

# Computing and Quantum Computing

Malik Magdon-Ismail

November 28, 2022

# Contents

<b>1</b>	<b>Outline and Introduction to Classical Computing</b>	<b>3</b>
1.1	Outline . . . . .	3
1.2	What is classical computing? . . . . .	3
1.3	Unsolvable Computing Problems . . . . .	5
1.4	What Changes When We Move to Quantum Computing . . . . .	6
<b>2</b>	<b>Turing Machines and Unsolvable Problems</b>	<b>7</b>
2.1	Turing Machines . . . . .	7
2.2	Program Testing is Unsolvable . . . . .	8
2.3	The Halting Problem is Unsolvable . . . . .	8
<b>3</b>	<b>Efficiency: The Class P</b>	<b>10</b>
<b>4</b>	<b>Solvable versus Verifiable</b>	<b>12</b>
4.1	Polynomially Verifiable and NP . . . . .	13
4.2	Nondeterministic Polynomial (NP) and Polynomially Verifiable . . . . .	14
<b>5</b>	<b>A Hardest Problem in NP: CIRCUIT-SAT</b>	<b>16</b>
5.1	Boolean Circuits . . . . .	16
5.2	Cook-Levin Theorem . . . . .	16
5.2.1	Fast TMs and Small Circuits . . . . .	16
5.2.2	Proof of the Cook-Levin Theorem . . . . .	17
5.3	Example: Solving CLIQUE using CIRCUIT-SAT . . . . .	17
<b>6</b>	<b>NP-Completeness: Other NP-Complete Problems.</b>	<b>19</b>
6.1	Reducing CIRCUIT-SAT to 3-SAT . . . . .	19
6.2	Other NP-complete Problems: INDEPENDENT-SET, CLIQUE, VERTEX-COVER . . . . .	20
<b>7</b>	<b>Linear Algebra and Complex Vector Spaces</b>	<b>22</b>
7.1	Complex Numbers . . . . .	22
7.2	Complex Vector Spaces . . . . .	23
7.3	Basis . . . . .	24
7.4	Spectral Theorem . . . . .	25
7.5	Hadamard Matrix . . . . .	25
<b>8</b>	<b>Quantum Mechanics</b>	<b>26</b>
8.1	Postulates of Quantum Mechanics . . . . .	26
8.2	Spin . . . . .	29
<b>9</b>	<b>Dynamics</b>	<b>31</b>
9.1	Classical Dynamics . . . . .	31
9.2	Quantum Dynamics . . . . .	32
9.3	Ensembles of Independent Particles . . . . .	34

<b>10 Classical Computing Using Linear Algebra</b>	<b>37</b>
10.1 Classical Bits . . . . .	37
10.2 Quantum Bits . . . . .	38
10.3 Classical Computing Gates . . . . .	39
10.4 Circuits . . . . .	40
<b>11 Reversible Gates and Quantum Gates</b>	<b>42</b>
11.1 Reversible Gates . . . . .	42
11.1.1 Controlled-NOT . . . . .	42
11.1.2 Toffoli Gate . . . . .	43
11.2 Quantum Gates . . . . .	44
11.2.1 Building Larger Gates . . . . .	45
11.2.2 Practice . . . . .	46
11.3 No Cloning Theorem . . . . .	46
<b>12 Unitary Operator for Classical Functions</b>	<b>48</b>
12.1 The Deutsch-Jozsa Problem . . . . .	48
12.2 Converting Boolean Functions to Unitary Operators . . . . .	49
<b>13 Testing Balance of 1-bit Functions</b>	<b>52</b>
13.1 Applying $U_f$ to Superpositions . . . . .	52
13.2 Untangling the Output . . . . .	55
13.3 Quantum Circuit for 1-bit Deutsch-Jozsa . . . . .	56
<b>14 Quantum Circuits</b>	<b>57</b>
14.1 Finding the Operator for a Circuit . . . . .	57
14.2 Building a Circuit for an Operator . . . . .	60
<b>15 Testing Balance of <math>n</math>-bit Functions</b>	<b>61</b>
15.1 Deutsch-Jozsa Algorithm . . . . .	61
<b>16 Philosophy of Quantum Algorithms</b>	<b>65</b>
16.1 Directly Building a Circuit for $U_f$ . . . . .	66
16.2 Circuit Uniqueness . . . . .	67
<b>17 Learning the Weights in a Linear Function</b>	<b>69</b>
17.1 Circuit for Bernstein-Vazirani . . . . .	71
17.2 Algebraic Proof . . . . .	73
<b>18 The Search Problem</b>	<b>74</b>
18.1 Searching for a Unique Element . . . . .	74
18.2 Quantum Circuit for $f$ . . . . .	74
18.3 Quantum Search – Warm Up . . . . .	76
<b>19 Grover’s Iteration</b>	<b>79</b>
19.1 Operator for Reflecting About the Average . . . . .	82

<b>20 Analysis of Grover’s Search Algorithm</b>	<b>86</b>
20.1 Grover’s Coupled Recurrence . . . . .	87
20.2 Solving Grover’s Recurrence . . . . .	88
20.3 Unknown Number of Solutions . . . . .	90
<b>21 Quantum Error Correction</b>	<b>91</b>
21.1 Quantum Redundancy . . . . .	92
21.2 Modeling the Error . . . . .	93
21.3 Detecting Bit-Flip Error . . . . .	94
21.4 Correcting Bit-Flip Error . . . . .	96
21.5 Generalizing to Other Errors . . . . .	97
<b>22 Quantum Circuit for 3-SAT</b>	<b>98</b>
22.1 An Instance of 3-SAT . . . . .	98
22.2 General Case . . . . .	100
<b>23 Quantum Factoring</b>	<b>101</b>
23.1 Factoring and Period Finding . . . . .	101
23.2 Algorithm for Factoring . . . . .	105
<b>24 Quantum FFT and Period Finding</b>	<b>106</b>
<b>25 Quantum Cryptography and Key Exchange</b>	<b>107</b>
<b>26 Quantum Teleportation</b>	<b>108</b>
<b>27 Quantum Information Theory</b>	<b>109</b>

# 1 Outline and Introduction to Classical Computing

## 1.1 Outline

1. Theory of classical computing. Why? Quantum computing is not a sprint, it's a marathon. At least you will have something golden to put into the bank even if QC does not materialize in your lifetime. Also, to study QC and its potential benefits, we need a baseline.
2. Why study QC now if we don't really have Q-computers? Historically, algorithms precede the computing machines that implement them. The Babylonians new how to multiply, but we only had the calculator 4,000 years later. Turing developed the theory of classical computing and the Universal Turing Machine (UTM) in the 1930s, but we only had programmable computers 30 years later.

Let us study QC-algorithms now and hope we do not have to wait 4,000 years for the first viable QC. There are many challenges to be solved, both algorithmically in terms of quantum algorithm design paradigms, and practically in terms of stable physical components that can implement these algorithms.

3. What is QC?
4. QC Formalism
5. QC algorithms and ~~building~~<sup>simulating</sup> our first QC.
6. What does it take to get a viable QC. The fundamental premise of classical computing is that you can write a bit to a register and it is “permanent.” So that when we run algorithms we can remember and refer to things as needed. Well, the bit is not quite permanent. It can get flipped, for example by ambient gamma and other EM rays with probability about  $10^{-12}$  per hour. We have error correcting codes (ECC) to fight this. For example store the bit 0 as 000. Now two bits need to get flipped before it is not recognizable. This is why you pay the big bucks for ECC memory.

So we need a QC to be able to store the equivalent of a bit “permanently”. That is already a challenge, as well as getting good quantum-ECC.

7. If we do get Q-computers, what will become a sprint is to develop good quantum programming languages. A programming language is just an interface between your mind, where you build algorithms, and the hardware. A compiler is the interpreter.

What does it take to have a quantum-programming language. For starters, we need a good formalism within which to design quantum algorithm, which also allows mixing of classical and quantum algorithms.

## 1.2 What is classical computing?

- What is computing?
- Computing uses algorithms to solve computing problems.
- What is an algorithm? What is a computing problem?

A computing problem is a language. For example, consider the computing task of deciding if a number is odd. Define the language

$$\mathcal{L} = \{0, 000, 00000, 0000000\} = \{0^{2^{k-1}} \mid k \in \mathbb{N}\}. \quad (1)$$

The alphabet we will use is binary. Any string can be converted to binary. Testing if a string  $w$  is in  $\mathcal{L}$  amounts to “computing oddness.” A DFA can solve this problem.

$$[\text{Draw the DFA}] \quad (2)$$

The notion of an algorithm is embedded implicitly in the DFA which is used solve the problem. Every DFA is an “algorithm.” The DFA is easy to build using standard physical components. For example a vending machine.

Formulating a computing problem as a language has withstood the test of time. Example problems. Prime factorization,

$$\mathcal{L} = \{(N, K) \mid N \text{ has a prime factor } K\}. \quad (3)$$

Shortest path problem. Equality,

$$\mathcal{L} = \{0^{n-1}1^n \mid n \geq 0\} \quad (4)$$

There is no DFA to solve this problem.

*Proof.* By contradiction. Assume there is some DFA with  $k$  states solves this problem. Show there are two strings which end in the same state, but one is in  $\mathcal{L}$  and one is not. This gives the desired contradiction, because that state cannot be both accepting and rejecting. ■

This is a strong theorem. It is not about DFA-programming skills. It is not you and I that cannot find the DFA. No one can. It provably does not exist. So what now? Since we would like to solve equality, we need a more powerful machine.

The fundamental problem with the DFA is that it needs to remember how many 0s have passed. This is not possible with a finite number of internal states which can function as a primitive but bounded counter. We need an external scratch paper. So we add an unbounded RAM (tape) and allow the DFA to move around this ram, read and write. This computer is a Turing Machine.

$$[\text{Draw high-level TM}] \quad (5)$$

This machine is easy to implement modulo the infinite tape. But in practice we can implement it and run it on any specific input string. If the machine comes to the end of the tape and needs more tape, we just fetch it more tape (memory on demand). The Turing Machine is the definition of an algorithm. This is the Church Turing thesis.

This notion of algorithms which need to dynamically create more memory because they do not know how much memory they will need for their specific input instance are common in computing. For example dynamic hashing for efficient search. We want to hash with no collisions, so the table size depends on the number of items hashed. If ahead of time you don’t know how many items will be hashed, you increase the table size (add memory) on an as needed basis. This type of algorithm is very important to Google.

When you have a theory of computing based on some computing machine, it is very important to make sure you can build it. Otherwise what use is the computer if it can’t be built?

So, you have designed your fancy TM and the engineer who will build it lives on the other side of the country. What do you do? Well you write down the description of your TM and email it to them. What does this description look like?

$$\text{states, transition/move/read/write instructions, halting states, etc..} \quad (6)$$

The engineer gets this description, reads it and builds the machine. Your description is just some big string of characters (for example in ASCII) which we can convert to binary. Let  $M$  be your machine and  $\langle M \rangle$  this binary string which describes  $M$ .

$$M \rightarrow \langle M \rangle, w \xrightarrow{\text{email}} \boxed{\text{engineer} \xrightarrow{\text{build}} M \rightarrow \text{run on input } w} \rightarrow \text{YES/NO.} \quad (7)$$

The boxed process performed by the engineer is quite complex. Do you think we could build a Turing machine to implement it instead of using an engineer? It would have to work no matter what its input  $\langle M \rangle, w$  was. This was one of the stunning results from Turing's seminal work. It is possible to build a single Turing machine which does the engineer's job. Well not quite. This TM does not have the ability to build and run  $M$ . It simulates the building and running of  $M$  on its tape. This grand TM is called a *Universal Turing Machine* (UTM). It is what we call a computer today.  $\langle M \rangle$  is the program fed into the computer and  $w$  is the input to the program.

### 1.3 Unsolvability of Computing Problems

The requirement that we build the computing machine to implement any algorithm to solve a computing problem is necessary. Otherwise computing is useless. This requirement is also very limiting. Let us see a remarkable consequence.

If it is buildable, you have to be able to describe it to your engineer. In which case you have to be able to write down a description, which is essentially some massive binary string. So every computing machine has a binary description. This means every computing problem that can be solved by a computing machine has an associated binary description. Further, two different computing problems cannot be solved by the same computing machine.

$$\text{solvable computing problem} \leftrightarrow \text{computing machine } M \text{ for problem} \leftrightarrow \text{description } \langle M \rangle \text{ of } M. \quad (8)$$

Mathematically, there is an injection from solvable problems to finite binary strings (the description of the TM that solves the problem).

$$|\{\text{solvable computing problems}\}| \leq |\{\text{all finite binary strings}\}| = \text{countable.} \quad (9)$$

Here is a lexicographic ordering of all finite binary strings,

$$B = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, \dots\} \quad (10)$$

Consider any infinite binary string, for example,

$$10011101 \dots \quad (11)$$

Define a computing problem  $\mathcal{L}$  for this infinite binary string as follows. Align each bit with the corresponding position in  $B$  with 1 meaning the string in  $B$  is in  $\mathcal{L}$  and 0 meaning the string in  $B$  is not in  $\mathcal{L}$ . Our infinite binary string defines the language

$$\mathcal{L} = \{\varepsilon, 00, 01, 10, 000, \dots\} \quad (12)$$

Every infinite binary string defines a different problem. Hence,

$$|\{\text{all computing problems}\}| = |\{\text{all infinite binary strings}\}| = \text{uncountable}. \quad (13)$$

The conclusion is that there are uncountably many computing problems that do not have a computing machine that solves them. The only requirement here is that a computing machine should be buildable, that is describable.

**Theorem 1.1.** There are countably many computing machines and there are uncountably many computing problems that cannot be solved by a computing machine.

There are many unsolvable computing problems. To catch and put some of them on display, we need to delve deeper into Turing Machines. Before we do, let's see how QC will change things.

## 1.4 What Changes When We Move to Quantum Computing

The problems we want to solve don't change. So a computing problem is still a language, and there are uncountably many of those.

We still want to build quantum computers to solve problems, so a quantum computer must be describable. But then Theorem 1.1 says there are uncountably unsolvable computing problems, even on a quantum computer.

So, what do we gain? What changes? Efficiency. We hope that problems that can't be solved efficiently on classical computers can be solved efficiently on quantum computers.



## 2 Turing Machines and Unsolvable Problems

Let us summarize. Any theory of computing machines must involve computers that are buildable. It does not matter whether we are talking about DFA, classical computer, biological computers, molecular computers or quantum computers. Hence a computer  $M$  must be describable, that is there is a binary encoding  $\langle M \rangle$  that can be used to build  $M$ . This means the computing machines are countable and can be listed,

$$\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle M_5 \rangle, \langle M_6 \rangle, \dots \quad (14)$$

A computing problem is specified by an infinite binary string. There are uncountably many computing problems. This means there are uncountably many problems which are not solvable by our computing machines. The only thing we require is that a computing machine be describable.

### 2.1 Turing Machines

Consider the following TM run on inputs 101.

$$[\text{Draw infinite looping TM}] \quad (15)$$

This TM infinite loops by jiggling left and right. The powerful ability of a TM to move at will is what leads to this possibility of infinite loops. A TM  $M$  run on input  $w$  can do one of three things.

$$M(w) = \begin{cases} \text{HALT with YES;} \\ \text{HALT with NO;} \\ \text{infinite loop.} \end{cases} \quad (16)$$

If  $M$  always halts,  $M$  is a decider. If  $M$  may infinite loop, it is a recognizer. The possibility of an infinite loop is unacceptable in practice. Hence, a solvable problem (language) is one which can be solved by a decider. A problem  $\mathcal{L}$  is recognizable if there exists  $M$  for which

$$\begin{aligned} w \in \mathcal{L} &\rightarrow M(w) = \text{HALT with YES} \\ w \notin \mathcal{L} &\rightarrow M(w) = \text{HALT with NO or infinite loop} \end{aligned} \quad (17)$$

That is, there is a recognizer which “solves” the problem. However the possibility of an infinite loop is not a practical solution. A problem  $\mathcal{L}$  is decidable if there exists  $M$  for which

$$\begin{aligned} w \in \mathcal{L} &\rightarrow M(w) = \text{HALT with YES} \\ w \notin \mathcal{L} &\rightarrow M(w) = \text{HALT with NO} \end{aligned} \quad (18)$$

That is, there is a decider which solves the problem. Note, every decider is a recognizer, so

$$\{\text{deciders}\} \subset \{\text{recognizers}\}. \quad (19)$$

Also, every decidable language is also recognizable. Since both deciders and recognizers are describable, they are both countable. Hence there are uncountably many undecidable problems and uncountably many unrecognizable problems.

In practice, we do not build a new TM for every problem we want to solve. We just build the universal TM  $U_{\text{TM}}$  which can simulate the operation of *any*  $M$  on *any* input  $w$ ,

$$U_{\text{TM}}(\langle M \rangle \# w) = \begin{cases} \text{HALT with YES,} & \text{if } M(w) = \text{HALT with YES;} \\ \text{HALT with NO,} & \text{if } M(w) = \text{HALT with NO;} \\ \text{infinite loop,} & \text{if } M(w) = \text{infinite loop.} \end{cases} \quad (20)$$

## 2.2 Program Testing is Unsolvable

It is not possible to write a program which can test if another program works correctly on a particular input. Define the set of programs with their accepting inputs  $\mathcal{L}_{\text{TM}}$ ,

$$\mathcal{L}_{\text{TM}} = \{\langle M \rangle \# w \mid M \text{ is a TM and } M(w) = \text{HALT and YES}\}. \quad (21)$$

$\mathcal{L}_{\text{TM}}$  consists of programs and their inputs which terminate successfully. Can we solve the problem  $\mathcal{L}_{\text{TM}}$ . That means, given an input program  $\langle M \rangle$  and its input  $w$ , you must determine if  $M$  halts on  $w$  with YES or not. Isn't this problem trivially solvable:

$$\text{Build } M \text{ and run it on } w, \text{ outputting the result.} \quad (22)$$

We don't build and run  $M$ , we just simulate running  $M$  using  $U_{\text{TM}}$ . Not so fast. The problem is that  $M$  may infinite loop. In this case our first solution does not output an answer. What next? Since we can just build and run  $M$ , we need to do something more sophisticated, that looks more deeply at the inner workings of  $M$ .

Can we build another program  $A_{\text{TM}}$  that takes as input  $\langle M \rangle \# w$  and halts with YES if  $M$  halts with YES on  $w$ , and halts with NO otherwise,

$$A_{\text{TM}}(\langle M \rangle \# w) = \begin{cases} \text{halt with YES,} & \text{if } M(w) = \text{halt with YES;} \\ \text{halt with NO,} & \text{otherwise.} \end{cases} \quad (23)$$

$A_{\text{TM}}$  is a decider for  $\mathcal{L}_{\text{TM}}$ . Unfortunately, the answer is no. There is no program which can test another program's correctness. It just does not exist.  $A_{\text{TM}}$  *does not exist*.

*Proof.* (Contradiction) Assume the program  $A_{\text{TM}}$  exists. We can use  $A_{\text{TM}}$  anyway we want. In particular, we can use it as a subroutine in any other program. Define the program  $D$  as follows.  $D$  takes as input just the encoding  $\langle M \rangle$  of a TM.

$D(\langle M \rangle)$  :

- 1: Run  $A_{\text{TM}}(\langle M \rangle \# \langle M \rangle) \rightarrow \text{YES/NO}$ .
- 2: If YES, output NO. If NO, output YES.

The input to  $A_{\text{TM}}$  is two strings, it can be any two strings. Hence, the first step is a valid input to  $A_{\text{TM}}$ , and since  $A_{\text{TM}}$  is a decider,  $D$  is also a decider. This is because the first step must halt, and the second step is trivial.

$$[\text{Draw Table showing Diagonal Argument Contradiction}] \quad (24)$$

The contradiction comes when we ask what  $D$  says when its input is  $\langle D \rangle$ . If it says YES, then  $A_{\text{TM}}(\langle D \rangle \# \langle D \rangle) = \text{YES}$  by definition of  $A_{\text{TM}}$ , which means  $D$ , by definition, must say the opposite, which is NO, a contradiction. If it says NO, then  $A_{\text{TM}}(\langle D \rangle \# \langle D \rangle) = \text{NO}$  by definition of  $A_{\text{TM}}$ , which means  $D$ , by definition, must say the opposite, which is YES, a contradiction. Either possibility leads to a contradiction. Hence  $A_{\text{TM}}$  cannot exist.  $\blacksquare$

## 2.3 The Halting Problem is Unsolvable

Now the floodgate is open. We can use the non-existence of  $A_{\text{TM}}$  to prove non-existence of other programs using the general methodology of reduction. Consider a program  $\langle M \rangle$  to decide a language  $\mathcal{L}$ . If  $M$  can be used as a subroutine to build  $A_{\text{TM}}$ , then since  $A_{\text{TM}}$  does not exist,  $M$  cannot exist.

This would prove that  $\mathcal{L}$  is undecidable. Let us see how this method of reduction works in a concrete case. Consider the halting problem,

$$\mathcal{L}_{\text{HALT}} = \{\langle M \rangle \# w \mid M \text{ is a TM and } M \text{ halts on } w\}. \quad (25)$$

Note, the difference between  $\mathcal{L}_{\text{TM}}$  and  $\mathcal{L}_{\text{HALT}}$ . To belong in  $\mathcal{L}_{\text{HALT}}$ ,  $M$  just needs to halt on  $w$ , it does not need to say yes. A program that tests if another program halts is super powerful. For example, it could be used to resolve many conjectures in mathematics like Goldbach's conjecture or the twin-primes conjecture. Just write a program to verify if some property holds for all integers by testing the property one by one. If the property fails for any integer the program should halt and report the counterexample. Now testing if this program halts or not will resolve the conjecture that the property holds for all integers.

If we have a program  $H_{\text{TM}}$  that solves  $\mathcal{L}_{\text{HALT}}$ , then we can build  $A_{\text{TM}}$  as follows.

$A_{\text{TM}}(\langle M \rangle \# w)$  :

- 1: Run  $H_{\text{TM}}(\langle M \rangle \# w)$ . If NO, halt with NO. If YES, run step 2.
- 2: Build and run  $M$  on  $w$ , outputting the result.

The  $A_{\text{TM}}$  we constructed always halts and always outputs the correct answer. The build and run method works here because we only use it when we know that  $M$  halts. The conclusion is that  $H_{\text{TM}}$  cannot exist and so  $\mathcal{L}_{\text{HALT}}$  is undecidable. The domino puzzle (PCP) treated as a computing problem where the input is an arbitrary set of dominos is unsolvable. Many problems are unsolvable. Uncountably many.

Halting and programing testing are very important in computer science. Anytime someone comes up with a program to solve a problem, it is important to show that your program always halts and always gives the correct answer. We have just showed that  $A_{\text{TM}}$  and  $H_{\text{TM}}$  do not exist. So what is going on in industry. Are programmers routinely putting out code that is incorrect and untested and/or does not always terminate?

The undecidability of  $\mathcal{L}_{\text{TM}}$  and  $\mathcal{L}_{\text{HALT}}$  are in the general sense. There is no *general* program that can test *any* other program for correctness. However, through ingenuity and proof methods, it may be possible to test a *specific* program for correctness.

Also note, general autograder for CS1-assignments are not possible. So in what sense are CS1 assignments autogradeable? General antivirus programs are not possible. So what exactly are you paying for in antivirus software?

### 3 Efficiency: The Class P

To summarize the Church-Turing thesis, an algorithm is a Turing-Machine. A problem is solvable by an algorithm if there is a TM-decider for the corresponding language.

We talk about efficiency in the context of a specific problem.

$$\mathcal{L} = \{0^n \# 1^n, n \geq 0\}. \quad (26)$$

The simplest TM for this problem zig-zags checking off each 0 with its corresponding 1,

$$[\text{Draw TM for this problem}]. \quad (27)$$

What is the runtime of this TM. This question is ambiguous. On what input? In computer science we identify the “size” of the input by a parameter  $n$ , and we want the runtime for the worst input of size  $n$ . Further, we are only interested in  $n \rightarrow \infty$ , that is asymptotic analysis. This algorithm makes  $n$  scans, and each scan takes about  $n$  steps, so the TM has  $\Theta(n^2)$  runtime. This is the time-complexity of the algorithm/TM.

What if we are interested in the time-complexity of the problem itself, that is the time-complexity of  $\mathcal{L}$ ? How hard a problem is  $\mathcal{L}$ ? There is another TM for  $\mathcal{L}$  that is based on halving the number of unmarked bits in each scan. The number of scans required is  $O(\log n)$ , and each scan is  $n$  operations, so this is a TM with  $O(n \log n)$  runtime. If we are talking about the time-complexity of a problem, we must consider the time-complexity of the best TM for the problem. In this case, you cannot beat  $n \log n$ .

Let us think outside the box and envision a TM with two tapes and two independent read-write heads. The input is on one tape. In one scan you can copy the 0s to the second tape and then mark-off the 1s on the first tape with the 0s on the second tape. So now we have a  $O(n)$  runtime, but with a different TM-architecture. This new architecture is a parallel architecture. It requires us to be able to operate two TM's independently in parallel. So, by moving to a new architecture, we can improve the runtime even more, by a logarithmic factor. So, it looks like the time-complexity of a problem depends on the TM architecture.

Things are now getting complicated. Should we use the best possible architecture? Should we fix the architecture to 1-tape. But in practice, we do use parallelism. The solution we will take is to define a time-complexity by considering broad classes. This has stood both the test of time and practice.

In the end, practice does care about the difference between  $O(n^2)$  versus  $O(n \log n)$  versus  $O(n)$ . But there is a sharp theoretical divide between polynomial runtimes and non-polynomial, i.e. exponential. In practice polynomial TMs are tolerable, exponential ones are intolerable. In theory also, polynomial is a property of a problem, not a particular TM-architecture. This is the extended Church-Turing thesis: A problem which is polynomial on a single tape deterministic TM is polynomial on any reasonable TM-architecture.

Let us justify polynomial as a well-defined architecture independent notion of time-complexity that is to be sought by algorithms/TMs. To do this, we prove the simulation theorem. We show how to simulate our two-tape TM for  $\mathcal{L}$  using one tape. The two tapes are interleaved to form one tape. The heads of the two tapes are marked on the single tape. When the two-tape TM performs some operation using a particular head, the one-tape simulator first finds the mark for the head, and only then can it perform the operation on the relevant tape. If the two tape TM runs in time  $T$ ,

the search for a head's tape takes time  $O(T)$ . This means the runtime for the one tape simulator is  $O(T^2)$ , because we need to find one of the heads at most  $T$  times. We therefore have the simulation theorem,

**Theorem 3.1.** A  $K$ -tape TM with runtime  $t(n)$  can be simulated by a 1-tape TM with runtime  $O(t(n)^2)$ .

So the speedup in going from sequential to parallel is at most square-root. This means if you are polynomial in a parallel architecture, you are at most polynomial<sup>2</sup> on a sequential architecture. If a problem is polynomial on one architecture, it is polynomial on any architecture, . . . , except we are not sure about non-deterministic TM-architectures and Quantum-architectures.

Parallel architectures cannot give you exponential speed up, but could these other two architectures do that. We don't know. Nevertheless, we will see some polynomial speedups for some surprising problems by going to quantum architectures.

So in our theoretical classification, a problem is either polynomial, in the class  $P$ , or exponential. We know many problems that provably do not have polynomial solutions. Testing if a program stops in fewer than  $2^n$  steps on an input of size  $n$  is one such problem. Other problems are solving chess, and in general games of strategy. Many useful problems are polynomial.

There are some super-useful problems in practice: clique, 3-SAT, TSP, . . . . We don't know if these problems are polynomial or exponential. They are on the boundary. These problems are the keys to defining the beautiful theory of NP-completeness. We discuss that next.

## 4 Solvable versus Verifiable

We considered the frequent basket problem that your manager might ask you to solve. For a big input, it won't be easy. Instead of telling your manager you couldn't solve it, you can tell your manager that no one can solve it efficiently today. This is related to the theory of NP-completeness.

Every solvable problem can either be solved by a fast TM or not. If not, every TM for the problem runs in super-polynomial time, i.e. exponential time. So every problem is either fast or slow, where the speed of a problem refers to the speed with which it can be solved. We do not know if the frequent basket problem is fast or slow. It lies in a hazy boundary between fast and slow consisting of all the problems which we cannot definitively place into one or other category.

Here is another example. Consider two sets

$$S_1 = \{3, 5, 3, 11, 6, 2\} \quad S_2 = \{3, 6, 2, 11, 6, 2\} \quad (28)$$

and the following questions.

LARGE-SUM Is there a small subset (at most size  $K$ ) whose sum is at least half the total?

PARTITION Is there any subset whose sum is exactly half the total?

(29)

There are many efficient algorithms to solve the first question. In the general case where the input set has  $n$  elements, an  $n \log n$  solution first sorts the elements and checks the sum of the largest  $K$ . An  $n \log K$  solution uses a minheap to keep track of the top- $K$  elements and checks their sum. A student even suggested an  $O(n)$  algorithm using the linear-time  $K$ -selection algorithm, and then processing the top  $K$  elements larger than that element.

The second problem looks very similar. Instead of asking for a small subset with a large sum, it asks for any subset with an exact sum. Can you partition the set into two equal halves. Well, this small change makes the problem very hard, and this second problem is very similar to the frequent baskets problem. We can't solve it fast and we don't know if it can be solved fast. This problem is also in the hazy region between fast and slow. We are going to build a theory for exactly those problems, because they occur so often in practice and they are all related. We won't cover techniques to solve those problems (you need to take the course in approximation algorithms for that).

Let's focus on partition. Check that  $\text{PARTITION}(S_1) = \overline{\text{NO}}$  and  $\text{PARTITION}(S_2) = \overline{\text{YES}}$ . So imagine your manager gave you  $S_1$  and  $S_2$  and asked you to solve partition. You would answer  $\overline{\text{NO}}$  and  $\overline{\text{YES}}$  respectively. But your manager wasn't born yesterday. You are asked to convince them. They want a proof that your answer is correct. They want a proof that is easy for them to check.

How would you prove that your  $\overline{\text{NO}}$ -answer for  $S_1$  is correct? There is no smart way other than a brute-force check that every subset fails.

How would you prove that your  $\overline{\text{YES}}$ -answer for  $S_2$  is correct? That's easy, just present the subset  $\{3, 6, 6\}$  or the string 110010 which identifies that subset with its 1s. Given this subset, it is easy for your manager to verify the yes answer is correct. This subset 110010 is called the evidence or *certificate*. Without this evidence, it is hard for your manager to verify the  $\overline{\text{YES}}$  answer without actually solving the problem. With the evidence, it is trivial to verify the yes answer.

## 4.1 Polynomially Verifiable and NP

The class of problems NP stands for nondeterministic polynomial. It is also equal to the class of problems whose  $\overline{\text{YES}}$  answer is polynomially verifiable. PARTITION is a problem whose  $\overline{\text{YES}}$ -answer is polynomially verifiable. Let us formally define this.

Consider a problem  $\mathcal{L}$ . The problem  $\mathcal{L}$  is polynomially verifiable if there exists a polynomial certifier  $C$  for  $\mathcal{L}$ . The certifier  $C$  is a TM that verifies if a  $\overline{\text{YES}}$  answer is correct. Specifically, consider any  $w \in \mathcal{L}$ . Then there must exist some evidence  $E$  so that  $C(w\#E) = \overline{\text{YES}}$ . The runtime of the certifier must be bounded by a polynomial in the length of  $w$ .

In our partition example,  $w = S_2$ . The evidence is  $E = 110010$ . And the certifier is a TM whose input is  $w$  and  $E$ , summarised in the algorithm:

- 1: Compute  $\text{sum}(w)$ .
- 2: Compute the sum of  $w$  masked by  $E$ , that is  $\sum_i w[i] \times E[i]$ .
- 3: **if**  $2 \times \text{masked-sum} = \text{sum}$  **then**
- 4:     return  $\overline{\text{YES}}$
- 5: **else**
- 6:     return  $\overline{\text{NO}}$

It is clear that this verifier generalizes to a general instance of PARTITION.

**Definition 4.1** (Polynomially verifiable). A language  $\mathcal{L}$  is polynomially verifiable if there exists a TM  $C$  such that:

- (i) For every  $w \in \mathcal{L}$ , there exists an evidence  $E$  with  $|E| \leq \text{poly}(|w|)$  such that  $C(w\#E)$  has runtime at most a polynomial in  $|w|$  and returns  $\overline{\text{YES}}$ .
- (ii) For every  $w \notin \mathcal{L}$  and every  $E$  with  $|E| \leq \text{poly}(|w|)$ ,  $C(w\#E)$  has runtime at most a polynomial in  $|w|$  and returns  $\overline{\text{NO}}$ .

Notice the asymmetry between  $\overline{\text{YES}}$  inputs and  $\overline{\text{NO}}$  inputs. Also notice that the evidence is given, and we do not have to worry about how the evidence was obtained. That is, the runtime/effort needed to find the correct evidence is not part of the runtime of the certifier.

Let us look at another example.

$$[\text{Draw small graph on 5 vertices}] \tag{30}$$

A graph can be represented as a binary string using the edge sequence

$$e_{1,2}e_{1,3}e_{1,4}e_{1,5}e_{2,3}e_{2,4}e_{2,5}e_{3,4}e_{3,5}e_{4,5}. \tag{31}$$

For our graph, the edge sequence is 1010110101. Note, the number of vertices can be inferred from the number of edges. The CLIQUE problem asks if the graph has a subset of vertices of a particular size such that every pair of vertices in the subset are neighbors. For example size 3. In this case the answer is  $\overline{\text{YES}}$ . What evidence would allow us to quickly verify this answer. The clique itself, which could be represented by the binary string 11010, where the 1s in the string identify the vertices in the clique.

In our case the input is 1010110101#11. the second 11 is binary for 3, the size of the desired clique. The evidence is 11010. The certifier  $C(1010110101\#11\#11010)$  is summarized in the algorithm:

- 1: Compute  $\text{sum}(E)$  and verify that it is at least  $K$ , the desired clique size.

- 2: Identify the clique from E.
- 3: **for** each pair of vertices in the clique **do**
- 4: check that the edge for that pair in the edge-sequence is 1.

This algorithm clearly generalizes and runs in polynomial time in the input size. One can get a linear algorithm for the verifier.

Many popular problems are polynomially verifiable: TSP, Clique, 3-SAT, Knapsack, Independent Set, Vertex Cover, Dominating Set, . . . . It means we can quickly verify a  $\overline{\text{YES}}$  instance given the right evidence. Yet, we don't have polynomial algorithms for any of these problems. Every problem in P that is polynomially solvable is also polynomially verifiable.

## 4.2 Nondeterministic Polynomial (NP) and Polynomially Verifiable

When we study DFA, we encounter nondeterministic finite automata (NFA) as a useful tool to address concatenation and Kleen star.

[Draw example NFA] (32)

When you run the NFA on 100101, the computation branches and we accept if any one of the branches accepts. Using subset states, we can implement an NFA by a DFA with a possible exponential increase in the number of states. So nondeterminism does not add any additional computing capability to DFA. It is just a useful tool.

We can extend nondeterminism to Turing Machines.

[Draw examples of det. TM instruction and nondet. TM instruction] (33)

A nondeterministic TM, at each step, can try out different possibilities. Again this results in a branching of the computation, and the TM accepts if any branch accepts. The runtime is the runtime of the longest branch. So we imagine running all branches simultaneously, and so a TM provides some kind of unbounded parallelism. A nondeterministic TM can be implemented using a deterministic TM, but with an exponential increase in runtime.

The class NP are the problems that can be solved in polynomial time on a nondeterministic TM. It turns out that a computing problem that is polynomially verifiable can be solved in polynomial time on a nondet. TM. Here is the intuition in the context of our clique-problem. Here is a nondet TM  $M$  which uses our certifier as a subroutine in a nondeterministic way. The input graph plus  $K$  is 1010110101#11.

$$1010110101\#11 \rightarrow M \left\{ \begin{array}{l} \text{try } E = 00000 \rightarrow \text{run } C(10110110101\#11\#00000) \rightarrow \overline{\text{YES}} \text{ or } \overline{\text{NO}} \\ \text{try } E = 00001 \rightarrow \text{run } C(10110110101\#11\#00001) \rightarrow \overline{\text{YES}} \text{ or } \overline{\text{NO}} \\ \text{try } E = 00010 \rightarrow \text{run } C(10110110101\#11\#00010) \rightarrow \overline{\text{YES}} \text{ or } \overline{\text{NO}} \\ \text{try } E = 00011 \rightarrow \text{run } C(10110110101\#11\#00011) \rightarrow \overline{\text{YES}} \text{ or } \overline{\text{NO}} \\ \text{try } E = 00100 \rightarrow \text{run } C(10110110101\#11\#00100) \rightarrow \overline{\text{YES}} \text{ or } \overline{\text{NO}} \\ \vdots \\ \text{try } E = 11111 \rightarrow \text{run } C(10110110101\#11\#11111) \rightarrow \overline{\text{YES}} \text{ or } \overline{\text{NO}} \end{array} \right. \quad (34)$$

The nondeterminism is only in trying out different choices for E which causes the TM to branch (exponential number of branches). Each branch runs in polynomial time because the verifier  $C$  is



polynomial. Hence this is a polynomial TM. If any branch accepts, it means there is a 3-clique. This is guaranteed by the properties of the certifier - if there is no 3-clique then the certifier says no for all possible evidence.

It is a little trickier to show that if a problem is in  $NP$  then the problem is polynomially verifiable. The evidence is the sequence of choices made by the nondet. TM on any  $\overline{\text{YES}}$ -branch. The certifier simply runs the nondet. TM using only these choices to verify the  $\overline{\text{YES}}$ . Since the nondet. TM is polynomial, the certifier is also polynomial.

As you can see, the nondet. TM above has unbounded parallelism. To make this computation naively sequential would require running each branch in sequence and since there are exponentially many branches, this is an exponential slowdown. Could there be some other way to make this computation sequential that does not result in an exponential slowdown? That is THE BURNING QUESTION in computer science. Does  $P = NP$ ?

We are asking if polynomially verifiable problems are also polynomially solvable. The smart money says no. Intuitively how could you possibly condense exponentially many branches to a polynomial runtime. Alternatively, polynomial verifiability is easy given the evidence, which is the solution itself. So just because one can quickly verify a solution, it seems like a big leap to think that one can quickly find the solution. Nevertheless, we have no proof either way.

Our next task is to unravel some of the beautiful structure surrounding the polynomially verifiable problems. In particular to identify a hardest such problem, whatever that means.

## 5 A Hardest Problem in NP: CIRCUIT-SAT

Recall our last lecture discussed polynomially solvable (P) versus polynomially verifiable (NP). There is a hardest problem in NP. This problem is so hard that if you can solve it in polynomial time, you can use the solver as a subroutine to solve any NP-problem in polynomial time. We call such hardest problems NP-complete.

### 5.1 Boolean Circuits

A Boolean circuit has vertices which can be logic gates ( $\wedge, \neg, \vee$ ) or inputs, some hard-coded and others user specified.

$$\text{[Draw Example Boolean Circuit]} \tag{35}$$

The output of this circuit is

$$y = (1 \wedge \overline{x_1}) \wedge (x_1 \vee x_2). \tag{36}$$

$x_1x_2 = 01$  is a satisfying assignment of this circuit, which means  $y = 1$ . In general, a circuit is a DAG where the vertices with in-degree 0 are the inputs. The output vertex has out-degree 0. All other vertices are logic gates which have inputs from other vertices and outputs to other vertices.

CIRCUIT-SAT: Given a circuit (DAG) with inputs  $x_1, \dots, x_n$ , is there a satisfying assignment to  $x_1, \dots, x_n$ ?

CIRCUIT-SAT is in NP since a satisfying assignment can be verified by just running the circuit (linear time traversal). The evidence is just the satisfying assignment.

A solver for CIRCUIT-SAT is to try all  $2^n$  assignments. That is an exponential algorithm. Is there a polynomial algorithm? None is known. It is conjectured that there is no algorithm substantially faster than  $2^n$ .

### 5.2 Cook-Levin Theorem

CIRCUIT-SAT is harder than all NP-problems. This means that every other problem in NP is *polynomially reducible* to CIRCUIT-SAT. Specifically, if an alien gave you a black box that solves CIRCUIT-SAT in polynomial time, you can literally use this black-box to solve any NP-problem in polynomial time. This is a deep result, and proving it has two main challenges. The proof must work for any NP problem, *without knowing any specific details of the problem*. We don't know any details of how the black box works, yet we must show that this specific black box can be used to solve any NP problem. We will need a link between fast TMs and small circuits.

#### 5.2.1 Fast TMs and Small Circuits

Consider any fast TM. It can operate on any  $w$ , returning  $\text{TM}(w)$  is  $\text{poly}(|w|)$ -time. The nice thing about a TM is that it can take any-length input  $w$ . In contrast, a circuit has a fixed hard-coded input length. So, let us consider a TM with a fixed input  $w$  of length  $n$  and assume it runs in time  $t(n) \leq \text{poly}(n)$ .

Each step of the TM is something like

$$\text{if in state } q \text{ at slot } i \text{ reading } 0: \text{ write } 1, \text{ move L and transition to state } s \tag{37}$$

In this instruction, the TM head is now at slot  $i - 1$ . This instruction can be implemented by a small circuit. Producing a vertex  $i$  which is now outputting 1 and moving to a new vertex corresponding to the next instruction which has inputs the output of vertex/slot  $i - 1$  as well as current state  $s$  (a binary encoding of it). The next instruction is then also implemented by a small circuit. All these small circuits are put together to form one mega circuit which simulates the working of this TM on the input  $w$ . This mega-circuit is constructed in time  $\text{poly}(t(n))$  and has size  $\text{poly}(t(n))$ .

**Conclusion.** A fast TM on input  $w$  can be quickly converted to a small circuit.

### 5.2.2 Proof of the Cook-Levin Theorem

Consider any NP-problem  $\mathcal{L}$ . It has a fast certifier  $C$  such that  $C(w, E)$  is  $\overline{\text{YES}}$  if  $E$  verifies that  $w \in \mathcal{L}$  and  $\overline{\text{NO}}$  otherwise.  $C$  is a fast TM on input  $w, E$ . This means it can be quickly converted into a small circuit that runs  $C$  on any input  $w, E$ .

[Draw picture of this conversion]. (38)

Treating  $w$  as hard-coded and  $E$  as the “input”, we can now build a fast decider for  $\mathcal{L}$ .

[Draw picture]. (39)

- 1: Given input  $w$
- 2: Transform  $C$  into a certifier circuit for  $w$ , with  $E$  now unknown.
- 3: Use the black-box CIRCUIT-SAT solver to decide if there is a satisfying assignment for  $E$ .
- 4: Output what the black-box says.

The runtime of our decider is  $\text{poly}(t(n))$  to convert  $C$  into its circuit. The circuit has size  $\text{poly}(t(n))$ . Since the black box is polynomial in the size of its input, its runtime is  $\text{poly}(\text{poly}(t(n)))$ . So the total runtime is  $\text{poly}(t(n)) + \text{poly}(\text{poly}(t(n)))$  which is  $\text{poly}(t(n))$ . Since  $t(n) \leq \text{poly}(|w|)$ , the runtime of our decider is  $\text{poly}(|w|)$  as required. We have used the CIRCUIT-SAT black box to solve  $\mathcal{L}$  in polynomial time.

**Theorem 5.1.** CIRCUIT-SAT is NP-complete. This means any problem in NP is polynomially reducible to CIRCUIT-SAT.

In a nutshell,  $C$  is a TM-certifier for  $\mathcal{L}$ . Given  $w$ , we would like to know if there is an evidence  $E$  for  $w$  that satisfies  $C$ . By definition,  $w \in \mathcal{L}$  if and only if the answer is yes. We quickly convert  $C$  into a small circuit-certifier for with  $w$  hard-coded in there. We want to ask the same question, is there an evidence  $e$  that satisfies the circuit-certifier with  $w$  hard-coded. The black-box for CIRCUIT-SAT answers this question in polynomial time. Hence using this black-box, we have a polynomial solver for  $\mathcal{L}$ .

### 5.3 Example: Solving CLIQUE using CIRCUIT-SAT

Let us see how this process works for a specific example, namely CLIQUE. Consider the CLIQUE problem with clique size 3 on this graph,

[Draw graph on 4 nodes]. (40)

The binary sequence encoding of this graph is

$$\langle G \rangle = e_{12}e_{13}e_{14}e_{23}e_{24}e_{34} = 100111. \quad (41)$$

The answer is yes, and the evidence is the vertex subset encoded in

$$E = x_1x_2x_3x_4 = 0111. \quad (42)$$

Let us build a circuit verifier for this problem. It will be clear that the circuit verifier will work for any graph of 4 vertices and any evidence of length 4. The circuit will be small and we will build it “quickly” which is evidenced by us building it manually. (We don’t do things manually unless we can do them quickly 😊.)

First we need a circuit whose output verifies if the evidence is a vertex-subset of size at least 3. We could take an AND of all length-3 subsequences and an OR of all these ANDs.

$$[\text{Draw or of ands of 3-subsets of } E]. \quad (43)$$

This, in general, produces a circuit that is too large. A better solution is to first sort the string, and then take an and of the first  $K$  its. This will verify if there are at least  $K$  ones. A circuit to sort a string of  $n$ -bits can be built with  $O(n \log n)$  gates. This is illustrated below.

$$[\text{Draw circuit sorter}]. \quad (44)$$

The and needs  $O(K)$  gates, so ultimately the circuit has  $O(n \log n)$  gates. We now need a circuit that processes every edge  $e_{i,j}$ . If  $e_{i,j} = 1$ , the circuit outputs 1. If  $e_{i,j} = 0$ , the output is 1 if either  $x_i = 0$  or  $x_j = 0$ . The processing of  $e_{i,j}$  is to compute the output  $e_{i,j} \vee \overline{x_i} \vee \overline{x_j}$ . This is a small circuit with 5 gates. We need one such gadget for every edge, so that is  $\binom{n}{2}$  gadgets, each with 5 gates, checking if any of the edges violates the clique-constraint. Since all clique constraints must be satisfied, we take an AND of all the outputs of these gadgets. The final output is the AND of this output and the and of the the output of the size checker circuit.

$$[\text{Draw final circuit}]. \quad (45)$$

In the final circuit we now treat the evidence as the input as evidence and ask if the circuit is satisfiable. This is where the CIRCUIT-SAT black box is needed.

## 6 NP-Completeness: Other NP-Complete Problems.

To solve any problem  $\mathcal{L} \in \text{NP}$ , build its corresponding certifier circuit for input  $w$  and use a black box solver for CIRCUIT-SAT to solve the problem. If the black box solver is polynomial time, then you have a polynomial solution for your problem  $\mathcal{L}$ . CIRCUIT-SAT is harder than any NP problem and we say that any NP problem is polynomially reducible to CIRCUIT-SAT,

$$\text{NP} \leq_p \text{CIRCUIT-SAT}. \quad (46)$$

Are there other problems with this property, that are harder than any problem in NP? Yes. And showing so is considerably easier than what we had to do to prove this for CIRCUIT-SAT. The reason is that we can use the fact that CIRCUIT-SAT is NP-complete. Specifically consider any problem  $\mathcal{L}_* \in \text{NP}$  and a polynomial black box solver for  $\mathcal{L}_*$ . If we can show that we can use this black box solver to polynomially solve CIRCUIT-SAT, then we would have actually built a black box solver for CIRCUIT-SAT. So, we can polynomially solve any problem  $\mathcal{L} \in \text{NP}$  by first building the solver for CIRCUIT-SAT and then using it to solve  $\mathcal{L}$ . This means the polynomial solver for  $\mathcal{L}_*$  can be used to polynomially solve any problem in NP and

$$\text{NP} \leq_p \mathcal{L}_*. \quad (47)$$

That is,

$$\text{CIRCUIT-SAT} \leq_p \mathcal{L}_* \quad \rightarrow \quad \text{NP} \leq_p \mathcal{L}_*. \quad (48)$$

### 6.1 Reducing CIRCUIT-SAT to 3-SAT

Consider an instance of circuit sat. Here is the circuit which computes  $(1 \wedge \overline{x_1}) \wedge (x_1 \vee x_2)$ ,

$$[\text{Draw circuit.}] \quad (49)$$

Any circuit computes a Boolean expression, so satisfiability of general Boolean expressions is clearly harder than CIRCUIT-SAT. Let us analyze further the circuit above and convert it into satisfiability of a very structured Boolean expression. First replace all non-input vertices by variables that indicate the output of the vertex. We have variables  $v_1, v_2, v_3, v_4, v_5$ ,

$$[\text{Draw circuit with gates and constants replaced by variables.}] \quad (50)$$

The outputs of the variables corresponding to the specific gates in our circuit are given by

$$v_1 = 1 \quad (51)$$

$$v_2 = \overline{x_1} \quad (52)$$

$$v_3 = v_1 \wedge v_2 \quad (53)$$

$$v_4 = x_1 \vee x_2 \quad (54)$$

$$v_5 = v_3 \wedge v_4. \quad (55)$$

These are the propagation rules of the circuit. They explicitly tell us, step by step, how to compute the output of the circuit. For example, for  $x_1 = x_2 = 1$ :

$$v_1 = 1; \quad v_2 = 0; \quad v_3 = 0; \quad v_4 = 1; \quad v_5 = 0. \quad (56)$$

If we wish the output of the circuit to be 1, we additionally require  $v_5 = 1$ . Hence if the following six conditions can be simultaneously satisfied, for some choices of  $x_1, x_2$ , then the circuit is satisfiable:

$$v_1 = 1 \tag{57}$$

$$v_2 = \overline{x_1} \tag{58}$$

$$v_3 = v_1 \wedge v_2 \tag{59}$$

$$v_4 = x_1 \vee x_2 \tag{60}$$

$$v_5 = v_3 \wedge v_4 \tag{61}$$

$$v_5 = 1. \tag{62}$$

We can reduce each condition to a Boolean expression which is true if and only if the condition is satisfied.

$$\begin{aligned} v_1 = 1 &\rightarrow (v_1) \\ v_2 = \overline{x_1} &\rightarrow (v_1 \vee x_1) \wedge (\overline{v_1} \vee \overline{x_1}) \\ v_3 = v_1 \wedge v_2 &\rightarrow (v_3 \vee \overline{v_1} \vee \overline{v_2}) \wedge (\overline{v_3} \vee v_1) \wedge (\overline{v_3} \vee v_2) \\ v_4 = x_1 \vee x_2 &\rightarrow (\overline{v_4} \vee x_1 \vee x_2) \wedge (v_4 \vee \overline{x_1}) \wedge (v_4 \vee \overline{x_2}) \\ v_5 = v_3 \wedge v_4 &\rightarrow (v_5 \vee \overline{v_3} \vee \overline{v_4}) \wedge (\overline{v_5} \vee v_3) \wedge (\overline{v_5} \vee v_4) \\ v_5 = 1 &\rightarrow (v_5). \end{aligned} \tag{63}$$

You should verify the condition on the left is true if and only if the Boolean expression on the right is true. Also convince yourself that a general circuit with its propagation rules can be converted to a set of Boolean expressions like this. The number of Boolean expression equals the number of non-input vertices. Since every conditions must be satisfied, we need every Boolean expression to evaluate to true. Each Boolean expression is the AND of clauses, where each clause is an OR of at most 3 terms. Since we want all 13 clauses to be true, we need to satisfy a massive AND of all the clauses in all the Boolean expressions, a CNF. If there are  $n$  non-input vertices and  $d$  input vertices, the number of clauses is at most  $3n$  and the number of variables is  $n + d$ . Every one of these clauses is satisfiable by appropriately setting the  $n + d$  Boolean variables if and only if the circuit is satisfiable. hence we define the NP-problem 3-SAT.

3-SAT: Given a set of  $n$  clauses over variables  $x_1, \dots, x_n$ , where each clause is the OR of at most 3 terms, is there an assignment to  $x_1, \dots, x_n$  for which every clause is true?

That is, can all clauses be satisfied. Clearly, 3-SAT is in NP. We can solve any instance of CIRCUIT-SAT with  $n$  non-input vertices and  $d$  input vertices by first transforming it to the corresponding set of at most  $3n$  clauses and using a black box solver for 3-SAT on these clauses.

$$[\text{Draw workflow involving 3-sat solver as a black box}] \tag{64}$$

We have proved

$$\text{CIRCUIT-SAT} \leq_p \text{3-SAT}. \tag{65}$$

That is, 3-SAT is NP-complete.

## 6.2 Other NP-complete Problems: INDEPENDENT-SET, CLIQUE, VERTEX-COVER

CIRCUIT-SAT is a relatively complex problem to deal with, involving complex circuits. But, 3-SAT is relatively easy to handle, consisting only of a bunch of clauses with each clause being an OR of at most 3 terms. 3-SAT is a power-tool for finding other NP-complete.

We use the general methodology. To show that  $L$  is NP-complete, we polynomially reduce any known NP-complete problem  $\mathcal{L}_*$  to  $\mathcal{L}$ . That is, we show how we can solve  $\mathcal{L}_*$  if we have a black box polynomial solver for  $\mathcal{L}$ . Let's see this in action by proving that INDEP-SET is NP-complete. We show that 3-SAT is polynomially reducible to INDEP-SET. Consider an instance of 3-SAT, for example,

$$(y \vee x \vee z)(\bar{x} \vee z)(\bar{x} \vee \bar{z})(x \vee \bar{y} \vee \bar{z}). \quad (66)$$

These clauses are satisfiable if and only if one can pick exactly one term from each clause to make true without any conflicts between terms picked to be made true. A conflict is for example picking both  $x$  and  $\bar{x}$  to make true. They conflict because both cannot be made true.

Build a graph as follows. Each clause is a clique with a vertex corresponding to each term. Vertices in different cliques are linked if they correspond to conflicting terms.

$$[\text{Draw the resulting graph.}] \quad (67)$$

If there is an independent set whose size equals the number of clauses, then it means exactly one vertex is picked from each clique and the vertices do not conflict which means they can all be made true. So, if we have a black box solver for INDEP-SET, we can solve 3-SAT, that is

$$3\text{-SAT} \leq_p \text{INDEP-SET}, \quad (68)$$

and so INDEP-SET is NP-complete. We leave it as an exercise for you to formally define the independent set problem. Also, show that CLIQUE and VERTEX-COVER are NP-complete by polynomially reducing them to INDEP-SET. So now we have the NP-complete problems

$$\text{CIRCUIT-SAT, 3-SAT, INDEP-SET, CLIQUE, VERTEX-COVER.} \quad (69)$$

Karp, in his seminal 1972 paper, gave 21 problems in diverse domains that are NP-complete. This list has since exploded to several thousands of problems occurring frequently in practice and in very diverse areas.

We need to solve these problems in practice. These problems are typically phrased as optimization problems. What do we do?

1. Change the objective to an easier one for which there is a polynomial solution.
2. Change the problem.
3. Approximate the solution.

## 7 Linear Algebra and Complex Vector Spaces

We begin our study of QC by reviewing some of the useful math. The complex numbers are perhaps one of humanity's greatest inventions, along with induction, language, written storage of knowledge, the wheel, etc. You will never encounter a complex number in the real world, yet they are essential for how we model the world, and without them there would be no quantum mechanics. They also play an important role in algorithms like the FFT.

Linear algebra is instrumental to the study of linear operators. There are two formulations of quantum mechanics, the PDE approach due to Schrödinger and the matrix mechanics approach due to Heisenberg. We will follow the latter approach which is heavily based in linear operators. We are going to try to learn about quantum computing algorithms without learning quantum mechanics. To build quantum computers, however, you will have to become one with quantum mechanics.

### 7.1 Complex Numbers

To solve  $x^2 + 1 = 0$ , we invented  $i = \sqrt{-1}$ , which is basically a placeholder for the solution of this equation. And hence the entire field of complex analysis developed around this simple concept. It is a miracle that this “imaginary” concept has such an influence on all things real. Mathematically, the complex numbers are algebraically closed. This is the content of the fundamental theorem of algebra. Specifically, every polynomial with complex coefficients (for example  $x^2 + 1 = 0$ ) has at least one complex solution.

A complex number with real part  $a$  and imaginary part  $b$  is written

$$x = a + ib. \tag{70}$$

Complex numbers can be added, subtracted, multiplied and divided using the usual rules of real-algebra and the fact that  $i^2 = -1$ . A complex number can be represented as a point on the Cartesian plane with coordinates  $(a, b)$ ,

$$[\text{Draw picture, with argand representation.}] \tag{71}$$

In polar coordinates, the representation is  $(r, \theta)$ , where  $a = r \cos \theta$  and  $b = r \sin \theta$ , and so

$$\begin{aligned} r^2 &= a^2 + b^2 \\ \tan \theta &= b/a. \end{aligned} \tag{72}$$

The polar representation is not unique and  $(r, \theta + 2k\pi)$  for  $k \in \mathbb{Z}$  is the same complex number. The spectacular theorem of Euler is a power-tool,

$$e^{i\theta} = \cos \theta + i \sin \theta. \tag{73}$$

This establishes the link between calculus and the number  $e$ , geometry and the ratios of sides of triangles and the complex numbers. This is a truly remarkable relation. Since  $(e^{i\theta})^n = e^{in\theta}$ , we immediately get Demoiivre's relation,

$$(\cos x + i \sin x)^n = e^{inx} = \cos nx + i \sin nx. \tag{74}$$



Euler's formula allows us to multiply and take powers of complex numbers effortlessly:

$$\begin{aligned} (r, \theta)^n &= (r^n, n\theta) \\ (r_1, \theta_1) \times (r_2, \theta_2)^n &= (r_1 r_2, \theta_1 + \theta_2) \\ (r, \theta)^{1/n} &= (r^{1/n}, (\theta + 2k\pi)/n). \end{aligned} \tag{75}$$

The complex conjugate of a complex number is obtained by negating the imaginary part.

$$x = a + ib \rightarrow x^* = a - ib. \tag{76}$$

In the polar representation,  $x = (r, \theta) \rightarrow x^* = (r, -\theta)$ .

## 7.2 Complex Vector Spaces

The complex vector space  $\mathbb{C}^n$  of dimension  $n$  consists of all vectors with  $n$  components, where the components are complex numbers. Here is an example of a complex vector and a complex matrix.

$$x = \begin{bmatrix} 1 - i \\ 2i \\ -1 \end{bmatrix} \quad A = \begin{bmatrix} 1 - i & 1 + i & 3 \\ 2i & -1 & 1 + 2i \end{bmatrix}. \tag{77}$$

$A$  is an operator from  $\mathbb{C}^3 \mapsto \mathbb{C}^2$ . We compute the matrix-vector product  $Ax$  in the usual way. We define the transpose in the usual way. We can also take the complex conjugate of a matrix by taking the complex conjugate of every component. An important operation is to take the transpose of the complex conjugate, called the dagger,

$$x^\dagger = [1 + i \quad -2i \quad -1] \quad A^\dagger = \begin{bmatrix} 1 + i & -2i \\ 1 - i & -1 \\ 3 & 1 - 2i \end{bmatrix}. \tag{78}$$

You should verify that

$$(A + B)^\dagger = A^\dagger + B^\dagger, \quad (AB)^\dagger = B^\dagger A^\dagger. \tag{79}$$

The inner product  $\langle x, y \rangle = x^T y$  from real vector spaces won't work in complex vector spaces. One reason is that the norm  $\|x\|^2 = \langle x, x \rangle$  should be non-negative. This is not true when  $x, y$  can be complex. Instead, we use the dagger and define the inner product

$$\langle x, y \rangle = x^\dagger y. \tag{80}$$

This definition implies linearity in the second argument,

$$\langle x, ay + bz \rangle = a\langle x, y \rangle + b\langle x, z \rangle. \tag{81}$$

What is  $\langle ax + bz, y \rangle$ ? Note, one could instead insist on linearity in the first argument, in which case  $\langle x, y \rangle = y^\dagger x$ . Note that the inner product is not symmetric. For operators  $A, B$ ,

$$\langle Ax, By \rangle = (Ax)^\dagger By = x^\dagger A^\dagger By. \tag{82}$$

An operator  $A$  (matrix) is hermitian if  $A^\dagger = A$ , sometimes called self-adjoint. The matrix is unitary if  $A^{-1} = A^\dagger$ , so

$$AA^\dagger = A^\dagger A = I. \tag{83}$$

A quantum computer is a quantum physical system. Unitary operators play an important role because the time evolution of a quantum system (quantum computer) is driven by a unitary operator, specifically  $e^{itH}$  where  $H$  is the Hamiltonian operator. These details won't be important for us. What will matter is that an algorithm on a quantum computer takes the state from an initial configuration to a final configuration in which we measure a result. Thus, any algorithm is performing a time-evolution of the initial state and hence must be a unitary operator. The general form of the  $2 \times 2$  unitary operator, up to a multiplicative phase is

$$U = \begin{bmatrix} r_1 & \sqrt{1-r_1^2}e^{i\varphi_1} \\ \sqrt{1-r_1^2}e^{i\varphi_2} & -r_1e^{i(\varphi_1+\varphi_2)} \end{bmatrix} \quad (84)$$

Hermitian operators will be important because every observable is related to a hermitian operator. The final result of an algorithm in a quantum computer must be a measurable observable, and hence must be related to some hermitian operator. The way it is related to the hermitian operator is through its eigenvectors and eigenvalues (eigen meaning "special"). For a matrix  $A$ , a nonzero vector  $v$  for which

$$Av = \lambda v \quad (85)$$

is called an eigenvector of  $A$  and  $\lambda$  is the associated eigenvalue.

### 7.3 Basis

A orthonormal basis in an  $n$ -dimensional complex vector space is a collection of  $n$  pairwise orthogonal unit vectors. More generally a basis is a largest set of linearly independent vectors. The standard basis consists of the column vectors in

$$E = [e_1, e_2, \dots, e_n] = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & \dots & 1 \end{bmatrix}. \quad (86)$$

The basis is represented by the square matrix  $E$ . The basis is orthonormal if  $E^\dagger E = I_n$ . That is if  $E$  is unitary. The components of a vector  $v$  in the basis  $E$ , given by  $[x_1, x_2, \dots, x_n]^T$  represent  $v$  as a linear combination of the basis vectors,

$$v = [e_1, e_2, \dots, e_n] \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}. \quad (87)$$

The components in one orthonormal basis can be transformed into another orthonormal basis. Let  $F$  be another orthonormal basis in which  $v$  has components  $[z_1, z_2, \dots, z_n]^T$ . Then

$$v = Fz \rightarrow z = F^\dagger v = F^\dagger E v. \quad (88)$$

The matrix  $F^\dagger E$  is a basis transformation matrix that transforms the components  $x$  in basis  $E$  to the components  $z$  in basis  $F$ .

## 7.4 Spectral Theorem

For any  $n$ -dimensional hermitian operator  $H$ , one can construct an orthonormal basis composed of its eigenvectors. This is the spectral theorem. That is, there is a unitary basis  $U$  for which

$$HU = U\Lambda, \quad (89)$$

where  $\Lambda$  is a diagonal matrix composed of eigenvalues of  $H$  along the diagonal. This is called the spectral theorem. Applying  $U^\dagger$  to both sides,

$$U^\dagger HU = \Lambda, \quad (90)$$

that is  $H$  can be diagonalized by a unitary matrix. The spectral theorem is a very powerful tool.

## 7.5 Hadamard Matrix

A particularly important quantum gate is the Hadamard gate, which is related to the Hadamard matrix. The unnormalized Hadamard matrix  $Q_n$  is defined by

$$Q_0 = [1]; \quad Q_{n+1} = \begin{bmatrix} Q_n & Q_n \\ Q_n & -Q_n \end{bmatrix}. \quad (91)$$

For example,

$$Q_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}; \quad Q_2 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}. \quad (92)$$

The normalized Hadamard matrix  $H_n$  simply normalizes the matrix so each column has unit norm,

$$H_n = 2^{-n/2} Q_n. \quad (93)$$

The Hadamard matrix has many useful properties.

1.  $H_n$  is a  $2^n \times 2^n$  matrix.
2.  $H_n$  is hermitian and unitary,  $H_n^\dagger H_n = I_{2^n}$ .
3.  $Q_n$  has only entries  $\pm 1$ .
4. The first row and column of  $Q_n$  are all 1's. The sum of every other column and row is 0.

## 8 Quantum Mechanics

We hope to learn about QC algorithms with minimal knowledge of quantum mechanics. However, if you want to build a quantum computer, well that will be a physical system governed by the laws of quantum mechanics. So you'll need to know some quantum mechanics. At least we will give you the postulates and how they are relevant to building a stable quantum computer.

In quantum mechanics, we use  $|\psi\rangle$  to represent the state of the system you are interested in, for example the state of a quantum computer, which includes its input and output. The quantum computer is a black box that takes this state to a new state. In this new state, we measure some observable which corresponds to the result,

$$[\text{Draw setup with interaction to environment and measuring result.}] \tag{94}$$

Quantum mechanics postulates rules for:

- How the state evolves under the quantum computing algorithm as well as interactions with the environment.
- What happens when you measure the result.

We don't know why the rules are this way. We do know that these rules, together with a huge amount of mathematics, are remarkably accurate at describing the physical world, especially at tiny scales where the rules of classical mechanics fail. This is similar to how we do not know why Newton's law is  $F = ma$ . Why isn't it  $F = ma^2$ , for example? However, we do know that Newton's law combined with a lot of mathematical knowhow is remarkably accurate at describing what happens in the real world at large scales – the classical regime. How did we find the rules of QM. Similar to how Newton found the laws for classical motion. Guesswork, observation and then experimental verification.

### 8.1 Postulates of Quantum Mechanics

The state of a system is specified by a vector  $|\psi\rangle$  in a complex vector space with the standard inner product. Physicists use the bra-ket notation to denote inner product. So the inner product of  $|\psi\rangle$  and  $|\phi\rangle$  is written  $\langle\phi|\psi\rangle$  where

$$\langle\phi|\psi\rangle = (|\phi\rangle)^\dagger|\psi\rangle. \tag{95}$$

We assume the state is normalized to 1, which is required by the probabilistic way in which measurements are done, so

$$\langle\psi|\psi\rangle = 1. \tag{96}$$

The system's state evolves with time, so it starts at  $|\psi(0)\rangle$  and evolves to  $|\psi(t)\rangle$  over time  $t$ . This evolution is driven by the energy operator, or Hamiltonian operator  $\hat{H}$ .<sup>1</sup> Specifically,

$$|\psi(t)\rangle = e^{-i\hat{H}t}|\psi(0)\rangle. \tag{97}$$

(There is a constant  $\hbar$ , the reduced plank constant, which sets the units for energy. We set it to 1 for simplicity of presentation.) Just understanding this formula is non-trivial. What is the exponent of an operator? This is defined using the Taylor series for the exponential. It is a non-trivial exercise

---

<sup>1</sup>We sometimes use the hat notation,  $\hat{A}$ , to emphasize something is an operator (matrix).

for the reader to show  $e^{-i\hat{H}t}$  is a unitary operator whenever  $\hat{H}$  is hermitian, which it is. This means that time-evolution preserves geometry, in particular the norm of the state stays 1,

$$\langle \psi(t) | \psi(t) \rangle = (e^{-i\hat{H}t} |\psi(0)\rangle)^\dagger e^{-i\hat{H}t} |\psi(0)\rangle \quad (98)$$

$$= (|\psi(0)\rangle)^\dagger (e^{-i\hat{H}t})^\dagger e^{-i\hat{H}t} |\psi(0)\rangle \quad (99)$$

$$= (|\psi(0)\rangle)^\dagger |\psi(0)\rangle \quad (100)$$

$$= \langle \psi(0) | \psi(0) \rangle \quad (101)$$

$$= 1. \quad (102)$$

An observable is associated to a hermitian operator. There are operators for position, momentum, energy, spin, etc. Let us consider the observable  $A$ , associated to the operator  $\hat{A}$ . Since  $\hat{A}$  is hermitian it has an orthonormal eigenbasis for the state space,  $|\phi_1\rangle, \dots, |\phi_n\rangle$  (assuming the state-space is  $n$ -dimensional), and the eigenvector  $|\phi_i\rangle$  has an associated eigenvalue  $\lambda_i$ . This means we can get the “coordinates” of  $|\psi\rangle$  in this eigenbasis,

$$|\psi\rangle = \sum_{i=1}^n a_i |\phi_i\rangle. \quad (103)$$

The  $a_i$  can be complex. In state  $|\psi\rangle$ , when you measure the observable  $A$ , the result will be one of the eigenvalues  $\lambda_i$  of the associated operator  $\hat{A}$ . Which eigenvalue you observe is random. The probability to observe  $\lambda_i$  is

$$\text{probability to observe } \lambda_i = \|a_i\|^2 = a_i^* a_i. \quad (104)$$

These probabilities are determined from the expansion of  $|\psi\rangle$  in the eigenbasis  $\hat{A}$ . We can also compute the expected value of the observable,

$$\mathbb{E}[A] = \sum_i \lambda_i \|a_i\|^2 \quad (105)$$

We can get a convenient expression for this expected value as follows. Note that

$$\hat{A}|\psi\rangle = \sum_{i=1}^n a_i \lambda_i |\phi_i\rangle. \quad (106)$$

Computing  $\langle |\psi\rangle, \hat{A}|\psi\rangle \rangle$  gives

$$\langle |\psi\rangle, \hat{A}|\psi\rangle \rangle = \sum_{j=1}^n a_j^* |\phi_j\rangle^\dagger \sum_{i=1}^n a_i \lambda_i |\phi_i\rangle \quad (107)$$

$$= \sum_{j=1}^n \sum_{i=1}^n \lambda_i a_j^* a_i |\phi_j\rangle^\dagger |\phi_i\rangle. \quad (108)$$

Since  $|\phi_j\rangle^\dagger |\phi_i\rangle = \delta_{ij}$  because the eigenbasis is orthonormal, we have that

$$\langle |\psi\rangle, \hat{A}|\psi\rangle \rangle = \sum_{j=1}^n \sum_{i=1}^n \lambda_i a_j^* a_i \delta_{ij} \quad (109)$$

$$= \sum_{j=1}^n \lambda_j a_j^* a_j = \mathbb{E}[A]. \quad (110)$$

To avoid cumbersome notation, we write the inner product  $\langle |\psi\rangle, \hat{A}|\psi\rangle \rangle$  as  $\langle \psi | \hat{A} | \psi \rangle$ .

Another weird thing happens. The state changes upon measurement. If the measurement is  $\lambda_i$ , then the state collapses to the corresponding eigenvector,  $|\psi\rangle \rightarrow |\phi_i\rangle$ . When the state is one of these eigenvectors, we call it a pure state, otherwise it is a mixed state. State collapse is always to a pure state. This state collapse is perhaps the most counter-intuitive aspect of quantum mechanics. It is also counterintuitive that the properties of the state, when measured are random. Let us summarize.

1. [State] The state  $|\psi\rangle$  is a complete representation of a system. It is normalized, so  $\langle \psi | \psi \rangle = 1$ .
2. [Time Evolution] The state evolves according to the Hamiltonian operator  $|\psi(t)\rangle = e^{-i\hat{H}t}|\psi(0)\rangle$ .
3. [Observables] An observable  $A$  corresponds to an operator  $\hat{A}$ . These operators for the standard observables are specified in the postulates. We did not give them here.
4. [Measurement] When you measure an observable  $A$  with corresponding operator  $\hat{A}$ , the possible measurements are the eigenvalues of  $\hat{A}$ . If  $\hat{A}$  has orthonormal eigenbasis  $|\phi_i\rangle$  with corresponding eigenvalues  $\lambda_i$ , then we can expand  $|\psi\rangle$  in this basis,

$$|\psi\rangle = \sum_{i=1}^n a_i |\phi_i\rangle. \tag{111}$$

The eigenvalue  $\lambda_i$  is measured with probability  $\|a_i\|^2$  and  $\mathbb{E}[A] = \langle \psi | \hat{A} | \psi \rangle$ .

5. [State Collapse Upon Measurement] If the measured value is  $\lambda_i$ , then the state collapses to the corresponding eigenvector of  $\hat{A}$ ,  $|\psi\rangle \rightarrow |\phi_i\rangle$ .
6. There are some other postulates regarding different types of particles, e.g. the state is anti-symmetric with respect fermion-exchange. We won't go into those details.

How do these postulates of quantum mechanics affect us when we are trying to build a quantum computer. Let us go back to our workflow figure,

$$\text{[Draw the Workflow again.]} \tag{112}$$

When we create the initial state of the quantum computer, we have to accept that it is a quantum state. The algorithm evolves this state into a final state and we extract the result of the computation by measuring some property of the final state. We assume the result is the output of our quantum algorithm - some unitary operator that evolves the state. Unfortunately, our computer interacts with the universe, so the actual picture looks more like

$$\text{[Draw the Workflow again with Universe Interaction.]} \tag{113}$$

The universe interaction contaminates our algorithm by also driving some of the state evolution. We can try to isolate the quantum computer from the universe, but, because we have to ultimately make the measurement, there will always be some evolution of the final state that is not due to the quantum algorithm. Hence the result we measure will be contaminated. We need ways to error correct.

Then there is the measured result being nondeterministic. This won't do. If our quantum algorithm is supposed to tell if a number is prime, the answer must be a deterministic yes or no.

Hence, we have to ensure that the final state is a pure state, so the result is definitive. This impacts the design of quantum-algorithms as well as the measurement process, which should keep the state pure. Lastly, from the practical perspective one has to find a quantum system that can store the quantum equivalent of a bit, keep it stable, allow for the robust evolution of that state under an algorithm and reliable measurement of the state to obtain the final result. These are all non-trivial requirements, which largely make quantum computation a theoretical construct at the moment. The path to a viable quantum computer is more like a marathon, not a sprint. So patience and endurance are necessary. The first step is to develop the seeds, the theory of quantum algorithms.

## 8.2 Spin

Some particles have a property called spin. For example, spin-up or spin-down. Such particles with two possible spins are very convenient because the spin can encode a bit. Spin-up could be a 1 and spin-down a 0. Unfortunately, spin depends on the axis of rotation. So when we measure spin, we do so with respect to an axis. We define operators corresponding to the three common axes  $x, y, z$ . These are the three spin operators (in units with  $\hbar = 1$ ),

$$\hat{S}_x = \frac{1}{2} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \hat{S}_y = \frac{1}{2} \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad \hat{S}_z = \frac{1}{2} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \quad (114)$$

Let us consider the spin operator  $\hat{S}_z$ . When you measure spin about the  $z$ -axis, you produce a result  $\lambda \mathbf{e}_3$ , where  $\lambda$  is the magnitude or amount of spin, and  $\mathbf{e}_3 = [0, 0, 1]^T$  is the basis vector in the  $z$ -direction. The  $\mathbf{e}_3$  just tells you that it is the  $z$ -spin that you have measured. So you measured a spin of  $\lambda$  about the  $z$ -axis, i.e. the spin is  $\lambda \mathbf{e}_3$ . What are the possible values you can get for  $\lambda$ ? We go back to the postulates to figure that out. We need to find the eigenvalues of  $\hat{S}_z$ . It is an easy exercise to find the eigenvalue, eigenvector pairs

$$\lambda_z^+ = \frac{1}{2}, |\phi_z^+\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \lambda_z^- = -\frac{1}{2}, |\phi_z^-\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \quad (115)$$

So, the possible measurements for the  $z$ -spin are  $\pm \frac{1}{2} \mathbf{e}_3$ . Similarly, you can compute the eigenvalue-eigenvector pairs for  $\hat{S}_x$  and  $\hat{S}_y$ ,

$$\lambda_x^+ = \frac{1}{2}, |\phi_x^+\rangle = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}, \quad \lambda_x^- = -\frac{1}{2}, |\phi_x^-\rangle = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}; \quad (116)$$

$$\lambda_y^+ = \frac{1}{2}, |\phi_y^+\rangle = \begin{bmatrix} 1/\sqrt{2} \\ i/\sqrt{2} \end{bmatrix}, \quad \lambda_y^- = -\frac{1}{2}, |\phi_y^-\rangle = \begin{bmatrix} 1/\sqrt{2} \\ -i/\sqrt{2} \end{bmatrix}. \quad (117)$$

The possible measurements for the  $x$ -spin are  $\pm \frac{1}{2} \mathbf{e}_1$  and for the  $y$ -spin are  $\pm \frac{1}{2} \mathbf{e}_2$ , where  $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$  are the standard basis in 3 dimensions. In practice, one can measure the spin in any direction defined by a unit vector  $\mathbf{v} = [v_x, v_y, v_z]$ . Such a vector can be represented by its two polar coordinates  $\varphi, \vartheta$ ,

$$\mathbf{v} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} \sin \vartheta \cos \varphi \\ \sin \vartheta \sin \varphi \\ \cos \vartheta \end{bmatrix}. \quad (118)$$

(Since we only have a limited set of notation, please distinguish between the azimuthal angle  $\varphi$  and the eigenbasis-vector  $|\phi\rangle$ .) Abusing notation and defining the vector of operators  $\hat{\mathbf{S}} = [\hat{S}_x, \hat{S}_y, \hat{S}_z]^T$ , we define the spin-operator for the direction  $\mathbf{v}$  by

$$\hat{S}_{\mathbf{v}} = \mathbf{v} \cdot \hat{\mathbf{S}} = v_x \hat{S}_x + v_y \hat{S}_y + v_z \hat{S}_z = \frac{1}{2} \begin{bmatrix} \cos \vartheta & e^{-i\varphi} \sin \vartheta \\ e^{i\varphi} \sin \vartheta & -\cos \vartheta \end{bmatrix} \quad (119)$$

This definition of the spin-operator in the  $\mathbf{v}$ -direction might look arbitrary, but it is chosen by requiring consistency conditions. First, the eigenvalues are  $\lambda_{\mathbf{v}}^{\pm} = \pm \frac{1}{2}$  as required because those are the possible spin measurements. Let us consider the pure state defined by the eigenvector  $|\phi_{\mathbf{v}}^+\rangle$ , the positive spin state about the axis  $\mathbf{v}$ . If we measure the  $x$ -spin, the expected value should be  $v_x/2$ . Similarly, the expected  $y$ -spin and  $z$ -spin should be  $v_y/2$  and  $v_z/2$  respectively. That means the expected spin is  $+\frac{1}{2}$  in the  $\mathbf{v}$  direction. We want

$$\mathbb{E}[\hat{\mathbf{S}}] = \begin{bmatrix} \langle \phi_{\mathbf{v}}^+ | \hat{S}_x | \phi_{\mathbf{v}}^+ \rangle \\ \langle \phi_{\mathbf{v}}^+ | \hat{S}_y | \phi_{\mathbf{v}}^+ \rangle \\ \langle \phi_{\mathbf{v}}^+ | \hat{S}_z | \phi_{\mathbf{v}}^+ \rangle \end{bmatrix} = \begin{bmatrix} \sin \vartheta \cos \varphi \\ \sin \vartheta \sin \varphi \\ \cos \vartheta \end{bmatrix}. \quad (120)$$

Indeed, this is the case. Prove it. First find the spin-up eigenvector,

$$|\phi_{\mathbf{v}}^+\rangle = \begin{bmatrix} \cos \vartheta/2 \\ e^{i\varphi} \sin \vartheta/2 \end{bmatrix}. \quad (121)$$

Show that  $\langle \phi_{\mathbf{v}}^+ | \hat{S}_x | \phi_{\mathbf{v}}^+ \rangle = \sin \vartheta \cos \varphi$ . Similarly prove the other two components of (120).

Here is a useful exercise. A particle starts with  $x$ -spin up. You measure the  $z$ -spin and then measure the  $x$ -spin again. What are the probabilities for the four possible outcomes:

$$\begin{array}{cc} + & + \\ + & - \\ - & + \\ - & - \end{array} \quad (122)$$



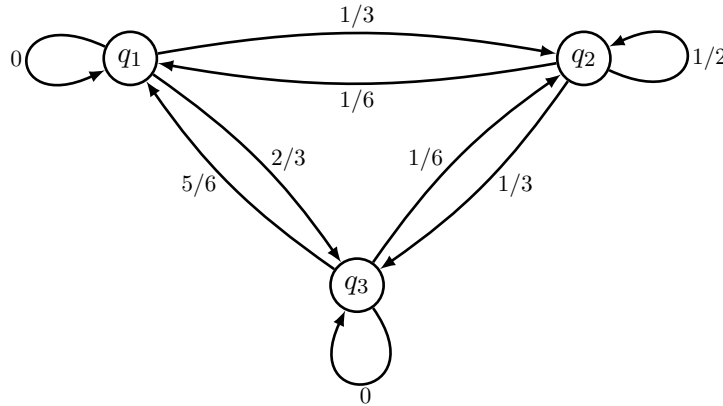
## 9 Dynamics

Why do we need to study dynamics. Let's look at a classical computer, the Turing machine. The TM starts off with the input written on the tape. The TM's state plus what is on the tape plus where its read-write head is can be viewed as the system's configuration. The TM-instructions (i.e., the algorithm) tell the TM what to do depending on what it reads on the tape. It can write something, transition states and move left or right, thus the system's configuration evolves to a new configuration. This dynamics is driven by the specifics of the algorithm. So, an algorithm dictates the dynamics of the TM's configuration. When the dynamics ends, we look at the final configuration to "measure" the answer delivered by the algorithm on the starting input.

None of this high-level discussion changes when we move to quantum algorithms. A quantum algorithm dictates the dynamics of the configuration of the quantum computer, starting from some initial configuration. When the dynamics ends, we measure the configuration of the quantum computer to determine our answer. Some of the details change in moving from the classical realm to the quantum realm, and these small changes make all the difference.

### 9.1 Classical Dynamics

A ball starting in vertex  $q_1$  transitions according to the following graph,



(123)

An arrow indicates a possible transition from vertex  $q_i$  to  $q_j$ , and the weight on an arrow is the probability of that transition. For example, after 1 transition, the probability is  $1/3$  to be in  $q_2$  and  $2/3$  to be in  $q_3$ . Collect these probabilities into a transition matrix  $T$ , where  $T_{ij}$  is the probability to transition from  $q_j$  to  $q_i$ . Column  $j$  in  $T$  are the probabilities to transition from  $q_j$ . We have

$$T = \begin{bmatrix} 0 & 1/6 & 5/6 \\ 1/3 & 1/3 & 1/6 \\ 2/3 & 1/3 & 0 \end{bmatrix}. \quad (124)$$

Notice that every column sums to 1 because from any vertex, the ball must transition to some vertex. Such a  $T$  is called a stochastic matrix, and the ball follows a Markov chain. Our matrix  $T$  also has rows which sum to 1 (called doubly stochastic). Such a Markov chain can be run backwards in time using  $T^T$ , that is the process is reversible. We can represent the state of the ball by the

probabilities it is in each vertex. The start state and the state after one transition are

$$|\psi_0\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad |\psi_1\rangle = \begin{bmatrix} 0 \\ 1/3 \\ 2/3 \end{bmatrix}. \quad (125)$$

You can verify that  $|\psi_1\rangle = T|\psi_0\rangle$ . In general, you can show that the state after  $k$  transitions is

$$|\psi_k\rangle = T^k|\psi_0\rangle. \quad (126)$$

To show this, use induction. You have to show that if the vertex probabilities at step  $k$  are given by  $|\psi_k\rangle$ , then the vertex probabilities at step  $k + 1$  are given by  $T|\psi_k\rangle$ . The system dynamics is captured by the transition matrix  $T$  and we say that  $T$  is the propagator for the system, propagating the state into the future.

## 9.2 Quantum Dynamics

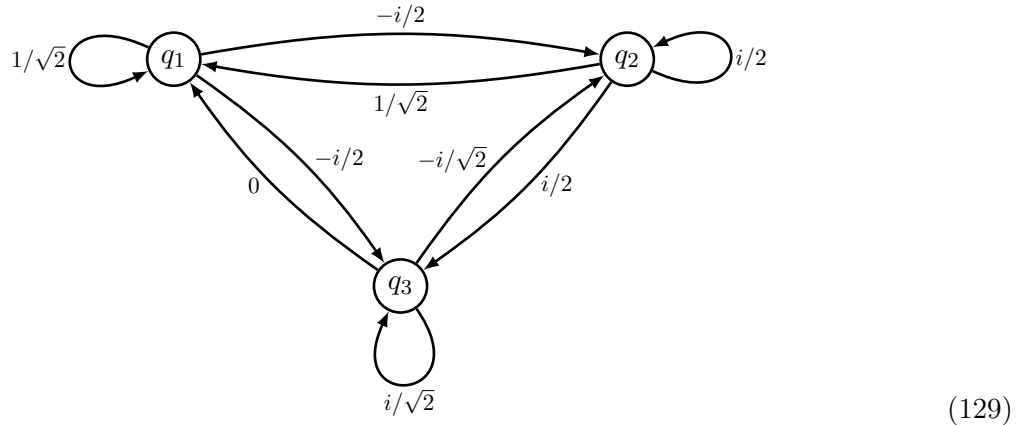
The move to quantum dynamics has two important aspects. The first is the state  $|\psi\rangle$  which represented the probabilities to be in each vertex is replaced by a state  $|\psi\rangle$  with *amplitudes* to be in each vertex. Here are some possible states.

$$|\psi_0\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad |\psi_0\rangle = \begin{bmatrix} i \\ 0 \\ 0 \end{bmatrix}, \quad |\psi_0\rangle = \begin{bmatrix} \sqrt{2/3} \\ 0 \\ -\sqrt{1/3} \end{bmatrix}. \quad (127)$$

The immediate signs that the amplitudes are not probabilities are that they can be negative or even complex numbers, and they do not sum to 1. The amplitudes are converted into probabilities by computing the norm squared of the amplitude. The probabilities corresponding to the above amplitudes are

$$P_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad P_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad P_0 = \begin{bmatrix} 2/3 \\ 0 \\ 1/3 \end{bmatrix}. \quad (128)$$

These are valid non-negative real probabilities summing to 1, that is  $\langle\psi_0|\psi_0\rangle = 1$ . Just like in classical dynamics, the amplitudes propagate forward in time using a propagator  $U$ . Unlike in the classical case where the propagator (probability transition matrix) is a real stochastic matrix, the propagator in a quantum dynamics can have complex entries. Consider the following dynamics,



These quantum dynamics can be summarized in the matrix

$$U = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ -i/2 & i/2 & -i/\sqrt{2} \\ -i/2 & i/2 & i/\sqrt{2} \end{bmatrix}. \quad (130)$$

The first column of  $U$ , for example, has the amplitudes to transition from vertex  $q_1$  to  $q_i$ . The state after one step of propagation is

$$|\psi_1\rangle = U|\psi_0\rangle. \quad (131)$$

For  $|\psi_1\rangle$  to be valid amplitudes, the norm squared of its entries must be valid probabilities and sum to 1. That is  $\langle\psi_1|\psi_1\rangle = 1$  for any  $|\psi_0\rangle$ . That is, for all  $|\psi_0\rangle$

$$\langle\psi_1|\psi_1\rangle = \langle\psi_0|U^\dagger U|\psi_0\rangle = \langle\psi_0|\psi_0\rangle = 1. \quad (132)$$

This is accomplished by requiring  $U$  to be unitary, as can be verified for our choice of  $U$ . In quantum mechanics, this is accomplished because the propagator in time is  $e^{-i\hat{H}t}$  which is unitary because  $\hat{H}$  is hermitian. The evolution of any quantum system like a quantum computer is driven by a unitary operator and hence preserves the normalization of a state. Let's see the evolution of our three example starting states under our unitary matrix  $U$ :

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{U} \begin{bmatrix} 1/\sqrt{2} \\ -i/2 \\ -i/2 \end{bmatrix}, \quad \begin{bmatrix} i \\ 0 \\ 0 \end{bmatrix} \xrightarrow{U} \begin{bmatrix} i/\sqrt{2} \\ 1/2 \\ 1/2 \end{bmatrix}, \quad \begin{bmatrix} \sqrt{2/3} \\ 0 \\ -\sqrt{1/3} \end{bmatrix} \xrightarrow{U} \begin{bmatrix} i/\sqrt{3} \\ 0 \\ -i\sqrt{2/3} \end{bmatrix}. \quad (133)$$

Something very interesting happens in our 3rd example above. This is one of the new phenomena that the quantum amplitudes bring. The final amplitude for  $q_2$  is 0. There are starting amplitudes for  $q_1$  and  $q_3$ . Each of  $q_1$  and  $q_2$  have an amplitude to transition to  $q_2$  *individually*. Together, these amplitudes cancel to give 0. This cannot happen in the classical setting. If  $q_1, q_3$  have probabilities to transition to  $q_2$  individually, then together those probabilities add, and cannot possibly give 0. This cancellation happens in the quantum system because we are adding amplitudes which are complex numbers. This phenomenon where non-zero amplitudes add to give 0 is called interference. This is a long known phenomenon in wave propagation, so in some sense quantum systems have wave-like properties.

You can now square the final amplitudes to get the new probabilities to be in each vertex for each of our examples above. Notice, in the classical setting the probabilities directly evolve into probabilities under action by  $T$ . In the quantum setting it is not possible to find a linear operator to evolve probabilities to probabilities. In the background you have the quantum state which is the amplitudes. In the background these amplitudes evolve to new amplitudes under the action of the linear operator  $U$ . Once the evolution of amplitudes is done, we can get the probabilities of being each vertex. If we measure the state, we will get the probabilities to be in each vertex.

**Classical** We can calculate the new probabilities from previous probabilities. We can actually see the ball as it makes its transitions from one vertex to another governed by these probabilities.

**Quantum** We do not see amplitudes. We surrender to a two-step process where the background amplitudes evolve. We do not see where the particle is through this evolution. In the end, we can measure the particle and the amplitudes (after taking their squared-norms) tells us the probabilities to observe the particle in a vertex. After we take the measurement, the amplitudes collapse to a pure state.

### 9.3 Ensembles of Independent Particles

What happens if we have two particles? Consider two independent balls following the quantum dynamics in the previous section. Label the particles' starting amplitudes as  $\mathbf{a}$  and  $\mathbf{b}$ ,

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \xrightarrow{U} U\mathbf{a}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \xrightarrow{U} U\mathbf{b}. \quad (134)$$

The particles each evolve independently under  $U$ . First, how do we represent the state of the two-particle system. Since each particle can be in  $\{q_1, q_2, q_3\}$ , there are 9 possibilities for the pair,

$$\{q_1q_1, q_1q_2, q_1q_3, q_2q_1, q_2q_2, q_2q_3, q_3q_1, q_3q_2, q_3q_3\}. \quad (135)$$

These are the so-called product states, obtained by taking the Cartesian product of the states available to  $\mathbf{a}$  and the states available to  $\mathbf{b}$ ,

$$\{q_1, q_2, q_3\} \times \{q_1, q_2, q_3\}. \quad (136)$$

The particles are independent, so

$$\mathbb{P}[\mathbf{a} \text{ is in } q_i \text{ and } \mathbf{b} \text{ is in } q_j] = \mathbb{P}[\mathbf{a} \text{ is in } q_i] \times \mathbb{P}[\mathbf{b} \text{ is in } q_j] = \|a_i\|^2 \|b_j\|^2. \quad (137)$$

But, since  $\|a_i b_j\|^2 = \|a_i\|^2 \|b_j\|^2$ , we obtain the correct probabilities if the amplitude of a possible outcome is the product of amplitudes. Therefore the state of the two-particle system is represented by the vector of amplitudes

$$\begin{array}{l} q_1q_1 \\ q_1q_2 \\ q_1q_3 \\ q_2q_1 \\ q_2q_2 \\ q_2q_3 \\ q_3q_1 \\ q_3q_2 \\ q_3q_3 \end{array} \begin{bmatrix} a_1b_1 \\ a_1b_2 \\ a_1b_3 \\ a_2b_1 \\ a_2b_2 \\ a_2b_3 \\ a_3b_1 \\ a_3b_2 \\ a_3b_3 \end{bmatrix} = \begin{bmatrix} a_1\mathbf{b} \\ a_2\mathbf{b} \\ a_3\mathbf{b} \end{bmatrix}. \quad (138)$$

The final vector on the right is the tensor product  $\mathbf{a} \otimes \mathbf{b}$  (depending on who you are talking to, it may also be called the Kronecker product),

$$\mathbf{a} \otimes \mathbf{b} = \begin{bmatrix} a_1\mathbf{b} \\ a_2\mathbf{b} \\ a_3\mathbf{b} \end{bmatrix}. \quad (139)$$

You take  $\mathbf{a}$  and multiply each component of  $\mathbf{a}$  by an *entire*  $\mathbf{b}$ . The amplitudes for a 2-particle system is the tensor product of the individual amplitudes,

$$|\psi_{\mathbf{a}\mathbf{b}}\rangle = |\psi_{\mathbf{a}}\rangle \otimes |\psi_{\mathbf{b}}\rangle. \quad (140)$$

This is the case for *independent* particles. Not every 9-dimensional vector of amplitudes is the tensor product of two 3-dimensional amplitudes. We encourage the energetic reader to construct such a

9-dim vector which is not a tensor product. If the state of the two particle system is represented by such a 9-dimensional vector of amplitudes, those two particles cannot be independent. We say that those particles are entangled, and the state is an entangled state. This construction generalizes. For three particles,  $\mathbf{a}$ ,  $\mathbf{b}$  and  $\mathbf{c}$ , the joint state is

$$\mathbf{a} \otimes \mathbf{b} \otimes \mathbf{c}. \quad (141)$$

We encourage the reader to show that if the individual amplitudes  $\mathbf{a}$ ,  $\mathbf{b}$ ,  $\mathbf{c}$  are normalized, then so is the tensor product  $\mathbf{a} \otimes \mathbf{b} \otimes \mathbf{c}$ .

What happens under independent evolution? The new states are  $U\mathbf{a}$  and  $U\mathbf{b}$ . So, from our prior discussion, the new joint state is  $U\mathbf{a} \otimes U\mathbf{b}$ . Let us first compute  $U\mathbf{a}$ ,

$$U\mathbf{a} = \begin{bmatrix} U_{11}a_1 + U_{12}a_2 + U_{13}a_3 \\ U_{21}a_1 + U_{22}a_2 + U_{23}a_3 \\ U_{31}a_1 + U_{32}a_2 + U_{33}a_3 \end{bmatrix}. \quad (142)$$

We can now compute  $U\mathbf{a} \otimes U\mathbf{b}$  by multiplying each component of  $U\mathbf{a}$  with an entire copy of  $U\mathbf{b}$ . Therefore, we have that

$$U\mathbf{a} \otimes U\mathbf{b} = \begin{bmatrix} (U_{11}a_1 + U_{12}a_2 + U_{13}a_3)U\mathbf{b} \\ (U_{21}a_1 + U_{22}a_2 + U_{23}a_3)U\mathbf{b} \\ (U_{31}a_1 + U_{32}a_2 + U_{33}a_3)U\mathbf{b} \end{bmatrix}. \quad (143)$$

Can we write this as the evolution of the 2-particle state  $\mathbf{a} \otimes \mathbf{b}$  under some operator  $V$ ,

$$U\mathbf{a} \otimes U\mathbf{b} = V(\mathbf{a} \otimes \mathbf{b}). \quad (144)$$

Indeed we can, and the question is what is  $V$ ? Note that

$$(U_{11}a_1 + U_{12}a_2 + U_{13}a_3)U\mathbf{b} = (U_{11}Ua_1\mathbf{b} + U_{12}Ua_2\mathbf{b} + U_{13}Ua_3\mathbf{b}) \quad (145)$$

$$= \begin{bmatrix} U_{11}U & U_{12}U & U_{13}U \end{bmatrix} \begin{bmatrix} a_1\mathbf{b} \\ a_2\mathbf{b} \\ a_3\mathbf{b} \end{bmatrix} \quad (146)$$

$$= \begin{bmatrix} U_{11}U & U_{12}U & U_{13}U \end{bmatrix} \mathbf{a} \otimes \mathbf{b}. \quad (147)$$

Similarly, we can write the other components in (143) to get

$$U\mathbf{a} \otimes U\mathbf{b} = \underbrace{\begin{bmatrix} U_{11}U & U_{12}U & U_{13}U \\ U_{21}U & U_{22}U & U_{23}U \\ U_{31}U & U_{32}U & U_{33}U \end{bmatrix}}_V \mathbf{a} \otimes \mathbf{b}. \quad (148)$$

We can now identify  $V$ . To obtain  $V$ , start with  $U$  and multiply each entry in  $U$  by an entire copy of the full matrix  $U$ . This is very similar to the tensor product of two vectors, and we can define the tensor product of two matrices  $A \otimes B$ : start with  $A$  and multiply every entry by an entire copy of  $B$  ( $A$  and  $B$  need not have the same dimensions),

$$A \otimes B = \begin{bmatrix} A_{11}B & A_{12}B & \cdots & A_{1m}B \\ A_{21}B & A_{22}B & \cdots & A_{2m}B \\ \vdots & & & \\ A_{d1}B & A_{d2}B & \cdots & A_{dm}B \end{bmatrix}. \quad (149)$$

If  $A$  is  $d \times m$  and  $B$  is  $\ell \times n$  then  $A \otimes B$  is  $d\ell \times mn$ . The tensor product of two vectors is a special case of the tensor product of two matrices.

Getting back to the evolution of  $k$  independent particles, if you have independent particles with amplitudes given by the states  $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_k$ , the joint state is the tensor product

$$\mathbf{a}_1 \otimes \mathbf{a}_2 \otimes \dots \otimes \mathbf{a}_k. \quad (150)$$

If the particles independently evolve according to their own unitary matrices  $U_1, \dots, U_k$ , then the evolution of the joint state is driven by the tensor product  $U_1 \otimes \dots \otimes U_k$ ,

$$\mathbf{a}_1 \otimes \mathbf{a}_2 \otimes \dots \otimes \mathbf{a}_k \rightarrow U_1 \mathbf{a}_1 \otimes U_2 \mathbf{a}_2 \otimes \dots \otimes U_k \mathbf{a}_k \quad (151)$$

$$= (U_1 \otimes \dots \otimes U_k)(\mathbf{a}_1 \otimes \mathbf{a}_2 \otimes \dots \otimes \mathbf{a}_k). \quad (152)$$

It is an exercise for the reader to show that the tensor product of unitary matrices is unitary. This implies that the normalized joint tensor product state remains normalized.<sup>2</sup> We also leave it to the reader to show that the tensor product is associative,

$$A \otimes (B \otimes C) = (A \otimes B) \otimes C. \quad (153)$$

Note, the tensor product is generally not commutative,  $A \otimes B \neq B \otimes A$  (find a counter example). The two matrices  $A \otimes B$  and  $B \otimes A$  do however have the same entries and they can be transformed into each other by a permutation of rows and columns.

---

<sup>2</sup>It implies more, that the norm of all vectors are preserved when acted upon by  $U_1 \otimes \dots \otimes U_k$ , not just vectors which are tensor products.

## 10 Classical Computing Using Linear Algebra

To move toward quantum computing, we need to find an appropriate way to generalize classical computing into the quantum domain. This will become seamless within a linear algebraic framework for two reasons. We have built considerable machinery in linear algebra, and we have already seen that quantum dynamics has a nice linear algebraic formulation based on hermitian operators for observables and unitary operators for evolution of state.

### 10.1 Classical Bits

The fundamental object is the bit which can take on two states,  $|0\rangle$  and  $|1\rangle$ . In the vector representation, the classical bits are the standard basis vectors in a 2-dimensional vector space,

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \quad (154)$$

This won't change when we move to the quantum realm. The only change will be that the classical bits are the standard basis vectors in a 2-dimensional complex vector space. What about a two-bit system, for example the first bit is 0 and the second is 1,  $|01\rangle$ ,

$$|01\rangle = |0\rangle \otimes |1\rangle = \begin{bmatrix} 1 & \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ 0 & \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}. \quad (155)$$

All the possible 2-bit states correspond to all the four standard basis vectors in 4-dimensions,

$$\begin{array}{cccc} |00\rangle & |01\rangle & |10\rangle & |11\rangle \\ \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \end{array}. \quad (156)$$

This generalizes to an  $n$ -bit system. The classical state has  $n$ -bits. There are  $2^n$  of these. In the vector representation, these classical states are the standard basis vectors in  $2^n$ -dimensions,

$$\begin{array}{cccccc} |00\dots 00\rangle & |00\dots 01\rangle & |00\dots 10\rangle & \dots & |11\dots 10\rangle & |11\dots 11\rangle \\ \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix} & \dots & \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \end{array}. \quad (157)$$

We can leverage the vector representation and allow any state  $|\psi\rangle$ ,

$$|\psi\rangle = \begin{bmatrix} p_0 \\ p_1 \end{bmatrix}. \quad (158)$$

Now,  $p_0$  is the probability the bit is  $|0\rangle$  and  $p_1$  the probability the bit is  $|1\rangle$ ,  $p_0 + p_1 = 1$ . For a two probabilistic bit system with  $|\psi\rangle = \begin{bmatrix} p_0 \\ p_1 \end{bmatrix}$  and  $|\phi\rangle = \begin{bmatrix} t_0 \\ t_1 \end{bmatrix}$  being independent bits,

$$|\psi\phi\rangle = |\psi\rangle \otimes |\phi\rangle = \begin{bmatrix} p_0 \begin{bmatrix} t_0 \\ t_1 \end{bmatrix} \\ p_1 \begin{bmatrix} t_0 \\ t_1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} p_0 t_0 \\ p_0 t_1 \\ p_1 t_0 \\ p_1 t_1 \end{bmatrix} = \begin{bmatrix} \mathbb{P}[|00\rangle] \\ \mathbb{P}[|01\rangle] \\ \mathbb{P}[|10\rangle] \\ \mathbb{P}[|11\rangle] \end{bmatrix}. \quad (159)$$

You never see a probabilistic bit. The probabilistic bit  $|\psi\rangle$  is just a representation of the state incorporating uncertainty about what the bit is. When you “measure” the state, you will always see either the bit  $|0\rangle$  or  $|1\rangle$ .

## 10.2 Quantum Bits

The leap from classical bits to quantum bits using the vector representation is trivial. We simply replace probabilities with amplitudes and the state of a qubit is now a complex vector,

$$|\psi\rangle = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}, \quad (160)$$

where  $a_0$  is the amplitude for  $|0\rangle$  and  $a_1$  is the amplitude for  $|1\rangle$ . You will never see a qubit. You can only “measure” a qubit, producing a bit, either  $|0\rangle$  or  $|1\rangle$ . The probability to measure  $|0\rangle$  is  $\|a_0\|^2$ , and the probability to measure  $|1\rangle$  is  $\|a_1\|^2$ , so  $\|a_0\|^2 + \|a_1\|^2 = 1$ . One convenient general representation for a qubit is

$$|\psi\rangle = \begin{bmatrix} \cos \theta \\ e^{i\phi} \sin \theta \end{bmatrix}. \quad (161)$$

For a two qubit system  $|\psi\rangle = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$  and  $|\phi\rangle = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}$ , the joint state is

$$|\psi\phi\rangle = |\psi\rangle \otimes |\phi\rangle = \begin{bmatrix} a_0 \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \\ a_1 \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_0 b_0 \\ a_0 b_1 \\ a_1 b_0 \\ a_1 b_1 \end{bmatrix}; \quad \begin{bmatrix} \mathbb{P}[|00\rangle] \\ \mathbb{P}[|01\rangle] \\ \mathbb{P}[|10\rangle] \\ \mathbb{P}[|11\rangle] \end{bmatrix} = \begin{bmatrix} \|a_0 b_0\|^2 \\ \|a_0 b_1\|^2 \\ \|a_1 b_0\|^2 \\ \|a_1 b_1\|^2 \end{bmatrix}; \quad (162)$$

Consider the joint state

$$\begin{bmatrix} a_0 b_0 \\ a_0 b_1 \\ a_1 b_0 \\ a_1 b_1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} \\ 0 \\ 0 \\ 1/\sqrt{2} \end{bmatrix}. \quad (163)$$

This is a valid set of amplitudes because their squared norms sum to 1. However, this state is not the tensor product of two single bit states. This is because  $a_0 b_1 = 0 \rightarrow a_0 = 0$  or  $b_1 = 0$ , and both cases are not possible: If  $a_0 = 0$  then  $a_0 b_0 \neq 1/\sqrt{2}$ ; if  $b_1 = 0$  then  $a_1 b_1 \neq 1/\sqrt{2}$ . So this joint state cannot be from two independent qubits. Such qubits are called entangled.

One natural question is how do we implement a qubit in practice? The requirements are we should be able to create it and keep it stable.<sup>3</sup> We should be able to evolve the qubit according to a

<sup>3</sup>Classical computing would be a disaster if bit were unstable and kept flipping at random.



“quantum algorithm”. We should be able to measure the final qubit without significantly perturbing the state. Some possibilities are:

- Electron orbits in small atoms. The ground-state orbit would be  $|0\rangle$  and the excited state would be  $|1\rangle$ .
- Photon polarization. Photons have a 2-dimensional polarization, left-right or up-down, which can be used to store the two states.
- Fundamental particle spin. For example the spin of an electron is either up or down in the direction of measurement.

The last word on implementing qubits is an ongoing research question.

### 10.3 Classical Computing Gates

Bits and qubits are both represented as vectors. Classical computing gates operate on input bits to produce output bits. We show how to represent classical gates as matrices so that their action on input bits can be implemented by multiplying the corresponding matrix with the input bits represented as a vector.

Let us begin with NOT,

$$\begin{aligned}
 |\psi\rangle &\xrightarrow{\text{NOT}} |\xi\rangle \\
 |0\rangle &\rightarrow |1\rangle & |1\rangle &\rightarrow |0\rangle \\
 \begin{bmatrix} 1 \\ 0 \end{bmatrix} &\rightarrow \begin{bmatrix} 0 \\ 1 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \end{bmatrix} &\rightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix}
 \end{aligned} \tag{164}$$

The action of  $\text{NOT}$  can be implemented by the matrix

$$\text{NOT} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{165}$$

This is verified by

$$\text{NOT} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{166}$$

The columns of the  $\text{NOT}$  are constructed directly from its action on the standard basis vectors, which are the classical bits.  $\text{NOT}$  can now be applied to any state  $|\psi\rangle = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$ ,

$$\text{NOT} \cdot |\psi\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} a_1 \\ a_0 \end{bmatrix}. \tag{167}$$

The action of  $\boxed{\text{NOT}}$  is simply to flip amplitudes. Let us now consider AND,

$$\begin{array}{c}
 |\psi\rangle \text{---} \\
 |\phi\rangle \text{---}
 \end{array}
 \begin{array}{|c|}
 \hline
 \text{AND} \\
 \hline
 \end{array}
 \longrightarrow |\xi\rangle$$

$$\begin{array}{cccc}
 |00\rangle \rightarrow |0\rangle & |01\rangle \rightarrow |0\rangle & |10\rangle \rightarrow |0\rangle & |11\rangle \rightarrow |1\rangle \\
 \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 1 \end{bmatrix}
 \end{array} \tag{168}$$

The action of  $\boxed{\text{AND}}$  can be implemented by the matrix

$$\boxed{\text{AND}} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{169}$$

This is verified by

$$\boxed{\text{AND}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{170}$$

The columns of the  $\boxed{\text{AND}}$  are constructed directly from its action on the standard basis vectors (in 4-dimensions), which are the classical bits.  $\boxed{\text{AND}}$  can now be applied to an arbitrary 2-bit state,

$$\boxed{\text{AND}} \cdot |\psi\rangle = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} a_0 + a_1 + a_2 \\ a_3 \end{bmatrix}. \tag{171}$$

We leave it for the reader to show

$$\boxed{\text{OR}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}, \quad \boxed{\text{NAND}} = \boxed{\text{NOT}} \cdot \boxed{\text{AND}} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}. \tag{172}$$

## 10.4 Circuits

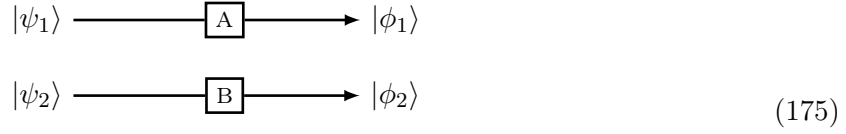
We can build circuits by combining gates sequentially (in series) or combining gates in parallel. First let us consider sequential,

$$|\psi\rangle \text{---} \boxed{\text{A}} \text{---} \boxed{\text{B}} \longrightarrow |\xi\rangle \tag{173}$$

The linear operator for this circuit is obtained from the individual linear operators by multiplication,

$$|\phi\rangle = BA|\psi\rangle. \tag{174}$$

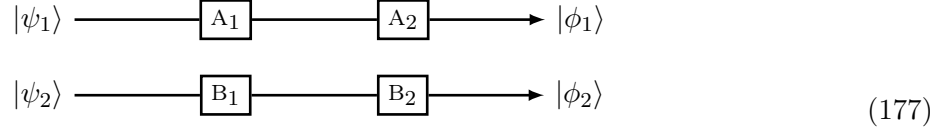
For the parallel case,



The linear operator for this circuit is obtained by using the tensor product,

$$|\psi_1\rangle \otimes |\psi_2\rangle \rightarrow A|\psi_1\rangle \otimes B|\psi_2\rangle = (A \otimes B)(|\psi_1\rangle \otimes |\psi_2\rangle). \quad (176)$$

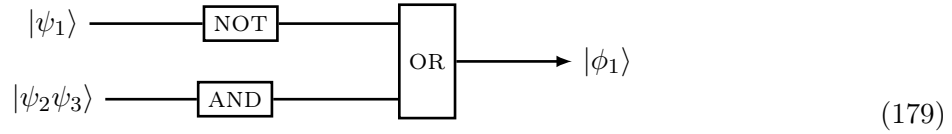
We can combine sequential and parallel,



The linear operator for this circuit is a tensor product of products,

$$|\psi_1\rangle \otimes |\psi_2\rangle \rightarrow A_2 A_1 |\psi_1\rangle \otimes B_2 B_1 |\psi_2\rangle = (A_2 A_1 \otimes B_2 B_1)(|\psi_1\rangle \otimes |\psi_2\rangle). \quad (178)$$

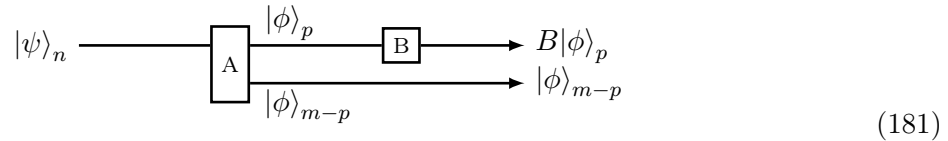
Let us do an example. Consider the circuit,



The linear operator for this circuit is  $\boxed{\text{OR}} \cdot (\boxed{\text{NOT}} \otimes \boxed{\text{AND}})$ , and the reader can verify that

$$\boxed{\text{OR}} \cdot (\boxed{\text{NOT}} \otimes \boxed{\text{AND}}) = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (180)$$

One last example. Often we apply a circuit, take some of the bits into one circuit and leave the others alone. Suppose the input is  $n$ -bits, and the first circuit outputs  $m$  bits. We take the first  $p$  of those  $m$  bits into another circuit and leave the others alone.



We can analyze this circuit as follows. The output of  $A$  is  $|\phi\rangle_p \otimes |\phi\rangle_{m-p} = A|\psi\rangle_n$ . Then we operate in parallel on  $|\phi\rangle_p$  and  $|\phi\rangle_{m-p}$ ,

$$|\phi\rangle_p \otimes |\phi\rangle_{m-p} \rightarrow B|\phi\rangle_p \otimes |\phi\rangle_{m-p} = B|\phi\rangle_p \otimes I_{2^{m-p}}|\phi\rangle_{m-p} = (B \otimes I_{2^{m-p}}) \underbrace{(|\phi\rangle_p \otimes |\phi\rangle_{m-p})}_{A|\psi\rangle_n} \quad (182)$$

That is, parallel to  $B$ , we are implementing the identity. We have,

$$|\phi\rangle_p \otimes |\phi\rangle_{m-p} \rightarrow (B \otimes I_{2^{m-p}}) \cdot A \cdot |\psi\rangle_n. \quad (183)$$

This circuit is implemented by the linear operator  $(B \otimes I_{2^{m-p}}) \cdot A$ .

## 11 Reversible Gates and Quantum Gates

We saw some classical gates. We will now take the bridge over to quantum gates and prove our first little result relating to quantum computing, namely the **no cloning** theorem. We begin with reversible gates.

### 11.1 Reversible Gates

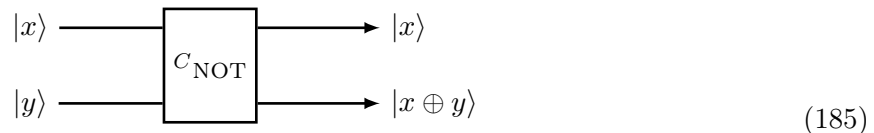
NOT is reversible, from the output you can determine the input. AND is not reversible. From the output, you cannot always determine the input. If the output is  $|0\rangle$ , you do not know if the input was  $|00\rangle$ ,  $|01\rangle$  or  $|10\rangle$ . Information is lost. Erasing information dissipates energy in the form of heat. This is based on statistical thermodynamic considerations and the 2nd law of thermodynamics. Bennet had the idea that if one could do computing with reversible processes, then energy is not lost – the ultimate in green computing. Landauer gave a lower bound on the minimum energy dissipation in erasing a bit. The debate is still ongoing whether this lower bound can be achieved or if reversible computation can be accomplished without energy input. Independent of this debate, it is still interesting to see if we can construct reversible classical gates, and further, this is a first step toward quantum gates which requires more than reversible. Quantum gates must also be unitary.

#### 11.1.1 Controlled-NOT

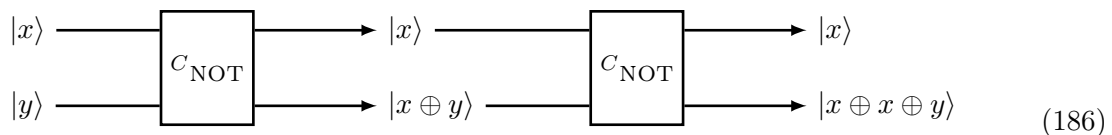
Let us consider XOR, which is not reversible,

$$\text{XOR} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}. \quad (184)$$

A useful trick to obtain reversible gates is to pass some of the input bits unchanged to the output so that the computation is reversible. Here is the idea in action with XOR,



We are computing the XOR, but passing through  $|x\rangle$  for the sole purpose of being able to reconstruct the input given the output. This gate is called controlled-NOT, even though we are computing XOR (we will see why soon). The name is not important. You should convince yourself that if you run the output through another  $C_{\text{NOT}}$  gate, you can recover the input,



The reason we have recovered the input is because  $x \oplus x \oplus y = 0 \oplus y = y$ . This gate is called the controlled not, because when  $|x\rangle = |1\rangle$ , the  $y$ -output gets negated, otherwise the  $y$ -output is unchanged. Hence whether  $y$  is negated is *controlled* by the value of  $x$ . What is the linear operator for  $C_{\text{NOT}}$ . The action of  $C_{\text{NOT}}$  on the standard basis vectors is

$$|00\rangle \rightarrow |00\rangle \quad |01\rangle \rightarrow |01\rangle \quad |10\rangle \rightarrow |11\rangle \quad |11\rangle \rightarrow |10\rangle, \quad (187)$$

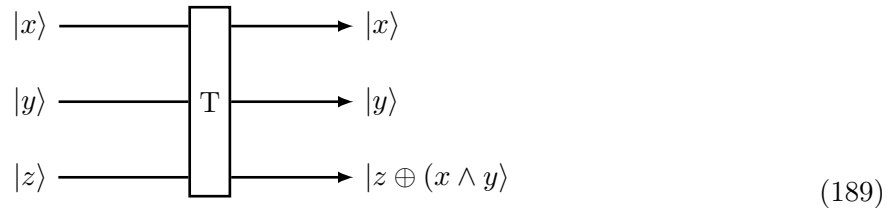
from which we get the operator,

$$C_{\text{NOT}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \quad (188)$$

You can verify that this matrix is its own inverse,  $C_{\text{NOT}}^2 = I_4$ . This strategy of using controlling bits is one we will reuse over and over.

### 11.1.2 Toffoli Gate

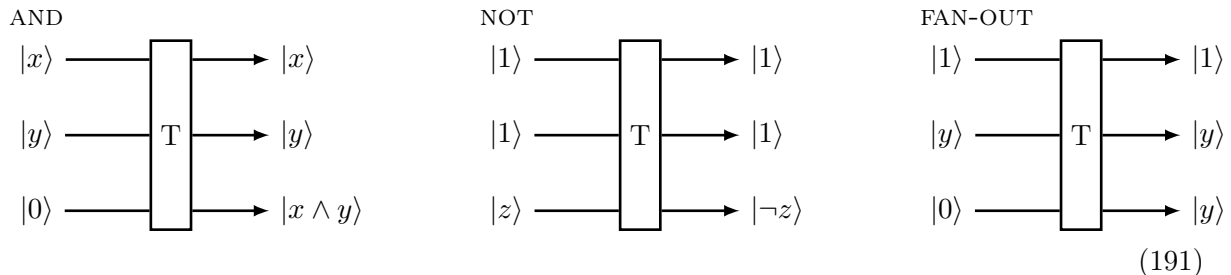
Another very interesting reversible gate is the Toffoli gate, which uses two controlling bits to compute the XOR,



By considering the action of  $T$  on the standard basis, you should be able to construct the linear operator for  $T$ ,

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \quad (190)$$

By setting the inputs appropriately, we can implement AND, NOT and FAN-OUT, as you can verify:



FAN-OUT is essentially a copy operation. Given these three operations, we can implement any Boolean circuit, since  $\{\text{AND}, \text{NOT}, \text{FAN-OUT}\}$  are a universal set of gates. In practice, AND and NOT alone are universal because we can implement FAN-OUT by just splitting the wire (assuming bits are implemented using voltage and currents). So this one Toffoli gate is universal. It is also worth observing that  $T$  is its own inverse,  $T^2 = I_8$ , and further that  $T$  is unitary. There are other universal gates with these properties, for example the Fredkin gate.

It is a useful exercise for the reader to construct the NAND operation using one Toffoli gate, and the OR operation using two Toffoli gates.

## 11.2 Quantum Gates

Quantum gates are unitary operators that operate on qubits. From the theoretical perspective, that's all there is to it. However, from the practical perspective one has to consider what unitary operators can be implemented efficiently by natural physical processes. So in classical computing, we settled on NAND because we can implement this gate very efficiently (space and time) using semiconductor transistors. If we had not invented semiconductor transistors, the face of computing in terms of what gates we use might look very different today.

So too with quantum gates. We will consider several quantum gates, and there are different sets of universal gates – unitary operators. We can build algorithms by combining these universal gates. But which algorithms will stand the test of time will depend on which quantum gates become efficiently implementable, if any. Quantum computing is a marathon, not a sprint.

Here are some example unitary operators for one qubit. Each is a valid quantum gate. Whether these gates can be implemented on a massive scale using natural physical processes will not be our concern. Among the operators we have already seen, the Hadamard, NOT,  $CNOT$  and Toffoli are quantum gates (unitary). Just as a reminder, the Hadamard gate is

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \quad (192)$$

Some other valid quantum gates are:

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad R(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix} \quad (193)$$

The operator  $R(\theta)$  simply shifts the phase of the amplitude for  $|1\rangle$ . The phase gate  $S = R(\pi/2)$  is a useful special case of the phase shift gate because it is one of the core gates in the Gottesman-Knill theorem. This phase shift-operator plays an important role in quantum gates because it is useful for building larger gates from smaller ones. A general unitary operator for one qubit, up to an overall phase, can be written

$$U = \begin{bmatrix} r & \sqrt{1-r^2}e^{i\phi_1} \\ \sqrt{1-r^2}e^{i\phi_2} & -re^{i(\phi_1+\phi_2)} \end{bmatrix} \quad (194)$$

There is one other operation in the quantum computing world that we don't have in the classical computing world, namely measurement. We can measure the qubit to produce a classical bit  $|0\rangle$  or  $|1\rangle$ . In such a case the qubit collapses to the classical bit measured. We will denote this measurement operation  $\boxed{?}$ ,

$$|\psi\rangle = \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} \xrightarrow{\quad} \boxed{?} \begin{cases} \text{prob } \|a_0\|^2 \rightarrow |0\rangle \\ \text{prob } \|a_1\|^2 \rightarrow |1\rangle \end{cases} \quad (195)$$

The measurement operation is non-linear, non-unitary and non-reversible. It returns a classical bit and also collapses the state to that classical bit.

### 11.2.1 Building Larger Gates

Any unitary operation on an  $n$ -qubit state is a valid quantum gate. But, we are not going to go off and build a new massive quantum gate for every quantum algorithm. Instead, we would build a small set of gates and combine those in various ways to get larger gates (unitary operators). This is similar to what we do with classical circuits. We have the core universal gates, for example  $\{\text{NAND}\}$ , and from those we can build circuits to implement arbitrary Boolean functions. In classical circuit theory the interesting question is what Boolean functions can be implemented with polynomially many basic gates. A similar issue faces quantum computing. What unitary operators can be implemented using a small set of core quantum gates, and can these unitary operators be classically simulated efficiently. Relevant to these issues are the Solovay-Kitaev theorem and the Gottesman-Knill theorem which we give at the for informational completeness.

Let us begin with what operations can we use to combine quantum gates (unitary operators). We can combine serially, which is the product. We can combine in parallel, which is the tensor product. There is one other operation which is important which we have seen before which is controlling an operator with a bit. So if  $U$  is a quantum gate, we define the controlled- $U$ , or  ${}^C U$  as the quantum gate which does nothing if the controlling bit is  $|0\rangle$  and implements  $U$  if the controlling bit is  $|1\rangle$ . The controlled gate is the quantum equivalent of the classical IF . . . THEN . . . controlling mechanism which we are all familiar with from classical programming. The circuit diagram below represents this operation,

$$\begin{array}{ccc} |x\rangle & \xrightarrow{\oplus} & |x\rangle \\ & \downarrow & \\ |y\rangle_n & \xrightarrow{U} & (\delta_{x0}I + \delta_{x1}U)|y\rangle \end{array} \quad (196)$$

If  $U = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ , then by considering the action of  ${}^C U$  on the standard basis  $|00\rangle, |01\rangle, |10\rangle, |11\rangle$  we find

$$|0\rangle \otimes |0\rangle \rightarrow |0\rangle \otimes |0\rangle |0\rangle \otimes |1\rangle \rightarrow |0\rangle \otimes |1\rangle |1\rangle \otimes |0\rangle \rightarrow |1\rangle \otimes (a|0\rangle + c|1\rangle) |1\rangle \otimes |1\rangle \rightarrow |1\rangle \otimes (b|0\rangle + d|1\rangle). \quad (197)$$

The reader can now verify that

$${}^C U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix}. \quad (198)$$

In general, adding the controlling bit doubles the size of the state and the dimensions of the operator and produces the operator

$${}^C U = \begin{bmatrix} I_{2^n} & 0_{2^n \times 2^n} \\ 0_{2^n \times 2^n} & U \end{bmatrix}. \quad (199)$$

Notice, that since  $U$  is unitary, so is  ${}^C U$ ,

$$({}^C U)^\dagger {}^C U = \begin{bmatrix} I_{2^n} & 0_{2^n \times 2^n} \\ 0_{2^n \times 2^n} & U^\dagger \end{bmatrix} \begin{bmatrix} I_{2^n} & 0_{2^n \times 2^n} \\ 0_{2^n \times 2^n} & U \end{bmatrix} = \begin{bmatrix} I_{2^n} & 0_{2^n \times 2^n} \\ 0_{2^n \times 2^n} & U^\dagger U \end{bmatrix} = I_{2^{n+1}}. \quad (200)$$

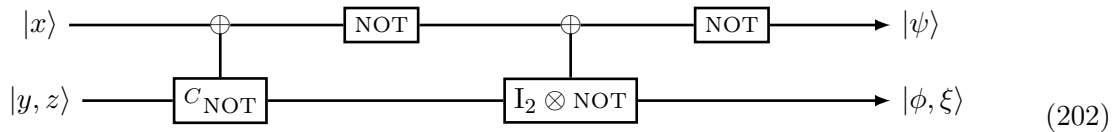
Here are some of the basic one and two qubit quantum gates

$$I_2, \quad I_4, \quad \text{NOT}, \quad {}^C \text{NOT}, \quad H, \quad R(\cos^{-1}(3/5)). \quad (201)$$

It turns out that these gates are universal, in that they can be combined to approximate any unitary operator within accuracy  $\epsilon$ . The number of gates needed is  $O(4^n \log(1/\epsilon))$ , which is essentially the content of the Solovay-Kitaev theorem. The Gottesman-Knill theorem says that if you combine only  $C_{\text{NOT}}, H, R(\pi/2)$ , which are generators of the Clifford algebra, then this quantum gate can be polynomially simulated on a classical computer.

### 11.2.2 Practice

To get some practice with quantum gates and building unitary operators from basic quantum gates, see if you can use combine the gates in (201) on page 45 to implement the unitary operator in (219) on page 50. Remember that a controlling bit can be used to operate on the other bits in one case and not in another. A controlling bit is the quantum equivalent of the IF ... THEN ... We suggest you try to interpret and analyze the following circuit, where  $x, y, z$  are bits.



You may review the mechanics of a controlling bit in the discussion before eq. (196) on page 45.

More generally, Let  $A$  and  $B$  be two unitary operators on  $n$  bits.  $A$  and  $B$  are  $2^n \times 2^n$  matrices. Show how to use a controlling bit to implemet the unitary operator

$$\begin{bmatrix} A & \mathbf{0}_{2^n \times 2^n} \\ \mathbf{0}_{2^n \times 2^n} & B \end{bmatrix} \tag{203}$$

### 11.3 No Cloning Theorem

Before the fancy quantum stuff starts, let us first see one limitation of quantum gates. In classical computing you can move or copy a file (bits). In quantum computing, you can only move, not copy. Star Trek got it right. When Kirk teleports down to a plannet the copy of Kirk on the Enterprise must disappear. In quantum computing, cut-and-paste is possible but copy-and-paste is not.

**Theorem 11.1** (No Cloning). There is no unitary operator that can copy a quantum state exactly.

Let’s prove this. In general, what would it mean to clone a qubit. Something like:

$$|x\rangle \otimes |0\rangle \xrightarrow{U} |x\rangle \otimes |x\rangle. \tag{204}$$

You have a joint state with two qubits. The first state is the qubit you would like to clone, and the second starts in some default state, say  $|0\rangle$ . Under the action of  $U$ , the second qubit becomes an exact copy of  $x$ . Let us consider an arbitrary state  $|x\rangle = c_0|0\rangle + c_1|1\rangle$ .

$$U((c_0|0\rangle + c_1|1\rangle) \otimes |0\rangle) = (c_0|0\rangle + c_1|1\rangle) \otimes (c_0|0\rangle + c_1|1\rangle) \tag{205}$$

$$= \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} \otimes \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} c_0^2 \\ c_0c_1 \\ c_1c_0 \\ c_1^2 \end{bmatrix}. \tag{206}$$



But,  $U$  is a linear operator and tensor product is also a linear operator, so

$$U((c_0|0\rangle + c_1|1\rangle) \otimes |0\rangle) = U(c_0|0\rangle \otimes |0\rangle + c_1|1\rangle \otimes |0\rangle) \quad (207)$$

$$= c_0|0\rangle \otimes c_0|0\rangle + c_1|1\rangle \otimes c_1|1\rangle \quad (208)$$

$$= \begin{bmatrix} c_0 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} c_0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ c_1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ c_1 \end{bmatrix} = \begin{bmatrix} c_0^2 \\ 0 \\ 0 \\ c_1^2 \end{bmatrix}. \quad (209)$$

The first step is because the tensor product is linear; the second step is because  $U$  is linear. We have a contradiction (two different expressions for the same quantity) unless one of  $c_0$  or  $c_1$  are zero, in which case we are cloning a classical bit. No linear operator can clone an arbitrary qubit, but you can clone classical bits (e.g., the Toffoli gate).

## 12 Unitary Operator for Classical Functions

The basic framework for a typical quantum algorithm is:

1. Initialize qubits into pure classical states.
2. Place the qubits into a superposition of states.
3. Run the quantum algorithm (unitary operator) on the superposition of states.
4. Measure the final qubits to get the answer.

In theory steps 2 and 3 can be represented by a single unitary operator. But, step 2 is often generic, whereas step 3 is the one which will depend intricately on the problem we are solving, hence we keep them separate. In a typical problem we have some classical Boolean function  $f$  and we ask some question of this function. The main speedup from quantum algorithms comes from step 2. By running the function on a superposition of classical input states, the algorithm somehow gets global information about  $f$  on all inputs. This is because of linearity. When  $f$  is applied to a superposition, you get the superposition of the outputs when  $f$  is individually applied to each classical state. So, we get information about  $f$  on all these classical states simultaneously. The task is then to unravel all this data to get the information we need to answer the question about  $f$ .

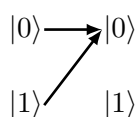
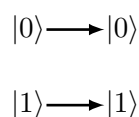
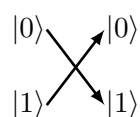
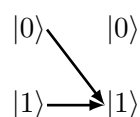
If you want a simple analogy from the physical world, go outside to take a look around. You have just one eye (okay two eyes), but play along, or just close one eye. This eye is receiving all the electromagnetic radiation bouncing off every object. All these EM rays are simultaneously hitting every retinal cell from every object. So, your eye is applying the “get-EM-radiation”-function simultaneously to every object in your environment. You then untangle all this data simultaneously arriving from every object to get specific information about specific objects, like the red bird is 2 meters away at my 2-o’clock.

We will motivate this with a simple, albeit contrived, example. Immediately, we will notice that an arbitrary Boolean function will rarely be invertible, let alone unitary. This won’t do in the quantum realm, because quantum gates are unitary. Our task will be to convert the Boolean function into a unitary operator first, and then ask the same question of this unitary operator.

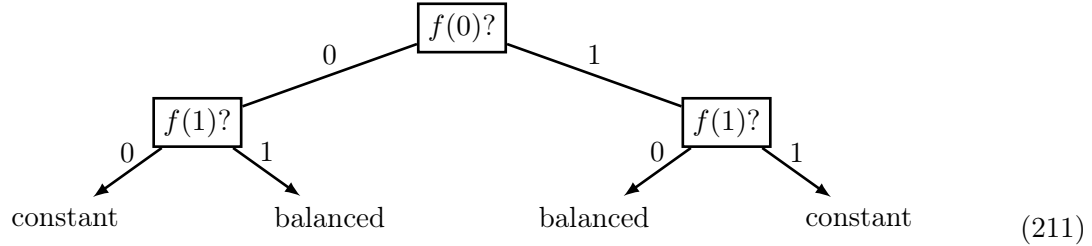
### 12.1 The Deutsch-Jozsa Problem

Let  $f$  be a Boolean function on  $n$  bits,  $f : \{0, 1\}^n \mapsto \{0, 1\}$ . The function is constant if  $f(x) = |0\rangle$  for all  $x \in \{0, 1\}^n$  or  $f(x) = |1\rangle$  for all  $x \in \{0, 1\}^n$ . The function is balanced if  $f(x) = |0\rangle$  for half of the  $x \in \{0, 1\}^n$  and  $f(x) = |1\rangle$  for the other half.

Of course there are all kinds of other functions in between balanced and constant. Note that there are  $2^{2^n}$  Boolean functions on  $n$  bits. Here are the 4 Boolean functions on 1 bit together with the linear operator for each function:

$f_{00}$  Constant $\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$	$f_{01}$  Balanced $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$f_{10}$  Balanced $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$f_{11}$  Constant $\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$	(210)
---	---	---	---	-------

The task is to determine if the function is balanced or constant, assuming it is one of the two. The function  $f$  is given as a black box linear operator. You cannot see inside the black-box. You can, however, apply the function to any input, and using as few function evaluations as possible, you must determine if  $f$  is constant or balanced. Here is an algorithm for the one bit case,  $n = 1$ .



The algorithm needs to use the black box for  $f$  two times. For the general  $n$  bit case, evaluate  $f$  on any  $1 + 2^{n-1}$  different  $n$ -bit inputs. If the function ever outputs two different values, it is balanced. Otherwise it is constant. This is exponentially many evaluations. A randomized algorithm is to choose  $k$  random  $n$ -bit inputs. If  $f$  is constant on these  $k$  inputs say constant, otherwise balanced. When you say balanced you will always be right. When you say constant, the chances you are wrong is  $1/2^{k-1}$ . The baby miracle of the day is we will see a quantum algorithm that only needs to use the black box for  $f$  once. Somehow, by applying  $f$  just once, it is able to get information about  $f$  on all inputs.

## 12.2 Converting Boolean Functions to Unitary Operators

The Deutsch-Josza problem is well defined, but it is not yet fit for the quantum realm because as we saw earlier, it may not be a unitary operator. Not to worry. It is always possible to represent any Boolean function as a unitary operator that effectively computes the function. To do this, we use a controlling bit. The general setup for a classical function taking an  $n$ -bit input is

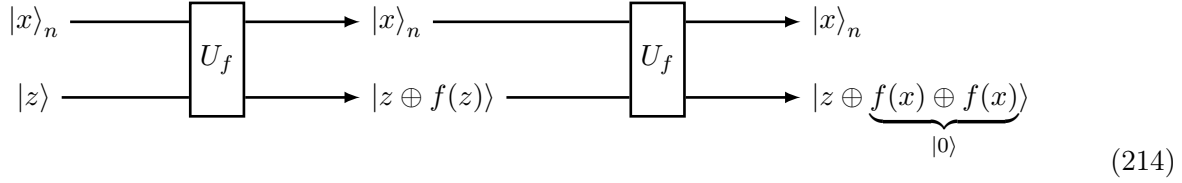
$$|x\rangle_n \longrightarrow \boxed{f} \longrightarrow |f(x)\rangle \quad (212)$$

For  $n > 1$ , such a function takes  $n$  bits to one bit and hence cannot be invertible, let alone unitary. We define a unitary implementation of  $f$  using a controlling bit  $z$ , similar to the way we defined  $C_{\text{NOT}}$ . This unitary implementation is  $U_f$ ,

$$\begin{array}{ccc} |x\rangle_n & \longrightarrow & \boxed{U_f} & \longrightarrow & |x\rangle_n \\ |z\rangle & \longrightarrow & & \longrightarrow & |z \oplus f(x)\rangle \end{array} \quad (213)$$

Let's examine all the ingredients in this construction. First, the number of input bits is  $n + 1$  and the same for the number of output bits. For  $U_f$  to be reversible, the output must contain information about  $|z\rangle$  and  $|x\rangle_n$ . Clearly  $|x\rangle_n$  can be reconstructed, and information about  $x$  is in the  $(n + 1)$ th bit. The function  $f$  is also accessible in  $U_f$  by simply setting  $|z\rangle = |0\rangle$ . The first  $n$  qubits are referred to as the input qubits or input registers. The bottom qubit is the output register, because that is where all the information about  $f(x)$  is stored. However, that is only the

case when a classical pure state is fed into  $U_f$ . As we will see, something strange happens when  $U_f$  operates on a superposition. First, let's show that  $U_f$  is invertible. Indeed,  $U_f$  is its own inverse,



The final output is  $|x\rangle_n \otimes |z\rangle$  because  $|z\rangle \oplus |0\rangle = |z\rangle$ . Let us now show that  $U_f$  is unitary no matter what the Boolean function  $f$  is ( $U_f$  is also real and hermitian). To see this, let us work with a concrete case. In general an  $n$ -bit Boolean function is a linear operator that maps  $\mathbb{C}^{2^n} \mapsto \mathbb{C}^{2^n}$ , and hence is a  $2^n \times 2^n$  matrix. Consider the two-bit Boolean function

$$|00\rangle \rightarrow |1\rangle, \quad |01\rangle \rightarrow |1\rangle, \quad |10\rangle \rightarrow |0\rangle, \quad |11\rangle \rightarrow |1\rangle,
 \tag{215}$$

From the action of  $f$  on the standard basis, we get the linear operator

$$f = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}.
 \tag{216}$$

The action of  $U_f$  is given by

$$|x, z\rangle \xrightarrow{U_f} |x, z \oplus f(x)\rangle.
 \tag{217}$$

The action of  $U_f$  on the standard basis vectors is given by

$ 000\rangle \rightarrow  001\rangle$	$ 001\rangle \rightarrow  000\rangle$	$ 010\rangle \rightarrow  011\rangle$	$ 011\rangle \rightarrow  010\rangle$	$ 100\rangle \rightarrow  100\rangle$	$ 101\rangle \rightarrow  101\rangle$	$ 110\rangle \rightarrow  111\rangle$	$ 111\rangle \rightarrow  110\rangle$
$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$

(218)

We can now immediately write down the matrix for  $U_f$ ,

$$U_f = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} \boxed{\text{NOT}} & \mathbf{0}_{2 \times 2} & \mathbf{0}_{2 \times 2} & \mathbf{0}_{2 \times 2} \\ \mathbf{0}_{2 \times 2} & \boxed{\text{NOT}} & \mathbf{0}_{2 \times 2} & \mathbf{0}_{2 \times 2} \\ \mathbf{0}_{2 \times 2} & \mathbf{0}_{2 \times 2} & \boxed{I_2} & \mathbf{0}_{2 \times 2} \\ \mathbf{0}_{2 \times 2} & \mathbf{0}_{2 \times 2} & \mathbf{0}_{2 \times 2} & \boxed{\text{NOT}} \end{bmatrix}.
 \tag{219}$$

Observe that  $U_f$  has a very nice structure. It is block-diagonal, composed of  $(2 \times 2)$ -blocks along the diagonal,  $2^n$  of them. Each block is either NOT or  $I_2$ . If the first  $n$  bits of the corresponding state map to 1, the block is NOT, otherwise the block is  $I_2$ . You should convince yourself that this block-diagonal structure is true in general for any Boolean function on  $n$ -bits. From this block-diagonal structure, you should be able to prove that  $U_f$  is real, hermitian and unitary.

**Summary.** In the classical world,  $f$  is given as a black-box linear operator. In the quantum world we construct a unitary operator  $U_f$  from  $f$  using a controlling bit  $z$ .  $U_f$  is a black box unitary operator implemented by some collection of quantum gates (physical processes). I cannot tell you what those physical processes are – that is, what will quantum hardware look like, but who cares. Lets build quantum algorithms based off unitary operators and hope that some day we can implement these algorithms on quantum hardware. If the quantum hardware (gates) are very different from what we assume here, you will we have the necessary machinery to adapt the your algorithms to whatever the quantum hardware be. Just as you learned to program in Pascal. It's no biggie to program those same algorithms in C++ or Python.

### 13 Testing Balance of 1-bit Functions

Recall the Deutsch-Jozsa problem for 1-bit Boolean functions. Here are the four possible functions,

$f_{00}$ $ 0\rangle \longrightarrow  0\rangle$ $ 1\rangle \longrightarrow  1\rangle$ Constant $\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$	$f_{01}$ $ 0\rangle \longrightarrow  0\rangle$ $ 1\rangle \longrightarrow  1\rangle$ Balanced $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$f_{10}$ $ 0\rangle \longrightarrow  1\rangle$ $ 1\rangle \longrightarrow  0\rangle$ Balanced $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$f_{11}$ $ 0\rangle \longrightarrow  0\rangle$ $ 1\rangle \longrightarrow  1\rangle$ Constant $\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$
--	--	--	--

(220)

A function  $f$  is given as a black box,

$$|x\rangle \longrightarrow \boxed{f} \longrightarrow |f(x)\rangle \quad (221)$$

The task is to determine if  $f$  is balanced or constant. A classical algorithm evaluates  $f$  twice, on the two possible inputs  $|0\rangle$  and  $|1\rangle$ . In the quantum realm, the function is given as a black box unitary operator  $U_f$ ,

$$\begin{array}{c} |x\rangle \\ |z\rangle \end{array} \longrightarrow \boxed{U_f} \longrightarrow \begin{array}{c} |x\rangle \\ |z \oplus f(x)\rangle \end{array} \quad (222)$$

The unitary operator  $U_f$  is a  $4 \times 4$  matrix. The idea behind a quantum algorithm is to evaluate  $U_f$  on a superposition of states. This gives global information about  $f$  on all possible inputs. The task is to then untangle this information and see if we can figure out whether  $f$  is balanced or constant. Take as an analogy the eye which gathers, in one shot, the electromagnetic signals from your entire environment, from all the objects. Then your brain unravels this superposition of EM-signals to conclude specific information about specific objects in specific locations.

Let's warmup with a simple computation,

$$\begin{array}{c} |0\rangle \\ |1\rangle \end{array} \longrightarrow \boxed{U_f} \longrightarrow \begin{array}{c} |0\rangle \\ |\overline{f(0)}\rangle \end{array} \quad (223)$$

Clearly, this computation gives only information about  $f(0)$ , if we measure the bottom bit.

#### 13.1 Applying $U_f$ to Superpositions

Let's now consider a superposition for the top bit,

$$\begin{array}{c} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ |1\rangle \end{array} \longrightarrow \boxed{U_f} \longrightarrow \begin{array}{c} ? \\ ? \end{array} \quad (224)$$

Let's work out what the output is. The input state is  $(|0\rangle + |1\rangle)/\sqrt{2} \otimes |1\rangle$ , and using linearity of the tensor product, the input is

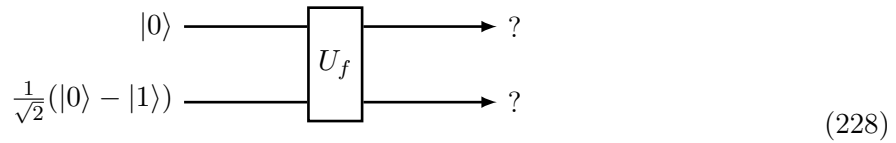
$$\frac{1}{\sqrt{2}}|0\rangle \otimes |1\rangle + \frac{1}{\sqrt{2}}|1\rangle \otimes |1\rangle = \frac{1}{\sqrt{2}}|0, 1\rangle + \frac{1}{\sqrt{2}}|1, 1\rangle. \quad (225)$$

You should work out what this input is as a vector. Now, since  $U_f$  is a linear operator, we have

$$U_f \left( \frac{1}{\sqrt{2}}|0, 1\rangle + \frac{1}{\sqrt{2}}|1, 1\rangle \right) = \frac{1}{\sqrt{2}}U_f(|0, 1\rangle) + \frac{1}{\sqrt{2}}U_f(|1, 1\rangle) \quad (226)$$

$$= \frac{1}{\sqrt{2}}|0, \overline{f(0)}\rangle + \frac{1}{\sqrt{2}}|1, \overline{f(1)}\rangle \quad (227)$$

This is an interesting state. If the function is constant, then the output state is a tensor product, but if the function is balanced, then the output is entangled. Hence, we already see qualitatively different behavior depending on whether the function is balanced or entangled. Let us consider a superposition for the first bit,



Again, we can use linearity of the tensor product to identify the input as

$$\frac{1}{\sqrt{2}}|0, 0\rangle - \frac{1}{\sqrt{2}}|0, 1\rangle. \quad (229)$$

Applying  $U_f$  to this and using linearity, the output is

$$\frac{1}{\sqrt{2}}U_f(|0, 0\rangle) - \frac{1}{\sqrt{2}}U_f(|0, 1\rangle) = \frac{1}{\sqrt{2}}|0, f(0)\rangle - \frac{1}{\sqrt{2}}|0, \overline{f(0)}\rangle \quad (230)$$

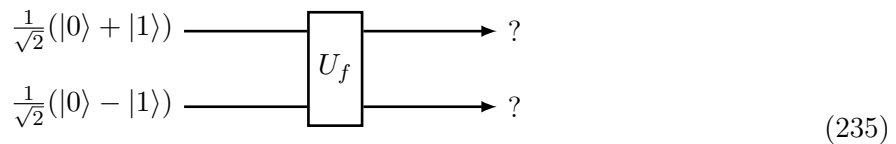
$$= \begin{cases} \frac{1}{\sqrt{2}}(|0, 0\rangle - |0, 1\rangle) & f(0) = 0 \\ \frac{-1}{\sqrt{2}}(|0, 0\rangle - |0, 1\rangle) & f(0) = 1 \end{cases} \quad (231)$$

$$= \frac{(-1)^{f(0)}}{\sqrt{2}}(|0, 0\rangle - |0, 1\rangle). \quad (232)$$

$$= \frac{(-1)^{f(0)}}{\sqrt{2}}(|0\rangle \otimes |0\rangle - |0\rangle \otimes |1\rangle). \quad (233)$$

$$= \frac{(-1)^{f(0)}}{\sqrt{2}}|0\rangle \otimes (|0\rangle - |1\rangle). \quad (234)$$

The reader should verify/justify every step in the derivation above. The last step uses linearity of the tensor product. Let's go all out and see what happens with a superposition on both input bits,



The input is  $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ . By linearity of the tensor product, this input is

$$\frac{1}{2}|0, 0\rangle - \frac{1}{2}|0, 1\rangle + \frac{1}{2}|1, 0\rangle - \frac{1}{2}|1, 1\rangle. \quad (236)$$

Applying  $U_f$  to this input and using linearity of  $U_f$ , we get

$$\begin{aligned} & \frac{1}{2}U_f(|0, 0\rangle) - \frac{1}{2}U_f(|0, 1\rangle) + \frac{1}{2}U_f(|1, 0\rangle) - \frac{1}{2}U_f(|1, 1\rangle) \\ = & \frac{1}{2}|0, f(0)\rangle - \frac{1}{2}|0, \overline{f(0)}\rangle + \frac{1}{2}|1, f(1)\rangle - \frac{1}{2}|1, \overline{f(1)}\rangle && \text{[definition of } U_f\text{]} \\ = & \frac{(-1)^{f(0)}}{2}(|0, 0\rangle - |0, 1\rangle) + \frac{(-1)^{f(1)}}{2}(|1, 0\rangle - |1, 1\rangle) && \text{[Check this]} \\ = & \frac{(-1)^{f(0)}}{2}(|0\rangle \otimes |0\rangle - |0\rangle \otimes |1\rangle) + \frac{(-1)^{f(1)}}{2}(|1\rangle \otimes |0\rangle - |1\rangle \otimes |1\rangle) && \text{[joint state tensor product]} \\ = & \frac{(-1)^{f(0)}}{2}|0\rangle \otimes (|0\rangle - |1\rangle) + \frac{(-1)^{f(1)}}{2}|1\rangle \otimes (|0\rangle - |1\rangle) && \text{[tensor product linearity]} \\ = & \underbrace{\frac{1}{\sqrt{2}}((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle)}_{\text{top qubit}} \otimes \underbrace{\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)}_{\text{bottom qubit}} && \text{[tensor product linearity]} \end{aligned} \quad (237)$$

Again, the reader should verify every step in the derivation above. The interesting point is that the output is a tensor product, that is, we can clearly identify the top qubit output state and the bottom qubit output state. The output qubits are not entangled,

$$\begin{array}{ccc} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) & \longrightarrow & \frac{1}{\sqrt{2}}((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle) \\ \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) & \longrightarrow & \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{array} \quad (238)$$

Notice that the top output qubit contains information about both  $f(0)$  and  $f(1)$ . Interestingly, the state of the bottom qubit has been left unchanged. This might be counterintuitive. In the definition of  $U_f$  it left the top bits unchanged. That was the definition of  $U_f$  for classical bits. Here we see that when  $U_f$  operates on general qubits, it may not leave the top qubit unchanged. In the end,  $Q_f$  is just a unitary operator. Its action on classical bits just serves to define how  $U_f$  operates on a basis, which completely defines the linear operator. This linear operator is a matrix, and we can always convert the input to a vector and apply the matrix to the vector. Our analysis above gives the result. It is imperative for the reader to work through the details of the next example.

**Example 13.1.** Let  $f$  be  $f_{11}$  from the beginning of the lecture.

1. Show that

$$U_f = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (239)$$



2. Show that the input and output to  $U_f$  in (238) are the vectors

$$|\text{input}\rangle = \begin{bmatrix} 1/2 \\ -1/2 \\ 1/2 \\ -1/2 \end{bmatrix}, \quad |\text{output}\rangle = \begin{bmatrix} -1/2 \\ 1/2 \\ -1/2 \\ 1/2 \end{bmatrix}. \quad (240)$$

3. Verify the following computation,

$$U_f \cdot |\text{input}\rangle = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1/2 \\ -1/2 \\ 1/2 \\ -1/2 \end{bmatrix} = \begin{bmatrix} -1/2 \\ 1/2 \\ -1/2 \\ 1/2 \end{bmatrix} = |\text{output}\rangle. \quad (241)$$

### 13.2 Untangling the Output

We now know how  $U_f$  acts on superpositions. Lets get back to the problem at hand, to determine if  $f$  is balanced. The recap of where we are is:

$$\begin{array}{ccc} \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) & \xrightarrow{\quad U_f \quad} & \frac{1}{\sqrt{2}}((-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle) \\ \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) & \xrightarrow{\quad U_f \quad} & \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{array} \quad (242)$$

We can now relate to the human eye which receives all the EM-rays from the environment. We need to untangle the signal. In our case, the signal is the top qubit which contains information about  $f$  on all inputs. So, let's measure the top qubit. The result is  $|0\rangle$  with probability  $1/2$  and  $|1\rangle$  with probability  $1/2$ . That's no help, because the result of the measurement is independent of  $f$ , even though the top bit is dependent on  $f$ . Let's take a closer look at the top qubit in the output,

$$\frac{1}{\sqrt{2}} \left( (-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle \right) = \begin{bmatrix} (-1)^{f(0)}/\sqrt{2} \\ (-1)^{f(1)}/\sqrt{2} \end{bmatrix}. \quad (243)$$

Recall the Hadamard matrix,

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \quad (244)$$

What happens if we hit this top qubit with the Hadamard matrix,

$$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} (-1)^{f(0)}/\sqrt{2} \\ (-1)^{f(1)}/\sqrt{2} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} (-1)^{f(0)} + (-1)^{f(1)} \\ (-1)^{f(0)} - (-1)^{f(1)} \end{bmatrix} \quad (245)$$

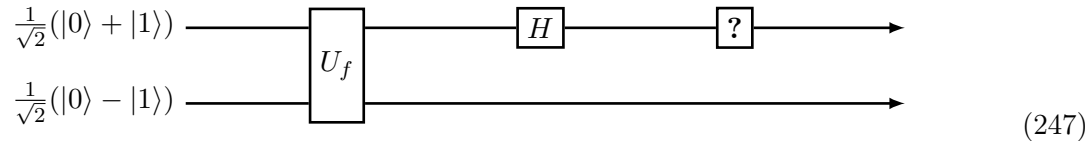
$$= \begin{cases} \begin{bmatrix} \pm 1 \\ 0 \end{bmatrix} & f \text{ is constant} \\ \begin{bmatrix} 0 \\ \pm 1 \end{bmatrix} & f \text{ is balanced} \end{cases} \quad (246)$$

We are at the punchline. The Hadamard matrix is the great untangler of superpositions. By applying the Hadamard matrix to the top qubit, it has untangled the superposition of information

to always give a pure state, a classical bit. If you measure the top qubit after applying the great untangler, you will *always* measure  $|0\rangle$  if  $f$  is constant, and you will *always* measure  $|1\rangle$  if  $f$  is balanced. Done! We have a way to test if  $f$  is balanced by evaluating  $f$ , i.e. applying  $U_f$ , once.

### 13.3 Quantum Circuit for 1-bit Deutsch-Jozsa

The final quantum algorithm that achieves the miracle of testing if a function is balanced using just one evaluation of the function is represented in the following circuit.



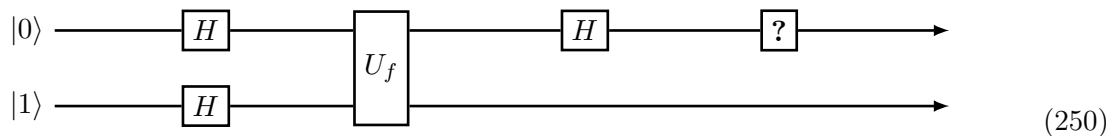
Recall that  $\boxed{?}$  is the measurement operation. So, the circuit above indicates that we apply  $H$  to the top qubit and then measure the top qubit. We would be done, except for a practical consideration. We evaluate  $U_f$  on a joint state which is a superposition. However, we cannot in general create arbitrary qubit states, in particular the superpositions that are input into  $U_f$ . It is easy to create pure states, i.e. classical bits. For example if electron spin is how we are encoding the bit, then the bit can be set to  $|\uparrow\rangle$  by appropriately setting a magnetic field, and similarly for  $|\downarrow\rangle$ .

How do we get these superposition states to feed into  $U_f$ , given that we can only start with pure classical bits. We need to apply some unitary operator to the classical states, and also hope that unitary operator can be implemented as a quantum gate by some physical process. Luckily we do not need to look too far for this unitary operator. The Hadamard will do, because note that,

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \quad (248)$$

$$H|1\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \quad (249)$$

These are exactly what we need. So the mighty Hadamard is not only the great untangler of superposed states, but it is also the great creator of superposed states. The final quantum circuit to solve the 1-bit Deutsch Jozsa problem is



We can now write down the unitary operator that corresponds to the circuit above,

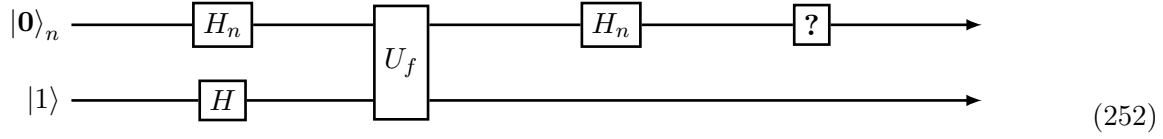
$$(H \otimes I) \cdot U_f \cdot (H \otimes H). \quad (251)$$

Note that  $H \otimes H$  is a Hadamard matrix of dimension 4. We are applying this unitary operator to  $|0\rangle \otimes |1\rangle$ , but this operator can be applied to any input  $|\psi\rangle$ . The next exercise is imperative.

**Example 13.2.** Compute the operator in (251) for  $f_{11}$  in Example 13.1. Apply this operator to the input  $|0\rangle \otimes |1\rangle = [0, 1, 0, 0]^T$ . Explain how the output vector agrees with the top bit being  $|0\rangle$ .

## 14 Quantum Circuits

We just saw our first quantum circuit that tests balance of 1-bit function,



Let us briefly discuss building and analyzing quantum circuits. We must guess what gates will be standard when quantum hardware is widely available. Presumably these gates will be stable and implemented in some physical quantum substrate, much like today we implement Boolean logic gates like NAND in semiconductors. With no other guide, let us focus on some useful gates,

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \quad \text{NOT} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; \quad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}; \quad R(\theta) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix}. \quad (253)$$

Additionally, we one can multiply by a phase  $e^{i\phi}$ ,<sup>4</sup> use controlling bits to control the output of a gate, and also take the controlled XOR of output qubits, providing all this operations are unitary. We already saw the use of controlling bits, and we used the controlled XOR when we converted a Boolean function  $f$  to its unitary equivalent  $U_f$ . Using these standard gates plus the additional operations, we can pretty much implement any unitary operator we need. We address two tasks:

1. Given a quantum circuit that uses the standard gates and controlling inputs, what does it do.

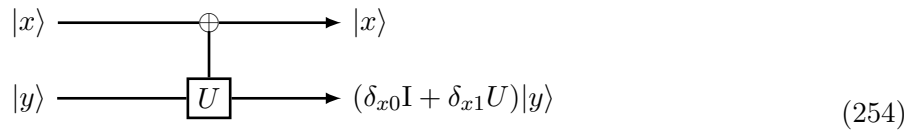
That is, what is the unitary operator it implements.

2. Given a task, i.e., a unitary operator, find a circuit that implements the operator.

As we discuss several examples, we will also see conventions for graphically defining the circuits.

### 14.1 Finding the Operator for a Circuit

Let  $U$  be the unitary operator  $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ . The operator  $U$  operates on 1 qubit. We start with a simple example, the quantum equivalent of IF...ELSE..., the controlled  $U$  in (196) on page 45,



The notation on the bottom qubit says if  $x = 0$  (when  $\delta_{x0} = 1$ ) then the output is  $|y\rangle$  and  $U$  does not operate. If  $x = 1$  (when  $\delta_{x1} = 1$ ) then  $U$  operates and produces  $U|y\rangle$ . To analyze this circuit, we see what it does on the standard basis  $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ ,

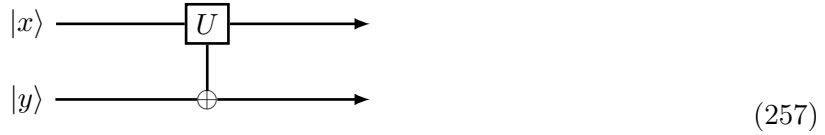
$$|00\rangle \rightarrow |00\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad |01\rangle \rightarrow |01\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad |10\rangle \rightarrow |1\rangle \otimes \begin{bmatrix} a \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ a \\ c \\ 0 \end{bmatrix} \quad |11\rangle \rightarrow |1\rangle \otimes \begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} 0 \\ b \\ d \\ 0 \end{bmatrix}. \quad (255)$$

<sup>4</sup>This is just unitary evolution according to a constant Hamiltonian.

We can now write down the operator for controlled- $U$ ,

$$c_U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & U \end{bmatrix}. \quad (256)$$

Let us use this method for several similar examples. What if we flip the controlling bit and  $U$ ,



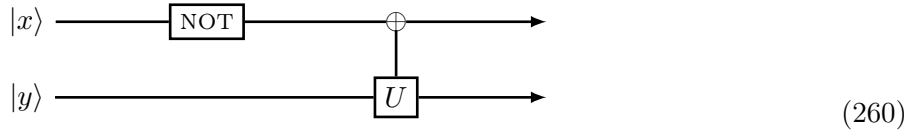
Verify that the standard basis transforms as

$$|00\rangle \rightarrow |00\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad |01\rangle \rightarrow \begin{bmatrix} a \\ c \end{bmatrix} \otimes |1\rangle = \begin{bmatrix} 0 \\ a \\ 0 \\ c \end{bmatrix} \quad |10\rangle \rightarrow |10\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad |11\rangle \rightarrow \begin{bmatrix} b \\ d \end{bmatrix} \otimes |1\rangle = \begin{bmatrix} 0 \\ b \\ 0 \\ d \end{bmatrix}. \quad (258)$$

We can now write down the operator,

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & a & 0 & b \\ 0 & 0 & 1 & 0 \\ 0 & c & 0 & d \end{bmatrix}. \quad (259)$$

What if we negate the controlling bit first,

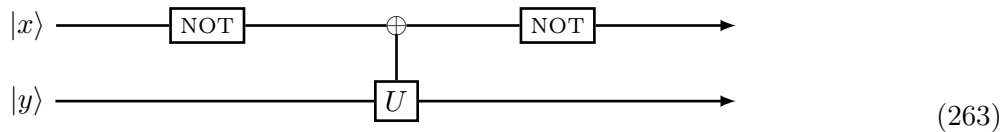


$$|00\rangle \rightarrow |1\rangle \otimes \begin{bmatrix} a \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ a \\ c \end{bmatrix} \quad |01\rangle \rightarrow |1\rangle \otimes \begin{bmatrix} b \\ d \end{bmatrix} = \begin{bmatrix} 0 \\ b \\ d \end{bmatrix} \quad |10\rangle \rightarrow |00\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad |11\rangle \rightarrow |01\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}. \quad (261)$$

We can now write down the operator,

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ a & b & 0 & 0 \\ c & d & 0 & 0 \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ U & \mathbf{0} \end{bmatrix}. \quad (262)$$

This is not quite the equivalent of IF NOT  $x \dots$  because  $x$  is negated at the end. To get a true negatively controlled  $U$ , we can negate  $x$  back to its original bit after it has done its controlling,

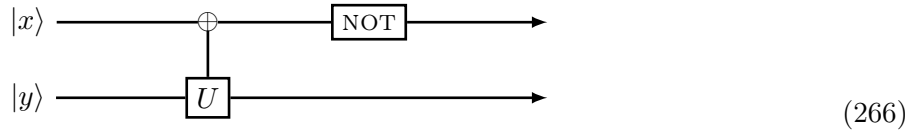


$$|00\rangle \rightarrow |0\rangle \otimes \begin{bmatrix} a \\ c \\ 0 \\ 0 \end{bmatrix} \quad |01\rangle \rightarrow |0\rangle \otimes \begin{bmatrix} b \\ d \\ 0 \\ 0 \end{bmatrix} \quad |10\rangle \rightarrow |10\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad |11\rangle \rightarrow |11\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \quad (264)$$

We can now write down the operator,

$$\begin{bmatrix} a & b & 0 & 0 \\ c & d & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} U & \mathbf{0} \\ \mathbf{0} & I \end{bmatrix}. \quad (265)$$

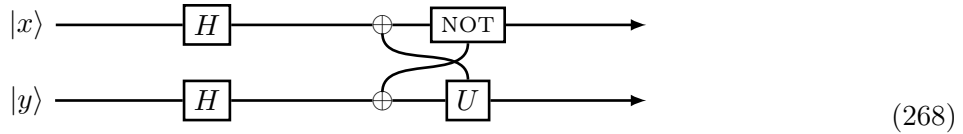
One final example which we leave to the reader. If we negate only after controlling,



show that the resulting operator is  $\begin{bmatrix} \mathbf{0} & U \\ I & \mathbf{0} \end{bmatrix}$ . So, given any operator  $U$ , we see that we can construct the following four operators using either controlling bits or negated controlling bits,

$$\begin{bmatrix} I & \mathbf{0} \\ \mathbf{0} & U \end{bmatrix} \quad \begin{bmatrix} \mathbf{0} & I \\ U & \mathbf{0} \end{bmatrix} \quad \begin{bmatrix} U & \mathbf{0} \\ \mathbf{0} & I \end{bmatrix} \quad \begin{bmatrix} \mathbf{0} & U \\ I & \mathbf{0} \end{bmatrix}. \quad (267)$$

Let us find the operator for this complicated circuit,



Note the use of two controlling bits simultaneously. That is perfectly fine. Let us try to figure out what happens when we hit the basis vector  $|00\rangle$  with this circuit. The first “level” of the circuit produces  $H|0\rangle \otimes H|0\rangle$ . That was easy enough. But now when we feed this to the second “level”, it is not clear what happens.

We saw that the circuit has two levels in sequence. The action plan is to figure out the operator for each level, and then we get the operator for the whole circuit by multiplying. The operator for the first level is easy, it is just  $H \otimes H$ .

Let us see what happens if the standard basis is fed directly into the second level,

$$|00\rangle \rightarrow |00\rangle \quad |01\rangle \rightarrow \text{NOT}|0\rangle \otimes |1\rangle \quad |10\rangle \rightarrow |1\rangle \otimes U|0\rangle \quad |11\rangle \rightarrow \text{NOT}|1\rangle \otimes U|1\rangle. \quad (269)$$

We can now write down the operator for the second level,

$$\begin{bmatrix} 1 & 0 & 0 & b \\ 0 & 0 & 0 & d \\ 0 & 0 & a & 0 \\ 0 & 1 & c & 0 \end{bmatrix}. \quad (270)$$

Note that this is not a unitary operator unless  $U$  is diagonal, so it could not be implemented by a quantum mechanical circuit. Nevertheless, it is a useful exercise to flex our muscles. The (generally non-unitary) operator for the circuit is then

$$\begin{bmatrix} 1 & 0 & 0 & b \\ 0 & 0 & 0 & d \\ 0 & 0 & a & 0 \\ 0 & 1 & c & 0 \end{bmatrix} \cdot (H \otimes H). \quad (271)$$

## 14.2 Building a Circuit for an Operator

Finding the circuit for an operator is like computer programming. Writing a program to solve a problem requires creativity. There is no prescription. And so it is with quantum computer programming. Given a unitary operator that solves a task, one has to use some creativity to find a circuit to implement the operator. There are some useful tricks to know, starting with (267). Here are a few more tricks. Given  $n$ -qubit unitary operators  $A, B$ , we can get

$$\begin{bmatrix} A & \mathbf{0} \\ \mathbf{0} & B \end{bmatrix} \quad (272)$$

using the trick

$$\begin{bmatrix} A & \mathbf{0} \\ \mathbf{0} & B \end{bmatrix} = \begin{bmatrix} A & \mathbf{0} \\ \mathbf{0} & I \end{bmatrix} \begin{bmatrix} I & \mathbf{0} \\ \mathbf{0} & B \end{bmatrix}. \quad (273)$$

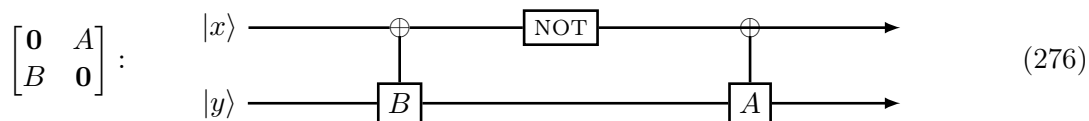
We know circuits for both operators on the right (see (267)) which we run in sequence. Consider

$$\begin{bmatrix} \mathbf{0} & A \\ B & \mathbf{0} \end{bmatrix} \quad (274)$$

(It is a useful exercise to show that this operator is indeed unitary.) We use the identity

$$\begin{bmatrix} \mathbf{0} & A \\ B & \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{0} & I \\ I & \mathbf{0} \end{bmatrix} \begin{bmatrix} A & \mathbf{0} \\ \mathbf{0} & B \end{bmatrix}. \quad (275)$$

The first operator on the RHS is  $\text{NOT} \otimes I$ . Applying these in sequence gives this circuit for  $\begin{bmatrix} \mathbf{0} & A \\ B & \mathbf{0} \end{bmatrix}$ ,

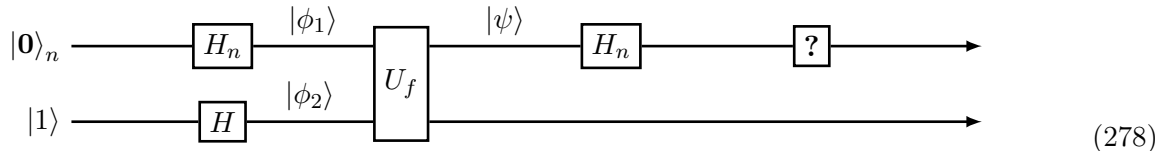


Verify the circuit above. Practice makes perfect. Try to construct a circuit for this operator,

$$U = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad (277)$$

## 15 Testing Balance of $n$ -bit Functions

Assume the  $n$ -qubit function  $f$  is either balanced or constant. Recall that  $U_f$  is the  $2^{n+1} \times 2^{n+1}$  unitary operator that is equivalent to  $f$ . A simple extension of the circuit that tested balance of a 1-bit function is



It is going to be a heavy linear algebra lift to figure out what this circuit is doing. The only way to learn this stuff is to work the algebra yourself.

### 15.1 Deutsch-Jozsa Algorithm

Let's analyze this circuit in the two cases,  $f$  is constant and  $f$  is balanced. First note that the input to the first  $H_n$  is  $|0\rangle \otimes |0\rangle \otimes \cdots \otimes |0\rangle = |00\cdots 0\rangle$  which is the first standard basis vector. Therefore  $H_n|0\rangle_n$  is just the first column of  $H_n$ . We know that the first column of  $H_n$  is just a  $2^n$ -vector of 1's, normalized. Hence,

$$|\phi_1\rangle = H_n|0\rangle_n = \frac{1}{2^{n/2}} \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 1 \\ 1 \end{bmatrix} = \frac{1}{2^{n/2}} \sum_{i=1}^{2^n} e_i = \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle. \quad (279)$$

The RHS is simply a statement that  $H_n$  is the great entangler, constructing a uniform superposition of every possible  $n$ -bit pure state starting from  $|0\rangle$ . A much simpler computation gives

$$|\phi_2\rangle = H|1\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \quad (280)$$

Therefore, the input to  $U_f$  is

$$|\phi_1\rangle \otimes |\phi_2\rangle = \left( \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle \right) \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \quad (281)$$

$$= \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \quad (282)$$

In the last step we used linearity of the tensor product. We now apply  $U_f$  to this sum of tensor products to get  $|\psi\rangle$ . Using linearity of  $U_f$ , we get

$$\frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} U_f \left( |x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \right). \quad (283)$$

Recall that  $|x\rangle$  is a classical pure state, a standard basis vector. We have

$$U_f \left( |x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \right) = \frac{1}{\sqrt{2}} U_f(|x\rangle \otimes |0\rangle - |x\rangle \otimes |1\rangle) \quad (284)$$

$$= \frac{1}{\sqrt{2}} U_f(|x\rangle \otimes |0\rangle) - \frac{1}{\sqrt{2}} U_f(|x\rangle \otimes |1\rangle) \quad (285)$$

$$= \frac{1}{\sqrt{2}} |x\rangle \otimes |f(x)\rangle - \frac{1}{\sqrt{2}} |x\rangle \otimes |\overline{f(x)}\rangle \quad (286)$$

$$= (-1)^{f(x)} |x\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} \quad (287)$$

The first step uses linearity of tensor product; the second step uses linearity of  $U_f$ ; the third step uses the definition of  $U_f$ , because  $|x\rangle$  is a classical pure state; the last step is for the reader to verify, depending on whether  $f(x) = 0$  or  $f(x) = 1$ . Plugging this back into the expression for the output from  $U_f$ , we get

$$\frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} |x\rangle \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} \quad (288)$$

$$= \left( \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} |x\rangle \right) \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}, \quad (289)$$

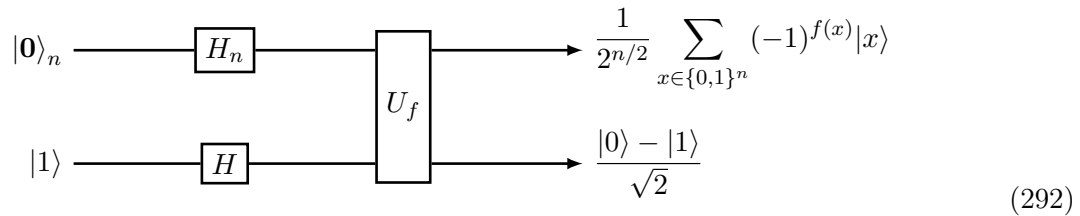
where the last step uses linearity of the tensor product. The state on the top  $n$  output qubits is

$$|\psi\rangle = \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} |x\rangle. \quad (290)$$

and the state on the bottom output is

$$\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \quad (291)$$

In terms of the circuit diagram, here is where we are,



Something strange. All of the information about  $f$  is in the input registers, even though  $U_f$  was designed with the information about  $f$  in the output bit. This is one of the strange things that can happen when an operator hits a superposition of states. For a pure classical state as input, the information about  $f$  would be in the output qubit. This is not so when the input is a superposition.

To get the final output of the top  $n$  qubits, we hit  $|\psi\rangle$  with  $H_n$ , now acting as the great untangler. Let's first consider the case where  $f$  is constant. Then

$$|\psi\rangle = \pm \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} |x\rangle = \pm H_n |0\rangle, \quad (293)$$



where the  $\pm$  depends on whether  $f(x) = 0$  or  $f(x) = 1$ . In this case,  $H_n|\psi\rangle$  is given by

$$H_n|\psi\rangle = \pm H_n^2|0\rangle = \pm|0\rangle, \quad (294)$$

because  $H_n^2 = I$ . So, when we measure the top  $n$  qubits, they will all be zero, because no other classical state has non-zero amplitude. Let's see what happens if  $f$  is balanced. In this case  $H_n|\psi\rangle$  is complicated. Let us write

$$|\psi\rangle = \frac{1}{2^{n/2}} \begin{bmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \\ \vdots \\ \psi_{2^n} \end{bmatrix} \quad (295)$$

Each  $\psi_i$  is  $\pm 1$  because  $|\psi\rangle$  is a linear combination of all the basis vectors with each amplitude being  $(-1)^{f(x_i)}/2^{n/2}$ . Since  $f$  is balanced, half of the  $\psi_i$  are  $+1$  and the other half are  $-1$ . This means  $\mathbf{1}^T|\psi\rangle = 0$ . This is interesting, because when we consider  $H_n|\psi\rangle$ , we get

$$H_n|\psi\rangle = \frac{1}{2^{n/2}} \begin{bmatrix} \mathbf{h}_1^T \\ \mathbf{h}_2^T \\ \mathbf{h}_3^T \\ \vdots \\ \mathbf{h}_{2^n}^T \end{bmatrix} \begin{bmatrix} \psi_1 \\ \psi_2 \\ \psi_3 \\ \vdots \\ \psi_{2^n} \end{bmatrix}. \quad (296)$$

The  $\mathbf{h}_i^T$  are the rows of  $H_n$ . The first entry in  $H_n|\psi\rangle$  is the amplitude for the first basis vector  $|000\cdots 0\rangle$ . So, the first entry in  $H_n|\psi\rangle$  is the amplitude for the pure state  $|000\cdots 0\rangle$ . The first row of  $H_n$  is all ones,  $\mathbf{h}_1^T = \mathbf{1}^T$ . Because  $\mathbf{1}^T|\psi\rangle = 0$ ,  $H_n|\psi\rangle$  has zero in its first entry,

$$H_n|\psi\rangle = \begin{bmatrix} 0 \\ ? \\ ? \\ \vdots \\ ? \end{bmatrix}. \quad (297)$$

While we do not know the other entries in  $H_n|\psi\rangle$ , the fact that the first entry is 0 is huge. It means the amplitude to measure all bits 0 is zero when  $f$  is balanced. If  $f$  is balanced, at least one of the measured bits will be 1. If  $f$  is constant, all measured bits will be 0. We have our algorithm:

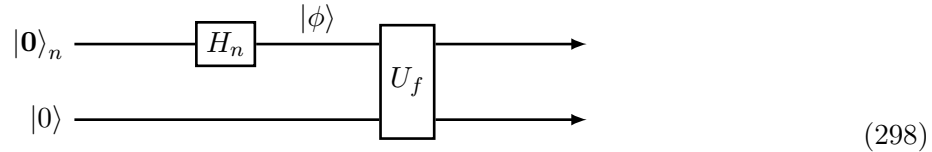
- 1: Run the circuit.
- 2: Measure the top  $n$  qubits.
- 3: **if** all qubits are 0 **then**
- 4:    $f$  is constant.
- 5: **else**
- 6:    $f$  is balanced.

Our analysis proves the algorithm works. Yes, the problem is contrived, requiring the function to be either constant or perfectly balanced. It is interesting to analyze other cases. We have our first quantum algorithm. With one function evaluation it tests balance, where the classical algorithm requires  $2^{n-1} + 1$  function evaluations. That is an exponential gain.

However, is it really. If we are to implement this circuit by explicitly encoding  $U_f$  then all efficiency gains are lost since  $U_f$  is exponential in size. The information theoretic gain is still there, though, that one function “evaluation” suffices. The quantum gains come when the function is specified directly as  $U_f$  using a small circuit, implemented using basic quantum gates. By one application of this circuit, we test if  $f$  is balanced or constant. There is no information paradox here. We traded 1-bit of information of  $f$  on some input with 1-bit of information regarding the relationships between  $f$  on several inputs. In both cases, we have received just 1-bit of information.

## 16 Philosophy of Quantum Algorithms

Let's take stock. A quantum algorithm somehow miraculously gets global information about a Boolean function  $f$  by simultaneously evaluating it on all possible inputs at once. To accomplish this, the typical first step is to place the starting pure state into an appropriate superposition of all the pure states. Here is an example,



The input to  $U_f$  is  $|\phi\rangle \otimes |0\rangle$ , where

$$|\phi\rangle = \frac{1}{2^{n/2}} \sum_{z \in \{0,1\}^n} |z\rangle. \quad (299)$$

The sum is over all  $n$ -bit pure states. The definition of  $U_f$  for pure states is

$$U_f(|z\rangle \otimes |0\rangle) = |z\rangle \otimes |f(z)\rangle. \quad (300)$$

Using linearity of  $U_f$  and tensor product, we find that the output is

$$|\text{output}\rangle = \frac{1}{2^{n/2}} \sum_{z \in \{0,1\}^n} |z\rangle \otimes |f(z)\rangle. \quad (301)$$

A lot of the magic in a quantum algorithm happens here. We have miraculously applied  $f$  to every possible input  $z$ , and all this information is contained in the output state. The rest of the magic in the quantum algorithm is to see if we can untangle all this information to get at what we want. One way to untangle is to measure the output. Now the state collapses by randomly picking one of the pure states,  $|z_0\rangle \otimes |f(z_0)\rangle$ . So, in the end we get only the single bit of information  $f(z_0)$ . We are only entitled to one bit of information about  $f(x)$  because we only made one evaluation of  $U_f$ .

So what is going on with testing balance of  $f$ . It seems like we are getting information about  $f$  globally, a lot more than one bit of information. No. It is still just one bit of information, a binary outcome on whether  $f$  is balanced or constant. It is just that the nature of the information is different. Rather than the 1-bit of information giving the value of  $f(z_0)$ , it gives information on the relationship between values of  $f$ . But, it is still one bit of information.

Can we get a free-lunch by continuing to measure the output? First we get  $|z_0\rangle \otimes |f(z_0)\rangle$ . The state collapses to this pure state and from then we always get  $|z_0\rangle \otimes |f(z_0)\rangle$ . So we don't get additional information about  $f$  by continuing to measure. Quantum state collapse prevents this. Here is a new idea. Before measuring, make several identical copies of  $|\text{output}\rangle$ . Each copy will collapse independently to possibly different pure states, giving  $f$  for those pure states. Thus, we extract multiple bits from one evaluation of  $U_f$ . This too is prevented. Why? Because you cannot clone quantum states, by the no-cloning theorem we proved earlier. Everything neatly fits together.

## 16.1 Directly Building a Circuit for $U_f$

You may be worried about where we got  $U_f$ . All the information by  $f$  is stored in  $U_f$  (twice in fact), so if we explicitly construct the  $2^n \times 2^n$  matrix  $U_f$ , then we are basically examining  $f$  on every input. And certainly, if the quantum circuit is doing no more than explicitly computing the matrix product  $U_f(|\phi\rangle \otimes |0\rangle)$ , then the runtime will be exponential. For this to be practical in any way, it is necessary for us to be given  $U_f$  as a black-box quantum circuit, presumably one that only uses  $\text{poly}(n)$  quantum gates. Let's see an example of this.

Fix an  $n$ -bit parameter  $a = [a_1, \dots, a_n]$  and define a function on  $n$  bits  $x = [x_1, \dots, x_n]$ ,

$$f(x) = a_1x_1 \oplus a_2x_2 \oplus a_3x_3 \oplus \dots \oplus a_nx_n. \quad (302)$$

The addition is modulo 2, equivalent to XOR.  $f(x)$  is like an inner product for binary vectors.  $f$  takes the inner product of  $x$  with  $a$ , a fixed parameter. For  $a = [1, 0, 1]$ , we have

$x_1$	$x_2$	$x_3$	$f(x)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

(303)

We can immediately write down  $U_f$  as a block diagonal with  $I$  when  $f(e_i) = 0$  and NOT otherwise,

$$\begin{bmatrix}
 \boxed{I_2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} \\
 0_{2 \times 2} & \boxed{\text{NOT}} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} \\
 0_{2 \times 2} & 0_{2 \times 2} & \boxed{I_2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} \\
 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & \boxed{\text{NOT}} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} \\
 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & \boxed{\text{NOT}} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} \\
 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & \boxed{I_2} & 0_{2 \times 2} & 0_{2 \times 2} \\
 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & \boxed{\text{NOT}} & 0_{2 \times 2} \\
 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & \boxed{I_2}
 \end{bmatrix} \quad (304)$$

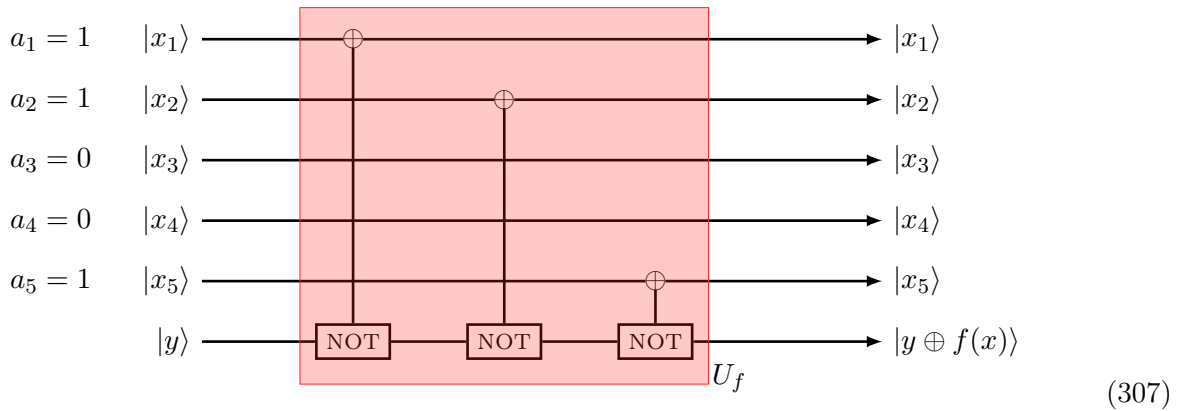
You can now try to implement this unitary operator using the tricks that we saw earlier. But that is not the purpose of this exercise. Looking at the form for  $f(x)$ , we can try to build the circuit directly. Recall that the output of  $U_f$  on a pure state is

$$U_f(|x\rangle \otimes |y\rangle) = |x\rangle \otimes |y \oplus f(x)\rangle. \quad (305)$$

The bottom (output) qubit is just

$$|y \oplus a_1x_1 \oplus a_2x_2 \oplus a_3x_3 \oplus \dots \oplus a_nx_n\rangle. \quad (306)$$

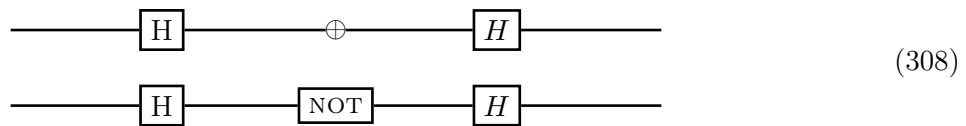
Starting with the  $y$ -bit, process each  $x_i$ . If the corresponding  $a_i$  is 0, do nothing. If the corresponding  $a_i$  is 1, flip the  $y$ -bit provided the  $x_i$ -bit is 1. That is,  $x_i$  is controlling a NOT on  $y$ . It is only such a control bit when  $a_i$  is 1. Here is the circuit that implements this logic,



We went directly from the definition of an important function to its circuit. This circuit generalizes to  $n$ -bits and any  $a$ , and is compact (linear in size). The matrix  $U_f$  grows exponentially, which does not matter if we are given  $U_f$  as a compact black-box circuit using standard quantum gates.

## 16.2 Circuit Uniqueness

The circuit for a given operator is not unique. Consider this circuit,



The reader may verify that the operator for this circuit is

$$(H \otimes H) \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot (H \otimes H) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \quad (309)$$

Now consider the circuit where we remove the Hadamards and reverse the roles of the controlling bit and the controlled bit,



Again, please verify that the operator for this circuit is

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \quad (311)$$

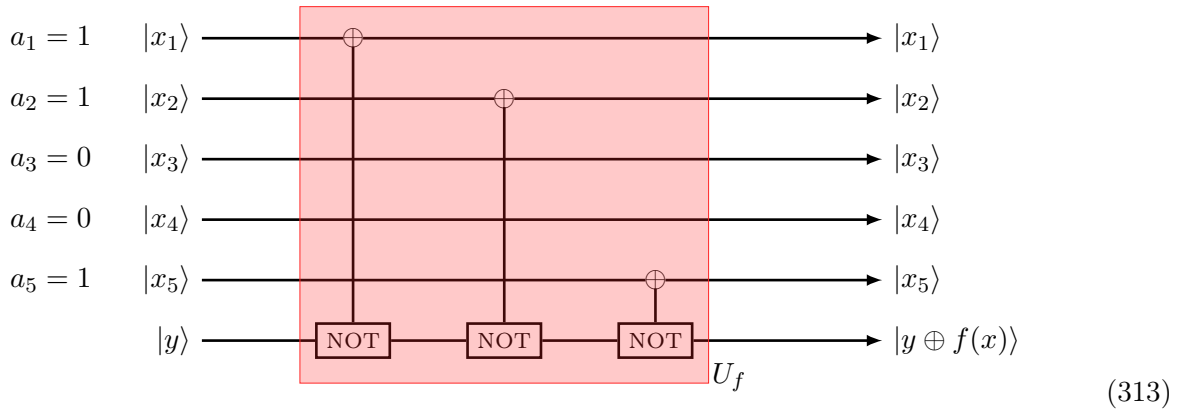
These two circuits implement the same operator. So circuits for a given operator are not unique. It is a very hard task to find the minimum sized circuit for a given operator. It is also interesting that sandwiching a controlled not between Hadamards is equivalent to simply switching the roles of the controlling and controlled bits. This particular fact will be useful later.

## 17 Learning the Weights in a Linear Function

In the last lecture we considered the Boolean version of a linear function,

$$f(x) = a_1x_1 \oplus a_2x_2 \oplus a_3x_3 \oplus \cdots \oplus a_nx_n, \quad (312)$$

where  $a = a_1, \dots, a_n$  and  $x = x_1, \dots, x_n$  are  $n$ -bit vectors. Think of  $a$  as a hidden parameter and  $x$  is the input to the function  $f$ . This is a convenient function because we can directly construct a compact circuit for  $U_f$ . For  $a = 11001$ , the circuit for  $U_f$  is



Given  $f$  (or  $U_f$ ) as a black box, the Bernstein-Vazirani problem is to infer the parameters  $a$ .

A prominent example of this problem arose in the context of college rankings. Here is the a simplified story. U.S. News maintains a ranking of colleges. A university has a set of features

$$x = [\text{in-coming SAT, \% small classes, faculty-to-student-ratio, } \dots] \quad (314)$$

Many of the features in  $x$  are only known to the university and so they are self-reported. U.S. News computes a score  $a^T x$  for a set of proprietary weights. U. S. News heavily guards the privacy of these weights. The universities are ranked according to this score. Why are the weights guarded? Because if some university were to know those weights, they could strategically improve their features for the large weights, thereby manipulating the rankings. Every year a university's features change, and they see their ranking (a proxy for the score). They see this every year, hence they are repeatedly querying a function like  $f$  with different  $x$ . At some point they have learned  $a$  and can strategically improve their rankings. This actually happened.

Let's get back to  $f$  in (312). Observe that

$$\begin{aligned} f(1000 \cdots 00) &= a_1 \\ f(0100 \cdots 00) &= a_2 \\ f(0010 \cdots 00) &= a_3 \\ f(0001 \cdots 00) &= a_4 \\ &\vdots \\ f(0000 \cdots 10) &= a_{n-1} \\ f(0000 \cdots 01) &= a_n \end{aligned} \quad (315)$$

With  $n$  function evaluations, we have learned  $a$ . If  $n$  is large enough, the weights are safeguarded because many queries are needed before we can learn  $a$ . Each query gives us 1 bit of information,

so with  $n$  queries we get the  $n$ -bits of information we need, namely  $a_1, \dots, a_n$ . We cannot hope to do better using queries that only reveal 1-bit of information. Here is an information theoretic proof. There are fewer than  $2^m$  possible outcomes using  $m$  one-bit queries. Since there are  $2^n$  possible choices for  $a$ , if  $2^m < 2^n$ , by pigeonhole, any mapping from the set of  $\{a\}$  to the query outcomes assigns at least  $a$ 's to the same query outcome. Those two  $a$ 's aren't disambiguated.

In the classical world, the weights are safe. In the quantum world this is not the case. A single evaluation of  $U_f$  suffices to reveal  $a$ . We begin our quantum algorithm by applying  $U_f$  to a superposition of states. This is usually the first bit of magic in any quantum algorithm. We get information on  $f$  for all inputs using just one evaluation of  $U_f$ . The second bit of magic comes when it is time to untangle this information to get at what we need. Here is the first step,

$$(316)$$

We have already derived this first step many times, using linearity of tensor product and linearity of  $U_f$  (see, for example, the discussion accompanying the Deutsch-Jozsa algorithm on page 48). The reader should derive it one more time themselves and commit this first step to memory. This is typically the start of any quantum algorithm.

How do we untangle the information in the first  $n$  qubits. Since  $H$  is also the great untangler, lets try  $H$  and see what happens. At this point it is good to tinker with an example. Let  $a = 01$ .

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

$$(317)$$

We can immediately write down  $U_f$ ,

$$U_f = \begin{bmatrix} \boxed{I_2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} \\ 0_{2 \times 2} & \boxed{\text{NOT}} & 0_{2 \times 2} & 0_{2 \times 2} \\ 0_{2 \times 2} & 0_{2 \times 2} & \boxed{I_2} & 0_{2 \times 2} \\ 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & \boxed{\text{NOT}} \end{bmatrix} \quad (318)$$

The circuit we are proposing to extract  $a$  is

$$(319)$$



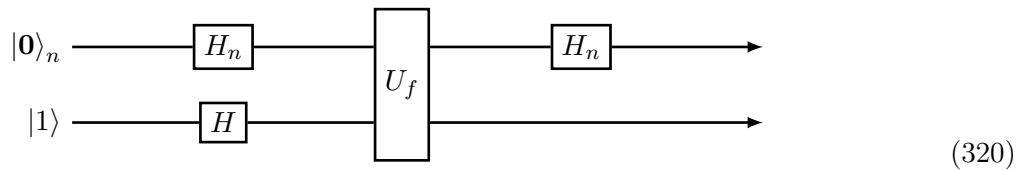
The operator for this circuit is  $(H_2 \otimes I) \cdot U_f \cdot (H_2 \otimes H)$ . The input to the circuit is  $|001\rangle = [0, 1, 0, 0, 0, 0, 0, 0]^T$ . It is now just an exercise in careful linear algebra to verify

$$(H_2 \otimes I) \cdot U_f \cdot (H_2 \otimes H) \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{matrix} |000\rangle \\ |001\rangle \\ |010\rangle \\ |011\rangle \\ |100\rangle \\ |101\rangle \\ |110\rangle \\ |111\rangle \end{matrix}.$$

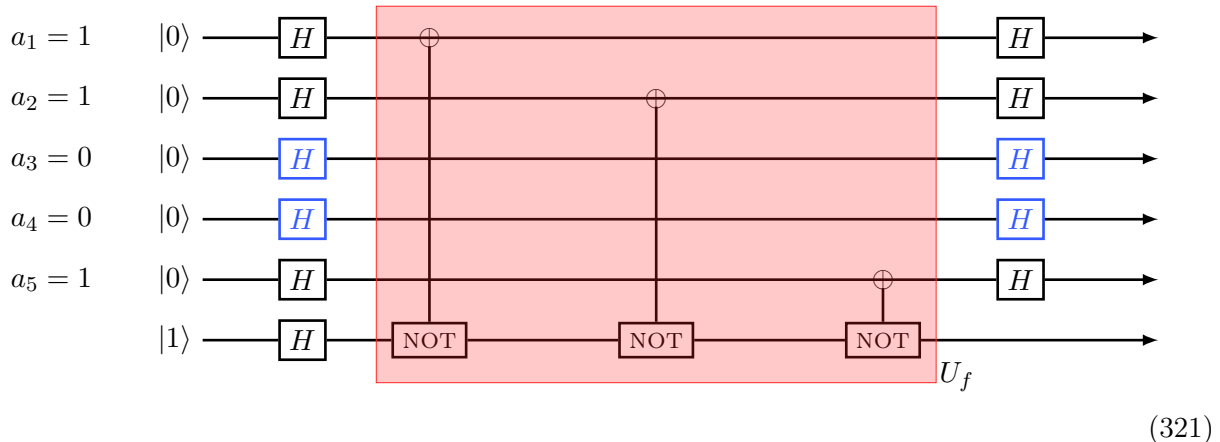
On the right we show the pure state corresponding to each amplitude. We also highlighted the first two bits of the pure state. Note that the amplitude is only non-zero when the first two bits are 01, which is exactly  $a$ . If we measure the top two qubits, we get  $a$ . Is this always the case. Yes!

### 17.1 Circuit for Bernstein-Vazirani

We claim, in general, that the circuit

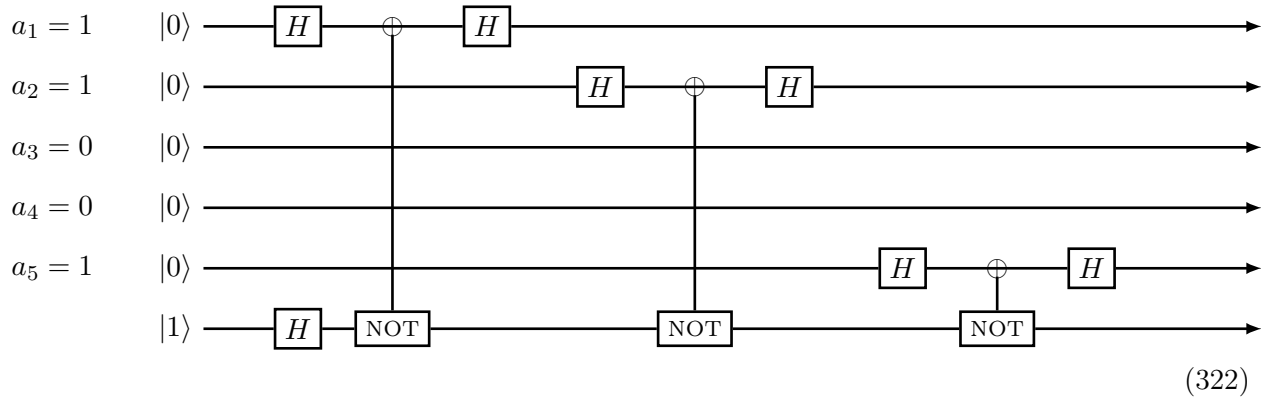


Here is a proof by picture on an example with  $a = 11001$ . We directly examine the circuit above using the circuit for  $U_f$ . The circuit to learn  $a$  is

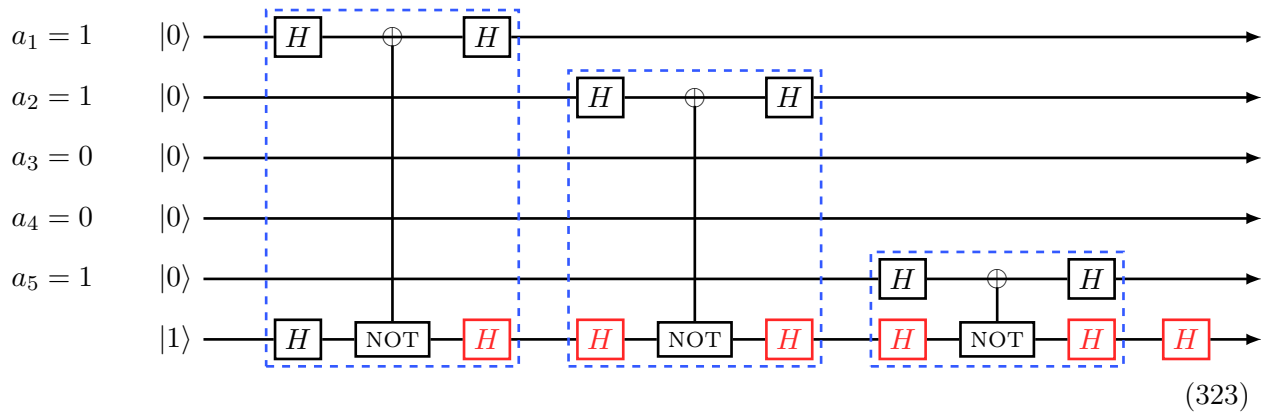


Note that on the third and fourth qubits where  $a_3 = a_4 = 0$ , the input bit is hit by  $H^2 = I$  in sequence. The circuit can be simplified by removing pairs of Hadamards in sequence (the Hadamards in blue). We can also slide an operator along a qubit wire up to any other operator or controlling bit. For example, the Hadamard on the right in the top qubit wire can slide all the way left up to the controlling bit. We cannot slide it further, because that would interfere with the controlling operator. Similarly, we can slide the left Hadamard on the 2nd qubit wire to the right, and the

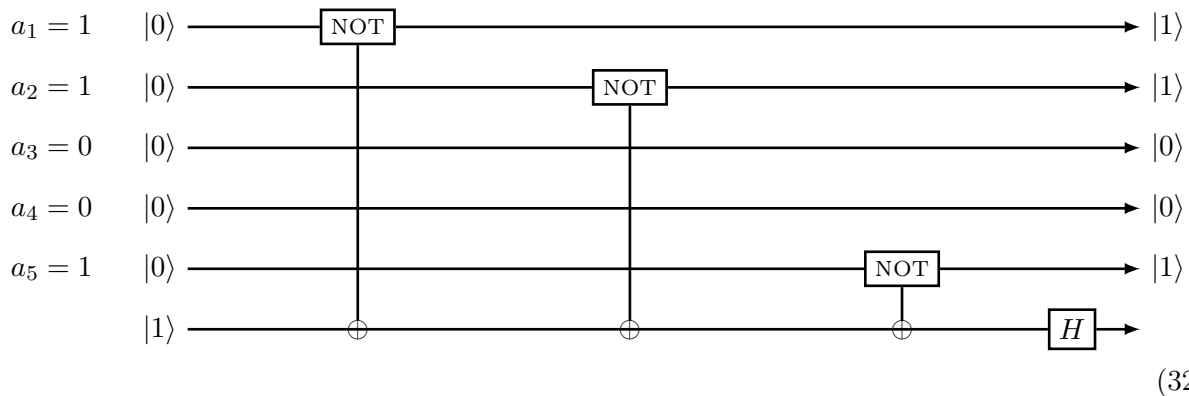
right Hadamard on this same wire to the left. After removing Hadamards in sequence and sliding the left Hadamards right and the right Hadamards left, the simplified circuit is



Now a trick. After each NOT, add a pair of Hadamards in sequence, because  $H^2 = I$ . We get,



We highlighted three 2-qubit circuits in blue dashed boxes. Each is a  $C_{\text{NOT}}$  sandwiched between Hadamards. We saw in Section 16.2 on page 67 that such a circuit is equivalent to a  $C_{\text{NOT}}$ , where the roles of the controlling bit and negated bit are interchanged. Hence, we get the equivalent circuit



For this simple circuit, one can now find the outputs of the top 5 qubits, because the controlling bit is  $|1\rangle$  and so all the NOTs are activated on input qubits  $\{1,2,5\}$  and the other input qubits are unaltered. This circuit produces  $a$  at the output. The construction generalizes to any  $n$  and  $a$ .

## 17.2 Algebraic Proof

It is an instructive exercise in manipulating Hadamards to work through the algebraic proof that the circuit in the previous section produces  $a$  on the outputs of the top  $n$  qubits. We need to analyze

$$\begin{aligned}
& H_n \left( \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} |x\rangle \right) \\
&= \frac{1}{2^{n/2}} \sum_{x \in \{0,1\}^n} (-1)^{a \cdot x} H_n |x\rangle \\
&= \frac{1}{2^{n/2}} \sum_{x_1, \dots, x_n \in \{0,1\}^n} (-1)^{a_1 x_1 \oplus a_2 x_2 \oplus \dots \oplus a_n x_n} H|x_1\rangle \otimes H|x_2\rangle \otimes \dots \otimes H|x_n\rangle
\end{aligned} \tag{325}$$

Let us consider the case  $a_1 = 0$ . We get

$$\begin{aligned}
& \frac{1}{2^{n/2}} \sum_{x_1, \dots, x_n \in \{0,1\}^n} (-1)^{a_2 x_2 \oplus \dots \oplus a_n x_n} H|x_1\rangle \otimes H|x_2\rangle \otimes \dots \otimes H|x_n\rangle \\
&= \frac{1}{2^{n/2}} \sum_{x_1 \in \{0,1\}} H|x_1\rangle \otimes \sum_{x_2, \dots, x_n \in \{0,1\}^n} (-1)^{a_2 x_2 \oplus \dots \oplus a_n x_n} H|x_2\rangle \otimes \dots \otimes H|x_n\rangle \\
&= \frac{1}{2^{n/2}} H(|0\rangle + |1\rangle) \otimes \sum_{x_2, \dots, x_n \in \{0,1\}^n} (-1)^{a_2 x_2 \oplus \dots \oplus a_n x_n} H|x_2\rangle \otimes \dots \otimes H|x_n\rangle \\
&= |0\rangle \otimes \frac{1}{2^{(n-1)/2}} \sum_{x_2, \dots, x_n \in \{0,1\}^n} (-1)^{a_2 x_2 \oplus \dots \oplus a_n x_n} H|x_2\rangle \otimes \dots \otimes H|x_n\rangle.
\end{aligned} \tag{326}$$

The last step follows because  $H^2 = I$ , so  $H(|0\rangle + |1\rangle) = \sqrt{2}H^2|0\rangle = \sqrt{2}|0\rangle$ . So the first bit is in the pure state  $|0\rangle$ . Now suppose  $a_1 = 1$ . Summing over  $x_1$ , we get

$$\begin{aligned}
& \frac{1}{2^{n/2}} \sum_{x_2, \dots, x_n \in \{0,1\}^n} (-1)^{a_2 x_2 \oplus \dots \oplus a_n x_n} H|0\rangle \otimes H|x_2\rangle \otimes \dots \otimes H|x_n\rangle + \\
& \frac{1}{2^{n/2}} \sum_{x_2, \dots, x_n \in \{0,1\}^n} (-1)^{1 \oplus a_2 x_2 \oplus \dots \oplus a_n x_n} H|1\rangle \otimes H|x_2\rangle \otimes \dots \otimes H|x_n\rangle \\
&= \frac{1}{2^{n/2}} H(|0\rangle - |1\rangle) \otimes \sum_{x_2, \dots, x_n \in \{0,1\}^n} (-1)^{a_2 x_2 \oplus \dots \oplus a_n x_n} H|x_2\rangle \otimes \dots \otimes H|x_n\rangle \\
&= |1\rangle \otimes \frac{1}{2^{(n-1)/2}} \sum_{x_2, \dots, x_n \in \{0,1\}^n} (-1)^{a_2 x_2 \oplus \dots \oplus a_n x_n} H|x_2\rangle \otimes \dots \otimes H|x_n\rangle.
\end{aligned} \tag{327}$$

The last step is because  $H(|0\rangle - |1\rangle) = \sqrt{2}H^2|1\rangle = \sqrt{2}|1\rangle$ . This time, the first bit is in the pure state  $|1\rangle$ . We have shown that the first bit is in the pure state  $|a_1\rangle$ . The remaining sum is exactly of the same form as the full sum, but over one fewer bit. It follows by induction that every output qubit  $j$  is in the pure state  $|a_j\rangle$ . Measuring the  $n$  output qubits gives  $|a_1\rangle \otimes |a_2\rangle \otimes \dots \otimes |a_n\rangle$ .

## 18 The Search Problem

You have a container of objects, for example a list of  $2^n$  elements  $L = \{z_1, z_2, \dots, z_{2^n}\}$  and some target element  $z_*$ . The task is to find the element  $z_*$  in  $L$ , or equivalently to return the index of the element  $z_*$  in  $L$ . We can formulate this task using Boolean functions. Let  $x$  index the elements,  $x$  has  $n$  bits. Define the function

$$f(x) = \begin{cases} 1 & \text{if } z_x = z_*; \\ 0 & \text{if } z_x \neq z_*. \end{cases} \quad (329)$$

The goal is to find an  $x$  for which  $f(x) = 1$ . So, we can formulate the general task of search using Boolean functions. Given a black-box function  $f(x)$ , find an  $x$  for which  $f(x) = 1$ . Here is an example. Given a graph as an edge list,  $G = e_{1,2}e_{1,3} \cdots e_{n-1,n}$  define the function  $f(x|G, K)$  with parameters  $G$  and  $K \in \mathbb{N}$ . The function takes an  $n$  bit input  $x = x_1 \cdots x_n$  that identifies a vertex set. The function outputs 1 if and only if  $x$  is a clique in  $G$  of size at least  $K$ . We saw such “certifiers” before when we talked about NP-completeness. Finding an  $x$  for which  $f(x|G, K) = 1$  is equivalent to finding a clique in  $G$  of size at least  $K$ .

### 18.1 Searching for a Unique Element

We first consider a simple version of the problem. There is exactly one  $x_*$  on which evaluates to 1. We have access to a black-box function  $f(x)$

$$f(x) = \begin{cases} 1 & \text{if } x = x_*; \\ 0 & \text{if } x \neq x_*. \end{cases} \quad (330)$$

The classical algorithm evaluates  $f(x)$  on each of the  $2^n$  possible  $x$  to find  $x_*$ . The worst case runtime is  $2^n$ . A randomized algorithm which samples  $M$  of the  $x_i$  has a success probability  $M/2^n$ . A success probability of  $1/2$  can be achieved with  $2^n/2$  evaluations of  $f$ .

### 18.2 Quantum Circuit for $f$

For a quantum search algorithm, we are given a quantum circuit  $U_f$  that implements  $f(x)$ .

$$\begin{array}{ccc} |x\rangle & \xrightarrow{\quad} & \boxed{U_f} & \xrightarrow{\quad} & |x\rangle \\ |y\rangle & \xrightarrow{\quad} & & \xrightarrow{\quad} & |y \oplus f(x)\rangle \end{array} \quad (331)$$

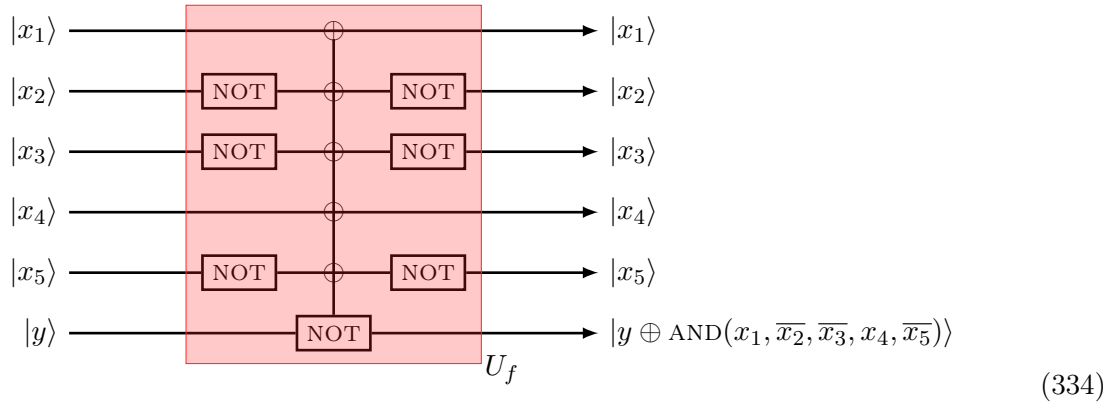
One can always specify the function using a small quantum circuit. This does not mean the function itself is given as a small circuit, only that it is possible to do so. To specify  $f$ , it suffices to specify  $x_*$ . Let us take an example with  $x_* = 10010$ , that is,

$$f(x) = \text{AND}(x_1, \overline{x_2}, \overline{x_3}, x_4, \overline{x_5}). \quad (332)$$

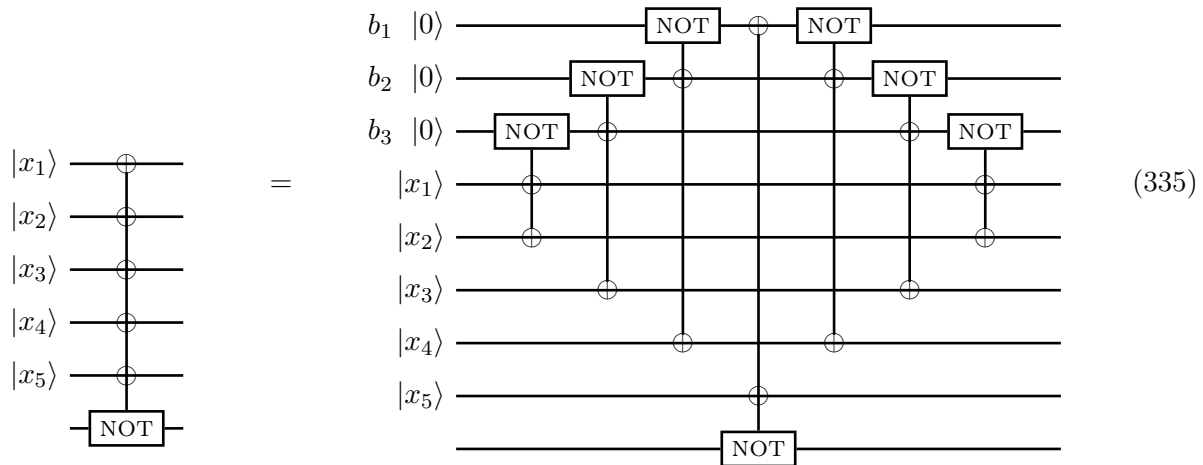
We need a circuit for  $U_f$  that implements

$$\begin{array}{ccc} |x\rangle & \xrightarrow{\quad} & \boxed{U_f} & \xrightarrow{\quad} & |x\rangle \\ |y\rangle & \xrightarrow{\quad} & & \xrightarrow{\quad} & |y \oplus \text{AND}(x_1, \overline{x_2}, \overline{x_3}, x_4, \overline{x_5})\rangle \end{array} \quad (333)$$

So,  $U_f$  needs to implement a controlled-NOT of  $|y\rangle$ , controlled by  $\text{AND}(x_1, \bar{x}_2, \bar{x}_3, x_4, \bar{x}_5)$ ,



The circuit above is nice and compact, but it does not use standard gates. You cannot ask your hardware team to manufacture quantum gates on demand. We need a circuit that uses only standard gates, for example c-NOT, cc-NOT (Toffoli), Hadamard, etc. We need to implement a massive AND of  $|x_1\rangle, \dots, |x_5\rangle$  and use this to control the NOT on  $|y\rangle$ . If we have auxiliary qubits, we can do this using singly and doubly controlled gates. These auxiliary qubits are sometimes called ancillary qubits. Here is a simple way to do this.



Let's analyze the circuit equivalence above to make sure it works. Note that the ccNOT gates are just Toffoli gates. For classical inputs, the first NOT on ancillary qubit  $b_3$  emits  $|1\rangle$  if and only if  $|x_1\rangle = |x_2\rangle = |1\rangle$ . Then,  $b_2$ 's NOT emits  $|1\rangle$  if and only if  $b_3$  and  $|x_3\rangle$  are both  $|1\rangle$  which is if and only if  $|x_1\rangle = |x_2\rangle = |x_3\rangle = |1\rangle$ . Then,  $b_1$ 's NOT emits  $|1\rangle$  if and only if  $b_2$  and  $|x_4\rangle$  are both  $|1\rangle$  which is if and only if  $|x_1\rangle = |x_2\rangle = |x_3\rangle = |x_4\rangle = |1\rangle$ . Finally, the bottom not is controlled if and only if  $b_1$  and  $|x_5\rangle$  are both  $|1\rangle$  which is if and only if  $|x_1\rangle = |x_2\rangle = |x_3\rangle = |x_4\rangle = |x_5\rangle = |1\rangle$ , as desired. Note, the remaining three Toffoli gates after the bottom not simply undo the actions of the first three Toffoli gates as the Toffoli gate is its own inverse.

The circuit for  $U_f$  can be compact. That does not mean the circuit given as a black-box will be implemented in the most compact way. Note also that ancillary qubits are costly to initialize and maintain in a stable state. The construction above needs  $n - 2$  ancillary qubits. As a challenge, try to implement this multiply controlled not using just one ancillary qubit.

### 18.3 Quantum Search – Warm Up

Let's start with the following setup.

$$|\phi\rangle = \frac{1}{2^{n/2}} \sum_{|x\rangle \in \{0,1\}^n} |x\rangle \otimes |f(x)\rangle \quad (336)$$

By now, we are expert in the analysis that leads to the expression for  $|\phi\rangle$  above,

$$|\phi\rangle = U_f(H_n|0\rangle_n \otimes |0\rangle) \quad (337)$$

$$= U_f\left(\frac{1}{2^{n/2}} \sum_x |x\rangle \otimes |0\rangle\right) \quad (338)$$

$$= \frac{1}{2^{n/2}} \sum_x U_f(|x\rangle \otimes |0\rangle) \quad (339)$$

$$= \frac{1}{2^{n/2}} \sum_x |x\rangle \otimes |f(x)\rangle. \quad (340)$$

This is great, in that  $|\phi\rangle$  contains information about  $f(x)$  on all  $x$ , but if we measure, the state will collapse uniformly randomly to some pure state  $|x_0\rangle \otimes |f(x_0)\rangle$ . The amplitudes of every pure state are exactly the same,  $1/2^{n/2}$ . This is just random guess and check. The probability to get the correct  $x_*$  is  $1/2^n$ , exactly the same as classical guess and check.

Here is a second try. We leave you to derive the output state for this familiar setup.

$$\begin{aligned} & \frac{1}{2^{n/2}} \sum_x |x\rangle \\ & \frac{|0\rangle - |1\rangle}{\sqrt{2}} \end{aligned} \quad (341)$$

Now, all the information about  $f$  is contained in the top  $n$  qubits. The amplitude for each pure state is  $(-1)^{f(x)}/2^{n/2}$ . The amplitude for  $|x_*\rangle$  is  $-1/2^{n/2}$ . The amplitude for all the other  $|x\rangle$  is  $1/2^{n/2}$ . This is good. There is an asymmetry between  $x_*$  and all the other  $x$ . However, if we measure the top  $n$  bits, we still have the same problem as guess and check. The probabilities are  $1/2^n$  for all  $x$ , so we get a random  $|x_0\rangle$ . It's worse, because we don't even recover the value  $f(x_0)$ . So far, we haven't been able to do any better than random guess and check. Our short term goal is to find some way to do better than random guess and check with one function evaluation. Anything better, no matter how small the improvement, will be a breakthrough.

Here is an example. Suppose  $x_* = 10$ , so

$$f(00) = 0 \quad f(01) = 0 \quad f(10) = 1 \quad f(11) = 0. \quad (342)$$

and

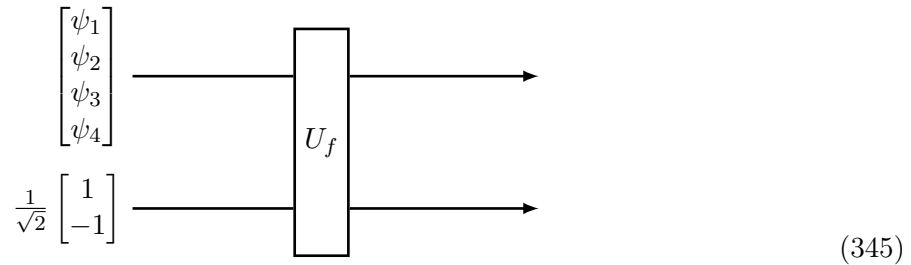
$$U_f = \begin{bmatrix} \boxed{I_2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} \\ 0_{2 \times 2} & \boxed{I_2} & 0_{2 \times 2} & 0_{2 \times 2} \\ 0_{2 \times 2} & 0_{2 \times 2} & \boxed{\text{NOT}} & 0_{2 \times 2} \\ 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & \boxed{I_2} \end{bmatrix} \quad (343)$$

The output state for this  $U_f$  is

$$|\phi\rangle = \begin{bmatrix} 1/2 \\ 1/2 \\ -1/2 \\ 1/2 \end{bmatrix} \otimes \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}. \quad (344)$$

As we already said, measuring the two two qubits give a uniform distribution over all pure states. But all is not lost, because we have introduced an asymmetry for the state  $|10\rangle$ . Suppose we could somehow subtract  $1/2$  from each amplitude for the top qubits. Then we would get  $[0, 0, -1, 0]^T$ . Now if we measure, we are golden. The result will be  $|10\rangle$  with probability 1. It turns out we will be able to do something like that, which is what Grover's iteration accomplishes. But for that you have to stay tuned to the next lecture.

A crucial ingredient in Grover's iteration is the following operator. Let's analyze it.



This operator is just  $U_f$ , but with its bottom bit set to  $(|0\rangle - |1\rangle)/\sqrt{2}$ . So, this is a linear operator on the top two bits. This discussion will generalize to  $n$ -bit functions, even if the function evaluates to 1 on more than one input. Let us compute the output. We want  $U_f(|\psi\rangle \otimes \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix})$ ,

$$\begin{bmatrix} \boxed{I_2} & 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} \\ 0_{2 \times 2} & \boxed{I_2} & 0_{2 \times 2} & 0_{2 \times 2} \\ 0_{2 \times 2} & 0_{2 \times 2} & \boxed{\text{NOT}} & 0_{2 \times 2} \\ 0_{2 \times 2} & 0_{2 \times 2} & 0_{2 \times 2} & \boxed{I_2} \end{bmatrix} \begin{bmatrix} \psi_1 \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \\ \psi_2 \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \\ \psi_3 \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \\ \psi_4 \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \end{bmatrix} \quad (346)$$

The highlighted red NOT only operates on the  $\psi_3$  component. Its effect is to flip the top and bottom

component, which is equivalent to multiplying by  $-1$ . The result is

$$U_f \left( |\psi\rangle \otimes \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \right) = \begin{bmatrix} \psi_1 \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \\ \psi_2 \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \\ \psi_3 \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix} \\ \psi_4 \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \psi_1 \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \\ \psi_2 \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \\ -\psi_3 \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \\ \psi_4 \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \psi_1 \\ \psi_2 \\ -\psi_3 \\ \psi_4 \end{bmatrix} \otimes \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}.$$

The end effect of this operator is quite simple. It leaves the bottom qubit unaltered, and flips the sign of the amplitude for the component corresponding to  $x_*$ , the pure state on which  $f$  evaluates to 1. This is a general fact about this operator. If  $f$  had  $m$  solutions to  $f(x) = 1$ , then this operator, treated as an operator on the state of the top  $n$  qubits, flips the sign only of those components corresponding to the inputs  $x_*$  for which  $f(x_*) = 1$ , leaving the bottom qubit unaltered. This is a useful operator that can be invoked anytime we wish to flip the sign of a special few components, the ones corresponding to the pure states for which  $f = 1$ .

It is a useful exercise to consider the following  $f$ ,

$$f(00) = 1 \quad f(01) = 0 \quad f(10) = 1 \quad f(11) = 0. \quad (347)$$

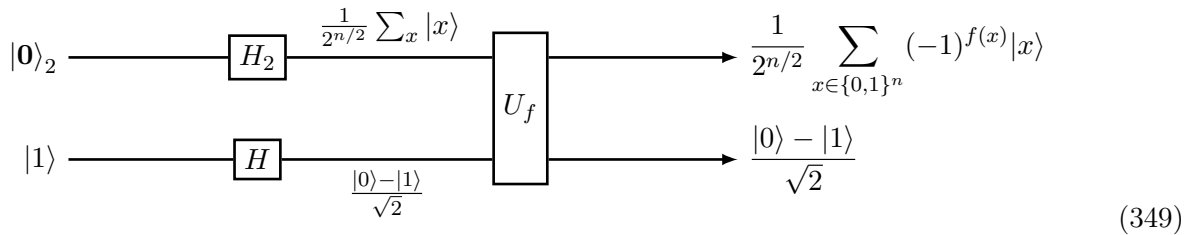
Construct  $U_f$  and show that the operator in (345) flips the sign of two components of  $|\psi\rangle$ .



## 19 Grover's Iteration

We are considering this setup for a 2-bit function  $f$ ,

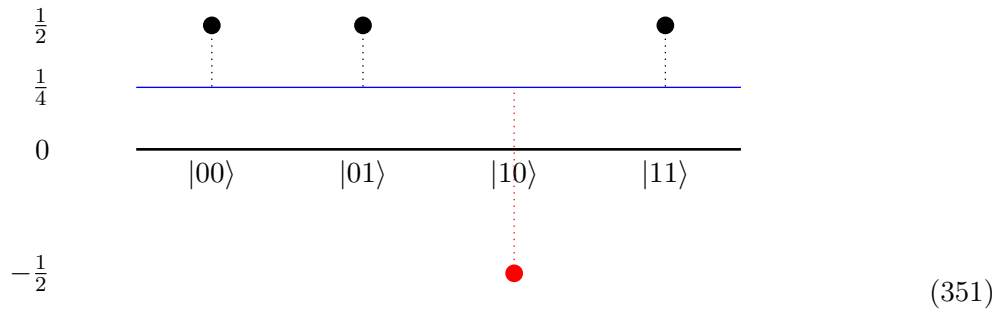
$$f(00) = 0 \quad f(01) = 0 \quad f(10) = 1 \quad f(11) = 0, \quad (348)$$



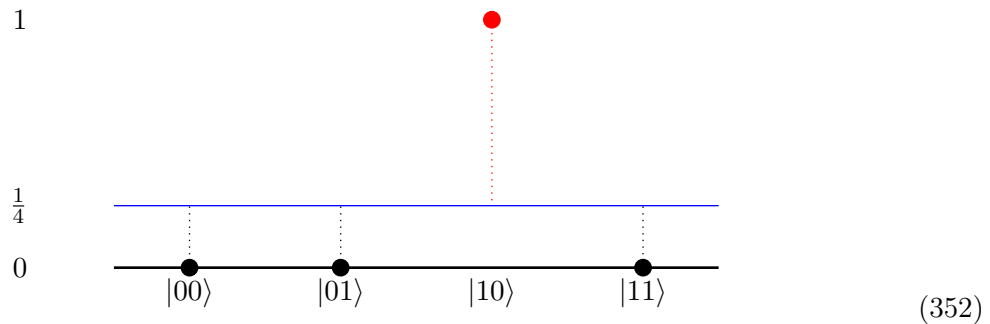
The state of the top qubits is

$$\begin{bmatrix} 1/2 \\ 1/2 \\ -1/2 \\ 1/2 \end{bmatrix}. \quad (350)$$

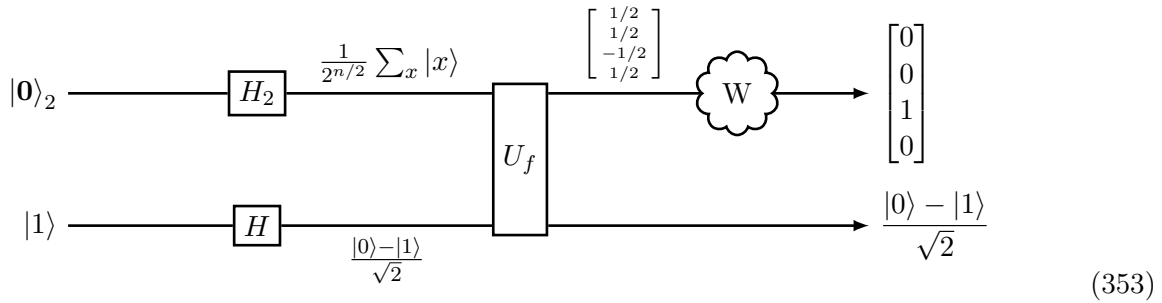
Measuring the top qubits gives a random pure state. We successfully find  $x_* = 10$  with probability  $1/4$ . Our goal is to amplify the amplitude for  $|10\rangle$  so that the measurement will yield  $x_*$  with a probability greater than  $1/4$ . No matter how small the improvement above  $1/4$ , it is a breakthrough. Here is Grover's great insight, illustrated with a pictorial view of these four amplitudes.



The blue line is the average amplitude. Since most amplitudes are  $1/2$ , the average (blue line) is positive and the distance from the positive amplitudes to the average is smaller than from the negative amplitude for  $x_*$  to the average. Suppose we reflect all the amplitudes around the average.



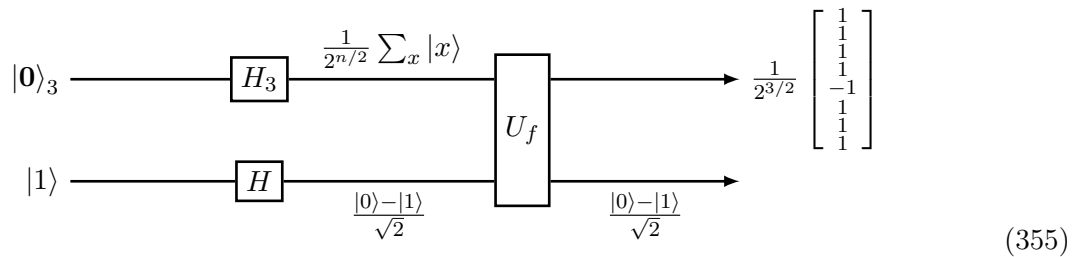
A miracle has occurred. The amplitude for  $|10\rangle$  is 1, and all other amplitudes are 0. A measurement yields  $|x_*\rangle$  with probability 1. Let's call this wierd operation of reflecting the amplitudes around the average  $W$ . The situation after applying applying  $W$  is summarized in the picture below.



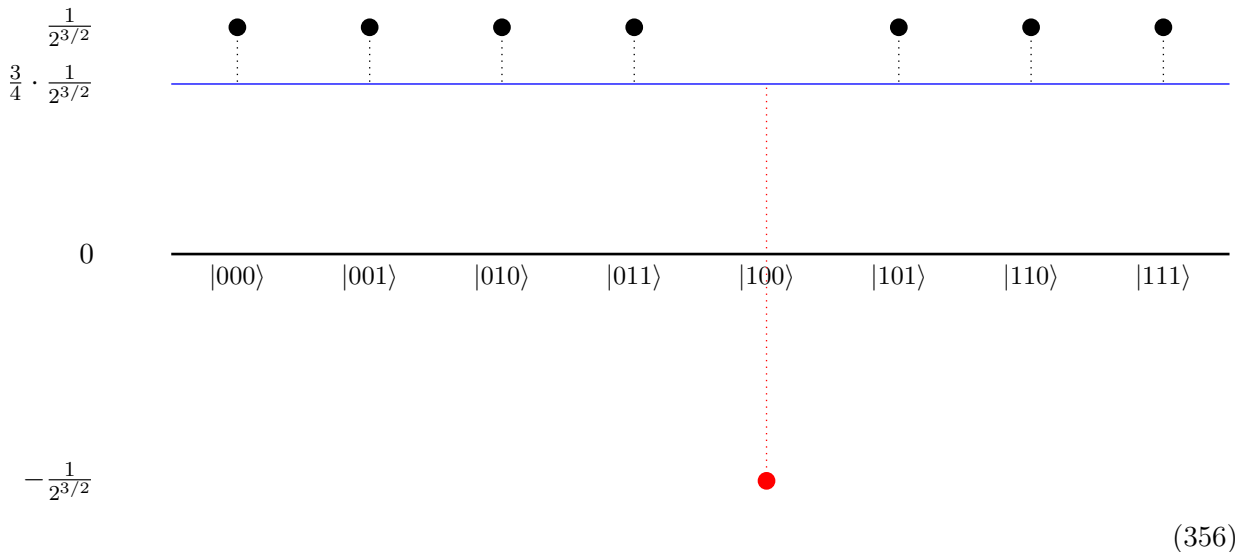
The situation is similar if  $n > 2$ . Let's consider  $n = 3$  with  $x_* = 100$ , so

$$f(000) = 0, \quad f(001) = 0, \quad f(010) = 0, \quad f(011) = 0, \quad f(100) = 1, \quad f(101) = 0, \quad f(110) = 0, \quad f(111) = 0. \quad (354)$$

Here is the situation before we apply our wierd operation  $W$ ,

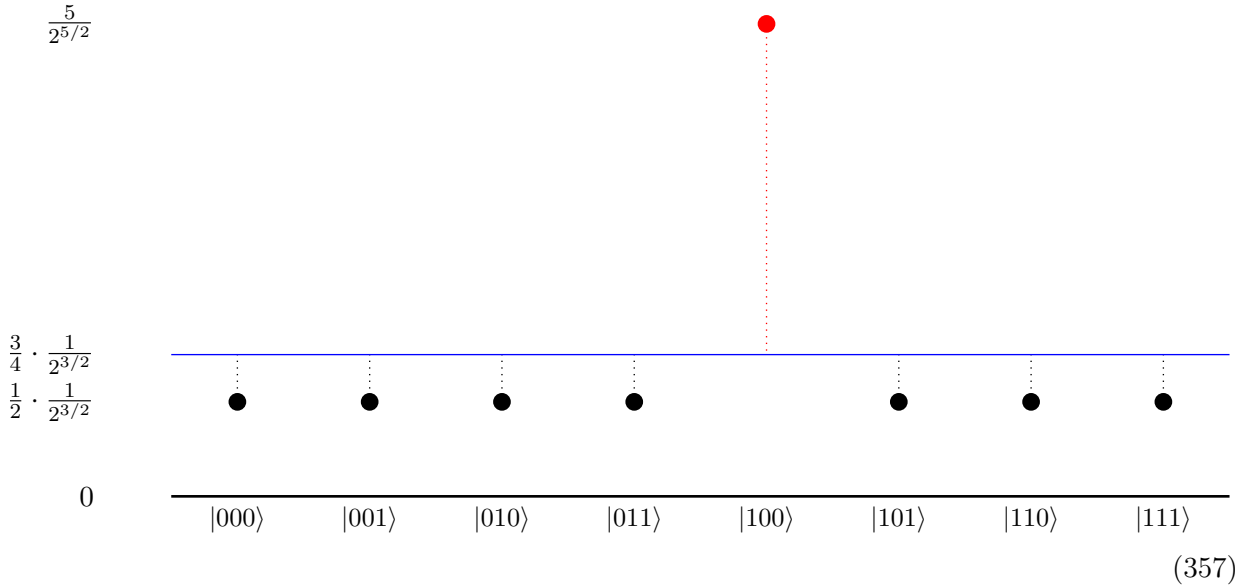


Pictorially, the amplitudes before applying  $W$  are:



The black amplitudes are at  $1/2^{3/2}$ , and the red one is at  $-1/2^{3/2}$ . The average (blue line) is at  $3/2^{7/2}$ , which is much closer to the black amplitudes (relatively speaking) than for the case  $n = 2$ .

To apply  $W$ , we reflect all the amplitudes about the average blue line. The results is



The probability to measure  $|100\rangle$  after one use of  $U_f$  is

$$\mathbb{P}[\text{success}] = \left(\frac{5}{2^{5/2}}\right)^2 = \frac{25}{32} = 0.78125 \gg \frac{1}{8}. \tag{358}$$

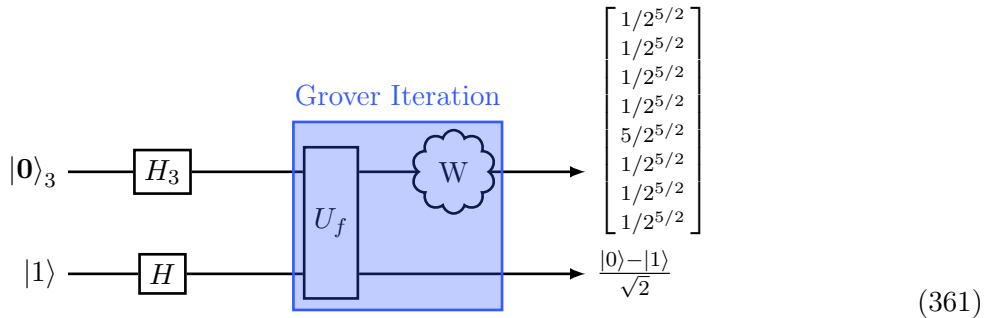
It is not as big of a miracle as with the  $n = 2$  case, where the probability of success was 1. For the general  $n$ -bit function, the situation is similar. After applying  $U_f$ , all amplitudes are  $1/2^{n/2}$ , except for the solution  $|x_*\rangle$  whose amplitude is  $-1/2^{n/2}$ . The average is

$$\frac{1}{2^n} \times \frac{2^n - 2}{2^{n/2}}. \tag{359}$$

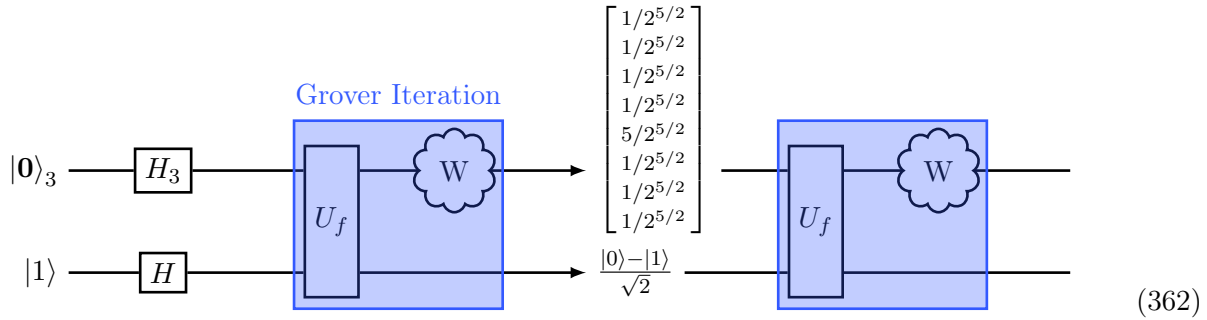
Reflecting the amplitude of  $x_*$  about this average gives

$$\mathbb{P}[\text{success}] = \frac{1}{2^n} \left(3 - \frac{4}{2^n}\right)^2. \tag{360}$$

When  $n$  gets large, the probability of success is approximately  $9/2^n$ , which is miniscule. You might think this is nothing to write home about, but it is a breakthrough, because it is 9 times bigger than random guess and check! Applying  $U_f$  followed by  $W$  is one Grover iteration,



What if we apply a second Grover Iteration,



We leave it to the reader to show that the state for the top 3 qubits is

$$\begin{bmatrix}
 -1/2^{7/2} \\
 -1/2^{7/2} \\
 -1/2^{7/2} \\
 -1/2^{7/2} \\
 11/2^{7/2} \\
 -1/2^{7/2} \\
 -1/2^{7/2} \\
 -1/2^{7/2}
 \end{bmatrix} . \tag{363}$$

The probability of success is now

$$\mathbb{P}[\text{success}] = \left( \frac{11}{2^{7/2}} \right)^2 = \frac{121}{128} \approx 0.95, \tag{364}$$

and we are in business. We can go wild and continue doing Grover iterations. The obvious cost is more evaluations of  $U_f$ , but something worse happens. We leave it to you to show that with another Grover iteration, the probability of success goes down. There is an optimal number of Grover Iterations. In this case you can see why. The non-solution components have become negative. So those amplitudes will get larger in absolute value after flipping the sign of the  $|100\rangle$  component and then reflecting about the average. We will analyze this in the next lecture. Before that, let us examine more carefully this wierd operation W. It turns out to not be so wierd.

### 19.1 Operator for Reflecting About the Average

The operation W which reflects about the average is an important technique for amplifying the amplitude of specific components of the state. Starting from state  $|\psi\rangle$ , reflecting about the average transforms each component  $\psi_i$  as follows,

$$\psi_i \rightarrow \bar{\psi} - (\psi_i - \bar{\psi}) = 2\bar{\psi} - \psi_i. \tag{365}$$

In vector form,

$$W|\psi\rangle = 2\bar{\psi} - |\psi\rangle. \tag{366}$$

Averaging is a linear operation, so  $W$  is a linear operation. Indeed,  $\bar{\psi} = \mathbf{1}^T |\psi\rangle / 2^n$ , and we have

$$W|\psi\rangle = \frac{2}{2^n} \mathbf{1} \mathbf{1}^T |\psi\rangle - |\psi\rangle, \quad (367)$$

from which we identify the operator  $W$  as

$$W = \frac{2}{2^n} \mathbf{1} \mathbf{1}^T - I. \quad (368)$$

$W$  is hermitian, and even more, it is unitary,

$$W^\dagger W = \left( \frac{2}{2^n} \mathbf{1} \mathbf{1}^T - I \right) \left( \frac{2}{2^n} \mathbf{1} \mathbf{1}^T - I \right) \quad (369)$$

$$= \frac{4}{2^{2n}} \mathbf{1} \mathbf{1}^T \mathbf{1} \mathbf{1}^T - \frac{4}{2^n} \mathbf{1} \mathbf{1}^T + I \quad (370)$$

$$= \frac{4}{2^{2n}} \mathbf{1} \underbrace{(\mathbf{1}^T \mathbf{1})}_{2^n} \mathbf{1}^T - \frac{4}{2^n} \mathbf{1} \mathbf{1}^T + I \quad (371)$$

$$= \frac{4 \cdot 2^n}{2^{2n}} \mathbf{1} \mathbf{1}^T - \frac{4}{2^n} \mathbf{1} \mathbf{1}^T + I \quad (372)$$

$$= I. \quad (373)$$

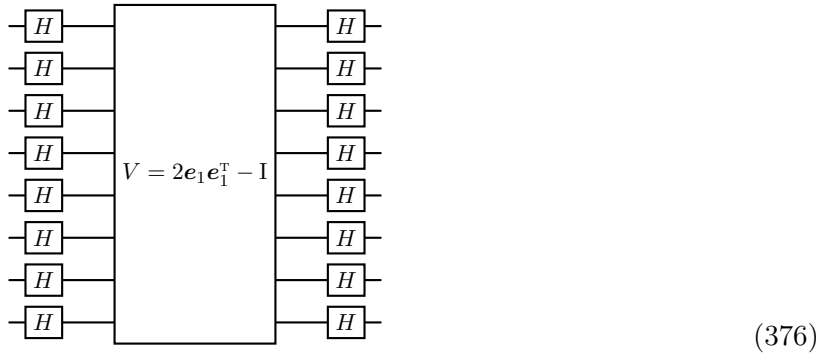
This means  $W$  can be implemented by a quantum circuit. In fact,  $W$  can be implemented by a compact circuit using the standard one and two bit gates. Let's see how. We need to first massage  $W$  into a more suitable form. Recall that  $H_n |\mathbf{0}\rangle_n = \mathbf{1} / 2^{n/2}$ . Since  $|\mathbf{0}\rangle_n$  is the first standard basis vector  $\mathbf{e}_1$ ,

$$W = 2H_n \mathbf{e}_1 \mathbf{e}_1^T H_n - I. \quad (374)$$

Using the fact that  $H_n^2 = I$ ,

$$W = H_n \underbrace{(2\mathbf{e}_1 \mathbf{e}_1^T - I)}_V H_n. \quad (375)$$

Since  $H_n = H^{\otimes n}$ , we have that  $W$  is the circuit



We now need a circuit for  $V$  to be done. It suffices to get a circuit for  $-V$  because this has the effect of just flipping the sign of all the amplitudes in  $|\psi\rangle$ , which is a benign operation to the state in quantum mechanics (it does not affect the measurement probabilities). We need a circuit for

$$-V = I - 2\mathbf{e}_1 \mathbf{e}_1^T. \quad (377)$$

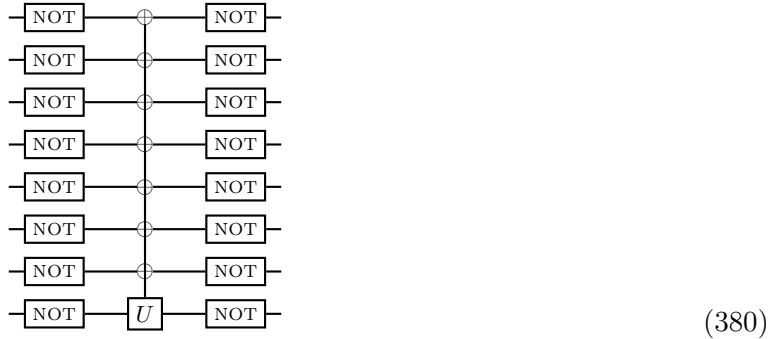
Let us first see what the operator  $-V$  looks like, as a matrix. We need to examine its action on the pure basis states  $e_i$ . Using  $e_i^T e_j = \delta_{ij}$ ,

$$\begin{cases} -V e_1 = -e_1; \\ -V e_i = e_i & i \neq 1. \end{cases} \quad (378)$$

We can now write down  $-V$  as a matrix,

$$-V = [-e_1 \ e_2 \ e_3 \ \dots \ e_{2^n}] = \begin{bmatrix} -1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}. \quad (379)$$

Recall that a matrix of the form  $\begin{bmatrix} 1 & 0 \\ 0 & U \end{bmatrix}$  is a multiply controlled- $U$ . Our  $-V$  is of the form  $\begin{bmatrix} U & 0 \\ 0 & 1 \end{bmatrix}$ , also a multiply controlled- $U$  with some minor role modifications, such as negations. The reader may wish to review Lecture 11 and compute the operator for the following circuit with  $U = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ ,



We need to analyze the action of the circuit on the pure basis states. The only non-trivial cases are when  $U$  is activated, because otherwise this circuit acts as the identity. We only need to consider the case where all the top bits are 0,  $|000 \dots 00\rangle$  and  $|000 \dots 01\rangle$ .

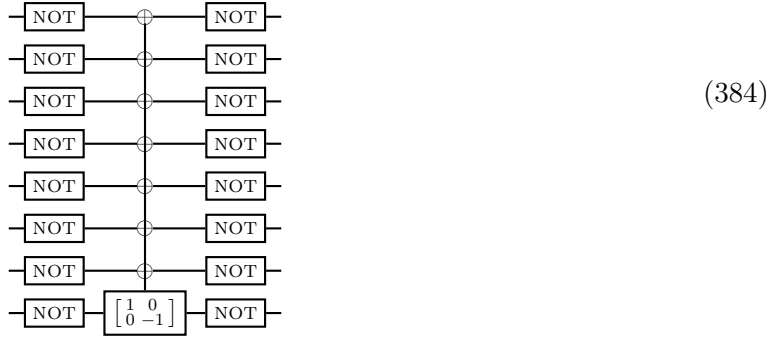
$$|0\rangle \otimes |0\rangle \otimes |0\rangle \otimes \dots \otimes |0\rangle \otimes |0\rangle \rightarrow |0\rangle \otimes |0\rangle \otimes |0\rangle \otimes \dots \otimes |0\rangle \otimes \text{NOT} \cdot U|1\rangle \quad (381)$$

$$|0\rangle \otimes |0\rangle \otimes |0\rangle \otimes \dots \otimes |0\rangle \otimes |0\rangle \rightarrow |0\rangle \otimes |0\rangle \otimes |0\rangle \otimes \dots \otimes |0\rangle \otimes \text{NOT} \cdot U|0\rangle. \quad (382)$$

Since  $\text{NOT} \cdot U|1\rangle = \begin{bmatrix} d \\ b \end{bmatrix}$  and  $\text{NOT} \cdot U|0\rangle = \begin{bmatrix} c \\ a \end{bmatrix}$ , we have that the operator for this circuit is

$$\begin{bmatrix} d & c & 0 & 0 & \dots & 0 & 0 \\ b & a & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{bmatrix}. \quad (383)$$

By inspection, we can write down the circuit for  $-V$ ,



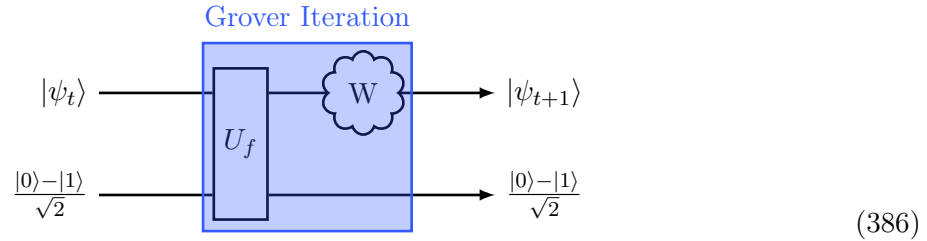
The operator being controlled is the Pauli spin- $z$  operator  $\sigma_z$ , and is considered a basic 1-qubit gate. Alternatively, you can implement it as  $H \cdot \text{NOT} \cdot H$ ,

$$\sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = H \cdot \text{NOT} \cdot H. \quad (385)$$

We can now use a construction such as in (335) on page 75 to implement the multiply controlled circuit above using standard gates. Plugging  $-V$  into (375), we get the circuit for  $W$ .

## 20 Analysis of Grover's Search Algorithm

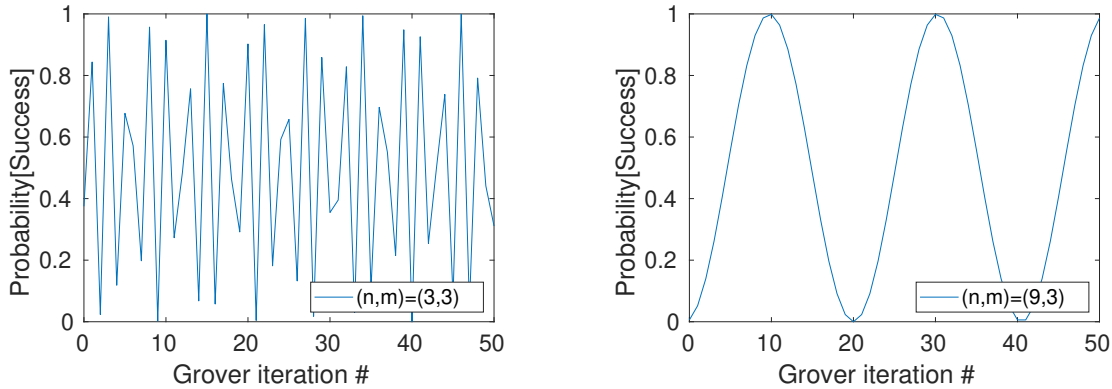
It should not surprise you that computer scientists have had a healthy dose of mathematics. You need to take your medicine because a computer is essentially a mathematical device. If you want to analyze any algorithm on any architecture, you need to do so mathematically. We are going to analyze Grover's iteration, in the general case where  $f(x) = 1$  has  $m$  solutions.



In this setup,  $U_f$  flips the sign of all the components corresponding to the pure states which are solutions to  $f(x) = 1$ . Then,  $W$  reflects all the amplitudes about the average. The Grover iteration begins with  $|\psi_0\rangle = \mathbf{1}/2^{n/2}$ . Here is an example with  $n = 3$  and  $m = 3$ . The components corresponding to solutions of  $f(x) = 1$  are in red (amplitudes are rounded).

	$ \psi_0\rangle$	$ \psi_1\rangle$	$ \psi_2\rangle$	$ \psi_3\rangle$	$ \psi_4\rangle$	$ \psi_5\rangle$	
$ 000\rangle$	$\begin{bmatrix} 0.354 \\ 0.354 \\ 0.354 \end{bmatrix}$	$\begin{bmatrix} -0.177 \\ 0.530 \\ 0.530 \end{bmatrix}$	$\begin{bmatrix} -0.442 \\ -0.088 \\ -0.088 \end{bmatrix}$	$\begin{bmatrix} -0.044 \\ -0.575 \\ -0.575 \end{bmatrix}$	$\begin{bmatrix} 0.420 \\ -0.199 \\ -0.199 \end{bmatrix}$	$\begin{bmatrix} 0.254 \\ 0.475 \\ 0.475 \end{bmatrix}$	(387)
$ 001\rangle$	$\begin{bmatrix} 0.354 \\ 0.354 \\ 0.354 \end{bmatrix}$	$\begin{bmatrix} -0.177 \\ 0.530 \\ 0.530 \end{bmatrix}$	$\begin{bmatrix} -0.442 \\ -0.088 \\ -0.088 \end{bmatrix}$	$\begin{bmatrix} -0.044 \\ -0.575 \\ -0.575 \end{bmatrix}$	$\begin{bmatrix} 0.420 \\ -0.199 \\ -0.199 \end{bmatrix}$	$\begin{bmatrix} 0.254 \\ 0.475 \\ 0.475 \end{bmatrix}$	
$ 010\rangle$	$\begin{bmatrix} 0.354 \\ 0.354 \\ 0.354 \end{bmatrix}$	$\begin{bmatrix} -0.177 \\ 0.530 \\ 0.530 \end{bmatrix}$	$\begin{bmatrix} -0.442 \\ -0.088 \\ -0.088 \end{bmatrix}$	$\begin{bmatrix} -0.044 \\ -0.575 \\ -0.575 \end{bmatrix}$	$\begin{bmatrix} 0.420 \\ -0.199 \\ -0.199 \end{bmatrix}$	$\begin{bmatrix} 0.254 \\ 0.475 \\ 0.475 \end{bmatrix}$	
$ 011\rangle$	$\begin{bmatrix} 0.354 \\ 0.354 \\ 0.354 \end{bmatrix}$	$\begin{bmatrix} -0.177 \\ 0.530 \\ 0.530 \end{bmatrix}$	$\begin{bmatrix} -0.442 \\ -0.088 \\ -0.088 \end{bmatrix}$	$\begin{bmatrix} -0.044 \\ -0.575 \\ -0.575 \end{bmatrix}$	$\begin{bmatrix} 0.420 \\ -0.199 \\ -0.199 \end{bmatrix}$	$\begin{bmatrix} 0.254 \\ 0.475 \\ 0.475 \end{bmatrix}$	
$ 100\rangle$	$\begin{bmatrix} 0.354 \\ 0.354 \\ 0.354 \end{bmatrix}$	$\begin{bmatrix} -0.177 \\ 0.530 \\ 0.530 \end{bmatrix}$	$\begin{bmatrix} -0.442 \\ -0.088 \\ -0.088 \end{bmatrix}$	$\begin{bmatrix} -0.044 \\ -0.575 \\ -0.575 \end{bmatrix}$	$\begin{bmatrix} 0.420 \\ -0.199 \\ -0.199 \end{bmatrix}$	$\begin{bmatrix} 0.254 \\ 0.475 \\ 0.475 \end{bmatrix}$	
$ 101\rangle$	$\begin{bmatrix} 0.354 \\ 0.354 \\ 0.354 \end{bmatrix}$	$\begin{bmatrix} -0.177 \\ 0.530 \\ 0.530 \end{bmatrix}$	$\begin{bmatrix} -0.442 \\ -0.088 \\ -0.088 \end{bmatrix}$	$\begin{bmatrix} -0.044 \\ -0.575 \\ -0.575 \end{bmatrix}$	$\begin{bmatrix} 0.420 \\ -0.199 \\ -0.199 \end{bmatrix}$	$\begin{bmatrix} 0.254 \\ 0.475 \\ 0.475 \end{bmatrix}$	
$ 110\rangle$	$\begin{bmatrix} 0.354 \\ 0.354 \\ 0.354 \end{bmatrix}$	$\begin{bmatrix} -0.177 \\ 0.530 \\ 0.530 \end{bmatrix}$	$\begin{bmatrix} -0.442 \\ -0.088 \\ -0.088 \end{bmatrix}$	$\begin{bmatrix} -0.044 \\ -0.575 \\ -0.575 \end{bmatrix}$	$\begin{bmatrix} 0.420 \\ -0.199 \\ -0.199 \end{bmatrix}$	$\begin{bmatrix} 0.254 \\ 0.475 \\ 0.475 \end{bmatrix}$	
$ 111\rangle$	$\begin{bmatrix} 0.354 \\ 0.354 \\ 0.354 \end{bmatrix}$	$\begin{bmatrix} -0.177 \\ 0.530 \\ 0.530 \end{bmatrix}$	$\begin{bmatrix} -0.442 \\ -0.088 \\ -0.088 \end{bmatrix}$	$\begin{bmatrix} -0.044 \\ -0.575 \\ -0.575 \end{bmatrix}$	$\begin{bmatrix} 0.420 \\ -0.199 \\ -0.199 \end{bmatrix}$	$\begin{bmatrix} 0.254 \\ 0.475 \\ 0.475 \end{bmatrix}$	
$\mathbb{P}[\text{success}]$	0.375	0.844	0.023	0.990	0.119	0.677	

Success occurs if a red pure state is measured, so the success probability is the sum of squares of the red entries in the state. We plot the probability of success versus Grover iteration # below,





There are two takeaways from the example above. The first takeaway is repeated application of the Grover iteration does not monotonically increase the probability of success. The success probability oscillates. On the left, we show the the example with  $n = 3$  and  $m = 3$ . The behavior looks rather erratic. On the right, we show the case with  $n = 9, m = 3$ . As you can see things get much smoother. We wish to perform the fewest iterations (evaluations of  $U_f$ ), so we need to decide when to stop iterating. We should stop when the success probability stops increasing. In this case, after 1 iteration. This does not give the maximum probability, but one that is high enough that it can be boosted to as large a probability that we want by repeating the whole process. For example, if we run for 1 iteration and repeat three times, we get a success probability of 0.996.

The second takeaway is the red entries corresponding to solutions stay equal. This is because they start that way and  $U_f, W$  treat them symmetrically. The same is true of the black entries corresponding to non-solutions. Let  $x_t$  be the value of a red entry at iteration  $t$  and  $y_t$  the value of a black entry. The initial condition is

$$x_0 = y_0 = \frac{1}{2^{n/2}}. \quad (388)$$

The probability of success after  $t$  iterations is

$$\mathbb{P}[\text{success}] = mx_t^2. \quad (389)$$

Our first task is to find  $x_t, y_t$  after  $t$  iterations.

## 20.1 Grover's Coupled Recurrence

Here is how the vector  $\begin{bmatrix} x_t \\ y_t \end{bmatrix}$  is updated in the first step of the a Grover iteration, where  $U_f$  is applied,

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} \rightarrow \begin{bmatrix} -x_t \\ y_t \end{bmatrix} \quad (390)$$

In the second step, we apply  $W$ . The average amplitude is  $((2^n - m)y_t - mx_t)/2^n$  and each amplitude  $\psi_i \rightarrow 2\bar{\psi} - \psi_i$ , so

$$x_{t+1} = 2 \times \frac{(2^n - m)y_t - mx_t}{2^n} + x_t \quad (391)$$

$$y_{t+1} = 2 \times \frac{(2^n - m)y_t - mx_t}{2^n} - y_t. \quad (392)$$

Simplifying and defining  $q = m/2^{n-1}$ ,

$$x_{t+1} = (1 - q)x_t + (2 - q)y_t \quad (393)$$

$$y_{t+1} = -qx_t + (1 - q)y_t. \quad (394)$$

$$(395)$$

It is convenient to define the vector  $\mathbf{z}_t = \begin{bmatrix} x_t \\ y_t \end{bmatrix}$ . Then,

$$\mathbf{z}_{t+1} = \mathbf{A}\mathbf{z}_t, \quad (396)$$

where  $A$  is the matrix

$$A = \begin{bmatrix} 1 - q & 2 - q \\ -q & 1 - q \end{bmatrix}. \quad (397)$$

This is a recurrence for a vector, a generalization of a simple recurrence. Since  $A$  is not diagonal, this is a coupled recurrence, or discrete dynamical system. We will solve this recurrence using a general technique for solving such recurrences, a technique that is worth mastering and storing away for future use.

## 20.2 Solving Grover's Recurrence

### 20.3 Unknown Number of Solutions

To set the number of Grover iterations, we need to know  $m$ . For the algorithm to be viable in practice, it must handle three cases,  $m = 0$ ,  $m \in o(2^{n/2})$  and  $m \in \Omega(2^{n/2})$ . We can use standard algorithmic techniques to build on the algorithm with known  $m$  within a guess  $m$  and check framework.

## 21 Quantum Error Correction

If the quantum state of your qubits changes during a computation, the result is corrupted. This changing of the state is called decoherence, and it is inevitable because the quantum computer interacts with everything. This is also true in classical computing, but it's more benign. The interactions of a classical computer with its environment are weak in comparison with the quantum setting. Nevertheless, many weak interactions can build up to a large perturbation. Luckily, dissipative processes, e.g. energy loss as heat to the environment, quickly dampen individual weak interactions making it hard for them to buildup. Even still, we do need to worry about bit flips in classical computers, and that is why you pay the big bucks for EC-RAM (error correcting RAM). If a bit flips, we observe it and correct it. This can only be done if there is some redundancy which tells us what the original bit was. Here is a simple example of the workflow,

$$0 \xrightarrow{\text{encode}} 000 \xrightarrow[\text{error}]{\text{observe}} 010 \xrightarrow[\text{correct}]{\text{error}} 000 \quad (398)$$

The classical bit is encoded into 3 identical bits, providing redundancy. We periodically observe the bits at a frequency higher than the rate of bit-flips due to the environment (e.g. from cosmic rays). The rate of observation must be high enough to ensure that at most one bit-flip will occur between observations. When an error is observed, it is corrected to the majority bit. Can we implement the same workflow for protecting the integrity of qubits. The first difference in the quantum setting is that a general qubit can be in a superposition,

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle. \quad (399)$$

This means there are a continuum of possible errors. Any change in  $\alpha, \beta$  constitutes an error. A change from  $(|0\rangle + |1\rangle)/\sqrt{2}$  to  $(|0\rangle - |1\rangle)/\sqrt{2}$  could be disastrous in a quantum algorithm, even though it does not materially affect the state because the measurement probabilities are unchanged. To do quantum error-correction, we aim for something like

$$|\psi\rangle \xrightarrow{\text{encode}} |\psi\rangle \otimes |\psi\rangle \otimes |\psi\rangle \xrightarrow[\text{error}]{\text{observe}} |\psi\rangle \otimes |\psi'\rangle \otimes |\psi\rangle \xrightarrow[\text{correct}]{\text{error}} |\psi\rangle \otimes |\psi\rangle \otimes |\psi\rangle \quad (400)$$

There are two challenges here, and these are fundamental challenges because they are at the heart of quantum mechanics. The first step, encoding, is quantum state cloning. The last step also looks like state cloning, because we are copying  $|\psi\rangle$  onto  $|\psi'\rangle$ . You cannot clone quantum states. Even worse, how do you observe the error? You won't know that it is the middle qubit that changed. Any of the qubits could have changed. So you have to "measure" all three qubits. When you measure, the state collapses to one of the 8 possible pure states  $|000\rangle, \dots, |111\rangle$ . If there has been no corruption, you will observe  $|000\rangle$  with probability  $\|\alpha\|^2$ . Suppose you do observe  $|000\rangle$ . Now what? You have no idea what the original state was. All information on  $\alpha, \beta$  is lost. So not only does measuring the state to detect the error destroy the state. The result of the measurement cannot even tell us if there was an error. Measuring  $|000\rangle$ , only tell us the original state had nonzero projection on  $|000\rangle$ .

- There are a continuum of possible errors resulting from coupling to *anything* in the universe.
- We cannot check for error because measuring the state destroys it.
- Even if we could check for error, we restore the original state via a copy because of no-cloning.

These roadblocks spell doom for quantum error correction. Here is what we need to have any chance:

- Ability to build in redundancy without cloning.
- Ability to check for errors without actually measuring the state.
- Ability to correct any errors without cloning.

For quantum computing to become reality, we need radically new breakthrough ideas.

## 21.1 Quantum Redundancy

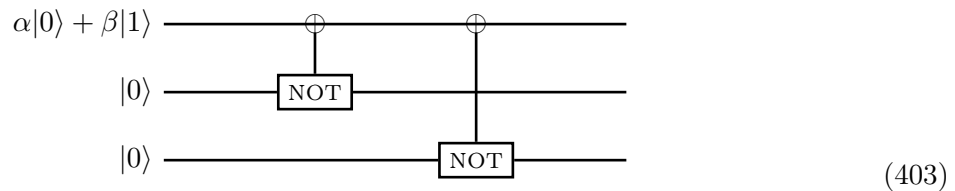
Luckily some of these breakthrough ideas have been developed. Let's begin with quantum redundancy. The key idea is that instead of copying the entire state, which we cannot do because of no-cloning, we are going to entangle the state with auxilliary qubits, to ensure that information about the state is contained in multiple qubits. Ideally, we would like, but can't have:

$$|\psi\rangle \rightarrow |\psi\rangle \otimes |\psi\rangle \otimes |\psi\rangle. \quad (401)$$

We can only clone classical bits,  $|0\rangle \rightarrow |000\rangle$  and  $|1\rangle \rightarrow |111\rangle$ . How about a general version of this,

$$\alpha|0\rangle + \beta|1\rangle \rightarrow \alpha|000\rangle + \beta|111\rangle? \quad (402)$$

This is a 3-qubit encoding of the original state that emphasizes the superposition by enforcing it on all three qubits "simultaneously". If you measure, all three bits will be the same. So, the measured pure state is cloned. Since the output is a 3-qubit state, and quantum operators are unitary, the input must be three qubits, the other two qubits start in state  $|0\rangle$ . Here is a circuit for this encoding,



One way to verify that the circuit works and produces  $\alpha|000\rangle + \beta|111\rangle$  is to compute the unitary operator  $U$  for this circuit by computing its action on the basis states. Apply  $U$  it to the starting state and verify that you get the desired state. We ask you to compute  $U$  and verify

$$\left[ \begin{array}{c} U \end{array} \right] \left[ \begin{array}{c} \alpha \\ 0 \\ 0 \\ 0 \\ \beta \\ 0 \\ 0 \\ 0 \end{array} \right] = \left[ \begin{array}{c} \alpha \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \beta \end{array} \right] \quad (404)$$

We will reason directly from the circuit's gates. Label the first controlled-NOT as  $\text{cNOT}_{12}$  to indicate it is qubit 1 controlling a NOT on qubit 2. Similarly label the second controlled-NOT as  $\text{cNOT}_{13}$ . These operators are linear. By linearity of tensor product, the starting state is

$$(\alpha|0\rangle + \beta|1\rangle) \otimes |0\rangle \otimes |0\rangle = \alpha|0\rangle \otimes |0\rangle \otimes |0\rangle + \beta|1\rangle \otimes |0\rangle \otimes |0\rangle \quad (405)$$

Applying the  $\text{cNOT}_{12}$  and using linearity,

$$\text{cNOT}_{12}(\alpha|0\rangle \otimes |0\rangle \otimes |0\rangle + \beta|1\rangle \otimes |0\rangle \otimes |0\rangle) \quad (406)$$

$$= \alpha \text{cNOT}_{12}(|0\rangle \otimes |0\rangle \otimes |0\rangle) + \beta \text{cNOT}_{12}(|1\rangle \otimes |0\rangle \otimes |0\rangle) \quad (407)$$

$$= \alpha|0\rangle \otimes |0\rangle \otimes |0\rangle + \beta|1\rangle \otimes |1\rangle \otimes |0\rangle \quad (408)$$

The first term is because the controlling bit is 0 and so the  $\text{cNOT}$  acts identity on the 2nd bit. The second term is because the controlling bit is 1 so the  $\text{cNOT}$  flips the 2nd bit. Now apply the  $\text{cNOT}_{13}$  and again use linearity to get

$$\text{cNOT}_{13}(\alpha|0\rangle \otimes |0\rangle \otimes |0\rangle + \beta|1\rangle \otimes |1\rangle \otimes |0\rangle) \quad (409)$$

$$= \alpha \text{cNOT}_{13}(|0\rangle \otimes |0\rangle \otimes |0\rangle) + \beta \text{cNOT}_{13}(|1\rangle \otimes |1\rangle \otimes |0\rangle) \quad (410)$$

$$= \alpha|0\rangle \otimes |0\rangle \otimes |0\rangle + \beta|1\rangle \otimes |1\rangle \otimes |1\rangle, \quad (411)$$

as desired. We can clone the classical bits not only when they are in isolation, but also when they are in a superposition, as is the case for a general quantum state. Since 3 qubits are used in the encoding, this is called a 3-qubit code.

## 21.2 Modeling the Error

There are three types of error that can occur to a qubit's state. This is because any single qubit error is caused by evolution under some  $2 \times 2$  unitary matrix,  $E$ . A basis for  $2 \times 2$  matrices is

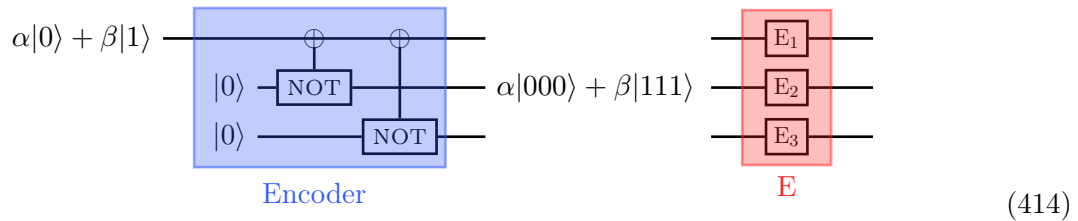
$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \quad (412)$$

so  $E$  can be written as a linear combination of these matrices,

$$E = (1 - \epsilon)I + \epsilon_x \sigma_x + \epsilon_y \sigma_y + \epsilon_z \sigma_z \quad (413)$$

The error is caused by the  $\epsilon_x \sigma_x + \epsilon_y \sigma_y + \epsilon_z \sigma_z$  term. The first of these, the  $\sigma_x$  term, is bit-flip (note that  $\sigma_x = \text{NOT}$ ). The third, the  $\sigma_z$  term, is a phase flip. The second, the  $\sigma_y$  term, is a combined phase and bit flip. To keep things simple while illustrating all the main ideas, we will only focus on the bit flip error. All the main ideas extend to the more general error, and you can read about it in more complete treatments. The main change is that you need to increase redundancy by encoding a qubit using more than just 3 qubits.

To model this simple bit-flip error, we will assume that to one of the qubits in our encoded state is applied at most one NOT. The error operator for our setting is illustrated in the circuit below,



Three 1-qubit error operators  $E_1, E_2, E_3$  operate independently on each qubit of the encoded state  $\alpha|000\rangle + \beta|111\rangle$ . Each  $E_i$  is either I or NOT, and at most one of the  $E_i$  is not I. One of four states can result after the error. If all  $E_i$  are I, then  $E = I$  and the resulting state is the uncorrupted state,

$$|\psi_0\rangle = \alpha|000\rangle + \beta|111\rangle. \quad (415)$$

If  $E_1 = \text{NOT}$ , then  $E = \text{NOT} \otimes I \otimes I$ . We know  $E$ 's action on pure states, it just flips the first bit. We use linearity to get  $E$ 's action on the encoded state,

$$E(\alpha|000\rangle + \beta|111\rangle) = \alpha E(|000\rangle) + \beta E(|111\rangle) = \alpha|100\rangle + \beta|011\rangle. \quad (416)$$

Similarly, if  $E_2$  or  $E_3$  are NOT, the corrupted states are  $\alpha|010\rangle + \beta|101\rangle$  or  $\alpha|001\rangle + \beta|110\rangle$  respectively. The four options for the corrupted state are

$$\begin{aligned} |\psi_0\rangle &= \alpha|000\rangle + \beta|111\rangle & E &= I \otimes I \otimes I \\ |\psi_1\rangle &= \alpha|100\rangle + \beta|011\rangle & E &= \text{NOT} \otimes I \otimes I \\ |\psi_2\rangle &= \alpha|010\rangle + \beta|101\rangle & E &= I \otimes \text{NOT} \otimes I \\ |\psi_3\rangle &= \alpha|001\rangle + \beta|110\rangle & E &= I \otimes I \otimes \text{NOT} \end{aligned} \quad (417)$$

No pure state occurs in more than one of the corrupted states. This means these corrupted states are pairwise orthogonal.<sup>5</sup> So now, if you measure and (say) get  $|100\rangle$ , you know there has been an error. We have a fighting chance. You even know it was a bit flip in the first qubit of the encoded state. In contrast, if you measure the un-encoded state and get (say)  $|0\rangle$ , you don't know anything. We can at least detect the error. However, the state is destroyed, and all information on  $\alpha, \beta$  is lost. Detecting the error in this way is not going to allow for error-correction.

Our ability to detect the error crucially depends on the possible corrupted states being orthogonal. That way, the result of the measurement uniquely identifies not just the error, but the type of error. This concept will generalize to other types of errors. This requirement that the possible corrupted states should be pairwise orthogonal can only be accomplished if the encoding uses sufficiently many qubits. We can see this as follows. We start with a superposition of two pure states, so after a possible bit-flip the possible corrupted state will be a superposition of two pure states. Orthogonality requires that no pure state be repeated in any two of the four possible corrupted states, hence we need at least 8 pure states in the encoding. This means the encoded state must use at least 3 qubits which provides  $2^3 = 8$  pure states.

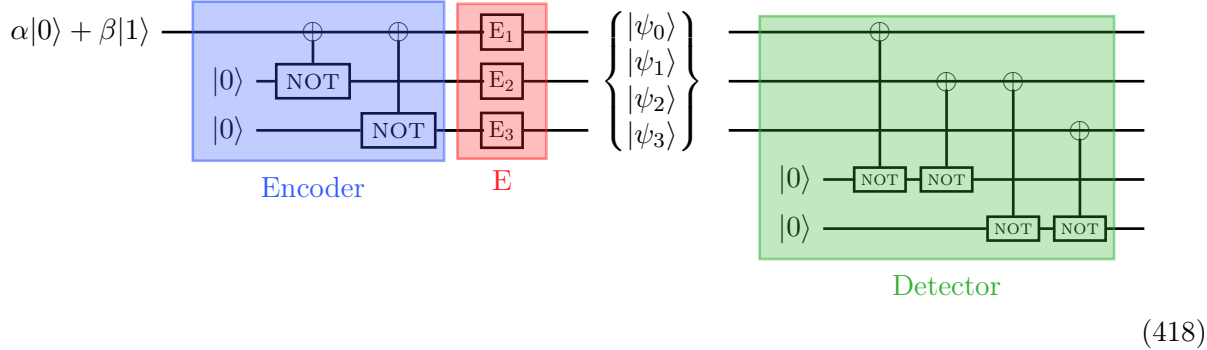
### 21.3 Detecting Bit-Flip Error

To detect the error, the great idea is not to measure the state itself. Rather, entangle the state with some ancillary qubits. Through entanglement, the state of the ancillary qubits will depend on which of the four corrupted states emerged after the error operator. By measuring the state of the

<sup>5</sup>You can also see this from the vector representations of the corrupted states,

$$|\psi_0\rangle = \begin{bmatrix} \alpha \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \beta \end{bmatrix}, \quad |\psi_1\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \beta \\ \alpha \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad |\psi_2\rangle = \begin{bmatrix} 0 \\ 0 \\ \alpha \\ 0 \\ 0 \\ \beta \\ 0 \\ 0 \end{bmatrix}, \quad |\psi_3\rangle = \begin{bmatrix} 0 \\ \alpha \\ 0 \\ 0 \\ 0 \\ 0 \\ \beta \\ 0 \end{bmatrix}.$$

ancillary qubits, we can reveal the error without measuring (and destroying) the state itself. Once the great idea is revealed, it is not so hard to accomplish. To entangle qubits, use one qubit to control an operator, for example NOT, on another. We give a circuit that accomplishes this with two ancillary qubits. Why must we have at least two ancillary qubits? Because measuring the ancillary qubits produces a pure state that must distinguish between 4 possible corrupted states. Hence, there must be at least 4 pure states for the ancillae, which requires at least two ancillary qubits.



In the detector, qubits 1 and 2 of the corrupted state control NOTs on the first ancillary. Label these controlled NOTs as  $\text{cNOT}_{14}$  and  $\text{cNOT}_{24}$  to identify the controlling and controlled bit. Then, qubits 2 and 3 of the corrupted state control NOTs on the second ancillary. Label these controlled NOTs as  $\text{cNOT}_{25}$  and  $\text{cNOT}_{35}$ . We need to examine the action of the sequence of operators

$$\text{cNOT}_{14}, \text{cNOT}_{24}, \text{cNOT}_{25}, \text{cNOT}_{35} \tag{419}$$

on the four possible states

$$|\psi_0\rangle \otimes |00\rangle, \quad |\psi_1\rangle \otimes |00\rangle, \quad |\psi_2\rangle \otimes |00\rangle, \quad |\psi_3\rangle \otimes |00\rangle. \tag{420}$$

By linearity of the tensor product,

$$\begin{aligned} |\psi_0\rangle \otimes |00\rangle &= \alpha|000\rangle \otimes |00\rangle + \beta|111\rangle \otimes |00\rangle \\ |\psi_1\rangle \otimes |00\rangle &= \alpha|100\rangle \otimes |00\rangle + \beta|011\rangle \otimes |00\rangle \\ |\psi_2\rangle \otimes |00\rangle &= \alpha|010\rangle \otimes |00\rangle + \beta|101\rangle \otimes |00\rangle \\ |\psi_3\rangle \otimes |00\rangle &= \alpha|001\rangle \otimes |00\rangle + \beta|110\rangle \otimes |00\rangle \end{aligned} \tag{421}$$

Let us now use linearity of the  $\text{cNOT}$  operators to perform the detector operations on  $|\psi_0\rangle \otimes |00\rangle$ . We apply the four  $\text{cNOT}$  operators to each pure state in the superposition and add the final results.

$$\begin{array}{l} |\psi_0\rangle \quad \alpha|000\rangle \otimes |00\rangle + \beta|111\rangle \otimes |00\rangle \\ \text{cNOT}_{14} \quad \alpha|000\rangle \otimes |00\rangle + \beta|111\rangle \otimes |10\rangle \\ \text{cNOT}_{24} \quad \alpha|000\rangle \otimes |00\rangle + \beta|111\rangle \otimes |00\rangle \\ \text{cNOT}_{25} \quad \alpha|000\rangle \otimes |00\rangle + \beta|111\rangle \otimes |01\rangle \\ \text{cNOT}_{35} \quad \alpha|000\rangle \otimes |00\rangle + \beta|111\rangle \otimes |00\rangle \\ \hline (\alpha|000\rangle + \beta|111\rangle) \otimes |00\rangle \end{array} \tag{422}$$

The last step uses linearity of the tensor product. We highlighted the bits involved in each operation, blue for controlling and red for controlled. In the first step, bit 1 controls bit 4: when bit-1 is 0,

bit-4 is unchanged; when bit-1 is 1, bit-4 flips. The other three steps are similar. The conclusion is,

$$|\psi_0\rangle \otimes |00\rangle \rightarrow |\psi_0\rangle \otimes |00\rangle \quad (423)$$

Let's repeat this calculation for  $|\psi_1\rangle \otimes |00\rangle$ ,

$$\begin{array}{l} |\psi_1\rangle \quad \alpha|100\rangle \otimes |00\rangle + \beta|011\rangle \otimes |00\rangle \\ \text{cNOT}_{14} \quad \alpha|100\rangle \otimes |10\rangle + \beta|011\rangle \otimes |00\rangle \\ \text{cNOT}_{24} \quad \alpha|100\rangle \otimes |10\rangle + \beta|011\rangle \otimes |10\rangle \\ \text{cNOT}_{25} \quad \alpha|100\rangle \otimes |10\rangle + \beta|011\rangle \otimes |11\rangle \\ \text{cNOT}_{35} \quad \alpha|100\rangle \otimes |10\rangle + \beta|011\rangle \otimes |10\rangle \\ \hline (\alpha|100\rangle + \beta|011\rangle) \otimes |10\rangle \end{array} \quad (424)$$

The conclusion is

$$|\psi_1\rangle \otimes |00\rangle \rightarrow |\psi_1\rangle \otimes |10\rangle \quad (425)$$

It's now your turn to verify these calculations for  $|\psi_2\rangle \otimes |00\rangle$  and  $|\psi_3\rangle \otimes |00\rangle$ .

$$\begin{array}{ll} |\psi_2\rangle \quad \alpha|010\rangle \otimes |00\rangle + \beta|101\rangle \otimes |00\rangle & |\psi_3\rangle \quad \alpha|001\rangle \otimes |00\rangle + \beta|110\rangle \otimes |00\rangle \\ \text{cNOT}_{14} \quad \alpha|010\rangle \otimes |00\rangle + \beta|101\rangle \otimes |10\rangle & \text{cNOT}_{14} \quad \alpha|001\rangle \otimes |00\rangle + \beta|110\rangle \otimes |10\rangle \\ \text{cNOT}_{24} \quad \alpha|010\rangle \otimes |10\rangle + \beta|101\rangle \otimes |10\rangle & \text{cNOT}_{24} \quad \alpha|001\rangle \otimes |00\rangle + \beta|110\rangle \otimes |00\rangle \\ \text{cNOT}_{25} \quad \alpha|010\rangle \otimes |11\rangle + \beta|101\rangle \otimes |10\rangle & \text{cNOT}_{25} \quad \alpha|001\rangle \otimes |00\rangle + \beta|110\rangle \otimes |01\rangle \\ \text{cNOT}_{35} \quad \alpha|010\rangle \otimes |11\rangle + \beta|101\rangle \otimes |11\rangle & \text{cNOT}_{35} \quad \alpha|001\rangle \otimes |01\rangle + \beta|110\rangle \otimes |01\rangle \\ \hline (\alpha|010\rangle + \beta|101\rangle) \otimes |11\rangle & (\alpha|001\rangle + \beta|110\rangle) \otimes |01\rangle \end{array} \quad (426)$$

The four possible states after going through the detector are as follows,

$$|\psi_0\rangle \rightarrow |\psi_0\rangle \otimes |00\rangle \quad (427)$$

$$|\psi_1\rangle \rightarrow |\psi_1\rangle \otimes |10\rangle \quad (428)$$

$$|\psi_2\rangle \rightarrow |\psi_2\rangle \otimes |11\rangle \quad (429)$$

$$|\psi_3\rangle \rightarrow |\psi_3\rangle \otimes |01\rangle \quad (430)$$

The state of the ancillae is highlighted. The important fact is that the ancillae are in a *different* pure state depending on the corrupted state.

## 21.4 Correcting Bit-Flip Error

Measuring the state of the ancillae produces a different pure state for each possible corrupted state of our 3-qubit encoded state. Depending on the outcome, we detect if there was an error, and if so which bit was flipped. We can then unflip that bit. For example, if we measure the ancillae as  $|11\rangle$ , we know qubit 2 in the encoded state was flipped, so we unflip it by applying NOT. This recovers the uncorrupted state. We have managed detected and corrected the bit-flip error without damaging the state. We also still do not know anything about  $\alpha, \beta$ .

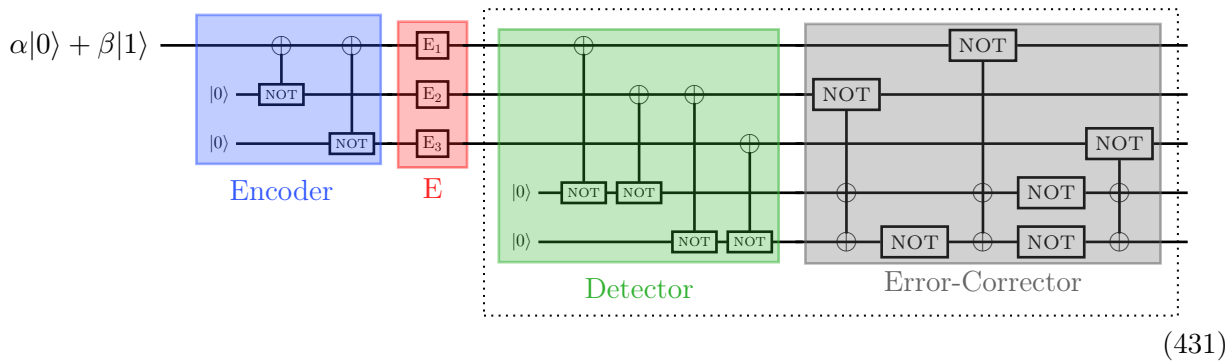
That could be the end of the story with respect to bit-flip errors, but there is some value in embellishing a little. We don't actually need to measure the ancillae. The reason is we know the



ancillae are in specific pure states, depending on the corrupted 3-qubit encoded state. Hence, we can use the ancillae to appropriately control NOTs on the corrupted 3-qubit state. This will restore the 3-qubit code to its uncorrupted state without the need for disruptive measuring. The logic we need to implement is

- 1: **if** ancillae =  $|00\rangle$  **then**
- 2:   Do nothing
- 3: **else if** ancillae =  $|10\rangle$  **then**
- 4:   Apply NOT<sub>1</sub>
- 5: **else if** ancillae =  $|11\rangle$  **then**
- 6:   Apply NOT<sub>2</sub>
- 7: **else if** ancillae =  $|01\rangle$  **then**
- 8:   Apply NOT<sub>3</sub>

This is accomplished by the error-corrector circuit in the gray box,



The output of this circuit for any  $\alpha, \beta$  is

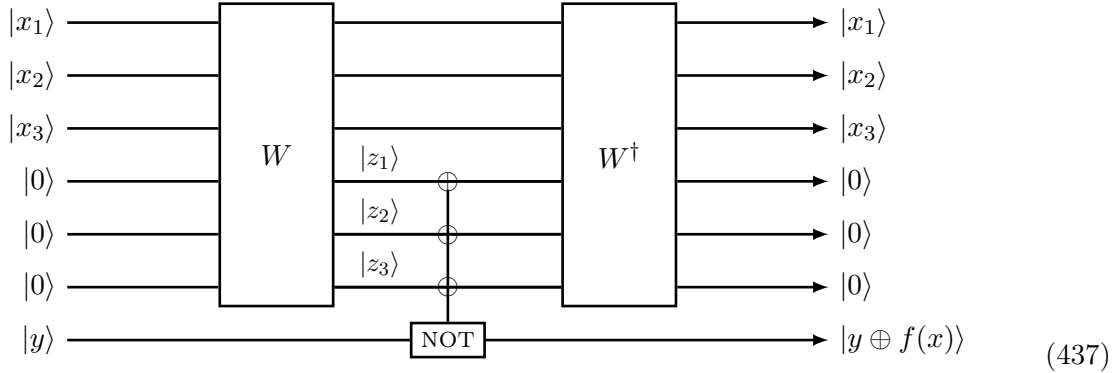
$$(\alpha|0\rangle + \beta|1\rangle) \otimes |\text{ancillae}\rangle. \tag{432}$$

The two ancillae are stored in some pure state, but we do not know what that state is. The full bit-flip quantum error correction circuit enclosed in the dotted rectangle above is applied periodically to correct bitflip error. To reuse the ancillae one has to reset them to  $|00\rangle$  which is most easily done by measuring them and applying NOT if needed.

## 21.5 Generalizing to Other Errors



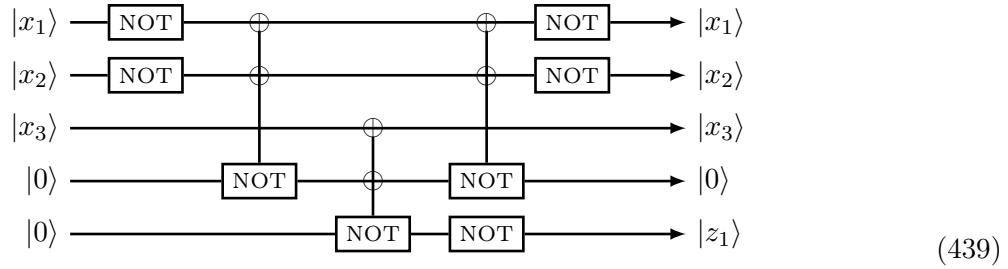
are  $|z_1\rangle, |z_2\rangle, |z_3\rangle$ . We discussed one circuit for controlling an operator with multiple bits in (335) on page 75. Here is a circuit implementing this idea that uses additional ancillary bits.



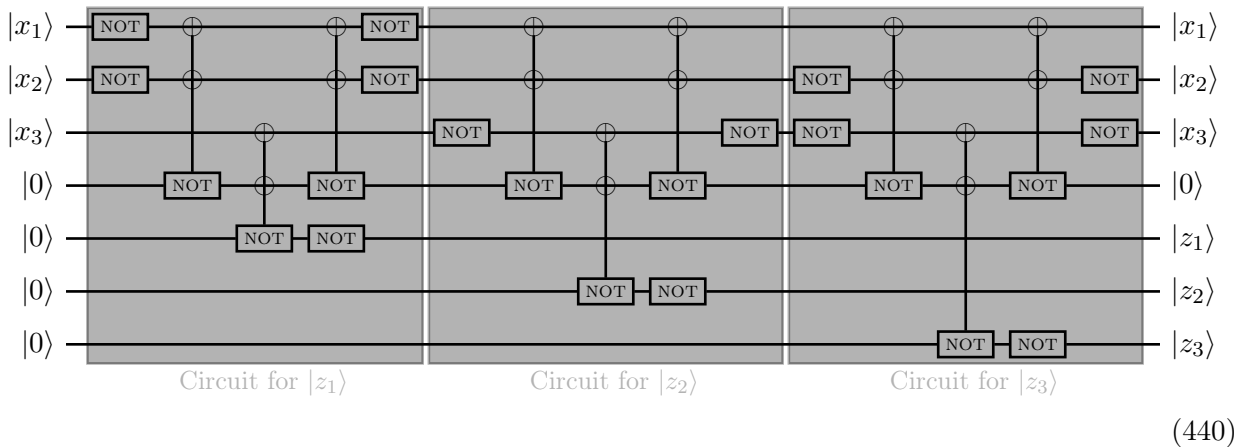
Let us find a circuit for  $|z_1\rangle$ . We use

$$z_1 = x_1 \vee x_2 \vee \overline{x_3} \stackrel{\text{equiv}}{\equiv} \text{NOT}(\overline{x_1} \wedge \overline{x_2} \wedge x_3). \quad (438)$$

Since a multiple AND is just a multiply controlled NOT on a  $|0\rangle$ -bit, the reader should verify in detail that the following quantum circuit computes  $|z_1\rangle$ ,



The NOTs and Toffolis on qubits 1,2,3,4 are undone by NOTs and Toffolis in reverse order, which is important so that the states  $|x_1\rangle, |x_2\rangle, |x_3\rangle, |0\rangle$  are reproduced on the top four qubits. This means that the ancillary  $|0\rangle$ -qubit that was restored to  $|0\rangle$  can be reused, together with  $|x_1\rangle, |x_2\rangle, |x_3\rangle$  to produce  $|z_2\rangle, |z_3\rangle$ . A similar construction works to get  $|z_2\rangle, |z_3\rangle$ . We leave it to the reader to go through the details and derive the full circuit to compute  $|z_1\rangle, |z_2\rangle, |z_3\rangle$  shown below,



The circuit in (467) can be plugged into (464) to implement  $U_f$ . Note that in a repeated use of  $U_f$  it is important to undo the effects of  $W$  by applying  $W^\dagger$  so that the three ancillary qubits that produced  $|z_1\rangle, |z_2\rangle, |z_3\rangle$  can be reused. Since the circuit only uses NOT and Toffoli, and both of these gates are their own inverse, one can simply apply the gates that have been used to get  $|z_1\rangle, |z_2\rangle, |z_3\rangle$  in reverse order to obtain  $W^\dagger$ . This inverse has to be performed after using the clauses  $|z_1\rangle, |z_2\rangle, |z_3\rangle$  to control the NOT on  $|y\rangle$ . In our example, we need three Toffolis and three NOTs to reset  $|z_1\rangle, |z_2\rangle, |z_3\rangle$  to  $|0\rangle, |0\rangle, |0\rangle$ .

## 22.2 General Case

Consider an instance of 3-SAT with  $n$  variables  $x_1, \dots, x_n$  and  $m$  clauses  $z_1, \dots, z_m$ . We need three Toffolis to implement each clause and at most seven additional NOTs. To reset each clause  $|z_i\rangle$  to  $|0\rangle$  we need one more Toffoli and NOT, so we need  $4m$  Toffolis and at most  $8m$  NOTs.

In terms of qubits, which are costly, we need  $n$  qubits for the variables,  $m + 1$  qubits initialized to  $|0\rangle$  to compute the clauses. We also need to implement the multiply controlled NOT on  $|y\rangle$  where the  $m$  clauses are the controlling bits. This requires  $m - 2$  qubits initialized to  $|0\rangle$  using the construction in (335) on page 75 and  $m - 1$  Toffolis. We can reuse the qubit that is reset to  $|0\rangle$  in (467), so we need an additional  $m - 1$  qubits. A more advanced construction does not even need these additional  $m - 1$  qubits, and can make do with the single  $|0\rangle$  from (467). In summary, the resources we need are at most:

Toffoli	$5m - 1$
NOT	$8m$
qubits	$n + 2m - 2$

## 23 Quantum Factoring

The input is an integer  $N > 2$ . The output is a non-trivial factor  $D$  of  $N$  if one exists ( $1 < D < N$  and  $D$  divides  $N$ ). If no such factor exists, then the algorithm should declare  $N$  to be prime. A simple classical algorithm tests if any of  $2, 3, \dots, \lfloor \sqrt{N} \rfloor$  divide  $N$ . If yes, you have a factor and if no,  $N$  is prime. Testing if  $i$  divides  $N$  is in  $\Theta(\log^3 N)$  using simple grade school algorithms, so the total runtime is in  $\Theta(\sqrt{N} \log^3 N)$ , which is an exponential algorithm. Exponential because the input size is  $O(\log_2 N)$ , the number of bits needed to specify  $N$ .

Shor's quantum algorithm for factoring runs in  $\text{poly}(\log N)$  time. It is the one example we have of exponential speedup against the best known classical algorithm. (Grover's search algorithm only gives quadratic speedup.) This does not, however, mean that quantum computing gives exponential speedups over classical computing because we do not know if a factor can be found in  $\text{poly}(\log N)$  time on a classical computer. Factoring is a complex number theoretic task. It is not clear how a quantum computer, by somehow using quantum parallelism, could help with factoring. We know how to test properties of functions using quantum parallelism. The general idea is to evaluate the function on a massive superposition, producing a state that has all the information you need to test the given property of the function. The remaining quantum-magic is to unravel all that information to get at the property you want. It turns out that the property of a function we will need is its period. Given a periodic function  $f : \mathbb{N} \mapsto [0, N - 1]$ , suppose

$$f(x + r) = f(x), \tag{441}$$

for some  $r \in [1, N - 1]$ . We want to identify  $r$ . A classical algorithm could compare  $f(x)$  to  $f(x + 1), f(x + 2), \dots, f(x + N - 1)$  to identify  $r$ . The runtime is  $\Theta(N)$  (exponential), which is even worse than the simple classical algorithm for factoring above. The plan is

- (1) Reduce factoring to finding the period of some function. [Number Theory]
- (2) Build the mathematical tools for period finding. [Fast Fourier Transform, FFT]
- (3) Build a quantum algorithm for period finding to plug into (1). [Quantum FFT]

The FFT is a landmark algorithm, both as a primitive in other algorithms and also as a tool for fast signal processing. Its impact has been immense. So, even if you have no interest in factoring or quantum computing, you will want to learn these mathematical tools.

### 23.1 Factoring and Period Finding

This section requires some basic number theory knowledge. The number theory is used to reformulate the factoring problem into one of testing the periodicity of a function defined on the natural numbers. The reader who wishes to skip the number theoretic justification and focus on the algorithmic details can move on to Section 23.2 without sacrificing the logical flow.

Since one can test if  $N$  is even, in which case 2 is a nontrivial factor, assume  $N$  is odd. As a working example, let  $N = 15$ . Consider an integer  $x > 1$  and let

$$\text{gcd}(N, x) = D. \tag{442}$$

If  $1 < D < N$ , then  $D$  is a nontrivial factor. For example, if  $x = 25$ , then  $\text{gcd}(15, 25) = 5$  and 5 is a factor of 15. If  $D = 1$ , then we cannot immediately get a factor from  $D$ . For example suppose

$x = 26$ . There is some progress we can make even in this case. Notice that  $26^2 = 676$  and  $675$  is divisible by  $15$ , so

$$26^2 \equiv 1 \pmod{15}, \tag{443}$$

where  $a \equiv b \pmod{c}$  if and only if  $c$  divides  $a - b$ , written  $c|(a - b)$ . Further,  $26 \pm 1$  is not divisible by  $15$ , but  $\gcd(26 \pm 1, 15) > 1$ , and either case gives a factor of  $15$ . Indeed,  $\gcd(25, 15) = 5$  and  $\gcd(27, 15) = 3$ . This example is not an isolated case. Indeed, consider any  $x > 1$  for which

$$x^2 \equiv 1 \pmod{N}; \tag{444}$$

$$x \not\equiv \pm 1 \pmod{N}. \tag{445}$$

We claim that  $\gcd(x \pm 1, N) > 1$ . Intuitively, if  $N|(x^2 - 1)$  then  $N|(x + 1)(x - 1)$  and  $N$  does not divide either term in the product, then the nontrivial factors of  $N$  must be spread across both terms. Let us formulate this explicitly as a lemma.

**Lemma 23.1.** Suppose  $x, N > 1$  with

$$x^2 \equiv 1 \pmod{N} \quad \text{and} \quad x \not\equiv \pm 1 \pmod{N}. \tag{446}$$

Then  $1 < \gcd(x + 1, N) < N$  and  $1 < \gcd(x - 1, N) < N$ .

The impact of this lemma is that if some  $x$  satisfies the conditions, then computing  $\gcd(x - 1, N)$  gives a nontrivial factor of  $N$ . Using Euclid's algorithm, one can compute the gcd in  $O(\log N)$ . We assume the reader is familiar with or can look up Euclid's algorithm.

*Proof.* Since  $x^2 \equiv 1 \pmod{N}$ , this means  $N|(x^2 - 1)$ , or  $N|(x + 1)(x - 1)$ . If  $\gcd(N, x + 1) = 1$ , then by Euclid's Lemma<sup>6</sup>,  $N|(x - 1)$  which contradicts  $x \not\equiv 1 \pmod{N}$ . Therefore  $\gcd(N, x + 1) > 1$ . Similarly, if  $\gcd(N, x - 1) = 1$ , then  $N|(x + 1)$  contradicting  $x \not\equiv -1 \pmod{N}$ . The upper bound  $\gcd(x \pm 1, N) < N$  follows because  $N$  does not divide  $x \pm 1$ . ■

The goal is to find an  $x$  satisfying the conditions in Lemma 23.1. We then get a factor efficiently by computing  $\gcd(x - 1, N)$ . To find such an  $x$ , we introduce the period of a number modulo  $N$ . Let us compute the powers of  $2$  modulo  $N = 15$ ,

	$2^1$	$2^2$	$2^3$	$2^4$	$2^5$	$2^6$	$2^7$	$2^8$	$2^9$	$2^{10}$	$2^{11}$	$2^{12}$	
$(\text{mod } 15)$	2	4	8	1	2	4	8	1	2	4	8	1	(447)

Observe that  $2^m \pmod{15}$  as a function of  $m$  is periodic and the period is  $4$ . That is,  $2^{m+4} \equiv 2^m \pmod{15}$ . This is immediate from  $2^4 \equiv 1 \pmod{15}$ , because<sup>7</sup>,

$$\begin{array}{rcl} 2^4 & \equiv & 1 \pmod{15} \\ 2^m & \equiv & 2^m \pmod{15} \\ \hline 2^4 \times 2^m & \equiv & 1 \times 2^m \pmod{15} \end{array} \tag{448}$$

<sup>6</sup>*Euclid's Lemma:* If  $d|ab$  and  $\gcd(d, a) = 1$  then  $d|b$ . The proof uses Bezout's identity:  $\gcd(d, a) = 1 = xd + ya$ . Multiply both sides by  $b$  to get  $b = xdb + yab$ . The RHS is divisible by  $d$  since  $d|ab$ , hence  $d|b$ .

<sup>7</sup>The reader should verify that  $a \equiv b \pmod{d}$  and  $c \equiv d \pmod{d}$  implies  $ac \equiv bd \pmod{d}$ .

Let's play this same game with  $x = 3$ ,

	$3^1$	$3^2$	$3^3$	$3^4$	$3^5$	$3^6$	$3^7$	$3^8$	$3^9$	$3^{10}$	$3^{11}$	$3^{12}$	
(mod 15)	3	9	12	6	3	9	12	6	3	9	12	6	(449)

Again, the function  $3^m \pmod{15}$  is periodic, and again with period 4. The crucial difference between  $2^m$  and  $3^m$  is that  $2^m$  is congruent to 1 modulo 15 for some  $m \geq 1$  but  $3^m$  is never congruent to 1 modulo 15. This is because  $\gcd(3, 15) = 3 > 1$ .<sup>8</sup> We say that the *order* of 2 modulo 15 is 4,

$$\text{order}(2) = 4 \pmod{15}. \quad (450)$$

For a general  $z$  with  $1 < z < N$ , the order of  $z$  modulo  $N$  is the smallest number  $r \geq 1$  for which

$$z^r \equiv 1 \pmod{N}. \quad (451)$$

Let's compute the orders of  $2, 3, \dots, 14$  modulo  $N = 15$ . The reader should verify,

$z$	2	3	4	5	6	7	8	9	10	11	12	13	14	
order( $z$ )	4	✓	2	✓	✓	4	4	✓	✓	2	✓	4	2	(452)

The order does not exist when  $\gcd(z, N) > 1$ , which we already proved. The green checkmarks indicate these cases. Why a green checkmark will become clear soon.

- When  $\gcd(z, N) = 1$  the order always exists and is less than  $N$ .
- The order appears to be always even.

The first claim is true. The second claim is almost true. Here is an example of odd order,

$$\text{order}(2) = 3 \pmod{7}, \quad (453)$$

because  $2^3 \equiv 1 \pmod{7}$ . To prove the first claim, consider the first  $N$  powers of  $z$  modulo  $N$ ,

$$z^1, z^2, z^3, \dots, z^N \pmod{N}. \quad (454)$$

None of these are congruent to 0 because  $\gcd(z, N) = 1$ . Therefore  $1 \leq z^i \pmod{N} \leq N - 1$ . By pigeonhole, two of these powers are congruent to each other modulo  $N$ , that is for  $i < j$ ,

$$z^j \equiv z^i \pmod{N}. \quad (455)$$

This means  $N \mid (z^j - z^i)$ , or  $N \mid z^i(z^{j-i} - 1)$ . Since<sup>9</sup>  $\gcd(z^i, N) = 1$ , we can apply Euclid's Lemma to conclude  $N \mid (z^{j-i} - 1)$ . This means

$$z^{j-i} \equiv 1 \pmod{N} \quad \rightarrow \quad \text{order}(z) \leq j - i \pmod{N}. \quad (456)$$

We have proved the following lemma,

<sup>8</sup>It is an exercise for the reader to prove that if  $\gcd(x, N) = D > 1$ , then  $\gcd(x^m, N) \geq D > 1$ . And also to prove that if  $x^m \equiv 1 \pmod{N}$  then  $\gcd(x^m, N) = 1$  (Bezout's identity can be used). Therefore if  $\gcd(x, N) > 1$  then  $x^m \not\equiv 1 \pmod{N}$  for any  $m \geq 1$ .

<sup>9</sup> $\gcd(z, N) = 1$  implies  $\gcd(z^i, N) = 1$ , which the reader can prove by induction or directly using Bezout's identity.

**Lemma 23.2.** If  $\gcd(z, N) = 1$  then  $r = \text{order}(z) \pmod N$  exists and  $r < N$ .

We went out of our way to highlight that the order was even, a fact which we conceded was not always true. You might have guessed that an even order is somehow important, and it is. Suppose  $z$  has even order  $r$ , that is  $z^r \equiv 1 \pmod N$ . Let  $x = z^{r/2}$ . For  $x$  to be an integer, it is necessary for  $r$  to be even. Then,

$$x^2 = z^r \equiv 1 \pmod N. \tag{457}$$

We have found an  $x$  that satisfies the first condition in Lemma 23.1. We also know that  $x \not\equiv 1 \pmod N$  because  $r$  is the order of  $z$ , that is the *smallest* power congruent to 1. If it also the case that  $x \not\equiv -1 \pmod N$ , then all the conditions of Lemma 23.1 are satisfied, and we can find a factor of  $N$  using  $\gcd(x - 1, N)$ . Let's work through another example,  $N = 21$ ,

$z$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$r$	6	✓	3	6	✓	✓	2	✓	6	6	✓	2	✓	✓	3	6	✓	6	2
$x = z^{r/2}$	8		✗	125			8		1000	1331		13			✗	4913		6859	20
$x \not\equiv -1?$	✓		✗	✗			✓		✓	✓		✓			✗	✗		✓	✗
factor?	✓	✓	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✓	✓	✗

(458)

The first row shows that of the 19 possibilities for  $z$ , eight of them immediately give a factor by computing  $\gcd(z, N)$ . The other 11 give  $\gcd(z, N) = 1$  and one can find the order. The third row shows that in 6 of those cases where  $\gcd(z, N) = 1$ , the order is even and  $z^{r/2} \not\equiv -1 \pmod N$ . The conclusion is shown in the last row. Of the 19 possibilities for  $z$ , five of them cannot be used to conveniently get a factor for  $N = 21$ . The other 14 possibilities for  $z$  give a nontrivial factor in one of two ways:

- $\gcd(z, N) > 1$  in which case  $\gcd(z, N)$  is a nontrivial factor.
- $\gcd(z, N) = 1$ , the order  $r$  of  $z$  is even and  $z^{r/2} \not\equiv -1 \pmod N$ , in which case  $\gcd(z^{r/2} - 1, N)$  gives a nontrivial factor.

For  $N = 21$ , the chances are  $14/19 \approx 74\%$  that a random  $z \in [2, N - 1]$  produces a nontrivial factor. This situation is not isolated. More generally, the probability is at least  $\frac{1}{2}$ .

**Lemma 23.3.** Given an odd composite  $N$ , let  $G_N \subseteq [1, N - 1]$  be those numbers less than  $N$  which are coprime with  $N$ . Modulo  $N$ , at least half the elements of  $G_N$  have even order and are not congruent to  $-1$ .

Lemma 23.3 implies the following result which leads to the algorithm in the next section. Pick a  $z$  randomly from  $[2, N - 1]$  and let  $p = \mathbb{P}[\gcd(z, N) > 1]$ . Success occurs if either  $\gcd(z, N) > 1$  or  $z$  has even order  $r$ , with  $z^{r/2} \not\equiv -1 \pmod N$ . Then,

$$\mathbb{P}[\text{Success}] = \underbrace{\mathbb{P}[\text{Success} \mid \gcd(z, N) > 1]}_1 \times p + \underbrace{\mathbb{P}[\text{Success} \mid \gcd(z, N) = 1]}_{\geq \frac{1}{2} \text{ by Lemma 23.3}} \times (1 - p) \tag{459}$$

$$\geq \frac{1 + p}{2}. \tag{460}$$

The chance of failure is at most  $1/2$ , so by repeatedly trying  $k$  independent  $z \in [2, N - 1]$ , we fail to get a factor with probability at most  $1/2^k$ . The most intense number theory of this section is about to come in the proof of Lemma 23.3. This proof is essential for the algorithm but not instructive. On a first reading you may skip to the algorithm in Section 23.2.



## 23.2 Algorithm for Factoring

The previous section suggests the following algorithm to get a factor of  $N$ . Generate a random  $z \in [2, N - 1]$ . If  $\gcd(z, N) > 1$ , you have a nontrivial factor. If not, but  $r = \text{order}(z) \pmod N$  is even, then  $\gcd(z^{r/2} - 1, N)$  may give a nontrivial factor. If all this fails, with probability at most  $1/2$ , you have to repeat the whole process. Here is the algorithm.

- 1: **Input:**  $N$ .
- 2: Test if  $N$  is prime using AKS (or Miller-Rabin). Continue if  $N$  is not prime.
- 3: **repeat**
- 4:   Pick a random  $z \in [2, N - 1]$ .
- 5:   Compute  $D = \gcd(z, N)$ .
- 6:   **if**  $D = 1$  **then**
- 7:     Compute  $r = \text{order}(z) \pmod N$ .
- 8:     **if**  $r$  is even **then**
- 9:       Compute  $D = \gcd(z^{r/2} - 1, N)$
- 10: **until**  $1 < D < N$

The algorithm outputs  $D$ . Each repetition fails with probability at most  $1/2$  so the expected number of repetitions to success is 2. The only potentially inefficient steps in the algorithm are steps 7 and 9. Step 9 can be implemented as follows. First compute  $a \in [0, N - 1]$  such that

$$z^{r/2} \equiv a \pmod N. \tag{461}$$

This can be done efficiently using fast modular exponentiation that uses squaring. The number of squaring steps needed is  $O(\log_2(r/2)) \in O(\log_2 N)$ . In each step one performs a constant number of multiplications of  $\log_2 N$ -bit numbers, so the runtime is  $O(\log_2^3 N)$ . Finally one can compute  $\gcd(a - 1, N)$  to complete step 9. So, the full algorithm is polynomial if step 7, finding the period of a number, can be performed in  $O(\text{poly}(\log_2 N))$  time. On a classical computer, we do not know how to do it. The entire contribution of quantum computing to factoring is to efficiently solve the period finding problem in step 7. The solution involves the quantum version of the FFT, which is similar to a Hadamard transform. We will discuss these details in the next section. We also mention that period finding can be used to directly crack the RSA cryptosystem, as opposed to indirectly doing so via factoring. Also note that period finding is a special case of the general problem of “phase estimation”, that is to find the eigenvalue of a unitary matrix for a given eigenvector.<sup>10</sup>

---

<sup>10</sup>Since a unitary matrix preserves the norm, all eigenvalues are phases.

## 24 Quantum FFT and Period Finding

## 25 Quantum Cryptography and Key Exchange

## 26 Quantum Teleportation

## 27 Quantum Information Theory