. Imagine an infinitely long tape divided into cells, each containing a symbol from an alphabet Γ (usually 0, 1, or blank) and a read/write head with a set of internal states Q. At each time step the head scans one cell and can:

- Read the current symbol
- Write a new symbol
- Move left or right one cell
- Change its internal state



The machine evolves according to a *transition function* (the "program") $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$. The **Church-Turing thesis** asserts that Turing machines can compute anything that *any* (classical) computer can compute. Turing machines are used to define resources and to classify problems by their requirements:

- **Time complexity**: $T(n)$ = maximum number of steps a TM takes on inputs of length n
- **Space complexity**: $S(n)$ = maximum number of tape cells used

This gives us fundamental complexity classes:

- P: problems whose solution requires a number of steps scaling as a polynomial in the size of the input, $O(n^k)$
- NP: problems whose solution can be *verified* with a polynomial number of steps
- PSPACE: problems that can be solved using polynomial space (memory)

The famous P vs NP question asks whether every efficiently verifiable problem is also efficiently solvable. Clearly, $\mathsf{P} \subseteq \mathsf{NP}$. However, it is not yet known if there are problems in NP that are not in P. It is conjectured that the inclusion is strict ($\mathsf{P} \subsetneq \mathsf{NP}$), and it is so widely believed that it is taken as an assumption in many results in complexity theory.

The major distinction in terms of computational cost is between polynomial and super-polynomial problems.

- **Polynomial problems:** The required resources are upper-bounded by a polynomial function in n: poly(n) (for example sum and multiplication).
- **Super-polynomial problems:** Require resources greater than the previous case: $>$ poly(n) (for example all known algorithms to find the prime factors of an integer number require exponential time).

Space and time are not completely independent: a computation can only touch one extra memory cell per step, so the space it uses is at most proportional to the time it takes, $s \lesssim t$. The converse fails, because memory can be reused: a computation may run for a long time in very little space.

Complexity theory is also concerned with proving relationships between complexity classes. For example we know that:

- $\mathsf{P} \subseteq \mathsf{PSPACE}$ because a polynomial-time algorithm can only access a polynomial amount of memory (think of using one memory cell for each time step).
- $\mathsf{NP} \subseteq \mathsf{PSPACE}$ because I can solve an NP problem by trying all different attempts while reusing the same memory space.

Therefore, the hierarchy of major complexity classes is:

$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP}$$

Here EXP is the class of problems solvable in exponential time. The separation $\mathsf{P} \subsetneq \mathsf{EXP}$ is known (it follows from the time hierarchy theorem), so at least one of these inclusions must be strict; it is conjectured that all of them are, but not a single one has been proven.

Reductions, hardness and completeness

Since we cannot compute the absolute difficulty of a problem, complexity theory compares problems to one another. We say that a problem A **reduces** to a problem B if any instance of A can be transformed, in polynomial time, into an instance of B whose solution yields the solution of the original instance; we write $A \leq_p B$. A reduction is a statement about *relative* difficulty: if $A \leq_p B$ then B is at least as hard as A, because an efficient algorithm for B immediately gives one for A: run the translation, then the algorithm. Equivalently, and this is how reductions are most often used, if A is known to be hard then so is B.

A problem B is said to be **NP-hard** if every problem in NP reduces to it. Note that B itself is not required to lie in NP: NP-hardness is a lower bound on difficulty and says nothing about membership, so an NP-hard problem may well be much harder than anything in NP. If in addition $B \in \mathsf{NP}$, then B

is **NP-complete**: it belongs to NP and is as hard as anything in it, making it a hardest representative of the class.

NP-complete problems are the pivots of the P vs NP question. Because every problem in NP reduces to any one of them, a polynomial-time algorithm for a single NP-complete problem would place all of NP inside P and settle the question. Conversely, proving that one of them requires super-polynomial time would prove $\mathsf{P} \neq \mathsf{NP}$.

A word of warning for what follows: it is tempting to hope that a quantum computer might solve NP-complete problems efficiently, but there is no evidence for this, and it is not what the known quantum algorithms do. Factoring, the flagship application of Chapter 2, is in NP but is *not* believed to be NP-complete. Grover's algorithm does apply to NP problems but delivers only a quadratic speed-up, which might be very useful practically, but does not change a super-polynomial running time into a polynomial one.

Connection to Logic Circuits

There's a deep relationship between Turing machines and Boolean circuits. While the TM for a given task is independent of the size of the input, a circuit only handles inputs of one fixed length, so a task corresponds to a whole *family* of circuits. A TM running in time $T(n)$ can be simulated by a circuit family $\{C_n\}$ of size $O(T(n)^2)$. Each circuit C_n handles inputs of length n, with gates computing AND, OR, NOT operations. Conversely, uniform circuit families correspond to TM computations. Uniform loosely means that a *single* TM, given the string 1^n as input, outputs a description of C_n in time scaling like a polynomial in n. Without this requirement a circuit family could smuggle in uncomputable information through the choice of C_n.

Remark 1.5. It is worth spelling out what this correspondence says about circuit size versus evaluation time on a Turing machine. The size of a circuit, measured by its number of gates, is the natural circuit-model analogue of the running time of a Turing machine, and the two are equivalent up to an overhead which scales as a polynomial in the size of the input. In one direction, a TM running in time $T(n)$ compiles into a circuit C_n of size $O(T(n)^2)$. In the other, a uniform family of circuits of size $S(n)$ can be evaluated by a Turing machine in time polynomial in $S(n)$ and n: the machine first writes down the description of C_n, which uniformity guarantees it can do in $\mathrm{poly}(n)$ steps, and then simulates the circuit gate by gate. Consequently

$$\mathsf{P} = \{\text{problems with a uniform family of circuits of size } \mathrm{poly}(n)\}\,,$$

and we may use "polynomial time" and "polynomial number of gates" interchangeably. This is why it is legitimate to define quantum complexity classes directly in terms of circuits, as we will do for BQP, without explicitly introducing *quantum* Turing machines.

1.1.5 Probabilistic Computation

So far we have talked about deterministic computation, in which a given input always corresponds to the same output. We can also introduce probabilistic algorithms: the same input can correspond to a different output on each run, according to some random variables. These algorithms induce a new complexity class:

- BPP **(Bounded-error Probabilistic Polynomial-time)**: the class of problems that admit a probabilistic algorithm running in polynomial time such that, for every input, the probability of obtaining a correct result is $p_{\text{success}} \geq 1/2 + \delta$ for some constant $\delta > 0$ independent of the input.

It is known that $\mathsf{P} \subseteq \mathsf{BPP}$, while the relationship between BPP and NP is not well understood. The class BPP immediately finds a counterpart in the quantum case:

- BQP **(Bounded-error Quantum Polynomial-time)**: problems solvable with quantum algorithms such that the probability of ultimately obtaining a correct answer is $p_{\text{success}} \geq 1/2 + \delta$, again for a constant $\delta > 0$.

It is conjectured that BPP $\neq$ BQP because, for example, factorization admits a quantum algorithm in BQP, but no classical BPP algorithm for it is known.

Given a probabilistic algorithm, we can apply Chernoff's bound to "amplify" the correct answer by iterating the algorithm. Taking a majority vote of the results of different iterations the error probability decreases exponentially. This implies that if we have a polynomial time probabilistic algorithm, for any *fixed* error probability we are admitting, no matter how small, the total time of the computation will still be polynomial (in the size of the input). The degree of the polynomial, however, will grow as we make the tolerated failure probability smaller.

1.2 Qubits

Summary

This section introduces the fundamental unit of quantum information: the qubit.

- **Qubit definition**: Quantum bits can exist in superpositions of $|0\rangle$ and $|1\rangle$ states
- **State representation**: Qubit states are described by complex amplitudes satisfying normalization conditions
- **Measurement**: Quantum measurement follows the Born rule, giving probabilistic outcomes
- **Bloch sphere**: Provides a geometric representation of single-qubit states
- **Multi-qubit systems**: Exhibit exponential state space growth with increasing qubit count
- **Quantum advantage**: The exponential growth of state space suggests potential computational advantages over classical systems

These foundational concepts provide the basis for understanding more advanced quantum computing models and algorithms discussed in subsequent chapters.

By analogy with classical logic, the most basic information carriers in the quantum case will be two-level systems. However, we assume that the laws of quantum physics apply, allowing phenomena like *superposition* and *entanglement*. When discussing information processing in a quantum world, the most basic unit is therefore a quantum bit, usually called qubit, a two-level quantum system with a ground state $|0\rangle$ and an excited state $|1\rangle$. Unlike a classical bit, which only has two possible states, a quantum bit has infinitely many states: all superpositions of $|0\rangle$ and $|1\rangle$,

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix},$$

where α and β are complex numbers satisfying $|\alpha|^2 + |\beta|^2 = 1$. A measurement of this qubit state in the so-called *computational basis* $|0\rangle, |1\rangle$ gives the result 0 with probability $|\alpha|^2$ and the result 1 with probability $|\beta|^2$.

1.2.1 Bloch Sphere Representation

A useful tool for visualizing a qubit state is the Bloch sphere. A state of the qubit is represented as a point on the surface of the sphere, which has radius 1. The two states of a classical bit correspond to the north and south poles on the sphere. The two states on opposite ends of the x axis are often denoted

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \quad \text{and} \quad |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle),$$

the so-called *diagonal* basis.

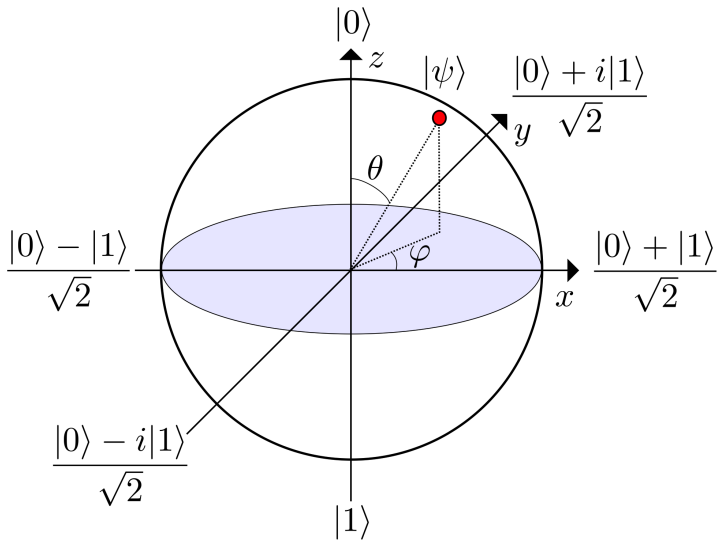


Figure 1.3: The Bloch-sphere representation of a qubit state. The north pole is the ground state $|0\rangle$ and the south pole is the excited state $|1\rangle$. To convert an arbitrary superposition of $|0\rangle$ and $|1\rangle$ to a point on the sphere, the parametrization $|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle$ is used.

1.2.2 Mixed States and Density Matrices

Throughout this chapter we describe the state of a qubit by a vector $|\psi\rangle$, a so-called *pure* state. This is not the most general description available. Whenever the preparation of the qubit is itself only known probabilistically, or whenever the qubit is part of a larger entangled system and we choose to look at it alone, the state has to be described by a **density matrix**

$$\rho = \sum_i p_i \, |\psi_i\rangle\langle\psi_i| \,,$$

a positive semi-definite operator of unit trace. Pure states are the special case $\rho = |\psi\rangle\langle\psi|$, singled out by $\rho^2 = \rho$, or equivalently $\mathrm{Tr}\,\rho^2 = 1$. The Born rule and expectation values are then written $p(m) = \mathrm{Tr}(E_m\rho)$ and $\langle A\rangle = \mathrm{Tr}(A\rho)$ where E_m is the element of a positive operator-valued measure (POVM). In the special case of a projective measurement E_m is a projector. For example, for a single-qubit measurement $E_m \in \{|0\rangle\langle 0|, |1\rangle\langle 1|\}$ and, say $P(0) = \mathrm{Tr}(|0\rangle\langle 0|\rho) = \langle 0|\rho|0\rangle$.

Geometrically, this fills in the Bloch sphere of Figure 1.3. Writing

$$\rho = \frac{1}{2}\left(\mathbb{I} + \vec{r}\cdot\vec{\sigma}\right), \qquad |\vec{r}| \le 1,$$

with $\vec{\sigma} = (X, Y, Z)$, the pure states are those with $|\vec{r}| = 1$ and occupy the surface, while the mixed states fill the interior; the centre $\vec{r} = 0$ is the maximally mixed state $\mathbb{I}/2$, which carries no information at all. The set of qubit states is therefore the Bloch *ball*, not just the Bloch sphere.

We will not need this formalism for the rest of the present chapter, where all states are pure and all evolutions unitary, but it becomes unavoidable from Part II onwards, as soon as we discuss noise, and whenever we consider the reduced state of a subsystem. Density matrices, the partial trace, the Pauli basis and a number of worked examples are collected in Appendix A; readers who have not met them before are encouraged to work through that appendix before Part II.

1.2.3 Multi-Qubit Systems and Superposition

If there are N qubits in a system, the total state of that system can be a superposition of 2^N different states: $|000\ldots00\rangle$, $|100\ldots00\rangle$, $|010\ldots00\rangle$, …, $|111\ldots10\rangle$, $|111\ldots11\rangle$ (note the abbreviated notation of the tensor product). N classical bits can be in one of these 2^N states, but not in a superposition of several of them.

To store all the information about a general N-qubit state, one needs to keep track of the 2^N amplitudes in the superposition. Since these are complex numbers, faithfully representing the state on a classical computer requires 2^N complex numbers, i.e. a number of classical bits growing exponentially in N. This provides intuition for why it may be hard for classical computers to simulate some quantum systems and gives a first hint that quantum computers can be more powerful than classical ones.

1.2.4 Physical Implementations

There are many physical implementations of qubits, e.g., superconducting qubits, trapped ions, natural atoms, photons, etc. For now, we assume that we have access to qubits without caring too much about how they are made.

1.3 Quantum circuits

Summary

The quantum circuit model provides the foundational framework for most quantum algorithms and serves as the standard model for quantum computation:

- **Qubit representation**: Quantum information is encoded in qubits using superposition and entanglement
- **Gate-based operations**: Computation proceeds through the application of quantum gates (unitary operations)
- **Measurement readout**: Final results are obtained through quantum measurements
- **Universal gate sets**: Small sets of gates can approximate any quantum computation
- **Circuit equivalence**: All universal gate sets are polynomially equivalent
- **Efficient approximation**: The Solovay-Kitaev theorem ensures efficient approximation of arbitrary unitaries
- **Classical simulation limits**: The Gottesman-Knill theorem identifies classes of quantum circuits that are classically simulable

The quantum circuit model has proven to be remarkably robust and versatile, serving as the basis for most quantum algorithm development and providing deep insights into the nature of quantum computation.

Analogous to the classical case, quantum computation studies the operations that can be performed and

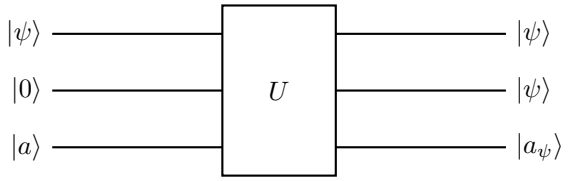
implemented on elementary information carriers: qubits. Generally, we perform operations on an input of N qubits, which are treated as an isolated system; thus, computation is a process of unitary evolution of the N qubits.
A quantum computation consists of 3 steps:

1. Preparation of the input
2. Implementation of the unitary transformation
3. Reading out the output

The third and final process involves the measurement of the state of the N-qubit system; this is a stochastic process. In this sense, every quantum algorithm is a probabilistic algorithm. Note that this step is present in the classical case too, but since the bit values are predetermined and not modified by the read-out itself we did not need to include this step explicitly. The unitary evolution induced by considering qubits isolated during the computation might also be seen as a restriction with respect to the classical case, in that it implies reversibility, which came at the expense of additional resources in the classical case. It might seem strange now but the first works on quantum computers were motivated by the potential limitations incurred when micro-processors would become small enough for quantum effects to become relevant (Feynman, 1982). Another fundamental limitation comes from the non-existence of the COPY gate, which is a central result in the quantum theory of computation:

Theorem 1.6 (The No-Cloning Theorem)**.** *If a system can be in one of two non-orthogonal states, $|\psi\rangle$ or $|\phi\rangle$ (meaning $\langle\psi|\phi\rangle \neq 0$), then no unitary transformation exists that takes the system's state as input and outputs two or more copies of that state.*

Proof. We will perform a proof by contradiction. Suppose a unitary transformation U exists that performs the cloning in the following way:



where $|\psi\rangle$ is the state of our system that we want to clone, $|0\rangle$ is an auxiliary state onto which we will copy the state $|\psi\rangle$, and $|a\rangle$ is an auxiliary system of unspecified dimension whose state could change to $|a_\psi\rangle$, possibly depending on the input. Suppose that U analogously clones the state $|\phi\rangle$, then

$$U|\psi\rangle|0\rangle|a\rangle = |\psi\rangle|\psi\rangle|a_\psi\rangle$$
$$U|\phi\rangle|0\rangle|a\rangle = |\phi\rangle|\phi\rangle|a_\phi\rangle$$

Unitary transformations preserve the scalar product. We take the inner product of the two initial states and the two final states:

$$(\langle\psi|\langle 0|\langle a|)\cdot(|\phi\rangle|0\rangle|a\rangle) = (\langle\psi|\langle\psi|\langle a_\psi|)\cdot(|\phi\rangle|\phi\rangle|a_\phi\rangle)$$

The left-hand side (LHS) evaluates to:

$$\text{LHS} = \langle\psi|\phi\rangle\langle 0|0\rangle\langle a|a\rangle = \langle\psi|\phi\rangle$$

The right-hand side (RHS) evaluates to:

$$\text{RHS} = \langle\psi|\phi\rangle\langle\psi|\phi\rangle\langle a_\psi|a_\phi\rangle = (\langle\psi|\phi\rangle)^2\langle a_\psi|a_\phi\rangle$$

Equating the two sides gives:

$$\langle\psi|\phi\rangle = (\langle\psi|\phi\rangle)^2\langle a_\psi|a_\phi\rangle$$

Since by hypothesis the states are non-orthogonal, $\langle\psi|\phi\rangle \neq 0$, so we can divide by it:

$$1 = \langle\psi|\phi\rangle \langle a_\psi|a_\phi\rangle \implies \langle\psi|\phi\rangle = \frac{1}{\langle a_\psi|a_\phi\rangle}$$

Taking the modulus of both sides gives:

$$|\langle\psi|\phi\rangle| = \frac{1}{|\langle a_\psi|a_\phi\rangle|}$$

However, from the Cauchy-Schwarz inequality, we know that for non-identical normalized states $|\langle\psi|\phi\rangle| < 1$, and for any normalized states $|\langle a_\psi|a_\phi\rangle| \leq 1$. The equation implies $|\langle\psi|\phi\rangle| \geq 1$, which is a contradiction. □

Remark 1.7. Note that if $|\psi\rangle$ and $|\phi\rangle$ were orthogonal, we would not have a contradiction (as $0 = 0$). This means it is possible to "clone" a set of mutually orthogonal states. This is precisely what happens in classical COPY computation, which acts on orthogonal states corresponding to classical bits, $|0\rangle$ and $|1\rangle$.

The diagram used in the proof is an example of a quantum circuit. Elements of general circuits are:

- Qubits, represented by horizontal wires, to encode data.
- State preparation, represented by kets, density matrices or "left-rounded" boxes to initialize the qubits in the input state we need to begin the computation.
- Quantum gates, represented as rectangular boxes or "decorated" vertical wires, with inputs and outputs, acting on the qubits to manipulate the data.
- Measurements on the qubits to read out the final result, typically represented as meters, or as mirror images of state preparations.

For brevity, we assume that it is possible to initialize our quantum computer in some simple reference state, e.g., $|000\ldots00\rangle$, and that we can read out the state of each qubit at the end of a computation in the computational basis $|0\rangle, |1\rangle$.

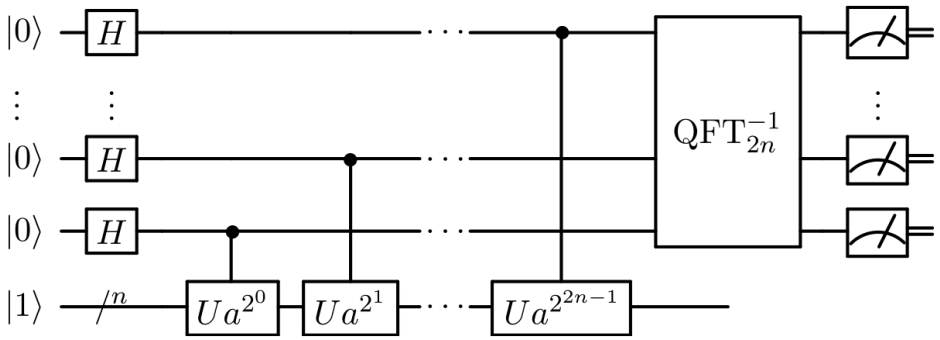


Figure 1.4: Circuit diagram of the quantum subroutine of Shor's algorithm.

1.3.1 Single-Qubit Gates

Operations on a single qubit move its state from $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ to $|\psi'\rangle = \alpha'|0\rangle + \beta'|1\rangle$ while preserving the norm $|\alpha|^2 + |\beta|^2 = 1 = |\alpha'|^2 + |\beta'|^2$. Such operations (gates) can be described by 2×2 unitary matrices. Hereafter we list some of the most common ones. First are the Pauli matrices:

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

Figure 1.5: Circuit representation of the X gate.

They form a group under multiplication, known as the Pauli group.

Definition 1.8 (Pauli Group)**.** *The Pauli group $\mathcal{P}_1$ on a single qubit is the group of 2×2 unitary matrices generated by the Pauli matrices:*

$$\mathcal{P}_1 := \{\pm I, \pm iI, \pm X, \pm iX, \pm Y, \pm iY, \pm Z, \pm iZ\}.$$

The Pauli group $\mathcal{P}_n$ on n qubits is the group of $2^n \times 2^n$ unitary matrices generated by the tensor products of n single-qubit Pauli matrices:

$$\mathcal{P}_n := \{P_1 \otimes \cdots \otimes P_n \mid P_i \in \mathcal{P}_1\}.$$

The X gate is the quantum equivalent of the classical **NOT** gate. The Pauli matrices generate rotations around the corresponding axes on the Bloch sphere when exponentiated, e.g.,

$$R_x(\vartheta) = \exp(-i\vartheta X/2) = \begin{pmatrix} \cos\frac{\vartheta}{2} & -i\sin\frac{\vartheta}{2} \\ -i\sin\frac{\vartheta}{2} & \cos\frac{\vartheta}{2} \end{pmatrix}.$$

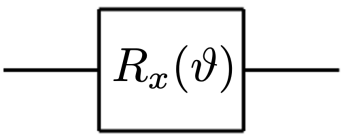


Figure 1.6: Circuit representation of the $R_x(\vartheta)$ gate.

Pauli matrices have many useful properties: notably they are unitary and Hermitian, and together with the identity they form a basis of the real vector space of 2×2 Hermitian matrices. For example, adding the X and Z gates and normalising, one obtains the Hadamard gate:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \frac{X+Z}{\sqrt{2}}.$$

Figure 1.7: Circuit representation of the Hadamard gate.

This gate transforms a qubit state from the computational basis $|0\rangle, |1\rangle$ to the diagonal basis $|+\rangle, |-\rangle$. The Hadamard gate is often used to create superposition states at the beginning of a quantum algorithm. The Z gate applies a phase factor -1 to the $|1\rangle$ part of the qubit state. Two gates that apply other phase factors are often given their own names, the S and T gates:

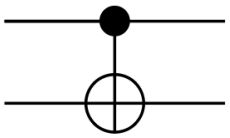
$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}, \quad T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}.$$

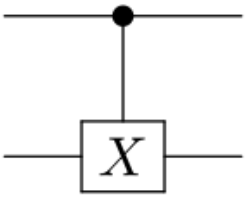
Note that $Z = S^2 = T^4$.

1.3.2 Multi-Qubit Gates

To realize useful quantum algorithms, we also need to be able to make two or more qubits interact through multi-qubit gates. One way to achieve this is to let the state of one qubit (the *control* qubit) control whether a certain single-qubit gate is applied to another qubit (the *target* qubit). One such gate is the

controlled-NOT (CNOT) gate: $$\mathrm{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = I_2 \oplus X.$$





where I_2 denotes the 2×2 identity matrix. Note that the two-qubit Hilbert space is spanned by the basis vectors $|00\rangle, |01\rangle, |10\rangle, |11\rangle$, in that order. This is the tensor product of the two single-qubit Hilbert spaces. The action of the CNOT gate is thus to do nothing if the first qubit is in state $|0\rangle$ and to apply NOT (an X gate) to the second qubit if the first qubit is in state $|1\rangle$.

Remark 1.9. This way of describing the gate, "if the control is 0 do nothing, if it is 1 apply X", is borrowed from the classical conditional, but it hides what makes these gates useful in the quantum case. Nothing requires the control qubit to be in a definite state. By linearity, feeding in a superposition produces a superposition of the two branches at once,

$$\mathrm{CNOT}\left(\alpha\,|0\rangle + \beta\,|1\rangle\right) \otimes |\psi\rangle = \alpha\;|0\rangle \otimes |\psi\rangle + \beta\;|1\rangle \otimes X\,|\psi\rangle\,,$$

and the same is true of any controlled unitary,

$$\mathrm{CU}\left(\alpha\,|0\rangle + \beta\,|1\rangle\right) \otimes |\psi\rangle = \alpha\;|0\rangle \otimes |\psi\rangle + \beta\;|1\rangle \otimes U\,|\psi\rangle\,.$$

The output carries a term in which the unitary *was* applied and a term in which it *was not*, and the two are correlated with the state of the control register. In general the result is therefore **entangled**, and the two registers can no longer be assigned separate states. Choosing $|\psi\rangle = |0\rangle$ and $\alpha = \beta = 1/\sqrt{2}$ gives the textbook example

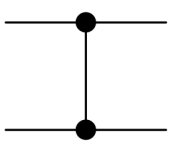
$$\mathrm{CNOT}\,|+\rangle\,|0\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right),$$

one of the four Bell states, produced from a product input by a single gate.

There is no classical counterpart to this: a classical control bit is either 0 or 1, and the controlled operation either happens or does not. Here both branches occur together, and the ability to subsequently apply a unitary to many branches at once, and, crucially, to make those branches interfere later, is what many quantum algorithms take advantage of.

The controlled-Z (CZ) gate can be defined in the same manner:

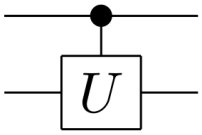
$$\mathrm{CZ} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} = I_2 \oplus Z.$$



Note that this is a symmetric gate: whether the control qubit is the first or the second one gives the same gate.

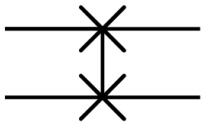
More generally, the single-qubit-controlled application of a unitary U takes the form:

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & & & \\ \vdots & \vdots & & U & \\ 0 & 0 & & & \end{pmatrix} = I_2 \oplus U.$$



There are also two-qubit gates that are not controlled operations. For example, the SWAP gate

$$\text{SWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

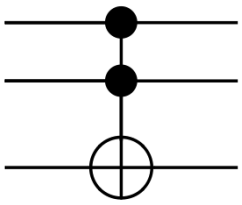


swaps the states of two qubits.

It is also possible to define gates involving more than two qubits. For three-qubit gates, the most well-known ones are the Toffoli and Fredkin gates.

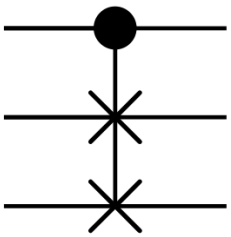
The Toffoli gate is a controlled-controlled-NOT (CCNOT), i.e., the state of the third qubit is flipped if and only if both the first two qubits are in state $|1\rangle$:

$$\text{Toffoli} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$



The Fredkin gate is a controlled-SWAP (CSWAP) gate, swapping the states of the second and third qubits if and only if the state of the first qubit is $|1\rangle$:

$$\text{Fredkin} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$



Most practical implementations of quantum computing have limited connectivity between qubits, only allowing for pairwise interactions between nearest-neighbor qubits. This can prohibit direct implementations of multi-qubit gates with three or more qubits, but it turns out that any multi-qubit gate can be decomposed into a number of single- and two-qubit gates.

1.3.3 Measurements in Quantum Circuits

In the quantum circuit model, the measurement is typically assumed to be in the computational basis. It assigns the observed quantum state to a single value, according to the Born rule: a measurement of a quantum state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ in the computational basis gives the outcome 0 with probability $|\alpha|^2$ and the outcome 1 with probability $|\beta|^2$.

Restricting our attention to the computational basis comes at no loss of generality, because a measurement in any other orthonormal basis can be obtained by applying a suitable unitary beforehand. Let $\{|b_0\rangle, |b_1\rangle\}$ be an orthonormal basis and let V be the unitary defined by $V|i\rangle = |b_i\rangle$. Then

$$|\langle b_i|\psi\rangle|^2 = \left|\langle i| V^\dagger |\psi\rangle\right|^2,$$

so measuring $|\psi\rangle$ in the basis $\{|b_i\rangle\}$ yields exactly the same outcome statistics as applying $V^\dagger$ to $|\psi\rangle$ and then measuring in the computational basis.

The most common instance is the X, or diagonal, basis. Since $H|0\rangle = |+\rangle$ and $H|1\rangle = |-\rangle$, and since $H^\dagger = H$, applying a Hadamard gate and then measuring in the computational basis is the same thing as measuring in the basis $\{|+\rangle, |-\rangle\}$, with the outcome 0 now standing for $|+\rangle$ and 1 for $|-\rangle$. In the same way, applying $S^\dagger$ followed by H and then measuring implements a measurement in the Y basis. Changing the measurement basis is thus not an additional primitive that the hardware must provide: it is a gate, followed by the standard read-out. The same trick will reappear constantly, for instance when measuring the expectation value of a Pauli operator.

The application of a quantum gate can be conditioned on the classical outcome of a measurement.

Figure 1.8: Circuit diagram of measurement. The two lines on the right hand side represent a classical bit.

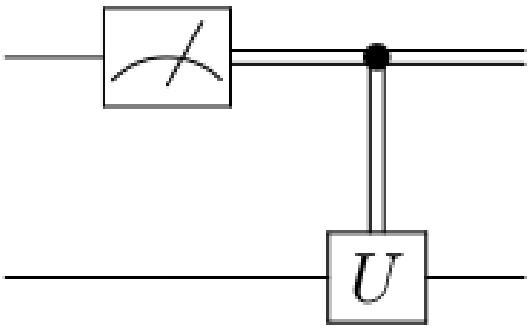


Figure 1.9: Circuit diagram of a quantum gate U conditioned on the measurement of the first qubit.

1.3.4 Universal sets of quantum gates

Are all the gates we specified above enough to carry out any quantum computation that we would like? Could we do any quantum computation using just a small subset of the gates above? These are questions about *universality*.

For quantum computers, a gate set is called universal if the gates therein can be used to approximate any unitary transformation on any number of qubits up to any desired precision:

Definition 1.10. *A set of quantum gates $\mathcal{G} = \{U_1, \ldots, U_k\}$ is called universal if for all $\epsilon > 0$, for all $n \in \mathbb{N}$ and for all unitary U over n qubits there is a circuit V of gates from $\mathcal{G}$ s.t.*

$$\max_{\||\psi\rangle\|=1} \|(U - V)|\psi\rangle\| < \epsilon.$$

Note that we need to allow for approximation because the set of circuits built from finitely many gates is discrete, while the unitary group is continuous.

To understand what makes a gate set universal for quantum computing, let us first see how a gate set can fail to be universal.

- **Inability to create superposition states**: Some gates, e.g., X and CNOT, map all states in the computational basis into other states in the computational basis (e.g., $\text{CNOT}|11\rangle = |10\rangle$). These gates can maintain superpositions, but they cannot create new ones.
- **Inability to create entanglement**: The Hadamard gate can create a superposition (e.g., $H|0\rangle = |+\rangle$), but it only acts on a single qubit, so it cannot create entanglement between two or more qubits. Clearly, no single-qubit gate can. Since an unentangled state of N qubits can be written as $(\alpha_1|0\rangle + \beta_1|1\rangle) \otimes \cdots \otimes (\alpha_N|0\rangle + \beta_N|1\rangle)$, it can be specified using only $2N$ amplitudes, which a classical computer would be able to simulate efficiently.

Let us try a slightly larger set of quantum gates:

Definition 1.11 (Clifford Group)**.** *The Clifford group $\mathcal{C}_n$ on n qubits is the group of $2^n \times 2^n$ unitary matrices that map the Pauli $\mathcal{P}_n$ group on n qubits onto itself:*

$$\mathcal{C}_n := \{V \in \mathcal{U}_{2^n} | \forall P \in \mathcal{P}_n, VPV^\dagger \in \mathcal{P}_n\}.$$

Equivalently, this is the group generated by the gate set $\{H, S, \text{CNOT}\}$.

If we take our gate set to be $\{H, S, \text{CNOT}\}$, we circumvent both the objections above. However, it turns out that this still is not enough to achieve universal quantum computation.

This is understood noting that, to global phases, the Clifford group $\mathcal{C}_n$ is a *finite* group. Indeed, a Clifford unitary V is determined, up to a phase, by the images $VP_iV^\dagger$ of a generating set of the Pauli group, and since conjugation merely permutes the finitely many elements of $\mathcal{P}_n$, there are only finitely many possibilities. A finite set of points cannot be dense in the continuum $SU(2^n)$, so by Definition 1.10 the Clifford gates cannot be universal: there exist unitaries that they do not merely fail to reach exactly, but fail to approximate at all. The T gate is the standard example: it sits at a minimum finite distance from every Clifford operation, no matter how long a circuit we are willing to write down.

What is then a universal gate set? Actually, replacing either the S or H gate in the last gate set considered above, $\{H, S, \text{CNOT}\}$, with *almost any* other single-qubit gate makes the set universal (Shi, 2002). One common universal gate set is $\{H, \text{CNOT}, T\}$. Almost any two-qubit gate on its own is also universal. Interestingly, $\{H, \text{Toffoli}\}$ is a universal gate set for quantum computations (Aharonov, 2003), while the Toffoli gate on its own is universal for *reversible* classical computation (and hence, given ancillas prepared in fixed states, for classical computation in general).

Remark 1.12. The sense in which $\{H, \text{Toffoli}\}$ is universal deserves a comment, because it is *not* the sense of Definition 1.10. Both H and Toffoli have purely real matrix entries, so every circuit built out of them is a real orthogonal matrix. Such circuits can therefore never form a dense subset of $SU(2^n)$: no product of H and Toffoli gates comes anywhere near the S gate, whose entries are not real. What is true is that $\{H, \text{Toffoli}\}$ generates a dense subgroup of the real orthogonal group $SO(2^n)$, and that this is enough to reproduce any quantum computation with only polynomial overhead, by encoding the real

and imaginary parts of the amplitudes separately: a complex computation on n qubits is carried out as a real computation on $n+1$ qubits. This weaker property — being able to reproduce the *output statistics* of any quantum circuit efficiently, rather than to approximate any unitary in operator norm — is usually called computational, or encoded, universality. It is the notion that matters for complexity-theoretic statements such as the definition of BQP, and it is a useful reminder that "universal" is not a single unambiguous property: one should always ask universal *in what sense*.

Once we have a set that can approximately realize any (special) unitary operation, it is a different question whether that approximation is fast and efficient. For example, if the number of gates needed would scale exponentially in $1/\epsilon$, there would probably not be any practical use for such a universal gate set. Luckily, there is a theorem telling us that the scaling for any universal gate set is much more benign:

Theorem 1.13 (Solovay-Kitaev Theorem Kitaev (1997))**.** *Let G be a finite subset of $SU(2)$ that is closed under inverses, and let $U \in SU(2)$. If the group generated by G is dense in $SU(2)$, then for any $\epsilon > 0$ it is possible to approximate U to operator norm error ϵ using $O(\log^4(1/\epsilon))$ gates from G. This gate sequence can be produced by a classical algorithm with a running time of $O(\log^3(1/\epsilon))$.*

See Dawson and Nielsen (2006) for a constructive proof. What this theorem essentially says is that if we have a gate set that can approximate any single-qubit rotation, then we only need relatively few gates from that set to do the approximation, so we can do the approximation fast. For certain gate sets, the number of gates can be brought down from $O(\log^4(1/\epsilon))$ to $O(\log(1/\epsilon))$ (Harrow, Recht, and Chuang, 2002). By adding some two-qubit gates to make the gate set universal, the total number of gates needed to approximate a quantum circuit of m constant-qubit gates is at most $O(m \log^4(m/\epsilon))$.

Example 1.14. Suppose we want to run a quantum algorithm, consisting in a unitary U, which can be written using N gates from a given gate set $\mathcal{G}$, for example $\mathcal{G} = \{S, H, T, \mathrm{CNOT}\}$: $U = U_N U_{N-1} U_{N-2} \cdots$, $U_j \in \mathcal{G}$. Suppose we have access to a machine that *cannot* implement the gates in $\mathcal{G}$ natively. However, it can implement a set of gates $\mathcal{G}'$ which satisfies the hypotheses of the SK theorem. Then we can replace each U_j in U with an approximation built from this new instruction set

$$\tilde{U}_j = V_m^{(j)} V_{m-1}^{(j)} \cdots \tag{1.1}$$

making some error δ

$$\left\| U_j - \tilde{U}_j \right\| \leq \delta \tag{1.2}$$

The errors of the individual gates then accumulate, but only additively. To see this, write the difference between the exact and the approximated circuit as a *telescoping* sum, in which the gates are swapped one at a time,

$$\begin{aligned} U - \tilde{U} &= U_N U_{N-1} \cdots U_1 \; - \; \tilde{U}_N \tilde{U}_{N-1} \cdots \tilde{U}_1 \\ &= \sum_{j=1}^{N} \tilde{U}_N \cdots \tilde{U}_{j+1} \left(U_j - \tilde{U}_j \right) U_{j-1} \cdots U_1, \end{aligned} \tag{1.3}$$

where consecutive terms of the sum differ by a single replacement $U_j \to \tilde{U}_j$, so that all the intermediate contributions cancel pairwise and only the two extreme products survive. Now take norms. The triangle inequality bounds the norm of the sum by the sum of the norms, and the operator norm is invariant under multiplication by unitaries, $\|VAW\| = \|A\|$ for V, W unitary, so each term contributes at most δ:

$$\left\| U - \tilde{U} \right\| \leq \sum_{j=1}^{N} \left\| U_j - \tilde{U}_j \right\| \leq N\delta. \tag{1.4}$$

If we now require that the overall error is $N\delta < \epsilon$ this implies $\delta < \epsilon/N$. Thanks to SK, approximating a single gate to accuracy δ costs $m = O\big(\log^4 \tfrac{1}{\delta}\big) = O\big(\log^4 \tfrac{N}{\epsilon}\big)$ gates, and the total unitary using the new gate set can be written

$$\tilde{U} = V_M V_{M-1} \cdots \tag{1.5}$$

using $M = O\big(N \log^4 \tfrac{N}{\epsilon}\big)$ gates: **we only incur a polylogarithmic overhead!**

Remark 1.15. Two provisos about the way the Solovay-Kitaev theorem was used in the example. First, the theorem as stated above concerns $SU(2)$, that is, single-qubit gates. It does generalise to $SU(d)$ for arbitrary d, with a bound of the same $O(\log^c(1/\epsilon))$ form, but the constant hidden inside the $O(\cdot)$ grows with the dimension d: approximating a unitary becomes harder as the space it acts on gets larger, and for unbounded d the statement carries no useful content.

Second, this is not a difficulty in the situation of the example, and the reason is worth making explicit. There we never approximate the full 2^n-dimensional unitary U in one go; we only ever approximate the individual gates U_j of the instruction set $\mathcal{G}$. If every gate of $\mathcal{G}$ acts on at most k qudits of local dimension D, then each U_j lives in $SU(D^k)$, and D^k is a constant fixed by the architecture — independent of the number n of qudits in the register and of the number N of gates in the circuit. The Solovay-Kitaev constant is then bounded once and for all, and the polylogarithmic overhead derived above holds uniformly in n and N. This is a further reason why it is convenient to require that an instruction set consist of gates acting on a bounded number of qudits, as every realistic architecture does anyway.

Together with the observations above, this gives us confidence that quantum computing has potential:

- We know how to find a universal gate set to approximate any unitary transformation.
- We know that we can implement that approximation efficiently.
- We know that we can implement any unitary operation on N qubits in a realistic experimental architecture that allows for a few single- and two-qubit gates.

Does that mean that we can run quantum algorithms that are faster than classical algorithms for some problems? This will be discussed in the next chapters.

Resources and further reading

The material on classical computation theory was largely based on a course given by Chiara Macchiavello at the University of Pavia. The rest of the chapter is adapted from the notes used by Ulysse Chabaud for this same course in past years, which are in turn based on the lecture notes of Kockum et al. (2023) and on the book of Nielsen and Chuang (2000).

References for this chapter

Aharonov, Dorit (2003). "A Simple Proof that Toffoli and Hadamard are Quantum Universal". In: *arXiv preprint quant-ph/0301040*. URL: https://arxiv.org/abs/quant-ph/0301040.

Dawson, Christopher M. and Michael A. Nielsen (2006). "The Solovay-Kitaev algorithm". In: *Quantum Information & Computation* 6.1, pp. 81–95. URL: https://arxiv.org/abs/quant-ph/0505030.

Feynman, Richard P. (1982). "Simulating physics with computers". In: *International Journal of Theoretical Physics* 21.6, pp. 467–488. DOI: 10.1007/BF02650179.

Harrow, Aram W., Benjamin Recht, and Isaac L. Chuang (2002). "Efficient discrete approximations of quantum gates". In: *Journal of Mathematical Physics* 43.9, pp. 4445–4451. DOI: 10.1063/1.1495899. URL: https://arxiv.org/abs/quant-ph/0111031.

Kitaev, A. Yu. (1997). "Quantum computations: algorithms and error correction". In: *Russian Mathematical Surveys* 52.6, pp. 1191–1249. DOI: 10.1070/RM1997v052n06ABEH002155.

Kockum, Anton Frisk et al. (2023). *Lecture notes on quantum computing*. arXiv:2311.08445. DOI: 10.48550/arXiv.2311.08445. URL: https://arxiv.org/abs/2311.08445.

Nielsen, Michael A. and Isaac L. Chuang (2000). *Quantum Computation and Quantum Information*. Cambridge University Press. ISBN: 978-0-521-63503-5.

Shi, Yaoyun (2002). "Both Toffoli and CNOT need little help to do universal quantum computing". In: *arXiv preprint quant-ph/0205115*. URL: https://arxiv.org/abs/quant-ph/0205115.

Chapter 2

The Quantum Fourier Transform and the Hidden Subgroup Problem

The Quantum Fourier Transform (QFT) is one of the most important building blocks in quantum computation. It is the quantum analogue of the classical Discrete Fourier Transform (DFT) and is a key component in many groundbreaking quantum algorithms, including Shor's algorithm for factoring and the quantum phase estimation algorithm. Its power lies in its ability to perform a Fourier transform on an exponentially large vector of amplitudes in polynomial time. This chapter will thus first introduce the classical DFT and its efficient implementation, the Fast Fourier Transform (FFT). We will then construct the quantum circuit for the QFT and analyze its remarkable efficiency. With the QFT in our toolkit, we will then introduce the Hidden Subgroup Problem (HSP). The HSP provides a powerful and unifying framework that encapsulates several famous quantum algorithms, including those of Deutsch, Simon, and Shor. We will see that these seemingly different problems are all instances of a single, more general problem: finding a hidden subgroup within a larger group structure. The QFT provides the key to solving the HSP for a large and important class of groups known as abelian groups.

This chapter closely follows chapters 4 and 6 of the lecture notes of de Wolf (2023).

2.1 The Quantum Fourier Transform (QFT)

Summary

- The quantum Fourier transform is an algorithm to compute the classical discrete Fourier transform using an amplitude encoding
- The classical DFT is very important for many applications in science and technology and relies on a fast algorithm known as Cooley-Tukey algorithm. It uses a "divide and conquer approach" where the problem is recursively broken in smaller instances of itself. This is particularly easy to explain for vectors in dimensions that are power of two $N = 2^n$. The runtime of the algorithm is $O(N \log N) = O(2^n n)$.
- The quantum algorithm exploits the possibility to use superpositions and controlled coherent operations and results in a runtime $O\left(n^2\right)$: an exponential advantage.

The Fourier transform is a fundamental mathematical tool used across science and engineering, primarily for converting signals from the time or spatial domain to the frequency domain. While it is often introduced in terms of a transformation between functions of continuous variables, there is a fairly straightforward modification that allows to use it for discrete signals, for example acquired through sampling. It is this discrete version that inspired the quantum counterpart, which operates on the

amplitudes of a quantum state.

2.1.1 The Discrete Fourier Transform (DFT)

We start by introducing the Discrete Fourier Transform.

Definition 2.1 (Discrete Fourier Transform)**.** *Let $\boldsymbol{x} = (x_0, x_1, \ldots, x_{N-1}) \in \mathbb{C}^N$ be a vector of N complex numbers. The **Discrete Fourier Transform (DFT)** of $\boldsymbol{x}$ is a vector $\boldsymbol{y} = (y_0, y_1, \ldots, y_{N-1}) \in \mathbb{C}^N$ where each component y_k is given by:*

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \, \omega_N^{jk} \tag{2.1}$$

where $\omega_N = e^{2\pi i/N}$ is a primitive N-th root of unity ($(\omega_N)^N = 1$).

The DFT can be represented as a linear transformation $\boldsymbol{y} = F_N \boldsymbol{x}$ given by an $N \times N$ matrix F_N, known as the Fourier matrix. The entries of this matrix are $(F_N)_{kj} = \frac{1}{\sqrt{N}} \omega_N^{jk}$. This matrix is unitary, meaning $F_N F_N^\dagger = F_N^\dagger F_N = I$, which is a crucial property for its implementation as a reversible quantum computation. It is also symmetric, so we have $F_N^\dagger = F_N^*$.

2.1.2 The Classical Fast Fourier Transform (FFT)

A naive calculation of the DFT using Equation 2.1 requires computing N sums, each with N terms. This results in a computational complexity of $O(N^2)$. While this is polynomial time, for large N, this can be prohibitively slow and would make many real-world applications unfeasible (for example digital communications, music, image and video processing, medical imaging...).

The **Fast Fourier Transform (FFT)** is a highly efficient algorithm for computing the DFT. The most common version, the Cooley-Tukey algorithm, is a divide-and-conquer algorithm that works by recursively breaking down a DFT of composite size $N = N_1 N_2$ into smaller DFTs of sizes N_1 and N_2. When N is a power of 2, say $N = 2^n$, this approach is particularly effective and easy to describe. Unless otherwise stated, we will restrict to this case for the rest of the chapter. The algorithm cleverly reuses intermediate results to avoid redundant calculations, achieving a remarkable complexity of $O(N \log N)$. This dramatic speedup made the DFT practical for a vast range of applications. To understand the basic idea, let us rewrite the definition (2.1), dropping the overall factor $1/\sqrt{N}$ which plays no role in the counting, and splitting the sum into even and odd indices:

$$y_k = \sum_{j=0}^{N-1} x_j e^{\frac{2\pi i}{N} jk} = \sum_{m=0}^{N/2-1} x_{2m} e^{\frac{2\pi i}{N}(2m)k} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{\frac{2\pi i}{N}(2m+1)k} \tag{2.2}$$

$$= \underbrace{\sum_{m=0}^{N/2-1} x_{2m} e^{\frac{2\pi i}{N/2} mk}}_{\text{DFT of even-indexed part of } x_j} + e^{\frac{2\pi i}{N}k} \underbrace{\sum_{m=0}^{N/2-1} x_{2m+1} e^{\frac{2\pi i}{N/2} mk}}_{\text{DFT of odd-indexed part of } x_j} = E_k + e^{\frac{2\pi i}{N}k} O_k \tag{2.3}$$

for $k = 0, \ldots, \frac{N}{2} - 1$. The last step suggests the possibility for a recursive implementation. An important point to note is that we do not need to calculate E_k, O_k explicitly for $k \geq N/2$. This is due to the periodicity of the complex exponential, $e^{\frac{2\pi i}{N/2} m(k+N/2)} = e^{\frac{2\pi i}{N/2} mk}$, which gives $E_{k+N/2} = E_k$ and $O_{k+N/2} = O_k$, while $e^{\frac{2\pi i}{N}(k+N/2)} = -e^{\frac{2\pi i}{N}k}$. A straightforward calculation then shows

$$y_{k+\frac{N}{2}} = E_k - e^{\frac{2\pi i}{N}k} O_k \tag{2.4}$$

so we can reuse the quantities computed for y_k. Combining E_k and O_k to obtain y_k, $y_{k+\frac{N}{2}}$ is called a *butterfly operation.*

The efficiency of the Fast Fourier Transform (FFT) algorithm, specifically the radix-2 Cooley-Tukey variant, stems from its divide-and-conquer approach. The computational complexity can be derived by analyzing its recurrence relation. Let $T(N)$ be the number of complex operations (multiplications and additions) required to compute an N-point Discrete Fourier Transform (DFT), where N is a power of 2. The algorithm recursively divides the N-point problem into two subproblems of size $N/2$,[1] which gives the recurrence relation:

$$T(N) = 2T(N/2) + C(N) \tag{2.5}$$

where $C(N)$ is the cost of the "combine" step. The combine step involves a series of butterfly operations. A single butterfly requires one complex multiplication and two complex additions/subtractions. Since this is performed for all $N/2$ values of k, the total number of operations in the combine step is proportional to N.

$$C(N) = \frac{N}{2} \times (1 \text{ mult} + 2 \text{ adds}) \implies C(N) \in O(N) \tag{2.6}$$

The recurrence relation thus becomes:

$$T(N) = 2T(N/2) + O(N) \tag{2.7}$$

This recurrence can be solved by considering the recursion depth. The problem of size N is repeatedly halved until we reach the base case, $T(1)$, which is $O(1)$. This division process creates a recursion tree of depth $\log_2(N)$. The total complexity is the product of the work per stage and the number of stages:

$$\text{Total Complexity} = (\text{Number of Stages}) \times (\text{Work per Stage}) = \log_2(N) \times O(N) \tag{2.8}$$

This gives the final time complexity of $\mathbf{T(N) = O(N \log N)}$. This is a dramatic improvement over the $O(N^2)$ complexity of the naive DFT calculation, making large-scale Fourier analysis computationally feasible.

2.1.3 The Quantum Algorithm for the Fourier Transform

The Quantum Fourier Transform is the quantum analogue of the DFT. Instead of transforming a vector of numbers, it transforms the amplitudes of a quantum state. For the remainder of this section, we will assume the dimension of our Hilbert space is $N = 2^n$ for some integer n, corresponding to a register of n qubits.

Definition 2.2 (Quantum Fourier Transform)**.** *The **Quantum Fourier Transform (QFT)** is a unitary transformation on n qubits that acts on the computational basis states $|j\rangle$ for $j \in \{0, 1, \ldots, N-1\}$ as follows:*

$$\hat{F}_N |j\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i jk/N} |k\rangle \tag{2.9}$$

Applying the QFT to an arbitrary state $|\psi\rangle = \sum_{j=0}^{N-1} x_j |j\rangle$ results in the state $\sum_{k=0}^{N-1} y_k |k\rangle$, where the amplitudes y_k are precisely the DFT of the amplitudes x_j. The power of the QFT lies in its efficient circuit implementation.

[1] Each subproblem computes a whole vector at once. Writing $\boldsymbol{x}^{\text{even}} = (x_0, x_2, \ldots, x_{N-2})$ and $\boldsymbol{x}^{\text{odd}} = (x_1, x_3, \ldots, x_{N-1})$, the vectors $\boldsymbol{E} = (E_0, \ldots, E_{N/2-1})$ and $\boldsymbol{O} = (O_0, \ldots, O_{N/2-1})$ are, up to the normalization dropped above, $F_{N/2}\,\boldsymbol{x}^{\text{even}}$ and $F_{N/2}\,\boldsymbol{x}^{\text{odd}}$: all $N/2$ components are obtained together, not one component at a time. If one stopped at this first level of the recursion, $\boldsymbol{E}$ and $\boldsymbol{O}$ could be obtained by explicit matrix multiplication, at a cost $2(N/2)^2 = N^2/2$ — only a constant-factor saving over the naive $O(N^2)$. Instead, one can continue the recursion: $\boldsymbol{E}$ and $\boldsymbol{O}$ are themselves DFTs of size $N/2$, which are split again into their own even and odd parts, and so on down to size 1, where $F_1 = 1$ and there is nothing to compute. Each level of the recursion returns all the even and odd components needed at that level, and the vector of that level is reconstructed from them by the combine step, i.e. by the butterfly operations. This is why $T(N/2)$ in the recurrence counts the cost of the *whole* half-size transform, computed once and then shared by all the y_k and $y_{k+N/2}$; recomputing it for each k would bring back the $O(N^2)$ cost.

The QFT Circuit

To derive an efficient circuit, we first rewrite the QFT definition using a binary representation for the states. Let the binary representation of j be $j_1 j_2 \dots j_n$, where $j = \sum_{l=1}^{n} j_l 2^{n-l}$. We also introduce the binary fraction notation $0.j_l j_{l+1} \dots j_m = \sum_{k=l}^{m} j_k 2^{l-k-1}$.

Let us massage the definition to show that the state QFT $|j\rangle$ can be written in a product form:

$$\begin{aligned}
\hat{F}_N |j_1 j_2 \dots j_n\rangle &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i j k/2^n} |k\rangle \\
\text{(writing } k \text{ in binary)} &= \frac{1}{\sqrt{N}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} e^{2\pi i j (\sum_{l=1}^{n} k_l 2^{n-l})/2^n} |k_1 \dots k_n\rangle \\
\text{(writing the exponential as product)} &= \frac{1}{\sqrt{N}} \sum_{k_1,\dots,k_n} \bigotimes_{l=1}^{n} e^{2\pi i j k_l 2^{-l}} |k_l\rangle \\
\text{(this step can be verified a posteriori)} &= \frac{1}{\sqrt{N}} \bigotimes_{l=1}^{n} \left(\sum_{k_l=0}^{1} e^{2\pi i j k_l 2^{-l}} |k_l\rangle \right) \\
&= \frac{1}{\sqrt{2^n}} \bigotimes_{l=1}^{n} \left(|0\rangle + e^{2\pi i j/2^l} |1\rangle \right) \qquad (2.10)
\end{aligned}$$

The phase of the l-th qubit is $e^{2\pi i j/2^l}$. Let's expand j in binary as well:

$$\frac{j}{2^l} = \frac{j_1 2^{n-1} + \dots + j_n 2^0}{2^l} = \sum_{m=1}^{n} j_m 2^{n-m-l}$$

Since $e^{2\pi i \cdot \text{integer}} = 1$, only the fractional part of this sum matters. The phase term becomes $e^{2\pi i (0.j_{n-l+1} \dots j_n)}$. So the state of the l-th qubit is:

$$\frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i (0.j_{n-l+1} \dots j_n)} |1\rangle \right)$$

This leads directly to a quantum circuit. Let's construct it from an example. For $N = 8$ we have three qubits and

$$\begin{aligned}
\hat{F}_N |j_1 j_2 j_3\rangle &= \frac{1}{\sqrt{8}} \left(|0\rangle + e^{2\pi i j/2} |1\rangle \right) \left(|0\rangle + e^{2\pi i j/4} |1\rangle \right) \left(|0\rangle + e^{2\pi i j/8} |1\rangle \right) \\
&= \frac{1}{\sqrt{8}} \left(|0\rangle + e^{2\pi i (j_1 \cdot 2^2)/2} e^{2\pi i (j_2 \cdot 2)/2} e^{2\pi i j_3/2} |1\rangle \right) \left(|0\rangle + e^{2\pi i j/4} |1\rangle \right) \left(|0\rangle + e^{2\pi i j/8} |1\rangle \right) \\
&= \frac{1}{\sqrt{8}} \left(|0\rangle + e^{\pi i j_3} |1\rangle \right) \left(|0\rangle + e^{2\pi i (j_1 \cdot 2^2)/4} e^{2\pi i (j_2 \cdot 2)/4} e^{2\pi i j_3/4} |1\rangle \right) \left(|0\rangle + e^{2\pi i j/8} |1\rangle \right) \\
&= \frac{1}{\sqrt{8}} \left(|0\rangle + e^{\pi i j_3} |1\rangle \right) \left(|0\rangle + e^{\pi i j_2} e^{\pi i j_3/2} |1\rangle \right) \left(|0\rangle + e^{2\pi i j/8} |1\rangle \right) \\
&\dots \\
&= \frac{1}{\sqrt{8}} \left(|0\rangle + e^{\pi i j_3} |1\rangle \right) \left(|0\rangle + e^{\pi i j_2} e^{\pi i j_3/2} |1\rangle \right) \left(|0\rangle + e^{\pi i j_1} e^{\pi i j_2/2} e^{\pi i j_3/4} |1\rangle \right)
\end{aligned}$$

Now, if we don't care about which qubit is in which state, we could prepare the third qubit in the state

$$|\psi_1\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle + e^{\pi i j_3} |1\rangle \right) = \frac{1}{\sqrt{2}} \left(|0\rangle + (-1)^{j_3} |1\rangle \right) = H|j_3\rangle \qquad (2.11)$$

by simply applying a Hadamard gate. The state

$$|\psi_2\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle + e^{\pi i j_2} e^{\pi i j_3/2} |1\rangle \right) \qquad (2.12)$$

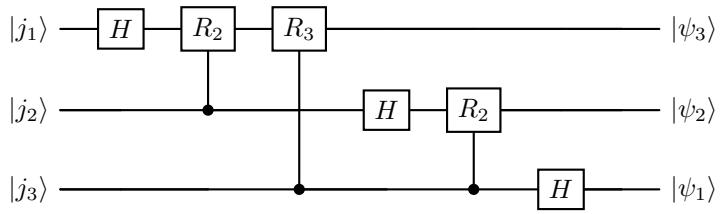
could be obtained on the second qubit if we apply a Hadamard to it, followed by the phase gate

$$R_m = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^m} \end{pmatrix} \tag{2.13}$$

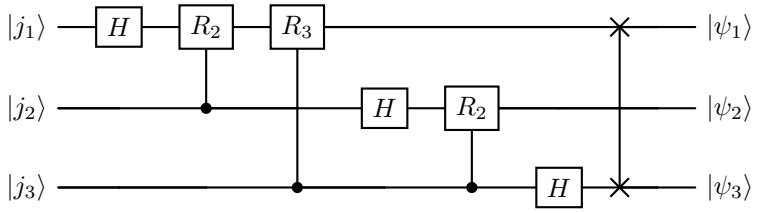
with $m = 2$, applied only if the third qubit was in state $|1\rangle$ *before* its Hadamard, i.e. if $j_3 = 1$. Note that $R_2 = S$ and $R_3 = T$. It is tempting to identify R_m with the rotation $R_z(2\pi/2^m) = e^{-i\pi Z/2^m}$, which differs from it only by the global phase $e^{-i\pi/2^m}$; but this identification is *wrong* here, because the gate is applied *conditionally*. In a controlled gate a global phase of the target becomes a relative phase between the $|0\rangle$ and $|1\rangle$ branches of the control, and the resulting circuit is no longer the QFT, not even up to a global phase. Similarly we can prepare

$$|\psi_3\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + e^{\pi i j_1} e^{\pi i j_2/2} e^{\pi i j_3/4} |1\rangle\right) \tag{2.14}$$

on the first qubit if we apply a Hadamard, a phase gate R_2 controlled on the second qubit before its Hadamard and a phase gate R_3 controlled on the third qubit before its Hadamard. We end up with the circuit



The only missing step is now to re-order the qubits, which can be achieved with a SWAP gate



In general, before the swap, the state of the first output qubit depends on all input qubits $(j_1, \ldots, j_n)$, the second depends on $j_2, \ldots, j_n$, and so on. We then have the following recipe for constructing the general circuit:

1. For the first qubit (input $|j_1\rangle$), apply a Hadamard gate. Then, for each subsequent qubit $k = 2, \ldots, n$, apply a controlled phase gate, controlled by qubit k and acting on qubit 1. The gate controlled by qubit k is R_k, with R_m the phase gate of equation (2.13).
2. For the second qubit (input $|j_2\rangle$), apply a Hadamard gate. Then apply controlled rotations from qubits $k = 3, \ldots, n$ with gates R_{k-1}.
3. Continue this process for all n qubits. The m-th qubit gets a Hadamard gate followed by controlled rotations R_{k-m+1} from qubits $k = m+1, \ldots, n$.
4. The resulting qubits are in the correct state, but in reverse order. A final series of SWAP gates is required to reverse the qubit order from $|k_n k_{n-1} \ldots k_1\rangle$ to $|k_1 k_2 \ldots k_n\rangle$.

Complexity and Comparison

Let's count the gates in the n-qubit QFT circuit.

- Number of Hadamard gates: n.
- Number of controlled-rotation gates: $(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}$.
- Number of SWAP gates: $\lfloor n/2 \rfloor$.

The total number of gates is $O(n^2)$. Since $n=\log_2 N$, the complexity in terms of N is $O((\log N)^2)$.

This is an **exponential speedup** over the classical FFT, which runs in $O(N \log N)=O(2^n n)$. However, there is a crucial caveat. After running the QFT, the result is stored as amplitudes in a quantum state. By the principles of quantum measurement, we cannot access all N amplitudes directly. Doing so would require running the algorithm and measuring $O(N)$ times, negating the speedup. The power of the QFT is realized when its output can be used to extract global properties of the state, such as its period, which is precisely what is done in Shor's algorithm and the solution to the HSP.

As a final remark, we note that in our complexity analysis we have assumed that all rotations R_m are elementary gates, so we counted each of them as requiring "unit time". If this were not the case, we would have to compile each of them using the native instructions of our gate set. However, since each acts on exactly two qubits, we can invoke the Solovay-Kitaev algorithm to argue that the complexity only suffers from a polylogarithmic overhead due to this compilation (see Remark 1.15: since each gate acts on two qubits, the Solovay-Kitaev constant is bounded).

2.2 The Hidden Subgroup Problem (HSP)

Summary

Group theory is a powerful language to describe symmetries. A number of elementary and also more sophisticated quantum algorithms rely on some symmetries of the problem. In particular, several famous algorithms such as Deutsch, Simon and period finding (used in Shor's algorithm) can be phrased as "identify a function from a group to a set under the promise that it is constant on a subgroup". This corresponds to identifying a "hidden" subgroup, encoded in the function in question. This allows to develop a general class of algorithms based on a generalization of the Fourier transform for finite abelian groups involving the group's characters, i.e. its one-dimensional representations. The standard DFT is then recovered as the FT on $\mathbb{Z}_N$. These algorithms will lead to quantum advantage thanks to the speedup inherited from the QFT.

The Hidden Subgroup Problem is a cornerstone of quantum algorithm design. It provides a general framework that describes a wide variety of problems that can be solved efficiently on a quantum computer, but are believed to be hard for classical computers.

2.2.1 A Group Theory Primer

To understand the HSP, we need some basic definitions from group theory.

Definition 2.3 (Group)**.** *A **group** is a set G together with a binary operation $\cdot : G \times G \to G$ that satisfies the following four axioms:*

1. ***Closure:** For all $a, b \in G$, the result $a \cdot b$ is also in G.*
2. ***Associativity:** For all $a, b, c \in G$, we have $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.*
3. ***Identity element:** There exists an element $e \in G$ such that for all $a \in G$, $a \cdot e = e \cdot a = a$.*
4. ***Inverse element:** For each $a \in G$, there exists an element $a^{-1} \in G$ such that $a \cdot a^{-1} = a^{-1} \cdot a = e$.*

*A group is called **abelian** (or commutative) if for all $a, b \in G$, $a \cdot b = b \cdot a$.*

Definition 2.4 (Subgroup and Cosets)**.** *A subset $H \subseteq G$ is a **subgroup** of G (denoted $H \leq G$) if H itself forms a group under the same operation as G. For any element $g \in G$, the **left coset** of H with respect to g is the set $gH = \{g \cdot h : h \in H\}$.*

The set of left cosets of H in G forms a partition of G. That is, every element of G belongs to exactly one left coset.

Example 2.5 (Important Groups)**.**

1. **The Integers:** The set of integers $\mathbb{Z}$ under addition $(+)$ is an infinite abelian group. The identity is 0 and the inverse of k is $-k$. Subgroups are of the form $k\mathbb{Z} = \{\dots, -2k, -k, 0, k, 2k, \dots\}$ for some integer k.
2. **Integers Modulo N:** The set $\mathbb{Z}_N = \{0, 1, \dots, N-1\}$ under addition modulo N is a finite abelian group. The subgroups are generated by divisors of N.
3. **The Multiplicative Group of Integers Modulo N:** The set $\mathbb{Z}_N^* = \{a \in \{1, \dots, N-1\} : \gcd(a, N) = 1\}$ under multiplication modulo N forms a finite abelian group. The size of this group is given by Euler's totient function, $\phi(N)$. This group is central to Shor's algorithm for factoring.
4. **The Boolean Group:** The set $(\mathbb{Z}_2)^n = \{0,1\}^n$ of n-bit strings under the operation of bitwise XOR $(\oplus)$ is a finite abelian group. Every element is its own inverse. This group is central to Simon's algorithm.

2.2.2 Problem Statement

The HSP is defined with respect to a function that "hides" a subgroup.

Problem 2.2.1 (The Hidden Subgroup Problem (HSP))**.** *Let G be a group, X be a finite set, and $f : G \to X$ be a function which we can evaluate on inputs of our choice, but whose inner workings we cannot inspect. The function f has the promise that there exists a subgroup $H \leq G$ such that for any $g_1, g_2 \in G$:*

$$f(g_1) = f(g_2) \quad \iff \quad g_1 H = g_2 H \tag{2.15}$$

This means that f is constant on the cosets of H and takes distinct values for different cosets. The goal of the HSP is to find a generating set for the hidden subgroup H.

Note that this is a purely classical problem: nothing in its statement refers to quantum mechanics. A function that can only be evaluated, but not inspected, is called a **black box** or **oracle**, and the cost of an algorithm is then measured by the number of evaluations, or *queries*, it makes, together with the rest of the computation. A classical algorithm queries f on one input g at a time.

A quantum computer can only use f through a unitary operation. We therefore say that a quantum algorithm has **quantum oracle access** to f if it can apply

$$U_f : |g\rangle\,|x\rangle \mapsto |g\rangle\,|x \oplus f(g)\rangle\,, \tag{2.16}$$

where the elements of X are encoded as bit strings, so that $\oplus$ (bitwise XOR) makes sense. This is the reversible construction of Section 1.1.3, written in Dirac notation, and like any unitary it can act on superpositions of inputs. It does not give the quantum computer any unfair advantage: from any classical circuit computing f, a circuit for U_f can be built with only polynomial overhead.

The difficulty of the HSP depends critically on the group G. If G is a finite abelian group, a quantum computer with oracle access to f can solve the HSP in time polynomial in $\log|G|$ — that is, polynomial in the number of qubits needed to store an element of G — as we will see in Section 2.2.4. Classically, by contrast, the number of queries needed can be exponential in $\log|G|$; Simon's problem, discussed below, is an example. If G is non-abelian, the problem is much harder, and efficient quantum algorithms are known only for specific cases. The famous Graph Isomorphism and certain Lattice Problems can be phrased as HSP over non-abelian groups.

2.2.3 Phrasing Famous Algorithms as HSP

We now show that several key quantum algorithms are just special instances of the HSP. We only recall the problems here; the algorithms themselves are then obtained all at once from the general algorithm of Section 2.2.4. Detailed explanations of the individual algorithms can be found in the book of Nielsen and Chuang (2000).

Deutsch's Problem as HSP

Deutsch's problem is the following: given a function $f : \{0,1\} \to \{0,1\}$, accessible only through the oracle U_f, determine whether f is constant ($f(0) = f(1)$) or balanced ($f(0) \neq f(1)$). Classically this requires two queries. The problem and its first quantum solution are due to Deutsch (1985); Deutsch's original algorithm used a single query but succeeded only with probability $1/2$. The deterministic single-query version usually taught today is due to Cleve et al. (1998).

- **Group G:** $G = \mathbb{Z}_2 = \{0,1\}$ with addition modulo 2.
- **Function f:** The given $f : \mathbb{Z}_2 \to \{0,1\}$.
- **Hidden Subgroup H:**
  - If f is **constant**, then $f(0) = f(1)$. This means f is constant on the entire group G. The hidden subgroup is $H = G = \mathbb{Z}_2$. The only coset is G itself.
  - If f is **balanced**, then $f(0) \neq f(1)$. The function is constant on the trivial subgroup $H = \{0\}$. The cosets are $\{0\}$ and $1H = \{1\}$, and f takes different values on them.
- **Goal:** Distinguish between $H = \{0\}$ and $H = \mathbb{Z}_2$. Finding the subgroup solves the problem.

Simon's Problem as HSP

Simon's problem is the following: given $f : \{0,1\}^n \to \{0,1\}^n$, we are promised that there exists a secret string $s \in \{0,1\}^n$ such that for all $x, y \in \{0,1\}^n$, $f(x) = f(y)$ if and only if $y \in \{x, x \oplus s\}$. The goal is to find s. (If $s = 0^n$ the promise simply says that f is injective.) Any classical algorithm needs exponentially many queries in n, whereas the quantum algorithm below needs $O(n)$. The problem and its quantum solution were introduced by Simon (1997); it was the first example of an exponential quantum speed-up relative to an oracle, and it directly inspired Shor's factoring algorithm.

- **Group G:** $G = (\mathbb{Z}_2)^n$, the group of n-bit strings with bitwise XOR.
- **Function f:** The given $f : (\mathbb{Z}_2)^n \to \{0,1\}^n$.
- **Hidden Subgroup H:** The condition $f(x) = f(y) \iff y = x \oplus s$ means that f is constant on cosets of the subgroup $H = \{0^n, s\}$. The cosets are of the form $\{x, x \oplus s\}$.
- **Goal:** Find the non-trivial generator s of the hidden subgroup H.

Period Finding as HSP (Shor's Algorithm)

The core of Shor's factoring algorithm is finding the period of a function. Given a function $f(x) = a^x \pmod N$ for coprime a, N, we need to find the smallest positive integer r such that $a^r \equiv 1 \pmod N$. This means $f(x + r) = a^{x+r} = a^x a^r \equiv a^x \pmod N = f(x)$, so f is periodic with period r.

- **Group G:** The natural group is the integers, $G = \mathbb{Z}$, and the exact hidden subgroup is $r\mathbb{Z}$. Since quantum computers operate on finite registers, one would like to replace $\mathbb{Z}$ by a finite cyclic group

$\mathbb{Z}_M$. For f to be well defined on $\mathbb{Z}_M$ we need $r \mid M$. A choice that works in principle is $M = \phi(N)$, where ϕ is Euler's totient function, i.e. the size of the multiplicative group $\mathbb{Z}_N^*$: the element a generates a cyclic subgroup $\{1, a, a^2, \ldots, a^{r-1}\}$ of $\mathbb{Z}_N^*$ of order r, and since the order of a subgroup divides the order of the group (Lagrange), $\phi(N) = kr$ for some integer k.

However, this choice is not available in practice: computing $\phi(N)$ is as hard as factoring N itself, which is the very problem we are trying to solve. Shor's algorithm therefore works instead over $\mathbb{Z}_{2^m}$ with $2^m \geq N^2$, a register size that can be chosen without knowing anything about N. Then r generally does *not* divide 2^m, f is only approximately periodic on the register, and the problem is only approximately an instance of the HSP. The price is paid in the classical post-processing, which recovers r from the measured outcome using continued fractions. The exact HSP picture below should therefore be read as the idealised version of Shor's algorithm.

- **Function f:** $f : \mathbb{Z}_M \to \mathbb{Z}_N$ defined by $f(x) = a^x \pmod N$.
- **Hidden Subgroup H:** The function is constant on sets $\{x, x + r, x + 2r, \ldots\}$. This corresponds to the cosets of the subgroup $H = r\mathbb{Z}_M = \{0, r, 2r, \ldots, (k-1)r\}$, where $k = M/r$. This is the subgroup generated by the period r.
- **Goal:** Find the generator r of the hidden subgroup H.

2.2.4 The Standard Algorithm for Abelian HSP

There is a general and efficient quantum algorithm for solving the HSP for any finite abelian group G. It relies on the ability to perform a Quantum Fourier Transform over the group G, which generalizes the QFT over $\mathbb{Z}_{2^n}$ we saw earlier.

Algorithm Steps

The algorithm uses two quantum registers. The first register stores an element of G, and the second stores an element from the range of f. Note that we do not need G and X to be sets of "numbers", as we can always consider labels for elements. In the following for example $|g\rangle_G$ will denote the element of an orthonormal basis which we use to represent the corresponding group element, and similarly for elements of X.

1. **Initialization:** Prepare the state $|0\rangle_G |0\rangle_X$, where $|0\rangle_G$ is the identity element of the group G.
2. **Create Superposition:** Apply a transformation to the first register to create a uniform superposition over all group elements:

$$|\psi_1\rangle = \frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle_G |0\rangle_X$$

   For $G = \mathbb{Z}_N$, this is simply $\hat{F}_N |0\rangle$. For $G = (\mathbb{Z}_2)^n$, this is achieved by applying a Hadamard gate to each qubit ($H^{\otimes n}$). In general it is achieved by $\mathrm{QFT}_G |0\rangle_G$, as follows from the definition of QFT_G below.

3. **Query the Oracle:** Apply the oracle U_f for the function f:

$$|\psi_2\rangle = U_f |\psi_1\rangle = \frac{1}{\sqrt{|G|}} \sum_{g \in G} |g\rangle_G |f(g)\rangle_X = \frac{1}{\sqrt{|G|}} \sum_{t \in T} \Big(\sum_{h \in H} |t \cdot h\rangle_G \Big) |f(t)\rangle_X$$

   where $T \subseteq G$ contains exactly one representative t of each coset tH. The last equality groups the elements of G by coset, $g = t \cdot h$, and uses the promise $f(t \cdot h) = f(t)$ to factor the second register out of each coset.

4. **Measure the Second Register:** Projectively measure the second register. Suppose the outcome is some value y_0. The state of the first register collapses to a uniform superposition over all elements g that map to y_0. By the promise of the HSP, this set is precisely a coset of the hidden subgroup H. Let this coset be g_0H for some $g_0 \in G$.

$$|\psi_3\rangle = \frac{1}{\sqrt{|H|}} \sum_{h \in H} |g_0 \cdot h\rangle$$

The state of the system is now $|\psi_3\rangle_G |y_0\rangle_X$. We can discard the second register and drop the subscripts.

5. **Apply Quantum Fourier Transform:** Apply the QFT over the group G (denoted QFT_G) to the first register.

$$|\psi_4\rangle = \mathrm{QFT}_G |\psi_3\rangle = \mathrm{QFT}_G \left(\frac{1}{\sqrt{|H|}} \sum_{h \in H} |g_0 \cdot h\rangle \right)$$

6. **Measure the First Register:** Measure the first register in the computational basis. The result will be a random element from a special subgroup related to H.

Analysis of the Algorithm

The crucial part of the algorithm is understanding the effect of the QFT in Step 5. To do this, we need the concepts of group characters and dual group.

Definition 2.6 (Character of an Abelian Group)**.** *For a finite abelian group G, a **character** is a homomorphism $\chi : G \to \mathbb{C}^*$, where $\mathbb{C}^*$ is the multiplicative group of non-zero complex numbers. That is, $\chi(g_1 \cdot g_2) = \chi(g_1)\chi(g_2)$.*

Definition 2.7 (Dual Group)**.** *The set of all characters forms a group $\hat{G}$ under pointwise multiplication*

$$\chi_3 = \chi_1 \cdot \chi_2 : g \mapsto \chi_1(g)\chi_2(g) \quad \forall g \in G \tag{2.17}$$

*called the **dual group**. For any finite abelian group, the two are isomorphic $G \cong \hat{G}$.*

For $G = \mathbb{Z}_N$, the characters are indexed by $k \in \{0, \ldots, N-1\}$ and are given by $\chi_k(j) = e^{2\pi i j k/N}$. In this case, $\hat{G}$ is also $\mathbb{Z}_N$.

The QFT_G maps each basis state $|g\rangle$ to a superposition of basis states $|k\rangle$ labelled by the characters $\chi_k \in \hat{G}$:

$$\mathrm{QFT}_G |g\rangle = \frac{1}{\sqrt{|G|}} \sum_{k \in \hat{G}} \chi_k(g) |k\rangle$$

Now let's apply this to the state $|\psi_3\rangle$ from Step 4:

$$\begin{aligned} |\psi_4\rangle &= \mathrm{QFT}_G \left(\frac{1}{\sqrt{|H|}} \sum_{h \in H} |g_0 \cdot h\rangle \right) \\ &= \frac{1}{\sqrt{|H||G|}} \sum_{h \in H} \sum_{k \in \hat{G}} \chi_k(g_0 \cdot h) |k\rangle \\ &= \frac{1}{\sqrt{|H||G|}} \sum_{k \in \hat{G}} \chi_k(g_0) \left(\sum_{h \in H} \chi_k(h) \right) |k\rangle \end{aligned}$$

The probability of measuring a state $|k\rangle$ is proportional to the squared magnitude of its amplitude. Let's analyze the sum over H: $\sum_{h \in H} \chi_k(h)$.

Definition 2.8 (Orthogonal Subgroup)**.** *Given a subgroup $H \leq G$, the **orthogonal subgroup** (or annihilator) $H^\perp \leq \hat{G}$ is defined as:*

$$H^\perp = \{k \in \hat{G} \mid \chi_k(h) = 1 \text{ for all } h \in H\}$$

Lemma 2.9. *For $k \in \hat{G}$, the sum $\sum_{h\in H} \chi_k(h)$ is equal to $|H|$ if $k \in H^\perp$, and is equal to 0 if $k \notin H^\perp$.*

Proof. If $k \in H^\perp$, then $\chi_k(h) = 1$ for all $h \in H$, so the sum is clearly $|H|$. If $k \notin H^\perp$, there exists some $h' \in H$ such that $\chi_k(h') \neq 1$. Then:

$$\chi_k(h') \sum_{h\in H} \chi_k(h) = \sum_{h\in H} \chi_k(h' \cdot h) = \sum_{h''\in H} \chi_k(h'')$$

where we used the substitution $h'' = h' \cdot h$. Since H is a subgroup, this is just a permutation of the elements. Let $S = \sum_{h\in H} \chi_k(h)$. We have $\chi_k(h')S = S$, which implies $(\chi_k(h') - 1)S = 0$. Since $\chi_k(h') \neq 1$, it must be that $S = 0$. □

Using this lemma, the state $|\psi_4\rangle$ simplifies dramatically. The amplitude is non-zero only for $k \in H^\perp$:

$$|\psi_4\rangle = \frac{\sqrt{|H|}}{\sqrt{|G|}} \sum_{k\in H^\perp} \chi_k(g_0)\,|k\rangle$$

The probability of measuring a particular $k \in H^\perp$ is:

$$P(k) = \left|\frac{\sqrt{|H|}}{\sqrt{|G|}}\chi_k(g_0)\right|^2 = \frac{|H|}{|G|}|\chi_k(g_0)|^2 = \frac{|H|}{|G|} = \frac{1}{|H^\perp|}$$

The last equality comes from the fact that $|H||H^\perp| = |G|$. This means that measuring the first register yields a **uniformly random element** from the orthogonal subgroup $H^\perp$. The phase factor $\chi_k(g_0)$ depends on which coset was measured, but it disappears when we take the probability, so the measurement outcome distribution is independent of g_0.

Note that the analysis above holds for any finite abelian group G, although we only wrote the characters explicitly for $\mathbb{Z}_N$. This loses essentially nothing, thanks to the fundamental theorem of finite abelian groups: every finite abelian group is isomorphic to a direct sum of cyclic groups,

$$G \cong \mathbb{Z}_{N_1} \oplus \mathbb{Z}_{N_2} \oplus \cdots \oplus \mathbb{Z}_{N_k},$$

and its characters are products of characters of the factors, $\chi_{(k_1,\ldots,k_k)}(g_1,\ldots,g_k) = \prod_l e^{2\pi i\, k_l g_l/N_l}$. Correspondingly, QFT_G is the tensor product $\hat{F}_{N_1} \otimes \cdots \otimes \hat{F}_{N_k}$; for $G = (\mathbb{Z}_2)^n$ this is $H^{\otimes n}$. Note that not every finite abelian group is cyclic: $(\mathbb{Z}_2)^2$ is not isomorphic to $\mathbb{Z}_4$.

Classical Post-processing

Each run of the quantum algorithm gives us a random element $k \in H^\perp$. This gives us information about H. For $G = \mathbb{Z}_N$, $k \in H^\perp$ means $\chi_k(h) = e^{2\pi ikh/N} = 1$ for all $h \in H$. If H is generated by r, this means $e^{2\pi ikr/N} = 1$, which implies kr/N is an integer. For $G = (\mathbb{Z}_2)^n$, $k \in H^\perp$ means $\chi_k(h) = (-1)^{k\cdot h} = 1$ for all $h \in H$. If $H = \{0, s\}$, this means $k \cdot s \equiv 0 \pmod 2$.

By running the algorithm multiple times, we collect several random elements $k_1, k_2, \ldots, k_m$ from $H^\perp$. With enough samples — $O(\log |G|)$ suffice — these elements generate $H^\perp$ with high probability. From $H^\perp$, we can then classically compute the hidden subgroup $H = (H^\perp)^\perp$. This final step involves solving a system of linear equations (over the appropriate field or ring) and is classically efficient.

The standard algorithm for abelian HSP thus provides a general recipe for exponential speedups for a wide class of problems, unifying Deutsch's and Simon's algorithms, and — in the idealised form discussed above, up to the approximation needed to work on a register of size 2^m — Shor's period finding, under a single elegant framework.

Resources and further reading

This chapter closely follows chapters 4 and 6 of de Wolf (2023).

References for this chapter

Cleve, R., A. Ekert, C. Macchiavello, and M. Mosca (1998). "Quantum algorithms revisited". In: *Proceedings of the Royal Society of London A* 454.1969, pp. 339–354. DOI: 10.1098/rspa.1998.0164.

Deutsch, David (1985). "Quantum theory, the Church-Turing principle and the universal quantum computer". In: *Proceedings of the Royal Society of London A* 400.1818, pp. 97–117. DOI: 10.1098/rspa.1985.0070.

De Wolf, Ronald (2023). *Quantum Computing: Lecture Notes*. arXiv: 1907.09415 [quant-ph]. URL: https://arxiv.org/abs/1907.09415.

Nielsen, Michael A. and Isaac L. Chuang (2000). *Quantum Computation and Quantum Information*. Cambridge University Press. ISBN: 978-0-521-63503-5.

Simon, Daniel R. (1997). "On the power of quantum computation". In: *SIAM Journal on Computing* 26.5, pp. 1474–1483. DOI: 10.1137/S0097539796298637.

Appendices

Appendix A

Preliminaries

A.1 Scope and conventions

The accent of the courses of quantum information and quantum computing is less on physics and more on the mathematical formalism. Since students come from different masters, this appendix reviews the basic formalism and mathematical tools we will need in the course, which you have probably seen before but not necessarily in the same language. The goal is to bring everyone to the same page and start setting the stage for the results to be derived during the rest of the course.

A.2 Quantum States

What are quantum states? A quantum state is the mathematical description of the information available about the system.

How do you represent a quantum state? Usually we would represent these quantum states in terms of physical quantities, for example, as position or momentum wave functions. However in a more information-theoretic setting, a quantum state is represented as a vector. These vectors belong to an 'abstract' vector space called the Hilbert space, this space is usually denoted with the letter $\mathcal{H}$. The Hilbert space is both complex and either finite or infinite dimensional, and allows for inner products. The finite dimensional Hilbert space allows for more discrete-like measurement outcomes but the infinite-dimensional space can have all kinds of outcomes, both discrete and continuous.

A.2.1 Kets and Bras

A generic quantum state is represented in terms of kets and bras. A 'ket' is denoted as $|\psi\rangle$, and it is a column vector; its conjugate transpose (equivalently its Hermitian adjoint, $\langle\psi| = |\psi\rangle^\dagger$) is called a 'bra' and it is a row vector. More technically this 'bra' is an element of the dual space, and so then we can think of it as a row vector. For the most part, we will limit ourselves to the finite-dimensional Hilbert space case, and we will also work in a 2^n-dimensional space. So then the kets and bras would look like this

$$|\psi\rangle = \begin{pmatrix} a \\ b \end{pmatrix}, \langle\psi| = \begin{pmatrix} a^* & b^* \end{pmatrix} \quad \text{and} \quad a, b \in \mathbb{C}, \tag{A.1}$$

this type of notation is either called bra-ket notation or Dirac notation in honour of Paul Dirac who introduced it. Dirac notation itself is just a way of writing vectors, and applies to any vector, normalized or not; it is the vectors that represent physical *states* which must in addition be normalized, as made precise by the normalization postulate below.

A.2.2 Inner Product

The inner product of the state $|\psi\rangle$ with itself is written $\langle\psi|\psi\rangle$. This is a Hermitian product: we multiply the conjugate transpose of the vector on the left with the vector on the right,

$$\langle\psi|\psi\rangle = \begin{pmatrix} a^* & b^* \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = a^*a + b^*b = |a|^2 + |b|^2, \tag{A.2}$$

which is a non-negative real number, and equals 1 precisely when $|\psi\rangle$ is normalized. For two different states $|\psi\rangle = \begin{pmatrix} a \\ b \end{pmatrix}$ and $|\phi\rangle = \begin{pmatrix} c \\ d \end{pmatrix}$, with $a, b, c, d \in \mathbb{C}$, the same rule gives

$$\langle\phi|\psi\rangle = \begin{pmatrix} c^* & d^* \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = c^*a + d^*b. \tag{A.3}$$

Two states are said to be **orthogonal** when this number vanishes. Note that the inner product is not symmetric but *conjugate* symmetric, $\langle\phi|\psi\rangle = \langle\psi|\phi\rangle^*$, so the order matters as soon as the entries are complex.

Example A.1. Take $|\psi\rangle = |+\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and $|\phi\rangle = |-\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}$. Then

$$\langle -|+\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \end{pmatrix} \cdot \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{2}(1 - 1) = 0, \tag{A.4}$$

so $|+\rangle$ and $|-\rangle$ are orthogonal, while each is normalized: $\langle +|+\rangle = \frac{1}{2}(1 + 1) = 1$.

Very General Notation

A more general way of writing the quantum state is to write it as a sum of linearly independent basis vectors

$$|\psi\rangle = \sum_{j=0}^{d-1} c_j \, |j\rangle \,, \tag{A.5}$$

the number of vectors forming the basis is governed by the dimension d of the Hilbert space. The basis $\{|j\rangle\}$ is orthonormal, meaning that

$$\langle j|k\rangle = \delta_{jk}, \tag{A.6}$$

where the Kronecker delta $\delta_{jk} = 1$ if $j = k$, and $\delta_{jk} = 0$ if $j \neq k$. We have indexed the basis from 0, so $j = 0, 1, 2, \ldots, d - 1$: this is the standard convention in quantum computation, and it is the one used throughout these notes, because for $d = 2^n$ the label j is then exactly the n-bit string stored in the register. When we work in the infinite-dimensional case, the sum in equation (A.5) would be replaced with an integral.

Normalization Postulate

The pure state of a system is a unit vector in $\mathcal{H}$, so this means that $\langle\psi|\psi\rangle = 1$, and as we will see shortly this allows us to use the quantum states to compute probabilities of measurement outcomes.

A.2.3 Qubits

We mentioned earlier that we are restricting ourselves to a finite-dimensional Hilbert space, and for most of this appendix we firmly place ourselves in the two-dimensional Hilbert space, so that means that the dimension $d = 2$. We do this because in a classical information setting, states are represented as bits (because they're binary) and they can be either 0 or 1. In a quantum information setting, states are represented as quantum bits or qubits, and they can be either 0 or 1 or in combinations of

both (superposition principle). So in other words a bit $b \in \{0,1\}$ is encoded into a qubit basis state $|b\rangle \in \{|0\rangle, |1\rangle\}$. The Hilbert space $\mathcal{H} \simeq \mathbb{C}^2$. The computational basis mentioned above is written as follows

$$|0\rangle = \begin{pmatrix}1\\0\end{pmatrix} \quad \text{and} \quad |1\rangle = \begin{pmatrix}0\\1\end{pmatrix}, \tag{A.7}$$

and all quantum states are written as

$$|\psi\rangle = \psi_0 |0\rangle + \psi_1 |1\rangle, \tag{A.8}$$

where $\psi_0 = \langle 0|\psi\rangle$, and $\psi_1 = \langle 1|\psi\rangle$, and according to the normalization postulate $\langle\psi|\psi\rangle = 1$, so that means that $|\psi_0|^2 + |\psi_1|^2 = 1$. The two numbers ψ_0, ψ_1 are called the **amplitudes** of the state; Chapter 1 writes the same state as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, so $\alpha \equiv \psi_0$ and $\beta \equiv \psi_1$.

Qudits

We could also choose to work in higher dimensions, for example when $d = 3$, we get qutrits, and a classical trit would be encoded in $\{|0\rangle, |1\rangle, |2\rangle\}$. Ququarts and ququints have dimension $d = 4$ and $d = 5$, respectively, and so their basis states would be labelled $j = 0, 1, 2, 3$ and $j = 0, 1, 2, 3, 4$. The general name for a d-level quantum system is a **qudit**. Note that a qudit is not a collection of bits: it is a single system whose state space has dimension d, and only for $d = 2^n$ can it be read as a register of n qubits.

A.3 The Born Rule

This is a rule to assign probabilities to different outcomes of a measurement. Given a state $|\psi\rangle$ and a projective measurement whose outcomes correspond to the elements $|x\rangle$ of an orthonormal basis, the Born rule gives the probability that the measurement returns the outcome x, that is, the probability that $|\psi\rangle$ collapses onto $|x\rangle$:

$$P(x) = |\langle x|\psi\rangle|^2. \tag{A.9}$$

Because the $|x\rangle$ form a basis and $|\psi\rangle$ is normalized, these numbers automatically sum to one, $\sum_x P(x) = 1$, so they do define a probability distribution. This is the simplest form of the Born rule; the general one, which also covers measurements that are not projections onto a basis, is given in Section A.4 below.

Example A.2. Measure the state $|\psi\rangle = |+\rangle$ in the basis $\{|+\rangle, |-\rangle\}$. The probability of the outcome $+$ is

$$\begin{aligned} P(+) = |\langle +|\psi\rangle|^2 &= \left|\tfrac{1}{\sqrt{2}}(\langle 0| + \langle 1|)\,\tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right|^2 \\ &= \left|\tfrac{1}{2}(\langle 0|0\rangle + \langle 0|1\rangle + \langle 1|0\rangle + \langle 1|1\rangle)\right|^2 \\ &= \left|\tfrac{1}{2}(1 + 0 + 0 + 1)\right|^2 = 1, \end{aligned} \tag{A.10}$$

and that of the outcome $-$ is

$$\begin{aligned} P(-) = |\langle -|\psi\rangle|^2 &= \left|\tfrac{1}{\sqrt{2}}(\langle 0| - \langle 1|)\,\tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right|^2 \\ &= \left|\tfrac{1}{2}(\langle 0|0\rangle + \langle 0|1\rangle - \langle 1|0\rangle - \langle 1|1\rangle)\right|^2 \\ &= \left|\tfrac{1}{2}(1 + 0 - 0 - 1)\right|^2 = 0. \end{aligned} \tag{A.11}$$

The outcome is certain, as it must be: $|+\rangle$ is an eigenstate of this measurement. Measuring the same state in the computational basis instead would give $P(0) = P(1) = 1/2$.

A.4 General Measurements: POVMs

The Born rule above was stated for a **projective** measurement: the outcomes correspond to the elements of an orthonormal basis, and the operator associated with outcome m is the projector $P_m = |m\rangle\langle m|$. These projectors satisfy

$$P_m P_{m'} = \delta_{mm'} P_m, \qquad \sum_m P_m = \mathbb{I}. \tag{A.12}$$

This is not the most general measurement allowed by quantum mechanics. The general notion is a **POVM**, short for *positive operator-valued measure*: a collection $\{E_m\}$ of operators, one per outcome, such that

$$E_m \geq 0 \quad \text{for every } m, \qquad \sum_m E_m = \mathbb{I}, \tag{A.13}$$

with the probability of outcome m on a state ρ given by

$$p(m) = \mathrm{Tr}(E_m \rho). \tag{A.14}$$

Positivity guarantees $p(m) \geq 0$ and completeness guarantees $\sum_m p(m) = 1$, so these are exactly the two conditions needed for the numbers $p(m)$ to be probabilities — nothing more is required. In particular the E_m need be neither orthogonal nor idempotent, and there may be more outcomes than the dimension of the space, which is impossible for a projective measurement.

Projective measurement is recovered as the special case $E_m = P_m$; for a pure state $\rho = |\psi\rangle\langle\psi|$ equation (A.14) then reduces to

$$p(x) = \mathrm{Tr}(|x\rangle\langle x|\,|\psi\rangle\langle\psi|) = |\langle x|\psi\rangle|^2, \tag{A.15}$$

which is the form used in the previous section. We will verify this identity explicitly in Example A.6 below. Throughout these notes, and in Chapter 1, E_m always denotes the POVM element associated with outcome m.

A.5 Density Matrices — Pure States

If we consider a pure state $|\psi\rangle$, then its density matrix usually denoted with the Greek letter ρ would be given by $\rho = |\psi\rangle\langle\psi|$. This is shown as

$$\rho = |\psi\rangle\langle\psi| = \begin{pmatrix}\psi_0\\ \psi_1\end{pmatrix}\begin{pmatrix}\psi_0^* & \psi_1^*\end{pmatrix} = \begin{pmatrix}|\psi_0|^2 & \psi_0\psi_1^*\\ \psi_0^*\psi_1 & |\psi_1|^2\end{pmatrix}, \tag{A.16}$$

and this is a new way of writing states, and it is also called a density operator. Density matrices of the form $|\psi\rangle\langle\psi|$ represent pure states, and this means that perfect knowledge of the system is available. Sometimes we have classical uncertainty, which we can represent by extending the definition of density matrices. Axiomatically, a density matrix is any operator satisfying:

1. $\rho = \rho^\dagger$. All density matrices are Hermitian.
2. $\mathrm{Tr}\,\rho = 1$. All density matrices are normalized.
3. $\rho \geq 0$. All density matrices are positive semi-definite.

(The first condition is in fact implied by the third, but it is customary to state both.) These three conditions define *all* density matrices, mixed ones included. Pure states are singled out by one further property, which is a criterion rather than an axiom:

- $\rho^2 = \rho$, equivalently $\mathrm{Tr}\,\rho^2 = 1$. A density matrix is pure if and only if it is a rank-one projector.

Exercise

Find the density matrix for the state $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Is it a pure state? The solution is shown below.

Example A.3. The following example will demonstrate that a pure density matrix satisfies all of the conditions written above. We work with the state $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$

$$\begin{aligned}\rho = |\psi\rangle\langle\psi| &= \left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right)\left(\frac{1}{\sqrt{2}}(\langle 0| + \langle 1|)\right) \\ &= \frac{1}{2}\left(|0\rangle\langle 0| + |0\rangle\langle 1| + |1\rangle\langle 0| + |1\rangle\langle 1|\right) \\ &= \frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix},\end{aligned} \tag{A.17}$$

so now we can test for Hermiticity

$$\rho^\dagger = \frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 0| + |0\rangle\langle 1| + |1\rangle\langle 1|) = \frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix} = \rho \tag{A.18}$$

and as we can see the density matrix is Hermitian. We can also verify the trace of the matrix easily just by summing up the diagonal entries of the matrix

$$\begin{aligned}\operatorname{Tr}\rho &= \operatorname{Tr}\begin{pmatrix}1/2 & 1/2\\ 1/2 & 1/2\end{pmatrix} \\ &= \frac{1}{2} + \frac{1}{2} = 1.\end{aligned} \tag{A.19}$$

A square matrix ρ is positive semi-definite (PSD) if $\langle\phi|\,\rho\,|\phi\rangle$ is a non-negative real number for every vector $|\phi\rangle$. Equivalently, ρ is Hermitian and all of its eigenvalues are non-negative. Here the first criterion is immediate, because $\rho = |\psi\rangle\langle\psi|$ gives

$$\langle\phi|\,\rho\,|\phi\rangle = \langle\phi|\psi\rangle\,\langle\psi|\phi\rangle = |\langle\phi|\psi\rangle|^2 \geq 0 \tag{A.20}$$

for every $|\phi\rangle$; for a quick check on a 2×2 Hermitian matrix it is instead enough to compute its trace and its determinant: the matrix is PSD if and only if *both* are non-negative, $\operatorname{Tr}\rho \geq 0$ and $\det\rho \geq 0$. (The determinant alone is not enough: $\operatorname{diag}(-1,-1)$ has $\det = 1$ but is negative definite.) Finally we can also check that it is a projector

$$\rho^2 = \frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix}\cdot\frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix} = \frac{1}{4}\begin{pmatrix}2 & 2\\ 2 & 2\end{pmatrix} = \frac{1}{2}\begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix} = \rho \tag{A.21}$$

Example A.4 (Born Rule)**.** If we have a quantum state ρ and a measurement with an outcome described by the operator E_m, the probability of that outcome is given by the Born rule in the form (A.14),

$$p(m) = \operatorname{Tr}(E_m\rho), \tag{A.22}$$

Take the state $\rho = |+\rangle\langle+|$ and the projective measurement in the computational basis, so that $E_0 = |0\rangle\langle 0|$ and $E_1 = |1\rangle\langle 1|$. What we are doing is computing the trace of a product of operators: the trace of a square matrix is the sum of its diagonal entries, so we first multiply the two matrices and then add the components along the diagonal.

$$\begin{aligned}p(0) = \operatorname{Tr}(E_0\rho) &= \operatorname{Tr}(|0\rangle\langle 0|\,|+\rangle\langle+|) \\ &= \operatorname{Tr}\left(\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix}\tfrac{1}{2}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}\right) \\ &= \operatorname{Tr}\left(\tfrac{1}{2}\begin{bmatrix}1 & 1\\ 0 & 0\end{bmatrix}\right) \\ &= \frac{1}{2} + 0 = \frac{1}{2}.\end{aligned} \tag{A.23}$$

Example A.5. What if we wanted to do the same thing that we saw in the previous example, but instead we just wanted to work in the operator notation (or Dirac notation), so when both operators are projectors of the kind $M = |\phi\rangle\langle\phi|$ and $\rho = |\psi\rangle\langle\psi|$, we don't have to work with matrices explicitly, and instead we can exploit the properties of the trace:

$$\mathrm{Tr}(cAB) = c\,\mathrm{Tr}(BA), \tag{A.24}$$

so now the same calculation would be

$$\begin{aligned} p = \mathrm{Tr}(|\phi\rangle\langle\phi|\,|\psi\rangle\langle\psi|) &= \mathrm{Tr}(\langle\phi|\psi\rangle\,|\phi\rangle\langle\psi|) \\ &= \langle\phi|\psi\rangle\,\mathrm{Tr}(|\phi\rangle\langle\psi|) \\ &= \langle\phi|\psi\rangle\,\langle\psi|\phi\rangle = |\,\langle\phi|\psi\rangle\,|^2, \end{aligned} \tag{A.25}$$

and if we do this with our above example, we get $|\langle 0|+\rangle|^2$, which we have already calculated in a previous section to be $1/2$.

It is worth isolating the identity that made this work, since we will use it repeatedly. For a rank-one operator $A = |u\rangle\langle v|$,

$$\begin{aligned} \mathrm{Tr}(A) &= \sum_i \langle i|\,A\,|i\rangle \\ &= \sum_i \langle i|\,(|u\rangle\langle v|)\,|i\rangle \\ &= \sum_i \langle i|u\rangle\,\langle v|i\rangle \\ &= \langle v|u\rangle \end{aligned} \tag{A.26}$$

and this is because $\mathbb{I} = \sum_i |i\rangle\langle i|$, and $\langle v|u\rangle = \langle v|\,\mathbb{I}\,|u\rangle = \langle v|\,(\sum_i |i\rangle\langle i|)\,|u\rangle = \sum_i \langle i|u\rangle\,\langle v|i\rangle$.

Example A.6. The same probability can be obtained without ever writing a matrix. Recall that the expectation value of an operator A in the state ρ is $\langle A\rangle = \mathrm{Tr}(A\rho)$; for a projector this expectation value *is* the probability of the corresponding outcome, so

$$p(0) = \mathrm{Tr}(E_0\rho) = \mathrm{Tr}(|0\rangle\langle 0|\,|+\rangle\langle +|) = \langle E_0\rangle\,, \tag{A.27}$$

the first step would be to find $E_0\rho$, so

$$\begin{aligned} E_0\rho &= |0\rangle\langle 0|\,|+\rangle\langle +| \\ &= (|0\rangle\langle 0|)\left(\tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right)\langle +| \\ &= |0\rangle\cdot\left(\tfrac{1}{\sqrt{2}}\,\langle 0|0\rangle + \tfrac{1}{\sqrt{2}}\,\langle 0|1\rangle\right)\cdot\langle +| \\ &= |0\rangle\cdot\left(\tfrac{1}{\sqrt{2}}\cdot 1 + 0\right)\cdot\langle +| \\ &= \tfrac{1}{\sqrt{2}}\,|0\rangle\,\langle +|\,, \end{aligned} \tag{A.28}$$

and now we can compute the trace

$$\begin{aligned} \mathrm{Tr}(E_0\rho) = \mathrm{Tr}\Big(\tfrac{1}{\sqrt{2}}\,|0\rangle\,\langle +|\Big) &= \tfrac{1}{\sqrt{2}}\,\mathrm{Tr}(|0\rangle\,\langle +|) \\ &= \tfrac{1}{\sqrt{2}}\left[\sum_i \langle i|\,|0\rangle\langle +|\,|i\rangle\right] \\ &= \tfrac{1}{\sqrt{2}}\,[\langle 0|0\rangle\,\langle +|0\rangle + \langle 1|0\rangle\,\langle +|1\rangle] \\ &= \tfrac{1}{\sqrt{2}}\left[\langle 0|0\rangle\left(\tfrac{1}{\sqrt{2}}\,\langle 0|0\rangle + \tfrac{1}{\sqrt{2}}\,\langle 0|1\rangle\right) + 0\cdot\langle +|1\rangle\right] \\ &= \frac{1}{\sqrt{2}}\left[(1\cdot(\tfrac{1}{\sqrt{2}} + 0))\right] \\ &= \frac{1}{2}, \end{aligned} \tag{A.29}$$

A shortcut is to recognize that for a rank-one operator $\mathrm{Tr}(|u\rangle\langle v|) = \langle v|u\rangle$, as shown in the previous example, so here $\mathrm{Tr}(|0\rangle\langle +|) = \langle +|0\rangle = 1/\sqrt{2}$ directly. Mind the order: it is $\langle v|u\rangle$, with the bra of the operator on the left.

Exercise

Compute the trace of $|+\rangle\langle +|$ in the $|+\rangle$ basis.

Solution. When working in the $|+\rangle$ basis, the basis vectors i are $i = \{|+\rangle, |-\rangle\}$, and so the trace would be as follows

$$\begin{aligned}\mathrm{Tr}(|+\rangle\langle +|) &= \sum_i \langle i|\,|+\rangle\langle +|\,|i\rangle \\ &= \langle +|+\rangle\,\langle +|+\rangle + \langle -|+\rangle\,\langle +|-\rangle \\ &= 1\cdot 1 + 0\cdot 0 \\ &= 1\end{aligned} \tag{A.30}$$

A.6 Density Matrices — Mixed States

What are mixed states? Thus far we were working with states that we shall now call pure states; systems described by a pure state are those whose state is given by a single vector. The other kind of state is a mixed state, and states like these are written as statistical mixtures of pure states (statistical ensemble). As we'll see, these states cannot be described by just one single ket, instead they are probabilistic mixtures of many kets.

Example A.7. We start with a mixture of $|0\rangle$ and $|1\rangle$ so

$$\rho_{\mathrm{mix}} = \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| = \frac{1}{2}\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} = \frac{1}{2}\mathbb{I}, \tag{A.31}$$

More generally, any mixture of the two computational basis states has the form

$$\rho_{\mathrm{diag}} = p\,|0\rangle\langle 0| + (1-p)\,|1\rangle\langle 1|\,, \tag{A.32}$$

where p is some probability such that $0 \le p \le 1$. Be careful here: (A.32) is *not* the most general qubit state. It is the most general state that is *diagonal in the computational basis*, and it has no off-diagonal entries. A general mixed state is a mixture $\rho = \sum_i p_i\,|\psi_i\rangle\langle\psi_i|$ of arbitrary pure states, and can perfectly well have non-zero off-diagonal entries — $|+\rangle\langle +|$ itself is an example. The genuinely general form for a single qubit is the Bloch decomposition given in Section A.7, $\rho = \frac{1}{2}(\mathbb{I} + \vec{r}\cdot\vec{\sigma})$ with $|\vec{r}| \le 1$, of which (A.32) is the special case $r_x = r_y = 0$.

The maximally mixed state can also be written in different bases: in the diagonal basis (also called the Hadamard, Fourier or X basis) it reads $\rho_{\mathrm{mm}} = \frac{1}{2}|+\rangle\langle +| + \frac{1}{2}|-\rangle\langle -|$, and in the Y basis $\rho_{\mathrm{mm}} = \frac{1}{2}|+i\rangle\langle +i| + \frac{1}{2}|-i\rangle\langle -i|$, where the subscript mm stands for maximally mixed. That the same operator $\frac{1}{2}\mathbb{I}$ arises from three different ensembles is the first sign of an important fact: the density matrix, not the ensemble that produced it, is what is physically meaningful. No measurement whatsoever can distinguish these three preparations.

Exercise

Show that this is not equal to $\rho = |+\rangle\langle +|$.

Solution. The $\rho = |+\rangle\langle +|$ state is defined to be

$$\rho = |+\rangle\langle +| = \frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 0| + |0\rangle\langle 1| + |1\rangle\langle 1|), \tag{A.33}$$

and we can see that this density matrix contains the coherence terms (A.17), whereas the state $\rho' = \frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 1|)$, does not contain the off-diagonal terms (A.31), in fact the off-diagonal components are zero, so it is maximally mixed.

Exercise

Come up with more examples of mixed states.

Solution. Plugging arbitrary values of p into equation (A.32) gives mixed states diagonal in the computational basis, for example $\rho = 0.3\,|0\rangle\langle 0| + 0.7\,|1\rangle\langle 1|$. For a mixed state that is *not* diagonal, mix states from different bases, for example $\rho = \frac{1}{2}\,|0\rangle\langle 0| + \frac{1}{2}\,|+\rangle\langle +|$, whose Bloch vector is $\vec{r} = (1/2, 0, 1/2)$ with $|\vec{r}| = 1/\sqrt{2} < 1$.

A.7 The Pauli Matrices

There are three Pauli matrices — X, Y and Z — and they all have these three main properties:

- They are all traceless.
- They are all Hermitian.
- They are all unitary.

The Pauli X matrix is also called the Pauli 1 matrix or σ_x:

$$X = \sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = 0 \cdot |0\rangle\langle 0| + 1 \cdot |0\rangle\langle 1| + 1 \cdot |1\rangle\langle 0| + 0 \cdot |1\rangle\langle 1| \tag{A.34}$$

The Pauli Y matrix is also called the Pauli 2 matrix:

$$Y = \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} = -i \cdot |0\rangle\,\langle 1| + i \cdot |1\rangle\,\langle 0| \tag{A.35}$$

The third Pauli matrix is the Pauli Z matrix:

$$Z = \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = |0\rangle\langle 0| - |1\rangle\langle 1| \tag{A.36}$$

The identity matrix $\mathbb{I}$ is an honorary Pauli matrix, and it's referred to as the Pauli 0 matrix:

$$\mathbb{I} = \sigma_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = |0\rangle\langle 0| + |1\rangle\langle 1| \tag{A.37}$$

Example A.8. In this example we will see how these Pauli matrices satisfy the three conditions mentioned above. We will work with the Pauli-Y matrix, but it might be helpful to repeat this exercise with the other two Pauli matrices as well. To verify whether the matrix is traceless, we can just add the values along the diagonal, and this shows that it has a trace of zero:

$$\operatorname{Tr}\sigma_y = \operatorname{Tr}\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} = 0 + 0 = 0, \tag{A.38}$$

Is the matrix Hermitian? So is $\sigma_y^\dagger = \sigma_y$

$$\sigma_y^\dagger = \begin{pmatrix} 0 & i \\ -i & 0 \end{pmatrix}^T = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} = \sigma_y \tag{A.39}$$

Is the matrix unitary? In other words, is its inverse equal to its adjoint? Inverting a 2×2 matrix with the adjugate formula, and using $\det\sigma_y = 0 \cdot 0 - (-i)(i) = -1$,

$$\sigma_y^{-1} = \frac{1}{\det\sigma_y}\begin{pmatrix} 0 & i \\ -i & 0 \end{pmatrix} = \frac{1}{-1}\begin{pmatrix} 0 & i \\ -i & 0 \end{pmatrix} = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} = \sigma_y. \tag{A.40}$$

Combining the last two results, $\sigma_y^{-1} = \sigma_y = \sigma_y^\dagger$, which is exactly the statement that σ_y is unitary. The same three checks work for σ_x and σ_z. Note in passing that all three Paulis square to the identity, $\sigma_i^2 = \mathbb{I}$, which is another way of seeing both the unitarity and the fact that their eigenvalues are ± 1.

Example A.9. The four Pauli operators (I, X, Y and Z) form a basis of the real vector space of 2×2 Hermitian operators: any such operator can be written as a real linear combination of these matrices. One Hermitian operator we have just learned about is the density matrix, and we will now see how an arbitrary single-qubit density matrix can be written as a combination of Pauli operators

$$\rho = \frac{1}{2}\left(\mathbb{I} + r_x X + r_y Y + r_z Z\right) = \frac{1}{2}\left(\mathbb{I} + \vec{r}\cdot\vec{\sigma}\right), \tag{A.41}$$

where $\vec{\sigma} = (X, Y, Z)$ and $\vec{r} = (r_x, r_y, r_z)$ is called the Bloch vector; it is a three-dimensional real vector lying in the unit ball, $|\vec{r}| \leq 1$. The coefficient of $\mathbb{I}$ is fixed to $1/2$ by the requirement $\operatorname{Tr}\rho = 1$, and the components are recovered as $r_i = \operatorname{Tr}(\sigma_i \rho)$, which follows from $\operatorname{Tr}(\sigma_i\sigma_j) = 2\delta_{ij}$ and $\operatorname{Tr}\sigma_i = 0$.

Exercise

Convert this matrix to Dirac notation:

$$A = \begin{pmatrix} 2 & 1+i \\ 1-i & 3 \end{pmatrix} \tag{A.42}$$

Solution. Each entry A_{jk} multiplies $|j\rangle\langle k|$, so reading the matrix off row by row,

$$A = 2\,|0\rangle\langle 0| + (1+i)\,|0\rangle\langle 1| + (1-i)\,|1\rangle\langle 0| + 3\,|1\rangle\langle 1|\,. \tag{A.43}$$

Note that $A = A^\dagger$: the diagonal entries are real and the off-diagonal ones are complex conjugates of each other, so A is Hermitian and the next example applies to it.

Example A.10. Any Hermitian operator can be written in the Pauli basis that was just shown above; here we show how to obtain the coefficients $\alpha_0, \alpha_x, \alpha_y$ and α_z for the same matrix as in the exercise,

$$A = \begin{pmatrix} 2 & 1+i \\ 1-i & 3 \end{pmatrix}, \tag{A.44}$$

which we have already observed is Hermitian. Writing $A = \alpha_0\mathbb{I} + \alpha_x X + \alpha_y Y + \alpha_z Z$, the individual coefficients are given by $\alpha_\mu = \frac{1}{2}\operatorname{Tr}(\sigma_\mu A)$, a consequence of $\operatorname{Tr}(\sigma_\mu\sigma_\nu) = 2\delta_{\mu\nu}$. We will use the fact that for a rank-one operator $|u\rangle\langle v|$ and any operator A, $\operatorname{Tr}(|u\rangle\langle v|\,A) = \langle v|\,A\,|u\rangle$

$$\begin{aligned} \alpha_0 = \frac{1}{2}\operatorname{Tr}(\mathbb{I}A) &= \frac{1}{2}\operatorname{Tr}\left((|0\rangle\langle 0| + |1\rangle\langle 1|)A\right) \\ &= \frac{1}{2}(\langle 0|\,A\,|0\rangle + \langle 1|\,A\,|1\rangle) = \frac{1}{2}(2+3) = \frac{5}{2} \end{aligned} \tag{A.45}$$

$$\begin{aligned} \alpha_x = \frac{1}{2}\operatorname{Tr}(XA) &= \frac{1}{2}\operatorname{Tr}\left((|0\rangle\langle 1| + |1\rangle\langle 0|)A\right) \\ &= \frac{1}{2}(\langle 1|\,A\,|0\rangle + \langle 0|\,A\,|1\rangle) = \frac{1}{2}\left((1-i) + (1+i)\right) = 1 \end{aligned} \tag{A.46}$$

$$\begin{aligned} \alpha_y = \frac{1}{2}\operatorname{Tr}(YA) &= \frac{1}{2}\operatorname{Tr}\left((-i\,|0\rangle\langle 1| + i\,|1\rangle\langle 0|)A\right) \\ &= \frac{1}{2}(-i\,\langle 1|\,A\,|0\rangle + i\,\langle 0|\,A\,|1\rangle) \\ &= \frac{1}{2}(-i(1-i) + i(1+i)) = -\frac{2}{2} = -1 \end{aligned} \tag{A.47}$$

$$\begin{aligned}\alpha_z &= \frac{1}{2}\operatorname{Tr}(ZA) \\ &= \frac{1}{2}\operatorname{Tr}\left(\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}\cdot\begin{pmatrix}2 & 1+i\\ 1-i & 3\end{pmatrix}\right) \\ &= \frac{1}{2}\operatorname{Tr}\left(\begin{pmatrix}2+0 & 1+i\\ -1+i & -3\end{pmatrix}\right) = \frac{1}{2}(2+(-3)) = -\frac{1}{2}\end{aligned} \tag{A.48}$$

As a check, substituting back,

$$\tfrac{5}{2}\mathbb{I} + X - Y - \tfrac{1}{2}Z = \begin{pmatrix}\frac{5}{2}-\frac{1}{2} & 1+i\\ 1-i & \frac{5}{2}+\frac{1}{2}\end{pmatrix} = \begin{pmatrix}2 & 1+i\\ 1-i & 3\end{pmatrix} = A, \tag{A.49}$$

where the off-diagonal entries come from $\alpha_x X + \alpha_y Y = X - Y$, whose $(0,1)$ entry is $1-(-i) = 1+i$. Note that A is not a density matrix: its trace is 5, not 1.

A.8 The Bloch Sphere

The Bloch sphere is a geometrical representation of a qubit. Any qubit can be written as

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle, \tag{A.50}$$

where $\varphi \in [0, 2\pi)$ and $\theta \in [0, \pi]$, and so a pure qubit can be specified by just two real numbers — the four real parameters of ψ_0, ψ_1 minus one for normalization and one for the global phase, which is unobservable. The components of the Bloch vector are the expectation values of the Pauli operators, $r_i = \operatorname{Tr}(\sigma_i\rho) = \langle\sigma_i\rangle$. The Bloch sphere is a unit sphere (so radius = 1): pure states lie on its surface, while mixed states lie strictly inside it, so that the set of all qubit states is the unit *ball*. The Bloch vector is defined in terms of the polar angle θ and the azimuthal angle φ

$$\vec{r} = \begin{pmatrix}\sin\theta\cos\varphi\\ \sin\theta\sin\varphi\\ \cos\theta\end{pmatrix} \tag{A.51}$$

Specific angles point to points on the north and south poles, and then other angles point to antipodal points on the equator. States that lie on opposite ends of each other are orthogonal to each other. These points also represent the Pauli eigenstates. The Z basis is the computational basis and it is represented with $|0\rangle$ and $|1\rangle$. The X basis, which Chapter 1 calls the diagonal basis (it also goes by Hadamard or Fourier basis), is represented with $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, and the Y basis is represented with $|+i\rangle = 1/\sqrt{2}(|0\rangle + i|1\rangle)$ and $|-i\rangle = 1/\sqrt{2}(|0\rangle - i|1\rangle)$

Exercise

Figure out how to go from the $|+\rangle$ state to the $|-\rangle$ state, and also figure out how to go from the $|0\rangle$ state to the $|1\rangle$ state. **Hint**: You multiply Pauli operators to these states.

Solution.

- Bit flip: $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$.
- Phase flip: $Z|+\rangle = |-\rangle$ and $Z|-\rangle = |+\rangle$.
- The Y gate is equivalent to a bit and phase flip, $Y = iXZ$.

Exercise

What are the Bloch vectors for the $|0\rangle, |1\rangle, |+\rangle, |-\rangle, |+i\rangle$ and $|-i\rangle$ states?

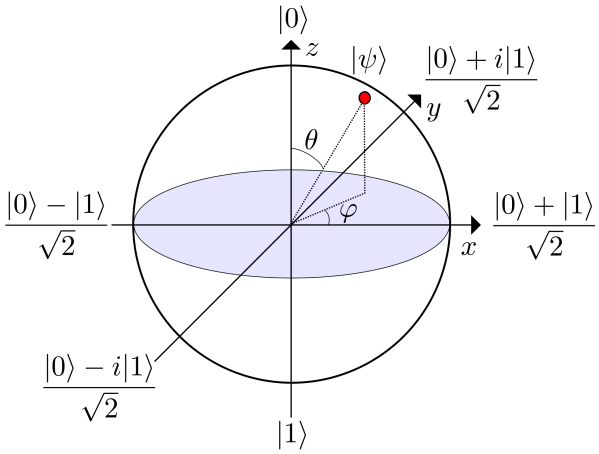


Figure A.1: Bloch sphere with Pauli-Z eigenstates $|0\rangle$ and $|1\rangle$ on the north and south poles respectively.

Solution.

- $|0\rangle : \theta = 0$ and φ arbitrary, so then the Bloch vector $\vec{r} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$
- $|1\rangle : \theta = \pi$ and φ arbitrary, so then the Bloch vector $\vec{r} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$
- $|+\rangle : \theta = \frac{\pi}{2}$ and $\varphi = 0$, so then the Bloch vector $\vec{r} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$
- $|-\rangle : \theta = \frac{\pi}{2}$ and $\varphi = \pi$, so then the Bloch vector $\vec{r} = \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}$
- $|+i\rangle : \theta = \frac{\pi}{2}$ and $\varphi = \frac{\pi}{2}$, so then the Bloch vector $\vec{r} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$
- $|-i\rangle : \theta = \frac{\pi}{2}$ and $\varphi = \frac{3\pi}{2}$, so then the Bloch vector $\vec{r} = \begin{pmatrix} 0 \\ -1 \\ 0 \end{pmatrix}$

A.9 Tensor Products

Tensor products are also known as Kronecker products and they describe multiple or multipartite states.

$$|a\rangle \otimes |b\rangle = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \otimes \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_1 b_1 \\ a_1 b_2 \\ a_2 b_1 \\ a_2 b_2 \end{bmatrix}, \tag{A.52}$$

so we can see that the dimensions multiply: the tensor product of two vectors in $\mathbb{C}^2$ lives in $\mathbb{C}^2 \otimes \mathbb{C}^2 \simeq \mathbb{C}^4$. Similarly we can see what happens if we tensor two square matrices $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$, and $B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$,

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B \\ a_{21}B & a_{22}B \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{bmatrix}, \tag{A.53}$$

and the resulting 4×4 matrix is an operator acting on $\mathbb{C}^2 \otimes \mathbb{C}^2 \simeq \mathbb{C}^4$. In general, for an $m \times m$ matrix A and an $n \times n$ matrix B, $A \otimes B$ is $mn \times mn$.

Example A.11. If you have system A in state $|1\rangle_A$ and system B in state $|0\rangle_B$, their total **bipartite** state would be $|1\rangle_A \otimes |0\rangle_B = |10\rangle_{AB}$

$$|1\rangle_A \otimes |0\rangle_B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \tag{A.54}$$

states of this form are called **separable** (or product) states: the two subsystems are uncorrelated. There exist states that cannot be written as $|\psi\rangle_A \otimes |\phi\rangle_B$ for any choice of $|\psi\rangle$ and $|\phi\rangle$, and these are precisely the **entangled** states. For pure states there is no middle ground: a pure bipartite state is either separable, and then the two subsystems are completely uncorrelated, or entangled.

Example A.12. The same rule applies to operators. Taking the tensor product of the Pauli X and Z matrices,

$$\begin{aligned} X \otimes Z &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ &= \begin{bmatrix} 0 \cdot Z & 1 \cdot Z \\ 1 \cdot Z & 0 \cdot Z \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix} \end{aligned} \tag{A.55}$$

Example A.13. One of the two factors can equally well be a density matrix:

$$\begin{aligned} |+\rangle\langle+| \otimes Z &= \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ &= \frac{1}{2} \begin{bmatrix} 1 \cdot Z & 1 \cdot Z \\ 1 \cdot Z & 1 \cdot Z \end{bmatrix} \\ &= \frac{1}{2} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & -1 & 0 & -1 \\ 1 & 0 & 1 & 0 \\ 0 & -1 & 0 & -1 \end{bmatrix} \end{aligned} \tag{A.56}$$

Exercise

What is the tensor product of $Z \otimes |+\rangle\langle+|$?

Solution.

$$\begin{aligned} Z \otimes |+\rangle\langle+| &= \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \otimes \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 \cdot \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} & 0 \cdot \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \\ 0 \cdot \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} & -1 \cdot \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \end{bmatrix} \\ &= \frac{1}{2}\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & -1 \\ 0 & 0 & -1 & -1 \end{bmatrix} \end{aligned} \tag{A.57}$$

A.10 Partial Trace

The partial trace is an operation that reduces the dimension of the Hilbert space, essentially you only want to trace out one system, while leaving the other system unchanged, and this is useful when we are interested in only a certain part of a coupled quantum system. So essentially if we have a bipartite state or a state with two systems - A and B, and we want to trace out system B and just be left with system A, then what we do is apply the identity operation on all of the basis vectors of system A, and then keep tracing over each of the basis vectors in system B.

$$\mathrm{Tr}_B(X) = \sum_i (\mathbb{I}_A \otimes \langle i|) X (\mathbb{I}_A \otimes |i\rangle) \tag{A.58}$$

Example A.14. If we have a bipartite qubit state with a computational basis $i = \{0, 1\}$, and $|\psi\rangle = |+\rangle_A |+\rangle_B$, and its density matrix is $\rho_{AB} = |\psi\rangle\langle\psi| = |{+}{+}\rangle\langle{+}{+}|$, then its partial trace over the system B would be

$$\begin{aligned} \rho_A = \mathrm{Tr}_B(\rho_{AB}) &= \sum_{i=0}^{1} (\mathbb{I}_A \otimes \langle i|) \, |{+}{+}\rangle\langle{+}{+}| \, (\mathbb{I}_A \otimes |i\rangle) \\ &= (\mathbb{I}_A \otimes \langle 0|) \, |{+}{+}\rangle\langle{+}{+}|_{AB} \, (\mathbb{I}_A \otimes |0\rangle) + (\mathbb{I}_A \otimes \langle 1|) \, |{+}{+}\rangle\langle{+}{+}|_{AB} \, (\mathbb{I}_A \otimes |1\rangle) \\ &= (\mathbb{I}_A \otimes \langle 0|_B) \, |+\rangle_A |+\rangle_B \langle+|_A \langle+|_B \, (\mathbb{I}_A \otimes |0\rangle_B) + (\mathbb{I}_A \otimes \langle 1|_B) \, |+\rangle_A |+\rangle_B \langle+|_A \langle+|_B \, (\mathbb{I}_A \otimes |1\rangle_B) \\ &= |+\rangle_A \cdot \langle 0|+\rangle_B \cdot \langle+|_A \cdot \langle+|0\rangle_B + |+\rangle_A \cdot \langle 1|+\rangle_B \cdot \langle+|_A \cdot \langle+|1\rangle_B \\ &= |+\rangle_A \cdot \tfrac{1}{\sqrt{2}}(\langle 0|0\rangle + \langle 0|1\rangle) \cdot \langle+|_A \cdot \tfrac{1}{\sqrt{2}}(\langle 0|0\rangle + \langle 1|0\rangle) \\ &\quad + |+\rangle_A \cdot \tfrac{1}{\sqrt{2}}(\langle 1|0\rangle + \langle 1|1\rangle) \cdot \langle+|_A \cdot \tfrac{1}{\sqrt{2}}(\langle 0|1\rangle + \langle 1|1\rangle) \\ &= |+\rangle_A \cdot \tfrac{1}{\sqrt{2}}(1+0) \cdot \langle+|_A \cdot \tfrac{1}{\sqrt{2}}(1+0) + |+\rangle_A \cdot \tfrac{1}{\sqrt{2}}(0+1) \cdot \langle+|_A \cdot \tfrac{1}{\sqrt{2}}(0+1) \\ &= \frac{1}{2}|+\rangle\langle+|_A + \frac{1}{2}|+\rangle\langle+|_A \\ &= |+\rangle\langle+|_A \end{aligned} \tag{A.59}$$

and so we get a pure reduced density matrix (prove as exercise). **Hint:** take the trace of the square. This is no accident: $|{+}{+}\rangle$ is a *product* state, the two qubits are uncorrelated, and discarding one therefore tells us nothing we did not already know about the other. The next example shows what happens when that is not the case.

Example A.15. This time we trace over system A with a Bell state, $|\Phi^+\rangle_{AB} = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, and so its density matrix would be

$$\begin{aligned} \rho_{AB} = \left|\Phi^+\right\rangle\left\langle\Phi^+\right| &= \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \, \frac{1}{\sqrt{2}}(\langle 00| + \langle 11|) \\ &= \frac{1}{2}(|00\rangle\langle 00| + |00\rangle\langle 11| + |11\rangle\langle 00| + |11\rangle\langle 11|), \end{aligned} \tag{A.60}$$

we carry out the same protocol that we saw earlier, except now we alter the expression for the partial trace very slightly to reflect that we are tracing out system A,

$$\begin{aligned}
\rho_B = \mathrm{Tr}_A(\rho_{AB}) &= \sum_{i=0}^{1} (\langle i| \otimes \mathbb{I}_B) \left|\Phi^+\right\rangle\!\left\langle\Phi^+\right| (|i\rangle \otimes \mathbb{I}_B) \\
&= \frac{1}{2}\Big[(\langle 0| \otimes \mathbb{I}_B) |00\rangle\langle 00| (|0\rangle \otimes \mathbb{I}_B) + (\langle 1| \otimes \mathbb{I}_B) |00\rangle\langle 00| (|1\rangle \otimes \mathbb{I}_B) \\
&\quad + (\langle 0| \otimes \mathbb{I}_B) |00\rangle\langle 11| (|0\rangle \otimes \mathbb{I}_B) + (\langle 1| \otimes \mathbb{I}_B) |00\rangle\langle 11| (|1\rangle \otimes \mathbb{I}_B) \\
&\quad + (\langle 0| \otimes \mathbb{I}_B) |11\rangle\langle 00| (|0\rangle \otimes \mathbb{I}_B) + (\langle 1| \otimes \mathbb{I}_B) |11\rangle\langle 00| (|1\rangle \otimes \mathbb{I}_B) \\
&\quad + (\langle 0| \otimes \mathbb{I}_B) |11\rangle\langle 11| (|0\rangle \otimes \mathbb{I}_B) + (\langle 1| \otimes \mathbb{I}_B) |11\rangle\langle 11| (|1\rangle \otimes \mathbb{I}_B) \Big] \\
&= \frac{1}{2} |0\rangle\langle 0|_B + \frac{1}{2} |1\rangle\langle 1|_B \\
&= \frac{1}{2}\mathbb{I}
\end{aligned} \tag{A.61}$$

Four of these eight terms vanish, namely the ones coming from the off-diagonal (coherence) terms of ρ_{AB}. For instance

$$(\langle i| \otimes \mathbb{I}_B) |00\rangle\langle 11| (|i\rangle \otimes \mathbb{I}_B) = \langle i|0\rangle \langle 1|i\rangle \; |0\rangle\langle 1|_B = 0, \tag{A.62}$$

because $\langle i|0\rangle \langle 1|i\rangle$ is $1 \cdot 0 = 0$ for $i = 0$ and $0 \cdot 1 = 0$ for $i = 1$; the same happens for $|11\rangle\langle 00|$. Only the two diagonal terms survive, each contributing $\frac{1}{2}$.

So the reduced state is the maximally mixed state (exercise: prove that it is not pure). This deserves emphasis, because it is the whole reason density matrices are needed at all. The global state $|\Phi^+\rangle$ is *pure*: we know everything there is to know about the pair. Yet the state of qubit B on its own is *maximally mixed*: we know nothing whatsoever about it, and a measurement in any basis gives a uniformly random outcome. No state vector $|\phi\rangle_B$ describes qubit B, so the density matrix is not a convenience here but a necessity. Entanglement is precisely what makes the parts less determined than the whole, and it is the reason that a subsystem of a pure state is generally mixed.

Exercise

Repeat the calculations above, tracing out the other subsystem in each case: compute Tr_A for the product state $|{+}{+}\rangle$, and Tr_B for the Bell state $|\Phi^+\rangle$. By the symmetry of both states you should find the same answers as above; verify this explicitly.

Exercise

Compute the reduced state of the first qubit for $|\psi\rangle = \cos\alpha\,|00\rangle + \sin\alpha\,|11\rangle$ as a function of α, and check that it interpolates between a pure state at $\alpha = 0$ and the maximally mixed state at $\alpha = \pi/4$. This is a first quantitative handle on "how entangled" a state is.

Resources and further reading

This appendix draws on the lecture notes of Griffiths (2014) and on Bertlmann (n.d.), as well as on Timothée Hoffreumon's lecture notes from the MAQI summer school and a set of lectures from the Freie Universität Berlin; the latter two are not publicly available and will be circulated by email.

References for this chapter

Bertlmann, Reinhold A. (n.d.). *Theoretical Physics T2: Quantum Mechanics. Lecture script, chapter 9, University of Vienna.* URL: `https://homepage.univie.ac.at/reinhold.bertlmann/pdfs/T2_Skript_Ch_9corr.pdf`.

Griffiths, Robert B. (2014). *Quantum Information Theory: Density Matrices. Lecture notes, Carnegie Mellon University*. URL: `https://quantum.phys.cmu.edu/QCQI/qitd114.pdf`.

Bibliography

Aharonov, Dorit (2003). "A Simple Proof that Toffoli and Hadamard are Quantum Universal". In: *arXiv preprint quant-ph/0301040*. URL: https://arxiv.org/abs/quant-ph/0301040.

Bertlmann, Reinhold A. (n.d.). *Theoretical Physics T2: Quantum Mechanics. Lecture script, chapter 9, University of Vienna*. URL: https://homepage.univie.ac.at/reinhold.bertlmann/pdfs/T2_Skript_Ch_9corr.pdf.

Cleve, R., A. Ekert, C. Macchiavello, and M. Mosca (1998). "Quantum algorithms revisited". In: *Proceedings of the Royal Society of London A* 454.1969, pp. 339–354. DOI: 10.1098/rspa.1998.0164.

Dawson, Christopher M. and Michael A. Nielsen (2006). "The Solovay-Kitaev algorithm". In: *Quantum Information & Computation* 6.1, pp. 81–95. URL: https://arxiv.org/abs/quant-ph/0505030.

Deutsch, David (1985). "Quantum theory, the Church-Turing principle and the universal quantum computer". In: *Proceedings of the Royal Society of London A* 400.1818, pp. 97–117. DOI: 10.1098/rspa.1985.0070.

De Wolf, Ronald (2023). *Quantum Computing: Lecture Notes*. arXiv: 1907.09415 [quant-ph]. URL: https://arxiv.org/abs/1907.09415.

Feynman, Richard P. (1982). "Simulating physics with computers". In: *International Journal of Theoretical Physics* 21.6, pp. 467–488. DOI: 10.1007/BF02650179.

Griffiths, Robert B. (2014). *Quantum Information Theory: Density Matrices. Lecture notes, Carnegie Mellon University*. URL: https://quantum.phys.cmu.edu/QCQI/qitd114.pdf.

Harrow, Aram W., Benjamin Recht, and Isaac L. Chuang (2002). "Efficient discrete approximations of quantum gates". In: *Journal of Mathematical Physics* 43.9, pp. 4445–4451. DOI: 10.1063/1.1495899. URL: https://arxiv.org/abs/quant-ph/0111031.

Kitaev, A. Yu. (1997). "Quantum computations: algorithms and error correction". In: *Russian Mathematical Surveys* 52.6, pp. 1191–1249. DOI: 10.1070/RM1997v052n06ABEH002155.

Kockum, Anton Frisk et al. (2023). *Lecture notes on quantum computing*. arXiv:2311.08445. DOI: 10.48550/arXiv.2311.08445. URL: https://arxiv.org/abs/2311.08445.

Nielsen, Michael A. and Isaac L. Chuang (2000). *Quantum Computation and Quantum Information*. Cambridge University Press. ISBN: 978-0-521-63503-5.

Shi, Yaoyun (2002). "Both Toffoli and CNOT need little help to do universal quantum computing". In: *arXiv preprint quant-ph/0205115*. URL: https://arxiv.org/abs/quant-ph/0205115.

Simon, Daniel R. (1997). "On the power of quantum computation". In: *SIAM Journal on Computing* 26.5, pp. 1474–1483. DOI: 10.1137/S0097539796298637.