

Quantum Information and Computation

Michael Walter, Ruhr University Bochum

Winter Semester 2023/24

Course Description

This course gives an introduction to quantum information and quantum computation from the perspective of theoretical computer science. We discuss the theoretical model of quantum bits and circuits, how to generalize computer science concepts to the quantum setting, how to design and analyze quantum algorithms and protocols for a variety of computational problems, and how to prove complexity theoretic lower bounds.

Acknowledgements

I would like to thank Kaniuar Bacho, Cedric Brügmann, Anurudh Peduri, and Tianwei Zhang for their help. Thanks also to Linus Mika Brandt, Sebastian Marks, Julia Oppermann, Oliver Stolz, Jurek Völp, and Luca Witt for pointing out typos. Finally, I would like to thank Ryan O'Donnell and Ronald de Wolf, whose excellent quantum computing lecture notes served as inspiration for this course (some lectures are rather directly based on theirs).

Last updated: [April 12, 2024 at 00:51](#).

Contents

| | |
|----------------------------------------------------------------------------------------------|-----------|
| Chapter 1: From classical to quantum computing | 5 |
| 1.1 Introduction | 5 |
| 1.2 Computational problems | 6 |
| 1.3 Probabilistic computing | 7 |
| 1.4 Quantum computing | 9 |
| 1.5 Outlook | 11 |
| Chapter 2: Quantum bits: states and operations | 13 |
| 2.1 States and Dirac notation | 13 |
| 2.2 Measurements and Unitaries | 17 |
| 2.3 Quantum Circuits | 19 |
| 2.4 Is all of this real? | 20 |
| 2.5 Elitzur-Vaidman quantum ‘bomb’ tester | 21 |
| 2.6 Beyond qubits | 22 |
| Chapter 3: Quantum computing with many qubits | 25 |
| 3.1 States and operations | 25 |
| 3.2 Product states and entangled states | 29 |
| 3.3 Unitaries on subsystems | 31 |
| 3.4 Measurements on subsystems | 33 |
| Chapter 4: Classical vs quantum computing and oracles | 35 |
| 4.1 What do quantum circuits and algorithms look like? | 35 |
| 4.2 From classical to reversible and quantum circuits | 36 |
| 4.3 A first quantum advantage: Deutsch’s problem and algorithm | 39 |
| 4.4 Outro: Time vs query complexity | 41 |
| Chapter 5: Simon’s quantum algorithm and classical lower bounds | 43 |
| 5.1 Classical algorithm | 43 |
| 5.2 Simon’s quantum algorithm | 44 |
| 5.3 Bonus: Classical lower bound | 46 |
| Chapter 6: Grover’s search algorithm and applications (or: SAT on a quantum computer) | 49 |
| 6.1 Search problem and classical algorithm | 49 |
| 6.2 Grover’s quantum search algorithm | 49 |
| Chapter 7: Quantum Fourier transform | 53 |
| 7.1 Fourier transform for \mathbb{Z}_N | 53 |
| 7.2 Excursion: Fast Fourier transform | 55 |
| 7.3 Quantum Fourier transform | 56 |
| Chapter 8: Shor’s quantum algorithm for discrete logs | 61 |
| 8.1 Motivation: Breaking Diffie-Hellman by discrete logs | 61 |
| 8.2 Shor’s quantum algorithm for discrete logs | 62 |

| | |
|-------------------------------------------------------------------------------|------------|
| Chapter 9: Quantum phase estimation | 65 |
| 9.1 Quantum phase estimation in the simplest setting | 65 |
| 9.2 Quantum phase estimation in the general case | 66 |
| Chapter 10: Shor’s quantum algorithm for factoring by order finding | 69 |
| 10.1 Motivation: Breaking RSA via factoring | 69 |
| 10.2 Factoring from order finding | 70 |
| 10.3 Shor’s quantum order finding algorithm | 70 |
| Chapter 11: Quantum query complexity | 73 |
| 11.1 Introduction and definitions | 73 |
| 11.2 The adversary bounds | 75 |
| Chapter 12: Quantum entanglement, superdense coding, and teleportation | 81 |
| 12.1 Bell states | 81 |
| 12.2 Superdense coding | 82 |
| 12.3 Teleportation | 83 |
| Chapter 13: No cloning and quantum money | 85 |
| 13.1 No cloning revisited | 85 |
| 13.2 Quantum money | 87 |
| 13.3 Security of Wiesner’s quantum money | 88 |
| 13.4 Bonus: Public verifiability and public-key quantum money | 89 |
| Chapter 14: Nonlocal games and the CHSH Game | 91 |
| 14.1 The CHSH game | 91 |
| 14.2 Classical strategies | 92 |
| 14.3 Quantum strategies | 92 |
| 14.4 Bonus: Rigidity of the quantum winning strategy | 97 |
| 14.5 Bonus: Non-signaling strategies | 100 |
| Chapter 15: The quantum road ahead | 103 |

Chapter 1

From classical to quantum computing

1.1 Introduction

Quantum computing is a relatively recent area of research that combines two revolutions of the past century:

- Quantum mechanics, which in the early 20th century was discovered to be the right theory to describe our world at the smallest scales. It was pioneered by the likes of Planck, Einstein, Heisenberg, Curie, Dirac, Schrödinger, etc.
- Computer science, needs no explanation to this crowd. It really set off with the invention of the digital computer (we all know Turing, von Neumann, Church, Hopper, etc), but aspects were already studied much earlier, e.g., by Babbage, Leibniz, Lovelace, etc.

Without quantum mechanics there would be no understanding of semiconductors, so quantum mechanics is surely important to understand how transistors etc. work. But as computer scientists we can usually ignore this: when we as theorists design algorithms or as practitioners implement these, we are usually very happy to think of our computers as abstract machines that employ Boolean operations to manipulate bits and bytes. Indeed, the very insight that this is possible is one of the most fundamental ideas that kick-started our field.

Quantum computing is *not* understanding the physics of semiconductors better. Instead, it asks a much more fundamental question:

What does computer science look like if we replace our familiar bits by “quantum bits”, and Boolean operations by the kind of “quantum operations” that nature allows us to apply to these?

We will learn all about quantum bits and operations in the coming weeks. But why should we ask such a question in the first place?

- *To solve problems faster:* We might be able to find certain clever *quantum algorithms* that allow future *quantum computers* to solve certain computational problems much faster than what is possible (or at least known to be possible) using ordinary computers. Some of you may already know that this is indeed the case.
- *To understand the ultimate power and limits of computation:* Since the world is really described by quantum mechanics, studying computers based on bits and bytes is simply not good enough if we care about what it *really* means to compute in our world. Our motivation might be intellectual curiosity, and this is a great reason for thinking about computer science. But there are also other reasons - for example when designing cryptosystems, we might want to come up with methods today that will be secure against attackers far in the future that may possess quantum computers, or we might want to understand whether the rules of quantum computing can give us more security than is possible otherwise.
- *To help overcome the end of Moore’s law:* For roughly half a century, computers were becoming faster at an exponential rate, but this is no longer the case, so perhaps we should start looking into new computational paradigms. Quantum computing may well be such a paradigm. There are many efforts world-wide to build quantum computing hardware, but the technological challenges are enormous (currently existing ones are small and do not yet work very well).

Thus, quantum computing takes seriously the idea that the world really plays by different rules (namely by the rules of quantum mechanics) than by the ones we usually like think about as computer scientists, and asks what this means for the theory (and hopefully at some point also for the practice) of computation.

As such it is no surprise that quantum computing is a very interdisciplinary field: it sits right at the intersection of computer science (which gives us the right questions to ask), mathematics (which gives us the tools to answer them), and physics (which gives us the rules of quantum mechanics by which we need to play). In this course, we will study quantum computing from the perspective of theoretical computer science (TCS), so we will need to understand how to generalize computer science concepts to the quantum setting. In the remainder of today’s lecture we will give some more context and highlight some cool results that we will discuss later in this term, to get you all motivated. Next week we will then kick off things more properly.

1.2 Computational problems

One central goal of TCS is to understand how to solve *computational problems*, by *algorithms*, and as *fast* as possible. Let us recall some well-known computational problems.

Problem 1.1: Multiplication

Given two n -digit numbers, compute their product.

For small n you know how to do this by heart, but imagine $n = 1000$ or so, then this is clearly annoying enough of a problem to need an algorithm. For example, in high school you learn the “long multiplication” algorithm (“schriftliche Multiplikation”), which allows you to solve this problem assuming you are patient enough (and do not run out of paper). Here is a reminder for how it works:

$$\begin{array}{r}
 4891 * 3722 \\
 \hline
 = \quad 14673 \\
 + \quad 34237 \\
 + \quad \quad 9782 \\
 + \quad \quad \quad 9782 \\
 \hline
 = \quad 18204302
 \end{array}$$

We see that in order to multiply two n -digit numbers using this method we perform $O(n^2)$ steps. (In the example we used decimal digits, while in computer science we usually use binary digits, but this does not make a difference.) Thus the number of steps is polynomial in the input size of this problem. We say that [Problem 1.1](#) can be solved in *polynomial time* – it is in the complexity class P. Such problems are usually thought of as “easy” or at least “tractable”.

In fact, we can do even better: Some of you may know the Karatsuba algorithm, which solves for [Problem 1.1](#) in time $O(n^{\log_2(3)})$, with $\log_2(3) \approx 1.58$. In 1997, Schönhage-Strassen gave an algorithm that runs in time $O(n \log(n) \log(\log(n)))$. Their algorithm is practically useful and outperforms simpler methods for numbers in the order of $n = 100000$ decimal digits. We will end up learning the basic idea of how it works when we discuss the Fourier transform. Very recently, in 2019, Harvey-van der Hoeven gave an algorithm that uses $O(n \log(n))$ time. Since the “big O” notation hides huge constants, their algorithm is not practical at all. Since $\log(n)$ grows much slower than n , these algorithms are much faster than the high school method when n is really large. Can we do even better? It is believed that $O(n \log(n))$ is optimal, but noone really knows! It’s remarkable that there are still deep open problems about as elementary of a computational problem as multiplying two numbers.

Problem 1.2: Factoring

Given an n digit number x that is not prime, find a nontrivial factor. That is, find a number $1 < p < x$ such that $p \mid x$ (x divides x).

Here is an easy “brute force” algorithm to address this problem: Test if $2 \mid x$, $3 \mid x$, and so forth. How many options do we have to test in the worst case in order to find a factor? If p is not a prime, then we can always find a factor p such that $p \leq \sqrt{x}$. Since x is an n -bit number, we have $x < 2^n$. Thus, in general we need to test

$$\sqrt{2^n} = 2^{n/2} = (1.415\dots)^n$$

options. Thus this naive algorithm runs in exponential time!

Can we do better? There are much more clever algorithms. The best algorithm is the so-called “generalized numbers field sieve”, and it is *believed* to run in $2^{O(n^{1/3})}$ time.¹ This is called “subexponential time” – it runs much faster than an exponential time algorithm, but much slower than a polynomial time algorithm. Famously, no polynomial-time algorithm for factoring is known that runs on ordinary ‘classical’ computers. (When quantum computer scientists say ‘classical’, they mean ‘non-quantum’.) The security of, e.g., the RSA cryptosystem relies on the hope that this is indeed a difficult problem. . .we will return to this hope momentarily.

It is also worth mentioning that the factoring problem is *not* believed to be NP-hard.

Problem 1.3: Satisfiability

Given an Boolean formula $f: \{0, 1\}^n \rightarrow \{0, 1\}$, is there a satisfying assignment? That is, is there $\mathbf{x} \in \{0, 1\}^n$ such that $f(\mathbf{x}) = 1$?

Here there is again an easy algorithm that runs in exponential time: simple try all 2^n many possible assignments \mathbf{x} . While there are more clever algorithms, this problem is NP-hard, and so it is believed that there should not be any polynomial-time algorithm (otherwise, $P = NP$). However, the situation is possibly even worse: computer scientists believe that, roughly speaking, any algorithm for the satisfiability problem as stated in [Problem 1.3](#) necessarily requires time $\tilde{\Omega}(2^n)$ in the worst case, hence it should be impossible to even just find an algorithm that runs, e.g., in time $O(1.999^n)$. This belief is known as the *strong exponential time hypothesis (SETH)*.

1.3 Probabilistic computing

In the 70s, people started to wonder whether randomness could help compute faster. This gave rise to the field of *probabilistic computing*, which equips algorithms with one new trick – the ability to toss a fair coin (bit) that returns heads (1) or tails (0) with 50% probability each.

Do algorithms become more powerful if they have the ability to make random choices? Of course, if the task is to produce true randomness or to exactly simulate a random process, then this is trivially true. But can randomness help solve problems faster that have nothing to do with randomness – especially if we are happy to allow the algorithm to fail with some tiny probability? For an example, let us consider the following problem:

Problem 1.4: Primality

Given an n -bit number, decide if it is a prime.

For example, key pairs for the RSA cryptosystem are generated by picking two large primes, which is only one reason for why are interested in solving [Problem 1.4](#) efficiently.

In 1976, Miller proposed a deterministic algorithm for the primality problem that takes time $\tilde{O}(n^6)$, but it relies on the so-called *generalized Riemann hypothesis (GRH)*, which is not known to be true. A breakthrough happened in 2002, when Agrawal-Kayal-Saxena discovered the first deterministic primality testing algorithm that really runs in polynomial-time (not relying on any unproven hypothesis). Their algorithm ran in

¹ **\tilde{O} -Notation:** We write $f(n) = \tilde{O}(g(n))$ if $f(n) = O(g(n) \log^k(g(n)))$ for some integer k . This is a convenient notation to ignore logarithms, which is often a good idea (recall that $\log(n)$ and its powers grows much slower than n). For example, you can simply write $\tilde{O}(n)$ regardless of whether your algorithm takes time $O(n)$, or $O(n \log(n))$, or even $O(n \log^{100}(n))$.

time $\tilde{O}(n^{12})$. It was later improved by Lenstra-Pomerance. Their algorithm is the deterministic state of the art; it runs in time $\tilde{O}(n^6)$. While polynomial time, this is not very practical.

In contrast, early on, in 1977, Solovay-Strassen found that there exists a *randomized algorithm* that runs in time $\tilde{O}(n^3)$ and succeeds up to an arbitrarily small (fixed) failure probability. This is already quite practical! In 1980, Rabin-Miller discovered an even faster randomized algorithm – it runs in time $\tilde{O}(n^2)$ for any fixed failure probability.² It is widely used in practice!

Here is another example:

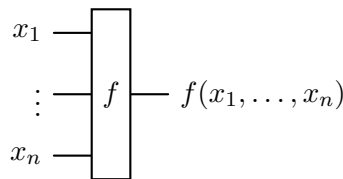
Problem 1.5: Zero testing

Given a polynomial function $f(x_1, \dots, x_n)$, decide if it always returns zero.

If the polynomial is given as a linear combination of monomials, then this is simple. But what if the polynomial is given by a mathematical formula? For example, do you see right away whether the following polynomial is zero or not?

$$f(x_1, \dots, x_n) = \det \begin{pmatrix} 1 & x_1 & \dots & x_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \dots & x_n^{n-1} \end{pmatrix} - \prod_{j < k} (x_k - x_j)$$

It is not so clear how to solve this problem efficiently. If you simply write out the determinant then there will be exponentially many terms, so this will not be efficient. Or how about the more difficult case when a formula but just a “black box” that evaluates f ?



It turns out that there is a simple *randomized* algorithm that even works in this latter case: Simply input a random point (x_1, \dots, x_n) into the function. If f is not the zero polynomial, then it is very unlikely that $f(x_1, \dots, x_n) = 0$. This can be made quantitative by the so-called Schwartz-Zippel lemma, and results in an algorithm that is efficient in theory and also in practice.

Thus, randomness appears to help us compute faster. Using randomness, we can find polynomial speedups over the best known deterministic algorithms also for many other problems. However, computer scientists believe that there is some limit to this: namely, it is believed that randomness only gives at most polynomial speedups (such as $\tilde{O}(n^2)$ instead of $\tilde{O}(n^6)$). In particular, many computer scientists believe that solving the satisfiability problem (Problem 1.3) should require time $\tilde{\Omega}(2^n)$ even when using randomness.

Summary of probabilistic computing:

- we get it by adding to deterministic computing one new trick – random coin tosses (or bit flips)
- by definition, this is useful to simulate random processes
- but even for many problems that do *not* involve randomness, we can get speedups over the best deterministic algorithms
- however, it is believed that we can only get polynomial speedups in this way

Before moving on, let us us briefly pause to think about how represent the state of programs mathematically.

When you have a deterministic program that uses n bits of memory, you would simply describe its state by a vector or bitstring $x \in \{0, 1\}^n$.

²More precisely, for any choice of k the two algorithms run in time $\tilde{O}(n^3k)$ or $\tilde{O}(n^2k)$, respectively, and fail with probability at most 2^{-k} .

In contrast, the state of a probabilistic program that uses n bits of memory will at any point be described by a *probability distribution*

$$\begin{pmatrix} p(00\dots 00) \\ p(00\dots 01) \\ \vdots \\ p(11\dots 11) \end{pmatrix}.$$

E.g., $p(00\dots 00)$ is the probability that $x_1 = x_2 = \dots = x_n = 0$ etc. This is a vector of exponential size 2^n , but of course this only arises in the mathematical description – we never have to write this down when running the program!

Similarly, the overall result of applying a probabilistic program operating on n bits can be described by a $2^n \times 2^n$ “transition matrix”:

$$\begin{pmatrix} p(0\dots 0|0\dots 0) & \dots & p(0\dots 0|1\dots 1) \\ \vdots & p(y_1\dots y_n|x_1\dots x_n) & \vdots \\ p(1\dots 1|0\dots 0) & \dots & p(1\dots 1|1\dots 1) \end{pmatrix},$$

where the entry $p(y_1\dots y_n|x_1\dots x_n)$ is the probability that the program outputs y if the input is x . Can you see what conditions this matrix needs to satisfy so that it maps probability distributions to probability distributions? For example, consider the following program:

```
def f(x):
    if x == 0:
        return 0
    return random_bit()
```

When given 0, it returns 0, while when given 1 it returns a uniformly random bit. This would be described by the transition matrix

$$\begin{pmatrix} p(0|0) & p(0|1) \\ p(1|0) & p(1|1) \end{pmatrix} = \begin{pmatrix} 1 & 1/2 \\ 0 & 1/2 \end{pmatrix}.$$

1.4 Quantum computing

Given the success of adding randomness to computation, researchers in the 1980s began to wonder whether one could similarly exploit *quantumness* for computation. The researchers that pioneered the idea of quantum computing include Nobel prize winner Richard Feynman, Yuri Manin, Paul Benioff, and David Deutsch, while Charles Bennett and Gilles Brassard following up on earlier work by Stephen Wiesner discovered quantum cryptography and information. Some of them, such as the physicist Feynman, were motivated by simulating quantum mechanical systems (it seems plausible that this should be easier if your computer is also quantum!), while others were more concerned with the ultimate limits of computation.

So, what do we get by replacing bits with quantum bits? We mention two highlights that revolutionized the field:

- In 1994, Shor discovered a quantum algorithm that can factor n -digit numbers in time $\tilde{O}(n^2)$. This is exponentially better than the best known classical algorithms, as we discussed below [Problem 1.2](#).
- In 1996, Grover discovered a quantum algorithm that can be used to solve the satisfiability problem ([Problem 1.3](#)) in time $\tilde{O}(\sqrt{2^n}) = \tilde{O}(1.415\dots^n)$. As discussed, this is exponentially better than the best known classical algorithms, and indeed believed to be impossible using classical algorithms.

In this course, we will learn exactly how this works (and much more). However, it is good to emphasize that quantum computers are not all-powerful, even in theory. In particular:

- It is strongly believed that they will *not* be able to solve NP-hard problems efficiently!

- Quantum computers do *not* simply compute faster or in parallel. Instead they rely on an entirely new model of computation (which however includes deterministic and randomized computation as special cases); hence entirely new computer science ideas are required to make use of them and design quantum algorithms such as the ones by Shor and Grover mentioned above.

So, how do quantum computers work? The state of n quantum bits is described by an 2^n -dimensional vector

$$\psi = \begin{pmatrix} \psi_{00\dots00} \\ \psi_{00\dots01} \\ \vdots \\ \psi_{11\dots11} \end{pmatrix}$$

This looks similar to probabilistic computing, but the numbers ψ_x are *not* probabilities. Instead they can be *negative* and even complex numbers. We will call them ‘amplitudes’ to remember that they are not probabilities. Instead of summing to one, their absolute values squared sum to one: $\sum_x |\psi_x|^2 = 1$. Geometrically, this means that ψ is a vector of norm (length) one. While this vector has exponentially many entries, this is only the mathematical description. In our computer it will simply corresponds to the state of n quantum bits. We should not be surprised – the same was true for probabilistic computing.

What operations can we apply to n quantum bits? First, we can again apply $2^n \times 2^n$ -matrices to manipulate the state of our n quantum bits:

$$U = \begin{pmatrix} U_{0\dots0,0\dots0} & \dots & U_{0\dots0,1\dots1} \\ \vdots & U_{y_1\dots y_n,x_1\dots x_n} & \vdots \\ U_{1\dots1,0\dots0} & \dots & U_{1\dots1,1\dots1} \end{pmatrix},$$

These matrices should send quantum states to quantum states, so they need to preserve the norms of vectors (we will learn that such matrices are called ‘unitary’). Second, we can extract answers by *measurement*. If we measure n quantum bits, we obtain n ordinary bits x_1, \dots, x_n with probability $|\psi_{x_1,\dots,x_n}|^2$. Note that this is a probability distribution! How can we use all this? The high-level idea of any quantum algorithm is as follows:

1. We start by initializing our quantum algorithm in some definite state, such as $\begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$.
2. We manipulate it using 2^n -dimensional linear algebra. For example, we will see that we can apply Fourier transforms very efficiently using quantum computers, and that this is key for Shor’s factoring algorithm!
3. Finally, we measure to extract the desired answer.

Of course the above can only give you a very rough idea. We will discuss the rules of quantum computing in detail in [Chapters 2](#) and [3](#).

Summary of quantum computing:

- it is again obtained by taking something familiar and adding one new trick - roughly speaking, we replace probabilities (which are always ≥ 0) by *complex numbers* (called *amplitudes*).
- by definition, this is useful to simulate *quantum* processes
- but even for many problems that do not involve quantumness, we can get speedups over the best *classical* (= non-quantum, i.e., deterministic or randomized) algorithms
- amazingly, sometimes these speedups can even be *exponential*!

The ideas of quantum computation and quantum information were revolutionary at the time. Nowadays they are well established in the scientific community and even beyond. For example, the 2023 Breakthrough Prize in Fundamental Physics was awarded to Bennett, Brassard, Deutsch, and Peter Shor (who we will

mention in a moment) for their fundamental contributions to quantum computation and information (exactly what we will study in this course), while the 2022 Nobel Prize in Physics went to Alain Aspect, John Clauser, and Anton Zeilinger for their experiments demonstrating some important phenomena that show that the quantum information is really as weird as we will see that it is. Many big (and also some smaller) tech companies such as IBM, Google, Amazon, Microsoft, etc. are actively working on building quantum computers, and many other companies are interested in the capabilities that quantum computers may bring to the table in the future.

1.5 Outlook

In the coming weeks we will first discuss how to mathematically describe quantum bits and quantum circuits (Chapters 2 and 3). Then we will design and analyze quantum algorithms for a variety of computational problems, including 3SAT, discrete logs, and factoring (Chapters 4 to 10). We will also learn techniques for how one can show that an algorithm is optimal (Chapter 11). So far, we mostly discussed using quantum computing to compute faster. In the last part of the course we think more deeply about the strange features of quantum information itself – why quantum bits cannot be cloned, what entanglement is and how to use it to teleport and beat non-quantum players in certain games, and why all this is useful for cryptography (Chapters 12 to 14). In the last lecture we will give a brief outlook on some other exciting topics that go beyond this first course (Chapter 15).

These lecture notes are meant to be self-contained and we will also provide video recordings. It can still be useful to look at other sources. We recommend two particularly excellent ones: the lecture notes by Ronald de Wolf, which are available at <https://arxiv.org/abs/1907.09415>, and Ryan O’Donnell lecture series at <https://www.youtube.com/playlist?list=PLm3J0oaFux3YL5qLskC6xQ24JpMwOAeJz>. If you are looking for a textbook that covers much more material, the book “Quantum Computation and Quantum Information” by Nielsen and Chuang is an oldie but goodie; it also contains a detailed refresher of the most relevant mathematics that we will need.

Chapter 2

Quantum bits: states and operations

In this chapter we will discuss the rules that tell us how to describe a single quantum bit and what we can do with it. These rules are also called the “axioms” of quantum computing.

2.1 States and Dirac notation

An ordinary *bit* is a “system” (“variable”, “degree of freedom”) that can be in one of two possible *states*. Here are some examples:

- A coin that can be in one out of two orientations: heads or tails.
- One of two voltage level: low or high. This is [one way](#) in which bits are stored in your computer.
- One out of two directions in which a piece of a [tape drive](#) is magnetized.

One of the great insights of computer science is that we do not want to care about all this, so we simply agree to use 0 and 1 for the two possible states, and move on with life.

Now consider a *random bit*, i.e., a bit whose state is not definite but described by probabilities. For example, consider the state of the x bit at the end of the execution of the following probabilistic program:

```
x = random_bit()
if x == 1:
    x = random_bit()
```

If `random_bit()` performs a fair coin toss then at the end of the program we have

- probability $p = 75\%$ that x is in state 0,
- probability $q = 25\%$ that x is in state 1.

We can summarize these two probabilities by the vector $\begin{pmatrix} 0.75 \\ 0.25 \end{pmatrix}$. More generally, if in a *probabilistic system or computation* a bit has probability p of being 0 and a probability q of being 1, then we can summarize the state of the random bit by the vector

$$\begin{pmatrix} p \\ q \end{pmatrix} = p \begin{pmatrix} 1 \\ 0 \end{pmatrix} + q \begin{pmatrix} 0 \\ 1 \end{pmatrix},$$

where $p, q \geq 0$ and $p + q = 1$. We can think of the state of the random bit as a “mixture” of the ‘definite’ states $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$, which correspond to 0 and 1, respectively.

How about if we have a *quantum bit*, that is, a bit in a quantum system or computation? In this case we again describe it by two numbers, but these can now be negative or even complex. These numbers are called “*amplitudes*”. For example, we might say that a quantum bit has

- amplitude $a = 0.6$ of being in state 0,
- amplitude $b = -0.8$ of being in state 1.

Again this can be described by a vector, here $\begin{pmatrix} 0.6 \\ -0.8 \end{pmatrix}$. More generally, if in a *quantum system or computation* a bit has amplitude a of being 0 and a amplitude b of being 1, then we can summarize the state of the quantum bit by the vector

$$\begin{pmatrix} a \\ b \end{pmatrix} = a \begin{pmatrix} 1 \\ 0 \end{pmatrix} + b \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \quad (2.1)$$

We say that the state of the quantum bit is a “*superposition*” of the ‘definite’ states $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$, which correspond to 0 and 1, respectively.

What constraints do the amplitudes need to satisfy? As mentioned, they can be arbitrary complex numbers, $a, b \in \mathbb{C}$, but we require that their squared absolute values sum to one: $|a|^2 + |b|^2 = 1$. Note that this is the case in our example, since $|0.6|^2 + |-0.8|^2 = 0.36 + 0.64 = 1$. Let us summarize this in a box (we call it a “first attempt”, since next time we will define these rules in general, for more than just one qubit):

Rule 1: State of a quantum bit

The state of a *quantum bit* or “*qubit*” is described by a complex unit vector

$$\begin{pmatrix} a \\ b \end{pmatrix} \in \mathbb{C}^2,$$

meaning $a, b \in \mathbb{C}$ and $|a|^2 + |b|^2 = 1$.

Geometrically, this means that the states of a quantum bit are *unit vectors*, that is, two-dimensional vectors of *norm* or *length* one. In contrast, states of a *random bit* are described by a line segment. This is visualized in Fig. 2.1:

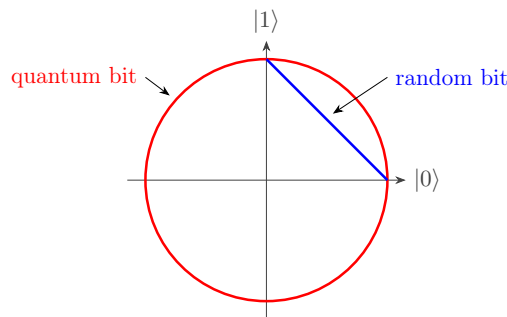


Figure 2.1: States of a quantum bit (red) vs states of a random bit (blue).

In the literature you will often read that “quantum states are unit vectors in a complex Hilbert space”. In finite-dimensions, a *Hilbert space* is simply a vector space with an inner product. (It’s like a Euclidean space, but in our case over the complex numbers, not over the reals.) In infinite dimensions, there is some additional requirement that makes the functional analysis work out fine, but this will not concern us further in this course.

2.1.1 Dirac notation

Before moving on and discussing what we can *do* with quantum bits, we briefly pause to introduce *Dirac notation*, which is extremely popular and used in almost all textbooks and papers:

- In Dirac notation, column vectors are denoted by putting a little flag around them. For example, we write $|\psi\rangle \in \mathbb{C}^d$ and not $\psi \in \mathbb{C}^d$. This is called a “*ket*”. You can write any label you like, such as $|v\rangle$, $|0\rangle$, $|\text{error}\rangle$, ...
- If $|\psi\rangle = \begin{pmatrix} \psi_1 \\ \vdots \\ \psi_d \end{pmatrix}$ is a column vector as above, then we define

$$\langle\psi| := (|\psi\rangle)^\dagger = (\overline{\psi_1} \quad \dots \quad \overline{\psi_d}),$$

which is a row vector.¹ This is called a “bra”.

- If $|\psi\rangle = \begin{pmatrix} \psi_1 \\ \vdots \\ \psi_d \end{pmatrix}$ and $|\phi\rangle = \begin{pmatrix} \phi_1 \\ \vdots \\ \phi_d \end{pmatrix}$ then their “bra-ket”² computes the inner product:

$$\langle\psi|\phi\rangle := \langle\psi| \cdot |\phi\rangle = (\overline{\psi_1} \quad \dots \quad \overline{\psi_d}) \begin{pmatrix} \phi_1 \\ \vdots \\ \phi_d \end{pmatrix} = \overline{\psi_1}\phi_1 + \dots + \overline{\psi_d}\phi_d = \sum_{j=1}^d \overline{\psi_j}\phi_j.$$

(A row vector is a $1 \times d$ matrix and a column vector is a $d \times 1$ matrix, so their product is a 1×1 -matrix – a number!)

- If we are interested in the norm $\|\psi\| := \sqrt{\sum_{j=1}^d |\psi_j|^2}$ of a vector, then we can compute its square by taking the inner product of the vector with itself:

$$\langle\psi|\psi\rangle = \sum_{j=1}^d |\psi_j|^2 = \|\psi\|^2.$$

In particular, if $\langle\psi|\psi\rangle = 1$ then, and only then, is $|\psi\rangle$ a quantum state.

Why do we use Dirac notation? Mostly for historical reasons (“everyone is doing it”), but once you get used to it you will find it quite convenient that you can see immediately from the notation what kind of mathematical object you are dealing with.

2.1.2 Some important qubit states

Let us discuss some useful states and practice Dirac notation while doing so. The examples we will discuss all come in pairs, with each pair defining a basis³ of \mathbb{C}^2 .

Example 2.1: Computational basis

First of all, we have the two standard basis vectors of \mathbb{C}^2 :

$$|0\rangle := \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle := \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

We also call these the *computational basis* states. Indeed, these are states because their norm is one, and they are pairwise orthogonal, meaning that $\langle 0|1\rangle = \langle 1|0\rangle = 0$.

We will think of the computational basis state as describing an ordinary bit, sitting inside our quantum bit. This is useful because in many quantum algorithms the input will be ordinary bits.

Example 2.2: Hadamard basis

Another basis that will be useful is the so-called *Hadamard basis* of \mathbb{C}^2 :

$$|+\rangle := \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad |-\rangle := \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ -1 \end{pmatrix}.$$

Naturally, these are called the “plus states” and the “minus state”.

This might perhaps seem somewhat unorthodox, but you may find it to be quite useful to remember these states...

¹On Practice Set 1 you learned the notation $A^\dagger = \overline{A^T} = (\overline{A})^T$ for the *adjoint* or conjugate transpose of a matrix A . The symbol \dagger is pronounced “dagger”. Here we use this in the special case that A is a column vector, i.e., a matrix of size $d \times 1$.

²Bracket! Get it?

³Whenever we say “basis” in these notes we will mean “*orthonormal basis*”, meaning that the basis vectors have norm one and any two of them are orthogonal. For the computational basis, this means $\langle 0|0\rangle = \langle 1|1\rangle = 1$ and $\langle 0|1\rangle = \langle 1|0\rangle = 0$.

Example 2.3: θ -rotated basis

Finally, for every θ , we define the θ -rotated basis:

$$|\theta\rangle := \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix}, \quad |\theta^\perp\rangle := \begin{pmatrix} -\sin(\theta) \\ \cos(\theta) \end{pmatrix}.$$

These are valid states, since $\cos^2(\theta) + \sin^2(\theta) = 1$ for every angle θ .

Can you find all these basis states in Fig. 2.1?

Using Dirac notation, any vector $|\psi\rangle \in \mathbb{C}^2$ can be written as

$$|\psi\rangle = a|0\rangle + b|1\rangle, \quad (2.2)$$

where $a, b \in \mathbb{C}$. Indeed, this is exactly the same as what we wrote in Eq. (2.1), except that we now use our new notation. Recall that this is called “*superposition*”. As an aside, note that we can compute the “*amplitudes*” a and b in Eq. (2.2) as follows:

$$a = \langle 0|\psi\rangle \quad \text{and} \quad b = \langle 1|\psi\rangle$$

You can see this either concretely, by writing everything in terms of two-dimensional vectors, or abstractly, by using that the computational basis is an orthonormal basis. Thus we can compute the squared norm of any vector $|\psi\rangle$ can be computed as follows:

$$\|\psi\|^2 = |a|^2 + |b|^2 = |\langle 0|\psi\rangle|^2 + |\langle 1|\psi\rangle|^2. \quad (2.3)$$

For a quantum state, this must be equal to one.

The above works for any orthonormal basis (and recall that mostly all bases that we will be dealing with will be orthonormal): For example, any vector $|\psi\rangle \in \mathbb{C}^2$ can also be written as a superposition of the Hadamard basis states,

$$|\psi\rangle = c|+\rangle + d|-\rangle, \quad (2.4)$$

where $c = \langle +|\psi\rangle$ and $d = \langle -|\psi\rangle$. For example,

$$\begin{aligned} |0\rangle &= \frac{1}{\sqrt{2}}|+\rangle + \frac{1}{\sqrt{2}}|-\rangle, \\ |1\rangle &= \frac{1}{\sqrt{2}}|+\rangle - \frac{1}{\sqrt{2}}|-\rangle, \end{aligned}$$

because $\langle +|0\rangle = \frac{1}{\sqrt{2}}$ etc. If this confuses you, check explicitly that these equations hold by writing everything as two-dimensional vectors.

2.1.3 What do qubits look like in practice?

At the beginning of this chapter we discussed how bits can take many different concrete manifestations. The same is true for quantum bits.

In fact, physicists believe that everything in nature is at its core described by quantum mechanics. This means that every bit in nature is really part of a quantum bit. However, if you take some arbitrary bit then it will be very hard to see (and make use of) its quantum nature. Therefore, when physicists want to build a quantum computer, they try to find or design qubits that are particularly easy to control.

Here are some examples of physical systems that scientists are currently trying to use as qubits:

- **Ions**, that is, charged atoms ($|0\rangle$ and $|1\rangle$ correspond to two energy levels)
- **Electrons** in a semiconductor device ($|0\rangle$ and $|1\rangle$ correspond to two states of the electron “spin”, which you can think of as a kind of magnetization)

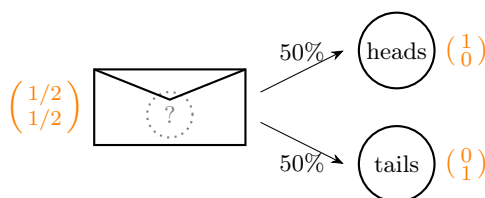
- **Superconductors** ($|0\rangle$ and $|1\rangle$ correspond to, e.g., charge or flux stored in a superconducting loop)
- **Photons**, which are the particles that light is made of ($|0\rangle$ and $|1\rangle$ correspond to, e.g., different polarizations of light). This, at last, is something you can try for yourself – we’ll come back to it later.

We see that all these have in common that they are *really small* and also *much more complicated* than, say, two voltage levels that are used to represent ordinary bits in a computer. Hence the difficulty of building larger and better quantum computers.

2.2 Measurements and Unitaries

What can we do with a qubit? Basically two things – we can manipulate it and we can measure it. We will start with measurement.

To motivate this, we start with an analogy. Suppose I toss a fair coin and hide the result in an envelope. As discussed, you would describe this by a random bit, with probability distribution $\begin{pmatrix} 1/2 \\ 1/2 \end{pmatrix}$. But if you open the envelope and look into it, then you will see a definite outcome – either 0 (heads) or 1 (tails) – and you should update your description to either $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ or $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$, depending on what you saw. This makes sense, because if you look into the envelope *again*, then you will of course see the same outcome! We might say that the state of the random bit has “collapsed” when you “measured” (i.e., look at) it. Here is a picture that visualizes the situation:




Quantum bits work very much the same:

Rule 2: Measuring a quantum bit

We can *measure a qubit in the computational basis*. If the qubit state is in state $|\psi\rangle$, then:

- with probability $|\langle 0|\psi\rangle|^2$, the outcome reads “0” and the state changes to $|0\rangle$,
- with probability $|\langle 1|\psi\rangle|^2$, the outcome reads “1” and the state changes to $|1\rangle$.

We will use the following symbol to denote measurements:  (the qubit comes in from the left and exits on the right; we sometimes write the outcome on top)

Below and on the homework you will learn how to measure in another basis.)

Note that the rule for computing probabilities is reasonable: Because qubit states are unit vectors, we necessarily have that

$$|\langle 0|\psi\rangle|^2 + |\langle 1|\psi\rangle|^2 = 1,$$

see Eq. (2.3), so the two probabilities add up to one. We can remember once and for all: “to compute the probability of an outcome, we need to compute the absolute value **squared** of the corresponding amplitude”.

For example, if our qubit is in state

$$|\psi\rangle = 0.6|0\rangle - 0.8i|1\rangle = \begin{pmatrix} 0.6 \\ -0.8i \end{pmatrix}$$

then with $|0.6|^2 = 36\%$ probability we see outcome “0” and the qubit state changes to $|0\rangle$, while with $|-0.8i|^2 = 64\%$ probability we see outcome “1” and the qubit state changes to $|1\rangle$. Because the state changes when we measure a qubit, we will always get the same result if measure it again!

In quantum computations, we usually measure qubits at the very end, in order to extract the answer to the problem we are trying to solve. The next rule tells us another way by which we can manipulate qubits:

Rule 3: Applying unitaries to a quantum bit

We can apply arbitrary unitary matrices to a qubit. If the state of the qubit is $|\psi\rangle$ and we apply a unitary U , then the state changes to $U|\psi\rangle$.

We will use the following symbol to denote unitaries: $|\psi\rangle \text{---} \boxed{U} \text{---} U|\psi\rangle$

Recall that a matrix U is called *unitary* if $U^\dagger U = I$, where I is the identity matrix. There are many equivalent characterizations, as you discussed in the tutorials. We restate them here for your convenience:

Remark 2.4: Unitaries – good to know

The following conditions are all equivalent for a $d \times d$ -matrix U :

- (a) $U^\dagger U = I$.
- (b) $U U^\dagger = I$.
- (c) U preserves inner products, i.e., $\langle Uv|Uw\rangle = \langle v|w\rangle$ for all $|v\rangle, |w\rangle \in \mathbb{C}^d$.
- (d) U preserves the length of vectors, i.e., $\|Uv\| = \|v\|$ for every $|v\rangle \in \mathbb{C}^d$.
- (e) U sends any orthonormal basis to an orthonormal basis.
- (f) U sends at least one orthonormal basis to an orthonormal basis.

The point is that if you manage to prove one of these conditions, then your matrix is unitary and you get all the other ones for free. Please make sure that you understand and remember this result.

In particular, characterization (d) tells us that unitaries send unit vectors (i.e., vectors of norm one) to unit vectors. Thus, if $|\psi\rangle$ is a qubit state then $U|\psi\rangle$ is again a qubit state.

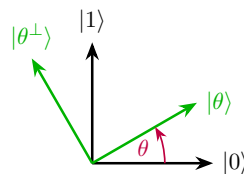
Unitaries that are used in quantum circuits are also called *gates*. Here are some examples:

Example 2.5: Rotations

For every $\theta \in \mathbb{R}$, we define the *rotation unitary*

$$R_\theta = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$$

Note that R_θ rotates the computational basis onto the θ -rotated basis (Example 2.3): $R_\theta|0\rangle = |\theta\rangle$ and $R_\theta|1\rangle = |\theta^\perp\rangle$. As a picture:



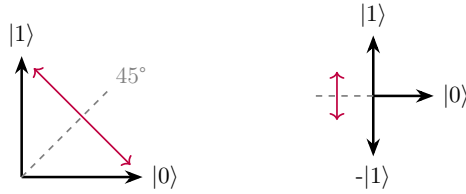
Example 2.6: Pauli matrices

The three *Pauli matrices* are defined as follows:

$$X = \text{NOT} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}.$$

Note that $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$, so the X gate acts as a NOT on the computational basis. For this reason we also call it the NOT gate.

We will mostly need the first two, X and Z . These can be visualized as reflections in \mathbb{R}^2 :



The third one, Y , can also be thought of as a reflection. Can you see how?

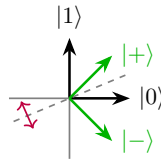
Example 2.7: Hadamard matrix

The *Hadamard matrix* is defined as

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad (2.5)$$

Note that H sends $|0\rangle \leftrightarrow |+\rangle$ and $|1\rangle \leftrightarrow |-\rangle$.

This is again a reflection:



Two other unitaries are the T -gate, which seems quite mysterious as well:

$$T = \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{i} = e^{i\pi/4} \end{pmatrix} \quad (2.6)$$

and the *identity matrix*, which is complete boring and leaves all states unchanged:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

(we will use I for any identity matrix, not just for the 2×2 one)

Unitaries have some interesting properties:

- They are invertible: the inverse of a unitary U is given by U^\dagger . How can we embed ordinary classical computation into this? After all, most Boolean gates such as AND and OR are not reversible!
- They have square roots: for every unitary U there exists a unitary V such that $V^2 = U$. For example, even the NOT unitary has a square root!

2.3 Quantum Circuits

When we think about quantum algorithms, we will usually describe them in terms of *quantum circuits*. We start with one or more qubits in some state (typically all qubits are initialized in the $|0\rangle$ state) and then compute by applying various unitary and measurement *gates*.

Let us practice and consider the following quantum circuit:



(Recall that we input qubits on the left and apply the gates from left to right.)

What happens when you apply this circuit to a qubit initialized in state $|0\rangle$? To answer this question, we compute what happens step by step:

1. First, the Hadamard gate changes the $|0\rangle$ state to $H|0\rangle = |+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$.
2. Next, we measure. With probability $(\frac{1}{\sqrt{2}})^2 = \frac{1}{2}$, we see outcome “0” and the state changes to $|0\rangle$. Otherwise we see outcome “1” and the state changes to $|1\rangle$.
3. Finally, we apply another Hadamard gate that changes $|0\rangle$ to $|+\rangle$ and $|1\rangle$ to $|-\rangle$.

Thus see that when we apply the above circuit to $|0\rangle$ then the overall effect is the following:

- With 50% probability, we see outcome “0” and the output state is $|+\rangle$.
- With 50% probability, we see outcome “1” and the output state is $|-\rangle$.

Let’s now ask a slightly harder question: what happens if we apply the above circuit to some *arbitrary* initial state $|\psi\rangle$? Here is a nice trick: Instead of expanding the initial state in the form $|\psi\rangle = a|0\rangle + b|1\rangle$, let us instead write it in the form of Eq. (2.4), that is,

$$|\psi\rangle = c|+\rangle + d|-\rangle,$$

where $c = \langle +|\psi\rangle$ and $d = \langle -|\psi\rangle$. Now we proceed as above:

1. The Hadamard gate changes the state $|\psi\rangle$ to $H|\psi\rangle = c|0\rangle + d|1\rangle$.
2. When we measure, with probability $|c|^2 = |\langle +|\psi\rangle|^2$ we see outcome “0” and the state changes to $|0\rangle$, and with probability $|d|^2 = |\langle -|\psi\rangle|^2$ we see outcome “1” and the state changes to $|1\rangle$.
3. As above, the second Hadamard gate that changes $|0\rangle$ to $|+\rangle$ and $|1\rangle$ to $|-\rangle$.

In summary:

Example 2.8: Hadamard measurement

If we apply the quantum circuit in (2.7) to a qubit in an arbitrary state $|\psi\rangle$, then:

- with probability $|\langle +|\psi\rangle|^2$, we see outcome “0” and the output state is $|+\rangle$.
- with probability $|\langle -|\psi\rangle|^2$, we see outcome “1” and the output state is $|-\rangle$.

Note that these rules look exactly the same as Rule 2, except that the standard basis is replaced by the Hadamard basis. Thus we call the quantum circuit in (2.7) a *Hadamard measurement*.

Note also that for $|\psi\rangle = |0\rangle$, we have $|\langle +|\psi\rangle|^2 = |\langle -|\psi\rangle|^2 = \frac{1}{2}$, and hence we recover what we computed above in the special case that $|\psi\rangle = |0\rangle$.

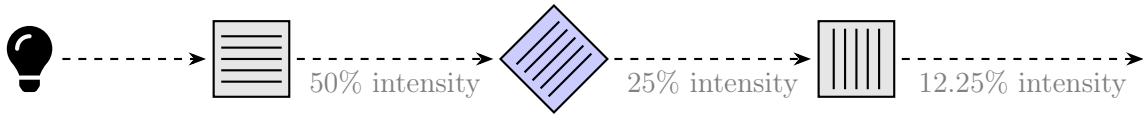
2.4 Is all of this real?

You may be wondering if these strange rules of quantum computing are really necessary (or even correct). Here is a simple experiment you can do at home (or in class). As mentioned earlier, light can be polarized. How can we polarize it? Simply take a polarizing filter (we’ll give you some in class) and rotate it in whichever direction you want to polarize, for example, horizontally, vertically, or diagonally. Here is what happens when you first place a horizontal and then a vertical filter:



Let’s try to explain what we see. Light emitted by the sun, or by a light bulb, is in in some random state, so it makes sense that the first filter should reduce the intensity by around half. And once the light is horizontally polarized, it will surely be absorbed by the second filter, which only lets through vertically polarized light. Thus, no light is let through at all. So far so good.

Now let’s ask ourselves what should happen if we insert another filter inbetween these two filters. Since filters only ever absorb light, it seems clear that no light should be let through either. However, here is the kicker:



If we add a diagonal polarizing filter in the middle, *more* light is let through! Watch [this video](#) if you don't believe it. How can this be?!

This might seem extremely strange, but in fact can be easily explained by the rules of quantum bits. As mentioned earlier, light consists of many small particles called photons. We can think of the polarization of a photon as defining a qubit: the basis state $|0\rangle$ corresponds to horizontal polarization, and $|1\rangle$ to vertical polarization. How do we describe polarizing filters?

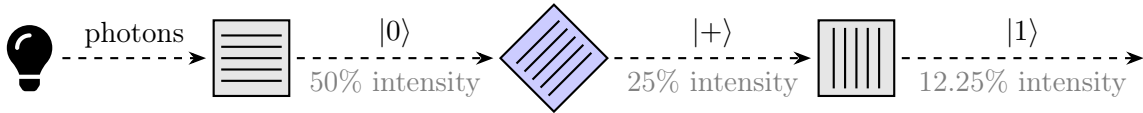
- A horizontally-oriented polarizing filter, , performs a measurement in the standard basis and only lets through $|0\rangle$.
- If we rotate it all the way to be vertically oriented, , it performs a measurement in the standard basis and only lets through $|1\rangle$.
- But if we rotate it halfway so that it is diagonally oriented, , then it performs a Hadamard measurement (precisely as in [Example 2.8](#)) and only lets through $|+\rangle$.

(In general, if we rotate the filter by some angle θ then it will correspond to measuring in the θ -rotated basis and only letting through the $|\theta\rangle$ state.)

Now the rules of quantum computing perfectly explain what we saw in the two experiments. In the first experiment, the first filter only lets through photons (qubits) in state $|0\rangle$ (and this happens roughly half the time). The second filter measures again in the standard basis but only lets through $|1\rangle$ – but this never happens, since if we measure a qubit in state $|0\rangle$ then we never get outcome “1”. Hence all photons are absorbed and no light is let through:



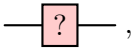
How about the second experiment? Again, the first filter only lets through photons (qubits) in state $|0\rangle$ (and this happens roughly half the time). The second filter perform a Hadamard measurement and only lets through state $|+\rangle$. If we measure a $|0\rangle$ in the Hadamard basis then this happens with probability 50%, as discussed above. The third filter again measures in the standard basis but only lets through $|1\rangle$. Since the Hadamard measurement has changed the state to $|+\rangle$, this now happens with probability $|\langle 1|+\rangle|^2 = \frac{1}{2} > 0$. Thus the overall probability of the photon making it through all three filters is $\frac{1}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{8} = 12.5\%$. This is precisely the intensity of the light that comes out:



Thus the rules of quantum measurements, which in general can change the state, helped save the day!

2.5 Elitzur-Vaidman quantum ‘bomb’ tester

You will discuss the following in the tutorials (and do even better on the homework)! Imagine someone hands you a box with two openings,



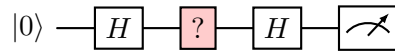
and they promise to you that one of the following is true:

- ⊙ In the first case, the box is empty: if you input in a qubit then it is simply let through unchanged.

⚡ In the second case, the box contains a device that measures the qubit in the computational basis. If the result is “0” then the resulting $|0\rangle$ state is let through, while if the result is “1” then it explodes.

(For example, in the second case the box could contain a horizontal polarizing filter, connected to a fuse that sets of the bomb whenever a photon is absorbed. This would be quite the delicate fuse...)

Your task is to determine which is the case, without setting off the bomb. It seems like there is not much you can do: if you input a $|0\rangle$ state then nothing happens in either case, while if you input a $|1\rangle$ state and there is a bomb then it will always explode. However, consider the following quantum circuit:



What happens in each case?

⊙ If the box is empty then the state before the measurement is $HH|0\rangle = H|+\rangle = |0\rangle$ and hence the measurement always gives outcome “0”.

⚡ In this case we input $H|0\rangle = |+\rangle$ into the box. The box measures and will explode with 50% probability. However, if it does not explode then the qubit will leave the box in the $|0\rangle$ state. The second Hadamard turns this into $H|0\rangle = |+\rangle$, and hence the final measurement gives outcome “0” or “1” with 50% probability each.

The punchline is this: *If the measurement at the end yields outcome “1” then we are certain that there is a bomb in the box, without the bomb having exploded – and this happens with $50\% \cdot 50\% = 25\%$ probability!* On this week’s homework you will do even better.

2.6 Beyond qubits

In ordinary computing, we mostly deal with bits, but in principle nothing stops us from considering systems with more than two states. For example, a “trit” is a system that has three possible values (say 0, 1, and 2; or “true”, “false”, and “don’t know”). In general, a “dit” is a system that can be in one of d distinct states, for some integer d . If such a dit is in a random state, it is naturally described by a probability distribution with d many outcomes.

We can generalize the rules of quantum bits in the same way to systems with d basic states:

Rule 1 for qudits: States

The state of a quantum system is described by a unit vector $|\psi\rangle \in \mathbb{C}^d$, where d is the number of basic states.

The standard or computational basis of such a “qudit” is denoted by

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}, \quad \dots, \quad |d-1\rangle = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix},$$

and we can write any vector in \mathbb{C}^d in the form

$$|\psi\rangle = \begin{pmatrix} \psi_0 \\ \psi_1 \\ \vdots \\ \psi_{d-1} \end{pmatrix} = \sum_{j=0}^{d-1} \psi_j |j\rangle.$$

Rule 2 for qudits: Measurement

We can measure a quantum system in state $|\psi\rangle$ in the computational basis. With probability $|\langle j|\psi\rangle|^2$, the outcome will be “ j ”, in which case the state changes to $|j\rangle$.

Rule 3 for qudits: Unitaries

We can apply an arbitrary unitary $d \times d$ matrix to a quantum system. If the state is $|\psi\rangle$ and we apply a unitary U , then the state becomes $U|\psi\rangle$.

However, just like we usually consider multiple bits in case we want more than two options, we will also in the quantum world usually consider multiple qubits if we are interested in a quantum system with more than two basic states. This will be the subject of next week's lecture.

Chapter 3

Quantum computing with many qubits

Today we will discuss what quantum computing looks like if we have more than one quantum bit. Let's first recall the classical situation. Two bits have four possible “definite” states: 00, 01, 10, 11. If we allow randomness, for example because we are analyzing a probabilistic algorithm, then we also consider “mixtures” of these. That is, we describe the state of two random bits by a probability distribution consisting of four probabilities: p_{00} , p_{01} , p_{10} , and p_{11} . More generally, if we have n bits, then there are 2^n definite states, namely the bitstrings of length n ; if the bits are allowed to be in a random state, we would describe them by 2^n probabilities, that is, by a vector of dimension 2^n . As explained in [Chapter 1](#), while this vector is exponentially large, this is only our mathematical description – in reality there are simply n bits that are in some uncertain (to us) state.

3.1 States and operations

How do we describe the state of multiple quantum bits? In the following comes some motivation and a reminder of the tensor product of two vector spaces – if you already know what $\mathbb{C}^2 \otimes \mathbb{C}^2$ means, you can skip over it and simply move to [Rule 1](#) below. Just like we did in [Chapter 2](#) for one qubit, we simply replace probabilities (nonnegative numbers that sum to one) by amplitudes (complex numbers whose squared absolute values sum to one). For example, a quantum state of two qubits is described by four amplitudes ψ_{00} , ψ_{01} , ψ_{10} , and ψ_{11} , such that $|\psi_{00}|^2 + |\psi_{01}|^2 + |\psi_{10}|^2 + |\psi_{11}|^2 = 1$. Similarly, quantum states of n qubits can be represented as vectors of dimension 2^n . However, writing quantum states as concrete vectors in \mathbb{C}^{2^n} quickly gets confusing. For example, already for two qubits it is hard to remember whether we write

$$\begin{pmatrix} \psi_{00} \\ \psi_{01} \\ \psi_{10} \\ \psi_{11} \end{pmatrix} \quad \text{or rather} \quad \begin{pmatrix} \psi_{00} \\ \psi_{10} \\ \psi_{01} \\ \psi_{11} \end{pmatrix}.$$

Instead, let us label the standard basis vectors by bitstrings, defining:

$$|00\rangle := \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad |01\rangle := \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad |10\rangle := \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad |11\rangle := \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Then every two-qubit state can be written in the form

$$|\psi\rangle = \psi_{00}|00\rangle + \psi_{01}|01\rangle + \psi_{10}|10\rangle + \psi_{11}|11\rangle = \sum_{a,b \in \{0,1\}} \psi_{ab}|ab\rangle.$$

This is much nicer to deal with and actually there is no reason at all to think about concrete vectors \mathbb{C}^4 – all we want is a vector space with a basis labeled by the bitstrings of length two. Mathematicians write $\mathbb{C}^2 \otimes \mathbb{C}^2$ for precisely this – a four-dimensional vector space that comes with a distinguished basis labeled by the

bitstrings of length two, i.e., $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$ (instead of $|0\rangle$, $|1\rangle$, $|2\rangle$, $|3\rangle$, like it would be for \mathbb{C}^4). More generally:¹

Definition 3.1: Tensor product of vector spaces

We denote by

$$\underbrace{\mathbb{C}^2 \otimes \dots \otimes \mathbb{C}^2}_{n \text{ times}} = (\mathbb{C}^2)^{\otimes n}$$

the vector space of dimension 2^n that comes with a distinguished standard or *computational basis*

$$|\mathbf{x}\rangle = |x_1, \dots, x_n\rangle = |x_1 \dots x_n\rangle$$

that is labeled by the bitstrings $\mathbf{x} \in \{0, 1\}^n$ of length n . We call this the *tensor product* of n copies of \mathbb{C}^2 , or the *n-th tensor power* of \mathbb{C}^2 .

We analogously define the tensor product when the vector spaces have arbitrary dimensions, such as $\mathbb{C}^d \otimes \mathbb{C}^{d'}$ etc. See also the footnote.

Above and in the following we use a bold font to denote bitstrings.

Rule 1: State of n quantum bits

The state of n qubits is described by a unit vector in $\mathbb{C}^2 \otimes \dots \otimes \mathbb{C}^2 = (\mathbb{C}^2)^{\otimes n}$. That is, quantum states of n qubits are vectors

$$|\psi\rangle = \sum_{\mathbf{x} \in \{0,1\}^n} \psi_{\mathbf{x}} |\mathbf{x}\rangle = \psi_{0\dots 00} |0\dots 00\rangle + \psi_{0\dots 01} |0\dots 01\rangle + \dots + \psi_{1\dots 11} |1\dots 11\rangle$$

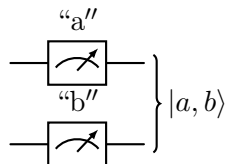
such that $\sum_{\mathbf{x} \in \{0,1\}^n} |\psi_{\mathbf{x}}|^2 = 1$.

You should think of the basis state $|\mathbf{x}\rangle = |x_1, \dots, x_n\rangle$ as: the first qubit is in state $|x_1\rangle$, the second qubit is in state $|x_2\rangle$, etc. Now the other rules we learned in [Chapter 2](#) generalize easily. We first explain what happens when one measures all qubits:

Rule 2: Measuring n quantum bits

Suppose n qubits are in state $|\psi\rangle \in (\mathbb{C}^2)^{\otimes n}$ and we measure *all* of them. Then, with probability $|\langle \mathbf{x} | \psi \rangle|^2 = |\psi_{\mathbf{x}}|^2$, we see outcomes “ x_1 ”, ..., “ x_n ”, in which case the state changes to $|\mathbf{x}\rangle = |x_1, \dots, x_n\rangle$.

You should really think of this as n separate measurements that you apply in parallel, like so:



Here we have $n = 2$ qubits, so we use two wires, and we label the outcomes a and b instead of x_1 and x_2 (just for the fun of it). By convention, the first (top) wire corresponds to the first qubit, and the second (bottom) wire corresponds to the bottom qubit. For example, if we measure both qubits of the two-qubit state

$$|\psi\rangle = \sqrt{0.3} |00\rangle + \sqrt{0.7} |11\rangle$$

then with 30% probability both measurements yield outcome “0” and the state changes to $|00\rangle$, and with 70% probability both measurements yield outcome “1” and the state changes to $|11\rangle$.

¹There is a more general and abstract definition of the tensor product of two vector spaces. If you know it: great! If you don't: what we explain in this chapter will be enough to understand all we do in this course.

Rule 3: Unitaries on n quantum bits

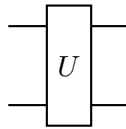
We can apply arbitrary unitaries to n qubits. If the state of the n qubits is $|\psi\rangle \in (\mathbb{C}^2)^{\otimes n}$ and we apply a unitary U on $(\mathbb{C}^2)^{\otimes n}$, then the state changes to $U|\psi\rangle$.

What is a unitary on $\mathbb{C}^2 \otimes \dots \otimes \mathbb{C}^2$? We can think of U either as an abstract *linear map* or *operator*, but also concretely as a $2^n \times 2^n$ matrix with respect to the computational basis:

$$U = \begin{pmatrix} U_{0\dots 00,0\dots 00} & \dots & U_{0\dots 00,1\dots 11} \\ \vdots & U_{\mathbf{x},\mathbf{y}} & \vdots \\ U_{1\dots 11,0\dots 00} & \dots & U_{1\dots 11,1\dots 11} \end{pmatrix}$$

Note that $U_{\mathbf{x},\mathbf{y}} = \langle \mathbf{x} | U | \mathbf{y} \rangle$. (Can you see why?) We say that U is unitary if this $2^n \times 2^n$ matrix is unitary. We always use lexicographic order to sort the computational basis (but we note that it does not actually matter in which order we sort this basis if all we want is to check if U is unitary).

We visualize n -qubit unitaries in the same way as before, except that we use n wires instead of just one. E.g., for $n = 2$:



Let's discuss some examples of important two-qubit unitaries.

Example 3.2: Swap gate

The *swap unitary* simply exchanges the state of the two qubits. It is clear how we should define it on the four basis states:

$$\begin{aligned} \text{SWAP} |00\rangle &= |00\rangle, \\ \text{SWAP} |01\rangle &= |10\rangle, \\ \text{SWAP} |10\rangle &= |01\rangle, \\ \text{SWAP} |11\rangle &= |11\rangle. \end{aligned}$$

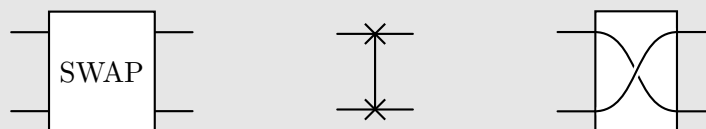
We can write this more succinctly as follows:

$$\text{SWAP} |a, b\rangle = |b, a\rangle \quad \text{for all } a, b \in \{0, 1\}.$$

The matrix representation of SWAP with respect to the computational basis $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ looks as follows:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

In quantum circuits, the following three notations are all quite usual:



Example 3.3: Controlled NOT gate

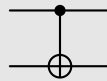
The *controlled NOT* or *CNOT* gate operates as you might expect on the four basis states:

$$\begin{aligned}\text{CNOT} |00\rangle &= |00\rangle, \\ \text{CNOT} |01\rangle &= |01\rangle, \\ \text{CNOT} |10\rangle &= |11\rangle, \\ \text{CNOT} |11\rangle &= |10\rangle\end{aligned}$$

or simply

$$\text{CNOT} |a, b\rangle = |a, a \oplus b\rangle \quad \text{for all } a, b \in \{0, 1\}. \quad (3.1)$$

In other words, if $a = 0$ then b is left untouched, while if $a = 1$ then b is inverted. We say that we apply a NOT gate to the second qubit, “controlled” on the first qubit. (Since we use X and NOT interchangeably, sometimes we also write CX instead of CNOT.) The CNOT gate looks as follows in quantum circuits, which is quite intuitive given what we just discussed:

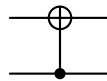


Here is its matrix representation:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \left(\begin{array}{c|c} I & 0 \\ \hline 0 & X \end{array} \right)$$

On the right-hand side we write CNOT as a 2×2 block matrix. Each block is itself a 2×2 -matrix.

Above we inverted the second qubit based on the value of the first one. We can also invert the first qubit based on the value of the second one. Naturally, this is denoted by



Can you write down a formula and matrix representation for this unitary?

Example 3.4: Controlled Z gate

Instead of the NOT gate, we can also apply other unitaries to the second qubit controlled on the first qubit. For example, the *controlled Z* gate is defined as follows on the basis states:

$$\begin{aligned}\text{CZ} |00\rangle &= |00\rangle, \\ \text{CZ} |01\rangle &= |01\rangle, \\ \text{CZ} |10\rangle &= |10\rangle, \\ \text{CZ} |11\rangle &= -|11\rangle.\end{aligned}$$

We can write this more succinctly:²

$$\text{CZ} |a, b\rangle = (-1)^{ab} |a, b\rangle \quad \text{for all } a, b \in \{0, 1\}.$$

Its matrix representation follows the same pattern:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} = \left(\begin{array}{c|c} I & 0 \\ \hline 0 & Z \end{array} \right)$$

Observe that, unlike for the CNOT gate, it does not matter which is the control and which is the target! Accordingly, the controlled Z also visualized in a symmetric way, as follows:



On the homework you will explore controlled gates more generally.

One important observation of [Rules 1 to 3](#) is that any two states that differ by an overall constant are indistinguishable, so they should really be considered “the same state”. To see this, suppose that

$$|\phi\rangle = \lambda |\psi\rangle.$$

for some $\lambda \in \mathbb{C}$. Since states have norm one, we must necessarily have $|\lambda| = 1$.³ If we measure the qubits of such a state, then we cannot see the difference, since the probabilities of measurement outcomes only depend on (the square of) the absolute value, and we have

$$|\langle \mathbf{x} | \phi \rangle|^2 = \underbrace{|\lambda|^2}_{=1} |\langle \mathbf{x} | \psi \rangle|^2 = |\langle \mathbf{x} | \psi \rangle|^2.$$

It also doesn’t help to apply unitaries before measuring, since

$$U |\phi\rangle = \lambda U |\psi\rangle$$

for every linear map U and hence $U |\phi\rangle$ and $U |\psi\rangle$ also only differ by an overall constant.

3.1.1 Some puzzles

So far so good! Is this all there is to know about quantum computing? Not quite. In ordinary computing, we usually apply gates to only some of the bits of our memory. But we don’t actually know how this works in quantum computing. Specifically, here are some important questions that we still have to address:

- Q1. Suppose Alice has one qubit in state $|\alpha\rangle \in \mathbb{C}^2$, and Bob has another qubit in state $|\beta\rangle \in \mathbb{C}^2$. How should we describe the joint state of their two qubits, which should be a vector $|\psi\rangle \in \mathbb{C}^2 \otimes \mathbb{C}^2$?
- Q2. Suppose Alice and Bob both have a qubit. If Alice applies a 2×2 unitary to her qubit alone, what does this do to their joint state?
- Q3. Suppose Alice and Bob both have a qubit. If Alice measures her qubit alone, then her outcome should be a bit. What are the probabilities of outcomes? What happens to the joint state depending on the outcome?

We’ll discuss these three questions in the following two sections.

3.2 Product states and entangled states

The answer to the first question is clear for basis states. If Alice has a qubit in state $|0\rangle$ and Bob has a qubit in state $|0\rangle$, then their joint state should simply be the basis state $|00\rangle$, and likewise for the other three combinations. Let us introduce a new operation “ \otimes ” that combines the basis states in this way:

$$|0\rangle \otimes |0\rangle := |00\rangle,$$

²Recall that $(-1)^n = 1$ if n is even and $(-1)^n = -1$ if n is odd. In particular, $(-1)^{ab} = -1$ if and only if $a = b = 1$.

³That is, $|\phi\rangle = e^{i\theta} |\psi\rangle$ for some angle θ . In the literature, people often say that the two states only differ by a “global phase”.

$$\begin{aligned} |0\rangle \otimes |1\rangle &:= |01\rangle, \\ |1\rangle \otimes |0\rangle &:= |10\rangle, \\ |1\rangle \otimes |1\rangle &:= |11\rangle. \end{aligned}$$

How should we combine states that are *not* basis states? Let's simply "multiply out"! This defines the following operation, which is known as the tensor product of vectors.

Definition 3.5: Tensor product of vectors

If $|\alpha\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle \in \mathbb{C}^2$ and $|\beta\rangle = \beta_0 |0\rangle + \beta_1 |1\rangle \in \mathbb{C}^2$, then we define their *tensor product* to be

$$\begin{aligned} |\alpha\rangle \otimes |\beta\rangle &= (\alpha_0 |0\rangle + \alpha_1 |1\rangle) \otimes (\beta_0 |0\rangle + \beta_1 |1\rangle) \\ &= \alpha_0\beta_0 |00\rangle + \alpha_0\beta_1 |01\rangle + \alpha_1\beta_0 |10\rangle + \alpha_1\beta_1 |11\rangle \in \mathbb{C}^2 \otimes \mathbb{C}^2. \end{aligned}$$

In the same we can define the tensor product of vectors of arbitrary dimension, e.g., when $|\alpha\rangle \in \mathbb{C}^d$ and $|\beta\rangle \in \mathbb{C}^{d'}$, or when $|\alpha\rangle$ or $|\beta\rangle$ themselves are vectors in a tensor product of vector spaces.

For example, recall the Hadamard basis states $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Then:

$$\begin{aligned} |+\rangle \otimes |+\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle), \\ |+\rangle \otimes |-\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{2}(|00\rangle - |01\rangle + |10\rangle - |11\rangle). \end{aligned}$$

Here is a more complicated example, in which we combine a one-qubit state with a two-qubit state to get a three-qubit state:

$$|0\rangle \otimes \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) = \frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|011\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |011\rangle).$$

If $|\alpha\rangle$ and $|\beta\rangle$ are unit vectors, then $|\alpha\rangle \otimes |\beta\rangle$ is again a unit vector. (Try proving this for yourself.) Thus the following rule makes sense and answers the first question (Q1).

Rule 4: Combining states

If the first qubit is in state $|\alpha\rangle \in \mathbb{C}^2$ and the second qubit is in state $|\beta\rangle \in \mathbb{C}^2$, then the overall state of the two qubits is $|\alpha\rangle \otimes |\beta\rangle \in \mathbb{C}^2 \otimes \mathbb{C}^2$. We can similarly combine states of more than one qubit (or qudit) each.

As a picture:

$$\left. \begin{array}{l} |\alpha\rangle \text{ ---} \\ |\beta\rangle \text{ ---} \end{array} \right\} |\alpha\rangle \otimes |\beta\rangle$$

Definition 3.6: Product and entangled states

A state $|\psi\rangle \in \mathbb{C}^2 \otimes \mathbb{C}^2$ is called a *product state* if it can be written in the form $|\psi\rangle = |\alpha\rangle \otimes |\beta\rangle$ for certain $|\alpha\rangle, |\beta\rangle \in \mathbb{C}^2$. Otherwise, $|\psi\rangle$ is called *entangled*.

We can't quite appreciate this right now, but entanglement plays a really important role in quantum information (roughly speaking, without entanglement, quantum computing would be no more powerful than ordinary computing). Here is an example of an entangled state:

Example 3.7: EPR pair

The following two-qubit state is called a (or “the”) *maximally entangled state* of two qubits, or an *EPR pair* (in honor to physicists Einstein, Podolski, and Rosen, who wrote an influential paper about the confusing properties of entangled states):

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle).$$

Why is it entangled? Suppose it is not! Then we could write it as a product state, i.e.,

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \stackrel{!}{=} |\alpha\rangle \otimes |\beta\rangle = \alpha_0\beta_0|00\rangle + \alpha_0\beta_1|01\rangle + \alpha_1\beta_0|10\rangle + \alpha_1\beta_1|11\rangle.$$

Then we would have

$$\frac{1}{2} = \frac{1}{\sqrt{2}} \cdot \frac{1}{\sqrt{2}} = (\alpha_0\beta_0)(\alpha_1\beta_1) = (\alpha_0\beta_1)(\alpha_1\beta_0) = 0 \cdot 0 = 0. \quad \zeta$$

But this is a contradiction. Thus the state is *not* a product state – instead, it is entangled.

3.3 Unitaries on subsystems

Now suppose that there are two qubits in some state $|\psi\rangle \in \mathbb{C}^2 \otimes \mathbb{C}^2$. Say, Alice has the first qubit and Bob the second one. If Alice applies a unitary U to her qubit (the first qubit), what will happen to the state $|\psi\rangle$?

The answer is clear for product states: if $|\psi\rangle = |\alpha\rangle \otimes |\beta\rangle$, then after applying U to the first qubit, the state should change to $U|\alpha\rangle \otimes |\beta\rangle$. You should read this as $(U|\alpha\rangle) \otimes |\beta\rangle$, and not as $U(|\alpha\rangle \otimes |\beta\rangle)$. In particular this tells us what happens with the basis states:

$$\begin{aligned} |00\rangle = |0\rangle \otimes |0\rangle &\mapsto U|0\rangle \otimes |0\rangle, \\ |01\rangle = |0\rangle \otimes |1\rangle &\mapsto U|0\rangle \otimes |1\rangle, \\ |10\rangle = |1\rangle \otimes |0\rangle &\mapsto U|1\rangle \otimes |0\rangle, \\ |11\rangle = |1\rangle \otimes |1\rangle &\mapsto U|1\rangle \otimes |1\rangle. \end{aligned}$$

Since a linear map is determined by its action on a basis, the above rules define a linear map on $\mathbb{C}^2 \otimes \mathbb{C}^2$ that we can apply to arbitrary states, not only product states. For example:

$$|00\rangle + |11\rangle \mapsto U|0\rangle \otimes |0\rangle + U|1\rangle \otimes |1\rangle.$$

The linear map thus defined is denoted by $U \otimes I$, which is a special case of a more general definition (the third kind of tensor product that we will learn today).

Definition 3.8: Tensor product of linear maps (“operators”)

If A and B are linear maps on \mathbb{C}^2 (i.e., 2×2 -matrices), then their *tensor product* $A \otimes B$ is the unique linear map on $\mathbb{C}^2 \otimes \mathbb{C}^2$ such that

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

for all $|\alpha\rangle, |\beta\rangle \in \mathbb{C}^2$. The right-hand side of the equation should be interpreted as $(A|\alpha\rangle) \otimes (B|\beta\rangle)$. Note that this defines the tensor product of operators in terms of the tensor product of vectors!

In the same way we define the tensor product of arbitrary linear maps between arbitrary vector spaces. We can even make sense of objects such as $\langle\gamma| \otimes B$, which is the linear map from $\mathbb{C}^2 \otimes \mathbb{C}^2$ to \mathbb{C}^2 defined by

$$(\langle\gamma| \otimes B)(|\alpha\rangle \otimes |\beta\rangle) = \langle\gamma|\alpha\rangle \cdot B|\beta\rangle \tag{3.2}$$

for all $|\alpha\rangle, |\beta\rangle \in \mathbb{C}^2$. Note that $\langle\gamma|\alpha\rangle$ is a number and hence the right-hand side is a vector in \mathbb{C}^2 .

The tensor product is bilinear, associative, and satisfies some other useful rules. For example, $(A \otimes B)(A' \otimes B') = AA' \otimes BB'$ whenever it makes sense. Now we can summarize the above discussion.

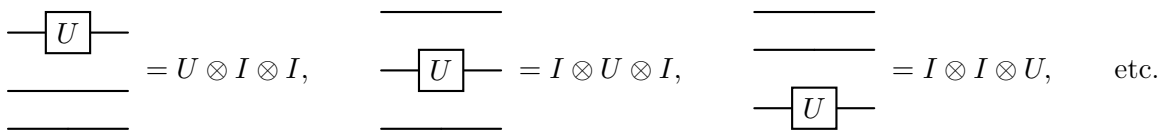
Rule 5: Unitaries on some of the qubits

If we apply a one-qubit unitary U to the first qubit of two qubits in state $|\psi\rangle$, then the result is $(U \otimes I)|\psi\rangle$. If we instead apply U to the second qubit, then the result is $(I \otimes U)|\psi\rangle$. We can similarly apply unitaries (of one or more qubits) in situations when we have more than two qubits.

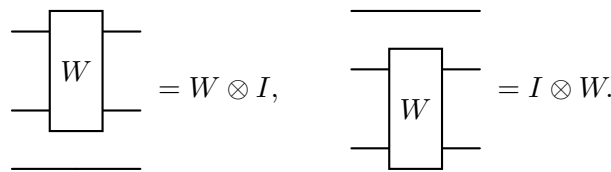
This answers question Q2. In pictures:



That is, if we apply the left-hand side circuit to an arbitrary two-qubit state $|\psi\rangle$ then we get $(U \otimes I)|\psi\rangle$, and similarly for the right-hand side one. Similarly:



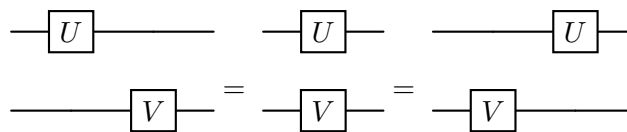
Or if W is a two-qubit unitary, then we have



As a consequence of the general rule mentioned above, we have:

$$(I \otimes V)(U \otimes I) = U \otimes V = (U \otimes I)(I \otimes V).$$

This tells us that it does not matter in which order we apply unitaries on different qubits! This is quite intuitive:

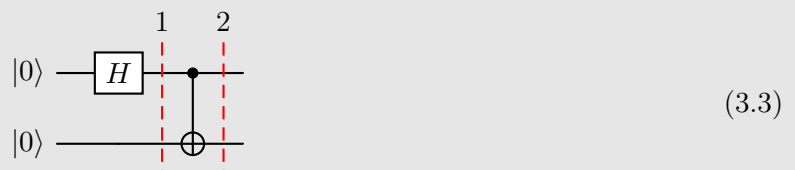


(Caution: circuit time goes left to right, while linear maps are applied right to left).

Let's discuss an interesting example:

Example 3.9: EPR circuit

Which state is created by the following quantum circuit?



By Rule 4, we start out in state $|00\rangle = |0\rangle \otimes |0\rangle$. We first apply a Hadamard gate on the first qubit. By

Rule 5, this means that the state changes to

$$(H \otimes I) |00\rangle = H |0\rangle \otimes |0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle).$$

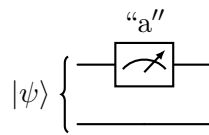
Finally, we apply the CNOT gate. By [Eq. \(3.1\)](#), the result is

$$\text{CNOT} \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle) = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle).$$

Thus, the circuit in [\(3.3\)](#) prepares the maximally entangled or EPR state from [Example 3.7](#).

3.4 Measurements on subsystems

The last question we need to address is what happens if we measure only some of the qubits but not all of them (Q3). For example, suppose again that there are two qubits in some state $|\psi\rangle \in \mathbb{C}^2 \otimes \mathbb{C}^2$. What happens if we only measure the first qubit? This is called a *partial measurement*:



The outcome should be a bit – but what are the probabilities of the two possible outcomes $a \in \{0, 1\}$? And what is the state afterwards? Again the answer is clear if the state is a product state $|\alpha\rangle \otimes |\beta\rangle$, but how about in general?

It is not so hard to figure out what the probabilities should be. Suppose instead of just measuring the first qubit, you measure both qubits, but you only care about the outcome of the first qubit. Since the joint probability of both outcomes is $|\psi_{ab}|^2$, by [Rule 2](#), the probability of seeing some outcome “ a ” for the measurement of the first qubit is computed by the marginal probability

$$p_a = |\psi_{a0}|^2 + |\psi_{a1}|^2.$$

It turns out that this is exactly the probability of outcome $a \in \{0, 1\}$ if we only measure the first qubit.

What should happen with the state? If we measure both qubits then we know that the state collapses into a basis state $|a, b\rangle$, corresponding to the outcomes that we see. If we only measure one qubit then we might guess that the state will “partially” collapse, meaning that we should keep precisely those terms that are consistent with the outcome that we see. That is, if we measure the first qubit and see outcome “ a ”, then the state should change to $\psi_{a0} |a0\rangle + \psi_{a1} |a1\rangle$. This is almost correct, except that this vector will not have norm one. How can we make it a unit vector? We simply divide by its norm. Since $\|\psi_{a0} |a0\rangle + \psi_{a1} |a1\rangle\|^2 = |\psi_{a0}|^2 + |\psi_{a1}|^2 = p_a$, the norm that we need to divide by is simply $\sqrt{p_a}$. Thus we arrive at the following rule for partial measurements:

Rule 6: Partial measurement

If two qubits are in state $|\psi\rangle = \psi_{00} |00\rangle + \psi_{01} |01\rangle + \psi_{10} |10\rangle + \psi_{11} |11\rangle$ and we measure the first qubit, then the following happens: for $a \in \{0, 1\}$, with probability

$$p_a = |\psi_{a0}|^2 + |\psi_{a1}|^2$$

we see outcome “ a ”, in which case the state changes to

$$\frac{1}{\sqrt{p_a}}(\psi_{a0} |a0\rangle + \psi_{a1} |a1\rangle) = |a\rangle \otimes \frac{1}{\sqrt{p_a}}(\psi_{a0} |0\rangle + \psi_{a1} |1\rangle).$$

Analogously we can compute what happens if we measure only the second qubit, or if we have more qubits, or if we measure multiple qubits, or if we have qudits, etc.

This answers question (Q3). It is a fun exercise to check that if you measure one qubit after the other (using [Rule 6](#) each time) then you obtain precisely [Rule 2](#).

Remark 3.10

Here is another way of stating [Rule 6](#) that might at first glance look a more abstract, but which I find easy to remember and use in practice: For concreteness, suppose you have a two-qubit state $|\psi\rangle \in \mathbb{C}^2 \otimes \mathbb{C}^2$ and you want to know what happens when you measure the first qubit. To this end, write your state as

$$|\psi\rangle = |0\rangle \otimes |\psi_0\rangle + |1\rangle \otimes |\psi_1\rangle,$$

for vectors $|\psi_0\rangle$ and $|\psi_1\rangle$ (in general these will *not* be unit vectors). This is often very easy to do by hand. Alternatively you can simply compute $|\psi_a\rangle = (\langle a| \otimes I) |\psi\rangle$ for each $a \in \{0, 1\}$ (see [Eq. \(3.2\)](#) if you are confused by this notation). Then the probability of outcome “ a ” is

$$p_a = \|\psi_a\|^2,$$

and if one sees this outcome then the two-qubit state changes to

$$|a\rangle \otimes \frac{|\psi_a\rangle}{\sqrt{p_a}} = |a\rangle \otimes \frac{|\psi_a\rangle}{\|\psi_a\|}.$$

Here is a variation of the above prescription that can also be useful: if instead you write

$$|\psi\rangle = \beta_0 |0\rangle \otimes |\phi_0\rangle + \beta_1 |1\rangle \otimes |\phi_1\rangle,$$

for complex numbers $\beta_0, \beta_1 \in \mathbb{C}$ and *unit vectors* $|\phi_0\rangle$ and $|\phi_1\rangle$, then the probability of outcome “ a ” is simply

$$p_a = |\beta_a|^2$$

and if one sees this outcome then the state changes to

$$|a\rangle \otimes |\phi_a\rangle.$$

All the above generalizes readily to measuring the second qubit, to more than two qubits, etc. You can discuss this in more detail in the exercises.

Phew! Now we know all the rules of quantum computing that we will need. From the next chapter onwards, we will have fun applying them.

Chapter 4

Classical vs quantum computing and oracles

In this lecture, we will start discussing how to compute with quantum circuits. The first topic that we need to discuss is whether quantum computing is even just as powerful as ordinary classical computing! To address this, we will show how to construct for any Boolean circuit a corresponding quantum circuit that allows us to compute the same result (and is at most a constant factor larger than the circuit we started with). Afterwards, we will discuss oracle problems and see a first example where quantum computing solve a problem more efficiently than any classical algorithm.

4.1 What do quantum circuits and algorithms look like?

Quantum algorithms are often thought of in terms of circuits. A typical *quantum circuit* looks as follows:

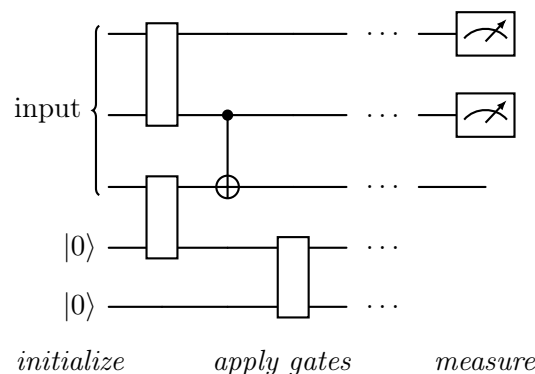


Figure 4.1: Anatomy of a typical quantum circuit

We start with some qubits that contain our problem input (e.g., if the task is to check whether a number is prime, then the input qubits might be initialized in a basis state $|x_1, \dots, x_n\rangle$ that encodes the number in binary). In addition, there might be some auxiliary qubits initialized in the $|0\rangle$ state (our “scratch space”). Then we successively apply gates. We will only consider unitary gates that act on one, two, or three qubits at a time (such as the ones discussed in [Chapters 2 and 3](#)).¹ At the end we measure one or more qubits to hopefully obtain the desired answer. It is also okay to have measurements in the middle of the computation – this is called an “intermediate measurement” –, although you will show in the exercises that measurements can always be postponed to the end.

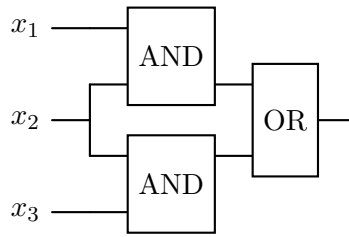
Usually, there is not just a single circuit, but the circuit depends on the problem size (e.g., the bitsize of the number that we want to factor). To get a *quantum algorithm*, we need to be able to generate this circuit for any possible problem size. To this end, we ask that there is a classical algorithm that tells us which quantum gates should be applied in which order (it can also take the outcomes of intermediate measurements into account to decide which gates should be applied next).²

¹In fact, there are *universal gate sets*, meaning that any unitary can be approximated to arbitrary precision by composing gates in the set. For example, the H , T , and CNOT gates defined in [Eqs. \(2.5\), \(2.6\) and \(3.1\)](#) are universal in this sense!

²We will not be more precise than this for the time being, since it will always be clear in all of our examples what the classical

4.2 From classical to reversible and quantum circuits

Suppose someone hands you a Boolean circuit computing some function $g: \{0, 1\}^n \rightarrow \{0, 1\}$ and asks you to construct a corresponding quantum circuit that allows computing the same result. (If you aren't sure what this should even mean, this is *good* – please continue reading!) For example, they might hand you the following Boolean circuit:



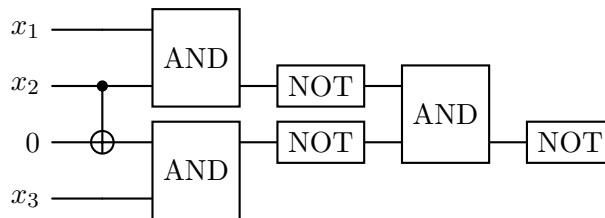
It computes the function $g(x_1, x_2, x_3) = \text{OR}(\text{AND}(x_1, x_2), \text{AND}(x_2, x_3))$.

Perhaps we can simply go ahead and replace each gate by a quantum gate? Unfortunately, for the AND and OR gates this will not work, for the interesting reason that those gates are irreversible, while unitary quantum gates can always be reversed. For example, we cannot simply define a “quantum AND gate” that maps $|x_1, x_2\rangle \mapsto |\text{AND}(x_1, x_2)\rangle$, since this would need to send $|00\rangle$, $|10\rangle$, and $|01\rangle$ to $|0\rangle$, which is clearly not reversible! Similarly, splitting the x_2 wire into two in order to copy the x_2 bit is not bijective either.

Note that this problem has nothing to do with quantum computing per se. What we are really asking is whether it is possible to compute any Boolean function by a reversible circuit (and hopefully so without any loss of efficiency). Let us first simplify the problem in two ways:

- We apply De Morgan’s law that says that $\text{OR}(y_1, y_2) = \text{NOT}(\text{AND}(\text{NOT}(y_1), \text{NOT}(y_2)))$;
- Instead of splitting the wire to copy the x_2 bit, we add an auxiliary 0 bit and CNOT x_2 onto it.

This results in the following Boolean circuit:



Since the NOT and CNOT gates are reversible, we are left with the problem that the AND gate is not reversible. Fortunately, there is a general “trick” to make any function reversible:

Definition 4.1: Reversible oracle

For a function $\mathbf{f}: \{0, 1\}^n \rightarrow \{0, 1\}^m$ with n input and m output bits, the *reversible oracle* (or *reversible embedding*) is the function $\mathbf{f}_{\text{rev}}: \{0, 1\}^{n+m} \rightarrow \{0, 1\}^{n+m}$ defined by $\mathbf{f}_{\text{rev}}(\mathbf{x}, \mathbf{y}) = (\mathbf{x}, \mathbf{y} \oplus \mathbf{f}(\mathbf{x}))$.

It is easy to pass between \mathbf{f} and its reversible oracle. Indeed, the latter is easy to compute from the former, and we can also recover the former from the latter: simply compute $\mathbf{f}_{\text{rev}}(\mathbf{x}, \mathbf{0}) = (\mathbf{x}, \mathbf{f}(\mathbf{x}))$ and look at the second part of the output. Note that \mathbf{f}_{rev} is indeed reversible. In fact, it is its own inverse: for all $\mathbf{x} \in \{0, 1\}^n$ and $\mathbf{y} \in \{0, 1\}^m$, we have

$$\mathbf{f}_{\text{rev}}(\mathbf{f}_{\text{rev}}(\mathbf{x}, \mathbf{y})) = \mathbf{f}_{\text{rev}}(\mathbf{x}, \mathbf{y} \oplus \mathbf{f}(\mathbf{x})) = (\mathbf{x}, \mathbf{y} \oplus \mathbf{f}(\mathbf{x}) \oplus \mathbf{f}(\mathbf{x})) = (\mathbf{x}, \mathbf{y}).$$

If we apply this to the AND gate, we get $\text{AND}_{\text{rev}}(x_1, x_2, y) = (x_1, x_2, y \oplus \text{AND}(x_1, x_2))$. This is called the Toffoli gate and it deserves its own definition.

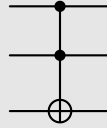
algorithm looks like that tells us which gates to apply when. However, we will have to be more precise when discussing quantum complexity theory.

Example 4.2: Classical Toffoli gate

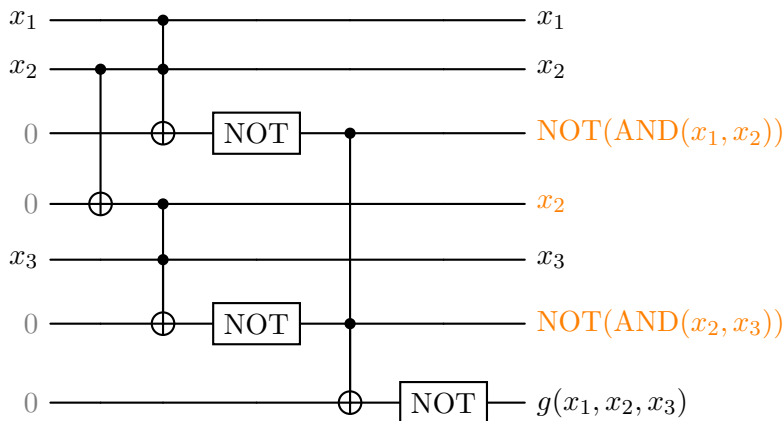
The (*classical*) *Toffoli gate* is the reversible three-bit gate defined as follows:

$$\text{CCNOT}: \{0, 1\}^3 \rightarrow \{0, 1\}^3, \quad (x_1, x_2, y) \mapsto (x_1, x_2, y \oplus \text{AND}(x_1, x_2))$$

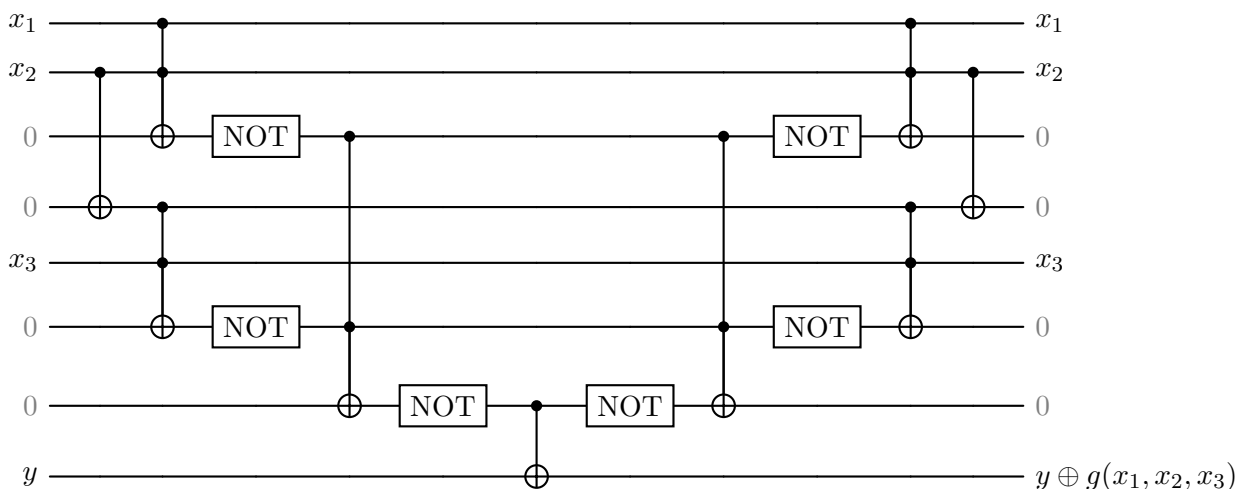
Note that it simply inverts the third bit if both the first and the second bit are true. Accordingly, it is also called the *CCNOT gate*, which is short for doubly controlled NOT gate. It is denoted as follows in circuits:



If we replace the AND gates in our circuit by Toffoli gates, with the third input bit (“ y ”) initialized to 0, we obtain the following circuit:³



On the right-hand side we show what is output on each wire. It looks like we have succeeded – the desired result is output on the very last wire. On the other hand, some of the other wires are outputting some intermediate results and the copy of x_2 (colored in orange) that we are not interested in. Fortunately, there is an easy way to get rid of this “garbage”: let us simply use another CNOT to copy the result onto a new wire and then run the above circuit in reverse (this is sometimes called “uncomputing”). Because we are only using gates that are their own inverses, we simply need to reverse the order in the second part:



We see that the resulting circuit precisely computes the reversible oracle g_{rev} of the function g ! While it uses some auxiliary bits initialized in state 0 to achieve this, it restores the state of these bits in state 0 so that they can be used again later – thus the “garbage” does not accumulate. Phew!

³This circuit can be simplified, but let’s not worry about this for the time being – the goal is to try to derive a completely mechanical procedure that works without any thinking!

The above procedure works in complete generality. Given a Boolean circuit computing any function $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$, we can compile it into a reversible circuit that computes the reversible oracle $f_{\text{rev}}: \{0, 1\}^{n+m} \rightarrow \{0, 1\}^{n+m}$, by the following steps:

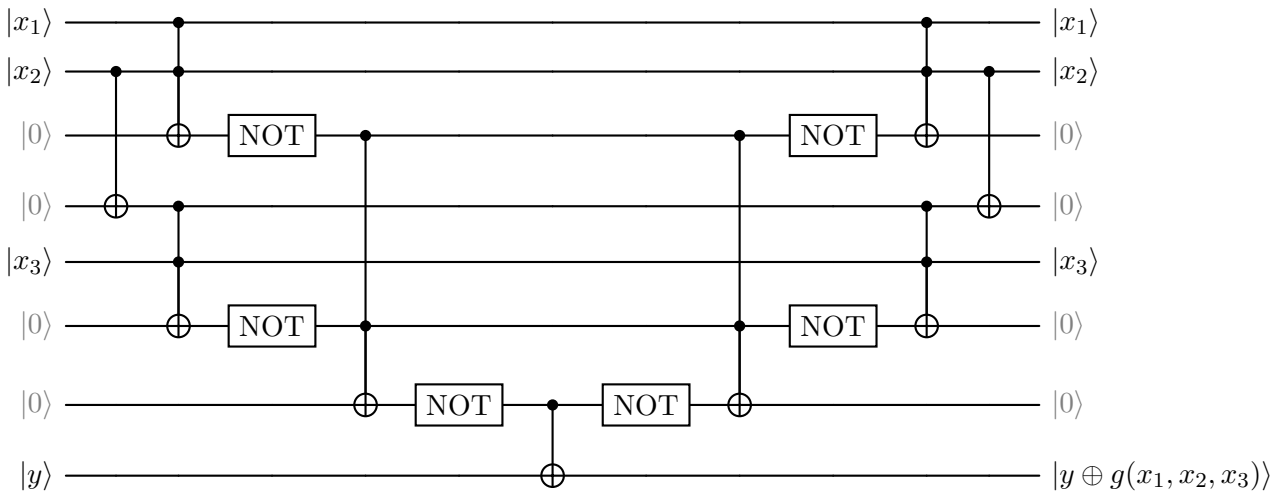
1. First preprocess the circuit so that it only consists of NOT, CNOT, and AND gates.
2. Replace each AND gate by a Toffoli (CCNOT) gate, with the third input a fresh bit that should be initialized to 0.
3. Use CNOT gates to copy the output bits onto fresh bits.
4. Finally, run step 2 in reverse.

We summarize the properties of this compilation in the following theorem:

Theorem 4.3: Compiling to reversible circuits

Given a Boolean circuit that computes some function $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$, the above procedure yields a reversible circuit that computes the reversible oracle $f_{\text{rev}}: \{0, 1\}^{n+m} \rightarrow \{0, 1\}^{n+m}$ and uses only NOT, CNOT, and Toffoli gates. Moreover, if the original circuit consists of T gates, the new circuit will consist of $O(T)$ gates and use $O(T)$ additional bits (which must be initialized in the 0 state, but will be returned in the 0 state at the end of the circuit).

Now that we have obtained a reversible circuit it is clear how to get a quantum circuit. Simply replace each classical NOT, CNOT, and Toffoli gate by the corresponding quantum gates, and instead of initializing the wires with 0 bits, initialize them with $|0\rangle$. In our running example, this will look as follows (while this looks almost identically as before, the circuit is now a quantum circuit):



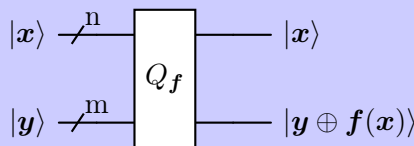
In general, this will implement the following functionality:

Definition 4.4: Standard quantum oracle

For a function $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$ with n input and m output bits, the *standard quantum oracle* is the unitary Q_f on $(\mathbb{C}^2)^{\otimes n} \otimes (\mathbb{C}^2)^{\otimes m}$ defined as follows on the computational basis:

$$Q_f |x, y\rangle = |x, y \oplus f(x)\rangle \quad \text{for all } x \in \{0, 1\}^n \text{ and } y \in \{0, 1\}^m$$

That is:



In other words, $Q_f |x, y\rangle = |f_{\text{rev}}(x, y)\rangle$.

For completeness, we also define the quantum Toffoli gate.

Example 4.5: Quantum Toffoli gate

The (*quantum*) Toffoli gate is the three-qubit gate that acts as follows on the computational basis:

$$\text{CCNOT} |x_1, x_2, y\rangle = |x_1, x_2, y \oplus \text{AND}(x_1, x_2)\rangle.$$

When used in circuits it is denoted in the same way as the classical Toffoli gate.

Remark 4.6

Interestingly, while it is not possible to compile the classical Toffoli gate into reversible one-bit and two-bit gates, it *is* possible to compile the quantum Toffoli gate into one-qubit and two-qubit gates. See [here](#) for how to do it using H , T and CNOT.

Then we obtain the following result as a direct consequence of [Theorem 4.3](#):

Corollary 4.7: Compiling to quantum circuits

Given a Boolean circuit that computes some function $f: \{0, 1\}^n \rightarrow \{0, 1\}^m$, the above procedure yields a quantum circuit that computes the standard quantum oracle Q_f and uses only NOT, CNOT, and Toffoli quantum gates. Moreover, if the original circuit consists of T gates, the new circuit will consist of $O(T)$ gates and use $O(T)$ additional qubits (which must be initialized in the $|0\rangle$ state, but will be returned in the $|0\rangle$ state at the end of the circuit).

The punchline of this section is this: *the cost of computing a function reversibly or quantumly is essentially of the same order as if we compute the function using ordinary Boolean circuits.*

Remark 4.8

So far we only discussed how to convert deterministic Boolean circuits into quantum circuits. What if our Boolean circuit also uses some randomness? This can easily be simulated quantumly, since we can always prepare a $|+\rangle = H|0\rangle$ state and measure it to get a uniformly random basis state $|0\rangle$ or $|1\rangle$.

4.3 A first quantum advantage: Deutsch's problem and algorithm

We just saw that the quantum oracle Q_f for a Boolean function f can be implemented at essentially the same cost as the classical one. Clearly, it is at least as useful as f , since we can always apply Q_f to $|\mathbf{x}, \mathbf{0}\rangle$ and measure to obtain $f(\mathbf{x})$. Is it more powerful? It turns out that this is indeed the case! We will now see a first such example (which you will generalize in the exercises).

Problem 4.9: Deutsch's problem

Given a function $g: \{0, 1\} \rightarrow \{0, 1\}$, decide if it is constant or not.

The question is of course how the function is given to us. Here we want to imagine that we only have the ability to evaluate the function g . That is, we want to think of g as an *oracle* or *black box*, that is, a subroutine that we can call or *query*, but know nothing about its implementation.

Clearly, any classical algorithm needs *at least two queries* to g to solve this problem. This is because if we only know $g(0)$, then $g(1)$ is still completely arbitrary, and vice versa. Randomness does not help either. On the other hand, two queries are enough, since of course we can simply query $g(0)$ and $g(1)$ and decide based on that, as in the following pseudocode:

```
def is_constant(g):  
    return g(0) == g(1)
```

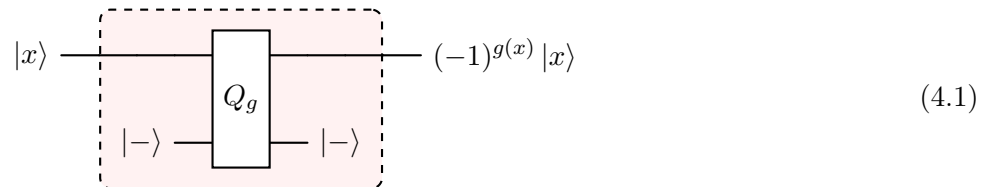
We will now see that there exists a quantum algorithm that can solve [Problem 4.9](#) using *only a single query* to the quantum oracle Q_g ! To get some intuition, let us see what happens if we apply Q_g with a Hadamard basis state in the second qubit. For $|+\rangle$, nothing happens at all, since for $x \in \{0, 1\}$ we have

$$Q_g |x, +\rangle = \frac{1}{\sqrt{2}}(Q_g |x, 0\rangle + Q_g |x, 1\rangle) = \frac{1}{\sqrt{2}}(|x, g(x)\rangle + |x, 1 \oplus g(x)\rangle) = |x, +\rangle.$$

This is perhaps strange, but not useful. However, if we instead use $|-\rangle$ then, for $x \in \{0, 1\}$, we get

$$Q_g |x, -\rangle = \frac{1}{\sqrt{2}}(Q_g |x, 0\rangle - Q_g |x, 1\rangle) = \frac{1}{\sqrt{2}}(|x, g(x)\rangle - |x, 1 \oplus g(x)\rangle) = (-1)^{g(x)} |x, -\rangle.$$

That is, if we input $|x\rangle$ in the first register and $|-\rangle$ in the second one, then the state remains the same if $g(x) = 0$, but it gets a minus sign if $g(x) = 1$. This is called the “phase kickback trick”. In pictures:



Note that the pink box implements a unitary that maps $|x\rangle$ to $(-1)^{g(x)} |x\rangle$. This is so useful that it deserves its own name. We call it the phase oracle, and it can be defined for any function $g: \{0, 1\}^n \rightarrow \{0, 1\}$ (with *one* output bit).

Definition 4.10: Quantum phase oracle

For a function $g: \{0, 1\}^n \rightarrow \{0, 1\}$ with n input bits and *one* output bit, the (*quantum*) *phase oracle* is the unitary P_g on $(\mathbb{C}^2)^{\otimes n}$ defined as follows on the computational basis:

$$P_g |\mathbf{x}\rangle = (-1)^{g(\mathbf{x})} |\mathbf{x}\rangle.$$

That is:



As just discussed, we can implement the phase oracle using one query to the standard quantum oracle. It is easy to see that this works not just for $n = 1$, but in fact for any n .

We will now see that by using it once we can solve Deutsch’s problem. Without further ado, here is the circuit:



Let’s compute the state step by step:

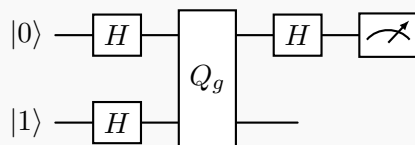
$$|0\rangle \mapsto |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \mapsto \frac{1}{\sqrt{2}}\left((-1)^{g(0)} |0\rangle + (-1)^{g(1)} |1\rangle\right) \mapsto \begin{cases} \pm |0\rangle & \text{if } g(0) = g(1) \\ \pm |1\rangle & \text{otherwise.} \end{cases}$$

Thus the measurement at the end will always yield outcome “0” if the function is constant, and “1” otherwise. Accordingly, one quantum query (to either P_g or Q_g) suffices to solve Deutsch’s problem!

Remark 4.11: Deutsch’s circuit in terms of the standard quantum oracle

If you find the phase oracle mysterious, you can also consider the following quantum circuit which uses

the standard quantum oracle instead of the phase oracle:



This circuit is completely equivalent to (4.2). To see this, note that we get it from (4.2) by replacing P_g with the pink box in (4.1), which implements it, and preparing the $|-\rangle$ state by starting with a $|1\rangle$ basis state and applying a Hadamard, since $H|1\rangle = |-\rangle$.

To summarize, the above quantum circuit shows clearly that the quantum oracles can offer an advantage. However, a factor two is perhaps not a very big advantage (especially because quantum oracles will typically be a bit more costly to implement, see Section 4.2). In the coming weeks, we will study more convincing problems where quantum algorithms offer an advantage – sometimes even an exponential one.

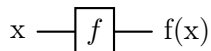
4.4 Outro: Time vs query complexity

We just saw a problem that took as its input not a bitstring, but an oracle or black box that it could call. Let us discuss some computational problems, how their input is specified, and how to quantify their complexity, in order to see that the above setting is quite natural:

- *Factoring a number*: Here the input is just a number which we would typically input in binary.
- *Minimize a function*: Here it is not so clear how the input should be given!

One option would be to restrict to some class of functions which can be specified by some type of formula. E.g., if we want to restrict to minimizing only polynomials, such as $f(x) = x^2 - 3x + 5$, we could input the sequence of coefficients 1, -3, 5 in binary.

The following option is often more natural: We could simply give our algorithm the ability to evaluate the function $f(x)$ for any x of choice:



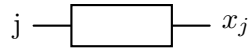
In other words, the input is given by an *oracle* or *black box*, that is, a subroutine that we can call but know nothing about its implementation.

For example, if you want to minimize an arbitrary convex function then *gradient descent* manages. It only needs to be able to evaluate a function and its gradient (derivative). Moreover, the latter can be estimated using finite differences. Here is some pseudo-code for it:

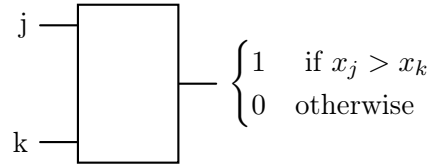
```
def gradient_descent_1d(f):
    step_size = ... # suitable step size
    T = ...        # suitable number of iterations
    x = x_0       # some starting point
    eps = 1e-5

    for t in range(T):
        grad = (f(x + eps) - f(x)) / eps
        x = x - step_size * grad
    return x
```

- *Find an element in an array*: Again, we could either specify the array by listing one element after the other, x_1, \dots, x_N , or we could provide access to a black box that inputs some index j and returns the j -th element of the array:



- *Sort an array*: Note that specifying the array is not enough – we also need to know how to compare elements. Once we realize this, we see that knowing the elements is not necessary at all. All we need to provide to our sorting algorithm is a black box that given two indices, tells us which of the corresponding entries of the array is larger:



Indeed, most search algorithms that you know are generic and work in this model.

There are many other examples. For examples, in computer graphics, we can describe geometric objects explicitly (e.g., by lists of triangles) or implicitly (e.g., by distance fields – functions that for any point compute the distance to the object).

From these examples we see that there are two natural ways of specifying the input for a computational problem:

- The input is given *explicitly* as a number, formula, array, etc. (usually encoded in binary, as a bitstring).
- The input is given by a *black box* or *oracle* that can be called or *queried* (that is, it is given by a subroutine that we can use in our algorithm, but we know nothing about its implementation).

Problems where the input involves an oracle are called *oracle problems*.

Accordingly there are different ways of quantifying the efficiency or *complexity* of an algorithmic *solution*:

- We can count the number of gates/elementary operations that are required to solve the problem. This is called *time complexity*.
- When the input is given by an oracle, we can also only count the number of queries (= calls) to the oracle that is required to solve the problem. In other words, we consider circuits that consist of ordinary gates and oracle queries, and we only count the latter. This is called *query complexity*.

Deutsch’s problem (Problem 4.9) is an example of an oracle problem, and we just saw that its quantum query complexity is one, while its classical query complexity is two.

Clearly, for oracle problems, the query complexity is always \leq the time complexity. Thus, in order to prove that an oracle problem is difficult, it is enough to show that many queries are needed to solve the problem. We will learn techniques for doing so in this course. On the other hand, if an oracle problem is easy, then one often (but not always!) finds that its time complexity is of the same order than the query complexity.

Chapter 5

Simon's quantum algorithm and classical lower bounds

Last week we discussed the Deutsch-Jozsa quantum algorithm, which gave an *exponential* speedup over any deterministic classical algorithm, and the Bernstein-Vazirani quantum algorithm, which offered a linear speedup over any possibly *randomized* classical algorithm.

This week, we will discuss Simon's quantum algorithm, which combines these two features: it offers an *exponential* speedup for a certain computational problem as compared to any possibly *randomized* classical algorithm! The problem looks as follows (we write $\mathbf{0}$ for the bitstring consisting of n zeros):

Problem 5.1: Simon's problem

You are given query access to a function $f: \{0, 1\}^n \rightarrow \{0, 1\}^n$ that is *periodic*, i.e., there is some $\mathbf{0} \neq \mathbf{s} \in \{0, 1\}^n$ such that $f(\mathbf{x}) = f(\mathbf{y})$ if and only if $\mathbf{y} = \mathbf{x}$ or $\mathbf{y} = \mathbf{x} \oplus \mathbf{s}$. Find the period \mathbf{s} !

Note that the function f is *two-to-one*. Indeed, for any \mathbf{z} in the image $f(\{0, 1\}^n)$, we have $f^{-1}(\mathbf{z}) = \{\mathbf{x}, \mathbf{x} \oplus \mathbf{s}\}$ for some $\mathbf{x} \in \{0, 1\}^n$. In particular, the image of f consists of 2^{n-1} distinct bitstrings.

We assume that we have query access to an oracle for the function f . As always, a classical query is simply a function call $f(\mathbf{x})$. Note that the function has n output bits; we learn all of them in a single query. For quantum queries we use the standard quantum oracle, which is the $2n$ -qubit unitary

$$Q_f |\mathbf{x}, \mathbf{z}\rangle = |\mathbf{x}, \mathbf{z} \oplus f(\mathbf{x})\rangle \quad \forall \mathbf{x}, \mathbf{z} \in \{0, 1\}^n. \quad (5.1)$$

(Since the function f has n output bits there is no quantum phase oracle in this case.)

5.1 Classical algorithm

To solve Simon's problem it suffices to find a collision, i.e., $\mathbf{x} \neq \mathbf{x}'$ such that $f(\mathbf{x}) = f(\mathbf{x}')$. Indeed, if we find such a collision then we know that $\mathbf{s} = \mathbf{x} \oplus \mathbf{x}'$.

This suggests a simple algorithm: simply draw random distinct bitstrings $\mathbf{x}(1) \neq \mathbf{x}(2) \neq \dots \neq \mathbf{x}(T) \in \{0, 1\}^n$, and hope for a collision. The "birthday paradox" suggests that $T = O(\sqrt{2^n})$ queries should suffice.

To get some intuition, we compute the *expected* number of collisions. Since there are $\binom{T}{2}$ pairs $\{i, j\}$, and for each pair the probability that we have a collision (i.e., $f(\mathbf{x}(i)) = f(\mathbf{x}(j))$) is $\frac{1}{2^n - 1}$, the expected number of collision is

$$\mathbf{E}[\text{collisions}] = \binom{T}{2} \frac{1}{2^n - 1} = \Theta\left(\frac{T^2}{2^n}\right).$$

Thus, $T = O(\sqrt{2^n})$ queries suffice so that we see a collision in expectation. This looks good, but we really want to find a collision with high probability (and the preceding does not show this!). What is the probability that there is *no* collision?

$$\Pr(\text{no collision within } T \text{ queries})$$

$$\begin{aligned}
&= \prod_{t=1}^{T-1} \Pr(\text{no collision in } (t+1)\text{-st query} \mid \text{no collision within first } t \text{ queries}) \\
&= \prod_{t=1}^{T-1} \left(1 - \frac{t}{2^n - t}\right) \leq \prod_{t=1}^{T-1} \left(1 - \frac{t}{2^n}\right) \leq \prod_{t=1}^{T-1} e^{-\frac{t}{2^n}} \\
&= e^{-\sum_{t=1}^{T-1} \frac{t}{2^n}} = e^{-\frac{(T-1)T}{2} \frac{1}{2^n}} = e^{-\Theta(T^2/2^n)}
\end{aligned}$$

since for the $(t+1)$ -st query there are $2^n - t$ remaining bitstrings to draw from (remember that the queries that we make are distinct) and precisely t out of those lead to a collision, namely $\mathbf{x}(t+1) = \mathbf{x}(k) \oplus \mathbf{s}$ for $k \in [t]$. The second inequality uses $1 + x \leq e^x$ which holds for all $x \in \mathbb{R}$. It follows that $T = O(\sqrt{2^n})$ queries suffice to find a collision with any desired constant probability.

Thus we obtain the following randomized algorithm, which works with any desired constant success probability: Query $T = O(\sqrt{2^n})$ function values. If we find a collision $\mathbf{f}(\mathbf{x}(i)) = \mathbf{f}(\mathbf{x}(j))$ for some $i \neq j \in [T]$, then we output $\mathbf{s} = \mathbf{x}(i) \oplus \mathbf{x}(j)$. Otherwise, report an error (or return some arbitrary guess).

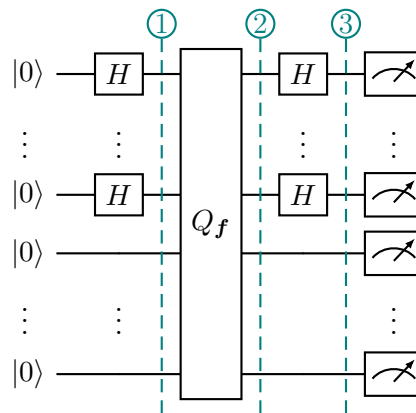
Lemma 5.2

There is a classical algorithm that uses $O(\sqrt{2^n})$ queries to solve Simon's problem with any desired constant success probability.

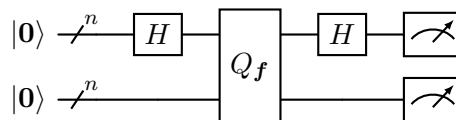
We will later see that this is essentially optimal – any classical algorithm needs $\Omega(\sqrt{2^n})$ queries to solve this problem with constant success probability.

5.2 Simon's quantum algorithm

We now describe a quantum algorithm that needs $O(n)$ queries – an exponential improvement. Consider the quantum circuit given by the left-hand side picture:



There are $2n$ qubits - the Hadamard unitaries are applied to the first n of them. The following is a common way of denoting this:



We would like to compute the joint probability of the measurement outcomes.¹ To this end we first compute the state at the indicated points in the circuit. Recall the general formula

$$H^{\otimes n} |\mathbf{x}\rangle = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{y} \in \{0,1\}^n} (-1)^{\mathbf{x} \cdot \mathbf{y}} |\mathbf{y}\rangle, \quad (5.2)$$

¹In fact, we will see that all we will care about is the outcomes of the measurements of the first n qubits. However, the analysis becomes slightly more illuminating if one measures all qubits. See also the practice set, where you can analyze what happens if you measure the last n qubits as soon as possible.

where $\mathbf{x} \cdot \mathbf{y} = x_1y_1 \oplus \dots \oplus x_ny_n$. The state at time ① is given by

$$(H^{\otimes n} \otimes I^{\otimes n}) |\mathbf{0}, \mathbf{0}\rangle = H^{\otimes n} |\mathbf{0}\rangle \otimes |\mathbf{0}\rangle = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x} \in \{0,1\}^n} |\mathbf{x}\rangle \otimes |\mathbf{0}\rangle = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x} \in \{0,1\}^n} |\mathbf{x}, \mathbf{0}\rangle.$$

To find the state at time ② we apply the oracle as in Eq. (5.1), and we find

$$Q_f \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x} \in \{0,1\}^n} |\mathbf{x}, \mathbf{0}\rangle = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x} \in \{0,1\}^n} |\mathbf{x}, \mathbf{f}(\mathbf{x})\rangle.$$

Next, we apply another layer of Hadamard's to the first n qubits, resulting in the following state at time ③:

$$\begin{aligned} & (H^{\otimes n} \otimes I^{\otimes n}) \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x} \in \{0,1\}^n} |\mathbf{x}, \mathbf{f}(\mathbf{x})\rangle \\ &= \frac{1}{\sqrt{2^n}} \sum_{\mathbf{x} \in \{0,1\}^n} H^{\otimes n} |\mathbf{x}\rangle \otimes |\mathbf{f}(\mathbf{x})\rangle \\ &= \frac{1}{2^n} \sum_{\mathbf{z} \in \{0,1\}^n} \sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{\mathbf{x} \cdot \mathbf{z}} |\mathbf{z}\rangle \otimes |\mathbf{f}(\mathbf{x})\rangle \\ &= \frac{1}{2^n} \sum_{\mathbf{z} \in \{0,1\}^n} \sum_{\mathbf{y} \in \text{im}(\mathbf{f})} \left((-1)^{\mathbf{x}_y \cdot \mathbf{z}} + (-1)^{(\mathbf{x}_y \oplus \mathbf{s}) \cdot \mathbf{z}} \right) |\mathbf{z}\rangle \otimes |\mathbf{y}\rangle \\ &= \frac{1}{2^{n-1}} \sum_{\mathbf{z} \in \mathbf{s}^\perp} \sum_{\mathbf{y} \in \text{im}(\mathbf{f})} (-1)^{\mathbf{x}_y \cdot \mathbf{z}} |\mathbf{z}\rangle \otimes |\mathbf{y}\rangle, \end{aligned}$$

To get from the second to the third line, we again used Eq. (5.2). To get from the third to the fourth line, we used that for every \mathbf{y} in the image of \mathbf{f} there exist precisely two \mathbf{x} such that $\mathbf{y} = \mathbf{f}(\mathbf{x})$; we arbitrarily pick one and call it \mathbf{x}_y , then the other one is $\mathbf{x}_y \oplus \mathbf{s}$, by the periodicity of \mathbf{f} . In the last line, we used the notation $\mathbf{s}^\perp = \{\mathbf{z} \in \{0,1\}^n : \mathbf{z} \cdot \mathbf{s} = 0\}$ for the orthogonal complement of \mathbf{s} modulo two. Note that \mathbf{s}^\perp is a subspace of $\{0,1\}^n \cong \mathbb{F}_2^n$. Since $\mathbf{s} \neq \mathbf{0}$, it has dimension $n - 1$ and hence cardinality 2^{n-1} . Since the image of \mathbf{f} also has cardinality 2^{n-1} , we see that the state has norm one, as it must have. This is a good sanity check to see that we did not make a mistake.

From the last formula we see that if we measure all qubits we obtain a uniformly random $\mathbf{z} \in \mathbf{s}^\perp$, where \mathbf{s} is the secret period, as well as a uniformly random function value $\mathbf{y} \in \text{im}(\mathbf{f})$ (which we will not actually care about). The key point is that \mathbf{s} can easily be determined from \mathbf{s}^\perp – it is the unique nonzero bitstring $\mathbf{s} \neq \mathbf{0}$ that is orthogonal to all elements of \mathbf{s}^\perp , hence it can be found by solving a linear system.

The idea of Simon's quantum algorithm is now very simple: We run the above circuit $n - 1$ times, collecting vectors $\mathbf{z}(1), \mathbf{z}(2), \dots, \mathbf{z}(n - 1)$. If we are lucky, the vectors will be linearly independent, hence their span will be equal to \mathbf{s}^\perp . We can then determine \mathbf{s} as explained above. What is the probability that we get lucky?

$$\begin{aligned} & \Pr\left(\mathbf{z}(1), \mathbf{z}(2), \dots, \mathbf{z}(n - 1) \text{ linearly independent}\right) \\ &= \prod_{t=1}^{n-1} \Pr\left(\mathbf{z}(t) \notin \text{span}\{\mathbf{z}(1), \dots, \mathbf{z}(t - 1)\} \mid \mathbf{z}(1), \dots, \mathbf{z}(t - 1) \text{ linearly independent}\right) \\ &= \prod_{t=1}^{n-1} \left(1 - \frac{2^{t-1}}{2^n}\right) = \left(1 - \frac{1}{2^n}\right) \left(1 - \frac{1}{2^{n-1}}\right) \cdots \left(1 - \frac{1}{4}\right) \left(1 - \frac{1}{2}\right) \\ &= \frac{1}{2} \left(1 - \frac{1}{4}\right) \cdots \left(1 - \frac{1}{2^{n-2}}\right) \left(1 - \frac{1}{2^{n-1}}\right) \\ &\geq \frac{1}{2} \left(1 - \frac{1}{4} - \frac{1}{8} - \cdots - \frac{1}{2^{n-2}} - \frac{1}{2^{n-1}}\right) \geq \frac{1}{4}, \end{aligned}$$

using the inequality $(1-a)(1-b) \geq 1-a-b$ for $a, b \geq 0$ and the geometric series $\frac{1}{4} + \frac{1}{8} + \cdots + \frac{1}{2^{n-1}} = \frac{1}{2} - \frac{1}{2^{n-1}} \leq \frac{1}{2}$. We conclude that the vectors $\mathbf{z}(1), \dots, \mathbf{z}(n - 1)$ span \mathbf{s}^\perp with probability at least $\frac{1}{4}$. A failure probability at

most $3/4$ does not sound particularly good, but note that we can easily check when failure occurs by checking whether the vectors are linearly dependent. We thus obtain a quantum algorithm that uses n quantum queries to either output \mathbf{s} correctly or report failure, and the latter happens with probability at most $3/4$. If we repeat this procedure k times, then the probability that failure occurs in every round is at most $(3/4)^k$ and hence can be made as small as we want. In summary:

Theorem 5.3: Simon’s quantum algorithm

Simon’s quantum algorithm described above uses $O(n)$ quantum queries to determine \mathbf{s} with any desired constant success probability.

5.3 Bonus: Classical lower bound

How many queries does a classical algorithm need? Above we saw that $T = O(\sqrt{2^n})$ queries suffice when we can use randomness, but is this also necessary? To show this, it seems natural to try to construct a function \mathbf{f} for which Simon’s problem is particularly difficult to solve. Now this will not quite work, since there is of course a very simple algorithm for any fixed function \mathbf{f} (simply output the correct answer, which is some fixed \mathbf{s}). However if we pick the function \mathbf{f} at random then the situation becomes more interesting. Indeed, suppose that we have a randomized algorithm A_{rand} for Simon’s problem that makes T queries and gives the right answer with probability at least some p for *every* periodic function \mathbf{f} . Then with probability at least p it will also give the right answer if we input a function \mathbf{f} chosen at random from some fixed distribution \mathcal{D} . Moreover, since we can think of a randomized algorithm as a deterministic algorithm chosen at random, there must even exist a deterministic algorithm A_{det} making T queries that does at least as well on the same distribution \mathcal{D} . That is:²

Lemma 5.4: Yao’s principle

Let \mathcal{D} be a probability distribution over instances of some computational problem P . Suppose any deterministic classical algorithm that solves P with probability p when given a random instance drawn from \mathcal{D} needs at least T queries. Then any randomized classical algorithm that solves P with probability p on arbitrary instances also needs at least T queries.

We will choose the distribution \mathcal{D} as follows: first pick a uniformly random $\mathbf{0} \neq \mathbf{s} \in \{0, 1\}^n$ and then let \mathbf{f} be a uniformly random function with this period \mathbf{s} .

Now consider an arbitrary *deterministic* algorithm for Simon’s problem that makes T queries, say $\mathbf{f}(\mathbf{x}(1)), \dots, \mathbf{f}(\mathbf{x}(T))$ for certain $\mathbf{x}(1), \dots, \mathbf{x}(T)$; without loss of generality the $\mathbf{x}(t)$ ’s are all distinct. Note that later queries may depend on the results of earlier ones. Let us slightly modify the algorithm by making one more query, namely $\mathbf{f}(\mathbf{x}(T + 1))$ for $\mathbf{x}(T + 1) = \mathbf{x}(1) \oplus \mathbf{y}$, where \mathbf{y} is the algorithm’s output. The point is the following: If the original algorithm’s output was the correct period \mathbf{s} , then in the modified algorithm we are certain to have seen a collision, i.e., $\mathbf{f}(\mathbf{x}(i)) = \mathbf{f}(\mathbf{x}(j))$ for some $i \neq j$. In other words, if we see *no* collision then the algorithm will necessarily failed to solve Simon’s problem. Thus:

$$\Pr(\text{algorithm fails}) \geq \Pr(\text{no collision within } T + 1 \text{ queries})$$

We will bound the right-hand inductively. Suppose no collision has occurred in the first t queries, i.e., $\mathbf{f}(\mathbf{x}(1)) \neq \dots \neq \mathbf{f}(\mathbf{x}(t))$. Then $\mathbf{f}(\mathbf{x}(1)), \dots, \mathbf{f}(\mathbf{x}(t))$ are independent from \mathbf{s} by construction of \mathcal{D} . It follows that $\mathbf{x}(1), \mathbf{x}(2), \dots, \mathbf{x}(t), \mathbf{x}(t + 1)$ are also independent of \mathbf{s} , and hence:

$$\begin{aligned} & \Pr(\text{no collision within } t + 1 \text{ queries} \mid \text{no collision within } t \text{ queries}) \\ &= \Pr(\mathbf{s} \notin \{\mathbf{x}(i) \oplus \mathbf{x}(t + 1)\}_{i=1}^t \mid \mathbf{s} \notin \{\mathbf{x}(i) \oplus \mathbf{x}(j)\}_{i,j=1}^t) \\ &\geq 1 - \frac{t}{2^n - 1 - \binom{t}{2}} \end{aligned}$$

²More formally, we can summarize this idea as follows: $\max_{A_{\text{rand}} \text{ } T \text{ queries}} \min_{\mathbf{f}} \Pr(A_{\text{rand}} \text{ gives the right answer for } \mathbf{f}) \leq \min_{\mathcal{D}} \max_{A_{\text{det}} \text{ } T \text{ queries}} \Pr(A_{\text{det}} \text{ gives the right answer for } \mathbf{f} \sim \mathcal{D})$.

(to prove the bound, first show that it holds conditioned on any fixed sequence of $\mathbf{x}(i)$'s), and hence

$$\begin{aligned} \Pr(\text{algorithm fails}) &\geq \Pr(\text{no collision within } T + 1 \text{ queries}) \\ &\geq \prod_{t=1}^T \left(1 - \frac{t}{2^n - 1 - \binom{t}{2}}\right) = \prod_{t=1}^T \frac{2^n - 1 - \binom{t+1}{2}}{2^n - 1 - \binom{t}{2}} \\ &= \frac{2^n - 1 - \binom{T+1}{2}}{2^n - 1} = 1 - \frac{\binom{T+1}{2}}{2^n - 1} \\ &\geq 1 - \Theta(T^2/2^n). \end{aligned}$$

We conclude that the algorithm will fail with probability arbitrarily close to 1 unless $T = \Omega(\sqrt{2^n})$. Thus, Yao's principle implies the following result.

Theorem 5.5: Classical lower bound for Simon's problem

Any (possibly randomized) classical algorithm that solves Simon's problem with a constant success probability must make $\Omega(\sqrt{2^n})$ queries.

5.3.1 Lower bound for the decision version of Simon's problem

We can use the same idea to prove a stronger result, namely that $\Omega(\sqrt{2^n})$ queries are required even if we are only interested in the following simpler problem:

Problem 5.6: Simon's decision problem

You are given query access to a function $\mathbf{f}: \{0, 1\}^n \rightarrow \{0, 1\}^n$ such that, for some $\mathbf{s} \in \{0, 1\}^n$, we have $\mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{y})$ if and only if $\mathbf{y} = \mathbf{x}$ or $\mathbf{y} = \mathbf{x} \oplus \mathbf{s}$. Is $\mathbf{s} = \mathbf{0}$?

That is, either \mathbf{f} is bijective (this is the case that $\mathbf{s} = \mathbf{0}$) or it is periodic (with period $\mathbf{s} \neq \mathbf{0}$). Why is this problem simpler than the original Simon's problem (Problem 5.1)? This is because you can use any algorithm for the original Simon's problem to solve it: simply run such an algorithm and test whether its result is indeed the period, using two additional queries.

To analyze the difficulty of this problem we will again rely on Yao's principle. We choose the distribution \mathcal{D} as follows:

- With probability 1/2, the function \mathbf{f} is a random bijection of $\{0, 1\}^n$. In this case, $\mathbf{s} = \mathbf{0}$. Here the answer to the problem is "yes".
- With probability 1/2, we pick a random $\mathbf{0} \neq \mathbf{s} \in \{0, 1\}^n$ and we let \mathbf{f} be a random function with this period \mathbf{s} (as before). Here the answer to the problem is "no".

Consider any deterministic algorithm for Simon's decision problem. How does it perform on this distributions?

- With probability $\frac{1}{2}$, $\mathbf{s} = \mathbf{0}$ and \mathbf{f} is a random bijection. In this case, the query results $\mathbf{f}(\mathbf{x}(1)), \dots, \mathbf{f}(\mathbf{x}(T))$ are uniformly random bitstrings, subject only to the condition that they are pairwise distinct.
- With probability $\frac{1}{2}$, $\mathbf{s} \neq \mathbf{0}$, and \mathbf{f} is a random function with period \mathbf{s} . If we have no collision then the query results $\mathbf{f}(\mathbf{x}(1)) \neq \dots \neq \mathbf{f}(\mathbf{x}(T))$ are again uniformly random pairwise distinct bitstrings! Moreover, we know from above that we are very unlikely to have a collision unless $T = \Omega(\sqrt{2^n})$.

Now assume the algorithm makes fewer than $\Omega(\sqrt{2^n})$ queries. Then the query results $\mathbf{f}(\mathbf{x}(1)), \dots, \mathbf{f}(\mathbf{x}(T))$ have almost the same distribution. Accordingly, the probability that the algorithm outputs "yes" will be almost the same in both cases. But then the probability of solving Simon's decision problem for the above distribution correctly will be arbitrarily close to 1/2, since

$$\Pr(\text{correct}) = \frac{1}{2}(\Pr(\text{"yes"}|\text{first case}) + (1 - \Pr(\text{"yes"}|\text{second case}))) \approx \frac{1}{2}$$

We conclude that the algorithm will fail with probability arbitrarily close to 1/2 unless $T = \Omega(\sqrt{2^n})$. Again we can apply Yao's principle to lift this result to randomized algorithms:

Theorem 5.7: Classical lower bound for Simon's decision problem

Any (possibly randomized) classical algorithm that solves Simon's decision problem with success probability at least $2/3$ (or any other constant probability of success that is strictly larger than $1/2$) must make $\Omega(\sqrt{2^n})$ queries.

Chapter 6

Grover's search algorithm and applications (or: SAT on a quantum computer)

This week we will study another query algorithm where quantum computers have an advantage.

6.1 Search problem and classical algorithm

Problem 6.1: Grover's search problem

You are given query access to a function $g : \{0, 1\}^n \rightarrow \{0, 1\}$. Find $\mathbf{x} \in \{0, 1\}^n$ such that $g(\mathbf{x}) = 1$, or output "no solution" if no such x exists.

Elements $\mathbf{x} \in \{0, 1\}^n$ such that $g(\mathbf{x}) = 1$ are called "solutions" to the search problem, or "marked elements". Problem 6.1 models an extremely general kind of search problem. For example:

- *Satisfiability*: g is a boolean formula and \mathbf{x} an assignment of variables.
- *Knapsack*: \mathbf{x} is a bitstring representing a subset of items and $g(\mathbf{x}) = 1$ iff the total weight of the items is $\leq W$ and the total value is $\geq V$ (for some thresholds W and V).
- *Database search*: Given query access to a database, let

$$g(\mathbf{x}) = \begin{cases} 1 & \text{if entry at position } \mathbf{x} \text{ satisfies some search criterion,} \\ 0 & \text{otherwise.} \end{cases}$$

Usually, the function $g(\mathbf{x})$ is easy to evaluate for any given $\mathbf{x} \in \{0, 1\}^n$ (that is, in time $\text{poly}(n)$). The difficulty comes from the fact that there is an exponential number $N := 2^n$ of \mathbf{x} 's to check.

How well can we do? Any classical algorithm needs $\Omega(N)$ queries to solve the search problem for arbitrary g with probability $\geq \frac{2}{3}$. This is clear for deterministic algorithms, since such an algorithm will query locations $\mathbf{x}(1), \mathbf{x}(2), \dots$ in some definite order until it sees first solution, and in the worst case we will have to inspect all elements before we find the solution (or conclude that there is none). The same is true for randomized algorithms, as you will discuss in the exercises.

Similarly, if we *know* that there are $\geq T$ solutions then $\Theta(N/T)$ queries are necessary and sufficient to solve the search problem with probability $\geq \frac{2}{3}$. What if there are $\geq T$ solutions but we don't know T ? In this case, an *expected* number of $\Theta(N/T)$ queries are necessary and sufficient – simply query $g(\mathbf{x})$ at random locations $\mathbf{x} \in \{0, 1\}^n$ until a solution is found.

6.2 Grover's quantum search algorithm

Amazingly, there is a quantum algorithm for the search problem that offers a quadratic speedup over any classical algorithm.

Theorem 6.2: Grover's quantum search algorithm

Let $g: \{0, 1\}^n \rightarrow \{0, 1\}$ be a function and $N = 2^n$. There is a quantum algorithm that uses $O(\sqrt{N})$ quantum queries and $O(\sqrt{N} \log(N))$ other gates to solve the search problem for g with probability $\geq \frac{2}{3}$.

If we *know* that there are $\geq T$ solutions, $O(\sqrt{\frac{N}{T}})$ queries and $O(\sqrt{\frac{N}{T}} \log(N))$ other gates suffice to find a solution with probability $\geq \frac{2}{3}$. If we *know* that there are *exactly* T solutions, we can even achieve 100% success probability. If we *don't* know T , there still exists a quantum algorithm that finds a solution with an *expected* number of $O(\sqrt{\frac{N}{T}})$ queries and $O(\sqrt{\frac{N}{T}} \log(N))$ other gates.

Example 6.3: Application to Satisfiability

Given a Boolean formula g , [Theorem 6.2](#) shows that there is a quantum algorithm that solves the satisfiability problem ([Problem 1.3](#)) using $O(\sqrt{2^n}) = O(1.414\dots^n)$ queries to a quantum oracle for the formula. By [Chapter 4](#) we know that we can implement this oracle using a quantum circuit of polynomial size (in the description of the formula). Thus, quantum computers can solve the satisfiability problem (for formulas of subexponential size) in time $\tilde{O}(\sqrt{2^n})$. In contrast, it is believed that any classical algorithm must require time $\tilde{\Omega}(2^n)$ in the worst case, as discussed in [Chapter 1](#)!

In the remainder of this lecture we will prove a simplified version of [Theorem 6.2](#). The main simplifying assumption that we will make is the following: we assume that the number of solutions

$$T := \#\{\mathbf{x} \in \{0, 1\}^n : f(\mathbf{x}) = 1\}$$

is known to us! This is of course often unrealistic, but we will explain how to fix it later. Given the assumption, our goal is to solve the search problem using $O\left(\sqrt{\frac{N}{T}}\right)$ queries. Consider the following two states.

- The *uniform superposition* over all computational basis states:

$$|U\rangle = \frac{1}{\sqrt{N}} \sum_{\mathbf{x} \in \{0, 1\}^n} |\mathbf{x}\rangle$$

This one is easy to prepare ($|U\rangle = H^{\otimes n} |\mathbf{0}\rangle$), but if we measure it we get a solution with probability T/N (so we would have to repeat $O(N/T)$ times to find one with good probability).

- The “*good*” state, which is a superposition over all solutions:

$$|G\rangle = \frac{1}{\sqrt{T}} \sum_{\mathbf{x} \in \{0, 1\}^n : g(\mathbf{x})=1} |\mathbf{x}\rangle$$

If we measure this state then we get a solution with 100% probability, but it is a priori unclear how to prepare this state.

The key idea is that we will try to go from $|U\rangle$ (which is easy to prepare) to $|G\rangle$ (which solves our problem) using only $O(\sqrt{N/T})$ queries.

To better understand the situation, let us also define the “*bad*” state

$$|B\rangle = \frac{1}{\sqrt{N-T}} \sum_{\mathbf{x} \in \{0, 1\}^n : g(\mathbf{x})=0} |\mathbf{x}\rangle.$$

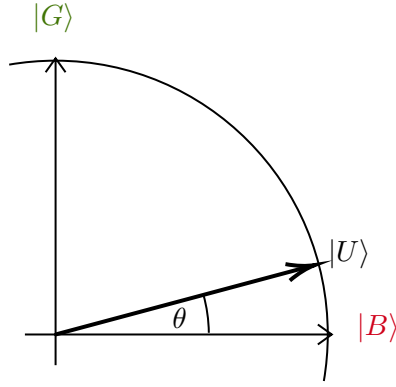
Then:

$$|U\rangle = \sqrt{1 - \frac{T}{N}} |B\rangle + \sqrt{\frac{T}{N}} |G\rangle = \cos(\theta) |B\rangle + \sin(\theta) |G\rangle,$$

where we choose the angle to be in $\theta \in [0, \pi/2]$. Note that, crucially,

$$\theta \geq \sin(\theta) = \sqrt{\frac{T}{N}} \tag{6.1}$$

because $\sin(x) \leq |x|$ for every $x \in \mathbb{R}$. We can visualize the situation as follows:



What natural operations do we have available?

1. The quantum phase oracle $P_g |\mathbf{x}\rangle = (-1)^{g(\mathbf{x})} |\mathbf{x}\rangle$, which sends

$$\begin{aligned} P_g |B\rangle &= |B\rangle \\ P_g |G\rangle &= -|G\rangle. \end{aligned}$$

Note that this is a reflection across the $|B\rangle$ -axis!

2. Inspired by the above, we can also reflect about the $|U\rangle$ -axis by using the unitary

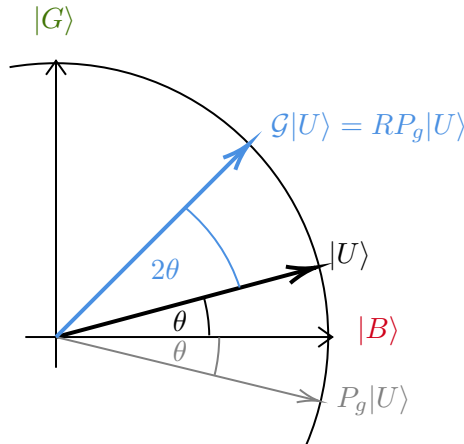
$$R = 2|U\rangle\langle U| - I.$$

Note that this unitary does not depend on the function. It is easy to implement using $O(n)$ gates, as you will discuss in the exercises.

The *Grover operator* is defined by combining these two operations:

$$\mathcal{G} = RP_g.$$

As it is a product of two reflections, it acts as a rotation in the two-dimensional subspace spanned by $|G\rangle$ and $|B\rangle$! The rotation angle is easily seen to be 2θ . To confirm this, simply consider the following picture:



If we start with the state $|U\rangle$, which is at an angle θ , and apply the Grover operator k times, then we obtain the state

$$\mathcal{G}^k |U\rangle = \cos((2k+1)\theta) |B\rangle + \sin((2k+1)\theta) |G\rangle.$$

Ideally, $(2k+1)\theta = \frac{\pi}{2}$ to obtain the good state, but k needs to be an integer. Thus we choose k as the integer closest to $k_{\text{ideal}} := \frac{\pi}{4\theta} - \frac{1}{2}$. Then:

$$k = O\left(\frac{1}{\theta}\right) = O\left(\sqrt{\frac{N}{T}}\right),$$

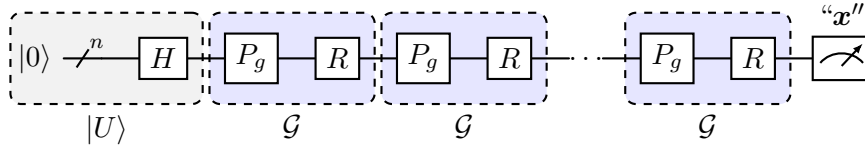


Figure 6.1: Grover’s quantum circuit as used in [Theorem 6.4](#). The Grover operator $\mathcal{G} = RP_g$ is repeated $k = O(\sqrt{\frac{N}{T}})$ times.

using [Ineq. \(6.1\)](#) in the second step. This is indeed the right number of iterations that we are aiming for! To see that this is a good choice, let us compute the probability that we obtain a solution if we measure $\mathcal{G}^k |U\rangle$ in the computational basis:

$$\begin{aligned} \Pr(\text{solution}) &= \sin^2((2k + 1)\theta) \\ &= \cos^2\left((2k + 1)\theta - \frac{\pi}{2}\right) \\ &= 1 - \sin^2\left((2k + 1)\theta - \frac{\pi}{2}\right) \\ &\geq 1 - \sin^2(\theta) = 1 - \frac{T}{N}. \end{aligned}$$

If the first equality bothers you, see the corresponding problem on the practice set. The inequality follows because $|k - k_{\text{ideal}}| = |k - (\frac{\pi}{4\theta} - \frac{1}{2})| \leq \frac{1}{2}$ and hence $|(2k + 1)\theta - \frac{\pi}{2}| \leq \theta$, using that $\sin(x)$ is monotonic for $x \in [-\pi/2, \pi/2]$.

We summarize what we have achieved:

Theorem 6.4: Baby Grover

Suppose we *know* that $g: \{0, 1\}^n \rightarrow \{0, 1\}$ has *exactly* T solutions. Then we can use the quantum circuit in [Fig. 6.1](#), which uses $O(\sqrt{\frac{N}{T}})$ queries to g , to output a solution with probability at least $1 - \frac{T}{N}$.

Let’s discuss this result:

- If $T \leq \frac{N}{3}$ then the success probability of the algorithm is $\geq 2/3$ as desired. What if this is not the case? There are two options:
 - If $T > \frac{N}{3}$ then we can simply (classically) query the function a few times at random locations to find a solution with probability $\geq 2/3$.
 - Alternatively, we can replace g by a new function

$$\tilde{g}: \{0, 1\}^{n+2}, \quad \tilde{g}(\mathbf{x}, y, z) := \begin{cases} g(\mathbf{x}) & \text{if } y = z = 0, \\ 0 & \text{otherwise,} \end{cases}$$

whose solutions are precisely of the form $(\mathbf{x}, 0, 0)$ for \mathbf{x} a solution to g . Thus, \tilde{g} has T solutions, but the search space has size $4N$. Hence if we apply Grover’s algorithm to \tilde{g} we obtain a solution with probability at least $\geq 3/4$. Note that \tilde{g} can be constructed from an oracle for g without knowledge of its inner workings nor of T .

- If T is known, it is in fact possible to improve the above circuit to rotate to $|G\rangle$ exactly and get an algorithm that succeeds with 100% probability.
- If T is not known (or only a lower bound is known), there are at least two options:
 1. Try different k in a systematic way.
 2. Estimate the number of solutions T ! We will learn how this works later in the course.

Either option allows proving [Theorem 6.2](#).

Chapter 7

Quantum Fourier transform

In [Chapter 5](#), we studied Simon's problem. The key ingredient in solving it was the *Hadamard transform* $H^{\otimes n}$, which maps the n -qubit standard basis to the n -qubit *Hadamard basis*

$$H^{\otimes n} |\mathbf{x}\rangle = \frac{1}{\sqrt{N}} \sum_{\mathbf{z} \in \{0,1\}^n} (-1)^{\mathbf{x} \cdot \mathbf{z}} |\mathbf{z}\rangle, \quad (7.1)$$

where $\mathbf{x} \cdot \mathbf{z} = x_1 z_1 \oplus \dots \oplus x_n z_n$ is the inner product modulo two and $N = 2^n$. To study the peculiar sign pattern on the right-hand side, let us define for every $\mathbf{x} \in \{0,1\}^n$ the function

$$\beta_{\mathbf{x}}: \{0,1\}^n \rightarrow \mathbb{C}, \quad \beta_{\mathbf{x}}(\mathbf{z}) := (-1)^{\mathbf{x} \cdot \mathbf{z}}. \quad (7.2)$$

It has the following magical property:

$$\beta_{\mathbf{x}}(\mathbf{a} \oplus \mathbf{b}) = \beta_{\mathbf{x}}(\mathbf{a})\beta_{\mathbf{x}}(\mathbf{b}) \quad \forall \mathbf{a}, \mathbf{b} \in \{0,1\}^n. \quad (7.3)$$

That is, adding the arguments (which are bitstrings) modulo two is the same as multiplying the function values (which are complex numbers).

Moreover, if you think of the functions $\beta_{\mathbf{x}}$ for $\mathbf{x} \in \{0,1\}^n$ as complex vectors of length $N = 2^n$, then these vectors are orthogonal.¹ Indeed, if this were not true then $H^{\otimes n}$ would not be unitary, so we could not use it in a quantum circuit!

7.1 Fourier transform for \mathbb{Z}_N

Today we want to discuss a unitary transformation that is adapted to the *cyclic group* $\mathbb{Z}_N = \mathbb{Z}/N\mathbb{Z}$, i.e., the set of integers modulo N (we identify any two integers if they differ by a multiple of N), along with addition modulo N as the operation. We can think of this concretely as $\{0, 1, \dots, N-1\}$ with addition modulo N . We will mostly write $+$ for the addition modulo N . For now we allow N to be an arbitrary positive integer.

When $N = 2$ then \mathbb{Z}_2 is the same as $\{0, 1\}$ with \oplus , so in this case we should simply take the Hadamard transform. However, in general, \mathbb{Z}_N is *not* the same as $\{0, 1\}^n$, even if $N = 2^n$, so we need to find a new definition. We take inspiration from the above and consider:

Definition 7.1: Characters of \mathbb{Z}_N

For any $x \in \mathbb{Z}_N$, we define

$$\chi_x: \mathbb{Z}_N \rightarrow \mathbb{C}, \quad \chi_x(z) = \omega_N^{xz},$$

where

$$\omega_N := e^{\frac{2\pi i}{N}}.$$

These functions are called the *characters* of the cyclic group \mathbb{Z}_N . The number ω_N is a so-called *primitive*

¹As an aside, note that this means in particular that any function $f: \{0,1\}^n \rightarrow \mathbb{C}$ can be written as a linear combination of the functions $\beta_{\mathbf{x}}$, i.e., we can find numbers $c_{\mathbf{x}} \in \mathbb{C}$ such that $f(\mathbf{z}) = \sum_{\mathbf{x} \in \{0,1\}^n} c_{\mathbf{x}} \beta_{\mathbf{x}}(\mathbf{z})$ for all \mathbf{z} . This is studied in the Fourier analysis of functions.

N -th root of unity, meaning that $\omega_N^N = 1$ but $\omega_N^k \neq 1$ for any $0 < k < N$.

These functions satisfy the magical property (7.3) from above, but with $(\{0, 1\}^n, \oplus)$ replaced by $(\mathbb{Z}_N, +)$:

$$\chi_x(a+b) = \omega_N^{x(a+b)} = \omega_N^{xa} \omega_N^{xb} = \chi_x(a) \chi_x(b) \quad \forall a, b \in \mathbb{Z}_N. \quad (7.4)$$

Conversely, any nonzero function $\chi: \mathbb{Z}_N \rightarrow \mathbb{C}$ that satisfies this property is equal to some χ_x .²

Definition 7.2: Fourier transform

For any $N \geq 2$, we define the *Fourier transform* of the cyclic group F_N as the linear map defined by

$$F_N: \mathbb{C}^N \rightarrow \mathbb{C}^N, \quad F_N |x\rangle := \frac{1}{\sqrt{N}} \sum_{z \in \mathbb{Z}_N} \omega_N^{xz} |z\rangle \quad \forall x \in \mathbb{Z}_N,$$

where we write $|z\rangle$ for $z \in \mathbb{Z}_N$ for the standard basis of \mathbb{C}^N . The map F_N is also called the *discrete Fourier transform*.^a

^aSome literature uses slightly different conventions, with ω_N^{-xy} in place of ω_N^{xy} , and they would call our definition of F_N the inverse Fourier transform.

Note the analogy between this definition and Eq. (7.1). For $N = 2$, F_2 is simply the Hadamard unitary H , but it is different otherwise.

If $|v\rangle = \sum_x v_x |x\rangle \in \mathbb{C}^N$ is an arbitrary vector, then its Fourier transform is given by

$$F_N |v\rangle = \sum_{x \in \mathbb{Z}_N} v_x F_N |x\rangle = \sum_{x \in \mathbb{Z}_N} v_x \frac{1}{\sqrt{N}} \sum_{z \in \mathbb{Z}_N} \omega_N^{xz} |z\rangle = \sum_{z \in \mathbb{Z}_N} \left(\frac{1}{\sqrt{N}} \sum_{x \in \mathbb{Z}_N} \omega_N^{xz} v_x \right) |z\rangle.$$

We can also write this without Dirac notation: if $\mathbf{v} \in \mathbb{C}^N$ is a vector, then its Fourier transform $F_N \mathbf{v}$ has coefficients

$$(F_N \mathbf{v})_z = \frac{1}{\sqrt{N}} \sum_{x \in \mathbb{Z}_N} \omega_N^{xz} v_x \quad (7.5)$$

for $z \in \mathbb{Z}_N$.

Lemma 7.3

The Fourier transform $F_N: \mathbb{C}^N \rightarrow \mathbb{C}^N$ is unitary for any N .

Proof. It is enough to show that the vectors $F_N |a\rangle$ are orthonormal. Indeed, for any two $a, b \in \mathbb{Z}_N$, we have

$$\begin{aligned} \langle F_N |a\rangle, F_N |b\rangle \rangle &= \frac{1}{N} \sum_{z \in \mathbb{Z}_N} \overline{\omega_N^{az}} \omega_N^{bz} = \frac{1}{N} \sum_{z \in \mathbb{Z}_N} \omega_N^{(b-a)z} = \frac{1}{N} \sum_{z=0}^{N-1} (\omega_N^{b-a})^z \\ &= \begin{cases} 1 & \text{if } a = b \pmod{N}, \\ \frac{1}{N} \frac{1 - \omega_N^{N(b-a)}}{1 - \omega_N^{b-a}} = \frac{1}{N} \frac{1-1}{1 - \omega_N^{b-a}} = 0 & \text{otherwise.} \end{cases} \end{aligned}$$

The last equality is clear for $a = b$, while for $a \neq b$ we used the formula $\sum_{z=0}^{N-1} \eta^z = \frac{1-\eta^N}{1-\eta}$ for the geometric sum of a number $\eta \neq 1$, applied to $\eta = \omega_N^{b-a}$. \square

It is clear from the definition $F_N^T = F_N$. Since it is also unitary, we see that the inverse Fourier transform is given by:

$$F_N^{-1} = F_N^\dagger = \overline{F_N},$$

where $\overline{F_N}$ means entrywise complex conjugation.

²To see this, note that we must have $\chi(0) = \chi(0+0) = \chi(0)^2$ and hence $\chi(0) = 1$. But then it follows that $1 = \chi(0) = \chi(1 + \dots + 1) = \chi(1)^N$, hence $\chi(1) = \omega_N^x$ for some $x \in \mathbb{Z}_N$, and we conclude that $\chi = \chi_x$.

7.2 Excursion: Fast Fourier transform

Next, we discuss how the Fourier transform can already be computed very efficiently classically, and we give two beautiful applications. We follow the lecture notes of de Wolf quite closely.

Naively, given a vector of length N (written down on paper, or in memory) we need $O(N^2)$ elementary operations to compute the Fourier transform, since this is the time it takes to multiply the $N \times N$ -matrix F_N with a vector. Is this optimal?

In fact one can do almost quadratically better. Suppose only for simplicity that $N = 2^n$ is a power of two. Then we can write, starting with Eq. (7.5), for any $\mathbf{v} \in \mathbb{C}^N$ that

$$\begin{aligned}
 (F_N \mathbf{v})_z &= \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \omega_N^{xz} v_x \\
 &= \frac{1}{\sqrt{N}} \sum_{x=0}^{N/2-1} \omega_N^{2xz} v_{2x} + \frac{1}{\sqrt{N}} \sum_{x=0}^{N/2-1} \omega_N^{(2x+1)z} v_{2x+1} \\
 &= \frac{1}{\sqrt{N}} \sum_{x=0}^{N/2-1} \omega_{N/2}^{xz} v_{2x} + \omega_N^z \frac{1}{\sqrt{N}} \sum_{x=0}^{N/2-1} \omega_{N/2}^{xz} v_{2x+1} \\
 &= \frac{1}{\sqrt{2}} \left(\frac{1}{\sqrt{N/2}} \sum_{x=0}^{N/2-1} \omega_{N/2}^{xz} (\mathbf{v}_{\text{even}})_x + \omega_N^z \frac{1}{\sqrt{N/2}} \sum_{x=0}^{N/2-1} \omega_{N/2}^{xz} (\mathbf{v}_{\text{odd}})_x \right) \\
 &= \frac{1}{\sqrt{2}} \left((F_{N/2} \mathbf{v}_{\text{even}})_{z \bmod N/2} + \omega_N^z (F_{N/2} \mathbf{v}_{\text{odd}})_{z \bmod N/2} \right).
 \end{aligned}$$

In the third step, we used that $\omega_N^2 = \omega_{N/2}$. In the subsequent step we introduced $\mathbf{v}_{\text{even}}, \mathbf{v}_{\text{odd}} \in \mathbb{C}^{N/2}$ as the vectors that contain the entries of \mathbf{v} at even and odd locations, respectively. Thus:

$$F_N \mathbf{v} = \frac{1}{\sqrt{2}} \left(\begin{bmatrix} F_{N/2} \mathbf{v}_{\text{even}} \\ F_{N/2} \mathbf{v}_{\text{odd}} \end{bmatrix} + \text{diag}(1, \omega_N, \dots, \omega_N^{N-1}) \begin{bmatrix} F_{N/2} \mathbf{v}_{\text{odd}} \\ F_{N/2} \mathbf{v}_{\text{even}} \end{bmatrix} \right)$$

Thus, we can compute F_N recursively by first computing $F_{N/2} \mathbf{v}_{\text{even}}$ and $F_{N/2} \mathbf{v}_{\text{odd}}$ and then using $O(N)$ elementary operations to combine the results. If we denote by C_N the resulting cost of computing F_N , we therefore have $C_N \leq 2C_{N/2} + O(N)$. This is solved by $C_N = O(N \log(N))$. Thus:

Theorem 7.4: Fast Fourier transform

There is a classical algorithm that, given a vector $v \in \mathbb{C}^N$, uses $O(N \log(N))$ elementary operations to compute the Fourier transform $F_N v$.

This is called *fast Fourier transform (FFT)*, and it is quadratically better than the naive $O(N^2)$ algorithm mentioned above. The FFT is very widely used in practice. Note that it is indeed very fast – after all, any algorithm for computing the Fourier transform will necessarily need $\Omega(N)$ operations even just to read the input or write the result!

7.2.1 Applications: Fast multiplication of polynomials and of integers

Given two polynomials $P = \sum_{j=0}^D p_j X^j$ and $Q = \sum_{k=0}^D q_k X^k$ of degree D , their product is

$$PQ = \left(\sum_{j=0}^D p_j X^j \right) \left(\sum_{k=0}^D q_k X^k \right) = \sum_{j,k=0}^D p_j q_k X^{j+k} = \sum_{\ell=0}^{2D} \left(\sum_{j=0}^{\ell} p_j q_{\ell-j} \right) X^{\ell}$$

Naively, it will take $O(D^2)$ elementary operations (additions, multiplications) to compute the coefficients of PQ . However, using the fast Fourier transform we can do better. First note that the above is the same as

$$PQ = \sum_{\ell=0}^{N-1} \left(\sum_{j=0}^{N-1} p_j q_{\ell-j \bmod N} \right) X^{\ell}$$

where $\mathbf{p} = (p_0, p_1, \dots, p_D, 0, \dots, 0)$ and $\mathbf{q} = (q_0, q_1, \dots, q_D, 0, \dots, 0)$ are vectors of length $N = 2D + 1$.³ Thus, what we really want to compute is the convolution $\mathbf{p} \star \mathbf{q} \in \mathbb{C}^N$, which is by definition the vector with coefficients given by

$$(\mathbf{p} \star \mathbf{q})_x = \sum_{y \in \mathbb{Z}_N} p_y q_{x-y \bmod N} \quad \forall x \in \mathbb{Z}_N.$$

It is not hard to check, and you will do so on the homework, that the Fourier transform of a convolution is the coefficientwise multiplication of the individual Fourier transforms:

$$\frac{1}{\sqrt{N}} F_N(\mathbf{p} \star \mathbf{q}) = F_N \mathbf{p} \cdot F_N \mathbf{q}, \quad \text{that is,} \quad \frac{1}{\sqrt{N}} (F_N(\mathbf{p} \star \mathbf{q}))_z = (F_N \mathbf{p})_z \cdot (F_N \mathbf{q})_z \quad \forall z \in \mathbb{Z}_N$$

It follows that the convolution

$$\mathbf{p} \star \mathbf{q} = \sqrt{N} F_N^{-1}(F_N \mathbf{p} \cdot F_N \mathbf{q}).$$

This can be computed using $O(N \log(N))$ elementary operations by using the fast Fourier transform three times.

Theorem 7.5: Fast polynomial multiplication

There is a classical algorithm that given the coefficients of two polynomials of degree D , computes the coefficients of their product using $O(D \log(D))$ elementary operations.

We can use this idea not only to multiply polynomials, but also integers. Indeed, suppose that

$$a = \sum_{j=0}^{n-1} a_j 2^j \quad \text{and} \quad b = \sum_{k=0}^{n-1} b_k 2^k$$

are two n -bit integers. We can interpret their digits as the coefficients of polynomials $A = \sum_{j=0}^{n-1} a_j X^j$ and $B = \sum_{j=0}^{n-1} b_j X^j$. Note that we can recover the two numbers as $a = A(2)$ and $b = B(2)$. Hence their product can be obtained by first multiplying the polynomials A and B and evaluating the result at $X = 2$:

$$ab = A(2)B(2) = (AB)(2).$$

This suggests that we can multiply two n -bit numbers in time $O(n \log n)$. But note that what we called an “elementary operation” above included operations such as multiplying two numbers, i.e., exactly what we are trying to solve here! Thus one needs to be more careful. Still, the general idea is sound and one finds that by using similar ideas one can multiply two n -digit numbers using $O(n \log(n) \log \log(n)) = \tilde{O}(n)$ elementary Boolean operations. This is the famous *Schönhage–Strassen algorithm* which we mentioned in [Chapter 1](#).

7.3 Quantum Fourier transform

The Fourier transform takes as input a vector of length N and gives as output a vector of length N . In [Section 7.2](#) we saw that it can be computed very efficiently on classical computers by using the FFT, which needs only $O(N \log(N)) = \tilde{O}(N)$ elementary operations. Since any algorithm that reads in a vector of length N (i.e., N numbers, written down on paper, or in memory, ...) must take $\Omega(N)$ operations this is near optimal. How could quantum computers even help?

Quantum computers have another way of storing $N = 2^n$ numbers, namely as the amplitudes of an n -qubit quantum state (we assume for simplicity that N is a power of two). Note that this is a different and exponentially smaller representation – instead of a list of N numbers (written in memory, or on paper) we are dealing with a quantum memory of only $n = \log(N)$ qubits.

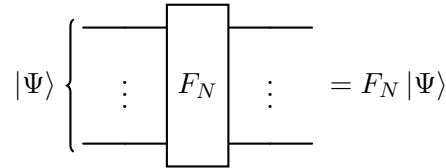
³Note that the sums are now over $\{0, \dots, N-1\} \cong \mathbb{Z}_N$, hence $\ell - j \bmod N$ may “wrap” around and so we need the zeros as padding to get the right result.

To be precise, quantum states of n qubits are vectors in $(\mathbb{C}^2)^{\otimes n}$, but we can identify them with vectors in \mathbb{C}^N as follows: While the computational basis $|\mathbf{x}\rangle$ of $(\mathbb{C}^2)^{\otimes n}$ is labeled by bitstrings $\mathbf{x} \in \{0,1\}^n$, the computational basis $|x\rangle$ of \mathbb{C}^N is labeled by integers $x \in \{0,1,\dots,N-1\}$ (or in \mathbb{Z}_N). We can identify the two by associating with the integer x the string of bits \mathbf{x} that make up its binary expansion:

$$x = x_1 2^{n-1} + \dots + x_{n-1} 2 + x_n = \sum_{j=1}^n x_j 2^{n-j}. \quad (7.6)$$

The way of ordering the digits might be strange, but later we will be interested in $x/2^n = \sum_{j=1}^n x_j/2^j$, from which we see that x_j is the j -th significant digit in the binary expansion of $x/2^n$.

With this identification in place, the Fourier transform F_N is an n -qubit unitary and we can ask how fast it can be implemented on a quantum computer. In other words, we are looking for an n -qubit quantum circuit that implements the following operation: For all n -qubit states $|\Psi\rangle \in (\mathbb{C}^2)^{\otimes n}$,



How many elementary quantum gates do we need to implement F_N ?

We will now see that there is a quantum circuit – called the *quantum Fourier transform (QFT)* – which implements F_N using $O(n^2) = O(\log^2(N))$ one-qubit and two-qubit gates. We will only use Hadamard gates, swap gates, and controlled R_s -gates, where

$$R_s = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi i}{2^s}} \end{pmatrix}.$$

The key calculation is the following. Using [Definition 7.2](#) and [Eq. \(7.6\)](#) to identify $\mathbf{x} \in \{0,1\}^n$ with $x \in \mathbb{Z}_N$, we have

$$\begin{aligned} F_N |\mathbf{x}\rangle &= \frac{1}{\sqrt{N}} \sum_{\mathbf{y} \in \mathbb{Z}_N} \omega_N^{x\mathbf{y}} |\mathbf{y}\rangle \\ &= \frac{1}{\sqrt{2^n}} \sum_{\mathbf{y} \in \{0,1\}^n} e^{2\pi i \frac{x\mathbf{y}}{2^n}} |\mathbf{y}\rangle \\ &= \frac{1}{\sqrt{2^n}} \sum_{\mathbf{y} \in \{0,1\}^n} e^{2\pi i \frac{x \sum_{j=1}^n y_j 2^{n-j}}{2^n}} |\mathbf{y}\rangle \\ &= \frac{1}{\sqrt{2^n}} \sum_{\mathbf{y} \in \{0,1\}^n} e^{\sum_{j=1}^n 2\pi i \frac{x y_j}{2^j}} |\mathbf{y}\rangle \\ &= \frac{1}{\sqrt{2^n}} \sum_{\mathbf{y} \in \{0,1\}^n} \bigotimes_{j=1}^n e^{2\pi i \frac{x y_j}{2^j}} |y_j\rangle \\ &= \bigotimes_{j=1}^n \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i \frac{x}{2^j}} |1\rangle \right). \end{aligned}$$

Note that this is a tensor product! It is therefore natural to try to construct the output state one qubit at a time. For example, if $n = 3$ then we have

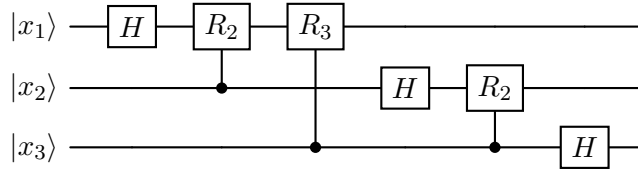
$$\begin{aligned} F_{2^3} |x_1, x_2, x_3\rangle &= \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i \frac{x}{2^1}} |1\rangle \right) \otimes \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i \frac{x}{2^2}} |1\rangle \right) \otimes \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i \frac{x}{2^3}} |1\rangle \right) \\ &= \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i \frac{4x_1 + 2x_2 + x_3}{2}} |1\rangle \right) \otimes \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i \frac{4x_1 + 2x_2 + x_3}{4}} |1\rangle \right) \otimes \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i \frac{4x_1 + 2x_2 + x_3}{8}} |1\rangle \right) \end{aligned}$$

$$\begin{aligned}
&= \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i \frac{x_3}{2}} |1\rangle \right) \otimes \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i \frac{2x_2+x_3}{4}} |1\rangle \right) \otimes \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i \frac{4x_1+2x_2+x_3}{8}} |1\rangle \right) \\
&= \frac{1}{\sqrt{2}} (|0\rangle + (-1)^{x_3} |1\rangle) \otimes \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i \frac{x_3}{4}} (-1)^{x_2} |1\rangle \right) \otimes \frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i \frac{x_3}{8}} e^{2\pi i \frac{x_2}{4}} (-1)^{x_1} |1\rangle \right).
\end{aligned}$$

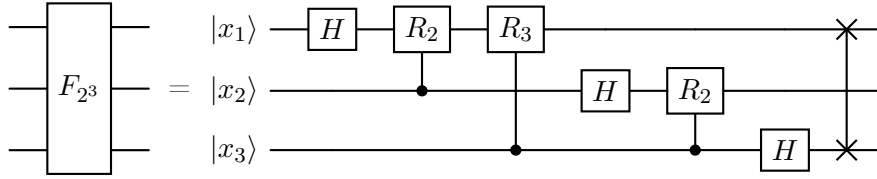
Thus we see that the first output qubit only depends on the last input, the second output bit on the last two inputs, etc. In particular:

- The desired state of the *first* qubit is prepared simply by applying a Hadamard on the *third* qubit.
- The desired state of the second qubit is prepared by applying a Hadamard to the second qubit and then applying a R_2 controlled on the third qubit. We have to do this before applying the Hadamard on the third qubit.
- The desired state of the *third* qubit is prepared by applying a Hadamard to the *first* qubit, then applying R_2 controlled on the second qubit, and finally R_3 controlled on the third qubit. Moreover, we have to do this *before* modifying the state of the second and third qubits.

Thus we arrive at the following circuit:



This circuit almost implements the 3-qubit QFT except for one flaw – the order of the output qubits is wrong and in fact reversed! We can easily fix this by swapping the first and the last qubit. Hence:



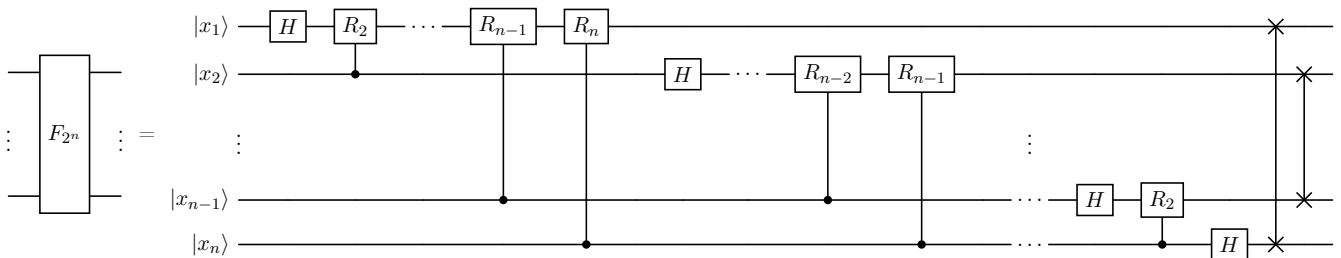
Now the pattern is clear. For n qubits, we similarly have

$$\begin{aligned}
e^{2\pi i \frac{x}{2^j}} &= e^{2\pi i \sum_{k=1}^n x_k 2^{n-k-j}} = e^{2\pi i \sum_{k=n-j+1}^n x_k 2^{n-k-j}} = e^{2\pi i \sum_{s=1}^j \frac{x_{n-j+s}}{2^s}} \\
&= \prod_{s=1}^j e^{\frac{2\pi i}{2^s} x_{n-j+s}} = e^{\frac{2\pi i}{2^j} x_n} \dots e^{\frac{2\pi i}{2^2} x_{n-j+2}} (-1)^{x_{n-j+1}}.
\end{aligned}$$

Hence the state of the j -th output qubit only depends on the last j digits of \mathbf{x} and is given by

$$\frac{1}{\sqrt{2}} \left(|0\rangle + e^{2\pi i \frac{x}{2^j}} |1\rangle \right) = R_j^{x_n} \dots R_2^{x_{n-j+2}} H |x_{n-j+1}\rangle.$$

This leads to the following circuit, which is the general n -qubit *quantum Fourier transform*:



It is clear that this circuit contains $O(n^2)$ gates!

Theorem 7.6: Quantum Fourier transform

Let $N = 2^n$. There exists a quantum circuit containing $O(n^2)$ one-qubit and two-qubit gates that, given an n -qubit state $|\Psi\rangle$, outputs the quantum state $F_N |\Psi\rangle$. This is called the *quantum Fourier transform (QFT)*.

For large s , the unitary R_s is very close to the identity, since then $e^{2\pi i/2^s} \approx 1$. Accordingly these gates can be left out if we are happy to tolerate some small error. More precisely, $O(n \log(n/\varepsilon))$ gates suffice to implement F_N up to error ε in operator norm. You can show this on the homework. This is almost optimal since the quantum Fourier transform F_N clearly needs $\Omega(n)$ gates.

The QFT appears to give an exponential improvement over the classical case – but note again that we are solving a rather different problem from the “classical” Fourier transform! Next week, we will see that the QFT is a key ingredient for many quantum algorithms that solve problems involving finding periods or pattern (just like was the case for the $H^{\otimes n}$). You can already make a head start on this week’s homework and try to solve a version of Simon’s problem for \mathbb{Z}_N . In the next lectures, we will learn how to use the QFT to find discrete logarithms and factor integers.

Remark 7.7

In [Theorem 7.6](#) we assumed that N is a power of two, that is, of the form $N = 2^n$, but this was only for convenience. In fact one can implement F_N by a quantum circuit of size $O((\log N)^2)$ even if N is not a power of two. We will use this fact in the following chapter.

Chapter 8

Shor's quantum algorithm for discrete logs

We follow the lecture notes of Childs quite closely. On last week's homework, you solved Simon's problem for \mathbb{Z}_N by using the quantum Fourier transform. Here we discuss a more involved (but also more interesting!) application.

8.1 Motivation: Breaking Diffie-Hellman by discrete logs

Recall *Diffie-Hellman key exchange* in its simplest form:

1. Alice and Bob publicly agree on a large prime p .

Since \mathbb{Z}_p is a field, every nonzero element has a multiplicative inverse. In fact, $\mathbb{Z}_p^\times = \mathbb{Z}_p \setminus \{0\}$ is a cyclic group (with respect to multiplication), i.e., it is isomorphic to \mathbb{Z}_{p-1} (with addition).

2. Alice and Bob also agree on an element $g \in \mathbb{Z}_p^\times$. For simplicity we suppose that g is a *generator* of this cyclic group, i.e., $\mathbb{Z}_p^\times = \{g, g^2, \dots, g^{p-1} = 1\}$.
3. Now Alice chooses some $a \in \mathbb{Z}_{p-1}$ uniformly at random, and sends $A = g^a \bmod p$ to Bob.
4. Bob similarly chooses some $b \in \mathbb{Z}_{p-1}$ uniformly at random, and sends $B := g^b \bmod p$ to Alice.
5. Finally, Alice computes $K := B^a = (g^b)^a = g^{ab} \bmod p$, and Bob computes $A^b = (g^a)^b = g^{ab} \bmod p$, which is the same as K !

Thus, Alice and Bob end up sharing a key K , while only revealing p , g , A , and B to the eavesdropper Eve. For the Diffie-Hellman protocol to be secure, it is necessary that the following problem is difficult:

Problem 8.1: Discrete logarithm problem

Given a prime p , a generator g of \mathbb{Z}_p^\times , and $x \in \mathbb{Z}_p^\times$, determine $s \in \mathbb{Z}_{p-1}$ such that $g^s = x$. The element s is called the *discrete logarithm* of x in \mathbb{Z}_p^\times with respect to g .

We will now see a quantum algorithm due to Shor (but *not* the same as his famous factoring algorithm) which solves the discrete log problem efficiently. This breaks the security of Diffie-Hellman key exchange completely. The idea is the following: Let $N = p - 1$ and consider the function

$$f: \mathbb{Z}_N \times \mathbb{Z}_N \rightarrow \mathbb{Z}_p^\times, \quad (\alpha, \beta) \mapsto x^\alpha g^{-\beta}.$$

What kind of function is f ? Note that

$$f(\alpha, \beta) = g^\delta \iff x^\alpha g^{-\beta} = g^\delta \iff g^{s\alpha - \beta} = g^\delta \iff s\alpha - \beta = \delta \pmod{N},$$

where s denotes the discrete logarithm of x . This means that the function is constant on the sets

$$f^{-1}(g^\delta) = \{(0, -\delta), (1, s - \delta), (2, 2s - \delta), \dots, (N - 1, (N - 1)s - \delta)\},$$

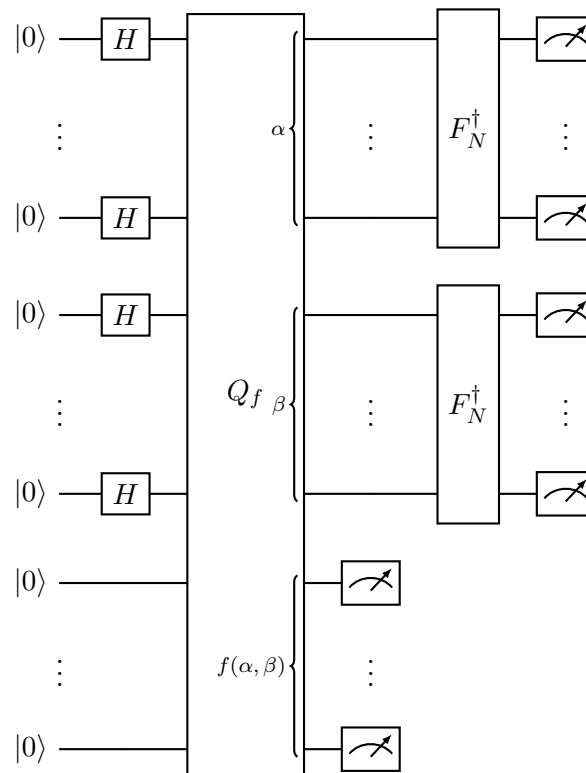
In particular it is an N -to-1 function. Note that we can think of the above as the set (in fact, a subgroup of $\mathbb{Z}_N \times \mathbb{Z}_N$)

$$H = \{(0, 0), (1, s), (2, 2s), \dots, (N - 1, (N - 1)s)\},$$

but “shifted” by $(0, -\delta)$. The idea behind our quantum algorithm for the discrete logarithm problem will be that we can efficiently generate a superposition over such a set, for a random δ .¹ Hopefully this is enough to recover the discrete logarithm s !

8.2 Shor’s quantum algorithm for discrete logs

We now describe Shor’s quantum algorithm for the discrete log problem, which shows that this is indeed the case. We shall assume $N = 2^n$. This is purely for convenience, since we did not explain how to implement the Fourier transform F_N by a quantum circuit when N is not a power of two (but it can be done using very similar ideas).² Consider the following quantum circuit:



This is a quantum circuit on $3n$ qubits. The basis states of the first n qubits encode the first argument, $\alpha \in \mathbb{Z}_N$, in the usual way, by identifying basis states $|\alpha\rangle \in (\mathbb{C}^2)^{\otimes n}$ with basis states $|\alpha\rangle \in \mathbb{C}^N$, using the binary expansion. Similarly, the second n qubits correspond to the second argument, $\beta \in \mathbb{Z}_N$. The last n qubits are used to store the function output in \mathbb{Z}_p^\times , which also has cardinality $N = p - 1$ (e.g., by identifying a number $y \in \mathbb{Z}_p^\times = \{1, \dots, N\}$ with the binary expansion of $y - 1 \in \{0, \dots, N - 1\}$). We indicate this by the braces. The pattern of the circuit looks similar to Simon’s algorithm.³

Thus, in the above circuit Q_f is not an “oracle” or “black box” (like in Simon’s problem), but it has to be implemented by a concrete quantum circuit that depends on the inputs p , g , and x . Fortunately, since the function f can be implemented by an efficient classical circuit, the quantum oracle Q_f can be implemented by an efficient quantum circuit, as you showed on a previous homework.

¹This is similarly to the situation of Simon’s algorithm, which relied on our ability to efficiently generate a superposition $\frac{1}{\sqrt{2}}(|\mathbf{x}\rangle + |\mathbf{x} \oplus \mathbf{s}\rangle)$, for a random “shift” \mathbf{x} .

²Note that if $N = 2^n$ then p is a so-called Fermat prime. These are actually not used in Diffie-Hellman, but again the crucial point is that nothing in our discussion depends on this fact.

³For folks who like Fourier transforms of abelian groups: $F_N \otimes F_N$ is the Fourier transform for the group $\mathbb{Z}_N \oplus \mathbb{Z}_N$.

What does the above circuit do? We compute the quantum state step by step:

$$|0, 0, 0\rangle \mapsto \frac{1}{N} \sum_{\alpha, \beta} |\alpha, \beta, 0\rangle \mapsto \frac{1}{N} \sum_{\alpha, \beta} |\alpha, \beta, f(\alpha, \beta)\rangle$$

Now we measure the last n qubits. Just like in Simon's algorithm, we see a uniformly random function value $g^\delta \in \mathbb{Z}_N^\times$, and the state of the first $2n$ qubits changes to

$$\frac{1}{\sqrt{N}} \sum_{\alpha} |\alpha, \alpha s - \delta\rangle.$$

Next, we apply the inverse Fourier transforms and obtain

$$\begin{aligned} \frac{1}{N^{3/2}} \sum_{\alpha, \mu, \nu} \omega_N^{-\mu\alpha} \omega_N^{-\nu(\alpha s - \delta)} |\mu, \nu\rangle &= \frac{1}{\sqrt{N}} \sum_{\mu, \nu} \left(\frac{1}{N} \sum_{\alpha} \omega_N^{-\mu\alpha - \nu(\alpha s - \delta)} \right) |\mu, \nu\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{\mu, \nu} \omega_N^{\nu\delta} \left(\frac{1}{N} \sum_{\alpha} \omega_N^{-(\mu + s\nu)\alpha} \right) |\mu, \nu\rangle = \frac{1}{\sqrt{N}} \sum_{\nu} \omega_N^{\nu\delta} | -s\nu, \nu\rangle \end{aligned}$$

since only the terms with $\mu + s\nu = 0$, i.e., $\mu = -s\nu$ survive. If we now measure, we obtain a pair $(-s\nu, \nu)$ of numbers modulo N . If ν is invertible, i.e., $\gcd(N, \nu) = 1$, then it has a multiplicative inverse ν^{-1} modulo N and we learn the discrete logarithm s by computing $s = -(-s\nu)\nu^{-1}$. By 'elementary' number theory, the probability that this is the case is $\Omega(1/\log \log N)$, so we only need to repeat this $O(\log \log N) = O(\log n) = \tilde{O}(1)$ times to succeed with high probability.

Moreover, running the circuit itself takes $\text{poly}(n)$ time – the most expensive part is Q_f , but this can be done in polynomial time by repeated squaring modulo N . Since the input to our discrete log problem are n -bit numbers, this means we can solve the discrete logarithm problem in polynomial time on a quantum computer!

Theorem 8.2: Shor's algorithm for discrete logs

There exists a quantum algorithm that solves the discrete logarithm problem ([Problem 8.1](#)) for n -bit numbers in time $\text{poly}(n)$, with probability $\geq 2/3$.

With the Diffie-Hellman cryptosystem disposed of, we will move on and break the RSA cryptosystem. This however requires some preparation.

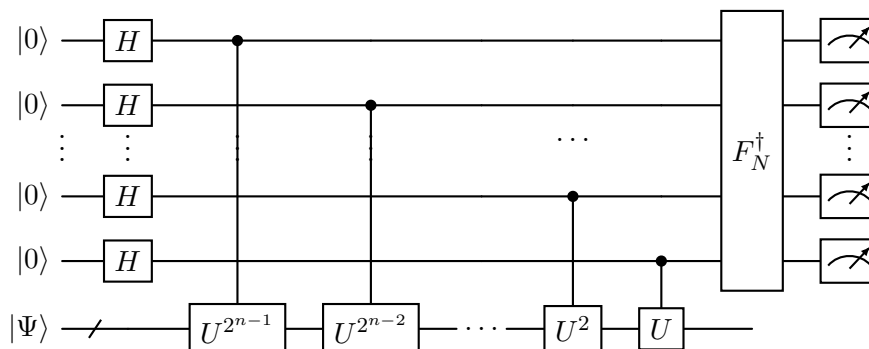
Chapter 9

Quantum phase estimation

Suppose you have a unitary U and someone hands you a quantum state $|\Psi\rangle$ that they promise is an eigenvector of U . How can you determine the corresponding eigenvalue, which is necessarily of the form $e^{2\pi i\phi}$ for some “phase” $0 \leq \phi < 1$?

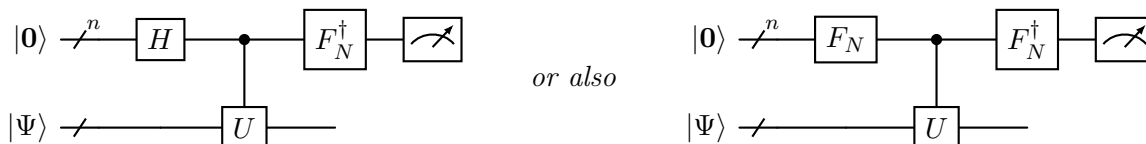
9.1 Quantum phase estimation in the simplest setting

On the homework you saw how to solve this problem exactly, by using the quantum Fourier transform, under the simplifying assumption that ϕ can be written exactly using n bits, i.e., $\phi = \frac{s}{N}$ for $N = 2^n$ and an integer $s \in \{0, \dots, N - 1\}$, by using a single run of the following circuit:



Namely, you showed that if $\phi = \frac{s}{N}$ then we obtain measurement outcome “ s ” with probability one. Recall that we identify numbers $s \in \{0, \dots, N - 1\}$ with bitstrings $\mathbf{s} = s_1 \dots s_n \in \{0, 1\}^n$ by the formula $s = s_1 2^{n-1} + s_2 2^{n-2} + \dots + s_{n-1} 2 + s_n$. Moreover, on another homework you could also show that if $|\Psi\rangle$ is a linear combination (superposition) $|\Psi\rangle = \sum_j \alpha_j |\Psi_j\rangle$, where each $|\Psi_j\rangle$ is a unit-norm eigenvector of the above type, with pairwise distinct $\phi_j = \frac{s_j}{N}$, then we see “ s_j ” with probability $|\alpha_j|^2$.

The above circuit can also be written in the following way, which might be easier to remember:



The first wire consists of n qubits and is identified with \mathbb{C}^N , and the “controlled unitary” should be interpreted as the unitary that sends

$$|x\rangle \otimes |\Psi\rangle \mapsto |x\rangle \otimes U^x |\Psi\rangle$$

for $x \in \{0, \dots, N - 1\}$. Here we used that $F_N |0\rangle = H^{\otimes n} |\mathbf{0}\rangle$.

9.2 Quantum phase estimation in the general case

We will now see that these results generalize to arbitrary unitaries, that is, even if the eigenvalues are *not* of the special form $\phi = \frac{s}{N}$.

Theorem 9.1: Quantum phase estimation

Suppose we can apply controlled powers of a unitary U and we have a state $|\Psi\rangle = \sum_j \alpha_j |\Psi_j\rangle$, where the $|\Psi_j\rangle$ are unit-norm eigenvectors of U with pairwise distinct eigenvalues $e^{2\pi i \phi_j}$. Then, the above circuit, applied with U , $|\Psi\rangle$, and the choice $n = b + O(\log(1/\varepsilon))$, outputs “ s ” such that $\hat{\phi} = \frac{s}{N}$ satisfies

$$|\hat{\phi} - \phi_j \bmod \mathbb{Z}| \leq \frac{1}{2^b} \quad (9.1)$$

(i.e., $\hat{\phi}$ is a b -bit approximation of ϕ_j), with probability at least $(1 - \varepsilon)|\alpha_j|^2$.

In particular, with probability at least $1 - \varepsilon$, our estimate $\hat{\phi}$ is close to *some* eigenvalue ϕ_j . Thus ε can be interpreted as a failure probability. Note also that the guarantees of quantum phase estimation can also be written as follows: if you would like to achieve

$$|\hat{\phi} - \phi_j \bmod \mathbb{Z}| \leq \gamma$$

with probability at least $(1 - \varepsilon)|\alpha_j|^2$, then you should choose $n = O(\log(1/\varepsilon) + \log(1/\gamma))$.

Remark 9.2

What does “ $|\dots \bmod \mathbb{Z}|$ ” in Ineq. (9.1) mean? Remember that ϕ and ϕ' determine the same complex number $e^{2\pi i \phi} = e^{2\pi i \phi'}$ if and only if $\phi - \phi'$ differ by an integer. Thus, when we measure the distance between two angles we should measure the distance modulo \mathbb{Z} . That is, we define:

$$|\phi - \phi' \bmod \mathbb{Z}| := \min_{k \in \mathbb{Z}} |\phi - \phi' - k|$$

Geometrically this means that we imagine the “phases” ϕ and ϕ' as points on a circle (with circumference one) and we measure the *shortest* distance between them along the circle. See also the picture below.

Proof. We now prove that the above quantum circuit indeed does the job. We start with the case when $|\Psi\rangle$ is an eigenvector, with eigenvalue $e^{2\pi i \phi}$ for some $0 \leq \phi \leq 1$, and show that the above circuit can still give a good approximation of ϕ if we choose n large enough. Let us write

$$\phi = \frac{s}{N} + \delta \quad \text{where } s \in \{0, 1, \dots, N-1\} \text{ and } 0 \leq \delta \leq \frac{1}{N}.$$

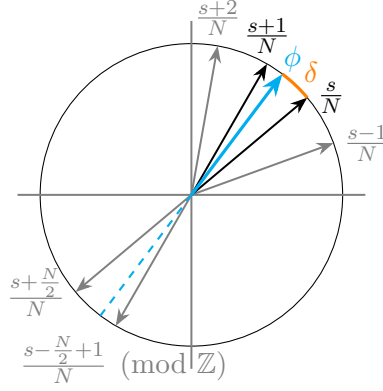
We will show that the probability of “bad” outcomes is small. When is an outcome $y \in \mathbb{Z}_N$ good or bad? How should we even translate outcomes into estimates of the eigenvalue? Inspired by what you showed on the homework, if the measurement outcome is “ y ” then we will estimate ϕ by

$$\hat{\phi} = \frac{y}{N}$$

How well does this procedure work? To analyze the situation, let us write

$$y = s + \Delta \pmod{N}, \quad \text{where } \Delta \in \left\{ -\frac{N}{2} + 1, \dots, \frac{N}{2} \right\}.$$

We can visualize this as follows:



We see from the picture that we estimate the phase ϕ by $\hat{\phi} = \frac{y}{N}$ then the error in the phase is precisely computed by

$$|\hat{\phi} - \phi \bmod \mathbb{Z}| = \left| \frac{\Delta}{N} - \delta \right| \leq \frac{1}{2} \quad (9.2)$$

In particular, the measurement outcomes for which

$$|\Delta| \leq M := \frac{N}{2^b} - 1 = 2^{n-b} - 1 \quad (9.3)$$

are such that the error in the phase is at most

$$\left| \frac{\Delta}{N} - \delta \right| \leq \frac{|\Delta|}{N} + \delta \leq \frac{1}{2^b} - \frac{1}{N} + \frac{1}{N} = \frac{1}{2^b},$$

i.e., we have found a b -bit approximation of ϕ ! Those are our good outcomes.

We now compute the probability of outcomes. The state of right before the inverse QFT is given by

$$\begin{aligned} & \frac{1}{\sqrt{N}} \sum_{\mathbf{x} \in \{0,1\}^N} |\mathbf{x}\rangle \otimes U^{x_n} U^{2x_{n-1}} \dots U^{2^{n-2}x_2} U^{2^{n-1}x_1} |\Psi\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{\mathbf{x} \in \{0,1\}^N} |\mathbf{x}\rangle \otimes U^{\mathbf{x}} |\Psi\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{\mathbf{x} \in \{0,1\}^N} |\mathbf{x}\rangle \otimes e^{2\pi i \mathbf{x} \phi} |\Psi\rangle \\ &= \left(\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} e^{2\pi i x \phi} |x\rangle \right) \otimes |\Psi\rangle. \end{aligned}$$

Hence after applying the inverse QFT we obtain the following state on the first n qubits:

$$\frac{1}{N} \sum_{y=0}^{N-1} \sum_{x=0}^{N-1} e^{2\pi i x \phi} \omega_N^{-xy} |y\rangle = \sum_{y=0}^{N-1} \left(\frac{1}{N} \sum_{x=0}^{N-1} e^{2\pi i x (\frac{s-y}{N} + \delta)} \right) |y\rangle.$$

Accordingly,

$$\begin{aligned} \mathbf{Pr}(\text{outcome } y) &= \left| \frac{1}{N} \sum_{x=0}^{N-1} e^{2\pi i x (\frac{s-y}{N} + \delta)} \right|^2 = \left| \frac{1}{N} \sum_{x=0}^{N-1} e^{2\pi i x (\frac{\Delta}{N} - \delta)} \right|^2 \\ &= \left| \frac{1}{N} \frac{1 - e^{2\pi i (\Delta - N\delta)}}{1 - e^{2\pi i (\frac{\Delta}{N} - \delta)}} \right|^2 = \left| \frac{1}{N} \frac{1 - e^{-2\pi i N\delta}}{1 - e^{2\pi i (\frac{\Delta}{N} - \delta)}} \right|^2 \\ &= \frac{1}{N^2} \frac{\sin^2(-\pi N\delta)}{\sin^2(\pi (\frac{\Delta}{N} - \delta))} \leq \frac{1}{4N^2} \frac{1}{(\frac{\Delta}{N} - \delta)^2} = \frac{1}{4} \frac{1}{(\Delta - N\delta)^2}, \end{aligned}$$

first using a geometric sum, then that Δ is an integer, and finally that $|\sin(y)| \leq 1$ for all $y \in \mathbb{R}$, while we have $|\sin(y)| = \sin|y| \geq \frac{2}{\pi}|y|$ for $|y| \leq \pi/2$. We can thus bound the probability of “bad” outcomes where the deviation Δ is large:

$$\begin{aligned} \Pr(|\Delta| > M) &\leq \sum_{x=-N/2+1}^{-M-1} \frac{1}{4} \frac{1}{(x - N\delta)^2} + \sum_{x=M+1}^{N/2} \frac{1}{4} \frac{1}{(x - N\delta)^2} \\ &\leq \frac{1}{4} \sum_{x=-N/2+1}^{-M-1} \frac{1}{x^2} + \frac{1}{4} \sum_{x=M+1}^{N/2} \frac{1}{(x-1)^2} \leq \frac{1}{2} \sum_{x=M}^{\infty} \frac{1}{x^2} \\ &\leq \frac{1}{2} \int_{M-1}^{\infty} \frac{1}{x^2} dx = \frac{1}{2} \left[-\frac{1}{x} \right]_{x=M-1}^{\infty} = \frac{1}{2(M-1)}. \end{aligned}$$

For our choice of $M = 2^{n-b} - 1$ in Ineq. (9.3), we have

$$\Pr(|\Delta| > M) \leq \varepsilon$$

provided we choose

$$n = b + \left\lceil \log \left(\frac{1}{2\varepsilon} + 2 \right) \right\rceil = b + O(\log \frac{1}{\varepsilon}).$$

In conclusion, we find that for this choice of n , quantum phase estimation returns a b -bit estimate of the phase ϕ of the eigenvalue of Φ up to a failure probability of $\leq \varepsilon$.

What happens if $|\Psi\rangle$ is not an eigenvector? We can always write

$$|\Psi\rangle = \sum_j \alpha_j |\Psi_j\rangle,$$

where the α_j are complex numbers and the $|\Psi_j\rangle$ are (necessarily orthogonal) unit norm eigenvectors with pairwise distinct eigenvalues $e^{2\pi i \phi_j}$ of U . Suppose that we run the above circuit on $|\Psi_j\rangle$ until right before the measurement. Then the overall state is of the form

$$|\Phi_j\rangle \otimes |\Psi_j\rangle,$$

where $|\Phi_j\rangle$ is an n -qubit state such that when we measure it we get an outcome y such that $\hat{\phi} = \frac{y}{N}$ is a b -bit estimate of ϕ_j with probability $\geq 1 - \varepsilon$. By linearity, if we run the above circuit with $|\Psi\rangle$ then the state right before the measurement becomes

$$\sum_j \alpha_j |\Phi_j\rangle \otimes |\Psi_j\rangle.$$

and hence the probability of outcome y is given by

$$\Pr(\text{outcome } y | \Psi) = \left\| \sum_j \alpha_j \langle y | \Phi_j \rangle |\Psi_j\rangle \right\|^2 = \sum_j |\alpha_j|^2 |\langle y | \Phi_j \rangle|^2 = \sum_j |\alpha_j|^2 \Pr(\text{outcome } y | \Psi_j)$$

where we used that the $|\Psi_j\rangle$ are pairwise orthonormal. Thus the probability of outcomes is the same as if we first picked j at random according to $p_j = |\alpha_j|^2$ and then ran the phase estimation circuit with $|\Psi_j\rangle$ in place of $|\Psi\rangle$.¹ We conclude that we obtain a b -bit approximation of eigenvalue ϕ_j with probability at least $(1 - \varepsilon)|\alpha_j|^2$. In particular, since $\sum_j |\alpha_j|^2 = \|\Psi\|^2 = 1$, we obtain a b -bit approximation of *some* eigenvalue ϕ_j with probability at least $1 - \varepsilon$. This establishes the result announced above. \square

¹This can also be seen by imagining a fictitious measurement in an eigenbasis of U , as in our analysis of Simon’s algorithm.

Chapter 10

Shor's quantum algorithm for factoring by order finding

Before we begin, recall some basics about modular arithmetic. While addition in \mathbb{Z}_M is always invertible, this is not in general true for multiplication. However, let $a \in \mathbb{Z}_M$ with $\gcd(a, M) = 1$. Then, and only then, a has a multiplicative inverse modulo M , meaning that there exists b such that $ab = 1 \pmod{M}$. We will say that a is invertible modulo M . The set of all such elements in \mathbb{Z}_M ,

$$\mathbb{Z}_M^\times = \{a \in \mathbb{Z}_M : \gcd(a, M) = 1\},$$

is called the *unit group*. For a prime number P , clearly $\gcd(a, P) = 1$ holds for all nonzero $a \neq 0 \pmod{P}$, so $\mathbb{Z}_P^\times = \{1, \dots, P-1\}$ as we discussed in the context of the discrete log problem, but if M is a composite number then the structure of \mathbb{Z}_M^\times is more complicated.

In any case, whether M is prime or not, if a is invertible modulo M then there exists a positive integer r such that $a^r = 1 \pmod{M}$.¹ The least such r is called the *order* of a modulo M . This is a basic notion in group theory and “elementary” number theory, but it is also interesting in applications. In particular, we will see that it allows us to find a nontrivial factor of a composite number, as we will see below. But first we recall the RSA cryptosystem, whose security depends on the difficulty of factoring.

10.1 Motivation: Breaking RSA via factoring

Recall the *RSA public key cryptosystem* in its simplest form. To generate a key pair, we proceed as follows:

1. Pick two large distinct primes P and Q and compute $M = PQ$.
2. Compute $\lambda := \text{lcm}(P-1, Q-1)$.
3. Choose some $e \in \{2, \dots, \lambda-1\}$ such that $\gcd(e, \lambda) = 1$. Thus e is invertible modulo λ . Compute its inverse d modulo λ .
4. Then the public key is (M, e) , and the private key is d .

To encrypt a message $A \in \mathbb{Z}_M$, simply compute the ciphertext $B = A^e \pmod{M}$. Then the ciphertext B can be decrypted by computing $B^d = A^{ed} = A \pmod{M}$. The last equality is not hard to see using the Chinese remainder theorem.²

Clearly, the cryptosystem would be completely broken if we could factor the number M (which is part of the public key). Indeed, once we know P and Q we can compute λ , and then d as the multiplicative inverse of e modulo λ . Fortunately, no efficient classical algorithms are known for factoring, as we discussed in [Chapter 1](#).

¹Why is this so? The sequence $a, a^2, a^3, \dots \pmod{M}$ at some point has to repeat itself, say, $a^k = a \pmod{M}$. In other words, $(a^{k-1} - 1)a = 0 \pmod{M}$. If we multiply this equation with the inverse b of a , then we obtain $a^{k-1} - 1 = (a^{k-1} - 1)ab = 0b = 0 \pmod{M}$, hence $a^{k-1} = 1 \pmod{M}$, as we wanted to show.

²By the Chinese remainder theorem, $\mathbb{Z}_M \cong \mathbb{Z}_P \times \mathbb{Z}_Q$, with componentwise multiplication. Thus it suffices to show that $a^{ed} = a \pmod{P}$ for every $a \in \mathbb{Z}_P$ (then this also holds for Q in place of P). Clearly, $a^{ed} = a$ for $a = 0$, so we may assume that $a \neq 0$. But then $a \in \mathbb{Z}_P^\times$, hence $a^{P-1} = 1$ (this holds for generators of \mathbb{Z}_P^\times and hence for any element of \mathbb{Z}_P^\times), hence $a^\lambda = 1$ (since λ is a multiple of $P-1$), and hence $a^{ed} = a$ (since $ed = 1 \pmod{\lambda}$ we can write $ed = k\lambda + 1$ for some k).

10.2 Factoring from order finding

However, we will now show that quantum computers can factor numbers in polynomial time. In more detail, we will see that factoring can be reduced (by a completely classical procedure) to another problem, *order finding*, and that there exists a quantum algorithm due to Shor that solves the latter in polynomial time. This quantum algorithm is just an application of quantum phase estimation, which we saw last week.

We first state the order finding problem formally and then discuss how it is related to factoring.

Problem 10.1: Order finding

Given two integers M and $a \in \{1, \dots, M\}$ such that $\gcd(a, M) = 1$, determine the order of a modulo M . That is, determine the smallest integer $r \geq 1$ such that $a^r = 1 \pmod{M}$.

Now suppose that we want to find a nontrivial factor of M . We may assume that M is odd (otherwise 2 is a factor!) and that it is not a power (if it is a power, i.e., $M = b^c$, then b can be efficiently found and is of course a nontrivial factor). Pick a uniformly random $a \in \{2, \dots, M-1\}$. If $\gcd(a, M) \neq 1$, this GCD is a nontrivial factor of M , so we are done! Otherwise, if $\gcd(a, M) = 1$, then a is invertible modulo M , as explained above. Let r denote the order of a modulo M . By ‘elementary’ number theory, with probability at least $1/2$, r is even and moreover $M \nmid a^{r/2} + 1$. Since r is the order of a , we also have that $M \nmid a^{r/2} - 1$. But then

$$M \mid a^r - 1 = (a^{r/2} + 1)(a^{r/2} - 1),$$

and neither factor on the right-hand side is a multiple of M . This means that $\gcd(M, a^{r/2} + 1)$ and $\gcd(M, a^{r/2} - 1)$ must be nontrivial factors of M ! You saw this very concretely on last week’s practice set. We conclude that computing the order of elements modulo M indeed suffices to find a nontrivial factor of M .

10.3 Shor’s quantum order finding algorithm

While we do not know any efficient classical algorithms for order finding (after all, we don’t know how to factor efficiently), Shor discovered in 1994 that there is an efficient *quantum algorithm for order finding*. For an m -bit number, it requires $O(m^2 \log m \log \log m)$ operations. Given our discussion of Fourier transforms this might seem quite natural, since the order is nothing but the period of the sequence

$$a^0 = 1, \quad a^1 = a, \quad a^2 \pmod{M}, \quad \dots, \quad a^r = 1 \pmod{M}, \quad a^{r+1} = a \pmod{M}, \quad \dots,$$

i.e., the period of the function $f(x) = a^x \pmod{M}$, and it seems perhaps intuitive that the quantum Fourier transform should help us determine this period very efficiently on quantum computers. However, note this is not completely straightforward since the order finding problem is *not* simply an instance of Simon’s problem for \mathbb{Z}_M , since while the map f is periodic as a function of $x \in \mathbb{Z}$, this is not so if we take, e.g., $x \in \mathbb{Z}_M$. Of course it is periodic with respect to $x \in \mathbb{Z}_r$ or any multiple of order r (e.g., the size of the unit group \mathbb{Z}_M^\times), but the order is precisely what we want to determine in the first place! Nevertheless, our intuition is correct and we will now show that there is a quantum algorithm that efficiently solves the order finding problem.

The key idea is to use quantum phase estimation for the following unitary:

$$U: \mathbb{C}^M \rightarrow \mathbb{C}^M, \quad |y\rangle \mapsto |ay \pmod{M}\rangle$$

That U is indeed a unitary follows because a has a multiplicative inverse modulo M . We will typically embed $\mathbb{C}^M \subseteq (\mathbb{C}^2)^{\otimes m}$, where m is the number of bits required to write M . To see why U is the right object to consider, note that it has eigenvectors

$$|\Psi_j\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \omega_r^{-jk} |a^k\rangle$$

for $j \in \mathbb{Z}_r$. Since

$$U |\Psi_j\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \omega_r^{-jk} |a^{k+1}\rangle = \omega_r^j \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \omega_r^{-j(k+1)} |a^{k+1}\rangle = \omega_r^j |\Psi_j\rangle$$

thus $|\Psi_j\rangle$ is an eigenvector with eigenvalue $\omega_r^j = e^{2\pi i \phi_j}$, where $\phi_j = \frac{j}{r}$.³ Of course, applying quantum phase estimation to U and $|\Psi_j\rangle$ directly is out of the question, since the formula for the latter involves the order r which is what we are interested in computing in the first place. However, observe the following beautiful fact:

$$\frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} |\Psi_j\rangle = \sum_{k=0}^{r-1} \left(\frac{1}{r} \sum_{j=0}^{r-1} \omega_r^{-jk} \right) |a^k\rangle = |a^0\rangle = |1\rangle.$$

This means that if we run quantum phase estimation for the unitary U and the state $|\Psi\rangle = |1\rangle$, which is of course straightforward to prepare, we get an approximation

$$\hat{\phi} \approx j/r$$

for some random $j \in \{0, \dots, r-1\}$.

Of course we are interested in the precise order r , so an approximation seems not good enough. However, suppose that we carry out phase estimation with $n = 2m + 1 + O(\log(1/\varepsilon))$ qubits. Then we obtain some $\hat{\phi}$ such that

$$\left| \hat{\phi} - \frac{j}{r} \right| \leq \frac{1}{2^{2m+1}} \leq \frac{1}{2M^2} \quad (10.1)$$

for (near uniformly) random $j \in \{0, 1, \dots, r-1\}$ (provided ε is small). The key point now is the following: We know that $\frac{j}{r}$ can be written as a reduced fraction with denominator $\leq r < M$. We claim that there exists at most one such number that is $\frac{1}{2M^2}$ -close to $\hat{\phi}$. Indeed, if $\frac{p}{q} \neq \frac{p'}{q'}$ are two fractions with denominator $q, q' < M$, then $|\frac{p}{q} - \frac{p'}{q'}| = |\frac{pq' - p'q}{qq'}| \geq \frac{1}{qq'} \geq \frac{1}{M^2}$. Thus Ineq. (10.1) ensures that *in principle* it is possible to uniquely determine j/r from $\hat{\phi}$. This can also be done efficiently *in practice*, for example by using *continued fractions*; see the lecture notes by de Wolf or the book by Nielsen and Chuang.

There is one last issue that we need to worry about in our use of phase estimation: While it seems clear that one can implement U efficiently on a quantum computer, how about its controlled powers? In fact, we can implement the entire sequence of controlled- U^{2^s} operations used by quantum phase estimation very efficiently: Note that what we want to implement is

$$|x, y\rangle \mapsto |x\rangle \otimes U^{2^{n-1}x_1} U^{2^{n-2}x_2} \dots U^{2x_{n-1}} U^{x_n} |y\rangle = |x\rangle \otimes U^x |y\rangle = |x\rangle \otimes |a^x y \bmod M\rangle,$$

where $x \in \{0, \dots, 2^n - 1\}$. In essence this boils down to computing the *modular exponentiation* $a^x \bmod M$ in a reversible way, and this can be done in time $O(nm \log m \log \log m)$ by using a fast multiplication subroutine mentioned in the last lecture (or in time $O(nm^2)$ using the naive one). You explored this on the homework. By our choice of n this is $O(m^2 \log m \log \log m)$ for fixed ε .

We are almost done – using quantum phase estimation and continued fractions we can find a *reduced* fraction that is equal to j/r for some uniformly random $j \in \{0, 1, \dots, r-1\}$. How can we determine the order r from this? Similarly as in our solution to Simon's algorithm for \mathbb{Z}_N , suppose that we run the above procedure twice, obtaining reduced fractions $\frac{p}{q} = \frac{j}{r}$ and $\frac{p'}{q'} = \frac{j'}{r}$. We are interested in the probability that j and j' have no common divisor, since in this case we can determine the order as $r = \text{lcm}(q, q')$. But

$$\Pr(j \text{ and } j' \text{ have a common divisor}) \leq \sum_{p \text{ prime}} \Pr(p|j_1 \text{ and } p|j_2) = \sum_{p \text{ prime}} \frac{1}{p^2} \leq \sum_{k=2}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6} - 1 \approx 64.5\%$$

and hence we learn the order r with probability at least $\approx 35.5\%$ (assuming the phase estimation subroutine did not fail either). You discussed this on a practice set. By merely repeating the above a suitable number of times we obtain the order of r with any desired probability of success. In summary:

³Note that the $|\Psi_j\rangle$ are just (inverse) Fourier modes for the cyclic group $\{a, a^2, \dots, a^r = 1\}$, which is isomorphic to \mathbb{Z}_r , so the calculations here are really the same that we did earlier when verifying the properties of the Fourier transform.

Theorem 10.2: Shor's algorithm for order finding

There is a quantum algorithm that, given an m -bit integer M and another integer $1 \leq a < M$, determines the order of a modulo M ([Problem 10.1](#)) in time $O(m^2 \log m \log \log m)$ with any desired constant probability of success.

As a consequence of [Theorem 10.2](#) and the discussion in [Section 10.2](#), we see that one can find a nontrivial factor of an m -bit integer M in time $O(m^2 \log m \log \log m)$, with any desired constant probability of success. In short: on a quantum computer, we can factor numbers in polynomial time!

Chapter 11

Quantum query complexity

In [Chapters 4 to 10](#) and in the exercises we discussed many computational problems where we saw that quantum computers could get an advantage over ordinary “classical” ones. But how can we be sure that we found optimal quantum algorithms for these problems? Similarly, what if you meet a really difficult computational problem in the future and you would like to show that even quantum computers cannot solve your problem efficiently? Such questions are studied in *quantum complexity theory*, which is also concerned with defining the quantum analogs of complexity classes such as P and NP.

11.1 Introduction and definitions

In this chapter, we will study the simplest possible setting which nevertheless is of great importance, namely computational problems that get their input by querying an oracle or black box and we care about solving the problem with a minimal number of queries. This subject is called *quantum query complexity*, and we already motivated it in [Section 4.4](#). Here is a short summary and taxonomy of some oracle problems that we met:

| | Search problem | Decision problem |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Promise problem | <p>Bernstein-Vazirani problem (see exercises)</p> <p>Problem: <i>Given query access to a “linear” function $g(\mathbf{x}) = \mathbf{x} \cdot \mathbf{s}$, find $\mathbf{s} \in \{0, 1\}^n$.</i></p> <ul style="list-style-type: none"> • Classical deterministic: n queries sufficient • Classical randomized: $\Omega(n)$ necessary • Quantum: 1 query necessary and sufficient | <p>Deutsch-Jozsa problem (see exercises)</p> <p>Problem: <i>Given query access to a function $g: \{0, 1\}^n \rightarrow \{0, 1\}$ that is either constant or balanced, which is the case?</i></p> <ul style="list-style-type: none"> • Classical deterministic: $\Theta(2^n)$ • Classical randomized: $\Theta(1)$ • Quantum: 1 query necessary and sufficient |
| | <p>Simon’s problem (Chapter 5)</p> <p>Problem: <i>Given query access to a “periodic” function f on $\{0, 1\}^n$, find the period!</i></p> <ul style="list-style-type: none"> • Classical randomized: $\Theta(\sqrt{2^n})$ queries • Quantum: $O(n)$ queries sufficient | <p>Simon’s decision problem (Section 5.3.1)</p> <p>Problem: <i>Given query access to a function f on $\{0, 1\}^n$ that is either “periodic” or bijective, which is the case?</i></p> <ul style="list-style-type: none"> • Classical randomized: $\Theta(\sqrt{2^n})$ queries • Quantum: $O(n)$ queries sufficient |
| Total problem | <p>Grover’s search problem (Chapter 6)</p> <p>Problem: <i>Given query access to $g: \{0, 1\}^n \rightarrow \{0, 1\}$, find $\mathbf{x} \in \{0, 1\}^n$ s.th. $g(\mathbf{x}) = 1$ or report no such \mathbf{x} exists.</i></p> <ul style="list-style-type: none"> • Classical deterministic: N queries necessary and sufficient • Classical randomized: $\Omega(N)$ necessary • Quantum: $O(\sqrt{N})$ queries sufficient | <p>“Decision Grover” a.k.a. OR_N (see below)</p> <p>Problem: <i>Given query access to $g: \{0, 1\}^n \rightarrow \{0, 1\}$, does there exist some $\mathbf{x} \in \{0, 1\}^n$ such that $g(\mathbf{x}) = 1$? In other words, compute the OR of the N bits in the function table (see Remark 11.1).</i></p> <ul style="list-style-type: none"> • Classical deterministic: N queries necessary and sufficient • Classical randomized: $\Omega(N)$ necessary • Quantum: $O(\sqrt{N})$ queries sufficient |

Deutsch’s problem ([Section 4.3](#))

Problem: *Given query access to a function $g: \{0, 1\} \rightarrow \{0, 1\}$, is it constant?*

- *Classical deterministic: 2 queries sufficient*
- *Classical randomized: 2 queries necessary*
- *Quantum: 1 query necessary and sufficient*

Here we distinguish between promise and total problems, as well as between search and decision problems. *Promise problems* are ones where the input is promised to have some special property (e.g., that the function is linear, or periodic, or either constant or balanced, etc.), while *total problems* are defined for any input. *Decision problems* are problems such that the solution is a single bit (typically, whether the input satisfies some property), in contrast to *search problems* which ask us to return an entire bitstring.

You will know from your introductory TCS course that computational problems often come in different variants. For example, Simon’s problem came in two variants ([Chapter 5](#)). Similarly, while the search problem ([Problem 6.1](#)) asks us to find a solution (or report that none exists), the corresponding decision problem simply asks if there exists a solution – we refer to it as “decision Grover” in the above table (the stated complexities follow directly from the results of [Chapter 6](#)).

Remark 11.1: Oracle vs. memory

We often think of the oracle as a function such as $g: \{0, 1\}^n \rightarrow \{0, 1\}$ which we would call to make a query. Of course, we can also equivalently think of it as a function $\{0, 1, \dots, N - 1\} \rightarrow \{0, 1\}$, or simply as a bitstring of length $N = 2^n$, namely as the corresponding function table:

$$g \equiv \boxed{g(0\dots 00)} \boxed{g(0\dots 01)} \quad \dots \quad \boxed{g(1\dots 11)} \equiv \boxed{g(0)} \boxed{g(1)} \quad \dots \quad \boxed{g(N-1)}$$

In other words, the input to our computational problems can also simply be thought of as a bitstring of size N stored in some memory and we are asking how many classical or quantum “memory accesses” are needed to decide some property or compute some Boolean function about it.

In this sense, the “decision Grover” problem simply asks if there is at least one 1-bit in the function table. In other words, it asks us to compute the N -bit OR function on the function table:

$$\text{OR}_N(g) := \text{OR}_N(g(0), g(1), \dots, g(N - 1)).$$

This explains the terminology in the last entry of the table.

Note that for most of the problems above, you do not currently know whether the quantum algorithms that we found are optimal. To study this more systematically, let us first make some definitions so that we know exactly what we are talking about. For simplicity we restrict to decision problems (which can be total or promise problems).

Definition 11.2: Query complexities

Let P be a decision problem which takes as input an n -bit function (or, equivalently, a bitstring of size $N = 2^n$). Then we define:

- The *deterministic query complexity* $D(P)$ of P is the minimum number of queries (to g) needed by any deterministic classical algorithm to solve this problem for any input $g: \{0, 1\}^n \rightarrow \{0, 1\}$.
- The *randomized query complexity* $R(P)$ of P is the minimum number of queries (to g) needed by any randomized classical algorithm to solve this problem with probability $\geq 2/3$ for any input $g: \{0, 1\}^n \rightarrow \{0, 1\}$.
- The *quantum complexity* $Q(P)$ of P is the minimum number of queries (to Q_g , see [Definition 4.4](#)) needed by any quantum algorithm to solve this problem with probability $\geq 2/3$ for any in-

put $g: \{0, 1\}^n \rightarrow \{0, 1\}$.

As discussed earlier the choice of success probability $2/3$ is mostly completely arbitrary. Clearly:

$$D(P) \geq R(P) \geq Q(P)$$

There are two natural questions:

1. *How large can the gap be?*
2. *How to prove that $Q(P) \geq \dots$ for some concrete problem P ?*

Regarding the first question, we already know that this can be exponential for promise problems. E.g., for Simon's (decision) problem we have $R = \Omega(\sqrt{2^n})$, while $Q = O(n)$. Interestingly, for *total* decision problems, quantum algorithms (and hence also randomized algorithms) offer at most a polynomial speedup in terms of query complexity!

Theorem 11.3: No super-polynomial quantum speedup for total decision problems

Let P be a total decision problem. Then, $D(P) \leq R^3(P)$ and $R(P) \leq D(P) \leq Q^6(P)$.

The remainder of this chapter will be concerned with the second question. To this end, we will derive a simple version of an extremely powerful technique called the “adversary bound”.

11.2 The adversary bounds

Recall that when we argued that Grover's search problem was difficult to solve on ordinary *classical* computers, we argued that even just detecting whether (1) there exists precisely one solution or (2) there is no solution at all should require $\Omega(N)$ queries. Intuitively, these two cases should be difficult to tell apart, since by flipping a single bit we can turn a “no” instance into a “yes” instance, and vice versa – meaning that we need to look at all N bits or at least a constant fraction of them to decide between the two options with constant probability of success. While quantum queries can access multiple bits in superposition, it still seems plausible that the above should also be the most difficult instances for quantum computers, except for the kind of “square root” advantage that we already know we may get from Grover's algorithm.

The adversary bound formalizes this intuition. To state it in its simplest version, let us introduce some notation: If P is a decision problem and f an input or *instance*, we will write

$$P(f) = \begin{cases} 1 & \text{if } f \text{ is a “yes”-instance of } P, \\ 0 & \text{if } f \text{ is a “no”-instance of } P. \end{cases}$$

Then, $P^{-1}(1)$ consists of the inputs for which the answer to the decision problem is “yes”, and $P^{-1}(0)$ consists of the ones for which the answer is “no”. To measure whether two inputs are close, we count the number of bits in the function table that have to be modified in order to convert one into the other. That is, for any two functions $f, g: \{0, 1\}^n \rightarrow \{0, 1\}$, we define

$$d(f, g) := |\{\mathbf{x} \in \{0, 1\}^n : f(\mathbf{x}) \neq g(\mathbf{x})\}|$$

Then we have the following result which we call the “super-basic adversary bound”:

Theorem 11.4: Super-basic adversary bound

Let P be a decision problem, $F \subseteq P^{-1}(0)$, and $G \subseteq P^{-1}(1)$. For every $f \in F$, let $G_f := \{g \in G : d(f, g) = 1\}$, and for every $g \in G$, let $F_g := \{f \in F : d(f, g) = 1\}$. Then,

$$Q(P) = \Omega\left(\sqrt{m\tilde{m}}\right),$$

where $m := \min_{f \in F} |G_f|$ and $\tilde{m} := \min_{g \in G} |F_g|$.

To apply this bound, we need to select some “no”-instances F and some “yes”-instances G . These should be selected such that there are many ways of converting any “no”-instance into a “yes”-instance by modifying one bit, and vice versa. Indeed, note that m is precisely defined such that for any “no”-instance in F , we can obtain at least m “yes”-instances by modifying a single bit, and similarly for \tilde{m} .

Example 11.5: Grover is optimal

Let us apply this to the decision version of Grover’s problem, which asks if a given function has a solution (equivalently, the goal is to compute the OR_N of the function table). For F we have no choice but to take the function that is everywhere zero, while for G we take the set of all n -bit functions that have a single solution (following the intuition given above):

$$F = \left\{ \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \right\}$$

$$G = \left\{ \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \end{bmatrix}, \dots, \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \right\}$$

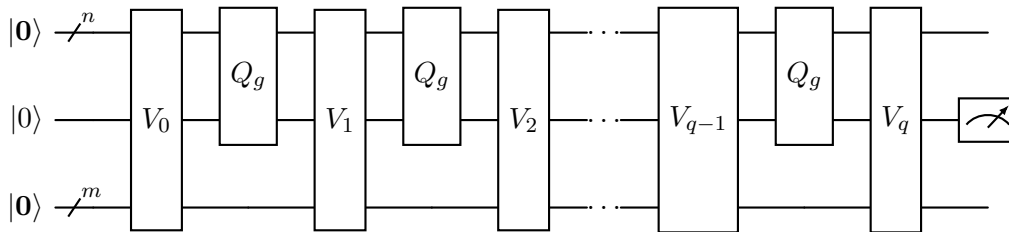
Clearly, $m = N$ (we can change any bit in the function table of the everywhere-zero function to get a function with a single one), while $\tilde{m} = 1$ (for any function in G we have to erase the single one bit in the function table to obtain the everywhere-zero function). Now [Theorem 11.4](#) yields the lower bound

$$Q(\text{OR}_N) = \Omega(\sqrt{N}).$$

The same is true for the search problem, since it is at least as difficult as the decision problem. On the other hand, we know that Grover’s algorithm achieves this query complexity. Thus we may conclude that the query complexity of both the search and the decision problem is $\Theta(\sqrt{N})$.

We will now prove the super-basic adversary bound.

Proof of Theorem 11.4. Consider any quantum algorithm that solves the problem with probability $\geq \frac{2}{3}$ using q queries. Without loss of generality, we can assume it has the following form:¹



where the result of the query algorithm is determined by the outcome of the measurement at the end (1 = yes, 0 = no). Let $|\Psi_g(t)\rangle$ denote the state right after the unitary V_t which is applied after the t -th query to the standard quantum oracle Q_g . That is:

$$|\Psi_g(t)\rangle = \begin{cases} V_0 |0, 0, 0\rangle & \text{if } t = 0, \\ V_t(Q_g \otimes I) |\Psi_g(t-1)\rangle & \text{if } t > 0 \end{cases}$$

We observe:

- For $t = 0$, the states $|\Psi_g(t)\rangle$ do not depend on the input g , since we have not yet made any query. In particular:

$$|\langle \Psi_f(0) | \Psi_g(0) \rangle| = 1 \quad (\forall f \in F, g \in G) \tag{11.1}$$

¹The first n qubits correspond to the input register of the quantum oracle, the middle qubit to its output, and the last m qubits are some arbitrary scratch space used by the algorithm. The unitaries V_0, V_1, \dots, V_q are obtained by grouping together all gates that happen between any two quantum queries Q_g , or before the first or after the last query, into a single unitary. By suitably permuting wires, we can always assume that we measure the middle qubit to obtain the result of the query algorithm (1 = yes, 0 = no). All other measurements can be replaced by controlled quantum gates, as we know from the exercises.

- For $t = q$, the states $|\Psi_g(t)\rangle$ are the states right before the measurement. Since the algorithm is assumed to solve the decision problem with probability $\geq \frac{2}{3}$, the states $|\Psi_f(q)\rangle$ for $f \in F$ (for which the answer is “no”) should be well distinguishable from the states $|\Psi_g(t)\rangle$ for $g \in G$ (for which the correct answer is “yes”). Indeed, you will show on the practice set that

$$|\langle \Psi_f(q) | \Psi_g(q) \rangle| \leq \sqrt{\frac{8}{9}} \quad (\forall f \in F, g \in G) \quad (11.2)$$

Thus all we need to do is to show that $\langle \Psi_f(t) | \Psi_g(t) \rangle$ changes little in each step! To formalize this idea, let us define

$$\Delta(t) := \sum_{(f,g) \in R} |\langle \Psi_f(t) | \Psi_g(t) \rangle|,$$

where

$$R := \{(f, g) \in F \times G : d(f, g) = 1\}.$$

We note that $|R| \geq |F| \cdot m$ and $|R| \geq |G| \cdot \tilde{m}$, hence

$$|F| \leq \frac{|R|}{m} \quad \text{and} \quad |G| \leq \frac{|R|}{\tilde{m}}, \quad (11.3)$$

which will be useful below. Then we have:

1. $\Delta(0) = |R|$ by Eq. (11.1).
2. $\Delta(q) \leq \sqrt{\frac{8}{9}}|R|$ by Ineq. (11.2).
3. We claim that $|\Delta(t+1) - \Delta(t)| \leq \frac{|R|}{\sqrt{m\tilde{m}}}$. To see this, fix any two $f \in F$ and $g \in G$ and write

$$\begin{aligned} |\Psi_f(t)\rangle &= \sum_{\mathbf{x} \in \{0,1\}^n, b \in \{0,1\}} \alpha_{\mathbf{x},b} |\mathbf{x}\rangle \otimes |b\rangle \otimes |\phi_{\mathbf{x},b}\rangle, \\ |\Psi_g(t)\rangle &= \sum_{\mathbf{x} \in \{0,1\}^n, b \in \{0,1\}} \tilde{\alpha}_{\mathbf{x},b} |\mathbf{x}\rangle \otimes |b\rangle \otimes |\tilde{\phi}_{\mathbf{x},b}\rangle. \end{aligned}$$

Then,

$$\langle \Psi_f(t) | \Psi_g(t) \rangle = \sum_{\mathbf{x} \in \{0,1\}^n} \sum_{b \in \{0,1\}} \bar{\alpha}_{\mathbf{x},b} \tilde{\alpha}_{\mathbf{x},b} \langle \phi_{\mathbf{x},b} | \tilde{\phi}_{\mathbf{x},b} \rangle. \quad (11.4)$$

On the other hand,

$$\begin{aligned} (Q_f \otimes I) |\Psi_f(t)\rangle &= \sum_{\mathbf{x} \in \{0,1\}^n, b \in \{0,1\}} \alpha_{\mathbf{x},b} |\mathbf{x}\rangle \otimes |b \oplus f(\mathbf{x})\rangle \otimes |\phi_{\mathbf{x},b}\rangle, \\ (Q_g \otimes I) |\Psi_g(t)\rangle &= \sum_{\mathbf{x} \in \{0,1\}^n, b \in \{0,1\}} \tilde{\alpha}_{\mathbf{x},b} |\mathbf{x}\rangle \otimes |b \oplus g(\mathbf{x})\rangle \otimes |\tilde{\phi}_{\mathbf{x},b}\rangle, \end{aligned}$$

and hence

$$\begin{aligned} \langle \Psi_f(t+1) | \Psi_g(t+1) \rangle &= \langle V_t(Q_f \otimes I) \Psi_f(t) | V_t(Q_g \otimes I) \Psi_g(t) \rangle \\ &= \langle (Q_f \otimes I) \Psi_f(t) | (Q_g \otimes I) \Psi_g(t) \rangle \\ &= \sum_{\mathbf{x} \in \{0,1\}^n} \sum_{b, \tilde{b} \in \{0,1\}} \bar{\alpha}_{\mathbf{x},b} \tilde{\alpha}_{\mathbf{x},\tilde{b}} \langle b \oplus f(\mathbf{x}) | \tilde{b} \oplus g(\mathbf{x}) \rangle \langle \phi_{\mathbf{x},b} | \tilde{\phi}_{\mathbf{x},\tilde{b}} \rangle \\ &= \sum_{\mathbf{x} \in \{0,1\}^n} \sum_{\substack{b, \tilde{b} \in \{0,1\} \text{ s.t.} \\ b \oplus f(\mathbf{x}) = \tilde{b} \oplus g(\mathbf{x})}} \bar{\alpha}_{\mathbf{x},b} \tilde{\alpha}_{\mathbf{x},\tilde{b}} \langle \phi_{\mathbf{x},b} | \tilde{\phi}_{\mathbf{x},\tilde{b}} \rangle \end{aligned} \quad (11.5)$$

For any fixed $\mathbf{x} \in \{0, 1\}^n$ with $f(\mathbf{x}) = g(\mathbf{x})$, the corresponding summands in Eqs. (11.4) and (11.5) are the same. Thus we have

$$\begin{aligned}
|\langle \Psi_f(t+1) | \Psi_g(t+1) \rangle - \langle \Psi_f(t) | \Psi_g(t) \rangle| &= \left| \sum_{\substack{\mathbf{x} \in \{0,1\}^n \text{ s.th.} \\ f(\mathbf{x}) \neq g(\mathbf{x})}} \bar{\alpha}_{\mathbf{x},0} \tilde{\alpha}_{\mathbf{x},1} \langle \phi_{\mathbf{x},0} | \tilde{\phi}_{\mathbf{x},1} \rangle + \bar{\alpha}_{\mathbf{x},1} \tilde{\alpha}_{\mathbf{x},0} \langle \phi_{\mathbf{x},1} | \tilde{\phi}_{\mathbf{x},0} \rangle \right| \\
&\leq \sum_{\substack{\mathbf{x} \in \{0,1\}^n \text{ s.th.} \\ f(\mathbf{x}) \neq g(\mathbf{x})}} \left(|\alpha_{\mathbf{x},0}| |\tilde{\alpha}_{\mathbf{x},1}| + |\alpha_{\mathbf{x},1}| |\tilde{\alpha}_{\mathbf{x},0}| \right) \\
&\leq \sum_{\substack{\mathbf{x} \in \{0,1\}^n \text{ s.th.} \\ f(\mathbf{x}) \neq g(\mathbf{x})}} \sqrt{P_f(\mathbf{x}) P_g(\mathbf{x})},
\end{aligned}$$

where the last step used the Cauchy-Schwarz inequality and we introduced the following abbreviations:

$$\begin{aligned}
P_f(\mathbf{x}) &= |\alpha_{\mathbf{x},0}|^2 + |\alpha_{\mathbf{x},1}|^2, \\
P_g(\mathbf{x}) &= |\tilde{\alpha}_{\mathbf{x},0}|^2 + |\tilde{\alpha}_{\mathbf{x},1}|^2.
\end{aligned}$$

Finally, we can compute

$$\begin{aligned}
|\Delta(t+1) - \Delta(t)| &\leq \sum_{(f,g) \in R} |\langle \Psi_f(t+1) | \Psi_g(t+1) \rangle - \langle \Psi_f(t) | \Psi_g(t) \rangle| \\
&\leq \sum_{(f,g) \in R} \sum_{\substack{\mathbf{x} \in \{0,1\}^n \text{ s.th.} \\ f(\mathbf{x}) \neq g(\mathbf{x})}} \sqrt{P_f(\mathbf{x}) P_g(\mathbf{x})} \\
&\leq \sqrt{\sum_{(f,g) \in R} \sum_{\substack{\mathbf{x} \in \{0,1\}^n \text{ s.th.} \\ f(\mathbf{x}) \neq g(\mathbf{x})}} P_f(\mathbf{x})} \sqrt{\sum_{(f,g) \in R} \sum_{\substack{\mathbf{x} \in \{0,1\}^n \text{ s.th.} \\ f(\mathbf{x}) \neq g(\mathbf{x})}} P_g(\mathbf{x})} \\
&= \sqrt{\sum_{f \in F} \sum_{\mathbf{x} \in \{0,1\}^n} P_f(\mathbf{x}) \underbrace{\sum_{\substack{g \in G_f \text{ s.th.} \\ f(\mathbf{x}) \neq g(\mathbf{x})}} 1}_{\leq 1}} \sqrt{\sum_{g \in G} \sum_{\mathbf{x} \in \{0,1\}^n} P_g(\mathbf{x}) \underbrace{\sum_{\substack{f \in F_g \text{ s.th.} \\ f(\mathbf{x}) \neq g(\mathbf{x})}} 1}_{\leq 1}} \\
&\leq \sqrt{|F|} \sqrt{|G|} \leq \sqrt{\frac{|R|}{m}} \sqrt{\frac{|R|}{\tilde{m}}} = \frac{|R|}{\sqrt{m\tilde{m}}}
\end{aligned}$$

where the second inequality uses the Cauchy-Schwarz inequality one more time, the underbraced inequalities are ≤ 1 since for every $f \in F$ and every $\mathbf{x} \in \{0, 1\}^n$ there exists at most one $g \in G_f$ such that $f(\mathbf{x}) \neq g(\mathbf{x})$ (namely the function which agrees everywhere with f except at \mathbf{x}), and vice versa, and we used Ineq. (11.3) in the last line. This concludes the proof of the claim.

The three points above show directly that $q = \Omega(\sqrt{m\tilde{m}})$, concluding the proof. \square

The above proof generalizes readily to a more general criterion, which we call the ‘‘basic adversary bound’’. You can practice this one in the exercises:

Theorem 11.6: Basic adversary bound

Let P be a decision problem, $F \subseteq P^{-1}(0)$, $G \subseteq P^{-1}(1)$, and let $R \subseteq F \times G$ be an arbitrary relation. For every $f \in F$, $g \in G$, and $\mathbf{x} \in \{0, 1\}^n$, let

$$\begin{aligned}
G_f &:= \{g \in G : (f, g) \in R\}, & G_{f,\mathbf{x}} &:= \{g \in G_f : f(\mathbf{x}) \neq g(\mathbf{x})\}, \\
F_g &:= \{f \in F : (f, g) \in R\}, & F_{g,\mathbf{x}} &:= \{f \in F_g : f(\mathbf{x}) \neq g(\mathbf{x})\}.
\end{aligned}$$

Then,

$$Q(P) = \Omega\left(\sqrt{\frac{m\tilde{m}}{\ell\tilde{\ell}}}\right),$$

where

$$\begin{aligned} m &:= \min_{f \in F} |G_f|, & \ell &:= \max_{f \in F, \mathbf{x} \in \{0,1\}^n} |G_{f,\mathbf{x}}|, \\ \tilde{m} &:= \min_{g \in G} |F_g|, & \tilde{\ell} &:= \max_{g \in G, \mathbf{x} \in \{0,1\}^n} |F_{g,\mathbf{x}}|. \end{aligned}$$

The relation R determines which pairs of “yes”- and “no”-instances you want to think of as “difficult to distinguish”. The parameters m and \tilde{m} are defined as before. The parameters ℓ and $\tilde{\ell}$ capture to which extent your “yes”- and “no”-instances are indeed difficult to distinguish (in the sense that for every “no” instance, a query to any single location \mathbf{x} only allows you to exclude only few “yes”-instances, and vice versa).

Remark 11.7

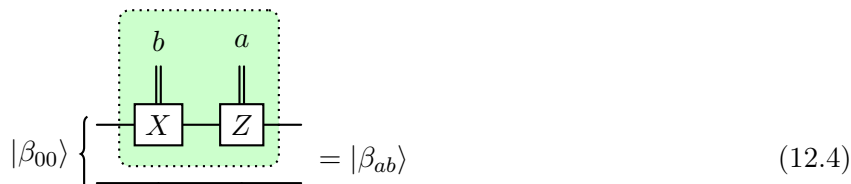
Note that we recover [Theorem 11.4](#) from [Theorem 11.6](#) by choosing the relation such that $(f, g) \in R$ if and only if $d(f, g) = 1$, since in this case $\ell = \tilde{\ell} = 1$ (assuming $m, \tilde{m} \geq 1$). This is because for this choice of relation, if $G_{f,\mathbf{x}}$ is nonempty then it will contain precisely the unique function that differs precisely at location \mathbf{x} from f , and vice versa.

As suggested by the terminology, there is a more general version of the adversary bound. Remarkably, there are versions of the adversary bound that not only give a lower bound on the quantum query complexity, but which capture the quantum query complexity of decision problems exactly (up to constants)!

- It is impossible to distinguish the four states by a measurement on the first (or second) qubit only. That is, the four Bell states are *locally indistinguishable* – if two parties Alice and Bob each have one qubit of a Bell state, neither Alice or Bob can tell the four states apart by a measurement of their qubit. In fact, regardless in which basis they measure their qubit, the outcome is always uniformly random.
- It is nevertheless possible to transform any of the four states into any other by applying a suitable unitary on the first qubit. Indeed, for all $a, b \in \{0, 1\}$,

$$|\beta_{ab}\rangle = (Z^a X^b \otimes I) |\beta_{00}\rangle = (I \otimes X^b Z^a) |\beta_{00}\rangle, \quad (12.3)$$

where X and Z are two of the Pauli matrices. We can visualize this in a circuit as follows:



It follows that if Alice and Bob share a Bell state (i.e., Alice has one qubit and Bob the other), either of them can apply a suitable combination of Pauli X or Z matrices to convert the state they start with into any of the other four Bell states.

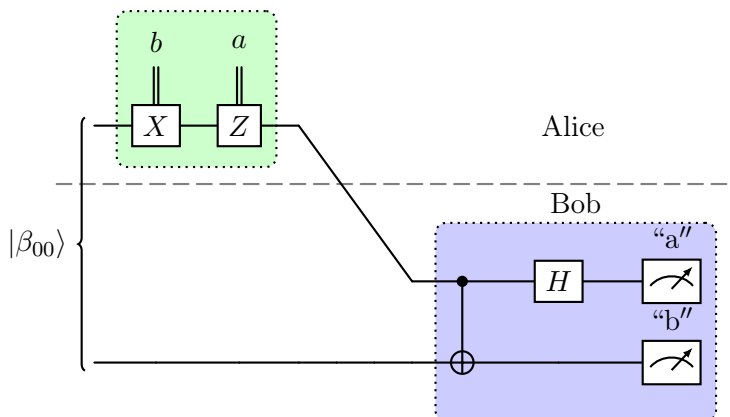
- The Bell states are all entangled. Since we already know that the EPR pair $|\beta_{00}\rangle$ is entangled, this follows from the previous point (since applying a local unitary will never turn an entangled state into a product state, or vice versa).

12.2 Superdense coding

The fact that any two Bell states can be converted into each other by a local operation is perhaps surprising. We can use it for a first application of entanglement.

Suppose that Alice wants to communicate n bits to Bob (without any error), but she is only allowed to send a single qubit. If Alice and Bob do not share any resources, this is possible only for $n = 1$, since the best that Bob can do is to measure the qubit that he receives in some basis, which gives him one bit of information.

Interestingly, if Alice and Bob share an EPR pair state this allows them to do better. Consider the following protocol, which is known as *superdense coding*:



Let us describe the protocol more formally: Alice and Bob start out in the EPR pair state $|\beta_{00}\rangle$, where the first qubit belongs to Alice and the second qubit belongs to Bob. Now suppose Alice wants to communicate two bits $a, b \in \{0, 1\}$. First, if $b = 1$ then she applies a Pauli X operator on her system. Next, if $a = 1$ then she applies a Pauli Z operator on her system. Finally, Alice sends over her qubit to Bob. Now both qubits belong to Bob, so he can perform a Bell measurement as in (12.2) to distinguish the states perfectly.

We claim that Bob’s measurement outcome is precisely Alice’s message. Indeed, by Eqs. (12.3) and (12.4), the joint state of the two qubits when Alice sends over her qubit is precisely the Bell state $|\beta_{ab}\rangle$, as indicated in the picture. Thus the Bell basis measurement can perfectly identify the state.

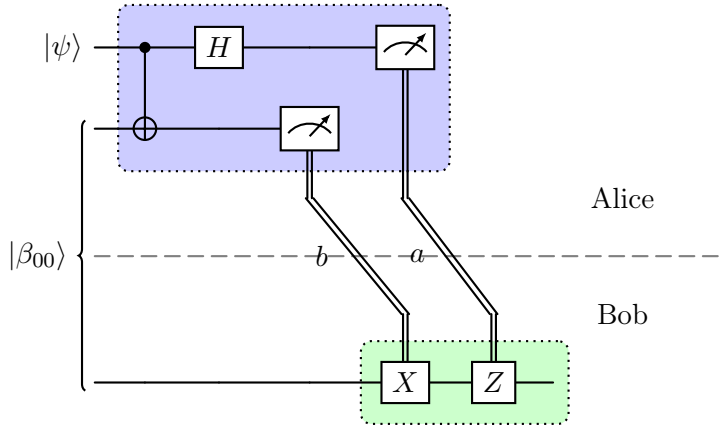
The superdense coding protocol allows us to *communicate two bits by sending a single qubit*. To achieve this, the protocol consumes one maximally entangled qubit state. This gives us some first insight into the power of entanglement as a resource for communication.

12.3 Teleportation

We now discuss another interesting protocol known as *teleportation*. Teleportation is dual to superdense coding since it achieves the opposite task: it lets you *transfer one qubit by sending two bits*.

Wait, does this even make sense? This is clearly impossible, since there are infinitely many quantum states, but only four bitstrings of length two. Even worse, if suppose that there would be a way to communicate a qubit in an arbitrary state by sending over *any* amount of classical information. Then we could clone the qubit state by simply copying the classical message and running Bob’s procedure twice. But you will show on the homework that cloning an arbitrary unknown quantum state is impossible!

In Section 12.2, we saw that a similar “no go” result could be overcome by using entanglement. Thus we can ask if our task here can in fact be achieved if Alice and Bob share some entanglement, say an EPR pair. Surprisingly, this is indeed the case – if Alice and Bob share an EPR pair then they can perfectly transmit an unknown quantum state by sending only classical information. The protocol that achieves this is known as the *teleportation* protocol. We first describe the protocol and then argue its correctness:



As visualized in the picture, Alice and Bob start out in a state $|\psi\rangle \otimes |\beta_{00}\rangle$, where $|\psi\rangle$ is some arbitrary state of Alice’s first qubit (which can be unknown to her). First, Alice performs a Bell measurement (12.2) on both her qubits. The measurement outcome are two bits a and b , which she subsequently sends over to Bob. Finally, if $b = 1$ then Bob applies a Pauli X operator; if $a = 1$ he applies a Pauli Z operator.

We claim that after the protocol has completed, the state of Bob’s qubit is $|\psi\rangle$ – that is, exactly the same state that started out in Alice’s first qubit! To verify this claim, we first compute the post-measurement state on Bob’s qubit for any possible pair of measurement outcomes a, b . To this end, we compute

$$\begin{aligned}
 \langle ab | \otimes I (H \otimes I \otimes I) (CNOT \otimes I) (|\psi\rangle \otimes |\beta_{00}\rangle) &= \langle \beta_{ab} | \otimes I (|\psi\rangle \otimes |\beta_{00}\rangle) \\
 &= \langle \beta_{00} | \otimes I (I \otimes Z^a X^b \otimes I) (|\psi\rangle \otimes |\beta_{00}\rangle) \\
 &= \langle \beta_{00} | \otimes I (|\psi\rangle \otimes |\beta_{ab}\rangle) \\
 &= \langle \beta_{00} | \otimes I (I \otimes I \otimes X^b Z^a) (|\psi\rangle \otimes |\beta_{00}\rangle) \\
 &= X^b Z^a (\langle \beta_{00} | \otimes I) (I \otimes |\beta_{00}\rangle) |\psi\rangle \\
 &= \frac{1}{2} X^b Z^a |\psi\rangle.
 \end{aligned}$$

where we first used Eq. (12.1), next the second, the first, and again the second identity in Eq. (12.3), then we

rearrange, and finally we used the following key identity: ¹

$$(\langle \beta_{00} | \otimes I)(I \otimes |\beta_{00}\rangle) = \frac{1}{2} \sum_{z,w=0}^1 (\langle z | \otimes \langle z | \otimes I)(I \otimes |w\rangle \otimes |w\rangle) = \frac{1}{2} I.$$

To see that the second equality holds, note that for any $x, y \in \{0, 1\}$, we have

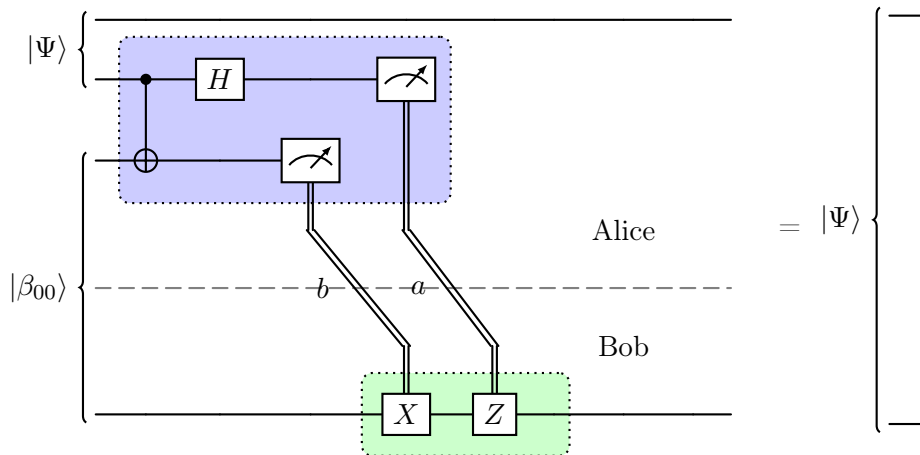
$$\langle x | (\langle \beta_{00} | \otimes I)(I \otimes |\beta_{00}\rangle) |y\rangle = \frac{1}{2} \sum_{z,w=0}^1 (\langle z | \otimes \langle z | \otimes \langle x |)(|y\rangle \otimes |w\rangle \otimes |w\rangle) = \frac{1}{2} \sum_{z,w=0}^1 \langle z z x | y w w \rangle = \frac{1}{2} \langle x | I | y \rangle.$$

Thus we recognize that each pair of outcomes (a, b) occurs with probability $\frac{1}{4}$, with corresponding post-measurement state

$$X^b Z^a |\psi\rangle.$$

Using $X^2 = Z^2 = I$, we see that Bob's operations precisely undo the Pauli operators and hence the final state of Bob's qubit is indeed $|\psi\rangle$. Thus the teleportation protocol behaves correctly.

In fact, the teleportation protocol not only transmits Alice's state to Bob, but it also *preserves any entanglement* or correlations it might have had with some other system. In other words, if we start with $|\Psi\rangle \otimes |\beta_{00}\rangle$, where $|\Psi\rangle$ is a quantum state of Alice's qubit and some arbitrary other quantum system, and we apply the teleportation protocol as above, then the resulting joint state of Bob's qubit and the other system will be $|\Psi\rangle$. This is visualized in the following figure:



You can prove this on the homework.

Quantum teleportation is somewhat analogous to the classical *one-time pad*, a protocol for transmitting a private probabilistic bit from Alice to Bob by using only public communication and a secret shared random bit. You can discuss this in the exercises.

Teleportation and superdense coding can be summarized by the following two “resource inequalities”:

$$\begin{aligned} \text{teleportation:} & \quad \text{EPR pair} + 2[c \rightarrow c] \geq [q \rightarrow q], \\ \text{superdense coding:} & \quad \text{EPR pair} + [q \rightarrow q] \geq 2[c \rightarrow c], \end{aligned}$$

where $[c \rightarrow c]$ denotes one bit of classical communication and $[q \rightarrow q]$ denotes one qubit of quantum communication. You can read the inequality sign as “is at least as good as” or “can be used to implement”. In other words, in the presence of unlimited shared entanglement, quantum communication is no more or less powerful as classical communication (at twice the rate). However, this does not trivialize quantum information theory, since in most cases of interest we do *not* have entanglement available ‘for free’!

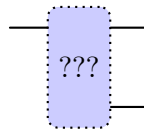
¹If the meaning of an expression such as $I \otimes |\beta_{00}\rangle$ confuses you, have a look at the exercises.

Chapter 13

No cloning and quantum money

13.1 No cloning revisited

Suppose someone hands you a box that they claim can perfectly clone qubit states, i.e., outputs $|\psi\rangle \otimes |\psi\rangle$ when given an arbitrary state $|\psi\rangle \in \mathbb{C}^2$ as input:



How could we test their claim? Here is one way to define such a *cloning test*:

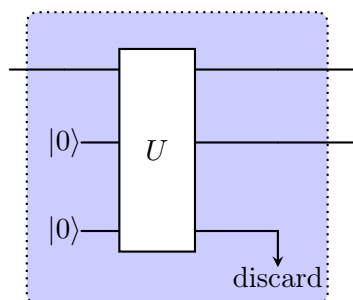
- Initialize a qubit in a state $|\psi\rangle$ chosen randomly from the four states $\{|0\rangle, |1\rangle, |+\rangle, |-\rangle\}$.
- Input it into the box.
- Measure each of the output qubits in the corresponding basis (standard for $|0\rangle$ and $|1\rangle$, Hadamard for $|+\rangle$ and $|-\rangle$).¹
- Accept if the outcomes are as they should be (e.g. if $|\psi\rangle = |1\rangle$ then the result should be “1” for both measurements).

A perfect cloner would pass this test 100% of the time.

Now, we know that it is *impossible* to perfectly clone an unknown state – even when we restrict to the four states above (or even just $|0\rangle$ and $|+\rangle$). You proved this on last week’s homework. Therefore the box that we are handed will never pass the test with 100% probability. But how well can we actually perform at this cloning test (or ‘game’)?

On last week’s homework you discussed various strategies for implementing such a cloner. The best strategy that you considered passed the test with probability $3/4$.² We will now show that this is the best one can achieve.

To this end, let us look at an arbitrary machine (box, protocol, ...) of the following form:



¹Recall that what we called a “Hadamard measurement” is to first apply H to a qubit, then carry out an ordinary measurement and then apply H again. See [Example 2.8](#).

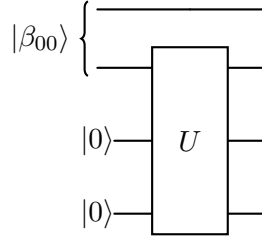
²Naively cloning in the standard basis only passes the test with probability $5/8$, so this is a nontrivial result.

Here, U is a unitary on $\mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^d$ for some arbitrary d . Even though this might not a priori be clear, this is the most general ansatz possible, as we discussed in class.

We would like to bound the probability of passing the cloning test for an arbitrary machine of the above form. The key idea is to consider the following state:

$$|\Omega\rangle = (I \otimes U)(|\beta_{00}\rangle \otimes |0\rangle \otimes |0\rangle).$$

Note that this is a quantum state of *four* systems. It corresponds to inputting half of an EPR pair into the machine and stopping before the last qubit is discarded:



As discussed in class, the relevance of the state $|\Omega\rangle$ is such that one can ‘extract’ from it the action of the box on arbitrary input states $|\psi\rangle$. Indeed, for a standard basis state $|x\rangle$, we have

$$\begin{aligned} \langle x| \otimes I \otimes I \otimes I |\Omega\rangle &= (\langle x| \otimes U)(|\beta_{00}\rangle \otimes |0\rangle \otimes |0\rangle) \\ &= (\langle x| \otimes U)\left(\frac{1}{\sqrt{2}} \sum_{y=0}^1 |y\rangle \otimes |y\rangle \otimes |0\rangle \otimes |0\rangle\right) \\ &= \frac{1}{\sqrt{2}} \sum_{y=0}^1 \langle x|y\rangle U(|y\rangle \otimes |0\rangle \otimes |0\rangle) \\ &= \frac{1}{\sqrt{2}} U(|x\rangle \otimes |0\rangle \otimes |0\rangle) \end{aligned}$$

and now the case of general $|\psi\rangle \in \mathbb{C}^2$ follows by linearity:³

$$(\langle \bar{\psi}| \otimes I \otimes I \otimes I) |\Omega\rangle = \frac{1}{\sqrt{2}} U(|\psi\rangle \otimes |0\rangle \otimes |0\rangle) \tag{13.1}$$

This is a very general trick that allows translating quantum operations into quantum states.

We can use the preceding to write down a simple and clean formula for the probability of passing the test. Now, the probability that we obtain outcome “ ψ ” for both qubits when we input $|\psi\rangle$ into the box and measure each of the two qubits in an orthonormal basis extending $|\psi\rangle$ is given by

$$\begin{aligned} \|\langle \psi| \otimes \langle \psi| \otimes I) U(|\psi\rangle \otimes |0\rangle \otimes |0\rangle)\|^2 &= \|\langle \bar{\psi}| \otimes I \otimes I \otimes I) |\Omega\rangle\|^2 \\ &= 2 \|\langle \bar{\psi}| \otimes \langle \psi| \otimes \langle \psi| \otimes I) |\Omega\rangle\|^2 \\ &= 2 \langle \Omega| (|\bar{\psi}\rangle \langle \bar{\psi}| \otimes |\psi\rangle \langle \psi| \otimes |\psi\rangle \langle \psi| \otimes I) |\Omega\rangle \\ &= 2 \langle \Omega| (|\bar{\psi}, \psi, \psi\rangle \langle \bar{\psi}, \psi, \psi| \otimes I) |\Omega\rangle, \end{aligned}$$

where we used [Eq. \(13.1\)](#) in the first step. Therefore, the probability of passing the cloning test is given by

$$\begin{aligned} \mathbf{Pr}(\text{pass}) &= \frac{1}{4} \left(2 \langle \Omega| (|000\rangle \langle 000| \otimes I) |\Omega\rangle + 2 \langle \Omega| (|111\rangle \langle 111| \otimes I) |\Omega\rangle \right. \\ &\quad \left. + 2 \langle \Omega| (|+++ \rangle \langle +++| \otimes I) |\Omega\rangle + 2 \langle \Omega| (|--- \rangle \langle ---| \otimes I) |\Omega\rangle \right) \\ &= \langle \Omega| Q \otimes I |\Omega\rangle, \end{aligned}$$

where we have introduced the operator

$$Q = \frac{1}{2} (|000\rangle \langle 000| + |111\rangle \langle 111| + |+++ \rangle \langle +++| + |--- \rangle \langle ---|).$$

³Note that $\langle \bar{\psi}| = \psi_0 \langle 0| + \psi_1 \langle 1|$ is linear in $|\psi\rangle$, unlike $\langle \psi| = \bar{\psi}_0 \langle 0| + \bar{\psi}_1 \langle 1|$. Concretely, $\langle \bar{\psi}|$ is simply the transpose of $|\psi\rangle$.

How can we bound such an expression? Clearly,

$$\Pr(\text{pass}) = \langle \Omega | Q \otimes I | \Omega \rangle \leq \|Q \otimes I\| = \lambda_{\max}(Q \otimes I) = \lambda_{\max}(Q),$$

where we used the fact that for positive semidefinite operators, the operator norm is the same as the largest eigenvalue.⁴ But it is easy to verify that the largest eigenvalue of Q is $\frac{3}{4}$. Therefore,

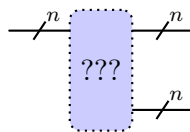
$$\Pr(\text{pass}) \leq \frac{3}{4},$$

which concludes the proof. Thus, we have shown that the maximal probability with which one can pass the cloning test is $\frac{3}{4}$.

It is easy to generalize the preceding result to n -qubit states of the form

$$|\Psi\rangle = |\psi_1\rangle \otimes \dots \otimes |\psi_n\rangle,$$

where each $|\psi_j\rangle \in \{|0\rangle, |1\rangle, |+\rangle, |-\rangle\}$. If we have a machine that claims to clone n -qubit states of this form,



and we subject it to a similar test as before (input a random n -qubit state of the above form and for each $j = 1, \dots, n$ measure the j -th qubit in the first and second output of the machine in the appropriate basis and check the result), then the probability of passing this test is

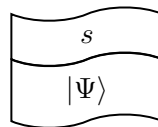
$$\Pr(\text{pass}) \leq \left(\frac{3}{4}\right)^n,$$

i.e., exponentially small in n . Can you see how to prove this precisely, generalizing the above argument?

13.2 Quantum money

Next we discuss an amusing application of the preceding ideas which is due to Wiesner (all the way back in the 1970s). His idea was the ‘no cloning’ principle should allow one to create and verify bills (banknotes) that are impossible to forge. Nowadays this and related ideas are referred to as *quantum money*.

Without further ado let us present Wiesner’s idea. In his scheme, bills consist of a (classical) serial number s and an n -qubit quantum state $|\Psi\rangle$, where n will play the role of a security parameter. Here is a visualization:

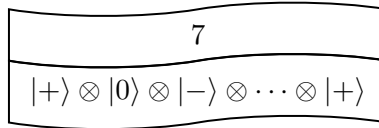


Protocol for issuing bills: Bills are issued by a central bank (just like in the real world) according to the following protocol: The bank chooses a serial number s according to some scheme, as well as a random n -qubit quantum state of the form

$$|\Psi\rangle = |\psi_1\rangle \otimes \dots \otimes |\psi_n\rangle,$$

where each ψ_j is chosen independently and uniformly at random from $\{|0\rangle, |1\rangle, |+\rangle, |-\rangle\}$. The bank issues the bill, i.e., the pair $(s, |\Psi\rangle)$, and it also stores the serial number s along with a description “ Ψ ” of the quantum state (which we can model, say, by a string in $\{0, 1, +, -\}^n$) in its records, stored away privately in the bank vault. For example, if the bank issues the bill

⁴Don’t remember the operator norm? See Practice Problem Set 8 and Homework Problem Set 7.



then it would also keep the following record:

| serial number | description of the state |
|---------------|--------------------------|
| 7 | +0-...+ |

Protocol for verifying bills: ⁵ Suppose someone hands you what they claim is a bill, but you are sceptical that this is valid. Then you could simply take the bill $(s, |\Psi\rangle)$ to the central bank, who will do the following: Using the serial number s printed on the bill, the bank will look up the description of what the quantum state should be according to its records, measure the qubits that make up $|\Psi\rangle$ in the corresponding basis, and check that each outcome is as expected.

In the example above, when handed a bill with serial number $s = 7$, the bank would measure the first qubit of $|\Psi\rangle$ in the Hadamard basis and check that the result is “+”, then the second qubit in the standard basis and check that the result is “0”, etc.

13.3 Security of Wiesner’s quantum money

The question we are interested of course is whether this scheme lives up to its promise. Are these bills indeed unforgeable or can they be counterfeited?

13.3.1 Security against simple counterfeiting attacks

For example, suppose that you have a bill $(s, |\Psi\rangle)$ in your hands – can you “clone” the bill and make two out of it? More precisely, we may ask the following question: *Is it possible for you to take your n -qubit state $|\Psi\rangle$ and turn it into two n -qubit states that are both accepted by the bank?* This is known as a *simple counterfeiting attack*. But if you think about it, the probability that the bank accepts both bills is exactly the same as the probability of passing the cloning test! Therefore, we deduce from the discussion at the end of [Section 13.1](#) that any simple counterfeiting attack will succeed with probability at most $(3/4)^n$. By making n large, we can quickly make this probability as small as we like. This shows that Wiesner’s money is secure against simple counterfeiting attacks, with n playing the role of a security parameter.

Note that in establishing this result we did *not* use any computational assumptions. In other words, the security against simple counterfeiting attacks that we just established is an information-theoretic or “unconditional” one!

13.3.2 Insecurity against more general attacks

The next question that we can ask, is whether the above attack is the most general model that we should consider. This is not so. For example, the above attacker model does not account for an attacker that interacts with the bank *multiple times*. In this case, it is not hard to craft an attack, provided that we have enough queries and that the bank returns the residual state after the query.

An Attack with a Verification Oracle. Let us model the bank as an *oracle* that takes as input a money bill (i.e., a state $|\Psi\rangle = |\psi_1\rangle \otimes \dots \otimes |\psi_n\rangle$ and a serial number s), runs the check as described above, and returns $\{0, 1\}$ (meaning reject/accept) along with the post-measurement state. Given access to such an oracle, the attack proceeds as follows:

- For all $j = 1, \dots, n$:

⁵Since the verification procedure we are about to describe involves the authority that issued the bills, perhaps we should rather think of the bills as unforgeable tokens that are issued and verified at the place of use (e.g., food tokens that you can buy and use at the cafeteria, or tokens for a carousel ride that you can buy and use at a fun fair, ...).

- For all $\phi \in \{0, 1, +, -\}$:
 - * Repeat the following m times:
 - Replace the j -th qubit with a fresh qubit initialized in state $|\phi\rangle$.
 - Query the oracle (i.e., the bank) with the resulting n -qubit state.
 - * If the oracle accepts each time then deduce that $|\psi_j\rangle$ must have been $|\phi\rangle$.
 - * Otherwise, proceed with the next iteration.

To see why the attack is successful, observe that the qubits of the state $|\Psi\rangle$ are tensored (thus not entangled), and therefore in the j -th iteration of the outer loop, the measurements carried out by the oracle will never impact any other qubit than the j -th one. We now want to argue that, once they exit the inner loop, the attacker has correctly guessed the state of the j -th qubit of the quantum money bill. We consider two cases:

- $|\phi\rangle = |\psi_j\rangle$: In this case, the state queried by the attacker is identical to $|\Psi\rangle$, the verification always passes. At this point, the attacker has correctly deduced that the j -th qubit must have been in state $|\phi\rangle = |\psi_j\rangle$.
- $|\phi\rangle \neq |\psi_j\rangle$: In this case, the verification oracle will accept with probability at most $\frac{1}{2}$. Therefore, if we repeat the procedure m times, it will accept each time with probability at most $\frac{1}{2^m}$. In other words, the verification oracle will decline at least once with probability at least $1 - \frac{1}{2^m}$, in which case the attacker has correctly determined that $|\phi\rangle \neq |\psi_j\rangle$ and continues with the next iteration.

Thus, after the above procedure the attacker has learned a full classical description of the state and can make as many copies as he wants – at a cost of at most $4n^2$ queries to the verification oracle and with an exponentially small failure probability.

On the homework you will discuss another attack using EPR pairs.

13.4 Bonus: Public verifiability and public-key quantum money

The above attack is not necessarily practical, as it is likely that the bank will stop answering verification queries at some point (think of what happens if you mistype your password). However it is useful to illustrate the pitfalls of this scheme. In particular, we cannot hope to use the same quantum money scheme to achieve *public verification*: This property allows anyone to verify the validity of a bill *without contacting the bank*, and it is of course highly desirable. Public verification allows users to transact without interacting with the bank for each transaction.

Assume that we want to construct such a scheme, then by definition the verification algorithm is public, thus it better be the case that the scheme is secure in the presence of a verification oracle!⁶ Looking a little closer to our attack, we can see that it teaches us two lessons if we want to build a scheme with public verification:

- Security cannot be information theoretic: Given enough queries, an attacker can always obtain a serial number.
- The state corresponding to a bill cannot consist of a tensor product of single qubits, since this would enable qubit-by-qubit guessing attacks, such as the one described above.

Public-Key Quantum Money. As discussed before, it would be more desirable if you could verify bills yourselves without involving any central authority (bank) at all. This is actually a topic of ongoing research, and schemes get routinely proposed (and in some cases, broken shortly after). Here we describe a particular family of schemes, that are currently the most “stable” in terms of security. In such schemes, bills are of the form:⁷

$$|\Psi\rangle = \frac{1}{\sqrt{2^{n/2}}} \sum_{x \in A} |x\rangle$$

⁶Note that this condition is necessary but not sufficient, since in the actual scheme we don’t even get to see the queries of the attacker to the verification oracle. However, for the sake of this exposition, we will think of the adversary with access to a verification oracle as the “right” definition of security.

⁷For convenience, we ignore the serial number here and we just concentrate on the quantum part of the bill.

where $A \subseteq \mathbb{F}_2^n$ is a randomly sampled linear subspace of dimension $n/2$ and s is a serial number randomly sampled. To verify the validity of a bill, one proceeds in two steps: (i) First test the membership of $|\Psi\rangle$ in A , then (ii) apply the QFT (this is actually the QFT over $(\mathbb{Z}/2\mathbb{Z})^n$, which can be implemented as $H^{\otimes n}$) and test membership in the dual subspace A^\perp .⁸ Given the description of A and A^\perp , both tests can be implemented efficiently by a unitary, since one can efficiently verify whether an x is linearly independent of A (A^\perp , respectively). Furthermore, note that verification is always successful on well-formed states since

$$\frac{1}{\sqrt{2^{n/2}}} \sum_{x \in A} |x\rangle \xrightarrow{\text{QFT}} \frac{1}{\sqrt{2^{n/2}}} \sum_{x \in A^\perp} |x\rangle.$$

For security, it can actually be shown that these states cannot be copied *even if the attacker has oracle access to a membership tester for A and A^\perp* ! In other words, this scheme does not suffer from the same weakness as Wiesner's original proposal, and no attack similar to the one described above is possible.

However, we are not done yet. If we want to achieve public verifiability, we need to provide an algorithm to run the above verification procedure, that can be run by all parties (even the attacker). In other words, we need to somehow implement the above mentioned membership oracles. Clearly, we cannot just publish (A, A^\perp) in the clear, since then all security would be immediately lost. Here the idea is to publish instead an *obfuscated version* of the program that computes membership in A (A^\perp , respectively). This way, anyone can perform the verification of the state, but there is a sense in which A and A^\perp remain hidden. There are several proposals from the literature on how to realize this obfuscation procedure (from various computational assumptions).

⁸Recall that the dual space of A consists of the vectors orthogonal to all $x \in A$. Furthermore, $\dim(A) + \dim(A^\perp) = n$.

Chapter 14

Nonlocal games and the CHSH Game

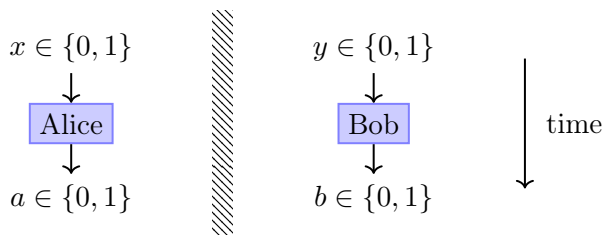
Over the past lectures we discussed some of the strange features of quantum mechanics. In particular, we explored superpositions (such as $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$) and entanglement ($|\Psi\rangle_{AB} \neq |\psi\rangle_A \otimes |\phi\rangle_B$), and we discussed how these features impose both challenges (e.g., non-orthogonal states cannot be distinguished perfectly, no cloning) and opportunities (e.g., entanglement gives rise to superdense coding).

Today, we will discuss another way of quantifying the distinction between classical and quantum mechanics, namely through the *correlations* predicted by these theories. A modern perspective of studying and comparing correlations is through the notions of a *nonlocal game*. This is closely related to the notion of a Bell inequality.

In a *nonlocal game*, we imagine that a number of *players* play against a *referee*. The referee hands them *questions* and the players reply with appropriate *answers* that win them the game. The players' goal is to collaborate and maximize their chances of winning. Before the game, the players meet and may agree upon a joint strategy – but then they move far apart from each other and cannot communicate with each other while the game is being played (this can be ensured by the laws of special relativity). The point then is the following: *Since the players are constrained by the laws of physics, we can design games where players utilizing a quantum strategy may have an advantage.* This way of reasoning about quantum correlations is eminently operational and quantitative, as we will see in the following.

14.1 The CHSH game

The *CHSH game* is a famous example of a nonlocal game proposed in 1969 by Clauser, Horne, Shimony and Holt. (Clauser, along with Aspect and Zeilinger, received the Nobel Prize in Physics 2022 “for experiments with entangled photons, establishing the violation of Bell inequalities and pioneering quantum information science.” In a moment we’ll learn what a Bell inequality is.) The following figure illustrates the setup:



(Perhaps this figure should be rotated, since in the circuits we drew so far, time usually goes from left to right!)

As is clear from the figure, it involves two players, Alice and Bob. Each receives as question a bit $x, y \in \{0, 1\}$ and their answers are likewise bits $a, b \in \{0, 1\}$. They win the game according to the following table:

| x | y | winning condition |
|-----|-----|-------------------|
| 0 | 0 | $a = b$ |
| 0 | 1 | $a = b$ |
| 1 | 0 | $a = b$ |
| 1 | 1 | $a \neq b$ |

The winning condition can be succinctly stated as follows:

$$xy = a \oplus b. \tag{14.1}$$

14.2 Classical strategies

It is easy to see that the CHSH game cannot be won if the players' strategies are described by a "local" and "realistic" theory. Here, "local" means that each player's answer does only depend on its immediate surroundings, and "realistic" means that the strategy assigns a definite answer to any possible question – before that question is being asked. Thus in a local and realistic theory we assume that

$$a = f(x) \quad \text{and} \quad b = g(y)$$

When we say that the players may jointly agree on a strategy before the game is being played, we mean that they may jointly select "answer functions" f and g . They can even do so in a correlated way, by using shared randomness. For example, when the players meet before the game is being played, they could flip a coin, resulting in some random $\lambda \in \{0, 1\}$, and agree on the functions $f_\lambda(x) = x \oplus \lambda$ and $g_\lambda(y) = y \oplus \lambda$. More generally, when Alice and Bob meet before the game is being played, they choose λ at random according to some probability distribution $p(\lambda)$ and then pick answer functions f_λ and g_λ depending on λ .¹

If the players' strategy can be described by the rules of our 'classical world', then the above would provide an adequate model. Thus, strategies of this form are usually referred to as *classical strategies* (or as *local hidden variable strategies*, with λ called a *hidden variable*).

Suppose now for sake of finding a contradiction that Alice and Bob can win the CHSH game perfectly using such a classical strategy. For simplicity, we assume that the strategy is deterministic (in the exercises you will show that randomness does not help). Then,

$$\begin{aligned} 1 &= 0 \oplus 0 \oplus 0 \oplus 1 \\ &= (f(0) \oplus g(0)) \oplus (f(0) \oplus g(1)) \oplus (f(1) \oplus g(0)) \oplus (f(1) \oplus g(1)) \\ &= 0. \end{aligned}$$

The first equality is plainly true, the second holds since we assumed that the players always give a correct answer, and the last equality holds all terms appear exactly twice, but $z \oplus z \equiv 0$ for any $z \in \{0, 1\}$. This is a contradiction!

We conclude that there is no perfect classical winning variable strategy for the CHSH game. Suppose, e.g., that the referee selects each possible question (x, y) with equal probability $1/4$. Then the game can be won with probability at most

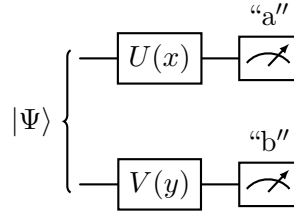
$$p_{\text{win}} \leq 3/4, \tag{14.2}$$

since the players must get at least one of the four possible answers wrong. This winning probability can be achieved by, e.g., the trivial strategy $f(x) = g(y) \equiv 1$. One can call [Ineq. \(14.2\)](#) a "Bell inequality". Indeed, in 1964 Bell was the first to show that classical strategies satisfy certain inequalities on correlations, which can be overcome by using EPR pairs. CHSH simplified his argument and gave an inequality equivalent to [Ineq. \(14.2\)](#).

14.3 Quantum strategies

In a *quantum strategy*, we imagine that the two players are described by quantum mechanics. Thus they start out by sharing an arbitrary joint state $|\Psi\rangle \in \mathbb{C}^{d_A} \otimes \mathbb{C}^{d_B}$ of two quantum systems, where the first quantum system is in Alice's possession and the second in Bob's possession, and upon receiving their questions $x, y \in \{0, 1\}$ they each perform some quantum circuit on their respective quantum system, depending on x and y , respectively, to finally obtain some outcomes $a, b \in \{0, 1\}$. We can model this by letting Alice and Bob apply certain unitaries $U(x)$ and $V(y)$, respectively, and then having each measure in the standard basis, as in the following figure:

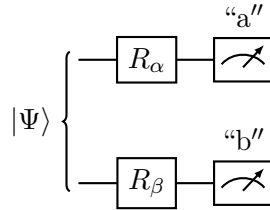
¹Equivalently, we can consider answer functions f and g that are correlated random variables.



While it might not be immediately obvious, any classical strategy is also a quantum strategy. You can show this as an exercise.

14.3.1 Beating the classical bound

On Homework Problem 2.1 you discussed the following situation:



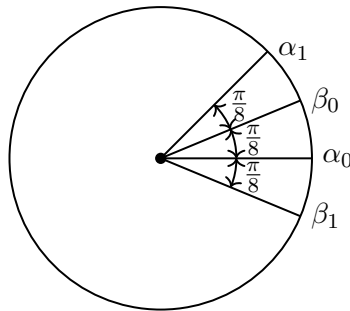
where $|\Psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ is an EPR pair and R_θ denotes the rotation matrix $R_\theta = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$. You found the following:

$$\Pr(a = b) = \cos^2(\alpha - \beta), \quad \Pr(a \neq b) = \sin^2(\alpha - \beta).$$

Now suppose Alice chooses an angle α_x depending on the question $x \in \{0, 1\}$ that she receives, and Bob chooses an angle β_y depending on the question $y \in \{0, 1\}$ that he receives. Then the winning probability is

$$p_{\text{win}} = \frac{1}{4}(\cos^2(\alpha_0 - \beta_0) + \cos^2(\alpha_0 - \beta_1) + \cos^2(\alpha_1 - \beta_0) + \sin^2(\alpha_1 - \beta_1))$$

If we choose the angles $\alpha_0 = 0$, $\alpha_1 = \frac{\pi}{4}$, $\beta_0 = \frac{\pi}{8}$, $\beta_1 = -\frac{\pi}{8}$, as in the following picture



then we obtain

$$|\alpha_0 - \beta_0| = |\alpha_0 - \beta_1| = |\alpha_1 - \beta_0| = \frac{\pi}{8}, \quad |\alpha_1 - \beta_1| = \frac{\pi}{4} + \frac{\pi}{8} = \frac{\pi}{2} - \frac{\pi}{8}.$$

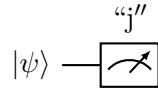
Thus:

$$p_{\text{win}} = \cos^2\left(\frac{\pi}{8}\right) = \left(\frac{1}{2}\sqrt{2 + \sqrt{2}}\right)^2 = \frac{1}{4}(2 + \sqrt{2}) = \frac{1}{2} + \frac{1}{2\sqrt{2}} \approx 85\%. \quad (14.3)$$

This is not only a theoretical prediction but it has actually been demonstrated in practice, first by Aspect et al in the 80s, and pretty recently in 2014 by Hansen et al in a very convincing fashion (avoiding many ‘loopholes’ that were neglected at the time by Aspect et al).

14.3.2 Digression: Measurements vs observables

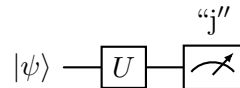
We would like to show that the quantum strategy discussed in [Section 14.3.1](#) is optimal, but this requires some preparation. Let us recap the notion of a quantum measurement. For a single qubit in state $|\psi\rangle$, we know that if we measure (in the standard basis), as in the following circuit,



then we obtain outcome $j \in \{0, 1\}$ with probability

$$\Pr(\text{outcome "j"}) = |\langle j|\psi\rangle|^2. \quad (14.4)$$

More generally, suppose we first apply a unitary and then measure, as in the following:

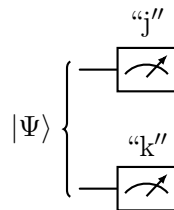


Then $|u_j\rangle := U^\dagger |j\rangle$ for $j \in \{0, 1\}$ is an orthonormal basis and the probability of outcome $j \in \{0, 1\}$ is

$$\Pr(\text{outcome "j"}) = |\langle j|U|\psi\rangle|^2 = |\langle u_j|\psi\rangle|^2, \quad (14.5)$$

as you showed on Homework Problem 1.3. Note that [Eq. \(14.5\)](#) looks like [Eq. \(14.4\)](#), except that the standard basis $\{|j\rangle\}$ is replaced by the basis $\{|u_j\rangle\}$. Accordingly, the above is called *measurement in the basis $\{|u_j\rangle\}$* . For example, when $U = U^\dagger = H$, we already know this under the name *Hadamard measurement*. We can analogously measure a qudit (\mathbb{C}^d) in an arbitrary orthonormal basis.

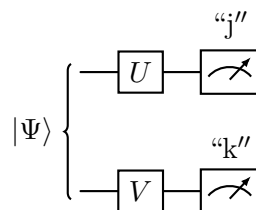
When we have two qubits or qudits and we measure both,



then we know that the joint probability of outcomes is

$$\Pr(\text{outcomes "j" and "k"}) = |\langle j, k|\Psi\rangle|^2.$$

As in the above we can also apply unitaries before measuring, as in the following:



Then, setting $|u_j\rangle := U^\dagger |j\rangle$ and $|v_k\rangle := V^\dagger |k\rangle$, the joint probability of outcomes is

$$\Pr(\text{outcomes "j" and "k"}) = |\langle u_j \otimes v_k|\Psi\rangle|^2. \quad (14.6)$$

We say that we measure the qudit in the basis $\{|u_j\rangle\}$ and the second in the basis $\{|v_k\rangle\}$.

Often we would like to associate measurement outcomes with some arbitrary real numbers. That is, when measuring a qudit in a basis $\{|u_j\rangle\}$, we would like to label the j -th outcome by some arbitrary real number a_j (in particular, we might allow the same number to appear multiple times). Mathematically this can be described as follows: Given a state $|\psi\rangle$, we first sample j from the probability distribution $p_j = |\langle u_j|\psi\rangle|^2$ and

then we apply the function $j \mapsto a_j$, to obtain a real-valued random variable, which we call the (*measurement outcome*). Then the expectation value of this random variable is by the [law of the unconscious statistician](#):

$$\mathbf{E}(\text{outcome}) = \sum_j p_j a_j = \sum_j |\langle u_j | \psi \rangle|^2 a_j = \sum_j \langle \psi | u_j \rangle \langle u_j | \psi \rangle a_j = \langle \psi | \sum_j a_j | u_j \rangle \langle u_j | \psi \rangle = \langle \psi | A | \psi \rangle,$$

where we introduced the operator

$$A = \sum_j a_j |u_j\rangle\langle u_j|.$$

We call A the *observable* associated with the given basis $\{|u_j\rangle\}$ and outcomes $\{a_j\}$. Mathematically, observables are simply Hermitian operators with eigenbasis $\{|u_j\rangle\}$ and corresponding eigenvalues $\{a_j\}$. Any Hermitian operator can be interpreted as an observable as above (in general the eigenbasis is not defined uniquely in case some eigenvalues a_j are repeated, but this poses no problem if all we want is to compute probabilities, as in the above). The fact that the expectation value can be computed by the simple formula

$$\mathbf{E}(\text{outcome}) = \langle \psi | A | \psi \rangle$$

is the main appeal of using observables.

For example, the Pauli Z matrix

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = |0\rangle\langle 0| - |1\rangle\langle 1| = (+1)|0\rangle\langle 0| + (-1)|1\rangle\langle 1|,$$

corresponds to measuring in the standard basis and labeling outcome 0 by +1 and 1 by -1, that is, $a_j = (-1)^j$, while the Pauli X matrix

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = |+\rangle\langle +| - |-\rangle\langle -| = (+1)|+\rangle\langle +| + (-1)|-\rangle\langle -|,$$

corresponds to performing a Hadamard basis measurement and labeling the outcomes again by ± 1 .

How about if we have a two-qudit system and we measure observable $A = \sum_j a_j |u_j\rangle\langle u_j|$ on the first qudit and observable $B = \sum_k b_k |v_k\rangle\langle v_k|$ on the second qudit? Then, using [Eq. \(14.6\)](#),

$$\mathbf{E}(\text{product of the two outcomes}) = \sum_{j,k} a_j b_k |\langle u_j \otimes v_k | \Psi \rangle|^2 = \sum_{j,k} a_j b_k \langle \Psi | u_j \otimes v_k \rangle \langle u_j \otimes v_k | \Psi \rangle = \langle \Psi | A \otimes B | \Psi \rangle.$$

Thus, if we measure the first qubit using A and the second qubit using B , then the expectation value of the product of the outcomes is precisely computed by tensor product $A \otimes B$ of the two observables. Note that $A \otimes B$ is an observable on the two-qudit system, with eigenvectors $|u_j\rangle \otimes |v_k\rangle$ and corresponding eigenvalues $a_j b_k$. The formula

$$\mathbf{E}(\text{product of the two outcomes}) = \langle \Psi | A \otimes B | \Psi \rangle$$

makes working with observables very convenient.

We just discussed how the tensor product observables gives a pleasant way of accessing the product of outcomes. However, in the winning condition (14.1) of the CHSH game we do not care about the product of the outcomes, but rather of their XOR, i.e., $a \oplus b$. Fortunately it is easy to convert one to the other, since $(-1)^a (-1)^b = (-1)^{a \oplus b}$. Thus, we shall model a general quantum strategy for the CHSH game by a state $|\Psi\rangle$ along with observables $A(x)$ and $B(y)$ with eigenvalues $\{\pm 1\}$, one for each $x, y \in \{0, 1\}$. As an exercise, write down the observables corresponding to the quantum strategy above. You should find

$$A(0) = Z, \quad A(1) = X, \quad B(1) = H, \quad B(0) = ZHZ = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ -1 & -1 \end{pmatrix}. \quad (14.7)$$

Note also that

$$\frac{1}{\sqrt{2}}(B(0) + B(1)) = Z \quad \text{and} \quad \frac{1}{\sqrt{2}}(B(1) - B(0)) = X, \quad (14.8)$$

but the relevance of this will become clear only later. Since

$$\mathbf{E}(\text{product of two outcomes}) = \mathbf{Pr}(\text{same}) - \mathbf{Pr}(\text{different})$$

and hence

$$\begin{aligned}\mathbf{Pr}(\text{same}) &= \frac{1}{2}(1 + \mathbf{E}(\text{product of two outcomes})), \\ \mathbf{Pr}(\text{different}) &= \frac{1}{2}(1 - \mathbf{E}(\text{product of two outcomes})),\end{aligned}$$

the probability of winning can be computed as

$$\begin{aligned}p_{\text{win}} &= \frac{1}{8}(1 + \langle \Psi | A(0) \otimes B(0) | \Psi \rangle + 1 + \langle \Psi | A(0) \otimes B(1) | \Psi \rangle + 1 + \langle \Psi | A(1) \otimes B(0) | \Psi \rangle + 1 - \langle \Psi | A(1) \otimes B(1) | \Psi \rangle) \\ &= \frac{1}{2} + \frac{1}{8} \langle \Psi | A(0) \otimes B(0) + A(0) \otimes B(1) + A(1) \otimes B(0) - A(1) \otimes B(1) | \Psi \rangle.\end{aligned}\quad (14.9)$$

14.3.3 Bonus: Tsirelson's bound for the CHSH game

We can write Eq. (14.9) as

$$p_{\text{win}} = \frac{1}{2} + \frac{1}{8} \langle \Psi | H | \Psi \rangle,$$

if we introduce the Hermitian operator

$$\begin{aligned}H &= A(0) \otimes B(0) + A(0) \otimes B(1) + A(1) \otimes B(0) - A(1) \otimes B(1) \\ &= A(0) \otimes (B(0) + B(1)) + A(1) \otimes (B(0) - B(1)).\end{aligned}\quad (14.10)$$

This formula is very convenient for finding the maximum winning probability. Indeed, note that if we fix the strategy then the state $|\Psi\rangle$ that maximizes the winning probability is simply the eigenvector for the largest eigenvalue of H . But how large can this largest eigenvalue be? Tsirelson solved this problem in 1980, here we give a variant of his argument. Consider the vectors

$$|\alpha\rangle = \begin{bmatrix} (A(0) \otimes I) |\Psi\rangle \\ (A(1) \otimes I) |\Psi\rangle \end{bmatrix} \quad \text{and} \quad |\beta\rangle = \begin{bmatrix} (I \otimes (B(0) + B(1))) |\Psi\rangle \\ (I \otimes (B(0) - B(1))) |\Psi\rangle \end{bmatrix}$$

which are both in $\mathbb{C}^{d_A} \otimes \mathbb{C}^{d_B} \oplus \mathbb{C}^{d_A} \otimes \mathbb{C}^{d_B} \cong \mathbb{C}^{2d_A d_B}$. Clearly, we have

$$\langle \alpha | \beta \rangle = \langle \Psi | H | \Psi \rangle,$$

but also

$$\begin{aligned}\|\alpha\|^2 &= \langle \alpha | \alpha \rangle = \langle \Psi | (A(0) \otimes I)^2 + (A(1) \otimes I)^2 | \Psi \rangle = 2 \\ \|\beta\|^2 &= \langle \beta | \beta \rangle = \langle \Psi | (I \otimes (B(0) + B(1)))^2 + (I \otimes (B(0) - B(1)))^2 | \Psi \rangle = 4,\end{aligned}$$

using that $A(0)^2 = A(1)^2 = B(0)^2 = B(1)^2 = I$. That is, $\|\alpha\| = \sqrt{2}$ and $\|\beta\| = 2$. Therefore, it follows by the Cauchy-Schwarz inequality that

$$p_{\text{win}} = \frac{1}{2} + \frac{1}{8} \langle \Psi | H | \Psi \rangle = \frac{1}{2} + \frac{1}{8} \langle \alpha | \beta \rangle \leq \frac{1}{2} + \frac{1}{8} \|\alpha\| \|\beta\| = \frac{1}{2} + \frac{\sqrt{8}}{8} = \frac{1}{2} + \frac{1}{2\sqrt{2}}.\quad (14.11)$$

This shows that any quantum strategy must satisfy the so-called *Tsirelson bound*:

$$p_{\text{win}} \leq \frac{1}{2} + \frac{1}{2\sqrt{2}}.$$

This is exactly achieved by the strategy in Section 14.3.1, as we observe from Eq. (14.3). Thus, the strategy described earlier is optimal!

14.3.4 Outlook: Device-independent quantum cryptography

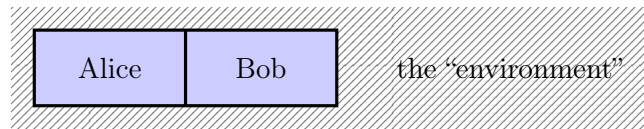
When the two players perform the optimal strategy described above then not only do their answers satisfy the winning condition but the marginal distributions are in fact completely *random*. In particular, $a \in \{0, 1\}$ is a uniformly random bit. This follows, e.g., from our discussion in the exercise class, where we showed that

$$\begin{aligned} \Pr(a = 0, b = 0) &= \frac{1}{2} \cos^2(\alpha - \beta), & \Pr(a = 0, b = 1) &= \frac{1}{2} \sin^2(\alpha - \beta), \\ \Pr(a = 1, b = 0) &= \frac{1}{2} \sin^2(\alpha - \beta), & \Pr(a = 1, b = 1) &= \frac{1}{2} \cos^2(\alpha - \beta). \end{aligned}$$

You can also easily verify this by hand, e.g., in the next exercise class.

The randomness obtained in this way is also *private*. We will only discuss this in a very heuristic sense and you are not expected to follow the details, but I would still like to give you an impression. Suppose that apart from Alice and Bob, there is also an evil eavesdropper (Evan) who would like to learn about the random bits generated in this way. Their joint state can be described by a pure state $|\psi\rangle_{ABE}$. If Alice and Bob indeed share the maximally entangled state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ (or for that matter any “pure” state) then it must be the case that $|\psi\rangle_{ABE} = |\Gamma\rangle_{AB} \otimes |\psi\rangle_E$. We will hopefully see how to formalize this statement later in the course. It follows that the measurement outcomes of Alice and Bob, hence in particular the random bit a , are completely uncorrelated from any measurement that Evan can do on his E system.

All this means that the referee can use the players’ answers to generate private randomness – they simply lock Alice and Bob (best thought of as quantum devices) into his laboratory, ensure that the devices cannot communicate, and interrogate them with questions, as in the following picture:



But of course, the referee cannot in general trust Alice and Bob to actually play the strategy above! So this observation might seem not very useful at first glance. . .

However, what if the optimal strategy for winning the CHSH game was actually unique? In this case, the referee could *test* Alice and Bob with randomly selected questions and check that they pass the test every time. After a while, the referee might be confident that the players are in fact able to win the CHSH game every time. But then, by uniqueness of the winning strategy, the referee should in fact know the precise strategy that Alice and Bob are pursuing! The referee in this case would *not* have to put any trust in Alice and Bob – they would prove their worth by winning the CHSH game every time around.

This remarkable idea for generating private random bits was first proposed by Colbeck. (Note that we need private random bits in the first place to generate the random questions – thus this protocol proposes to achieve a task known as *randomness expansion*. Private random bits cannot be generated without an initial seed of random bits.) The argument sketched so far is of course not rigorous at all: ignoring questions of robustness, we need to take into account that Alice and Bob may not behave the same way every time we play the game, may have a (quantum) memory, etc.

These challenges can be circumvented and secure randomness expansion protocols using completely untrusted devices do exist (see, e.g. the 2016 article *Certified randomness in quantum physics* by Acín and Masanes). This general line of research is known as *device-independent quantum cryptography*, since it does not rely on assumptions on the inner workings of the devices involved, but only on their observed correlations. Other applications include device-independent quantum key distribution.

14.4 Bonus: Rigidity of the quantum winning strategy

We will now argue that the optimal winning strategy for the CHSH game is indeed essentially unique (the argument will clarify what this means precisely), as alluded to in the previous section. This is known as the *rigidity property* of the CHSH game, or the fact that the EPR pair state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ can be *self-tested* (via the CHSH game).

The basic idea is simple: When we proved Tsirelson's inequality, the only ingredient that we used is the Cauchy-Schwarz inequality for carefully-chosen vectors $|\alpha\rangle$ and $|\beta\rangle$. Suppose that we have a quantum strategy that achieves the optimal winning strategy. In this case, the Cauchy-Schwarz inequality in [Ineq. \(14.11\)](#) holds with equality. This is the case if and only if $|\alpha\rangle = \frac{|\beta\rangle}{\sqrt{2}}$ (as discussed on the first problem set, the vectors should be proportional; but we know their norm and inner product, which fixes the proportionality constant), i.e.,

$$\begin{aligned} (A(0) \otimes I) |\Psi\rangle &= \left(I \otimes \frac{B(0) + B(1)}{\sqrt{2}} \right) |\Psi\rangle, \\ (A(1) \otimes I) |\Psi\rangle &= \left(I \otimes \frac{B(0) - B(1)}{\sqrt{2}} \right) |\Psi\rangle. \end{aligned} \tag{14.12}$$

As a warmup, let us assume that we know that we are dealing with two qubits and that the observables $A(x)$, $B(y)$ are as described in [Section 14.3.1](#) above. In this case, [Eq. \(14.12\)](#) determines the state $|\Psi\rangle$ uniquely! Indeed, using [Eqs. \(14.7\)](#) and [\(14.8\)](#), [Eq. \(14.12\)](#) read

$$(Z \otimes I) |\Psi\rangle = (I \otimes Z) |\Psi\rangle, \quad \text{or equivalently} \quad (Z \otimes Z) |\Psi\rangle = |\Psi\rangle, \tag{14.13}$$

$$(X \otimes I) |\Psi\rangle = (I \otimes X) |\Psi\rangle, \quad \text{or equivalently} \quad (X \otimes X) |\Psi\rangle = |\Psi\rangle, \tag{14.14}$$

If $|\Psi\rangle = \Psi_{00}|00\rangle + \Psi_{01}|01\rangle + \Psi_{10}|10\rangle + \Psi_{11}|11\rangle$ is a general two-qubit state then [Eq. \(14.13\)](#) implies that $\Psi_{01} = -\Psi_{01}$ and $\Psi_{10} = -\Psi_{10}$, hence $\Psi_{01} = \Psi_{10} = 0$, and [Eq. \(14.14\)](#) implies that $\Psi_{00} = \Psi_{11}$. Together, it follows that $|\Psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ (up to an irrelevant overall phase).

Of course, if we consider an arbitrary winning strategy then we do *not* know what the $A(x)$, $B(y)$ look like either – apart from that they are Hermitian and have eigenvalues ± 1 , hence they square to the identity. Thus let us consider a general strategy, given by a state $|\Psi\rangle \in \mathbb{C}^{d_A} \otimes \mathbb{C}^{d_B}$ and observables $A(x)$, $B(y)$ with eigenvalues ± 1 , and let us assume that it achieves the optimal winning probability, so that [Eq. \(14.12\)](#) holds. In the strategy in [Section 14.3.1](#), we found that $A(0) = Z$ and $A(1) = X$, and likewise for $(B(0) \pm B(1))/\sqrt{2}$ ([Eqs. \(14.7\)](#) and [\(14.8\)](#)). The Pauli matrices have the property that $X^2 = Z^2 = I$ and they “anticommute”, meaning that $XZ = -ZX$. The key idea is to show that these relations are implied by [Eq. \(14.12\)](#), hence hold for a general quantum strategy that achieves the optimal winning probability. Indeed, we already know that $A(0)^2 = A(1)^2 = I$ and the $(B(0) \pm B(1))/\sqrt{2}$ anticommute, since

$$\frac{B(0) + B(1)}{\sqrt{2}} \frac{B(0) - B(1)}{\sqrt{2}} = \frac{B(1)B(0) - B(0)B(1)}{2} = -\frac{B(0) - B(1)}{\sqrt{2}} \frac{B(0) + B(1)}{\sqrt{2}},$$

where we used that $B(y)^2 = I$ for $y \in \{0, 1\}$. The point now is we can use [Eq. \(14.12\)](#) to see that

$$\begin{aligned} (A(0)A(1) \otimes I) |\Psi\rangle &= \left(I \otimes \frac{B(0) - B(1)}{\sqrt{2}} \frac{B(0) + B(1)}{\sqrt{2}} \right) |\Psi\rangle \\ &= -\left(I \otimes \frac{B(0) + B(1)}{\sqrt{2}} \frac{B(0) - B(1)}{\sqrt{2}} \right) |\Psi\rangle = -(A(1)A(0) \otimes I) |\Psi\rangle, \end{aligned}$$

meaning that the $A(x)$'s anticommute on the state $|\Psi\rangle$:

$$(A(0)A(1) \otimes I) |\Psi\rangle = -(A(1)A(0) \otimes I) |\Psi\rangle. \tag{14.15}$$

Suppose for a moment that this holds on the level of operators, i.e., that

$$A(0)A(1) = -A(1)A(0). \tag{14.16}$$

Now suppose that $|\phi\rangle$ is an eigenvector of $A(0)$ with eigenvalue λ , then

$$A(0)A(1) |\phi\rangle = -A(1)A(0) |\phi\rangle = -\lambda A(1) |\phi\rangle,$$

so $A(1) |\phi\rangle$ is an eigenvector of $A(0)$ with eigenvalue $-\lambda$. By assumption, $A(0)$ has only eigenvalues ± 1 (it squares to the identity). Moreover, $A(1)^2 = I$, so if we apply $A(1)$ twice we return to the original eigenvector

that we started with. This means that $A(1)$ exchanges the two eigenspaces of $A(0)$; in particular that they must have the same multiplicity m_A . This implies that there is a unitary $U_A: \mathbb{C}^{d_A} \rightarrow \mathbb{C}^2 \otimes \mathbb{C}^{m_A}$ such that

$$U_A A(0) U_A^\dagger = Z \otimes I \quad \text{and} \quad U_A A(1) U_A^\dagger = X \otimes I.$$

Similarly, we already discussed that the operators $(B(0) \pm B(1))/\sqrt{2}$ anticommute with each other, and using Eq. (14.12) and the fact that $A(x)^2 = I$ for $x \in \{0, 1\}$ we find that they square to the identity on the state $|\Psi\rangle$:

$$\left(I \otimes \left(\frac{B(0) \pm B(1)}{\sqrt{2}} \right)^2 \right) |\Psi\rangle = |\Psi\rangle. \quad (14.17)$$

Let us also assume for the moment that this holds on the level of operators, i.e.,

$$I \otimes \left(\frac{B(0) + B(1)}{\sqrt{2}} \right)^2 = I \otimes \left(\frac{B(0) - B(1)}{\sqrt{2}} \right)^2 = I. \quad (14.18)$$

Thus, by the same argument as above we can find a unitary $U_B: \mathbb{C}^{d_B} \rightarrow \mathbb{C}^2 \otimes \mathbb{C}^{m_B}$ such that

$$U_B \frac{B(0) + B(1)}{\sqrt{2}} U_B^\dagger = Z \otimes I \quad \text{and} \quad U_B \frac{B(0) - B(1)}{\sqrt{2}} U_B^\dagger = X \otimes I.$$

In other words, up to a change of basis, the two operators look precisely as in Eqs. (14.7) and (14.8)! Now, Eq. (14.12) implies that

$$\begin{aligned} (Z \otimes I \otimes Z \otimes I)(U_A \otimes U_B) |\Psi\rangle &= (U_A \otimes U_B) |\Psi\rangle \\ (X \otimes I \otimes X \otimes I)(U_A \otimes U_B) |\Psi\rangle &= (U_A \otimes U_B) |\Psi\rangle \end{aligned}$$

where $(U_A \otimes U_B) |\Psi\rangle$ is a state in $\mathbb{C}^2 \otimes \mathbb{C}^{m_A} \otimes \mathbb{C}^2 \otimes \mathbb{C}^{m_B}$. Now it follows from the above that

$$(U_A \otimes U_B) |\Psi\rangle = W \left(\frac{1}{\sqrt{2}} (|00\rangle + |11\rangle) \otimes |\gamma\rangle \right),$$

where W is the unitary that rearranges the tensor factors in the obvious way and $|\gamma\rangle$ is some arbitrary state in $\mathbb{C}^{m_A} \otimes \mathbb{C}^{m_B}$. This is exactly what we wanted to show: *After a change of basis, Alice and Bob share a maximally entangled pair of qubits (along with possibly some irrelevant state $|\gamma\rangle$ on some other system), and the observables act as in the strategy of Section 14.3.1 on this pair of qubits.*

We still need to overcome one last hurdle. Namely, in the above argument we used that Eqs. (14.16) and (14.18), but in fact we only know these identities to hold on the state $|\Psi\rangle$, see Eqs. (14.15) and (14.17).

Why are we satisfied with anticommutativity on the state, and how does it impact the argument? We can expand

$$|\Psi\rangle = \sum_{i=1}^r s_i |e_i\rangle \otimes |f_i\rangle,$$

where the $|e_i\rangle$ and $|f_i\rangle$ are orthonormal and $s_i > 0$. This is called the *Schmidt decomposition* and we will discuss it in more detail in a later lecture. If $r = d_A = d_B$, then it is not hard to see that Eqs. (14.15) and (14.17) imply Eqs. (14.16) and (14.18), respectively. In general, however, this is not the case, but we can consider the subspace $\mathcal{H}_A := \text{span}\{|e_i\rangle\}$ of \mathbb{C}^{d_A} and $\mathcal{H}_B := \text{span}\{|f_i\rangle\}$ of \mathbb{C}^{d_B} . Then, $|\Psi\rangle \in \mathcal{H}_A \otimes \mathcal{H}_B$, the $A(x)$'s are block-diagonal with respect to $\mathcal{H}_A \oplus \mathcal{H}_A^\perp$, and the $B(y)$'s are block-diagonal with respect to $\mathcal{H}_B \oplus \mathcal{H}_B^\perp$ (the latter follows from Eq. (14.12)). Thus we may restrict Alice's Hilbert space and the observables $A(x)$'s to the subspace \mathcal{H}_A , and Bob's Hilbert space and the observables $B(y)$'s to \mathcal{H}_B , and the restricted observables satisfy Eqs. (14.16) and (14.18), so the argument we gave above applies. In this way we can arrive at the following rigidity theorem for the CHSH game:

Theorem 14.1: Rigidity of the CHSH game

Consider an optimal quantum strategy for the CHSH game given by observables $A(x), B(y)$ for $x, y \in \{0, 1\}$ with $A(x)^2 = I$ and $B(y)^2 = I$, and a state $|\Psi\rangle \in \mathbb{C}^{d_A} \otimes \mathbb{C}^{d_B}$. Then there are isometries $V_A: \mathbb{C}^2 \otimes \mathbb{C}^{m_A} \rightarrow \mathbb{C}^{d_A}$ and $V_B: \mathbb{C}^2 \otimes \mathbb{C}^{m_B} \rightarrow \mathbb{C}^{d_B}$ and a state $|\gamma\rangle \in \mathbb{C}^{m_A} \otimes \mathbb{C}^{m_B}$ s.th.

1. $|\Psi\rangle = (V_A \otimes V_B)(|\Phi\rangle \otimes |\gamma\rangle)$, where $|\Phi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ is the EPR pair,
2. $V_A^\dagger A(0)V_A = Z \otimes I$, $V_A^\dagger A(1)V_A = X \otimes I$, and similarly for $B(y)$.

There is also a robust version of this result: if a quantum strategy achieves an almost optimal winning probability, then it is close to the strategy of [Section 14.3.1](#) in some precise sense. This then is the “proper” rigidity theorem for the CHSH game. For a beautiful proof of this result using approximate group representation theory, see Vidick’s 2018 UCSD summer school notes *Quantum multiplayer games, testing and rigidity*.

There are many further aspects that one could discuss. For example, how do winning probabilities and optimal strategies behave when one plays many instances of a game – either in multiple rounds (sequentially) or even at the same time (in parallel)? It is clear that if p is the optimal winning probability for a single instance then for n instances the winning probability is at least p^n – but conceivably one might be able to do better by using strategies that exploit correlations or entanglement in a clever way. . .

14.5 Bonus: Non-signaling strategies

It is interesting to ask “why” it is not possible to win the CHSH game perfectly. One systematic way of studying this is to note that in both the classical and the quantum setting, a choice of strategy determines a conditional probability distribution $p(a, b|x, y)$, where $a, b, x, y \in \{0, 1\}$, which is the only data needed to compute the winning probability:

$$p_{\text{win}} = \frac{1}{4} \sum_{x, y} \sum_{a, b \text{ s.th. } a \oplus b = xy} p(a, b|x, y)$$

Abstracting away, what property should the conditional distribution $p(a, b|x, y)$ satisfy? Since Alice and Bob are locked away from each other in separate rooms and cannot learn about each other’s questions, we should require that the marginal distribution of Alice’s output,

$$p(a|x, y) := \sum_b p(a, b|x, y),$$

should not depend on Bob’s question y . Similarly the marginal distribution of Bob’s output,

$$p(b|x, y) := \sum_a p(a, b|x, y),$$

should not depend on Alice’s question x . Conditional probability distributions with this property are called *non-signaling*. It is easy to verify that this holds for both classical and quantum strategies.

Is non-signaling responsible for the fact that we cannot win the CHSH game perfectly? It turns out that the answer is “no”. Consider the following conditional probability distribution, which is known as the *PR (Popescu-Rohrlich) box*:

$$p(a, b|x, y) = \begin{cases} \frac{1}{2} & \text{if } a \oplus b = xy, \\ 0 & \text{otherwise.} \end{cases}$$

Clearly under this distribution Alice and Bob win the CHSH game with 100% probability. On the other hand, it is easy to see that this distribution is *non-signaling* in the sense defined above. Indeed, note that for all $a, b, x, y \in \{0, 1\}$ it holds that $p(a|x, y) = \frac{1}{2}$ (for every a, x, y there is exactly one b such that $a \oplus b = xy$), and likewise $p(b|x, y) = \frac{1}{2}$.

Are PR boxes ‘real’? Clearly one has to go beyond quantum mechanics to realize them, but perhaps our understanding of physics is incomplete and they are still ‘physical’, and explained by some more generalized theory of nature? One interesting clue is from complexity theory. Consider the inner product function $\text{IP}(\mathbf{x}, \mathbf{y}) := \bigoplus_{i=1}^n x_i y_i$ on pairs of bit strings $x, y \in \{0, 1\}^n$. If Alice receives \mathbf{x} and Bob receives \mathbf{y} then to compute this function it is known that to compute $\text{IP}(\mathbf{x}, \mathbf{y})$ they in general need to send n bits of information. This is quite natural, since every bit x_i and y_i can in principle affect the output. In fact, even if they are allowed to use entanglement and if they are allowed to send qubits instead of bits they need to send $\Omega(n)$ qubits to compute the inner product function. *However, using a perfect winning strategy for the CHSH game (e.g., using PR boxes) one can compute the inner product by exchanging a single bit of information.* Alice and Bob simply run their winning strategy for the CHSH game n times, with questions x_i and y_i . Then they obtain answers a_i and b_i on their side that satisfy

$$\text{IP}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n x_i y_i = \sum_{i=1}^n a_i \oplus b_i = \left(\sum_{i=1}^n a_i \right) \oplus \left(\sum_{i=1}^n b_i \right).$$

Thus, a single bit of information – either the XOR of Alice’s outcomes, $\sum_{i=1}^n a_i$, or the XOR of Bob’s outcomes, $\sum_{i=1}^n b_i$, suffices for the other party to compute the inner product exactly. In fact, using a similar idea Alice and Bob could jointly compute *any* function $f(\mathbf{x}, \mathbf{y})$ of two bitstrings $\mathbf{x}, \mathbf{y} \in \{0, 1\}^n$ (one being Alice’s input, the other Bob’s input) by exchanging a single bit of information. This seems very counterintuitive, and perhaps not something we would expect to be true in our universe! If you like, you could see this as complexity-theoretic philosophical motivation why one might believe that there is no perfect winning strategy for the CHSH game. . .

Chapter 15

The quantum road ahead

See the course [Advanced Quantum Information and Computation](#) in the coming summer semester!