

Emulating Circuit-Based and Measurement-Based Quantum Computation

MEng4 Individual Project Final Report
Supervised by Dr Herbert Wiklicky

Joey Allcock
sa106@doc.ic.ac.uk

<http://www.doc.ic.ac.uk/~sa106/q/>

Department of Computing
Imperial College London

June 2010

Abstract

This thesis presents classical emulations of circuit-based and measurement-based quantum computation. In order to create emulators faithful to the theory of quantum computation we develop the required mathematics, quantum mechanics and quantum computing background. We justify our implementations by documenting the theory of each of these subjects in detail and showing our implementation directly follows from it.

In relation to the circuit model we discuss quantum bits (qubits), quantum gates, and quantum circuits with a view to implementing quantum algorithms. In addition, we develop a quantum circuit calculus in order to formally specify quantum circuit diagrams.

We base our discussion of the measurement-based approach upon the the one-way computer architecture and the measurement calculus. This calculus provides a formal description of measurement-based quantum computation and awards us a quantum *assembly language*; we structure our implementation around this language. Much of the measurement-based model can be equated to the circuit model, and we do so to help build intuition. We describe von Neumann and generalised projective measurement of qubits in non-standard bases; the fundamental operation of measurement-based computation.

The evaluation of our implementations covers correctness, accuracy, bounds and efficiency, and limitations. For both emulators we successfully implement the Deutsch and Deutsch-Jozsa algorithms. For the circuit model we also run Shor's algorithm for numbers up to 16, requiring 12 combined qubits. With some modification to memory management we are able to represent 13 entangled qubits. We describe the workings of each of these algorithms in detail, and from this we justify the correctness of our emulators. We give the running times of each algorithm, and we judge the accuracy of probabilistic actions and quantum state representation for each implementation. From these individual evaluations we provide some comparison between the presented models.

Although our emulators are both correct and accurate, we achieve this at the cost of run time and the number of qubits we are able to represent. In comparison to other circuit model simulations we achieve fewer combined qubits. For the measurement-based emulator we determine that a much lower precision representation may be used for qubits. Our investigation reveals that measurements destroy any rounding errors introduced into the system. This reduced precision would allow us to represent a greater number of combined qubits. With this change, our emulations would compete with the majority of existing simulations in terms of qubits represented and run time.

Acknowledgements

I would like to acknowledge the time and effort a number of people have generously given during the course of this project. Without this help it would not have been possible.

Herbert Wiklicky suggested the subject of this thesis and as supervisor has offered guidance, expertise and encouragement throughout. Chris Hankin gave careful and considered appraisals of the project as second supervisor. The opinion of a reader outside both the project and the field was extremely beneficial and appreciated. Jamie Leinhardt provided help with some of the more difficult mathematics involved in the quantum mechanics, as well as continued support over the year. My family has given the resources and belief needed to complete my education up to this point, and as such, this project is one of the many products of their continued efforts.

For the commitment these people and others have shown I am extremely grateful.

Contents

1	Introduction	1
1.1	Introduction	1
1.2	Emulation vs. Simulation	2
1.3	Contributions	3
2	The Quantum Circuit Model	5
2.1	Introduction	5
2.2	Background	5
2.2.1	Beginnings	6
2.2.2	Existing Solutions	6
2.2.3	Further Reading	7
2.3	Problem Specification & Overview of Implementation	8
2.4	Quantum Bits (Qubits)	8
2.4.1	Classical vs. Quantum Bits	8
2.4.2	Combining Qubits	10
2.4.3	Implementing Qubits	10
2.5	Quantum Gates	12
2.5.1	Classical Logic Gates	12
2.5.2	Quantum Gates	13
2.5.3	Combining Quantum Gates	15
2.5.4	Implementing Quantum Gates	18
2.6	Quantum Algorithms	20
2.6.1	Deutsch's Algorithm	20
2.6.2	Implementing Deutsch's Algorithm	21
2.6.3	Deutsch-Jozsa Algorithm	24
2.6.4	Implementing the Deutsch-Jozsa Algorithm	24
2.6.5	Shor's Algorithm	28
2.6.6	Implementing Shor's Algorithm	29
2.7	Developing a Quantum Circuit Calculus	33
2.8	Outcomes	36

3	Measurement-Based Quantum Computation	37
3.1	Introduction	37
3.2	Background	37
3.2.1	One-Way Quantum Computer	38
3.2.2	Measurement Calculus	39
3.2.3	Existing Simulations	40
3.3	Problem Specification & Overview of implementation	40
3.4	Measurements in Non-Standard Bases	41
3.4.1	Implementing Measurements in Non-Standard Bases	43
3.5	Measurement Patterns	45
3.5.1	Universal Gate Pattern	47
3.5.2	Implementing Measurement Patterns	48
3.6	Combining Patterns	50
3.6.1	Implementing the Combination of Patterns	51
3.6.2	Implementing Quantum Algorithms	53
3.7	Standardising Patterns & Reducing Emulation Resources	55
3.7.1	Standardising Patterns	55
3.7.2	Reducing Qubit State Space	56
3.7.3	Garbage Collecting Qubits	57
3.8	Outcomes	57
4	Evaluation	59
4.1	Introduction	59
4.2	The Quantum Circuit Model	59
4.2.1	Demonstration of Correctness	59
4.2.2	Accuracy	62
4.2.3	Efficiency and Bounds	65
4.2.4	Comparisons with Existing Solutions	68
4.2.5	Limitations	69
4.2.6	Quantum Circuit Calculus	69
4.3	Measurement-Based Quantum Computation	70
4.3.1	Demonstration of Correctness	70
4.3.2	Accuracy	73
4.3.3	Efficiency and Bounds	73
4.3.4	Limitations	75
4.4	Comparisons Between Models	76
4.4.1	Efficiency, Bounds and Accuracy	76
4.4.2	Equivalence & Conversion between Models	77
5	Conclusions	79
5.1	Introduction	79

5.2 Future Work	80
Bibliography	83
Appendix A Related Mathematics	89
A.1 Introduction	89
A.2 Imaginary Numbers	89
A.3 Complex Numbers	89
A.4 Operations On Complex Numbers - \mathbb{C}	90
A.4.1 Addition	90
A.4.2 Subtraction	90
A.4.3 Multiplication	90
A.4.4 Division	90
A.4.5 Conjugation	90
A.4.6 Modulus (Magnitude)	90
A.5 Representation of Complex Numbers	91
A.5.1 Cartesian Representation	91
A.5.2 Polar Representation	91
A.6 Complex Vector Spaces	92
A.7 Operations on Complex Vectors - \mathbb{C}^n	93
A.7.1 Addition	93
A.7.2 Subtraction	93
A.7.3 Multiplication by Scalar	94
A.7.4 Additive Inverse (Negative)	94
A.8 Operations on Complex Matrices - $\mathbb{C}^{m \times n}$	94
A.8.1 Addition	95
A.8.2 Subtraction	95
A.8.3 Multiplication by Scalar	95
A.8.4 Matrix Multiplication	96
A.8.5 Additive Inverse (Negative)	97
A.8.6 Multiplicative Inverse	97
A.8.7 Transposition	97
A.8.8 Conjugation	97
A.8.9 Dagger (Adjoint)	97
A.8.10 Inner Product (Dot Product)	98
A.8.11 Eigenvalues and Eigenvectors	98
A.8.12 Tensor Product (Kronecker Product)	98
A.8.13 Powers of Matrices	100
A.9 Further Reading	100
Appendix B Quantum Mechanics	101
B.1 Introduction	101
B.2 Deterministic vs. Probabilistic Systems	101

CONTENTS

B.2.1	Classical Deterministic Systems	101
B.2.2	Classical Probabilistic Systems	103
B.3	Invertibility and Reversibility	106
B.4	Interference and Superposition	108
B.5	Composite Systems	111
B.6	Quantum Mechanics and Ket Notation	113
B.6.1	Quantum States	113
B.6.2	Dynamics	115
B.6.3	Observables and Measurement	115
B.7	Further Reading	115
Appendix C	Symbols	117
C.1	Mathematics and Physics	117
C.2	Quantum Gates and Quantum Circuits	118

Chapter 1

Introduction

*We're not building this so that we can run Microsoft Office on it.
In fact, there is no reason to build anything to run Microsoft Office on.*
Julio Gea-Banacloche [1]

1.1 Introduction

Quantum computing is an interesting new field combining elements of computer science, physics and mathematics. It enables us to exploit certain quantum mechanical phenomena to vastly expand the possibilities of computation. The problem we face is that it is likely to be many years before we develop a viable physical implementation of such a computer. There is a vast amount of literature describing the theory of quantum computation and possible approaches to implementation. The infancy of the field has resulted in research covering a huge number of often very specific ideas which are spread across numerous architectures and approaches. Very little of this relates to the simulation or emulation of quantum computation on classical computers. Any development in new techniques, algorithms or architectures is largely theoretical, and computer scientists have no way to realise their ideas. For example, we are a long way from implementing Shor's polynomial time factoring algorithm which truly exemplifies the potential power of quantum computing [54, 55]. By creating emulations of a quantum computer on classical systems we are able to test, at least to some degree, the workings of such algorithms.

We are interested specifically in two approaches to quantum computation. The circuit model developed by Deutsch is the most documented and well understood quantum architecture [15]. This model is based upon quantum bits, quantum gates and quantum circuits. As it is so closely analogous to electrical circuits, it is often the favoured and most approachable model. An emulation of the circuit model exists; however we do not feel it sufficiently follows the relevant theory and should be classed as a simulation (see 1.2). The second approach we consider is measurement-based quantum computation using the one-way quantum computer architecture [37, 39, 43, 50, 51]. This recent idea has no classical analogue and currently is of great interest in the field; it is thought to more obviously lend itself to physical implementation than other models. There are currently no simulations of measurement-based quantum computation in the literature, for the one-way computer or any other architecture.

We aim to create emulations for both the circuit model and the measurement-based approach based on the one-way quantum computer. Due to the massive amounts of literature available, much of which is aimed at quantum physicists and mathematicians, we must summarise all related aspects of the field from the standpoint of a computer scientist. Only then will we appreciate the theory behind our emulation and so justify our implementation. Considering the amount of background research and knowledge required this step is a significant undertaking. To formally describe quantum circuits we will develop a circuit calculus to translate between quantum circuit diagrams and a formal notation. With our focus upon two models, we will also present a comparison between them based upon our emulators.

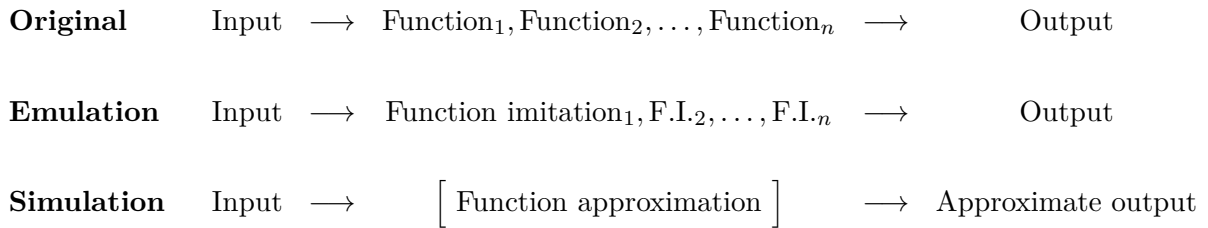


Figure 1.1: Emulation vs. simulation

In order to develop a strong notion of the required quantum theory and the problems we aim to overcome we must review and collate the literature. Currently the size of a quantum state space that can be simulated classically is restricted by the amount of memory available on a single computer [40]. We do not attempt to solve this likely impossible task; it is the main reason why quantum computation was originally postulated by Feynman and others [25]. Instead, we aim to implement faithful emulations, based upon the theory we develop, to the extent that we are able to correctly emulate quantum algorithms. We must develop the correct intuition for the properties of each model based upon the mathematical formulation of quantum mechanics. This step alone is challenging. To help us make the transition from quantum mechanics to an implementation of quantum computation we will make use of two calculi. For the circuit model we will develop our own calculus to describe circuits. To formally describe the measurement-based approach we will use the measurement calculus developed by Danos et al. [10, 11].

1.2 Emulation vs. Simulation

An important part of our implementation is dependent upon the use of the word *emulation*. In particular, differentiating between an emulation and simulation is a key aspect of understanding the challenges involved. We acknowledge there are many different meanings applied to this pair of words, and so for clarity we define our usage.

In computer science terms we use emulation to describe a program or device that is able to fully imitate another. In contrast, a simulation is almost identical to the normal function another program or device performs, but does not do so in the way the original program or device does. This may be via some abstract model of what is being simulated; emulation requires faithful reproduction of the object.

Given some input, a simulation will provide approximately the same output as the original. Emulation provides the same output as the original, and will also process the input in the same way as the original in order to determine its output. We show this notion in Figure 1.1. We see from this that emulation is the purest form of simulation.

In terms of simulating or emulating quantum computation, we are much more easily able to simulate the run of a quantum algorithm. We may do so to the least extent, such that given an input we simply look up the required output from a pre-computed table. More commonly, procedural programs approximately compute the quantum answer using classical algorithms and methods. Our emulation aims to perform every step described by the theories of quantum mechanics and computation, assuming a perfect environment without outside interference. We are emulating the theory behind quantum computation for the two models discussed.

1.3 Contributions

We propose that this thesis will present the following contributions to the field:

- A faithful emulation of quantum computation based on the quantum circuit model. With this we will implement the Deutsch, Deutsch-Jozsa and Shor quantum algorithms. A demonstration of correctness based upon these algorithms should confirm the success of the emulation.
- An emulation of measurement-based quantum computation using the one-way quantum computer architecture. We will construct this from the mathematical description of the measurement-based approach offered by the measurement calculus.
- A comprehensive background describing the required elements of quantum mechanics, the related mathematics, and quantum computing. Explanations are specifically aimed at computer-scientists with no prior knowledge of physics and only basic mathematical ability.
- A circuit calculus to formally describe quantum circuit diagrams and allow the two-way translation between diagrams and the calculus.
- An evaluation of each emulation and a comparison between the models based upon our emulators.

For those with no prior experience of quantum computation, we provide a concise introduction to the mathematics required for this project (Appendix A) followed by an overview of the quantum mechanics upon which quantum computation is based (Appendix B). We explain this difficult subject in simple terms, drawing parallels with classical analogues at every step. For both topics we suggest possible further reading; however all knowledge required for our purposes is present here. For convenience we present a list of symbols and quantum related materials for quick reference (Appendix C).

In the body of the report we first outline the background, theory, and our emulation of the circuit model, along with our presentation of a quantum circuit calculus (2). Following this, we do the same for the measurement-based approach (3). We then evaluate each emulator and compare the two models (4). Finally, we present our conclusions and possible future work (5).

Chapter 2

The Quantum Circuit Model

*Quantum computation is... a distinctively new way of harnessing nature...
It will be the first technology that allows useful tasks to be performed
in collaboration between parallel universes.*

David Deutsch [16]

2.1 Introduction

This project is on the subject of quantum computing (2.2); specifically, creating emulations of two different approaches to quantum computation. In this chapter we will cover the more developed approach of quantum circuits and detail our implementation of its emulation.

A quantum computer is a device designed to utilise the physical properties of quantum mechanics, including interference, superposition, measurement and entanglement (Appendix B). As such a computer exploits the laws of quantum physics, a profound change in the construction of the machine over classical computers is required. Continuing efforts are being made to create such a machine; however the aim of this project is to concentrate on emulating such a quantum computer, if it were to exist, on existing conventional hardware. As details of the physical implementation of a quantum computer are beyond the scope of this project, many of the advanced physics and mathematics associated with quantum computing can be overlooked. The relevant mathematics and quantum physics required for this project is detailed in Appendix A and Appendix B, and we would strongly recommend readers to begin there.

Each section in this chapter prepares us for the proceeding topics, often building on previous knowledge and examples. As such, we recommend approaching this chapter in the order that it is presented. In addition, there may be some crossover between topics as the material is covered in order to be most easily understood by the reader. We would like to acknowledge that our presentation of topics in this chapter is inspired by the style of explanations given in Yanofsky and Mannucci [62].

2.2 Background

Quantum computing is a field combining elements of physics, mathematics, and computer science.¹ By utilising properties of quantum mechanics, quantum computing provides new opportunities in computational performance. The emergence of quantum computation has changed our perspective on many fundamental aspects of computing: the nature of information and how it flows, new algorithmic design strategies and complexity classes, and the very structure of computational models [46].

The two appendices referenced above should have prepared the reader in the relevant mathematical and quantum physics background with an angle towards quantum computing. This section will present

¹Some hold the view that quantum computing is entirely encapsulated within the subject of physics, or others that the field is purely mathematical. This project concentrates upon the computer science aspects of the field, although understanding of the related physics and mathematical topics is an important requirement.

some of the history and motivation behind the subject, followed by an explanation of the key quantum computing topics we will require. These topics will merge with the description of our implementation.

2.2.1 Beginnings

The field of quantum computing is still in its infancy, yet it has managed to alter our perspective on computation in a variety of surprising ways. Quantum mechanics itself only emerged at the beginning of the 20th century, and since then has developed into one of the most successful scientific theories of all time [18].

Feynman became interested in the exponential growth of resources required when combining quantum systems in simulations on classical computers. He argued that only a quantum computer could simulate quantum physics effectively [25]; an idea that is still upheld by modern science.² He considered exploiting the physics of quantum systems to perform computational tasks; after all, quantum computation would surely be more suited to simulating quantum systems than was classically possible [26].

Deutsch created quantum generalisations of the Turing machine and the universal Turing machine, and demonstrated the things possible on the universal quantum computer (universal quantum Turing machine) that were impossible on its classical counterpart. This included generating truly random numbers, performing parallel calculations in a single register, and perfectly simulating physical systems with finite-dimensional state spaces.[14, 62]

Deutsch continued working in the field of quantum computing. His next breakthrough was to develop the quantum circuit architecture that we go on to describe in the following sections. He also showed that his new model was equivalent to the universal quantum computer he had developed, and vice-versa [15]. As more scientists became involved in the subject, quantum gates, shortly followed by quantum algorithms, pushed the discipline forwards.

Continuing efforts are being made to develop a scalable quantum computer; however, as yet, no viable hardware has been created with more than a few qubits. As a step towards a solution, DiVincenzo put forward five criteria in order to define what a quantum computer should be [21, 22]:

1. **Physically scalable:** A scalable physical system with well characterised qubits
2. **Arbitrary qubit initialisation:** The ability to initialise the state of the qubits to a simple fiducial state, such as $|000\dots\rangle$
3. **Fast quantum gates:** Long relevant decoherence times, much longer than the gate operation time
4. **Universal gate set:** A *universal* set of quantum gates
5. **Easy to read qubits:** A highly quantum efficient, qubit-specific measurement capability

For the purposes of this project we have no interest in the hardware implementation of a quantum computer, as we are only concerned with emulating the behaviour of such a device, if it were to exist.

We will look at quantum circuits, quantum gates, and mention some of the key quantum algorithms in the following sections. Our focus in the next chapter will be upon a new architecture called measurement-based quantum computation, for which we will ultimately develop a second emulator.

2.2.2 Existing Solutions

Research into quantum computing is vast and varied across the fields of computer science, mathematics and physics. Much of the current research effort is being put towards developing a viable physical implementation. More recently we see texts aimed specifically at computer scientists, with a number

²These were not the very first thoughts of quantum computing; however it is the common starting point to begin the story.

of simulations, experiments and discussions appearing. We consider a number of the most popular solutions describing themselves as emulations or simulations. For a longer list of quantum simulators and libraries, see [49].

2.2.2.1 QDD - A Quantum Computer Emulation Library

QDD describes itself as a C++ library emulating quantum computation based upon a Binary Decision Diagram representation of quantum states [32]. They claim the use of BDDs allows the modelling of large quantum states along with a high degree of performance.

Our first comment is to question the use of *emulation* in the title of this solution. We see immediately that the representation differs massively from the unitary matrices described by the theory, with the reasoning being for improved efficiency. On top of this, upon inspection of the source code we see classical algorithms used to solve the quantum problems, instead of applying the relevant theory. As such, we describe this solution purely as a simulation.

The benchmark provided is that QDD is able to factor a 16 bit number (requiring 24 qubits) using Shor's algorithm on a P200 with 64MB of RAM. This achievement is likely much ahead of what we will achieve in terms of qubits represented.

2.2.2.2 jQuantum - Quantum Computer Simulator

jQuantum is a Java implementation of a quantum computer simulator [13]. It includes an interactive GUI with which you are able to develop circuits and view qubit states as the circuit is running.

The simulation can run 15 qubits per quantum register. We experiment by factoring the number 437 using Shor's algorithm which takes less than a second to complete. A variation of Shor's algorithm is used such that the top and bottom sets of qubits are not combined.

From viewing the software's source code it is clear it has been written as a simulation, and makes no attempt to perform steps faithfully to quantum theory.

2.2.2.3 QCAD - GUI Environment for Quantum Computer Simulator

QCAD is written in C++ and provides similar functionality to jQuantum [60]. It has a less flexible circuit builder, but more options in terms of viewing qubit states and outputting images.

The author claims to simulate 20 qubits; however when we tested a circuit with 20 qubits the program froze for a number of minutes, before returning in an unusable and inconsistent state. Unfortunately the source code for QCAD is not available to the public so we cannot comment on how it is written.

2.2.2.4 QCE - A Simulator for Quantum Computer Hardware

QCE is a tool that emulates different hardware designs of quantum computers [41]. This tool touches on a different side of quantum emulation, providing a simulation of quantum computation on different physically realisable quantum hardware, as opposed to a particular approach.

Upon reviewing the related paper, we see that algorithms are implemented using elementary operations that change the state of qubit representations. The emulation goes into a greater level of detail than we aim to, and operations are taken down to the most basic hardware level such as the exact spin of qubit and timings of operations.

2.2.3 Further Reading

Although a new subject, there is a range of well written literature describing all aspects of quantum computing. As with our Appendices, we present only the information required for our purposes. For an introductory text aimed at a computer scientist with little or no maths expertise, we strongly recommend [62]. A cut down version of this text covers the key ideas of the circuit architecture,

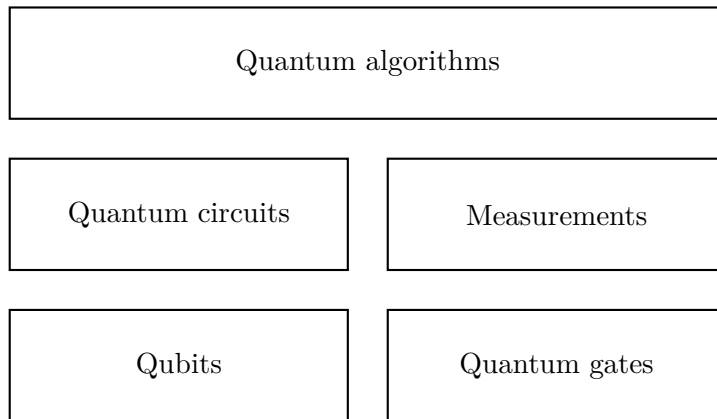


Figure 2.1: Structure of quantum circuit emulator

including a very brief look at Deutsch’s algorithm [61]. Much more information on many aspects of quantum computing is available in the comprehensive text [46].

2.3 Problem Specification & Overview of Implementation

Our initial project aim is to develop an emulation of quantum computation using the circuit model. For this we require representations of qubits and quantum gates in order to develop quantum algorithms. Our representations must, most importantly, correctly emulate the properties and behaviour of their real life counterparts as defined by quantum theory.

We outline the general structure of our implementation in terms of quantum components in Figure 2.1. We present our description of the components of the emulator in the following sections from the bottom of this diagram to the top. The structure is based upon the requirements of a real quantum system. With qubits and quantum gates we are able to create quantum circuits, with which, along with taking measurements of qubits, we are able to create quantum algorithms.

In addition to our emulation, we will present a circuit calculus which we use to formally specify quantum circuits.

2.4 Quantum Bits (Qubits)

We will begin our story of quantum computing with the fundamental unit of quantum information, the quantum bit. With the benefit of quantum mechanical phenomena, the quantum bit holds much more power than its classical counterpart. In this section we will introduce the qubit and compare it to its classical equivalent. Most importantly, we will demonstrate how we are able exploit the phenomenon of superposition to store infinitely more information than is classically possible, although we see that we can only extract a classical bit of information from it.

2.4.1 Classical vs. Quantum Bits

A conventional bit can represent two states, termed 0 and 1. This allows us to store one piece of information: a yes or no (a Boolean value: `bool classicalBit;`). As such, the state of a bit can be described as³

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \text{or} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \quad (2.1)$$

³Our use of *or* here means *exclusive or* (xor).

We have used ket notation here; for a reminder, see Appendix B.6. In code, we have `classicalBit = false`; and `classicalBit = true`; . By definition of a Boolean value, there is no other state the classical bit can take.

With the ability to represent a superposition of states, a qubit can be a 0, 1, or a superposition of both. We represent a qubit ψ in terms of the classical bits,

$$|\psi\rangle = c_0|0\rangle + c_1|1\rangle, \quad (2.2)$$

where c_0 and c_1 are complex numbers ($c_0, c_1 \in \mathbb{C}$). In this form we say that $|0\rangle$ and $|1\rangle$ is the canonical basis of \mathbb{C}^2 , the two-dimensional complex vector space.⁴ If we take our qubit as a column vector we have

$$|\psi\rangle = \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}, \quad (2.3)$$

with the restriction that $|c_0|^2 + |c_1|^2 = 1$. In code: `std::complex<double> qubit[2] = { c0, c1 };`

We can see that the pure states of a qubit are equal to the states of a classical bit. We recall from Appendix B that when a measurement of quantum system is taken, the system will collapse to a single state. When measurements of a qubit are made in the standard basis, the qubit's state collapses to a pure state (a classical bit), and so we never see (measure) a superposition state, only ever what looks like a classical bit. The different forms we will use to represent qubits are all interchangeable:

$$|\psi\rangle = c_0|0\rangle + c_1|1\rangle = c_0 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + c_1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}. \quad (2.4)$$

Building on our previous discussion, any vector $V \in \mathbb{C}^2$ can be made into a qubit. If we take the very first complex number examples in our text as V (see Appendix A.4A.3),

$$V = \begin{bmatrix} 2 + 3i \\ 17 + 5i \end{bmatrix}, \quad (2.5)$$

we can take the modulus, giving⁵

$$|V| = \sqrt{\begin{bmatrix} 2 - 3i & 17 - 5i \end{bmatrix} \begin{bmatrix} 2 + 3i \\ 17 + 5i \end{bmatrix}} = \sqrt{13 + 314} = \sqrt{327}. \quad (2.6)$$

We can now normalise the entries to give the qubit

$$\psi = \frac{2 + 3i}{\sqrt{327}}|0\rangle + \frac{17 + 5i}{\sqrt{327}}|1\rangle. \quad (2.7)$$

For clarity, the probability that we will measure a 0 (state $|0\rangle$) is $\left|\frac{2+3i}{\sqrt{327}}\right|^2 = \frac{13}{327}$, and the probability of measuring a 1 (state $|1\rangle$) is $\left|\frac{17+5i}{\sqrt{327}}\right|^2 = \frac{314}{327}$. Our restriction that the probabilities sum to 1 holds: $\frac{13}{327} + \frac{314}{327} = 1$.

⁴We have chosen to define qubits as a superposition of two states; however it may in fact be defined as a superposition of any number of states. The literature uses two states simply as it analogous to classical bits. This decision will likely be most strongly influenced by factors relating to the physical implementation of quantum computers.

⁵For a reminder of taking the modulus of a complex vector see A.4.6, in particular Equation A.13.

2.4.2 Combining Qubits

Now that we have an understanding of individual qubits, we must look how to use more than one qubit together. A common collection of classical bits is eight, termed a byte. We have our eight bits, individually represented as

$$10011011, \tag{2.8}$$

$$|1\rangle, |0\rangle, |0\rangle, |1\rangle, |1\rangle, |0\rangle, |1\rangle, |1\rangle, \tag{2.9}$$

or any of the other forms described in 2.4.1.

We can combine these states using the tensor product:⁶

$$|1\rangle \otimes |0\rangle \otimes |0\rangle \otimes |1\rangle \otimes |1\rangle \otimes |0\rangle \otimes |1\rangle \otimes |1\rangle. \tag{2.10}$$

If we take Equation 2.10 as a qubit, it is an element of the vector space $(\mathbb{C}^2)^{\otimes 8} = \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2$. We can see that this is of dimension $2^8 = 256$, so the resulting complex vector could only be an element of \mathbb{C}^{256} . If we attempted to represent this as a column vector, we would have a vector of 256 rows. When in a pure state, as our example here is, the vector would contain a single entry 1, with the remaining entries all 0.

Generalising this for complex vectors in the complex vector space \mathbb{C}^{256} we would have

$$|\psi\rangle = \left[\begin{array}{cccccccc} c_0 & c_1 & c_2 & \dots & c_{254} & c_{255} \end{array} \right]^T \tag{2.11}$$

where we maintain our probability restriction via

$$\sum_{j=0}^{255} |c_j|^2 = 1. \tag{2.12}$$

As we can see, to write the state of eight qubits we require 256 complex numbers.

To represent a 32 qubit register we require $2^{32} = 4,294,967,296$, and to represent a 64 qubit register we require a massive $2^{64} = 18,446,744,073,709,551,616$; over eighteen quintillion.

2.4.3 Implementing Qubits

We require a number of things from our representation of a qubit: a validity check to ensure the modulus squared of our qubit entries sum to 1, correct measurement properties and accurate probabilities, and the ability to combine qubits. Where we are performing operations upon individual or combined qubits we require fast and efficient multiplications.

As our implementation will be in C++ we have chosen to use the Boost C++ libraries, specifically the uBLAS library which provides basic linear algebra constructions and operations [12]. As such, to represent a qubit we use the `ublas::vector` type instead of an array, and our above representation of a qubit is replaced by

```
1 ublas::vector<std::complex<double>> qubit(2);
2 qubit(0) = c0;
3 qubit(1) = c1;
```

Where we need to instantiate large vectors we quickly generate large amounts of code. To ease this we write simple functions to instantiate vectors from arrays:

```
1 ublas::vector<std::complex<double>> qubit = ublas::makeVec(2,
2     (int[]) { c0, c1 });
```

⁶When combining qubits we must remember that the tensor product operator is not commutative: $|A\rangle \otimes |B\rangle \neq |B\rangle \otimes |A\rangle$. For a reminder of the tensor product see A.8.12.

This line represents $|\psi\rangle = c_0|0\rangle + c_1|1\rangle$ shown in Equation 2.2. We will see how the `makeVec` shortcut becomes very useful when instantiating matrices.

By using the uBLAS library we immediately have access to the basic linear algebra operations we will require. If we determine that quicker or more efficient algorithms are required, we can simply replace the uBLAS operations with our own versions. We must bear in mind that our aim here is to implement a quantum emulation, and speed concerns are secondary to correctness.

We provide an outline of our qubit class:

```

1 typedef std::complex<double> CD;
2 class qubit {
3 private:
4     ublas::vector<CD> data;
5     bool isValid(void);
6     int collapse(int);
7 public:
8     qubit(CD, CD);
9     qubit(ublas::vector<CD>);
10    virtual ~qubit(void);
11    ublas::vector<CD> getData(void);
12    int setData(ublas::vector<CD>);
13    int getSize(void);
14    bool measure(void);
15 };

```

Our next requirement is to be able to validate the state of a qubit. We take this from $\sum_{j=0}^{255} |c_j|^2 = 1$ defined in Equation 2.12 above. For this we show the `isValid()` function:

```

1 bool qubit::isValid(void)
2 {
3     double sum = 0;
4     for (unsigned int i = 0; i < data.size(); ++i)
5         sum += std::norm(data[i]);
6     return (std::fabs(sum - 1) < ALLOWABLE_QUBIT_ERROR);
7 }

```

We run this function before measurement to ensure the state we are in has not been distorted too much by rounding errors. Assuming correct and accurate function, with valid inputs, this function should never return `false`. In a real quantum system no such validation is required as it is inherent in its operation. If validation fails, we may wish to run a normalisation function on the qubit state; however doing so would distort the quantum state.

With a valid qubit state, we are then able to measure our qubit in the standard basis $|0\rangle$ and $|1\rangle$. The behaviour we require is that a result (0 or 1) is returned with the correct probability according to the state of the qubit. Once a measurement has been completed the state of the qubit should collapse to this state. The requirements for this behaviour are described in B.6.3. Our functions are defined as

```

1 bool qubit::measure(int qubit)
2 {
3     // Invalid qubit state
4     if (!isValid())
5         return -1;
6
7     double rand = getRand(0, RAND_RANGE);
8     for (unsigned int i = 0; i < data.size(); ++i)
9     {
10        rand -= std::norm(data[i]) * RAND_RANGE;
11        if (rand <= 0)

```

```

12     {
13         collapse(i);
14         return getBit(i, qubit));
15     }
16 }
17 // Invalid measurement
18 return -1;
19 }

```

```

1 int qubit::collapse(int entry)
2 {
3     for (unsigned int i = 0; i < data.size(); ++i)
4         data(i) = 0;
5     data(entry) = 1;
6     return entry;
7 }

```

Where we wish to combine qubits we take the tensor product of their two data vectors and create a new qubit, as described in 2.4.2. We calculate the tensor product of two vectors with a function that simply implements the tensor product definition (without algorithmic improvements):

```

1 ublas::vector<CD> kron(ublas::vector<CD> v1, ublas::vector<CD> v2)
2 {
3     unsigned int v1size = v1.size();
4     unsigned int v2size = v2.size();
5     ublas::vector<CD> v3(v1size * v2size);
6     for (unsigned int i = 0; i < v1size; ++i)
7         for (unsigned int j = 0; j < v2size; ++j)
8             v3((i * v2size) + j) = v1[i] * v2[j];
9     return v3;
10 }

```

With this, combining two qubits simply becomes

```

1 qubit combinedQubit(kron(qubit1.getData(), qubit2.getData()));

```

2.5 Quantum Gates

In order to manipulate our newly discovered qubits, as with classical bits, we require logical gates. Gates are operators upon the system, and as we describe dynamics with matrices in our quantum mechanics appendix, we shall do the same with gates.

2.5.1 Classical Logic Gates

The most simple classical gate is the NOT gate (inverter). It takes one input, inverts it (changes a 0 to a 1 and vice-versa), and then returns it via a single output. We shall model this gate as a 2 by 2 matrix

$$\text{NOT} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \quad (2.13)$$

We can test this model by providing inputs $|0\rangle$ and $|1\rangle$, and ensuring we receive the expected output, $|1\rangle$ and $|0\rangle$ respectively, after matrix multiplication:

$$\text{NOT}|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |1\rangle, \quad (2.14)$$

$$\text{NOT}|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |0\rangle. \quad (2.15)$$

Where gates have multiple inputs or outputs we require some extra thought, and must consider the definition of matrix multiplication.⁷ Where we have an input vector of dimension 2 and output vector of the same size we require an operator which is 2 by 2. Where we have an input vector of dimension 4 and output of dimension 2 we require a 4 by 2 operator. We can in fact generalise this such that with a 2^n dimension input and 2^m dimension output vector, our operator will be of size 2^m by 2^n .

By considering the Boolean logic implied by the various classical gates, we can easily define matrices for the common classical logic gates:

$$\text{AND} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \text{NAND} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}, \quad (2.16)$$

$$\text{OR} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}, \quad \text{NOR} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad (2.17)$$

When combining classical gates, or in fact combining any operations, we have two ways in which this can be done. The operations can either be performed in parallel, at the same time, or in sequence, one followed by the other.

We will take two operations (matrices) M and N and combine them. We have shown in Appendix B how to perform operations in parallel by using the tensor product: $M \otimes N$. We can just as easily perform the operations sequentially by performing matrix multiplication of the second matrix by the first: NM . As you would expect, we can build our NAND gate by combining our NOT and AND gates via matrix multiplication. As we build up these combinations we can begin to build up a circuit.

2.5.2 Quantum Gates

In our introduction to quantum mechanics our quantum dynamics have been defined by unitary matrices, and quantum gates are the same. Instead of column vectors representing classical bit states, we now replace these with their complex number quantum equivalents to allow interference to occur.

One of the most simple quantum gates is formed from the Hadamard matrix:⁸

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix}. \quad (2.18)$$

It is one of the most important matrices in quantum computing, and has the interesting property that it is its own inverse. We will come to see that all reversible gates are their own inverse.⁹ The reversibility of gates is a huge factor in computing. Where we erase information, as opposed to writing information, the result is energy loss and the generation of heat (known as Landauer's principle). It is an irreversible and energy-dissipating operation. Our reversible gates do not face this problem, and this leads us to believe that utilising reversible gates is a more energy efficient method of computation. This idea is described in more detail in [62].

⁷See A.8.4 for a reminder of matrix multiplication. In particular note that it is a function $\mathbb{C}^{m \times n} \times \mathbb{C}^{n \times p} \rightarrow \mathbb{C}^{m \times p}$.

⁸In general we will take out normalisation factors, in this case $\frac{1}{\sqrt{2}}$, to aid reading.

⁹We see that our NOT gate defined above is reversible:

$$\text{NOTNOT} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I. \quad (2.19)$$

Other than the identity gate, it is the only reversible classical logic gate of those we have described.

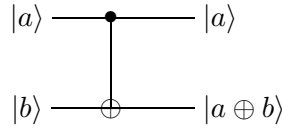


Figure 2.2: Controlled-NOT (CX) quantum gate

Another reversible quantum gate is the controlled-NOT gate (CX). This differs from a conventional NOT gate in that it can be turned on or off (controlled) by a control input. Figure 2.2 shows the gate with two inputs and two outputs. Our top input, $|a\rangle$, is the control, and bottom input, $|b\rangle$, will be inverted when $|a\rangle = 1$. The top output will remain $|a\rangle$ in all cases, while we can describe the lower output as the xor of both inputs, $|a \oplus b\rangle$.¹⁰ Combined, our input is $|a, b\rangle$ and output is $|a, a \oplus b\rangle$, giving us the matrix for the gate

$$CX = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \quad (2.20)$$

We can define this as shown in Figure 2.2¹¹, or as

$$CX|i, j\rangle = |i, i \oplus j\rangle. \quad (2.21)$$

We maintain reversibility by combining states in such a way that we are able to decompose outputs to return us to our inputs.

Any controlled- U (CU) gate can be generated from the gate $U = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ by forming the matrix

$$CU = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix}. \quad (2.22)$$

This construction is also valid for larger matrices. Note that creating an equivalent CU where the bottom qubit is the control is, perhaps surprisingly, not similar. We will explain this difference below.

The Toffoli (Figure 2.3) gate is similar to CX but has two controlling bits. The bottom bit flips only when both of the top two bits are $|1\rangle$. Our input is $|a, b, c\rangle$ and output is $|a, b, c \oplus (a \wedge b)\rangle$, producing the matrix

$$\text{Toffoli} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}. \quad (2.23)$$

The Toffoli gate has the additional property that it is universal, meaning with only Toffoli gates we can construct any other logical gate.

¹⁰Our use of \oplus here is to mean “exclusive or” (xor), otherwise termed addition modulo 2.

¹¹For a key of the circuit model symbols, see Appendix C.2.

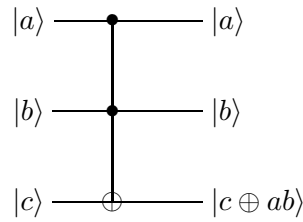


Figure 2.3: Toffoli quantum gate

We also have a second universal gate, the Fredkin gate. Where we have input $|0, b, c\rangle$, our output is unchanged, and where we have input $|1, b, c\rangle$, our output is $|1, c, b\rangle$ as the bottom two bits are swapped. This corresponds to the matrix

$$\text{Fredkin} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (2.24)$$

The Pauli operators will become important to us as they are used to change the basis of a qubit, and are defined as

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \quad (2.25)$$

We can see that our NOT gate is the Pauli X operator, and the 2 by 2 identity matrix I is also a Pauli operator.

Other common quantum gates include the phase gate (S), $\pi/8$ (T), controlled-Z (CZ) and controlled-phase (CS):

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, \quad T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}, \quad (2.26)$$

$$CZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}, \quad CS = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & i \end{bmatrix}. \quad (2.27)$$

To maintain the reversibility property of our quantum gates, it is clear that the number of inputs must always equal the number of outputs. As such, matrices representing quantum gates are always square. The gates we have described so far have a set number of inputs and outputs.

2.5.3 Combining Quantum Gates

For our purposes we must be able to manipulate an arbitrary number of qubits with our set of gates. We will first address a specific example.

We wish to apply the Hadamard gate to two qubits at the same time. For this we have two qubit inputs, and so require a 4 by 4 matrix operator. To get this we find the tensor product of the

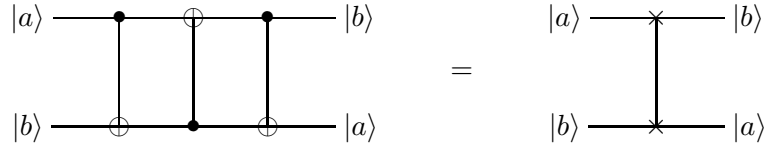


Figure 2.4: Defining the quantum *swap* gate in terms of three controlled- X gates

Hadamard gate and itself:

$$\begin{aligned}
 H \otimes H &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\
 &= \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}. \tag{2.28}
 \end{aligned}$$

Doing the same for three qubits would require $H \otimes H \otimes H$, and in fact for any n qubit inputs we have $H^{\otimes n}$. We can generalise this notion to any gate U by considering it as an application of a U gate to each qubit. Given n qubit inputs, we can perform the operator on all inputs by finding the operator $U^{\otimes n}$.

A second example again involves multiple qubits; however we only wish to apply an H gate to a single one. Where we wish to apply a Hadamard gate to only the second of two qubits, we create an operator through the tensor product $I \otimes H$. Or the middle of three qubits, we take the tensor product $I \otimes H \otimes I$. In general, if we wish to apply the gate U to the m th of n qubits we take the tensor product

$$I_{2^{m-1}} \otimes U \otimes I_{2^{n-m}}, \tag{2.29}$$

where I_x represents the x by x identity matrix. A final generalisation is to apply the gate U to p qubits out of n , beginning at the m th qubit:

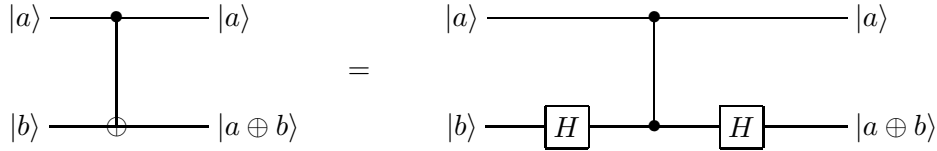
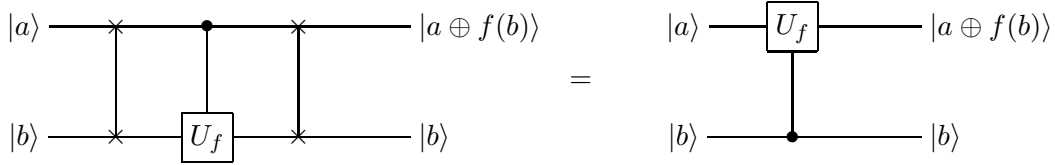
$$I_{2^{m-1}} \otimes U^{\otimes p} \otimes I_{2^{n-m-p+1}}. \tag{2.30}$$

By combining three gates together we will create a useful operator, the *swap* gate, illustrated in Figure 2.4. We use the CX gate described above, and we will flip the gate such that our bottom qubit is the control, and our X gate will be performed on the top qubit (we will call this XC):

$$CX = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad XC = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{2.31}$$

From the diagram we can see we require the result of the matrix multiplication of CX , XC and CX :

$$CXXCX = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$


 Figure 2.5: Equivalence between controlled- X gate and controlled- Z

 Figure 2.6: Creating a swapped controlled- U gate using *swap* gates

$$\text{swap} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (2.32)$$

In the process, by combining several gates together we have created our first quantum circuit.

Although we may represent the swap gate as a single gate in a circuit, we see it can be decomposed into CX gates. In turn, the CX gate may be decomposed (as it is equivalent) to two H gates and a CZ , as shown in Figure 2.5. In notation we have $CX = (I \otimes H) CZ (I \otimes H)$. The reverse is also true: $CZ = (I \otimes H) CX (I \otimes H)$. We say that a set of universal quantum gates is a set able to implement any other quantum gate, and so all operators may be decomposed into gates from this set. An example of such a set is the H , CX and T gates.

Like creating controlled- U gates, we have a general construction for creating swapped controlled- U gates. From a gate $U = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ we create an inverted controlled- U gate via

$$UC = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & a & 0 & b \\ 0 & 0 & 1 & 0 \\ 0 & c & 0 & d \end{bmatrix}. \quad (2.33)$$

We have created this elegant generalisation not usually found in the literature by combining our construction for controlled- U gates with the *swap* gate we have just defined (Figure 2.6):

$$\begin{aligned} UC &= \text{swap } CU \text{ swap} \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & a & 0 & b \\ 0 & 0 & 1 & 0 \\ 0 & c & 0 & d \end{bmatrix}. \end{aligned} \quad (2.34)$$

We have a similar construction where we have two control qubits:

$$\begin{aligned}
 UCC &= (I \otimes \text{swap})(\text{swap} \otimes I)(I \otimes \text{swap}) CCU (I \otimes \text{swap})(\text{swap} \otimes I)(I \otimes \text{swap}) \\
 &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & a & 0 & b \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & c & 0 & d \end{bmatrix}. \tag{2.35}
 \end{aligned}$$

For gates with more than two control qubits we follow a similar method. We use *swap* gates to reverse the order of inputs, perform our controlled- U , then reverse the order back.

With the information provided, we are ready to discuss our implementation of quantum gates, including combining gates in parallel and in series, whilst maintaining the behaviour of quantum systems.

2.5.4 Implementing Quantum Gates

We use the construction `ublas::matrix<CD>` to represent our gates. As with vectors, we initialise the basic quantum gates as constants. For example we define the Pauli gates

```

1 typedef std::complex<double> CD;
2 const CD I = CD(0, 1);
3 static const matrix *X_MAT = new matrix(ublas::makeMat(2, 2, (int[]) {
4     0, 1,
5     1, 0 }));
6 static const matrix *Y_MAT = new matrix(ublas::makeMat(2, 2, (CD[]) {
7     0, -I,
8     I, 0 }));
9 static const matrix *Z_MAT = new matrix(ublas::makeMat(2, 2, (int[]) {
10    1, 0,
11    0, -1 }));
    
```

from which we can use the tensor product to create gates of various required sizes and combinations. These are directly equivalent to the mathematical matrices we have been discussing so far.

We create a function to find the tensor product of matrices:

```

1 ublas::matrix<CD> kron(ublas::matrix<CD> m1, ublas::matrix<CD> m2)
2 {
3     unsigned int m2rows = m2.size1();
4     unsigned int m2cols = m2.size2();
5     unsigned int rows = m1.size1() * m2rows;
6     unsigned int cols = m1.size2() * m2cols;
7     ublas::matrix<CD> m3 = ublas::matrix<CD>(rows, cols);
8     for (unsigned int i = 0; i < rows; ++i)
9         for (unsigned int j = 0; j < cols; ++j)
10            m3(i, j) = m1(i/m2rows, j/m2cols) * m2(i%m2rows, j%m2cols);
11     return m3;
12 }
    
```

with which we can perform operations such as $U \otimes U$:

```

1 ublas::matrix<CD> u2 = kron(U_GATE, U_GATE);
    
```

For ease, we define a function to perform the operation $U^{\otimes n}$:

```

1  ublas::matrix<CD> kronPow(ublas::matrix<CD> m1, unsigned int size)
2  {
3      ublas::matrix<CD> m2 = ublas::matrix(m1);
4      while (m2.size1() < size)
5          m2 = kron(m2, m1);
6      return m2;
7  }

```

which will repeat the tensor product until the resulting matrix is of the desired size.

By exploiting our `kron()` and `kronPow()` functions above we can dynamically and easily generate new gates of any size in order to apply multiple gates in parallel.¹² We will show an example where we have three entangled qubits and wish to apply a Hadamard gate to only the third qubit. To accomplish this we simply take the tensor product of the identity gate with itself, followed by the tensor product of this with the Hadamard gate: $I \otimes I \otimes H$. In code we have

```

1  ublas::matrix<CD> combinedGate = kron(kron(I_GATE, I_GATE), H_GATE);

```

which, in this case, generates an 8 by 8 gate (matrix operator) performing an H gate to the third (bottom) qubit, and leaving the first and second qubits untouched.

We package these methods together and provide a simple `apply()` function with a number of method overloads to accept a single qubit input, multiple qubit inputs, a gate type selector, selections for different variations of a gate¹³, or simply a predetermined matrix representing a gate. We show the base `apply()` methods below:

```

1  int gate::apply(qubit *quIns [], GateType type, int opt, qubit *quOut)
2  {
3      // Apply new qubit data to pre-prepared output qubit
4      return quOut->setData(_apply(quIns, getGate(type, opt)));
5  }
6  ublas::vector<CD> gate::_apply(qubit *quIns [], ublas::matrix<CD> gateVal)
7  {
8      ublas::vector<CD> quIn = quIns[0]->getData();
9      // Combine qubits if multiple specified
10     for (unsigned int i = 1; quIn.size() < N; ++i)
11         quIn = kron(quIn, quIns[i]->getData());
12
13     // Perform gate * qubit matrix multiplication
14     ublas::vector<CD> quOut = ublas::prec_prod(gateVal, quIn);
15     return quOut;
16 }

```

From what we have shown we are able to generate gates of our choosing using the `kron` functionality, grouped together with the `getGate()` function. We are able to perform the operation upon one or multiple qubits using the `apply()` methods, with specific gates applying themselves to specific qubits. Armed with this, we proceed towards our goal of implementing quantum algorithms.

¹²We remember that gates applied in series are done via matrix multiplication, implemented by our UBLAS library. The application of gates in series effectively represents rounds of time, and is how we create circuits, and so how we form algorithms.

¹³We will require this functionality for certain algorithms such as Deutsch, where we use a U_{f_x} gate where $x \in \{1, 2, 3, 4\}$.

2.6 Quantum Algorithms

Now that we have outlined the concepts and our implementation of qubits and gates, we are able to move on to create quantum circuits, and ultimately emulate quantum algorithms.

Developing quantum algorithms requires a whole new way of thinking for scientists. With this said, the quantum circuit model is unmistakably analogous to electrical or Boolean logic circuits. As we move on to discuss the measurement-based model, we have no analogue, and forming intuition becomes much harder.

In this section we look at some of the more prominent algorithms to be developed for the quantum circuit model. We begin each algorithm with a brief description, followed by details of our implementation. We work our way up in difficulty to Shor's factoring algorithm which is able to factor numbers in polynomial time. There is no known classical algorithm able to do the same in this time.

Generally, there are four steps involved in quantum algorithms. They are:

1. input qubits are initialised into some classical start state;
2. the system is put into some superposition state;
3. the superposition state is acted upon via unitary operations;
4. some measurement of the system is taken, providing a classical output state.

2.6.1 Deutsch's Algorithm

Deutsch's algorithm creates four functions mapping the set of classical bit values to itself [14]. These four functions are:

$$\begin{aligned}
 f_1 &: 0 \mapsto 0, \quad 1 \mapsto 0 \\
 f_2 &: 0 \mapsto 1, \quad 1 \mapsto 1 \\
 f_3 &: 0 \mapsto 0, \quad 1 \mapsto 1 \\
 f_4 &: 0 \mapsto 1, \quad 1 \mapsto 0.
 \end{aligned} \tag{2.36}$$

We can see that every possible mapping $\{0,1\} \mapsto \{0,1\}$ is covered by these four cases. Without knowing which of these functions is being used, Deutsch's algorithm determines whether the function f is constant or balanced. We define constant as $f(0) = f(1)$ (as is the case for f_1 and f_2), and balanced as $f(0) \neq f(1)$ (as for f_3 and f_4). We can also represent these mappings as the matrices

$$f_1 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, \quad f_2 = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}, \quad f_3 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad f_4 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \tag{2.37}$$

taking the column number as the input, and row number as the output.¹⁴ We see that our constant functions have rows of 1s and 0s, whereas our balanced functions have diagonals. Our function f is represented in the circuit as the gate U_f . Which of the four functions U_f implements is unknown to the algorithm.

A classical algorithm would first evaluate $f(0)$, followed by $f(1)$, and then perform a comparison between the two. A quantum computer can be in a superposition of two basic states, and so is able to evaluate both $f(0)$ and $f(1)$ at the same time. We do this by creating an extremely simple circuit, with the U_f gate implementing the unknown function we wish to investigate, three H gates, and a

¹⁴We count columns and rows starting at 0, so column 0 of the matrix represents the case where the input is 0, and column 1 represents the input 1. Where we see an entry 1, the corresponding input to output according to the column and row numbers is part of the function's mapping. For further explanation, see Equation 2.39 in the description of U_f matrices below.

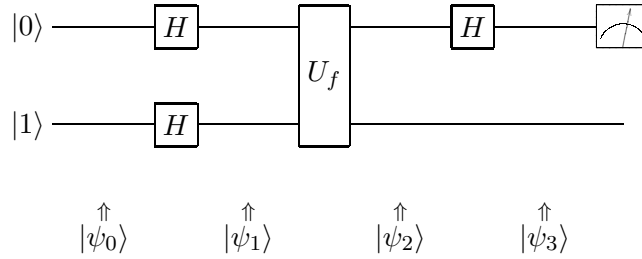


Figure 2.7: Quantum circuit implementing Deutsch’s algorithm

single measurement. This provides us the result with only one evaluation of f using both inputs at once. We show a diagram of the quantum circuit implementing Deutsch’s algorithm in Figure 2.7.

An important note to consider is that although the problem solved by this algorithm may appear contrived, it clearly demonstrates the advantage of our quantum solution over classical alternatives.

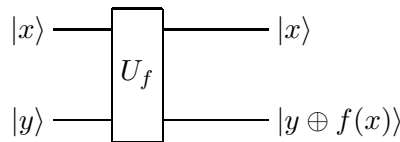
2.6.2 Implementing Deutsch’s Algorithm

We will demonstrate our implementation of Deutsch’s algorithm along with a more detailed explanation of each step. Before we explore the algorithm itself, we will first look at the corresponding U_f gates for the four f functions.

For our four functions defined above we must create quantum gates that implement f in an acceptable (reversible) manner. In general, we create a U_f gate acting on qubit inputs $|x\rangle$ and $|y\rangle$ as

$$U_f|x, y\rangle = |x, y \oplus f(x)\rangle. \tag{2.38}$$

Diagrammatically this is



With this, we define the mappings (inputs to outputs) we require for the four functions, the first two of which are constant functions and the second two balanced:

$$\begin{aligned} U_{f_1} : & \quad |0, 0\rangle \mapsto |0, 0\rangle, \quad |0, 1\rangle \mapsto |0, 1\rangle, \quad |1, 0\rangle \mapsto |1, 0\rangle, \quad |1, 1\rangle \mapsto |1, 1\rangle \\ U_{f_2} : & \quad |0, 0\rangle \mapsto |0, 1\rangle, \quad |0, 1\rangle \mapsto |0, 0\rangle, \quad |1, 0\rangle \mapsto |1, 1\rangle, \quad |1, 1\rangle \mapsto |1, 0\rangle \\ U_{f_3} : & \quad |0, 0\rangle \mapsto |0, 0\rangle, \quad |0, 1\rangle \mapsto |0, 1\rangle, \quad |1, 0\rangle \mapsto |1, 1\rangle, \quad |1, 1\rangle \mapsto |1, 0\rangle \\ U_{f_4} : & \quad |0, 0\rangle \mapsto |0, 1\rangle, \quad |0, 1\rangle \mapsto |0, 0\rangle, \quad |1, 0\rangle \mapsto |1, 0\rangle, \quad |1, 1\rangle \mapsto |1, 1\rangle. \end{aligned}$$

We show the corresponding matrix operator for a particular example U_{f_4} ,

$$U_{f_4} = \begin{matrix} & \mathbf{00} & \mathbf{01} & \mathbf{10} & \mathbf{11} \\ \mathbf{00} & 0 & 1 & 0 & 0 \\ \mathbf{01} & 1 & 0 & 0 & 0 \\ \mathbf{10} & 0 & 0 & 1 & 0 \\ \mathbf{11} & 0 & 0 & 0 & 1 \end{matrix}, \tag{2.39}$$

with the column headings representing inputs and the row headings showing outputs. For example,

the input 00 would provide the output 01, while the input 10 would give 10 as output.¹⁵ Ensuring that this follows the requirement given in Equation 2.38, our example input 00 gives $f(0) = 1$, and so

$$U_{f_4}|0,0\rangle = |0,0 \oplus f(0)\rangle = |0,0 \oplus 1\rangle = |0,1\rangle, \quad (2.40)$$

matching with our matrix. The input 10 gives $f(1) = 0$, and so

$$U_{f_4}|1,0\rangle = |1,0 \oplus f(1)\rangle = |1,0 \oplus 0\rangle = |1,0\rangle, \quad (2.41)$$

providing the expected result.

The corresponding matrices for our mappings are therefore

$$U_{f_1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad U_{f_2} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad U_{f_3} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad U_{f_4} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (2.42)$$

With these matrices determined we add them to our `getGate()` function (called by `apply()`) for use in our implementation of the algorithm.

In fact our implementation of the algorithm is extremely simple and is summed up in one function:

```

1  int algorithm::deutsch(int uf)
2  {
3      // Initialise qubits to |0> = [1,0] and |1> = [0,1] respectively
4      qubit q1 = qubit(1, 0);
5      qubit q2 = qubit(0, 1);
6
7      // Apply H gates to both qubits
8      gate<2>::apply(&q1, HGATE, &q1);
9      gate<2>::apply(&q2, HGATE, &q2);
10
11     // Add q1 and q2 to input array
12     qubit *qus3[] = { &q1, &q2 };
13     // Create output qubit
14     qubit q1_2 = qubit(0, 4);
15     // Apply Uf gate number 'uf', followed by an H gate
16     gate<4>::apply(qus3, UFGATE, uf, &q1_2);
17     gate<4>::apply(&q1_2, HGATE, &q1_2);
18
19     // Take measurement of top qubit
20     return q1_2.measure(0);
21 }

```

You can see this closely follows our diagram of the circuit (Figure 2.7).¹⁶ We will now step through the algorithm using a known function f_3 , and then generalise our calculations for any function f .

We begin by initialising our qubits to the state $|\psi_0\rangle = |0\rangle \otimes |1\rangle = |0,1\rangle$ (lines 4-5). After both qubits pass through H gates (lines 8-9) they are put into a superposition described by the state

$$|\psi_1\rangle = (H \otimes H)|0,1\rangle$$

¹⁵Our matrix shows this as we use the modulus squared of the corresponding matrix entry as the probability such a transformation will occur. Our example $00 \mapsto 01$ will always occur as the corresponding entry is 1, so the probability of this mapping occurring is $|1|^2 = 1$. As an example showing the opposite, $00 \mapsto 00$ will never occur, as its probability is $|0|^2 = 0$. With this in mind, we are able to interpret both complex entries and superpositions of states without a problem.

¹⁶Our algorithm here differs from the diagrammatical representation of the algorithm in line 17 as we apply an $H \otimes H$ gate as opposed to an $H \otimes I$ gate. As we do not measure the second qubit, this has no impact upon the correctness of the algorithm, and is done simply for brevity.

$$\begin{aligned}
 &= \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \\
 &= \frac{1}{2} (|0,0\rangle - |0,1\rangle + |1,0\rangle - |1,1\rangle).
 \end{aligned} \tag{2.43}$$

Using matrices we confirm this result:

$$\begin{aligned}
 |\psi_1\rangle &= \left(\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \right) |0,1\rangle \\
 &= \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}.
 \end{aligned} \tag{2.44}$$

We proceed assuming we are using the balanced function f_3 , and so will use the gate U_{f_3} . We apply this gate to both qubits, giving the state

$$\begin{aligned}
 |\psi_2\rangle &= U_{f_3}(H \otimes H)|0,1\rangle \\
 &= \frac{1}{\sqrt{2}} ((1)|0\rangle + (-1)|1\rangle) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \\
 &= \frac{1}{2} (|0,0\rangle - |0,1\rangle - |1,0\rangle + |1,1\rangle).
 \end{aligned} \tag{2.45}$$

Looking at the matrix representation we have

$$\begin{aligned}
 |\psi_2\rangle &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} (H \otimes H)|0,1\rangle \\
 &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \frac{1}{2} \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}.
 \end{aligned} \tag{2.46}$$

Our final step is to apply an H gate to the top qubit, leaving us with the final state before measurement

$$\begin{aligned}
 |\psi_3\rangle &= (H \otimes I)U_{f_3}(H \otimes H)|0,1\rangle \\
 &= |1\rangle \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \\
 &= \frac{1}{\sqrt{2}} (|1,0\rangle - |1,1\rangle).
 \end{aligned} \tag{2.47}$$

The matrix form gives us

$$\begin{aligned}
 |\psi_3\rangle &= \left(\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) U_{f_3}(H \otimes H)|0,1\rangle \\
 &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \frac{1}{2} \begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \end{bmatrix}.
 \end{aligned} \tag{2.48}$$

With the qubits in this state, when the top qubit is measured it will return $|1\rangle$ with a probability of 1, as expected for a balanced function.

To generalise the algorithm for any function, we proceed from the application of a now unknown U_f gate (line 16). Our state after this is

$$\begin{aligned}
 |\psi_2\rangle &= U_f(H \otimes H)|0, 1\rangle \\
 &= \frac{1}{\sqrt{2}} \left((-1)^{f(0)} |0\rangle + (-1)^{f(1)} |1\rangle \right) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \\
 &= \begin{cases} \frac{\pm 1}{\sqrt{2}} (|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) & \text{if } f \text{ is constant,} \\ \frac{\pm 1}{\sqrt{2}} (|0\rangle - |1\rangle) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) & \text{if } f \text{ is balanced.} \end{cases} \tag{2.49}
 \end{aligned}$$

Applying the H gate to the top qubit (line 17) gives us

$$\begin{aligned}
 |\psi_3\rangle &= (H \otimes I)U_f(H \otimes H)|0, 1\rangle \\
 &= \begin{cases} \pm 1|0\rangle \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) & \text{if } f \text{ is constant,} \\ \pm 1|1\rangle \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) & \text{if } f \text{ is balanced.} \end{cases} \tag{2.50}
 \end{aligned}$$

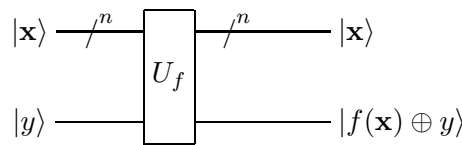
From this state we measure the top qubit as $|0\rangle$ if f is constant and $|1\rangle$ if f is balanced with 100% certainty (line 20). The state of the second qubit is irrelevant to us.

2.6.3 Deutsch-Jozsa Algorithm

The next step forward from Deutsch is to take a larger domain for the functions, moving on from $f : \{0, 1\} \mapsto \{0, 1\}$ to $f : \{0, 1\}^n \mapsto \{0, 1\}$ where $n \in \mathbb{N}$ and $n > 0$ [17]. The definition of a balanced function is now defined as one in which half of the inputs go to 0 and half go to 1. Constant functions are those where all inputs go to 0 or all go to 1. In any case where $n > 1$, there exist functions that are neither constant nor balanced which are not considered.

Assuming that we are provided with a function that is either balanced or constant, we are only able to determine which it is by evaluating the function. Whereas a classical implementation requires $2^{n-1} + 1$ evaluations, with our quantum algorithm we are again able to solve it in a single function evaluation.

Diagrammatically, Figure 2.8 shows a very close similarity to our circuit for Deutsch’s algorithm. The one key difference we notice is that our top wire now represents n qubits instead of just one. Our H gates are modified to handle this ($H^{\otimes n}$), and our U_f gate is now defined as



We must remember that our input $|x\rangle$ in fact represents $|x_0, x_1, \dots, x_{n-1}\rangle$, so we will use a bold typeface for clarity. Our function $f(\mathbf{x})$ now accepts $|\mathbf{x}\rangle$ of size n and returns a single bit.

2.6.4 Implementing the Deutsch-Jozsa Algorithm

As previously, we will begin our discussion by defining our functions. Where we take $n = 1$, our functions are exactly those described in our explanation of Deutsch’s algorithm. We will first look at the case where $n = 2$ before generalising our notion to any $n > 0$.

There are 16 possible functions with the domain $f : \{0, 1\}^2 \mapsto \{0, 1\}$, of which two are constant and six are balanced. The functions we are interested in are

$$f_1 : \quad 00 \mapsto 0, \quad 01 \mapsto 0, \quad 10 \mapsto 0, \quad 11 \mapsto 0$$

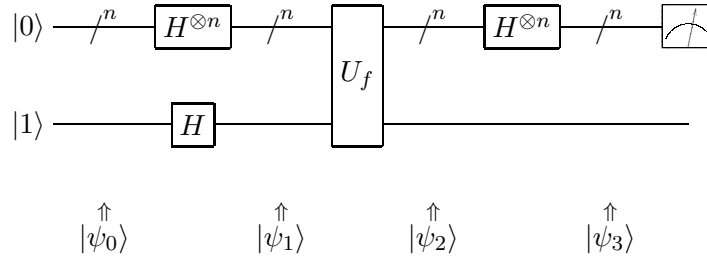


Figure 2.8: Quantum circuit implementing the general Deutsch-Jozsa algorithm

$$\begin{aligned}
 f_2 : \quad & 00 \mapsto 1, \quad 01 \mapsto 1, \quad 10 \mapsto 1, \quad 11 \mapsto 1 \\
 f_3 : \quad & 00 \mapsto 0, \quad 01 \mapsto 0, \quad 10 \mapsto 1, \quad 11 \mapsto 1 \\
 f_4 : \quad & 00 \mapsto 1, \quad 01 \mapsto 1, \quad 10 \mapsto 0, \quad 11 \mapsto 0 \\
 f_5 : \quad & 00 \mapsto 0, \quad 01 \mapsto 1, \quad 10 \mapsto 1, \quad 11 \mapsto 0 \\
 f_6 : \quad & 00 \mapsto 1, \quad 01 \mapsto 0, \quad 10 \mapsto 0, \quad 11 \mapsto 1 \\
 f_7 : \quad & 00 \mapsto 0, \quad 01 \mapsto 1, \quad 10 \mapsto 0, \quad 11 \mapsto 1 \\
 f_8 : \quad & 00 \mapsto 1, \quad 01 \mapsto 0, \quad 10 \mapsto 1, \quad 11 \mapsto 0,
 \end{aligned} \tag{2.51}$$

the first two of which are constant and the remainder are balanced. We see 2 examples of the 2 by 4 matrices representing these mappings:

$$f_1 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad f_3 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}. \tag{2.52}$$

Again, we see our constant function represented by rows of 1s and 0s, with our balanced functions now represented by even numbers of 1s and 0s in each row.

For any n we have 2^n possible inputs giving 2^n outputs (which are either 0 or 1). This gives us a total of 2^{2^n} possible functions satisfying our domain constraint. From this, we always have two constant functions outputting 0s for every input, or 1s for every input. We use the binomial coefficient,

$$\frac{m!}{(m-k)!k!}, \tag{2.53}$$

where $m = 2^n$ and $k = 2^{n-1}$ to calculate the number of balanced functions we expect for a particular n :

$$\frac{2^n!}{(2^n - 2^{n-1})!2^{n-1}!} = \frac{2^n!}{(2^{n-1}!)^2}. \tag{2.54}$$

From our example above, for $n = 2$ we have $\frac{4!}{(2!)^2} = 6$ balanced functions, as we listed.

As we showed, our U_f will implement the function f such that $U_f|\mathbf{x}, y\rangle = |\mathbf{x}, y \oplus f(\mathbf{x})\rangle$. The matrices representing the gate will be 2^{n+1} by 2^{n+1} . We show U_{f_1} and U_{f_3} of size 8 by 8 as examples,

using $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ and $0 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$:

$$U_{f_1} = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & I \end{bmatrix}, \quad U_{f_3} = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & 0 & X & 0 \\ 0 & 0 & 0 & X \end{bmatrix}. \tag{2.55}$$

We show our implementation of Deutsch-Jozsa which mirrors our circuit diagram (Figure 2.8):

```

1  /*
2   * N: number of qubits
3   * N2: 2^N
4   */
5  template<int N, int N2>
6  int algorithm::deutschJ(int uf)
7  {
8      // Initialise qubits to |0>, and control qubit to |1>
9      qubit *qus[N];
10     for (int i = 0; i < N - 1; ++i)
11         qus[i] = new qubit(1, 0);
12     qus[N - 1] = new qubit(0, 1);
13
14     // Apply H gates to all qubits
15     for (int i = 0; i < N; ++i)
16         gate<2>::apply(qus[i], HGATE, qus[i]);
17
18     // Apply Uf gate number 'uf', followed by H gate
19     qubit qubitGroup = qubit(0, N2);
20     gate<N2>::apply(qus, UFGATE, uf, &qubitGroup);
21     gate<N2>::apply(&qubitGroup, HGATE, &qubitGroup);
22
23     // Take measurement and test it
24     bool measure = false;
25     for (int i = 0; i < N - 1; ++i)
26         measure = measure || qubitGroup1.measure(i);
27     return measure;
28 }
    
```

We will step through the algorithm for $n = 2$ before generalising this to any n . Beginning with our start state, we describe this as $|0, 1\rangle$; however we must remember that our $|0\rangle$ in fact represents $|0\rangle^{\otimes n}$. For the case $n = 2$ this is simply $\psi_0 = |0, 0, 1\rangle$.

Applying our initial H gates gives the superposition

$$\begin{aligned}
 |\psi_1\rangle &= (H^{\otimes 2} \otimes H) |0\rangle^{\otimes 2} |1\rangle = (H \otimes H \otimes H) |0, 0, 1\rangle \\
 &= \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \\
 &= \frac{1}{2} (|00\rangle + |01\rangle + |10\rangle + |11\rangle) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \\
 &= \frac{1}{2\sqrt{2}} (|000\rangle - |001\rangle + |010\rangle - |011\rangle + |100\rangle - |101\rangle + |110\rangle - |111\rangle). \quad (2.56)
 \end{aligned}$$

Looking at the matrix operations involved we confirm the same result:

$$|\psi_1\rangle = \left(\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \right) |0, 0, 1\rangle$$

$$\begin{aligned}
 &= \frac{1}{2\sqrt{2}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \\
 &= \frac{1}{2\sqrt{2}} \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix}^T. \tag{2.57}
 \end{aligned}$$

Our next step is to apply the U_f gate, producing the state

$$\begin{aligned}
 |\psi_2\rangle &= U_f H^{\otimes 3} |001\rangle \\
 &= \frac{1}{2} \left((-1)^{f(00)} |00\rangle + (-1)^{f(01)} |01\rangle + (-1)^{f(10)} |10\rangle + (-1)^{f(11)} |11\rangle \right) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \tag{2.58}
 \end{aligned}$$

Finally, applying the $H^{\otimes n}$ gate to the top qubits gives us¹⁷

$$\begin{aligned}
 |\psi_3\rangle &= (H^{\otimes n} \otimes I) U_f (H^{\otimes n} \otimes H) |0, 0, 1\rangle \\
 &= \frac{1}{4} \left(\sum_{\mathbf{x} \in \{0,1\}^2} \sum_{\mathbf{y} \in \{0,1\}^2} (-1)^{f(\mathbf{x}) \oplus \mathbf{x} \cdot \mathbf{y}} |\mathbf{y}\rangle \right) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle),
 \end{aligned}$$

where we use $\mathbf{x} \cdot \mathbf{y} = x_0 y_0 \oplus x_1 y_1 \oplus \dots \oplus x_{n-1} y_{n-1}$ to be the sum of the bitwise product

We now consider the probability of measuring $|\mathbf{0}\rangle$ for the top qubits (which would indicate a constant function). We take $|\mathbf{y}\rangle = |\mathbf{0}\rangle$, meaning we also have $\mathbf{x} \cdot \mathbf{y} = \mathbf{x} \cdot \mathbf{0} = 0$ for all \mathbf{x} . We depend entirely then on the value of $f(\mathbf{x})$. With this assumption we have the state

$$\frac{1}{4} \left(\sum_{\mathbf{x} \in \{0,1\}^2} (-1)^{f(\mathbf{x})} |\mathbf{0}\rangle \right) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle). \tag{2.60}$$

Where our function is constant ($f(\mathbf{x}) = 0$ for all \mathbf{x} or $f(\mathbf{x}) = 1$ for all \mathbf{x}), our probability of measuring the top qubits as $|\mathbf{0}\rangle$ becomes

$$\frac{1}{4} \left(\sum_{\mathbf{x} \in \{0,1\}^2} (\pm 1) |\mathbf{0}\rangle \right) = \frac{1}{4} \pm 4 |\mathbf{0}\rangle = \pm 1 |\mathbf{0}\rangle. \tag{2.61}$$

If our function is balanced, half of the $f(\mathbf{x})$ evaluations will cancel the remaining half (by definition of a balanced function), giving us the probability of measuring $|\mathbf{0}\rangle$ as

$$\frac{1}{4} \left(\sum_{\mathbf{x} \in \{0,1\}^2} (-1)^{f(\mathbf{x})} |\mathbf{0}\rangle \right) = \frac{1}{4} 0 |\mathbf{0}\rangle = 0 |\mathbf{0}\rangle. \tag{2.62}$$

As the probability of measuring $|\mathbf{0}\rangle$ is 0, we know we must measure some other value.

¹⁷We clarify our sum here by providing a concrete example for $n = 2$:

$$\sum_{\mathbf{x} \in \{0,1\}^2} |\mathbf{x}\rangle = |00\rangle + |01\rangle + |10\rangle + |11\rangle. \tag{2.59}$$

We have shown here that a constant function results in a $|0\rangle$ measured, and a balanced function results in some other value, as we originally stated.

Looking at the states of the algorithm for any n , we have our start state $|\psi_0\rangle = |\mathbf{0}, 1\rangle = |0\rangle^{\otimes n}|1\rangle$. Next, the H gates create a superposition of all 2^n possible input states:

$$\begin{aligned} |\psi_1\rangle &= (H^{\otimes n} \otimes H) |\mathbf{0}, 1\rangle \\ &= \frac{1}{\sqrt{2^n}} \left(\sum_{\mathbf{x} \in \{0,1\}^n} |\mathbf{x}\rangle \right) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle). \end{aligned} \quad (2.63)$$

After applying U_f the state becomes

$$\begin{aligned} |\psi_2\rangle &= U_f (H^{\otimes n} \otimes H) |\mathbf{0}, 1\rangle \\ &= \frac{1}{\sqrt{2^n}} \left(\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f(\mathbf{x})} |\mathbf{x}\rangle \right) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle). \end{aligned} \quad (2.64)$$

The final step is to apply the $H^{\otimes n}$ gate to the top qubits, giving us our final state before measurement:

$$\begin{aligned} |\psi_3\rangle &= (H^{\otimes n} \otimes I) U_f (H^{\otimes n} \otimes H) |\mathbf{0}, 1\rangle \\ &= \frac{1}{2^n} \left(\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f(\mathbf{x})} \sum_{\mathbf{y} \in \{0,1\}^n} (-1)^{\mathbf{x} \cdot \mathbf{y}} |\mathbf{y}\rangle \right) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \\ &= \frac{1}{2^n} \left(\sum_{\mathbf{x} \in \{0,1\}^n} \sum_{\mathbf{y} \in \{0,1\}^n} (-1)^{f(\mathbf{x}) \oplus \mathbf{x} \cdot \mathbf{y}} |\mathbf{y}\rangle \right) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle). \end{aligned} \quad (2.65)$$

From this point we use the same reasoning as for our above run with $n = 2$. Considering the probability of measuring $|0\rangle$ for the top qubits, we take $|\mathbf{y}\rangle = |0\rangle$, giving $\mathbf{x} \cdot \mathbf{y} = \mathbf{x} \cdot \mathbf{0} = 0$ for all \mathbf{x} . With this assumption we have the state

$$\frac{1}{2^n} \left(\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f(\mathbf{x})} |\mathbf{0}\rangle \right) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle). \quad (2.66)$$

From here, if our function is constant, our probability of measuring the top qubits as $|\mathbf{0}\rangle$ is

$$\frac{1}{2^n} \left(\sum_{\mathbf{x} \in \{0,1\}^n} (\pm 1) |\mathbf{0}\rangle \right) = \frac{1}{2^n} \pm 2^n |\mathbf{0}\rangle = \pm 1 |\mathbf{0}\rangle, \quad (2.67)$$

and if balanced, it is

$$\frac{1}{2^n} \left(\sum_{\mathbf{x} \in \{0,1\}^n} (-1)^{f(\mathbf{x})} |\mathbf{0}\rangle \right) = \frac{1}{2^n} 0 |\mathbf{0}\rangle = 0 |\mathbf{0}\rangle. \quad (2.68)$$

We see the same outcome for any n as we did for our specific example of $n = 2$. For constant functions we measure the top qubits as $|\mathbf{0}\rangle$, and for balanced functions we measure some other value.

2.6.5 Shor's Algorithm

Shor discovered an algorithm that was able to exploit a quantum computer to calculate factors of large¹⁸ numbers in polynomial time [54, 55]. Utilising conventional computers, such a problem is

¹⁸Shor's paper gave the example of factoring the RSA 129 prime number.

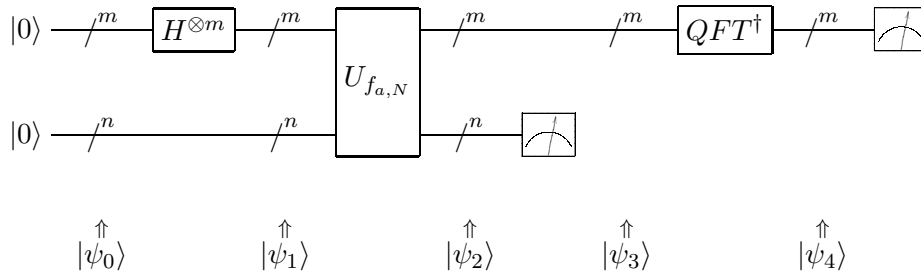


Figure 2.9: Quantum circuit implementing Shor's algorithm

extremely computationally intensive, so much so that elements of cryptography are based on the fact that it is difficult to factor integers on classical computers.

The algorithm itself is extremely complex to explain. Shor's algorithm takes a positive integer N as the input, and outputs a nontrivial factor of N if it exists. It can very briefly be summarised in these five steps [62]:

- Determine if N is prime or a power of a prime using a polynomial algorithm. If it is either, declare so and exit.
- Choose a random integer a , $1 < a < N$ and perform Euclid's algorithm to determine if $\gcd(a, N) = 1$. If it is not equal to 1 then choose a new a .
- Find a period p by using a quantum circuit (such as that defined in Figure 2.9).
- If p is odd or $a^p \equiv -1 \pmod{N}$ go to step 2 and choose a new a .
- Use Euclid's algorithm to calculate $\gcd\left(\left(a^{\frac{p}{2}} + 1\right), N\right)$ and $\gcd\left(\left(a^{\frac{p}{2}} - 1\right), N\right)$. Return at least one of the nontrivial solutions.

We obviously do not expect the reader to have a detailed understanding of Shor's work from this short excerpt; however it provides some insight into what is involved in the algorithm. We shall go on to explain each step in greater depth, along with our implementation of each stage in the next section.

2.6.6 Implementing Shor's Algorithm

As our brief introduction to the algorithm indicates, the first two and final two steps are performed on a classical computer using polynomial or better algorithms. The computationally expensive stage is the third step, and this is where we make use of quantum computation. Classically we can only perform this step in exponential time.

The first step is to evaluate whether our input integer N is a prime or a power of a prime. In our implementation we make the assumption that our input has been checked for these cases.

Our second step is to choose a random $a \in \mathbb{N}$ such that $1 < a < N$. We do this with the line `a = rand() % (N - 2) + 2;`. We must determine whether our a and N are coprime, meaning their only common positive factor is 1, or equivalently we say their greatest common divisor is 1. Our requirement is therefore $\gcd(a, N) = 1$. Any case where $\gcd(a, N) \neq 1$ means there is some factor of a and N other than 1, which means we have our factor of N .

We now move on to the quantum element of the algorithm, and construct the circuit

$$(M \otimes I) \left(QFT^\dagger \otimes I \right) (I \otimes M) U_{f_{a,N}} (H^{\otimes m} \otimes I) |0\rangle^{\otimes m} |0\rangle^{\otimes n} \quad (2.69)$$

described in Figure 2.9.¹⁹ With this circuit we will determine the period p of a certain function which we attempt to use to determine a factor of N (in steps 4 and 5). The function we aim to find the

¹⁹We should note that there are a number of different variations of Shor's algorithm in the literature. The version we show is the most common.

period of is

$$f_{a,N}(x) = a^x \text{ Mod } N. \tag{2.70}$$

In describing Shor’s algorithm we will use the standard example of $N = 15$. For this value of N , we show $\text{gcd}(a, N)$ for $1 < a < N$:

a	2	3	4	5	6	7	8	9	10	11	12	13	14
$\text{gcd}(a, 15)$	1	3	1	5	3	1	1	3	5	1	3	1	1

From our introductory text we know we can only use an a where $\text{gcd}(a, 15) = 1$, so immediately we discard those a that do not satisfy this. We show the results of the function $f_{a,N}(x)$ for the qualifying a where $x \geq 0$:

x	0	1	2	3	4	5	6	7	8	9	10	...	Period
$f_{2,15}(x)$	1	2	4	8	1	2	4	8	1	2	4	...	4
$f_{4,15}(x)$	1	4	1	4	1	4	1	4	1	4	1	...	2
$f_{7,15}(x)$	1	7	4	13	1	7	4	13	1	7	4	...	4
$f_{8,15}(x)$	1	8	4	2	1	8	4	2	1	8	4	...	4
$f_{11,15}(x)$	1	11	1	11	1	11	1	11	1	11	1	...	2
$f_{13,15}(x)$	1	13	4	7	1	13	4	7	1	13	4	...	4
$f_{14,15}(x)$	1	14	1	14	1	14	1	14	1	14	1	...	4

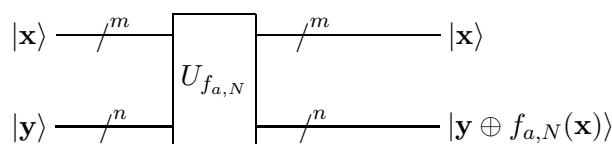
We find that when we work with larger values of a and N , it quickly becomes difficult to calculate $f_{a,N}$ due to numerical overflow. To overcome this we use re-arrangement to get

$$f_{a,N}(x) = \left(a \left(a^{x-1} \text{ Mod } N \right) \right) \text{ Mod } N. \tag{2.71}$$

From the table we see the cyclic behaviour of our functions; this behaviour is observed for all a that satisfy our coprime requirement. We define the period of a function as the minimum distance between two repeated numbers. In essence we do not particularly care about specific values of $f_{a,N}(x)$, but in fact we wish to find a p , the period, such that

$$f_{a,N}(p + x) = f(x). \tag{2.72}$$

Our first step in explaining the quantum circuit is to show our quantum implementation of the function $f_{a,N}$ as $U_{f_{a,N}}$. We require the input $|\mathbf{x}, \mathbf{y}\rangle$, where the bold face indicates the representation of multiple qubits, to be transformed into the output $|\mathbf{x}, \mathbf{y} \oplus f_{a,N}(\mathbf{x})\rangle$. Diagrammatically we have



We see that we have an input of size m qubits and output of size n . This implies we can break up $|\mathbf{x}\rangle$ into individual qubits as $|\mathbf{x}\rangle = |x_{m-1}, x_{m-2}, \dots, x_1, x_0\rangle$, and the same for $|\mathbf{y}\rangle$ with n . We know that due to the Mod N the maximum size of our output will be N . Representing this number in binary requires $n = \log_2 N$ bits. To be sure of determining the period of the function $f_{a,N}$ we must evaluate at least $0 \leq x \leq N^2$ function inputs. We find this requires $m = \log_2 N^2 = 2n$ bits. In total we require $3n = 3 \log_2 N$ qubits. One difference from previous algorithms is that here our values of $|\mathbf{x}\rangle$ and $|\mathbf{y}\rangle$ are now binary numbers such that $x = x_{m-1}2^{m-1} + x_{m-2}2^{m-2} + \dots + x_22^2 + x_12 + x_0$.

Evaluating our complete circuit, we begin with the state

$$\begin{aligned} |\psi_0\rangle &= |0\rangle^{\otimes m} |0\rangle^{\otimes n} \\ &= \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \end{bmatrix}^T, \end{aligned} \quad (2.73)$$

where this vector is of dimension mn . Applying H to the first m qubits gives us the superposition

$$\begin{aligned} |\psi_1\rangle &= (H^{\otimes m} \otimes I^{\otimes n}) |0\rangle^{\otimes m} |0\rangle^{\otimes n} \\ &= \frac{1}{\sqrt{2^m}} \sum_{\mathbf{x} \in \{0,1\}^m} |\mathbf{x}\rangle |0\rangle^{\otimes n}. \end{aligned} \quad (2.74)$$

We know our $U_{f_{a,N}}$ gate will evaluate the function and provide us the output in the bottom n qubits:

$$\begin{aligned} |\psi_2\rangle &= U_{f_{a,N}} (H^{\otimes m} \otimes I^{\otimes n}) |0\rangle^{\otimes m} |0\rangle^{\otimes n} \\ &= \frac{1}{\sqrt{2^m}} \sum_{\mathbf{x} \in \{0,1\}^m} |\mathbf{x}, f_{a,N}(\mathbf{x})\rangle \\ &= \frac{1}{\sqrt{2^m}} \sum_{\mathbf{x} \in \{0,1\}^m} |\mathbf{x}, a^{\mathbf{x}} \text{ Mod } N\rangle. \end{aligned} \quad (2.75)$$

We implement the $U_{f_{a,N}}$ gate using a series of n controlled unitary gates, each implementing $a^{x_j} \text{ Mod } N$ for each index j , $0 \leq j < m$ in the decomposition of x above.

At this point we are in a superposition of states with the bottom n qubits containing all 2^n evaluations of $f_{a,N}(\mathbf{x})$. Here we see the power of quantum computation; in one step we have evaluated all functions we require. For our valid values of a we recall that functions are periodic; our task is now to determine the period from our superposition.

Returning to our example of $N = 15$ we choose $a = 8$. Our value of $m = 2n = 8$, and so we have the state

$$\frac{1}{2^4} |0, 1\rangle + |1, 8\rangle + |2, 4\rangle + |3, 2\rangle + |4, 1\rangle + \dots + |254, 4\rangle + |255, 2\rangle. \quad (2.76)$$

When we measure the bottom qubits, we measure a single value from the function. In our case this will be either 1, 8, 4, or 2. In doing so, we enter the system into a superposition of all corresponding top qubit values with this measured value. If we were to measure 8, our superposition would include qubits 1, 5, 9 and 253. Our state would be

$$\frac{1}{2^6} |1, 8\rangle + |5, 8\rangle + |9, 8\rangle + \dots + |249, 8\rangle + |253, 8\rangle. \quad (2.77)$$

Where the fraction $\frac{4}{2^8} = \frac{1}{2^6}$ is the period, p , of our function, over 2^m . The difference between each entry in our superposition is now equal to the period of our function.

Our aim is to determine the period from the superposition. To do this we will use the quantum Fourier transform (QFT). We first provide some background. For a polynomial

$$P(x) = a_0 + a_1x^1 + a_2x^2 + a_3x^3 + \dots + a_{n-1}x^{n-1} \quad (2.78)$$

we are able to evaluate $P(x)$ for x_0, x_1, \dots, x_{n-1} by performing a matrix multiplication of the form

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} P(x_0) \\ P(x_1) \\ P(x_2) \\ \vdots \\ P(x_{n-1}) \end{bmatrix}, \quad (2.79)$$

where each row is a geometric series. We call this the Vandermonde matrix $\mathcal{V}(x_0, x_1, x_2, \dots, x_{n-1})$.

For our purposes we require the M th root of unity, $\omega^M = 1$, where $M = 2^m$. Our requirement is for a matrix of size M by M for the roots of unity $\omega_0 = 1, \omega_1, \omega_2, \dots, \omega^{M-1}$, so we use $\mathcal{V}(\omega_0, \omega_1, \omega_2, \dots, \omega^{M-1})$.

With this we define the discrete Fourier transform (DFT) as

$$DFT = \frac{1}{\sqrt{M}} \mathcal{V}(\omega_0, \omega_1, \omega_2, \dots, \omega^{M-1}), \quad (2.80)$$

giving

$$DFT_{j,k} = \frac{1}{\sqrt{M}} \omega^{jk}, \quad \text{and} \quad DFT_{k,j} = DFT_{j,k}^\dagger = \frac{1}{\sqrt{M}} \omega^{-kj}. \quad (2.81)$$

The DFT acts upon polynomials by evaluating them at different (equally spaced) points around a circle, and so the output will be periodic. If we were to use matrix multiplication, as we see above, we would output a periodic sequence.

If we were to reverse the DFT we would be able to convert a periodic sequence back into a polynomial. This is exactly what we do. Our DFT is unitary, meaning it is reversible. Applying DFT^\dagger to a vector (state) will modify the period from p to $\frac{2^m}{p}$ and eliminate any offset we have.

In our final circuit we require a quantum Fourier transform which is a quantum specialisation of the DFT we show, constructed entirely of H and $R(\alpha)$ gates. Upon measuring the final state we have

$$x = \frac{y2^m}{p} \quad (2.82)$$

for some integer y . We know 2^m to begin with and have now measured x , giving us $\frac{y}{p}$. From this we can determine p by reducing our fraction, leaving us with p as our denominator.

With this p we perform our final two classical steps to finish. One of the functions $\gcd\left(\left(a^{\frac{p}{2}} + 1\right), N\right)$ and $\gcd\left(\left(a^{\frac{p}{2}} - 1\right), N\right)$ will provide a non-trivial factor of N , and with this we have completed the algorithm.

We conclude by giving our implementation of Shor's algorithm, concentrating upon the quantum elements:

```

1  /* N: number to factorise
2  * LN: log_2(N) */
3  template<int N, int LN>
4  int algorithm::shor()
5  {
6      int a, factor = -1;
7      do
8      {
9          // Find random number 1 < a < N
10         // Check if a and N are coprime
11
12         const int inputQs = (1 << (2 * LN));
13         qubit input = qubit(CD(0, 0), inputQs);
14         // Apply H

```

```

15     gate<inputQs>::apply(&input, HGATE, &input);
16
17     const int outputQs = 1 << LN;
18     qubit output = qubit(CD(0, 0), outputQs);
19
20     qubit regs = qubit(CD(0, 0), inputQs * outputQs);
21     qubit *allQsGroup[] = { &input, &output };
22     // Apply U_f_a,N
23     gate<inputQs * outputQs>::apply(allQsGroup, UFNGATE, &allQs);
24
25     // Measure last LN qubits, equals x^a % N
26     int measureReg2 = 0, offset = 2 * LN;
27     for (int i = 0; i < LN; ++i)
28         measureReg2 += regs.measure(i + offset) * (1 << (LN-i-1));
29
30     // Apply QFT^dagger to top 2LN qubits
31     gate<inputQs * outputQs>::apply(&regs, QFTGATE, &regs);
32
33     // Measure top qubits
34     int measureReg1 = 0;
35     for (int i = 0; i < 2 * LN; ++i)
36         measureReg1 += regs.measure(i) * (1 << i);
37
38         // Perform classical steps
39     break;
40 }
41 while (true);
42 // Run final classical steps
43 return factor;
44 }
    
```

2.7 Developing a Quantum Circuit Calculus

In order to formally define quantum circuits, as presented in circuit diagrams, we develop a circuit calculus. With our definition we can translate forwards and backwards between diagrams and our calculus.

We first define the syntax of our calculus in Figure 2.10. Each circuit, \mathcal{C} , is made up of a series quantum operations, \mathcal{O} . Each operation specifies a list of input and output wires, $\mathcal{O}[\mathcal{W}_{input}, \mathcal{W}_{output}]$. The list of output wires from one operation must be the same as the input wires into the next operation, and the size of all wire lists must be identical to form a valid circuit. An operation can be a measurement in the standard basis, \mathcal{M} , a quantum gate, \mathcal{G} , or any combination via tensor product, \otimes , or multiplication, \times , of two or more operations. An individual gate can be any quantum gate, as long as it is specified as a unitary operator. Lines in circuit diagrams represent the path a qubit takes through a set of operations over time. For our purposes we call each uninterrupted section of this line a wire, and each is named uniquely.

To give an example of the calculus, we show the circuit used in Deutsch's algorithm (Figure 2.11). The first operation is formed by the tensor product of two h gates:

$$\mathcal{O}_1 := H \otimes H. \quad (2.83)$$

A circuit incorporating just this first operation would be written

$$\mathcal{C}_1 := (H \otimes H [\{w_1, w_2\}, \{w_3, w_4\}]). \quad (2.84)$$

The remaining operations in the circuit are

$$\mathcal{O}_2 := U_f, \quad \mathcal{O}_3 := H \otimes I, \quad \mathcal{O}_4 := M \otimes I. \quad (2.85)$$

$\mathcal{C} := (\mathcal{O}[\mathcal{W}_i, \mathcal{W}_j], \mathcal{O}[\mathcal{W}_j, \mathcal{W}_k], \dots),$	Circuits
$\mathcal{C}; \mathcal{C}$	Sequential concatenation of circuits
$\mathcal{G} := I, H, X, Z, CX, CZ, \dots$	Gates
$\mathcal{W} := \{w_i, w_j, \dots\},$	Wires
$\mathcal{W} \mathcal{W}$	Parallel combination of wires
$\mathcal{O} := \mathcal{M}, \mathcal{G},$	Operations
$\mathcal{O} \otimes \mathcal{O}, \mathcal{O} \times \mathcal{O}$	Combining operations

Figure 2.10: Circuit Calculus language syntax

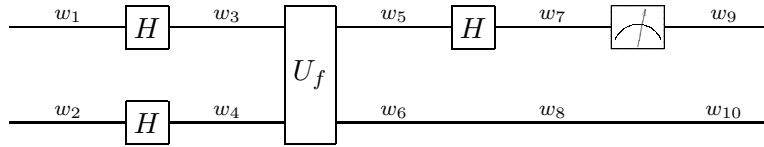


Figure 2.11: Quantum circuit implementing Deutsch's algorithm

As individual circuits, these operations make

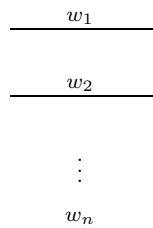
$$\begin{aligned}
 \mathcal{C}_2 &:= (U_f[\{w_3, w_4\}, \{w_5, w_6\}]) \\
 \mathcal{C}_3 &:= (H \otimes I[\{w_5, w_6\}, \{w_7, w_8\}]) \\
 \mathcal{C}_4 &:= (M \otimes I[\{w_7, w_8\}, \{w_9, w_{10}\}]).
 \end{aligned}
 \tag{2.86}$$

Combining all operations together to form the complete circuit we have

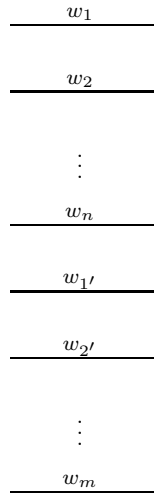
$$\begin{aligned}
 \mathcal{C}_{deutsch} &= \mathcal{C}_1 + \mathcal{C}_2 + \mathcal{C}_3 + \mathcal{C}_4 \\
 &= \left(H \otimes H[\{w_1, w_2\}, \{w_3, w_4\}], U_f[\{w_3, w_4\}, \{w_5, w_6\}], \right. \\
 &\quad \left. H \otimes I[\{w_5, w_6\}, \{w_7, w_8\}], M \otimes id[\{w_7, w_8\}, \{w_9, w_{10}\}] \right).
 \end{aligned}
 \tag{2.87}$$

We next show the semantics of the calculus in more detail.

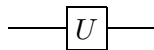
Wires We have a list of wires \mathcal{W} representing wires at one point in the circuit read from top to bottom. We show $\{w_1, w_2, \dots, w_n\}$ in the circuit model as



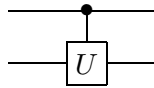
We can combine lists of wires \mathcal{W}_1 and \mathcal{W}_2 together in parallel via $\mathcal{W}_1 || \mathcal{W}_2$. In general we have $\{w_1, w_2, \dots, w_n\} || \{w_{1'}, w_{2'}, \dots, w_{m'}\} = \{w_{1'}, w_{2'}, \dots, w_n, w_{1'}, w_{2'}, \dots, w_{m'}\}$. Diagrammatically we show this combined list as



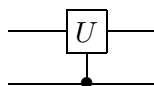
Gates We have two types of gate in a circuit. Those with control qubits and those without. The latter is a simple unitary operator $\mathcal{G} := U$ which applies to a single qubit:



A controlled gate applies to two or more qubits. A conventional controlled- U gate $\mathcal{G} := CU$ applies to two qubits, using the top qubit as the control and the bottom as the target:



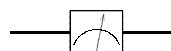
The reverse of this is to use the top qubit as the target and bottom as control. We would call this gate a swapped controlled- U , or UC :



We have left the set of gates open, as with the circuit model. As long as gates are defined as unitary operators they are allowable in the model and calculus.

We position a gate in a circuit with the abstraction of an operation described next.

Operations In the circuit model we can apply two operations. The first is a measurement in the standard basis $|0\rangle$ and $|1\rangle$. We have shown the symbol



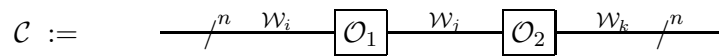
used to represent this. The second operation we can apply is a gate, as described above.

We combine operations in two ways, either in parallel (\otimes) or in series (\times). For parallel combination we take two operations \mathcal{O}_1 and \mathcal{O}_2 to form $\mathcal{O}_p := \mathcal{O}_1 \otimes \mathcal{O}_2$. In series we form $\mathcal{O}_s := \mathcal{O}_1 \mathcal{O}_2$. Diagrammatically we have



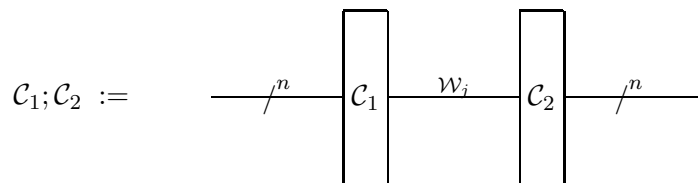
Circuits Our final and most complex construction is a circuit. A circuit is made up of a sequence of operations. Each operation has two associated lists of wires.

Given the example circuit $\mathcal{C} := (\mathcal{O}_1[\mathcal{W}_i, \mathcal{W}_j], \mathcal{O}_2[\mathcal{W}_j, \mathcal{W}_k])$, we have two operations \mathcal{O}_1 and \mathcal{O}_2 . For \mathcal{O}_1 we specify the input wires as the list \mathcal{W}_i , and \mathcal{W}_j as the output list. For \mathcal{O}_2 our input is \mathcal{W}_j and output is \mathcal{W}_k . Here we highlight an important point. In order for us to maintain a sequence of operations we require the output list of wires from one operation to be the input of the next. In our example the wire list \mathcal{W}_j is the output of \mathcal{O}_1 and the input of \mathcal{O}_2 . We also require that all lists of wires in a circuit are of the same length n . We show the circuit diagrammatically as



where we remember the size of the lists \mathcal{W} dictate the number of qubits n the wires here represent.

To sequentially concatenate two circuits we have a similar requirement. The output list of wires from the first circuit must be equal to the input list of wires of the second circuit. We have circuits \mathcal{C}_1 and \mathcal{C}_2 and concatenate them to form $\mathcal{C}_3 := \mathcal{C}_1; \mathcal{C}_2$. Partially defining $\mathcal{C}_1 := (\dots, \mathcal{O}[\mathcal{W}_i, \mathcal{W}_j])$ and $\mathcal{C}_2 := (\mathcal{O}[\mathcal{W}_j, \mathcal{W}_k], \dots)$ we see this requirement is satisfied by \mathcal{W}_j . We specify the general concatenation of two circuits diagrammatically as



With this calculus we are able to combine operations in parallel and in series, then combine these operations sequentially to form circuits. We place restrictions upon lists of wires as inputs and outputs of operations such that we have a correct flow of wires through the circuit.

2.8 Outcomes

To conclude this chapter we summarise what we have covered and achieved. We first describe some background of quantum computation, starting with a brief history of the field. After considering existing implementations of quantum computation using the circuit model, we specify what we aim to achieve and introduce the structure of our emulation.

From this point we begin to explain the theory of quantum computation, looking first at qubits, gates, then to developing circuits and implementing quantum algorithms. We detail this at a level understandable by a computer scientist and take care in describing concepts in detail but in simple terms. At each step we describe our implementation, supported by the preceding theory. What we ultimately produce is an explanation of all elements of quantum computing, along with a faithful, flexible and accurate emulation of it.

The final step of this chapter was the introduction of a quantum circuit calculus. We use this to specify and translate quantum circuit diagrams in order to formalise the diagram form.

Chapter 3

Measurement-Based Quantum Computation

3.1 Introduction

We will investigate an alternative model of quantum computation to that of quantum circuits and the use of unitary evolution presented so far [15]. In this chapter we consider measurement-based quantum computation (MBQC) [31, 50, 51, 43]. With this we make use of the entanglement and measurement of qubits in order to manipulate their states and thus perform computation. First suggested by physicists, it is a dramatically different approach to the circuit model, and since its inception has been of great interest to researchers in the field, both theoretically and practically [31]. Previously, measurements would be performed at the end of the computation in order to provide a classical output. With this model, measurements provide the foundation of operation through the appropriate selection of measurement bases.

To aid our steps towards implementing an emulation of a one-way quantum computer, we make use of a formal measurement calculus developed by Danos et. al [10, 11]. It is described as an assembly language for MBQC and provides a mathematical description of the model that we can work from. In addition, a standardisation algorithm is presented allowing both conversion between the one-way computer and circuit model, and the minimisation of auxiliary qubits required for computation.

With this background covered, we will present our implementation of an MBQC emulation. No such measurement-based emulation or simulation of any kind has been presented before. We also look towards implementing the standardisation algorithm presented in [10, 11], reducing the size of our representation to increase efficiency of the emulation, and otherwise improving our implementation using methods suggested in the literature.

3.2 Background

With the knowledge of circuit-based quantum computation we have covered, we will describe the specific approach we will be investigating, and highlighting how it differs from what we have seen so far.

Measurement-based quantum computation offers an alternative to the gate array (circuit) model we have described in the previous chapter. From the idea of MBQC came a number of alternative approaches, including teleportation [31], the measurement-based quantum Turing machine [43], and the one-way quantum computer [50, 51, 39]. In this text we are concerned with the later, and present the idea behind the cluster states used in the one-way computer invented by Raussendorf and Briegel [50, 51]. We concentrate on this approach primarily due to the evidence put forward by physicists which supports the feasibility of physically implementing such a computer [44, 10].

Although measurements destroy information of the quantum state, we can perform universal quantum computation using only measurements as computational steps [37]. In addition, we have shown measurements thus far as non-deterministic (probabilistic), and so we must justify how we are able to create deterministic behaviour using corrective adjustments. A secondary interest in MBQC

is that there is no classical equivalent; we are not simply creating a quantum generalisation analogous to an existing classical model. It may be that by looking beyond the classical models we understand well, and creating new architectures, we open up the true potential of quantum computation.

3.2.1 One-Way Quantum Computer

There are two very closely related measurement-based computational architectures. The first, which we will not cover here, is quantum computation via teleportation, based on the idea of teleporting information between quantum bits within a computer [31]. The second is the one-way quantum computer (1WQC) [50, 51, 45], also referred to as cluster state composition. We are interested in this as in the following section (see 3.2.2) we will present measurement calculus using the one-way computer as our approach.

Our description of the one-way computer begins with a two dimensional grid of finite size with a

$$|+\rangle = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) \quad (3.1)$$

state at each vertex. Where we provide input qubits at specified vertices, we use the relevant input state at that vertex, for example we use $|\psi\rangle$ at qubit 1 in Figure 3.1. From here we proceed to entangle qubits using controlled- Z gates¹,

$$CZ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} = ZC, \quad (3.2)$$

followed by measurements of entangled qubits in specific bases, in a required order. Where the required ordering can be grouped, measurements on multiple qubits can be performed in parallel. Dependencies of actions upon previous actions restrict the order in which those actions may be performed.

Any quantum gate array can be implemented as a pattern of single qubit measurements on a two dimensional cluster state. All measurements, M , are performed in one of two bases. The first is the standard basis $M_z = \{|0\rangle, |1\rangle\}$, and the second is the parametrised non-standard basis $M(\alpha) = \{|0\rangle \pm e^{i\alpha}|1\rangle\}$. The measurement $M(0)$ corresponds to measurement in the Pauli- X basis, and $M(\frac{\pi}{2})$ corresponds to the Pauli- Y basis. As we know, the quantum superposition collapses down to one of the eigenvalues of the basis in which it was measured. For M_z this is $|0\rangle$ and $|1\rangle$, and for non-standard measurement take use a measurement of $|+\rangle_\alpha$ as 0, and $|-\rangle_\alpha$ as 1.

As a general example we show the results of applying any unitary gate U to an input state $|\psi\rangle$ solely via measurements. Firstly, we can decompose any single qubit gate U into Euler angles α , β and γ ²:

$$U = R_x(\alpha) R_z(\beta) R_x(\gamma), \quad (3.3)$$

where

$$R_x(\theta) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & e^{i\theta} \\ e^{i\theta} & 1 \end{bmatrix}, \quad R_z(\theta) = \frac{1}{\sqrt{2}} \begin{bmatrix} e^{i\theta} & 1 \\ 1 & e^{-i\theta} \end{bmatrix}. \quad (3.4)$$

We begin with a line of qubits starting with $|\psi\rangle$ followed by four $|+\rangle$ states, as shown in Figure 3.1³. We entangle all neighbouring pairs with CZ , and then measure from left to right, excluding the final

¹The controlled- Z (sometimes referred to as the controlled-phase in the literature) has the property that $CZ = ZC$, where ZC is the swapped controlled- Z . We can show this from the generalisations of controlled and swapped controlled gates presented in 2.5. In the literature, CZ is also referred to as $\wedge Z$.

²We do not show a proof of this statement (see Jozsa [37]). Our selection of angles depends upon the unitary gate we wish to implement. Concrete examples are given in later sections.

³We represent the input qubit here as \star and output as \circ , and use \longrightarrow to represent entanglement with the neighbouring qubit. Remembering that $CZ = ZC$, we do not worry about the direction of the entanglement.

qubit	1	2	3	4	5
state	$ \psi\rangle$	$ +\rangle$	$ +\rangle$	$ +\rangle$	$ +\rangle$
entanglement	★ \rightarrow	● \rightarrow	● \rightarrow	● \rightarrow	○
measurement	M^X	$M(-\gamma(-1)^{s_1})$	$M(-\beta(-1)^{s_2})$	$M(-\alpha(-1)^{s_1+s_3})$	
measured value	s_1	s_2	s_3	s_4	

Figure 3.1: Applying a unitary gate via measurements [37]

qubit, in the bases shown.⁴ Qubit number 4 is then left in state $X^{s_2+s_4}Z^{s_1+s_3}U|\psi\rangle$. We can see this depends on the input qubit state $|\psi\rangle$, and we have achieved our unitary gate U with the addition of some known X and Y transformations (corrections).

With this we can see how straightforward it is to perform unitary operations simply by entangling qubits, and so forming a cluster state, followed by taking measurements. Where we wish to combine two gates sequentially, the output qubit(s), in this case qubit 4, feeds into the next measurement pattern (sequence) as the input qubit. All qubits from both patterns must be entangled before any measurements are applied, and dependencies must be maintained in the correct order between patterns.

This brief introduction has demonstrated the power of measurements, and shown how we create deterministic behaviour from a non-deterministic operation. The one-way quantum computer provides our architecture for MBQC, and underpins our discussion and implementation. We describe the model further in the following sections.

3.2.2 Measurement Calculus

Measurement calculus is a mathematical model describing the one-way quantum computer introduced in the previous section (3.2). In [10, 11] we are presented with a syntax and operational semantics for programs (termed patterns), along with a calculus for formally specifying and reasoning about these patterns. Through a standardisation step, it becomes possible to perform all qubit entanglements at the beginning of computation, revealing the dependencies of measurements (upon other measurements), followed by the measurements themselves, and finally by corrections.

This final point is extremely significant when we consider how this will benefit our emulation. With this we are able to determine what will occur before we even begin the computation. This provides us with much more information than with the circuit model, allowing us to create algorithms to manage resources, knowing exactly which qubits are entangled and what dependencies are involved.

Quantifying this statement, the existence of a standard form for any pattern means that patterns using no dependencies, or only the Pauli measurements, are only able to form a unitary belonging to the Clifford group. The significance of this is that it can be efficiently simulated by a classical computer.

We describe the syntax of the measurement calculus in Figure 3.2. We will go on to explain what each part represents in the following sections. In brief: a preparation action N_i puts a qubit i into the superposition state

$$|+\rangle = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle). \quad (3.5)$$

The entanglement E_{ij} entangles qubits i and j by applying a controlled- Z gate as

$$CZ (|i\rangle|j\rangle). \quad (3.6)$$

⁴Taking measurements in this order is extremely important as the measurement bases are adaptive, with each relying on the previous.

$S := 0, 1, s_i, S + S$	Signals
$A := N_i$	Preparations
E_{ij}	Entanglements
${}^t [M_i^\alpha]^s$	Measurements
X_i^s, Z_i^s	Corrections

Figure 3.2: 1-qubit based measurement language syntax

Corrections X_i^s and Z_i^s apply X and Z gates respectively when the signal $s = 1$. We take a measurement

$${}^t [M_i^\alpha]^s = M_i^{(-1)^s \alpha + t\pi} \quad (3.7)$$

in the basis

$$|+\alpha'\rangle = \frac{1}{\sqrt{2}} (|0\rangle + e^{i\alpha'} |1\rangle), \quad |-\alpha'\rangle = \frac{1}{\sqrt{2}} (|0\rangle - e^{i\alpha'} |1\rangle), \quad (3.8)$$

where $\alpha' = (-1)^s \alpha + t\pi$ and $\alpha \in [0, 2\pi]$. When we take a measurement of qubit i , we define the signal $s_i = 0$ if we measure $|+\alpha'\rangle$, and $s_i = 1$ if $|-\alpha'\rangle$ is measured.

A pattern is defined as three finite sets V, I, O , with a finite sequence of commands $A_n A_{n-1} \dots A_1$, read from right to left. Each command may depend upon preceding commands, and we must maintain the following constraints:

- (D0) no command depends on an outcome not yet measured;
- (D1) no command acts on a qubit already measured;
- (D2) no command acts on a qubit not yet prepared, unless it is an input qubit;
- (D3) a qubit i is measured if and only if i is not an output.

The set V is the pattern computation space. The set $I \subseteq V$ defines the input qubits in the pattern. $O \subseteq V$ gives us the output qubits. It may also be the case that qubits belong to both sets.

We have not attempted to cover all of the details presented by Danos et al. here. They have taken the different steps we discussed in the previous section and formalised a notation and language for measurement-based computation. They describe what they have created as a quantum *assembly language*. In the following sections we present the standardisation and minimisation of patterns based on the work presented in this paper.

Our aim is that we are able to exploit this work to effectively develop an implementation of the quantum programming language. As described, we hope that the standard form will benefit our aim to distribute the simulation.

3.2.3 Existing Simulations

Although we present a number of existing solutions for the simulation of the circuit model, there are currently no implementations of simulations of MBQC available in the literature. As such, we will go on to present the first emulation of measurement-based quantum computation.

3.3 Problem Specification & Overview of implementation

Our overall aim is to develop the theory required to construct a correct emulation of MBQC using the one-way quantum computer architecture. From this theory we will create an implementation of the emulation. It should function such that given a single pattern or series of patterns, we are able

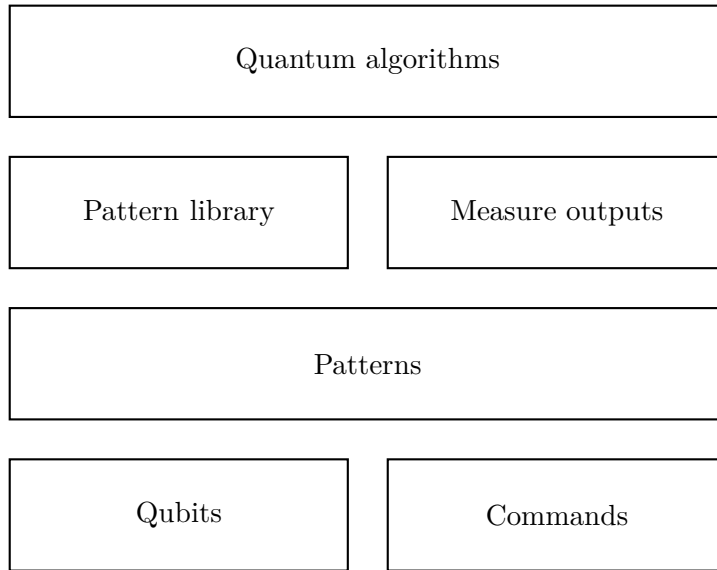


Figure 3.3: Structure of MBQC emulator

to combine them as necessary, process the actions, and provide correct output for the patterns given an input state or states. From this, we wish to provide an abstraction over patterns by allowing the input of a series of gates from a predefined library. Each gate will process the required pattern or combination of patterns. This abstraction also moves us towards an implicit translation from the circuit model to MBQC.

We define the structure of our emulation as in Figure 3.3. On top of this, we must define how to measure in non-standard bases and apply corrections (part of Commands), and then consider how to minimise and standardise patterns, as described in [10].

3.4 Measurements in Non-Standard Bases

Thus far we have only considered measuring qubits in the standard computational basis consisting of $|0\rangle$ and $|1\rangle$. In MBQC we manipulate states by taking measurements in different bases. Measurement calculus requires us to take orthogonal projections (measurements) over

$$|+\alpha\rangle = \frac{1}{\sqrt{2}} (|0\rangle + e^{i\alpha}|1\rangle), \quad |-\alpha\rangle = \frac{1}{\sqrt{2}} (|0\rangle - e^{i\alpha}|1\rangle), \quad (3.9)$$

so we must develop a notion for taking measurements in non-standard bases.

We will first describe von Neumann measurement, a type of projective measurement, in the standard basis [38, 57]. Formally describing an idea we are familiar with should build intuition, before generalising our notion to the new bases we require.

Our first step is to define our standard basis in terms of matrices we can use for measurement. We provide ket notation along with matrix representations:

$$M_0 = |0\rangle \cdot \langle 0| = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad M_1 = |1\rangle \cdot \langle 1| = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}. \quad (3.10)$$

The requirements placed upon these matrices as orthogonal projections are that:

- they are Hermitian, $M_j^\dagger = M_j$ (see Appendix A.8.9.1);
- that $M_j^2 = M_j$;

- and that the all matrices sum to the identity matrix, $\sum_j^m M_j = I$.

We define the probability of measuring state j as

$$P(j) = \langle \psi | M_j^\dagger M_j | \psi \rangle = \langle \psi | M_j | \psi \rangle. \quad (3.11)$$

Given state $|\psi\rangle = |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ we find the probabilities of measuring 0 or 1 in the standard basis as

$$P(0) = \langle + | M_0 | + \rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 0.5 \quad (3.12)$$

$$P(1) = \langle + | M_1 | + \rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 0.5. \quad (3.13)$$

This is consistent with our previous discussion of measurement, and our example is equivalent to applying an H gate to a start state of $|0\rangle$ and performing a measurement.

We next consider the state to which the qubit will collapse after measurement. With the standard basis we know this will always be $|0\rangle$ or $|1\rangle$ depending on which is measured. Formally, we define the collapsed state as

$$\frac{M_j |\psi\rangle}{\sqrt{\langle \psi | M_j^\dagger M_j | \psi \rangle}} = \frac{M_j |\psi\rangle}{\sqrt{\langle \psi | M_j | \psi \rangle}} = \frac{M_j |\psi\rangle}{\sqrt{P(j)}}. \quad (3.14)$$

For our example, we collapse to

$$\frac{M_0 |+\rangle}{\sqrt{P(0)}} = \sqrt{2} \left(\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle \quad (3.15)$$

or

$$\frac{M_1 |+\rangle}{\sqrt{P(1)}} = \sqrt{2} \left(\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle, \quad (3.16)$$

as we expected.

Our goal is to show the same for our new basis $|+\alpha\rangle$ and $|-\alpha\rangle$. We create our measurement matrices for $|+\rangle$ and $|-\rangle$,

$$M_+ = |+\rangle \cdot \langle +| = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad M_- = |-\rangle \cdot \langle -| = \frac{1}{2} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \quad (3.17)$$

which can now be used to determine the probabilities of measuring different states. Given $|\psi\rangle = |+\rangle$ as above,

$$P(+)=\langle + | M_+ | + \rangle = 1, \quad P(-)=\langle + | M_- | + \rangle = 0. \quad (3.18)$$

Then we generalise this to include the parameter α in $[0, 2\pi]$. Our measurement matrices are

$$M_{+\alpha} = |+\alpha\rangle \cdot \langle +\alpha| = \frac{1}{2} \begin{bmatrix} 1 & e^{i\alpha} \\ e^{i\alpha} & e^{2i\alpha} \end{bmatrix}, \quad M_{-\alpha} = |-\alpha\rangle \cdot \langle -\alpha| = \frac{1}{2} \begin{bmatrix} 1 & -e^{i\alpha} \\ -e^{i\alpha} & e^{2i\alpha} \end{bmatrix}. \quad (3.19)$$

We seek to confirm these satisfy the three requirements outlined above:

$$M_{+\alpha}^\dagger = \frac{1}{2} \begin{bmatrix} 1 & e^{-i\alpha} \\ e^{-i\alpha} & e^{-2i\alpha} \end{bmatrix} \neq M_{+\alpha}, \quad M_{-\alpha}^\dagger = \frac{1}{2} \begin{bmatrix} 1 & -e^{-i\alpha} \\ -e^{-i\alpha} & e^{-2i\alpha} \end{bmatrix} \neq M_{-\alpha}, \quad (3.20)$$

$$M_{+\alpha}^2 = \frac{1}{4} \begin{bmatrix} 1 & e^{2i\alpha} \\ e^{2i\alpha} & e^{4i\alpha} \end{bmatrix} \neq M_{+\alpha}, \quad M_{-\alpha}^2 = \frac{1}{4} \begin{bmatrix} 1 & e^{2i\alpha} \\ e^{2i\alpha} & e^{4i\alpha} \end{bmatrix} \neq M_{-\alpha}, \quad (3.21)$$

$$\sum_j M_j = \frac{1}{2} \begin{bmatrix} 1 & e^{i\alpha} \\ e^{i\alpha} & e^{2i\alpha} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} 1 & -e^{i\alpha} \\ -e^{i\alpha} & e^{2i\alpha} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & e^{2i\alpha} \end{bmatrix} \neq I. \quad (3.22)$$

We see that although the requirements are satisfied for some values of α , we have not satisfied any of the requirements for the general case. As such, we are unable to use von Neumann measurement for our implementation, and must choose a more general notion of measurement.

Relaxing the requirements, we use generalised measurement (also called positive operator valued measure) [46, 57]. Our sole requirement becomes

$$\sum_j^m M_j^\dagger M_j = I. \quad (3.23)$$

We show this holds for our $|+\alpha\rangle, |-\alpha\rangle$ measurement matrices:

$$\begin{aligned} \sum_j^m M_j^\dagger M_j &= \frac{1}{2} \begin{bmatrix} 1 & e^{-i\alpha} \\ e^{-i\alpha} & e^{-2i\alpha} \end{bmatrix} \frac{1}{2} \begin{bmatrix} 1 & e^{i\alpha} \\ e^{i\alpha} & e^{2i\alpha} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} 1 & -e^{-i\alpha} \\ -e^{-i\alpha} & e^{-2i\alpha} \end{bmatrix} \frac{1}{2} \begin{bmatrix} 1 & -e^{i\alpha} \\ -e^{i\alpha} & e^{2i\alpha} \end{bmatrix} \\ &= \frac{1}{2} \begin{bmatrix} 1 & e^{i\alpha} \\ e^{-i\alpha} & 1 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} 1 & -e^{-i\alpha} \\ -e^{-i\alpha} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I. \end{aligned} \quad (3.24)$$

With generalised measurement, probabilities are given by

$$P(j) = \langle \psi | M_j^\dagger M_j | \psi \rangle, \quad (3.25)$$

and the collapsed state by

$$\frac{M_j |\psi\rangle}{\sqrt{\langle \psi | M_j^\dagger M_j | \psi \rangle}} = \frac{M_j |\psi\rangle}{\sqrt{P(j)}}. \quad (3.26)$$

We see that these are identical to von Neumann measurement; however we do not reduce $M_j^\dagger M_j$ to M_j as this property is no longer a requirement of the measurement matrices.

With this description and reasoning behind our use of generalised measurement, we are ready to describe our implementation of it.

3.4.1 Implementing Measurements in Non-Standard Bases

As measurement plays such a key role in MBQC, we begin by demonstrating our implementation of this operation. As we presented for the circuit model, we stick closely to the theory we have described in order to implement an accurate emulation.

Our qubit implementation is largely consistent with our circuit representation:

```

1 typedef std::complex<double> CD;
2 class qubit {
3 private:
4     ublas::vector<CD> data;
5     bool isValid(void);
6     int apply(adjustment*);
7 public:
8     qubit(CD, CD);
9     qubit(ublas::vector<CD>*);
10     virtual ~qubit(void);
11     int measure(D, bool, int);
12     int apply(adjustmentType, int);
13 };
    
```

We are most interested here in the `measure()` function. We begin by defining our measurement matrices:

```

1  int qubit::measureQubit(D alpha, bool standardBasis, int qubit)
2  {
3      // Create  $M_j = |+\alpha\rangle$  and  $|-\alpha\rangle$ 
4      matrix plusMinus[2], measurementOps[2];
5      if (standardBasis)
6      {
7          plusMinus[0] = makeMat(2,2, (CD[]) { 1,0,0,0 }); //  $|0\rangle.\langle 0|$ 
8          plusMinus[1] = makeMat(2,2, (CD[]) { 0,0,0,1 }); //  $|1\rangle.\langle 1|$ 
9      }
10     else
11     {
12         plusMinus[0] = \*...\* { 0.5, 0.5*std::exp(I*alpha), //  $|+\rangle.\langle +|$ 
13             0.5 * std::exp(I*alpha), 0.5 * std::exp(I*2.0*alpha)};
14         // ...
15     }
16     //  $plusMinus = conj(M_j) * M_j$ 
17     for (unsigned int i = 0; i < 2; ++i)
18         measurementOps[i] = prec_prod(conj(plusMinus[i]), plusMinus[i]);

```

We now use the tensor product with identity gates to expand our matrices such that they measure the `qubit`th qubit of the combined state.

```

19     //  $P(j) = \langle \psi | I \otimes M_j \otimes I | \psi \rangle$  for  $j$  in  $\{+, -\}$ 
20     unsigned int n = iLog2(getSize()) - 1;
21     for (unsigned int i = 0; i < 2; ++i)
22     {
23         for (unsigned int b = qubit; b > 0; --b)
24             measurementOps[i] = kron(*I_MAT, measurementOps[i]);
25         for (unsigned int f = n - qubit; f > 0; --f)
26             measurementOps[i] = kron(measurementOps[i], *I_MAT);
27     }

```

We determine the probability of each measurement with the equation described above (Equation 3.25).

```

28     D rand = getRand(0, RAND_RANGE);
29     LD prob = -1.0;
30     unsigned int i;
31     for (i = 0; i < 2; ++i)
32     {
33         prob = std::abs(ublas::prec_inner_prod(ublas::prec_prod(
34             trans(conj(*getData())), measurementOps[i]), *getData()));
35         rand -= prob * RAND_RANGE;
36         // If random number reaches zero then choose this measurement
37         if (rand <= 0)
38             break;
39     }

```

Finally, we determine the collapsed state after measurement, defined in Equation 3.26:

```

40     // Modify our plus/minus to the right size
41     for (unsigned int b = qubit; b > 0; --b)
42         plusMinus[i] = kron(*I_MAT, plusMinus[i]);
43     for (unsigned int f = n - qubit; f > 0; --f)
44         plusMinus[i] = kron(plusMinus[i], *I_MAT);
45     // Calculate and copy collapsed state to qubit data
46     setData(ublas::prec_prod(plusMinus[i], *getData())/std::sqrt(prob));
47     return i;
48 }

```

qubit	1	2
state	$ \psi\rangle$	$ +\rangle$
entanglement	$\star \longrightarrow$	\circ
measurement	M^X	
measured value	s_1	
correction		X^{s_1}

 Figure 3.4: Measurement pattern \mathcal{H} equivalent to Hadamard Gate

3.5 Measurement Patterns

We describe MBQC programs as patterns, and we will use measurement calculus (see 3.2.2) to describe patterns we present in this section. We begin by demonstrating the pattern equivalent to the unitary transformation we know well, the Hadamard gate. We will then show a universal pattern able to implement any 2 by 2 unitary gate. This will require us to combine patterns together, so we also cover this topic. Tied in with these areas, we describe our implementation of these topics.

We remind readers of the Hadamard gate, a unitary transformation that places any pure state into a superposition of all pure states:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \quad (3.27)$$

Our aim is to demonstrate that we are able to implement this gate entirely via the measurements and corrections defined by the measurement calculus. This example pattern is taken from [10]. We require two qubits, with the first as the input and second as output:

$$V = \{1, 2\}, \quad I = \{1\}, \quad O = \{2\}. \quad (3.28)$$

The pattern actions are defined as

$$X_2^{s_1} M_1^0 E_{1,2} N_2. \quad (3.29)$$

We must recall two things: measurement calculus requires actions to be read from right to left, and usually we will omit preparation actions (N) from patterns, although it is included here for clarity. We recall that preparation actions are applied to all non-input qubits: N_j where $j \in V \setminus I$. We define the entire pattern as

$$\mathcal{H} := \left(\{1, 2\}, \{1\}, \{2\}, X_2^{s_1} M_1^0 E_{1,2} N_2 \right). \quad (3.30)$$

The pattern is shown diagrammatically in Figure 3.4. We will now show how this pattern is equivalent to the application of the Hadamard gate, as used in the circuit model.

In both the circuit model and measurement-based cases we begin with the input qubit $|\psi\rangle = a|0\rangle + b|1\rangle$ (we recall that any qubit can be described in this form, see 2.4). Our second qubit is prepared (N_2) in the state $|+\rangle$. Our entanglement action ($E_{1,2}$) combines our two qubits together to form the state

$$|\psi\rangle|+\rangle = (a|0\rangle + b|1\rangle)|+\rangle \xrightarrow{E_{1,2}} \frac{1}{\sqrt{2}} (a|00\rangle + a|01\rangle + b|10\rangle - b|11\rangle). \quad (3.31)$$

In matrices we have the same:

$$CZ|\psi\rangle|+\rangle = CZ \left(\begin{bmatrix} a \\ b \end{bmatrix} \otimes \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right)$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} a \\ a \\ b \\ b \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} a \\ a \\ b \\ -b \end{bmatrix}. \quad (3.32)$$

It is important that when describing qubits as entangled we always perform a CZ operation to combine the qubits.

Our next action is to perform a measurement with the parameter $\alpha = 0$, meaning the measurement is in the X basis. We may also write $M^X = M^0$. Using the formulae defined in the previous section, the probabilities of measuring each basis are

$$\begin{aligned} P(0) &= \frac{1}{\sqrt{2}} \begin{bmatrix} a & a & b & -b \end{bmatrix} M_{+0}^\dagger M_{+0} \frac{1}{\sqrt{2}} \begin{bmatrix} a \\ a \\ b \\ -b \end{bmatrix}^T \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} a & a & b & -b \end{bmatrix} \left(\frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right)^2 \frac{1}{\sqrt{2}} \begin{bmatrix} a \\ a \\ b \\ -b \end{bmatrix}^T \\ &= \frac{1}{2} (|a|^2 + |b|^2) \end{aligned} \quad (3.33)$$

$$\begin{aligned} P(1) &= \frac{1}{\sqrt{2}} \begin{bmatrix} a & a & b & -b \end{bmatrix} M_{-0}^\dagger M_{-0} \frac{1}{\sqrt{2}} \begin{bmatrix} a \\ a \\ b \\ -b \end{bmatrix}^T \\ &= \frac{1}{\sqrt{2}} \begin{bmatrix} a & a & b & -b \end{bmatrix} \left(\frac{1}{2} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right)^2 \frac{1}{\sqrt{2}} \begin{bmatrix} a \\ a \\ b \\ -b \end{bmatrix}^T \\ &= \frac{1}{2} (|a|^2 + |b|^2), \end{aligned} \quad (3.34)$$

and we collapse to the states

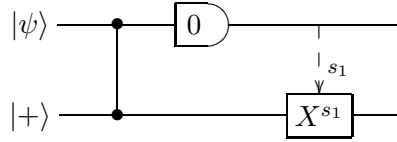
$$\begin{aligned} \frac{M_{+0}|\psi\rangle}{\sqrt{P(j)}} &= \frac{1}{\sqrt{2}} \left(\left(\frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \frac{1}{\sqrt{2}} \begin{bmatrix} a \\ a \\ b \\ -b \end{bmatrix}^T \right) \\ &= \frac{1}{4} \begin{bmatrix} a+b & a-b & a+b & a-b \end{bmatrix}^T \end{aligned} \quad (3.35)$$

$$\begin{aligned} \frac{M_{-0}|\psi\rangle}{\sqrt{P(j)}} &= \frac{1}{\sqrt{2}} \left(\left(\frac{1}{2} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \frac{1}{\sqrt{2}} \begin{bmatrix} a \\ a \\ b \\ -b \end{bmatrix}^T \right) \\ &= \frac{1}{4} \begin{bmatrix} a-b & a+b & -(a-b) & -(a+b) \end{bmatrix}^T. \end{aligned} \quad (3.36)$$

We notice here that we use $M_{+0} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. The identity matrix is tensored with our measurement matrix defined above as we are now working with two entangled qubits and require matrices of size 4 by 4. We use the identity matrix on the right side of the tensor product as we wish to measure the first qubit in the entangled two qubit state. If we wished to measure the second qubit we would take $M_{+0} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$. The requirement $\sum_j^m M_j^\dagger M_j = I$ is still maintained with these modified measurement matrices.

Using ket notation we have the state after measurement

$$\xrightarrow{M_1^0} \begin{cases} s_1 = 0, & \frac{1}{2} ((a+b)(|00\rangle + |10\rangle) + (a-b)(|01\rangle + |11\rangle)) \\ s_1 = 1, & \frac{1}{2} ((a-b)(|00\rangle - |10\rangle) + (a+b)(|01\rangle - |11\rangle)) \end{cases}$$


 Figure 3.5: Quantum circuit showing the \mathcal{H} pattern

$$\begin{aligned}
 &= \begin{cases} s_1 = 0, & |+\rangle \frac{1}{\sqrt{2}} ((a+b)|0\rangle + (a-b)|1\rangle) \\ s_1 = 1, & |-\rangle \frac{1}{\sqrt{2}} ((a-b)|0\rangle + (a+b)|1\rangle) \end{cases} \\
 &= \begin{cases} s_1 = 0, & |+\rangle \frac{1}{\sqrt{2}} \begin{bmatrix} a+b & a-b \end{bmatrix}^T \\ s_1 = 1, & |-\rangle \frac{1}{\sqrt{2}} \begin{bmatrix} a-b & a+b \end{bmatrix}^T. \end{cases} \quad (3.37)
 \end{aligned}$$

We see here a non-deterministic outcome; our measurements return different states. We have one final action to perform which will correct this:

$$\begin{aligned}
 &\xrightarrow{X_2^{s_1}} \begin{cases} s_1 = 0, & X_2^0 \left(|+\rangle \frac{1}{\sqrt{2}} ((a+b)|0\rangle + (a-b)|1\rangle) \right) \\ s_1 = 1, & X_2^1 \left(|-\rangle \frac{1}{\sqrt{2}} ((a-b)|0\rangle + (a+b)|1\rangle) \right) \end{cases} \\
 &= \begin{cases} s_1 = 0, & |+\rangle \frac{1}{\sqrt{2}} ((a+b)|0\rangle + (a-b)|1\rangle) \\ s_1 = 1, & |-\rangle \frac{1}{\sqrt{2}} ((a+b)|0\rangle + (a-b)|1\rangle). \end{cases}
 \end{aligned}$$

We leave qubit 2 in the same state for both branches, and have thus shown our pattern is deterministic.

For comparison, we show the steps of the \mathcal{H} pattern in a circuit diagram (Figure 3.5). We note that we must transform the standard basis measurement to measure in the basis $|+\alpha\rangle, |-\alpha\rangle$. To do so we use a 'D' symbol with the parameter α , which in this case is 0 (see 3.4).

From this, in comparison to the circuit model equivalent (a single H gate), we see that we require an auxiliary qubit and three operations, assuming measurement in the basis $|+\alpha\rangle, |-\alpha\rangle$ for any α is a single, basic operation. We must remember that in MBQC we restrict the operations we perform to only preparations, entanglement, measurement and corrections.

With this, we make implementation, either physical or emulation, vastly simpler. The circuit model requires us to perform measurement and unitary gates; however we require a great number of gates which can be applied at any point. This makes the physical implementation especially, far more difficult. If we restrict our available gates to two gates that are able to implement any unitary gate (a set of universal gates), for example, in the general case we increase the number of auxiliary qubits and number of actions greatly. In this case, MBQC looks much more favourable. We consider and quantify this conjecture and present a deeper comparison between models in the following chapter (see 4.4).

3.5.1 Universal Gate Pattern

In order to look towards implementing quantum algorithms, we require a greater number of patterns which are equivalent to the unitary gates we will use. At this stage we are not concerned with the minimisation of pattern size (the number of auxiliary qubits and actions required) as we are able to perform standardisation and minimisation steps.

We describe a universal gate pattern \mathcal{J} , able to implement a number of 2 by 2 unitary operators [11]. The matrix operation the pattern represents is described as

$$J(\alpha) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & e^{i\alpha} \\ 1 & -e^{i\alpha} \end{bmatrix}. \quad (3.38)$$

As we have focused on the H gate up to this point, we see that

$$J(0) = \begin{bmatrix} 1 & e^0 \\ 1 & -e^0 \end{bmatrix} = H. \quad (3.39)$$

Recalling that we defined the pattern $\mathcal{H} := (\{1, 2\}, \{1\}, \{2\}, X_2^{s_1} M_1^0 E_{1,2})$, we replace 0 with $-\alpha$, giving us

$$\mathcal{J}(\alpha) := (\{1, 2\}, \{1\}, \{2\}, X_2^{s_1} M_1^{-\alpha} E_{1,2}). \quad (3.40)$$

With the knowledge that the pattern \mathcal{J} is equivalent to the matrix J , we can see that the following unitary gates are equivalent to one or more uses of the \mathcal{J} pattern:

$$H = J(0) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad P(\alpha) = J(\alpha) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & e^{i\alpha} \\ 1 & -e^{i\alpha} \end{bmatrix}, \quad (3.41)$$

$$\begin{aligned} I &= H^2 = J(0)^2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \\ X &= P(\pi) H = J(\pi) J(0) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \\ Z &= H P(\pi) = J(0) J(\pi) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \end{aligned} \quad (3.42)$$

Beyond these examples, we can say that we are able to generate all unitary gates U over \mathbb{C}^2 . We are able to write any U as

$$U = e^{i\alpha} J(0) J(\beta) J(\gamma) J(\delta) \quad (3.43)$$

for some $\alpha, \beta, \gamma, \delta \in \mathbb{R}$ [11]. In the previous section (see Equation 3.3 and Equation 3.4) we show that any U may be written in terms of R_x and R_z . To show our new decomposition is also valid, we show J in terms of R :

$$\begin{aligned} R_x(\alpha) &= e^{-i\frac{\alpha}{2}} J(\alpha) J(0) \\ R_z(\alpha) &= e^{-i\frac{\alpha}{2}} J(0) J(\alpha). \end{aligned} \quad (3.44)$$

Using these equations it can be shown that both decompositions are equivalent.

We expand this notion to include controlled- U gates over $\mathbb{C}^2 \otimes \mathbb{C}^2$ by the decomposition

$$\begin{aligned} CU_{1,2} = J_1(0) J_1\left(\alpha + \frac{\beta + \gamma + \delta}{2}\right) J_2(0) J_2(\beta + \pi) J_2\left(\frac{-\gamma}{2}\right) J_2\left(\frac{-\pi}{2}\right) J_2(0) CZ_{1,2} \\ J_2\left(\frac{\pi}{2}\right) J_2\left(\frac{\gamma}{2}\right) J_2\left(\frac{-\pi - \delta - \beta}{2}\right) J_2(0) CZ_{1,2} J_2\left(\frac{-\beta + \delta - \pi}{2}\right), \end{aligned} \quad (3.45)$$

taken from and proved in [11].

From the decompositions of U and CU we are now capable of generating any gate we wish; we say that the set $\{J(\alpha), CZ\}$ generates all unitaries [11]. With this, we are now ready to demonstrate our implementation which emulates the behaviour we have introduced.

3.5.2 Implementing Measurement Patterns

The discussion of our implementation will include descriptions of our pattern representation, of each action (preparations, entanglements, measurements and corrections), and of our measurement calculus input.

Aiming to describe the pattern \mathcal{H} given in Equation 3.30 we give the code

```

1 pattern* patternLib::j(vector *input, D alpha)
2 {
3     int vArray[] = { 1, 2 };
4     patternInput inputArray[] = {
5         patternInput( 1, new qubit(input) )
6     };
7     int outputArray[] = { 2 };
8     patternAction actionArray[] = {
9         patternAction( PAE, 1, 2),
10        patternAction( PAM, 1, 0, -alpha ),
11        patternAction( PAX, 2, 1 )
12    };
13    return new pattern(
14        toVector(vArray),
15        toVector(inputArray),
16        toVector(outputArray),
17        toVector(actionArray));
18 }

```

as part of the `patternLib` class.

The `structs` used here and as part of the pattern implementation are

```

1 struct patternInput
2 {
3     int index;
4     qubit* qu;
5     int quI;
6 }
7
8 enum patternEntryType { EMID = 1, EIN = 2, EOUT = 4, EINOUT = 6 };
9 struct patternEntry
10 {
11     patternEntryType type;
12     qubit* qu;
13     int quI;
14 };
15
16 enum patternActionType { PAE = 1, PAX, PAZ, PAM, PAN, PAS };
17 struct patternAction
18 {
19     patternActionType type;
20     int indexI;
21     int indexJ;
22     D alpha;
23 };

```

As we see, we have almost identical syntax to that used for measurement calculus. Our `pattern` class implements each action of the calculus and stores all information required to define and process a pattern:

```

1 class pattern {
2 private:
3     int size;
4     bool processed;
5     std::map<int, int> measurements;
6     std::map<int, patternEntry> data;
7     std::vector<int> v;
8     std::vector<patternInput> input;

```

```

9         std::vector<int> output;
10        std::vector<patternAction> action;
11
12        int e(int, int);
13        int n(int);
14        int x(int, int);
15        int z(int, int);
16        int m(int, D, int, bool);
17    public:
18        pattern(std::vector<int>, std::vector<patternInput>,
19                std::vector<int>, std::vector<patternAction>);
20        pattern(pattern *p1, pattern *p2, bool);
21        virtual ~pattern();
22        int processActions(void);
23        std::vector<int> measureOutput(unsigned int, D, bool);
24    };
    
```

We describe measurement in the previous section; the `m()` action implements this and records the results of each measurements in `measurements` for use by dependent actions. Our `x()` and `y()` corrections are implemented as for the circuit model, as is `e()` with the modification that we retain references to the position of each qubit in the combined state.⁵ The action `n()` simply prepares a qubit with the state (data) $|+\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \end{bmatrix}^T$.

We call `processActions()` when we are ready to run a pattern. This will loop through a pattern's actions and perform each in order. With `measureOutput()` we are able to measure the resulting output qubits. Where we wish to minimise or standardise patterns we can do so in `processActions()` before we run actions.

3.6 Combining Patterns

In order to develop a series of patterns that together implement something more considerable, such as a quantum algorithm, we must be able to combine arbitrary patterns together. The formality of measurement calculus makes this a relatively straightforward task.

We define two patterns,

$$\begin{aligned} \mathcal{P}_1 &:= (V_1, I_1, O_1, C_1) \\ \mathcal{P}_2 &:= (V_2, I_2, O_2, C_2), \end{aligned}$$

which we combine as $\mathcal{P}_2\mathcal{P}_1$ to form

$$\mathcal{P}_3 := (V_1 \cup V_2, I_1, O_2, C_2C_1)$$

where we satisfy the condition $V_1 \cap V_2 = O_1 = I_2$. This requires that \mathcal{P}_1 has as many outputs as \mathcal{P}_2 has inputs, and renaming of qubits may be needed.

We give the example of the identity pattern described in terms of the universal gate pattern in Equation 3.42:

$$\begin{aligned} \mathcal{I} &:= \mathcal{H}\mathcal{H} \\ &= \left(\{1, 2\}, \{1\}, \{2\}, X_2^{s_1} M_1^0 E_{1,2} \right) \left(\{2, 3\}, \{2\}, \{3\}, X_3^{s_2} M_2^0 E_{2,3} \right) \\ &= \left(\{1, 2, 3\}, \{1\}, \{3\}, X_3^{s_2} M_2^0 E_{2,3} X_2^{s_1} M_1^0 E_{1,2} \right). \end{aligned} \quad (3.46)$$

⁵Where we have the actions $E_{1,2}$ followed by $E_{2,3}$ we retain the information of the qubit ordering (where they appear in the qubit state. This is described in the `patternEntry` struct as `quI`. In our example we have `quIs` of 0, 1 and 2 for qubits 1, 2 and 3 respectively, with the combined state $(I \otimes CZ)(CZ(|1\rangle \otimes |2\rangle)) \otimes |3\rangle$.

Here we have named our qubits such that no renaming is required for combination.

In reality, due to the properties of the identity gate, we are able to define the pattern simply as

$$\mathcal{I} := (\{1\}, \{1\}, \{1\}, \emptyset),$$

meaning no commands are performed. To help build some intuition about patterns, we show the equivalence of both patterns by performing the commands from the first definition of \mathcal{I} :

$$\begin{aligned}
 (a|0\rangle + b|1\rangle)|+\rangle|+\rangle &\xrightarrow{E_{1,2}} \frac{1}{\sqrt{2}}(a|00\rangle + a|01\rangle + b|10\rangle - b|11\rangle)|+\rangle \\
 &\xrightarrow{M_1^0} \begin{cases} s_1 = 0, & |+\rangle \frac{1}{\sqrt{2}}((a+b)|0\rangle + (a-b)|1\rangle)|+\rangle \\ s_1 = 1, & |-\rangle \frac{1}{\sqrt{2}}((a-b)|0\rangle + (a+b)|1\rangle)|+\rangle \end{cases} \\
 &\xrightarrow{X_2^{s_1}} \begin{cases} s_1 = 0, & |+\rangle \frac{1}{\sqrt{2}}((a+b)|0\rangle + (a-b)|1\rangle)|+\rangle \\ s_1 = 1, & |-\rangle \frac{1}{\sqrt{2}}((a+b)|0\rangle + (a-b)|1\rangle)|+\rangle \end{cases} \\
 &= |s_1\rangle \frac{1}{\sqrt{2}}((a+b)|0\rangle + (a-b)|1\rangle)|+\rangle \\
 &\xrightarrow{E_{2,3}} |s_1\rangle \frac{1}{2}((a+b)(|00\rangle + |01\rangle) + (a-b)(|10\rangle - |11\rangle)) \\
 &\xrightarrow{M_2^0} \begin{cases} s_2 = 0, & |s_1\rangle|+\rangle \frac{1}{\sqrt{2}}((2a)|0\rangle + (2b)|1\rangle) \\ s_2 = 1, & |s_1\rangle|-\rangle \frac{1}{\sqrt{2}}((2b)|0\rangle + (2a)|1\rangle) \end{cases} \\
 &\xrightarrow{X_3^{s_2}} \begin{cases} s_2 = 0, & |s_1\rangle|+\rangle \frac{1}{\sqrt{2}}((2a)|0\rangle + (2b)|1\rangle) \\ s_2 = 1, & |s_1\rangle|-\rangle \frac{1}{\sqrt{2}}((2a)|0\rangle + (2b)|1\rangle) \end{cases} \\
 &= |s_1\rangle|s_2\rangle(a|0\rangle + b|1\rangle). \tag{3.47}
 \end{aligned}$$

We see that qubit 3, our output, is left in the same state that qubit 1 began in. As per the properties of the identity gate, the state was left unchanged. We abuse notation slightly to reduce our branching, such that when branches are equal other than the value of a measured qubit, we replace this with $|s_i\rangle$. We no longer care about a qubit once measured, and we only show them here for clarity.

A second method of combination is to tensor two patterns, $\mathcal{P}_2 \otimes \mathcal{P}_1$, giving

$$\mathcal{P}_3 := (V_1 \cup V_2, I_1 \cup I_2, O_1 \cup O_2, C_2 C_1)$$

with the condition $V_1 \cap V_2 = \emptyset$. This condition can always be met through the renaming of qubits. The combined pattern \mathcal{P}_3 is equivalent to running \mathcal{P}_1 and \mathcal{P}_1 at the same time or separately in any order.

With this theory specified, we now look at how our emulation performs the two types of pattern combination.

3.6.1 Implementing the Combination of Patterns

We implement the combination of patterns by using a constructor in the `pattern` class. We first show the simpler case for the tensor product:

```

1 pattern::pattern(pattern *p1, pattern *p2, bool kron)
2   : processed(false), size(p1->getSize() + p2->getSize()),
3     v(p1->getV()), input(p1->getInput()), action(p1->getAction())
4 {
5     // Offset all of pattern two's indexes to avoid clashes
6     int offset = MAX(p1->getV());
    
```

```

7     std::vector<int> v2 = p2->getV();
8     std::vector<patternInput> input2 = p2->getInput();
9     std::vector<int> output2 = p2->getOutput();
10    std::vector<patternAction> action2 = p2->getAction();
11    // Add v1 and v2 together
12    for (unsigned int i = 0; i < v2.size(); ++i)
13        v.push_back(v2[i] + offset);
14
15    if (kron) // Tensor product of p1 and p2
16    {
17        // Offset and add inputs/outputs/actions from pattern two
18        for (unsigned int i = 0; i < input2.size(); ++i)
19        {
20            patternInput in = input2[i];
21            in.index += offset;
22            input.push_back(in);
23        }
24        output = p1->getOutput();
25        for (unsigned int i = 0; i < output2.size(); ++i)
26            output.push_back(output2[i] + offset);
27        for (unsigned int i = 0; i < action2.size(); ++i)
28        {
29            patternAction ac = action2[i];
30            ac.indexI += offset;
31            if (ac.indexJ > 0)
32                ac.indexJ += offset;
33            action.push_back(ac);
34        }
35    }

```

We see this implements $\mathcal{P}_3 := (V_1 \cup V_2, I_1 \cup I_2, O_1 \cup O_2, C_2 C_1)$ and ensures there are no crossovers between the indexes in \mathcal{P}_1 and those in \mathcal{P}_2 by offsetting all \mathcal{P}_2 indexes by the maximum index in \mathcal{P}_1 .

The standard combination is slightly more difficult as we must link the outputs of \mathcal{P}_1 to the inputs of \mathcal{P}_2 .

```

36    else
37    {
38        // Check that inputs/outputs match
39        // Take away the number of crossed over qubits
40        size -= input2.size();
41        // The new output qubits are those from p2
42        output = output2;
43        for (unsigned int i = 0; i < output.size(); ++i)
44            output[i] += offset;
45
46        // Update crossover output/input indexes
47        std::vector<patternInput> input2 = p2->getInput();
48        std::vector<int> output1 = p1->getOutput();
49        std::map<int, int> crossovers;
50        for (unsigned int i = 0; i < input2.size(); ++i)
51        {
52            crossovers[output1[i]] = input2[i].index + offset;
53            erase(v, output1[i]);
54        }
55
56        // Replace input1 entries referencing output1 using crossover
57        for (unsigned int i = 0; i < input.size(); ++i)
58            if (contains(crossovers, input[i].index))

```

```

59         input[i].index = crossovers[input[i].index];
60         // Replace action entries referencing input2 with output2
61         // ...
62         // Offset and add actions from pattern two
63         // ...
64     }
65     initialise();
66 }

```

We are now able to combine any two patterns in series or in parallel. We are ready to begin building up patterns able to implementing quantum algorithms.

3.6.2 Implementing Quantum Algorithms

Now that we are able to create patterns and combine them as we require, we can show how we implement quantum algorithms from this. We assume the reader is familiar with each algorithm after reading 2.6, and do not describe algorithmic details here.

Our implementation of Deutsch's algorithm is extremely concise as it acts as an abstraction over patterns, simply calling and combining pre-defined patterns as required:

```

1  int patternLib::deutsch(int opt)
2  {
3      vector input0 = ublas::makeVec(2, (CD[]) { 1, 0 });
4      vector input1 = ublas::makeVec(2, (CD[]) { 0, 1 });
5
6      pattern *p1_2 = new pattern(get(H_PAT, &input0),
7                                 get(H_PAT, &input1), true);
8      pattern *p1_3 = new pattern(p1_2, get(UF_PAT, opt), false);
9      pattern *p4_5 = new pattern(get(H_PAT), get(I_PAT), true);
10     pattern *p = new pattern(p1_3, p4_5, false);
11     p->processActions();
12     return patternLib::measureOutput(p, 0, -1, true);
13 }

```

We can see that our methodology for developing algorithms still revolves around the circuit model. We cannot escape this; it is extremely difficult to develop intuition for MBQC, so instead we effectively translate each part of our circuit implementation into patterns and combine them. We show the equivalent patterns for each gate and their combinations used in the Deutsch algorithm in Table 3.1. We use the concrete example of U_{f_2} .

The resulting pattern requires 7 qubits in total, in groups of 3 and 4. What we see is what would be the top qubit of a circuit diagram remains entirely separate to the bottom qubit. The top qubit of the circuit corresponds to the pattern qubits 1, 10 and 11, and the bottom to 3, 6, 8 and 12. Each group has one input and one output, and there are no entanglements between groups.

With this, we discover a problem with implementing Deutsch's algorithm in this way. Instead of having a black box gate that we do not know the function of (with an input and output of two qubits), we should instead have a black box pattern with the same inputs and outputs. In the case we have shown we see that the top and bottom circuit qubits separate entirely and there is no interaction at all, which is not the behaviour we desire.

Our implementation of Deutsch-Jozsa is similarly concise:

Gates / combinations	Equivalent pattern
H	$\mathcal{H} := (\{1, 2\}, \{1\}, \{2\}, X_2^{s_1} M_1^0 E_{1,2})$
$H \otimes H$	$(\{1, 2, 3, 4\}, \{1, 3\}, \{2, 4\}, X_4^{s_3} M_3^0 E_{3,4} X_2^{s_1} M_1^0 E_{1,2})$
I	$\mathcal{I} := (\{1\}, \{1\}, \{1\}, \emptyset)$
X	$\mathcal{X} := (\{1, 3, 4\}, \{1\}, \{4\}, X_4^{s_3} M_3^0 E_{3,4} X_3^{s_1} M_1^{-\pi} E_{1,3})$
$U_{f_2} = I \otimes X$	$\mathcal{U}_{f_2} := (\{1, 2, 4, 5\}, \{1, 2\}, \{1, 5\}, X_5^{s_4} M_4^0 E_{4,5} X_4^{s_2} M_2^{-\pi} E_{2,4})$
$U_{f_2}(H \otimes H)$	$(\{1, 3, 5, 6, 8, 9\}, \{1, 3\}, \{5, 9\}, X_9^{s_8} M_8^0 E_{8,9} X_8^{s_6} M_6^{-\pi} E_{6,8} X_6^{s_3} M_3^0 E_{3,6} X_5^{s_1} M_1^0 E_{1,5})$
$H \otimes I$	$(\{1, 2, 3\}, \{1, 3\}, \{2, 3\}, X_2^{s_1} M_1^0 E_{1,2})$
$(H \otimes I) U_{f_2}(H \otimes H)$	$(\{1, 3, 6, 8, 10, 11, 12\}, \{1, 3\}, \{11, 12\}, X_{11}^{s_{10}} M_{10}^0 E_{10,11} X_{12}^{s_8} M_8^0 E_{8,12} X_8^{s_6} M_6^{-\pi} E_{6,8} X_6^{s_3} M_3^0 E_{3,6} X_{10}^{s_1} M_1^0 E_{1,10})$

Table 3.1: Implementing Deutsch's algorithm using patterns

```

1  template<int N>
2  int patternLib::deutschJ(int opt)
3  {
4      vector *input0[N];
5      for (int i = 0; i < N; ++i)
6          input0[i] = new vector(ublas::makeVec(2, (CD[]) { 1, 0 }));
7      vector input1 = ublas::makeVec(2, (CD[]) { 0, 1 });
8
9      pattern *p1_2 = new pattern(get(H_PAT, -1, N, input0),
10                                get(H_PAT, (vector*[]) {&input1}), true);
11     pattern *p1_3 = new pattern(p1_2, get(UFN_PAT, opt, N), false);
12     pattern *p4_5 = new pattern(get(H_PAT, -1, N), get(I_PAT), true);
13     pattern *p = new pattern(p1_3, p4_5, false);
14     p->processActions();
15     return patternLib::measureOutput(p, 0, -1, true);
16 }
    
```

As we increase the value of n , the number of qubits requires increases massively. In our evaluation we quantify this statement.

Due to the complexity of the algorithm and the increased number of qubits required, we were unable to implement Shor's algorithm for our MBQC emulation. We discuss this limitation in the following evaluation chapter.

In order to increase our ability to implement more complex algorithms we look at two ideas. The first is translating patterns into a well defined standard form, and the second is to minimise the resource usage of patterns.

3.7 Standardising Patterns & Reducing Emulation Resources

The literature on measurement calculus presents a standardisation algorithm. In this section we investigate its implementation and benefits. The resulting standard form requires a pattern's commands to be in a certain order. Specifically, we begin with all the entanglements required for the pattern, this is followed by all measurements, and the final step is to perform corrections.

In this section we also look at minimising patterns and other ways of reducing the resources required to emulate patterns. When simulating a quantum system on a classical computer, the size of a quantum state space that can be represented is restricted by the amount of memory available on that computer [40]. As such, reducing memory usage increases the size of the state that can be represented on the same computer. We will look at different ways we can go about this.

3.7.1 Standardising Patterns

The formality of measurement calculus allows us to develop equivalences, and from these, rewrite rules. For example, the following combinations of corrections and measurements are equivalent, and can be used to generate equations from which we create rewrite rules:

$$\begin{aligned} X_i M_i^\alpha X_i &= M_i^\alpha X_i = M_i^{-\alpha} \\ Z_i M_i^\alpha Z_i &= M_i^\alpha Z_i = M_i^{\alpha+\pi}. \end{aligned} \quad (3.48)$$

In order to implement the standardisation of patterns we list the rewrite rules required to do so. We do not show the reasoning behind each rule as it is covered in depth in [11]. We first show the rules with which we can bring entanglements to the right (front) of the command list:

$$\begin{aligned} E_{i,j} X_i^s &\Rightarrow_{EX} X_i^s Z_j^s E_{i,j} \\ E_{i,j} X_j^s &\Rightarrow_{EX} X_j^s Z_i^s E_{i,j} \\ E_{i,j} Z_i^s &\Rightarrow_{EZ} Z_i^s E_{i,j} \\ E_{i,j} Z_j^s &\Rightarrow_{EZ} Z_j^s E_{i,j}. \end{aligned} \quad (3.49)$$

We see that corrections can be pushed left, behind measurements:

$$\begin{aligned} {}^t [M_i^\alpha]^s X_i^r &\Rightarrow_{MX} {}^t [M_i^\alpha]^{s+r} \\ {}^t [M_i^\alpha]^s Z_i^r &\Rightarrow_{MZ} {}^{t+r} [M_i^\alpha]^s. \end{aligned} \quad (3.50)$$

We also have some rules applicable where A_k is an action not applying to qubits i or j :

$$\begin{aligned} E_{i,j} A_k &\Rightarrow_{AE} A_k E_{i,j} \quad \text{where } A \text{ is not an entanglement} \\ A_k X_i^s &\Rightarrow_{AX} X_i^s A_k \quad \text{where } A \text{ is not a correction} \\ A_k Z_i^s &\Rightarrow_{AZ} Z_i^s A_k \quad \text{where } A \text{ is not a correction} \end{aligned}$$

We see with these rules that the sets V , I and O , the number of entanglements and measurements, and dependencies all remain as before standardisation. We say that all patterns \mathcal{P} can be rewritten to a unique standard pattern \mathcal{P}' via some number of rewrite steps: $\mathcal{P} \Rightarrow^* \mathcal{P}'$. The standardisation algorithm is then simply defined in three steps:

1. commute all preparation commands N to the right side;
2. commute all correction commands X and Z to the left size using the EX , EZ , MX and MZ rules;
3. commute all entanglement commands to the right size after the preparation commands.

As an example, we perform the algorithm on the \mathcal{X} pattern

$$\mathcal{X} := \left(\{1, 3, 4\}, \{1\}, \{4\}, X_4^{s_3} M_3^0 E_{3,4} X_3^{s_1} M_1^{-\pi} E_{1,3} \right). \quad (3.51)$$

$$\begin{aligned} & X_4^{s_3} M_3^0 E_{3,4} X_3^{s_1} M_1^{-\pi} E_{1,3} \\ \Rightarrow_{EX} & X_4^{s_3} M_3^0 X_3^{s_1} Z_4^{s_1} E_{3,4} M_1^{-\pi} E_{1,3} \\ \Rightarrow_{MX} & X_4^{s_3} M_3^{s_1} Z_4^{s_1} E_{3,4} M_1^{-\pi} E_{1,3} \\ \Rightarrow_{AZ} & X_4^{s_3} Z_4^{s_1} M_3^{s_1} E_{3,4} M_1^{-\pi} E_{1,3} \\ \Rightarrow_{AE} & X_4^{s_3} Z_4^{s_1} M_3^{s_1} M_1^{-\pi} E_{1,3,4}. \end{aligned} \quad (3.52)$$

Our final command sequence is in standard form. We are now able to replace our pattern equivalent of the X gate with this command sequence. We replace the sequence $E_{3,4} E_{1,3}$ with the combined version $E_{1,3,4}$ to ease reading.

The advantages this offers us is that we can perform all entanglements of a state at once, which will greatly improve the algorithmic complexity of on-the-fly entanglements during computation. Our state space will remain constant throughout computation, and thus requires less allocation of memory for combining state spaces.

We next look at ways of reducing our emulation resource requirements by minimising patterns and garbage collecting measured qubits. Having a standard form for this gives us a good base to begin our improvements from and a clear understanding of dependencies within a pattern. In respect to the garbage collection of qubits, knowing that our state space will not increase later in the computation is advantageous when developing our algorithm.

3.7.2 Reducing Qubit State Space

Perhaps more so than the circuit model, the quantum states of each qubit in MBQC are often very similar. Instead of always storing a dense vector for each state, we can indicate that the qubit is in one of a number of common states, such as $|+\rangle$. Only when it is in a less common state does it fall back to a dense vector representation when required. Matrices for measurements and operators would follow the same idea.

We can further follow this idea by restricting values of α for which the basis $|+\alpha\rangle, |-\alpha\rangle$ can take. This allows us to pre-compute measurement matrices, and reduces the common states a qubit will occupy.

Any idea or implementation following on from this would result in an approximation of MBQC. These improvements would only be made to increase simulation efficiency, and are not related to the true operation of quantum systems. Our aim was set out as an emulator, and so we do not take these ideas any further.

What we can do in this area is investigate reducing certain sequences of commands with equivalent but shorter sequences. These shorter command sequences would potentially also require fewer auxiliary qubits, and so reduce state space. Previously we have shown the equivalence between two measurement patterns implementing the I gate. A series of equivalences can be found and turned into rewrite rules in a similar way to the standardisation algorithm we presented in the previous sub-section. For our example we have

$$\left(\{1, 2, 3\}, \{1\}, \{3\}, X_3^{s_2} M_2^0 E_{2,3} X_2^{s_1} M_1^0 E_{1,2} \right) \Rightarrow \left(\{1\}, \{1\}, \{1\}, \emptyset \right).$$

With some investigation we can develop a number of similar and more general rules so that we are able to ensure all patterns or combined patterns can be reduced to their minimum form and so use the minimum number of qubits.

3.7.3 Garbage Collecting Qubits

Once a measurement has been performed and the result recorded, we are no longer interested in the measured qubit or its future state. We cannot perform commands on qubits once measured. One proposal is to remove these qubits from our state space once measured, meaning our emulation will become quicker as it has to work with smaller states. In a similar way, we can say that in a physical implementation we simply ignore qubits once measured, and so we can include this as part of an emulation.⁶

We describe our idea as follows. Assuming our pattern is in standard form, once we have performed all entanglements we are left with one or more entangled state spaces. Selecting the state space upon which the first measurement is performed, we begin with a state of dimension 2^n , where n is the number of entangled qubits. After measurement we record the result and then discard the qubit from the state space. This leaves us with a state of dimension 2^{n-1} . Following this reduction, all operations upon the state require exponentially less computation as the state is half the size it was previously, and so combined patterns acting on the state space must also be half the size.

We give a concrete example of our idea. We begin with the entangled state

$$CZ(CZ(|-\rangle \otimes |+\rangle) \otimes |+\rangle) = \frac{1}{2\sqrt{2}} \begin{bmatrix} 1 & 1 & 1 & -1 & -1 & -1 & 1 & -1 \end{bmatrix}^T. \quad (3.53)$$

We assume we take a measurement of the first qubit in the $|+_0\rangle, |-_0\rangle$ basis, and so we measure $|-_0\rangle$ and our state remains the same. We are now not interested in the value of the first qubit, and reduce the state space to be of dimension 4:

$$CZ(|+\rangle \otimes |+\rangle) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 1 & -1 \end{bmatrix}^T. \quad (3.54)$$

Implementation of this idea is more difficult than it may first seem as we must correctly remove the state of the measured qubit from the combined state space, bearing in mind the entanglements that have occurred.

3.8 Outcomes

We conclude by reminding readers of what we have covered and achieved. As with the circuit model, we have provided a detailed explanation and examples of measurement-based quantum computation, the one-way quantum computer architecture we use, and the measurement calculus, a formal notation. We introduce topics beginning with measurements in non-standard bases, and then develop a notion of patterns, as defined by the measurement calculus. Our next step is to combine patterns together to create patterns equivalent to quantum circuits. These circuits implement quantum algorithms, and so equivalence implies that our patterns do the same.

Our system successfully emulates the theory we have outlined, and we show implementations for the Deutsch and Deutsch-Jozsa algorithms. We describe the standardisation of patterns such that commands are in the order N, E, M, C , and show the various rewrite rules required to implement this. In our final section we also touch on some ideas for improvements to the amount of resources required to run our emulator.

⁶This is a very superficial view of the massive complexity of a physical implementation, but we aim to put our idea across in simple terms.

Chapter 4

Evaluation

4.1 Introduction

The aim of this chapter is to describe measures by which we may judge the success of the project objectives and provide further resulting discussion. Most obviously, our main criteria revolve around the implementation of our emulations. Our measures of achievement will be categorised into the circuit model (2), measurement-based quantum computation (3), and a comparison of the two approaches.

All benchmarks and results were obtained by running implementations on a 64 bit Intel Core 2 Duo 2.33Ghz running at 2.66Ghz with 2GB of DDR2 memory and an 80GB Intel solid state hard drive. The operating system environment is Cygwin under Windows 7 (64 bit).

4.2 The Quantum Circuit Model

Reaching our first aim involves implementing an emulator for the quantum circuit model. We must verify the correctness of the implementation, make a judgement upon its working accuracy, determine measures of efficiency, and test the bounds of the solution.

We successfully produced a working emulation that was able to run the Deutsch, Deutsch-Jozsa and Shor algorithms. Our most difficult aim was to run Shor on the number 15, requiring 12 qubits. Although this aim was achieved, when attempting to find factors of greater numbers we received an *out of memory* error. Running Shor on the number 17 (or any number up to 32) requires 15 qubits. To work with 12 qubits we require vectors and matrices of dimension 4,096. Using the `std::complex<double>` datatype, this number of entries alone takes 256MB for a single operator before we even consider the matrix construction and other facts. With 15 qubits we require their dimension to increase exponentially to 32,768, the reason for which is explained in [25]. This requires 16GB to represent a single operator.

The other aim relating to the circuit model was to develop a quantum circuit calculus. We evaluate our calculus in 4.2.6.

4.2.1 Demonstration of Correctness

Our first task is to determine whether our implementation functions correctly. We are unable to formally prove the correctness of our implementation as this is an extremely difficult task; however we will state the scenarios for which our implementation provides the correct result (to some degree of accuracy).¹

¹We discuss the accuracy of our implementation in 4.2.2.

Measuring a single qubit returns, probabilistically, the state entry with the greatest modulus squared value. Measurements were taken given the specified state. All qubit states correctly collapsed to the state measured.

Qubit state	Expected probability of measuring $ 1\rangle$	Actual probability of measuring $ 1\rangle$
$ 0\rangle = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$	0	0
$ 1\rangle = \begin{bmatrix} 0 & 1 \end{bmatrix}^T$	1	1
$ +\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \end{bmatrix}^T$	0.5	0.5
$ \psi\rangle = \frac{1}{2} \begin{bmatrix} \sqrt{3} & -i \end{bmatrix}^T$	0.25	0.25

We see that our emulation values are identical to the expected results. Measurement probabilities are implemented with the simple mathematical operation of the modulus of a complex number squared and we should see no errors here.

Applying a gate to a single qubit acts upon the state of the qubit as expected. Applying a gate equivalent to sequentially combined gates results in the same state as applying each gate individually, one after the other. Gates were applied to a qubit with the given state. For the combined gate $IXZH$, if each gate was applied individually, the qubit state for the intermediate steps are those stated in the rows above. We use the definition of $|\psi\rangle$ given in the table above.

Qubit state	Gate applied	Expected qubit state	Actual qubit state
$ \psi\rangle$	$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$ \psi\rangle$	$\begin{bmatrix} 0.8660 \\ -0.5i \end{bmatrix}$
$ \psi\rangle$	$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$\frac{1}{2} \begin{bmatrix} -i \\ \sqrt{3} \end{bmatrix}$	$\begin{bmatrix} -0.5i \\ 0.8660 \end{bmatrix}$
$\frac{1}{2} \begin{bmatrix} -i \\ \sqrt{3} \end{bmatrix}$	$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$\frac{1}{2} \begin{bmatrix} -i \\ -\sqrt{3} \end{bmatrix}$	$\begin{bmatrix} -0.5i \\ -0.8660 \end{bmatrix}$
$\frac{1}{2} \begin{bmatrix} -i \\ -\sqrt{3} \end{bmatrix}$	$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	$\frac{1}{2\sqrt{2}} \begin{bmatrix} -\sqrt{3} - i \\ \sqrt{3} - i \end{bmatrix}$	$\begin{bmatrix} -0.6124 - 0.3535i \\ 0.6124 - 0.3535i \end{bmatrix}$
$ \psi\rangle$	$IXZH = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}$	$\frac{1}{2\sqrt{2}} \begin{bmatrix} -\sqrt{3} - i \\ \sqrt{3} - i \end{bmatrix}$	$\begin{bmatrix} -0.6124 - 0.3535i \\ 0.6124 - 0.3535i \end{bmatrix}$

Our results show that our emulation correctly applies the given operators to qubit states, and that the combination of operators results in the same outcome as the application of successive individual operators.

The parallel combination of qubit states and gates follow the laws of the tensor product. When combining qubits or gates together, we ensure we have matrices of the correct dimension, and the ordering of combinations are correct.

Qubit/gate combination	Expected qubit state/gate	Actual qubit state/gate
$ 0\rangle \otimes 1\rangle$	$\begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^T$	$\begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^T$
$ +\rangle \otimes -\rangle$	$\frac{1}{2} \begin{bmatrix} 1 & -1 & 1 & -1 \end{bmatrix}^T$	$\begin{bmatrix} 0.5 & -0.5 & 0.5 & -0.5 \end{bmatrix}^T$
$ \psi\rangle \otimes \psi\rangle$	$\frac{1}{4} \begin{bmatrix} 3 & i\sqrt{3} & i\sqrt{3} & -1 \end{bmatrix}^T$	$\begin{bmatrix} 0.75 & 0.4330i & 0.4330i & -0.25 \end{bmatrix}^T$
$H \otimes I$	$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 0.7071 & 0 & 0.7071 & 0 \\ 0 & 0.7071 & 0 & 0.7071 \\ 0.7071 & 0 & -0.7071 & 0 \\ 0 & 0.7071 & 0 & -0.7071 \end{bmatrix}$
$X \otimes Y$	$\begin{bmatrix} 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \\ 0 & -i & 0 & 0 \\ i & 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 & 0 & -i \\ 0 & 0 & i & 0 \\ 0 & -i & 0 & 0 \\ i & 0 & 0 & 0 \end{bmatrix}$

We see that our resulting states and gates are of the correct size. We confirm the same for multiple qubits, but do not show the resulting matrices here due to their size. Using the tensor product we can successfully generate matrices up to dimension 2^{13} and vertices up to 2^{26} . Both constructions take over 1GB of memory.

We show the intermediate states for the Deutsch algorithm. To show the above features working together, we will show the intermediate states of Deutsch’s algorithm with f_3 , a constant function, implemented as normal by U_{f_3} . For reference, see Figure 2.7 for the circuit and positions of state labels. We discuss the general case and the case for f_3 in 2.6.2.

Algorithm step	Expected qubit state	Actual qubit state
$ \psi_0\rangle$	$\begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^T$	$\begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^T$
$ \psi_1\rangle$	$\frac{1}{2} \begin{bmatrix} 1 & -1 & 1 & -1 \end{bmatrix}^T$	$\begin{bmatrix} 0.5 & -0.5 & 0.5 & -0.5 \end{bmatrix}^T$
$ \psi_2\rangle$	$\frac{1}{2} \begin{bmatrix} 1 & -1 & -1 & 1 \end{bmatrix}^T$	$\begin{bmatrix} 0.5 & -0.5 & -0.5 & 0.5 \end{bmatrix}^T$
$ \psi_3\rangle$	$\frac{1}{\sqrt{2}} \begin{bmatrix} 0 & 0 & 1 & -1 \end{bmatrix}^T$	$\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T$
Measurement	1	1

We see one difference here in step $|\psi_3\rangle$. This is due to the use of an $H \otimes H$ gate in the

implementation instead of $H \otimes I$. We have this difference purely to reduce the complexity of operations in the emulation, and is an accepted variation. Both versions of the algorithm are equivalent but have a slightly different intermediate step.

We show the intermediate steps for the Deutsch-Jozsa algorithm. We now show a more complex version; the Deutsch-Jozsa for $n = 2$, meaning functions with the domain $f : \{0, 1\}^2 \mapsto \{0, 1\}$. We show a run for the constant function f_2 . For the circuit diagram (Figure 2.8) and for discussion of the algorithm and our implementation, see 2.6.4.

Algorithm step	Expected qubit state	Actual qubit state
$ \psi_0\rangle$	$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$	$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$
$ \psi_1\rangle$	$\frac{1}{2\sqrt{2}} \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}$	$\begin{bmatrix} 0.3535 \\ -0.3535 \\ 0.3535 \\ -0.3535 \\ 0.3535 \\ -0.3535 \\ 0.3535 \\ -0.3535 \end{bmatrix}$
$ \psi_2\rangle$	$\frac{1}{2\sqrt{2}} \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \\ -1 \\ 1 \\ -1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} -0.3535 \\ 0.3535 \\ -0.3535 \\ 0.3535 \\ -0.3535 \\ 0.3535 \\ -0.3535 \\ 0.3535 \end{bmatrix}$
$ \psi_3\rangle$	$\frac{1}{\sqrt{2}} \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$	$\begin{bmatrix} 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$
Measurement	0	0

We see that our first 3 states are the same, with a difference in step three. This is down to the same reason as described for Deutsch, and has no effect on the final measurement and result of the algorithm.

4.2.2 Accuracy

Now that we have determined with some certainty that our system is functionally correct, we next measure the accuracy of the implementation. We expect there to be two areas in which we will lose accuracy. The first is from rounding errors caused by limits in number representation which propagate during numerical operations. The second is from the pseudo-random number generator we use to implement probabilities. All classical computers suffer from this problem; as they are deterministic, it is impossible for them to generate a truly random number. No special attention has been placed upon achieving maximal accuracy. We expect some loss of accuracy, and in fact have implemented normalisation to mitigate rounding errors as far as possible. This discussion is to simply determine what accuracy we have achieved.

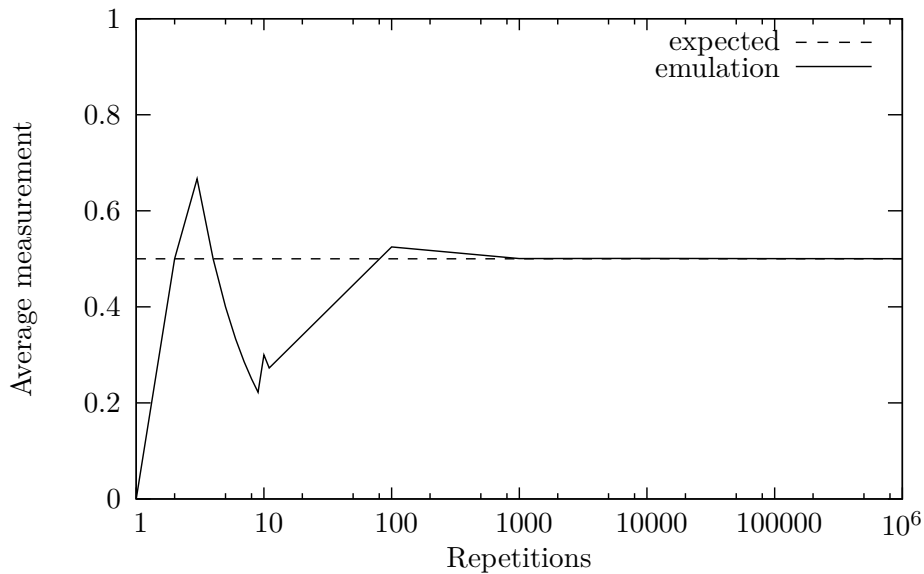


Figure 4.1: Number of repetitions against average measurement of the state $|+\rangle$

Determining the accuracy with which probabilistic behaviour is emulated. Following on from our experiments in the previous sub-section, we look at how faithfully our emulation implements probabilities. We will use a simple example to demonstrate this. Measuring the state $|+\rangle = |0\rangle + |1\rangle$ will result in a $|0\rangle$ or $|1\rangle$ with equal probability (0.5). We show our results for one run in the below table, and display this in Figure 4.1.

Repetitions	Average measurement	Repetitions	Average measurement
1	0	1,000	0.5005
5	0.4	100,000	0.5004
10	0.2222	1,000,000	0.5003
100	0.5248	10,000,000	0.4999

Results of each measurement can be either 0 or 1. Our expected probability of measuring either is 0.5, and so we expect after many repetitions we will converge to an average measurement of 0.5. We see that initially our results are erratic as there are too few results to produce a meaningful average. As the number of repetitions increases we convergence to 0.5 as expected. We see no significant bias in the measured probability, and so we can say the accuracy of our probabilities is acceptable.

Investigating the effects of different datatypes upon the accuracy of emulation versus the relative memory requirements. Altering the datatype we use to represent entries in quantum states and operators affects memory usage, run time, and accuracy. We begin by defining the size and properties of datatypes in our environment. The uBLAS libraries can only perform operations on matrices or vectors of the same type, so we cannot investigate the use of different datatypes for different operators or qubit states.

Datatype	Size (bytes)	Range	Representation error
<code>float</code>	4	$\pm 3.4 \times 10^{\pm 38}$	1.1×10^{-7}
<code>double</code>	8	$\pm 1.1 \times 10^{\pm 4932}$	2.2×10^{-16}
<code>long double</code>	12	$\pm 1.1 \times 10^{\pm 4932}$	1.0×10^{-19}
<code>complex<float></code>	8	$\pm 3.4 \times 10^{\pm 38} \pm 3.4i \times 10^{\pm 38}$	$(1 + i) 1.1 \times 10^{-7}$
<code>complex<double></code>	16	$\pm 1.7 \times 10^{\pm 308} \pm 1.7i \times 10^{\pm 308}$	$(1 + i) 2.2 \times 10^{-16}$
<code>complex<long double></code>	24	$\pm 1.1 \times 10^{\pm 4932} \pm 1.1i \times 10^{\pm 4932}$	$(1 + i) 1.0 \times 10^{-19}$
<code>ublas::matrix</code>	20	-	-
<code>ublas::vector</code>	12	-	-

Our investigation is to see the effects upon accuracy of emulation using each datatype. Our evaluation is also based upon the computation time and memory trade-offs of increased accuracy.² We will do this by developing a simple circuit which should repeatedly instigate rounding errors. Using each datatype we can see how far from the correct result we move.

The test circuit we create is a single qubit followed by a number of H gates in series. In the generation of the H gate we use a `long double` to represent the $\sqrt{2}$ to reduce the effect this has upon rounding error. We present the results in the following table (results are truncated for presentation purposes). We show state values for the first entry in the qubit state after an even number of H gates (the amount of $|0\rangle$ in the superposition) which should be equal to 1.0.

H gates	<code>complex<float></code>	<code>complex<double></code>	<code>complex<long double></code>
0	1.0	1.0	1.0
2	0.99999994039535522460...	0.99999999999999777955395074968...	0.99999999999999863336316158202...
10	0.9999988079071044921...	0.99999999999999222843882762390...	0.99999999999999316573160573762...
100	0.9999988079071044921...	0.999999999999992117416525161388...	0.999999999999993166273706823865...
10^3	0.9999988079071044921...	0.9999999999999921507232159001432...	0.9999999999999931663441799650771...
10^5	0.9999988079071044921...	0.999999999999992149501970573055587...	0.999999999999993166343258393230497...
10^7	0.9999988079071044921...	0.9999999999214953749770984359201...	0.9999999999316634324755120877270...
10^9	0.9999988079071044921...	0.9999999921495377086522182707994...	0.9999999931663432476596289899539...

²With more accuracy we require more memory to represent the increased precision. When performing operations upon the more accurate representations we have more numbers to perform the operation upon, and so computation time also increases.

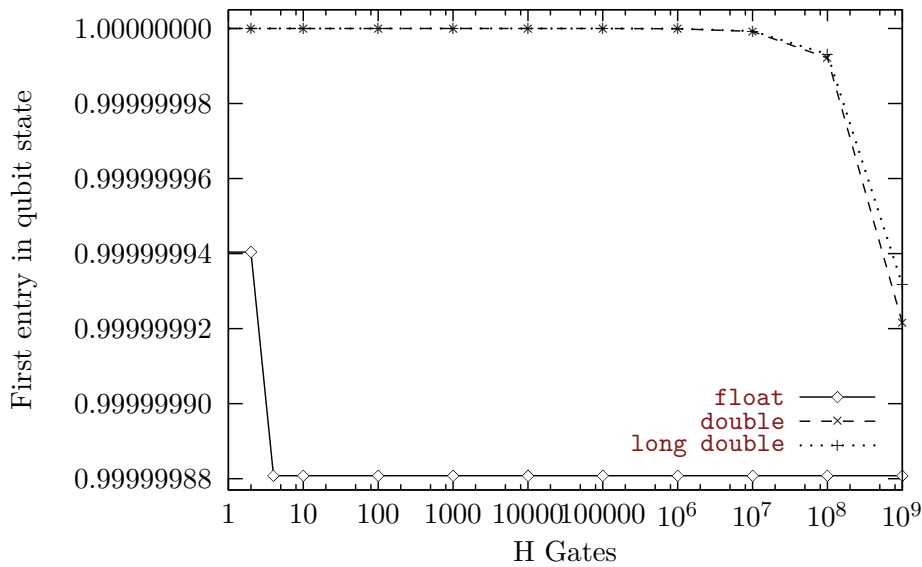


Figure 4.2: Number of H gates against first entry in qubit state

Plotting these results (Figure 4.2) gives us a simple visualisation justifying our decision to use the `double` datatype in our emulation. The maximum precision we have available using a primitive datatype is that of `long double` requiring 12 bytes (and 24 bytes for `complex<long double>`). For one third less space we are able to avoid rounding errors to nearly the same extent with `double`. We see after 10^9 gates the difference between values is only $1.0 \times 10^{-6}\%$ of the original value. For the space (memory) saving we make, this is an acceptable sacrifice.

We also see that after just four applications of the H gate our `float` representation has converged to a value with an error of $1.2 \times 10^{-5}\%$ of the original value. There is no change from this value when increasing the number of H gates. After four H gate applications, the `double` datatype has an error of just $3.3 \times 10^{-14}\%$.

In our emulation we would apply normalisation in the case when the sum of the modulus squared of entries of a state is greater or less than 1. Where this difference is above some defined threshold we consider the qubit state to be invalid. The threshold we use is 2×10^{-14} and has never been exceeded during emulation of algorithms or other use and testing. We deem this an adequate level of accuracy; we see that this threshold would be breached only after the application of 1000 H gates.

After each measurement we normalise our qubit state so we would have to apply hundreds of operators without measurement before we would see a problem. Generally, we only see rounding errors of this type where we multiply by a non-integer. Many gates are made up of integers so would not be counted in the same way. Other than contrived examples, we are unlikely to see significant errors introduced by rounding errors using the `double` datatype.

We must also note that any rounding error, significant or otherwise, would only affect the probabilities of $|0\rangle$ or $|1\rangle$ being measured. Any change in probabilities will only be observed after many repetitions of the circuit are completed. None of the algorithms presented would be affected by any rounding error introduced by any of the three datatypes.

4.2.3 Efficiency and Bounds

Efficiency was not the focus of our implementation; however we detail some investigation into how long different algorithms take to run, and the maximum number of qubits we can represent. We also take a brief look at the cause of this limitation.

Algorithm running times. The most obvious measure of our implementation is to see how long it takes to run each quantum algorithm. We repeat each algorithm 1,000,000 times on five occasions for the shorter algorithms, taking the average. For the longer running algorithms we reduce this to 100,000 or 10,000 repetitions. Where an algorithm can be given different inputs, we equally alternate between all possible (valid) inputs.

Algorithm	Average run time (ms)	Algorithm	Average run time (ms)
Deutsch / D-J, $n = 1$	0.03067	D-J, $n = 10$	6,921
Deutsch-Jozsa, $n = 2$	0.05853	D-J, $n = 11$	27,110
Deutsch-Jozsa, $n = 3$	0.1882	D-J, $n = 12$	100,100
Deutsch-Jozsa, $n = 4$	0.6167	Shor, $N = 4$	2.791
Deutsch-Jozsa, $n = 5$	2.149	Shor, $N = 6$	9.954
Deutsch-Jozsa, $n = 6$	10.68	Shor, $N = 8$	125.0
Deutsch-Jozsa, $n = 7$	41.80	Shor, $N = 9$	251.4
Deutsch-Jozsa, $n = 8$	164.5	Shor, $N = 12$	746.7
Deutsch-Jozsa, $n = 9$	674.0	Shor, $N = 15$	1,549

Due to the simplicity with which the Deutsch and Deutsch-Jozsa algorithms increase in qubit complexity, we can base most of our conclusions upon these. We see an exponential increase in run time as the number of qubits increases. We confirm this with a graph of the number of qubits against average run time shown in Figure 4.3. We see that for larger numbers of qubits the increase is slightly more than exponential, most likely due to the larger overheads associated with allocating increasing amounts of memory. These results match both the literature, the theory we have outlined, and our intuition.

When making comparisons between runs of Shor’s algorithm with different values of N we have a number of factors to consider:

- The number of qubits in each circuit is $3 \lceil \log_2(N) \rceil$, meaning we have an exponential increase in qubit state space as $\lceil \log_2(N) \rceil$ increases. For our values of N we seen an increase in $\lceil \log_2(N) \rceil$ for values of $N > 4$ then $N > 8$. Due to the state explosion problem, we are unable to represent enough qubits to emulate $N > 16$. We would require 15 qubits, which would need over 16GB of memory for each operator.
- As N increases we have a greater number of values of a to choose from as the range $1 < a < N$ increases so more repetitions may be required. The classical parts of the algorithm will also require more computation time.

With this in mind we show Figure 4.4, a graph of N against the average time taken for a single run. We see an exponential increase in run time as N increases, with an obvious step up from 6 to 8 as our state space increases. We also see a steady increase as the value of N grows. This is slightly higher than expected. We put this down to the increase in complexity of the algorithm as the number to be processed increases. Our assumption is that as N increases, the `gcd()` function becomes more

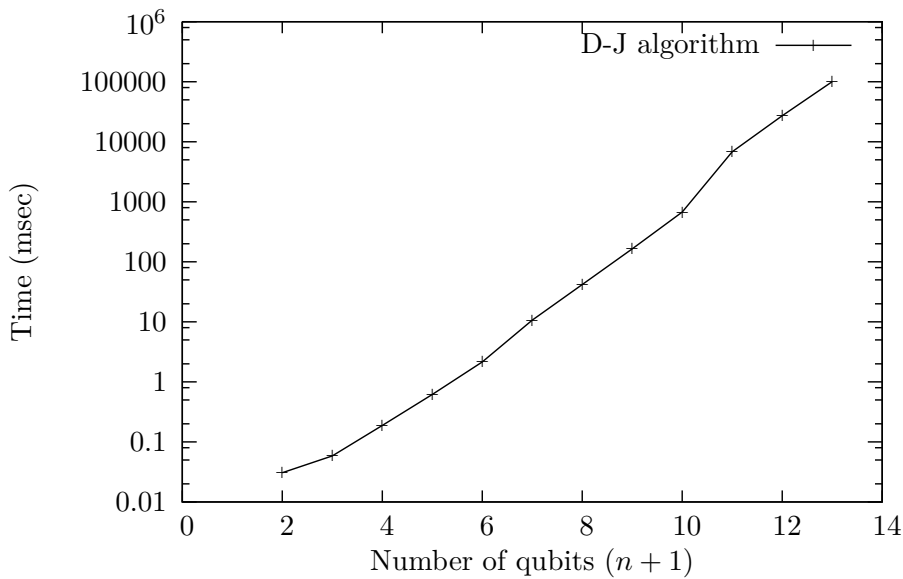


Figure 4.3: Number of qubits in the Deutsch-Jozsa algorithm against average run time

difficult to calculate and as factors become less dense (relative to the the range $1 < N$), we require more repetitions of the algorithm.

Investigate the number of qubits we can represent and perform operations upon. We look at the maximum number of qubits we are able to emulate in a circuit. We use the example circuits of Deutsch-Jozsa, Shor, and a test circuit. Our test involves the application of a single H gate to a number of combined qubits, followed by measurement of each qubit. The bounds we determine are shown in the following table:

Circuit	Maximum # qubits	Circuit	Maximum # qubits
Deutsch-Jozsa	13	H gates & measurement	13
Shor	12	H gates & measurement modified	14

We define these bounds as the maximum number of qubits that can be emulated in parallel in one circuit on the specified classical computer. When attempting to emulate more qubits we receive an error relating to the allocation of memory, implying we have utilised all memory available on the test machine. Monitoring resource usage confirms an exhaustion of available memory and supports this idea. In our introduction we mention that memory is the bounding factor of classical implementations of quantum systems, and we have confirmed this in our investigation. We see that we are only able to use 12 qubits for Shor's algorithm as the next number requires a further 3 qubits, so a total of 15.

To further investigate our idea, we develop a circuit to test the most simple case. We use the first step of the Deutsch-Jozsa algorithm (applying an H gate to all qubits). This gives us a maximum of 13 qubits, as with our other algorithms.

To push the bounds of our emulation, we concentrated specifically on increasing this limit. We were able to modify the memory handling of this simple example in two ways. We maintained minimum representations of states and operations at all points by freeing unneeded memory as soon as possible (using C++'s `delete`), and ordering our allocations and deletions such that we use the least amount

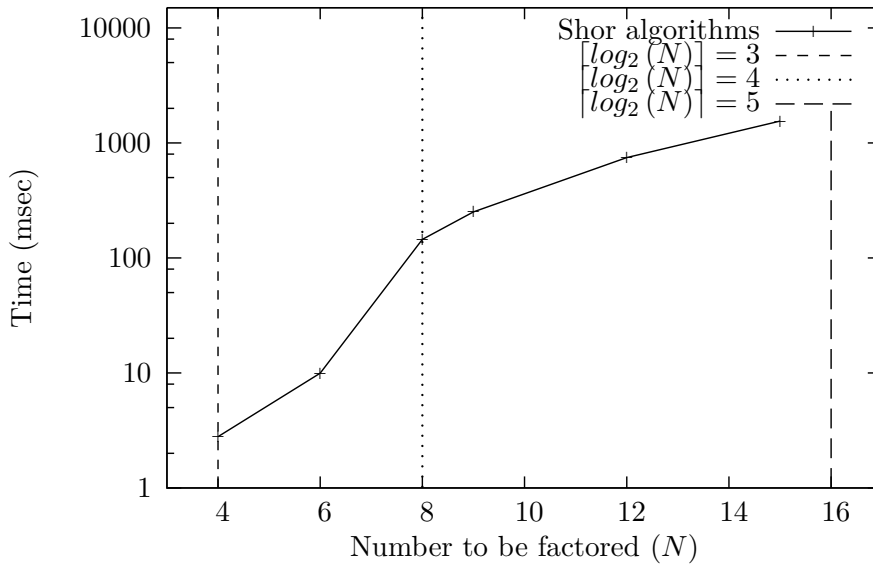


Figure 4.4: Number being factored in Shor’s algorithm against average run time

of memory at any one time. Where the tensor product was applied to qubits and gates, we ensured that the minimum amount of memory was used during the combination. For example where we had the operation $O_3 = O_1 \otimes O_2$, we expanded the memory used by O_1 and renamed this to O_3 instead of allocating entirely new memory.³ The combination of these ideas allowed us to represent an additional qubit, with a total of 14. Making these changes across the whole emulation would require additional thought and consideration but is most likely achievable.

4.2.4 Comparisons with Existing Solutions

Looking at the existing solutions we covered in our background we make comparisons with our own emulation. Our first consideration is that some implementations are too different from the emulator we have created to be fairly compared. We immediately discount QDD, due to its use of BDDs for state representation, and QCE, as it aims to emulate specific quantum hardware, not the theory of quantum computation.

We therefore consider jQuantum and QCAD, two quantum simulators. jQuantum is able to combine 15 qubits per quantum register. This is two more qubits than our general version, and one more than our modified version. We must remember that an increase in one qubit requires an exponential increase in state space. The author of QCAD claims it is able to run 20 qubits. During our testing this was not successful and caused the program to become unstable; however we assume that in some cases this is possible. The conclusion we must draw from this is that their representations of quantum state and operators must be vastly smaller, and so it is likely they are much less accurate. Our limited experiments were inconclusive when attempting to measure accuracy.

A second obvious advantage of these simulations over our implementation is the GUI they offer. This makes interaction with the software infinitely easier than what we have produced. Given more time, this would be the most obvious expansion to our emulators.

³There are a number of additional memory saving operation we did not implement as they fell outside of our aims. Firstly, memory could be allocated for one row (or column) of O_3 at a time as required, with an implementation of the tensor product compatible with this strategy. At the same time we may introduce swapping to disk such that we only require the necessary rows or columns of O_1 and O_2 to be in memory at any one time. Some implementations of the tensor product are open to such strategies. We have simply used the definition of the tensor product as our algorithm, as an increase in the representation of one qubit is not significant enough to warrant the additional effort for our goals.

The run times of algorithms in these simulations were comparable with what we achieved. In some cases they were slower, but this may be due to the overheads associated with a GUI.

In conclusion, our implementation is not massively behind these two simulations in terms of performing quantum computation, although an increase in our maximum state space size would be beneficial. We must remember that our emulator performs greater amounts of computation than these simulations, so greater resource requirements are reasonable.

4.2.5 Limitations

Despite generally meeting the aims set out for the emulation, it does have a number of weaknesses. On top of the limitations in accuracy, efficiency, and the other points discussed above we outline more general points here.

- We have not proved the correctness of our implementation. As mentioned, to even attempt to do so is extremely difficult, and there are limitless other cases we have not shown. Due to the imperfect nature of our classical representation, we will most likely never be able to prove its correctness, and only an approximate correctness. If we aimed to prove the correctness of an algorithm, for example, this would be a more achievable goal.
- We have shown that we are able to represent states accurately enough to correctly emulate the quantum algorithms we present in this report. Although we do not see any examples in the literature for which this is the case, for more complex circuits this level of accuracy may not be suitable; we may have hundreds of operations performed between measurements.
- The gates we make available for use in circuits are limited. We are able to approximately represent any quantum gate as we have several universal sets of gates amongst those available. We would be able to resolve this limitation by implementing a parametrised universal gate, such as $J(\alpha)$ presented in 3.5.1.
- We performed our evaluation using Cygwin on a Windows machine. Cygwin in particular is notoriously slow, and so our benchmarks are likely to be significantly slower than on an equivalent machine running Linux for example. We used this computer as it was the development machine and most testing had occurred on it.
- One of the biggest drawbacks of the emulator we have developed is that it requires good knowledge of the system to make good use of it. We have not provided a graphical user interface or implemented a formal input language (although we have developed a circuit calculus), and all circuits must be developed in code using the gate library and qubit functions.
- Our introduction clearly stated we are aiming to achieve a quantum emulation. With the inherent inaccuracies of the classical representation of quantum states, we will never achieve a perfect emulation in terms of accuracy. We are attempting to represent analogue systems using limited precision floating point numbers. We aim to imitate the function of a quantum system, which we have achieved, but for any input we can only provide an approximation (although a very accurate approximation) of the intermediate states. After measurement, our state resolves to $|0\rangle$ or $|1\rangle$, so these inaccuracies are no longer present. Stricter definitions of *emulation* than our own (see 1.2) could not be applied to our implementation.

4.2.6 Quantum Circuit Calculus

Our circuit calculus is able to represent any conventional circuit model diagram. We are able to translate our calculus into a diagram, and also a diagram into our calculus. Our presentation of the calculus is brief, and we only give a rough idea of its semantics; however it is suitable for our purpose.

We see two shortcomings of the calculus. We have no direct way of representing the difference between a wire of a circuit diagram representing multiple qubits ($/^n$) and the expanded form of this where each qubit is shown as a separate wire. Although equivalent, we are unable to distinguish

between the different forms using our calculus. Although we have not presented any examples of it in this text, some literature uses two parallel wires ($=$) to represent a classical bit value as opposed to a quantum value. For example, a qubit invariably takes a classical value after a measurement in the standard basis. Our calculus has no way of signifying a wire as containing a classical value.

A more significant factor to consider is that we have no way of specifying input or output qubits. We imply their presence through the use of wires; however, introducing qubits into the calculus would allow us to fully specify complete algorithms as opposed to circuits.

Further expansion of our discussion would include the demonstration of some common equivalences using the calculus (CX and CZ for example) and the development of a standard form and rewriting rules, as has been given for the measurement calculus in [11].

4.3 Measurement-Based Quantum Computation

Our second main aim was to develop an emulation of measurement-based quantum computation using the one-way quantum computer architecture and making use of the measurement calculus. We are interested in determining to what extent it is possible to efficiently emulate this on a classical computer, and what we are limited by. Much of the intention behind the evaluation of the quantum circuit model can equally be applied to our measurement-based implementation. As both emulations have a similar basis, we will refer to the evaluation of the circuit model for some aspects.

We will look at the correctness of the implementation, the accuracy our emulation achieves, and we investigate the bounds of our solution in terms of entangled qubits we can represent. We successfully emulated the Deutsch and Deutsch-Jozsa algorithms; however we were unable to implement Shor's algorithm due to its complexity. Without some form of minimisation we require more auxiliary qubits than with the circuit model, and so we were sooner restricted by memory than we would be with the circuit model. Where possible, we do apply equivalent patterns requiring fewer qubits; however we require more investigation into this area.

4.3.1 Demonstration of Correctness

We begin this section by showing the correctness of our implementation. For comparison, we will demonstrate the same scenarios as for the circuit model. The structure of implementation is based upon circuit model gates which we replace with equivalent patterns. As such, it is straightforward to implement circuits in terms of the combinations of patterns representing gates.

Measuring a single qubit returns, probabilistically, the value with the highest probability of being measured according to generalised measurement. We create patterns with which we are able to generate the listed qubit states from classical inputs, upon which we perform measurements in the specified basis. Our implementation adds the ability to make measurements in the standard basis (this is omitted from the measurement calculus) in order to provide direct comparison of our outputs with the circuit model.

The most fundamental part of MBQC is measurement, and we must be able to do this in the basis $|+\alpha\rangle, |-\alpha\rangle$ for any $\alpha \in [0, 2\pi]$. We show some examples in Table 4.2, concentrating on measurements with predictable outcomes for example purposes.

We see that each of the listed examples provides the same probabilities as we expect. We also confirmed that each set of generated measurement matrices conforms to the requirement set out in Equation 3.23 in 3.4. We also observe that our qubits correctly collapse to the measured state.

Applying a pattern representing a quantum gate produces an output qubit with the state expected. Applying a pattern equivalent to sequentially combined patterns results in the same state as applying each pattern individually, one after the other. Patterns were applied to a qubit with the given state. For the sequentially combined pattern \mathcal{IXZH} , if each pattern was applied individually,

Qubit state	Basis	Expected probability of measuring 1 ($ 1\rangle$ or $ -_{\alpha}\rangle$)	Actual probability of measuring 1
$ 0\rangle = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$	$ 0\rangle, 1\rangle$	0	0
$ 1\rangle = \begin{bmatrix} 0 & 1 \end{bmatrix}^T$	$ 0\rangle, 1\rangle$	1	1
$ +_0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \end{bmatrix}^T$	$ 0\rangle, 1\rangle$	0.5	0.5
$ +_0\rangle$	$ +_0\rangle, -_0\rangle$	0	0
$ +_{\pi}\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \end{bmatrix}^T$	$ +_0\rangle, -_0\rangle$	1	1
$ +_{\pi}\rangle$	$ +_{\pi}\rangle, -_{\pi}\rangle$	0	0
$ \psi\rangle = \frac{1}{2} \begin{bmatrix} \sqrt{3} & -i \end{bmatrix}^T$	$ 0\rangle, 1\rangle$	0.25	0.25
$ \psi\rangle$	$ +_0\rangle, -_0\rangle$	0.5	0.5
$ \psi\rangle$	$ +_{\pi}\rangle, -_{\pi}\rangle$	0.5	0.5
$ \psi\rangle$	$ +_{\frac{\pi}{2}}\rangle, -_{\frac{\pi}{2}}\rangle$	$\frac{1}{4}(2 - \sqrt{3})$	0.9330

Table 4.2: Measurement of single qubits

the qubit state for the intermediate steps are those stated in the rows above. We do not measure the output qubits in these examples. We use the definition of $|\psi\rangle$ given in the table above.

Our results (Table 4.3) show that we have successfully applied the patterns to our input qubits, resulting in the correct qubit state of the output qubit. We must note that the resultant state from the bottom two tests are of dimension 4, meaning they involve 2 qubits. We have separated the output (second) qubits in both cases by removing the first qubit, so we can clearly see the result we expect.

The parallel combination of qubits and patterns follow the laws of the tensor product. When combining qubit states or patterns together we ensure we have matrices of the correct dimension, and the ordering of combinations are correct.

Input qubit state	Pattern applied	Expected output state	Actual output state
$ \psi\rangle$	$I := (\{1\}, \{1\}, \{1\}, \emptyset)$	$ \psi\rangle$	$\begin{bmatrix} 0.8660 \\ -0.5i \end{bmatrix}$
$ \psi\rangle$	$\mathcal{X} := (\{1\}, \{1\}, \{1\}, X_1^1)$	$\frac{1}{2} \begin{bmatrix} -i \\ \sqrt{3} \end{bmatrix}$	$\begin{bmatrix} -0.5i \\ 0.8660 \end{bmatrix}$
$\frac{1}{2} \begin{bmatrix} -i \\ \sqrt{3} \end{bmatrix}$	$\mathcal{X} := (\{1\}, \{1\}, \{1\}, Z_1^1)$	$\frac{1}{2} \begin{bmatrix} -i \\ -\sqrt{3} \end{bmatrix}$	$\begin{bmatrix} -0.5i \\ -0.8660 \end{bmatrix}$
$\frac{1}{2} \begin{bmatrix} -i \\ -\sqrt{3} \end{bmatrix}$	$\mathcal{H} := (\{1, 2\}, \{1\}, \{2\}, X_2^{s_1} M_1^0 E_{1,2})$	$\frac{1}{2\sqrt{2}} \begin{bmatrix} -\sqrt{3} - i \\ \sqrt{3} - i \end{bmatrix}$	$\begin{bmatrix} -0.6124 - 0.3535i \\ 0.6124 - 0.3535i \end{bmatrix}$
$ \psi\rangle$	$\mathcal{IXZH} := (\{1, 2\}, \{1\}, \{2\}, X_2^{s_1} M_1^0 E_{1,2} Z_1^1 X_1^1)$	$\frac{1}{2\sqrt{2}} \begin{bmatrix} -\sqrt{3} - i \\ \sqrt{3} - i \end{bmatrix}$	$\begin{bmatrix} -0.6124 - 0.3535i \\ 0.6124 - 0.3535i \end{bmatrix}$

Table 4.3: Application of patterns

Qubit/gate combination	Expected qubit state/gate	Actual qubit state/gate
$ +\rangle \otimes -\rangle$	$\frac{1}{2} \begin{bmatrix} 1 & -1 & 1 & -1 \end{bmatrix}^T$	$\begin{bmatrix} 0.5 & -0.5 & 0.5 & -0.5 \end{bmatrix}^T$
$ \psi\rangle \otimes \psi\rangle$	$\frac{1}{4} \begin{bmatrix} 3 & i\sqrt{3} & i\sqrt{3} & -1 \end{bmatrix}^T$	$\begin{bmatrix} 0.75 & 0.4330i & 0.4330i & -0.25 \end{bmatrix}^T$
$\mathcal{H} \otimes \mathcal{I}$	$(\{1, 2, 3\}, \{1, 3\}, \{2, 3\}, X_2^{s_1} M_1^0 E_{1,2})$	
$\mathcal{CX} \otimes \mathcal{X}$	$(\{1, 2, 3, 4, 5\}, \{1, 5\}, \{1, 4, 5\}, X_5^1 Z_4^{s_3} Z_4^{s_2} Z_1^{s_2} M_3^0 M_2^0 E_{1,2,3,4})$	

We see that our resulting states and patterns are of the correct size and have the expected number of input and output qubits. We confirm the same for multiple qubits and for large patterns, but do not show the resulting matrices and commands here due to their size.

We look at the Deutsch algorithm. By implementing a quantum algorithm we can ensure that the system is functioning correctly. We have the additional work of moving it from a circuit model representation to the relevant measurement calculus representation, but we have shown this in the previous chapter to be

$$\left(\{1, 2, 3, 4, 5, 6, 7\}, \{1, 2\}, \{6, 7\}, X_6^{s_5} M_5^0 E_{5,6} X_7^{s_4} M_4^0 E_{4,7} X_4^{s_3} M_3^{-\pi} E_{3,4} X_3^{s_2} M_2^0 E_{2,3} X_5^{s_1} M_1^0 E_{1,5} \right) \quad (4.1)$$

for the function f_2 . For the circuit model we were able to show the intermediate qubit states between the application of each set of gates. For the measurement-based model, to show each step would take many pages and so is not given here. The computation of states branches at each measurement, resulting in 32 different cases by the end of this pattern.

We confirm that for all functions f we successfully determine whether it is constant or balanced by returning a 0 or 1 respectively.

We look at the Deutsch-Jozsa algorithm. We have similar trouble evaluating the intermediate states of the Deutsch-Jozsa algorithm. We allow the emulator to generate the combined patterns for $n = 2$, $n = 3$ and $n = 4$ and test that every value of f_x returns the correct result for the algorithm.

Our difficulty here stems from the lack of intuition we have for measurement-based computation. Calculations of intermediate qubit states are not simple operations, and so confirming our results are correct requires the generation of every possible branch for every measurement possibility. Such a task requires automation, so to test intermediate states we require another simulator for comparison or a test harness. Instead, we make the assumption that if for every possible input we measure our output qubits and they collapse to the correct state with a probability of 1 (therefore deterministically), we have implemented the algorithm and our emulator correctly.

4.3.2 Accuracy

With the belief that our emulation is correct, we now quantify the accuracy of our implementation. Working with measurements in the basis $|+\alpha\rangle$, $|-\alpha\rangle$ means we are likely to generate far more rounding errors than in the circuit model. This is due to two factors: we work with irrational numbers, and numbers requiring long decimal representations, and measurement in non-standard bases uses and generates long decimal representations.

We do not look at the accuracy with which probabilistic behaviour is emulated. We cover this in the previous section and use the same methods and technique to apply randomisation to probabilistic selection in both emulators.

Investigating the effects of different datatypes upon the accuracy of emulation versus the relative memory requirements. We cover the memory required for different datatypes in the previous section and refer to these figures where necessary. Our aim is to produce a similar investigation to determine if behaviour is the same with an equivalent measurement-based test.

Again, we have trouble evaluating our measurement-based model. If we aim to implement a series of \mathcal{H} patterns we have the problem that each requires an auxiliary qubit each, and so our state space grows with each application. Applying 11 \mathcal{H} patterns in series requires 12 entangled qubits (so a state space of dimension $2^{12} = 4096$) and takes a number of minutes to complete. What we see is at first slightly alarming.

We see no rounding error for the `float`, `double` or `long double` datatypes. The only inaccuracy we see is from the representation error that exists wherever an irrational or long number is stored as a finite floating point number.

After some consideration, we realise that after each measurement (which occurs every 3 operators in our \mathcal{H} pattern) we collapse to a definite state. For this example we collapse to the state $|+0\rangle$ or $|-0\rangle$. We calculate these collapse states from scratch every time we make a measurement. Any error present in the state space is eradicated when we take the measurement, and it only (minutely) affects the probability of measuring either state.

With this significant finding we can reduce our floating point representation to something smaller with no consequential loss of accuracy.

4.3.3 Efficiency and Bounds

Determining the efficiency and bounds of our emulation will be a key aspect in the comparison of our two emulations. We did not develop our implementation with the aim of maximising either efficiency (run time and resource usage), or the bounds (number of qubits), but we investigate what we have achieved.

By converting patterns to a standard form we are able to predict the pattern of entanglement and measurement, and so predict the required resources from the number of qubits, entanglements and measurements.

Algorithm running times. The most obvious measure of our implementation is to see how long it takes to run each quantum algorithm. We repeat each algorithm 1,000,000 times on five occasions for the shorter algorithms, taking the average. For the longer running algorithms we reduce this to 100,000 or 10,000 repetitions. Where an algorithm can be given different inputs, we equally alternate between all possible (valid) inputs.

Algorithm	Average run time (ms)	Algorithm	Average run time (ms)
Deutsch / D-J, $n = 1$	86.76	D-J, $n = 6$	683.4
Deutsch-Jozsa, $n = 2$	182.2	D-J, $n = 7$	857.0
Deutsch-Jozsa, $n = 3$	284.6	D-J, $n = 8$	1006
Deutsch-Jozsa, $n = 4$	404.3	D-J, $n = 9$	1253
Deutsch-Jozsa, $n = 5$	523.0	D-J, $n = 10$	1713

We expect the time taken for increasing values of n to steadily but exponentially increase. The increase will be much slower than for the circuit model. This is because our \mathcal{I} and \mathcal{X} patterns, which we use to create our U_f operators, do not increase the size of the quantum state space. These patterns are combined using the tensor product, meaning we effectively develop $n + 1$ parallel, independent circuits. As a result we have $n + 1$ different quantum registers, each with a similar number of qubits in each. This means our quantum state spaces remain relatively small in comparison to the overall number of qubits, and so we see our run times do not increase exponentially as they did with the circuit model.

We plot the graph of these results showing the number of qubits against average run time (Figure 4.5). Within the results we have a steady exponential increase, as we predicted, but we also see fluctuations; our line is not straight. We can put this down to the competing effects of increasing qubit state spaces and increasing algorithmic complexity for generating and combining patterns, acting against an increase in the number of qubit registers (groups of entangled qubits) we have.

Investigate the number of qubits we can represent and perform commands upon. We look at the maximum number of qubits we can entangle together. We are not particularly interested in the specific pattern with which we test the bounds, and so we develop a simple test pattern:

$$\mathcal{T} := (\{1, 2, \dots, n\}, \{1\}, \{n\}, E_{1,2,\dots,n}). \quad (4.2)$$

We are able to process this pattern for all values of n up to and including $n = 13$. When attempting to emulate $n = 14$ we receive an error relating to the allocation of memory, implying we have utilised all memory available on the test machine. Monitoring resource usage confirms an exhaustion of available memory and supports this idea. In our introduction we mention that memory is the bounding factor of classical implementations of quantum systems, and we have confirmed this in our investigation.

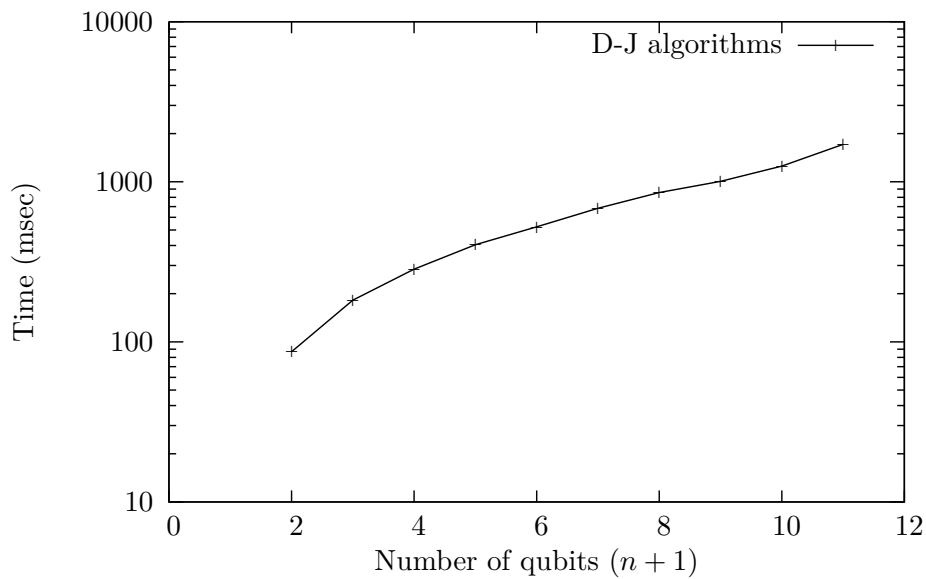


Figure 4.5: Number of qubits in the Deutsch-Jozsa algorithm against average run time

4.3.4 Limitations

Our emulation of MBQC was not as successful as that of the circuit model. This is due to a number of factors:

- There is far less literature available on the subject, and much of it is focused on the physical implementation aspect.
- No simulation of MBQC has been attempted before so everything we develop is new.
- The complexity of the algorithms required to perform four way entanglement, measurements in non-standard bases, and to manage dependent commands is much greater than any part of the circuit model. With the circuit model we have clear linear steps, whereas in MBQC we must consider many factors at any one time. To create the same kind of flexibility that the circuit model offers requires much more thought and development time. Due to limitations on time we were unable to develop our implementation as much as we hoped, and ultimately the complexity of Shor's algorithm meant we were unable to implement it other than at a very basic level.
- We do not exploit the full level of standardisation and minimisation discussed in our theory. We include a number of possible improvements in the following chapter.
- It is extremely difficult to develop an intuition for the measurement-based model, largely as it has no classical analogue. Developing an arbitrary pattern to solve a specific goal requires a significant amount of thought. As such, we are very much tied to the circuit model and base our implementation upon patterns which are equivalent to quantum gates. This results in patterns using more auxiliary qubits and commands than necessary, and means we must combine multiple patterns (representing gates) together to form circuits. If they were fully implemented, minimisation and standardisation would help ease this problem by ensuring we do not include unnecessary qubits or commands.
- We can implement any command sequence, and any pattern given that we have enough memory to represent the state spaces. What we lack is the ability to specify patterns in a simple language. Currently we use constructors to specify arbitrary patterns or call pre-defined patterns. A simple GUI would vastly improve the ease with which we can implement patterns.

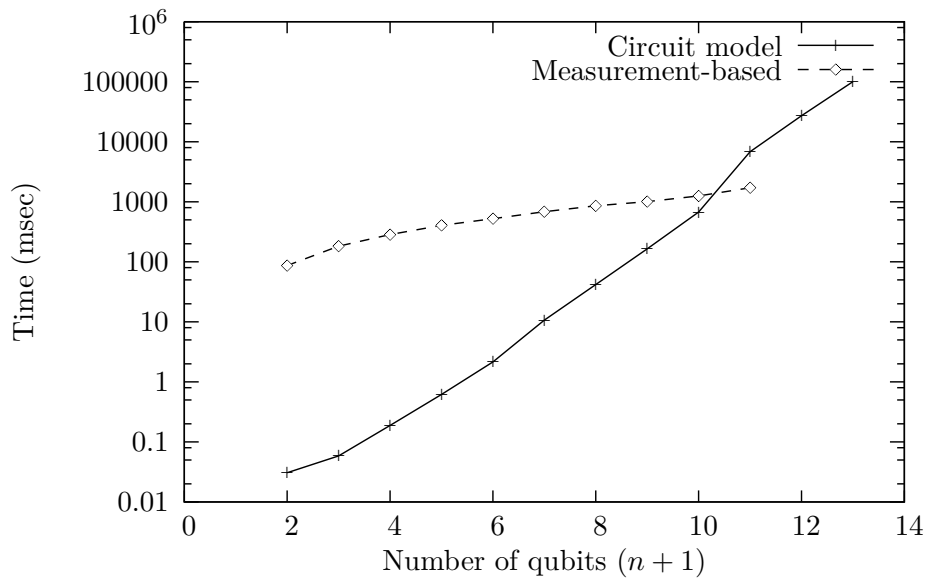


Figure 4.6: Number of qubits in the Deutsch-Jozsa algorithm against average run time

4.4 Comparisons Between Models

With an implementation and evaluation of both approaches, we make some comparisons between the two. These comparisons are made under very specific scenarios, and as such, it is likely to be difficult to draw any generalised conclusions with significant evidence. Even so, any comparison is useful for our purposes. Some additional discussion has been made during our description of each of the models.

4.4.1 Efficiency, Bounds and Accuracy

We use the time taken to run the Deutsch-Jozsa algorithm for different values of n as our benchmark of efficiency. To most easily make a comparison between models we plot results for both models (Figure 4.6).

From this we see that although the circuit model begins far in the lead, its timings increase past the run time of the MBQC between 10 and 11 qubits. This is a very interesting observation. It seems that the measurement model copes better with larger numbers of qubits, whereas the circuit model exhibits the state explosion problem. Our investigation led us to discover that our MBQC model was creating $n + 1$ parallel, independent circuits which reduces the maximum size of state spaces and improves the efficiency of computation.

It may be that we only see this behaviour from MBQC in this specific scenario as state spaces are kept separate, but this may also be down to an advantage of the approach. The next step to analysing this comparison further is to create other equivalent circuits with changeable numbers of combined qubits and repeat the experiment.

Our bounds, defined in terms of the number of combined qubits we can represent, are similar for both models. For the circuit model we see the general case of 13 qubits; however with some modifications we can increase this to 14. For MBQC we also have 13 qubits generally. We put this similarity down to the almost identical qubit state representations we have. This points to the state representation as an obvious bottle neck which we should concentrate upon improving in future development.

Finally, we compare the accuracy of both models. With the circuit model we describe a high level of accuracy and see that rounding errors would only have an effect after hundreds of gate applications. With MBQC we discovered an extremely interesting property. We saw that due to the frequency of

measurements (generally within 3 operations), rounding errors had no time to build up. Where we have patterns in standard form we apply all of our measurements together, followed by corrections. As the only corrections we apply are X and Y , which are both integer operators, rounding errors do not appear in this case either.

4.4.2 Equivalence & Conversion between Models

Our MBQC emulator is based upon much of the theory we developed for the circuit model. This includes the representation of qubits and corrections we use, as well as the way in which patterns combine to form a circuit. On top of this, we use measurement patterns that are equivalent to quantum gates in order to create circuits. The normal generation of patterns and circuits is equivalent to the circuit model.

Assuming our emulators both function correctly, implementing a circuit for the circuit model, and then combining the equivalent patterns of the circuit's component gates in the same way should produce identical outputs given the same input. As such, where we prove an algorithm or desire the behaviour of a specific circuit, we can equally show the same for the MBQC model when using equivalent patterns.

We have not demonstrated or investigated an equivalence of MBQC patterns for the circuit model. Beginning with the standard form of patterns, this may be an interesting line of investigation. We would likely find that some sequence of commands can be implemented with a unitary gate U .

Ultimately, whichever model we prefer for computation, given equivalences between models, we should be able to achieve the same computational result for any model. It is obvious from our investigations that developing an algorithm for the circuit model is infinitely easier than developing one using MBQC directly. However, MBQC may prove the most computationally effective, both physically and when emulating. We can then use equivalences to convert our new algorithm to MBQC, perform standardisation and minimisation of the pattern, and have the advantages of improved performance.

Chapter 5

Conclusions

At any rate, it seems that the laws of physics present no barrier to reducing the size of computers until bits are the size of atoms, and quantum behaviour holds sway.

Richard P. Feynman [25]

5.1 Introduction

To conclude this thesis we summarise what we have achieved, both in terms of deliverables and in terms of learning. We present three major contributions:

- An expansive and detailed summary of the theory of quantum computation, including the related mathematics and quantum mechanics.
- A circuit-based quantum computation emulator with which we implemented the Deutsch, Deutsch-Jozsa and Shor algorithms.
- We have developed the first measurement-based quantum computation emulator. We implement the Deutsch and Deutsch-Jozsa algorithms using patterns equivalent to quantum circuit gates. We were unable to implement Shor's algorithm due to the complexity of the problem.

The two emulators are the product of developing a sound understanding of the theory of quantum computation which is presented throughout the report. Ultimately we are able to accurately emulate quantum computation with up to 13 entangled qubits for both approaches. Our emulations are relatively fast and are flexible with respect to what circuits and patterns can be implemented with them.

In our evaluation we demonstrated the effects of the state explosion problem which prevented us from combining more than 13 qubits together. Our experimental results confirmed the hypothesis that memory would be our bounding factor for state representation.

While developing our MBQC theory we discovered that when measuring in the non-standard bases presented in the measurement calculus we cannot use von Neumann measurement for all values of α . We instead used generalised measurement (POVM).

The high accuracy of our qubit state space representations was unnecessary for our purposes, and investigations revealed that measurements removed any rounding errors. A decreased level of accuracy would allow a greater number of combined qubits to be emulated.

In our evaluation we present benchmarks for the algorithms we have implemented. For the circuit model a run of Shor for $N = 15$ takes 1.5 seconds. Our emulation fares well against existing solutions in this regard; however falls down due to the relatively small number of qubits that can be represented.

In our final section we consider a number of expansions to the project that could be considered in the future.

5.2 Future Work

We present a number of expansions to this project that we could investigate given more time. We first discuss methods of increasing the bounds of our implementation:

- There are two significant lines of investigation for increasing the bounds of emulation. The first is to increase the memory available on a single computer, for example using a super computer. The second is to distribute the implementation across many machines. Altering our system to utilise a super computer is a relatively straightforward task. Simply running our existing emulators with the additional resources offered by such a computer would see an immediate increase in bounds. From this we can make additional improvements such as threading computation so that we are able to utilise many CPU cores.
- Distributing the emulators we have developed is a difficult task. We must rethink the entire system and develop an implementation that is parallelisable and efficiently handles the distribution of resources (CPU and memory). We must also ensure that network communication does not slow down computation to the extent that any significant advantage of distribution is lost. For example, we should investigate the use of compression when sending large amounts of data. MPI (Message Passing Interface) would be the obvious distribution framework to use here. Targeting the use of regular workstation computers as opposed to high performance computers would be an advantageous approach and more readily usable by others.
- If we force operators and states to be written to the hard drive when memory becomes low, we only require enough physical memory to store one row and column of a state or operator in order to complete numerical operations. This would greatly slow down the run time but should vastly increase the limit of qubit representations. There are many algorithms that are able to perform these kinds of operations in an efficient manner to maximise the available resources.
- Our evaluation shows that our implementations are very accurate, but at the same time we use a significant amount of additional memory in order to achieve this. Reducing our aims for accuracy would allow us to decrease representation size, and therefore emulate more qubits with the same amount of memory.

We consider ways in which our deliverables could be made more useful and usable to potential users:

- We had no specific users in mind when developing the emulators, and as such, it became a specialised system requiring a good knowledge of its construction in order to use it. The most desirable solution would be a inclusive, polished, GUI based, end product. For the circuit model, we would allow the input of a circuit via the circuit calculus or via a circuit builder interface. For the measurement model, input would be via the measurement calculus. A similar builder for measurement patterns may be developed; however it is an unintuitive process and may prove too difficult to use. The output of quantum states and measurements should be in a number of forms such that the user can decide which is most meaningful to them for their purposes. We should be able to view any intermediate steps of computation; for example, the state of qubits between gates.
- Following on from the previous idea, outputting the intermediate steps of computation as a separate file, such as an image, text or \LaTeX would be useful for those wishing to understand a circuit. If this output was made human readable using surds, fractions and irrational numbers such as π , it would make working theoretically with algorithms much easier.
- An initial idea of this project was to incorporate teaching into the tool such that undergraduate students would be able to learn about quantum systems and quantum computation. The simulation of the two models followed as a part of this concept. Now that we have developed emulators, the addition of a teaching element to the tool would provide a useful introduction and explain the specifics of the tool while introducing the quantum topics.

- There is huge scope for further developing our circuit calculus. We have introduced the syntax and semantics of the calculus, but as we see from the measurement calculus, there is wide range of discussion that can result from it. An obvious expansion which we mention below is the translation between our circuit calculus and the measurement calculus. In our evaluation we mention two scenarios we do not account for. Specifically the specification of wires representing multiple qubits, and wires representing classical bits. Adding these features would enhance the flexibility of the calculus. In addition, the expansion to include input and output qubits in our calculus would allow us to specify complete quantum algorithms instead of circuits. As mentioned throughout this report, it is very difficult to develop intuition for MBQC. More so, we see circuit representations of measurement-based patterns in the literature. To expand our calculus to cover these cases would involve being able to specify the basis of measurements, linking dependent operators to the measurement upon which they depend, and allowing the arbitrary selection of control and target qubits in controlled gates.

Our final ideas for future investigation are into new theoretical areas:

- We have mentioned another measurement-based quantum computation approach called teleportation. An obvious expansion to what we have covered is to create an emulator for this architecture in a similar way to the emulators presented here. There are a number of other approaches to quantum computation which could be investigated in a similar way.
- Our MBQC emulator makes significant use of equivalences between quantum gates and measurement, and as such there is some implied conversion between models. Developing a full, two-way translation between models would be a significant achievement. This would likely be specified formally between calculi such as the circuit calculus we develop and the measurement calculus. From here the translation can be expanded to new approaches and we are then able to easily implement circuits and algorithms on any architecture.
- This idea leads onto a more detailed investigation into the comparisons between different models of computation. In our evaluation we have presented a brief and largely superficial look at the circuit and measurement-based models; however there is huge area for expansion in order to draw more generalised conclusions. Ultimately, successful model will be chosen based upon which lends itself best to physical implementation.
- We mention in 3.7.2 that we should be able to develop a series of rewrite rules in order to minimise sequences of commands and thus reduce the number of qubits a pattern requires. This is an interesting area to investigate as this will have application not only in our emulation, but also in real world implementations.

Bibliography

- [1] D. Abbott, C. R. Doering, C. M. Caves, D. M. Lidar, H. E. Brandt, A. R. Hamilton, D. K. Ferry, J. Gea-Banacloche, S. M. Bezrukov, and L. B. Kish. Dreams Versus Reality: Plenary Debate Session on Quantum Computing. *Quantum Information Processing*, 2(6):449–472, December 2003. doi: 10.1023/B:QINP.0000042203.24782.9a.
- [2] D. Aharonov. Quantum Computation. In D. Stauffer, editor, *Annual Reviews of Computational Physics*, volume VI. World Scientific, December 1998. URL <http://arxiv.org/abs/quant-ph/9812037v1>.
- [3] P. Aliferis and D. W. Leung. Computation by measurements: a unifying picture. *Physical Review A*, 70(6), December 2004. doi: 10.1103/PhysRevA.70.062314.
- [4] Amazon Inc. Amazon Elastic Compute Cloud (Amazon EC2), 2008. URL <http://aws.amazon.com/ec2/>.
- [5] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: An experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, November 2002. doi: 10.1145/581571.581573.
- [6] J. Bak and D. J. Newman. *Complex Analysis*. Springer, 2nd edition, 1996. ISBN 0387947566.
- [7] A. L. Beberg, D. L. Ensign, G. Jayachandran, S. Khaliq, and V. S. Pande. Folding@home: Lessons from eight years of volunteer distributed computing. In *IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 9781424437511. doi: 10.1109/IPDPS.2009.5160922.
- [8] G. Brassard. Searching a Quantum Phone Book. *Science*, 275(5300):627–628, January 1997. doi: 10.1126/science.275.5300.627.
- [9] I. Chuang. Quantum circuit viewer: qasm2circ, March 2005. URL <http://www.media.mit.edu/quanta/qasm2circ/>.
- [10] V. Danos, E. Kashefi, and P. Panangaden. The Measurement Calculus. *Journal of the ACM*, 54(2), April 2007. doi: 10.1145/1219092.1219096.
- [11] V. Danos, E. Kashefi, P. Panangaden, and S. Perdrix. Extended Measurement Calculus. In S. Gay and I. Mackie, editors, *Semantic Techniques for Quantum Computation*, chapter 7, pages 235–310. Cambridge University Press, New York, 2010. ISBN 052151374.
- [12] B. Dawes and D. Abrahams. Boost C++ Libraries. URL <http://www.boost.org/>.
- [13] A. de Vries. jQuantum: Quantum Computer Simulator, March 2010.
- [14] D. Deutsch. Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, July 1985. doi: 10.1098/rspa.1985.0070.

BIBLIOGRAPHY

- [15] D. Deutsch. Quantum Computational Networks. *Royal Society of London Proceedings Series A*, 425:73–90, September 1989.
- [16] D. Deutsch. *The Fabric of Reality: The Science of Parallel Universes and Its Implications*. Penguin Books, 1997. ISBN 014027541X.
- [17] D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London, Series A*, 439:553–558, October 1992.
- [18] P. A. M. Dirac. *The Principles of Quantum Mechanics (International Series of Monographs on Physics)*. Oxford University Press, USA, 4th edition, February 1982. ISBN 0198520115.
- [19] D. P. DiVincenzo. Quantum Computation. *Science*, 270(5234):255–261, October 1995. doi: 10.1126/science.270.5234.255.
- [20] D. P. DiVincenzo. Two-bit gates are universal for quantum computation. *Physical Review A*, 51(2):1015–1022, February 1995. doi: 10.1103/PhysRevA.51.1015.
- [21] D. P. DiVincenzo. Solid State Quantum Computing, 2000. URL http://www.research.ibm.com/ss_computing/.
- [22] D. P. DiVincenzo. The Physical Implementation of Quantum Computation. *Fortschritte der Physik*, H.-L., 2000. URL <http://arxiv.org/abs/quant-ph/0002077>. Experimental Proposals for Quantum Computation.
- [23] A. Edalat. Notes from the course 484: Quantum Computing. Department of Computing Imperial College London, 2009. Imperial College London. URL <http://www.doc.ic.ac.uk/~ae/teaching.html#quantum>.
- [24] A. Ekert, P. Hayden, and H. Inamori. Basic concepts in quantum computation. November 2000. URL <http://arxiv.org/abs/quant-ph/0011013v1>.
- [25] R. P. Feynman. Simulating Physics with Computers. *International Journal of Theoretical Physics*, 21(6):467–488, June 1982. doi: 10.1007/BF02650179.
- [26] R. P. Feynman. Quantum mechanical computers. *Foundations of Physics*, 16(6):507–531, June 1986. doi: 10.1007/BF01886518.
- [27] C. Fox. QCF: Quantum Computing Functions for MATLAB. Technical Report PARG-03-02, Pattern Analysis & Machine Learning Research Group, University of Oxford, 2003. URL <http://www.robots.ox.ac.uk/~parg/pubs/qcf.pdf>.
- [28] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall. Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004. URL <http://www.open-mpi.org/>.
- [29] J. Gilbert and L. Gilbert. *Linear Algebra and Matrix Theory*. Brooks Cole, 2nd edition, 2004. ISBN 0534405819.
- [30] D. T. Gillespie. *A quantum mechanics primer: An Elementary Introduction to the Formal Theory of Non-relativistic Quantum Mechanics*. John Wiley & Sons, New York, 1974. ISBN 0470299126.
- [31] D. Gottesman and I. Chuang. Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations. *November*, 402(6760):390–393, November 1999. doi: 10.1038/46503.

-
- [32] D. Greve. Qdd - a quantum computer emulation library, September 2007.
- [33] L. K. Grover. A fast quantum mechanical algorithm for database search. In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, New York, NY, USA, 1996. ACM Press.
- [34] L. K. Grover. Quantum Mechanics helps in searching for a needle in a haystack. *Physical Review Letters*, 79(2):325–328, July 1997. doi: 10.1103/PhysRevLett.79.325.
- [35] D. Hanneke, J. P. Home, J. D. Jost, J. M. Amini, D. Leibfried, and D. J. Wineland. Realization of a programmable two-qubit quantum processor. *Nature Physics*, 6:13–16, November 2009. doi: 10.1038/nphys-1453.
- [36] M. Hayward. Quantum Computing and Shor’s Algorithm. April 2008. URL <http://alumni.imsa.edu/~matth/quant/299/paper.pdf>.
- [37] R. Jozsa. An introduction to measurement based quantum computation. In D. G. Angelakis, editor, *Quantum Information Processing: from theory to experiment*, chapter 2, pages 137–158. IOS Press Inc., 2006. URL <http://arxiv.org/abs/quant-ph/0508124>.
- [38] P. Kaye, R. Laflamme, and M. Mosca. *An introduction to quantum computing*. Oxford University Press, 1st edition, November 2006. ISBN 019857049X.
- [39] D. W. Leung. Quantum computation by measurements. *International Journal of Quantum Information*, 2(1):33–43, March 2004. doi: 10.1142/S0219749904000055.
- [40] S. Lloyd. Universal Quantum Simulators. *Science*, 273(5278):1073–1078, August 1996. doi: 10.1126/science.273.5278.1073.
- [41] K. Michielsen and H. de Raedt. QCE: A Simulator for Quantum Computer Hardware, August 2003. URL <http://rugth30.phys.rug.nl/compphys0/qce.htm>.
- [42] G. E. Moorhouse. Notes from the seminar: Shor’s Algorithm for Factoring Large Integers. University of Wyoming, November 2000. URL <http://www.uwo.edu/moorhouse/slides/talk2.pdf>.
- [43] M. A. Nielsen. Quantum computation by measurement and quantum memory. *Physics Letters A*, 308(2-3):96–100, February 2003. doi: 10.1016/S0375-9601(02)01803-0.
- [44] M. A. Nielsen. Optical quantum computation using cluster states. *Physical Review Letters*, 93(4), July 2004. doi: 10.1103/PhysRevLett.93.040503.
- [45] M. A. Nielsen. Cluster-state quantum computation. *Reports on Mathematical Physics*, 57(1): 147–161, February 2006. doi: 10.1016/S0034-4877(06)80014-5.
- [46] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 1st edition, October 2000. ISBN 0521635039.
- [47] J. Niwa, K. Matsumoto, and H. Imai. General-purpose parallel simulator for quantum computing. *Physical Review A*, 66(6), December 2002. doi: 10.1103/PhysRevA.66.062317.
- [48] A. Peres. *Quantum Theory: Concepts and Methods*. Springer, 1st edition, 1993. ISBN 0792336321.
- [49] Quantiki. List of Quantum Computation Simulators. URL http://www.quantiki.org/wiki/List_of_QC_simulators.
-

BIBLIOGRAPHY

- [50] R. Raussendorf and H. J. Briegel. A One-Way Quantum Computer. *Physical Review Letters*, 86 (22):5188–5191, May 2001. doi: 10.1103/PhysRevLett.86.5188.
- [51] R. Raussendorf, D. E. Browne, and H. J. Briegel. Measurement-based quantum computation with cluster states. *Physical Review A*, 68(2), Aug 2003. doi: 10.1103/PhysRevA.68.022312.
- [52] B. S. Rubín. Our First Qubit Measurement, January 2009. URL http://bradrubin.com/Site/Blog/Entries/2009/1/3_Our_First_Qubit_Measurement.html.
- [53] J. H. Scanlon and B. Wieners. The Internet Cloud, September 1999. URL <http://www.thestandard.com/article/0,1902,5466,00.html>.
- [54] P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In S. Goldwasser, editor, *Proceedings of the 35th Annual Symposium on the Foundations of Computer Science*, volume 35, pages 124–134, 1994.
- [55] P. W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. doi: 10.1137/S0097539795293172.
- [56] J. Stolze and D. Suter. *Quantum Computing: A Short Course from Theory to Experiment*. Wiley-VCH, 2nd edition, 2008. ISBN 3527407871.
- [57] S. Turgut. Notes from the course PHYS 737: Generalized Measurements. Department of Physics Middle East Technical University, 2007. Middle East Technical University. URL <http://www.physics.metu.edu.tr/~sturgut/737/>.
- [58] M. Van den Nest, W. Dür, G. Vidal, and H. J. Briegel. Classical simulation versus universality in measurement-based quantum computation. *Physical Review A*, 75(1), January 2007. doi: 10.1103/PhysRevA.75.012337.
- [59] L. Vandersypen, M. Steffen, G. Breyta, C. Yannoni, M. Sherwood, and I. Chuang. Experimental realization of Shor’s quantum factoring algorithm using nuclear magnetic resonance. *Nature*, 414 (6866):883–887, 2001. doi: 10.1038/414883a.
- [60] H. Watanabe. QCAD: GUI environment for Quantum Computer Simulator, September 2009. URL <http://apollon.cc.u-tokyo.ac.jp/~watanabe/qcad/>.
- [61] N. S. Yanofsky. An Introduction to Quantum Computing. August 2007. URL <http://arxiv.org/abs/0708.0261>.
- [62] N. S. Yanofsky and M. A. Mannucci. *Quantum Computing for Computer Scientists*. Cambridge University Press, 1st edition, August 2008. ISBN 0521879965.

All web references accessed in May 2010.

Appendices

Appendix A

Related Mathematics

A.1 Introduction

An integral part of quantum mechanics is the understanding of complex numbers (A.3), and as such, it is an essential concept in quantum computing. Expanding on this, quantum theory relies heavily upon the use of linear algebra; more specifically, complex vector spaces (A.6). Both complex numbers and linear algebra will be covered here to a depth necessary to understand the mathematics involved in the project, with references to additional material. Those with a good understanding of these topics can begin at B and refer to previous sections where necessary. To highlight the unmistakable importance of these topics, it has been said that “quantum theory is cast in the language of complex vector spaces” [62], so we will begin with this.

A.2 Imaginary Numbers

Imaginary numbers were introduced as part of the theory algebraic equations, in short, to solve the equation

$$x^2 = -1. \tag{A.1}$$

In order to do so we must postulate that such a number i (termed the imaginary unit) exists such that

$$i^2 = -1 \quad \text{or} \quad i = \sqrt{-1}. \tag{A.2}$$

The set of imaginary numbers, \mathbb{I} , is made up of multiples of i with a real number, such as $2 \times i$, $3 \times i$, or $n \times i$ for any real number n .

A.3 Complex Numbers

Following on from this, a complex number, z , is a the composition of a real number and an imaginary number in the form

$$z = a + bi, \tag{A.3}$$

where a and b are termed the real and imaginary parts respectively and i denotes the imaginary unit.

The set of complex numbers is denoted \mathbb{C} , and with this it is possible to define both the sets of real numbers, \mathbb{R} , and imaginary numbers, \mathbb{I} , as subsets of \mathbb{C} . If the imaginary part of the complex number z is always taken as $b = 0$, the real number a is described by the complex number

$$z = a + 0i = a. \tag{A.4}$$

Conversely, if the real part of the complex number z is always taken as $a = 0$, the imaginary number b is described by the complex number

$$z = 0 + bi = bi. \tag{A.5}$$

A.4 Operations On Complex Numbers - \mathbb{C}

To understand how we will work with complex numbers, we will outline the results of various mathematical operations upon complex numbers $c_1 = a_1 + b_1i$ and $c_2 = a_2 + b_2i$. As the majority of readers should be familiar with the topic, we will simply include definitions of each operation as a reminder, and show worked examples where $c_1 = 2 + 3i$ and $c_2 = 17 + 5i$.

A.4.1 Addition

$$\begin{aligned} c_1 + c_2 &= (a_1 + b_1i) + (a_2 + b_2i) = (a_1 + a_2) + (b_1 + b_2)i \\ &= (2 + 3i) + (17 + 5i) = (2 + 17) + (3 + 5)i = 19 + 8i \end{aligned} \quad (\text{A.6})$$

A.4.2 Subtraction

$$\begin{aligned} c_1 - c_2 &= (a_1 + b_1i) - (a_2 + b_2i) = (a_1 - a_2) + (b_1 - b_2)i \\ &= (2 + 3i) - (17 + 5i) = (2 - 17) + (3 - 5)i = -15 - 2i \end{aligned} \quad (\text{A.7})$$

A.4.3 Multiplication

$$\begin{aligned} c_1 \times c_2 &= (a_1 + b_1i) \times (a_2 + b_2i) = (a_1a_2 - b_1b_2) + (a_2b_1 + a_1b_2)i \\ &= (2 + 3i) \times (17 + 5i) = (2 \times 17 - 3 \times 5) + (2 \times 5 + 3 \times 17)i = 19 + 61i \end{aligned} \quad (\text{A.8})$$

A.4.4 Division

For division of complex numbers we must recall that division is defined as the inverse of multiplication, giving us

$$\begin{aligned} \frac{c_1}{c_2} &= \frac{a_1 + b_1i}{a_2 + b_2i} = \frac{a_1a_2 + b_1b_2}{a_2^2 + b_2^2} + \frac{a_2b_1 - a_1b_2}{a_2^2 + b_2^2}i \\ &= \frac{2 + 3i}{17 + 5i} = \frac{2 \times 17 + 3 \times 5}{17^2 + 5^2} + \frac{17 \times 3 - 2 \times 5}{17^2 + 5^2}i = \frac{60}{314} \end{aligned} \quad (\text{A.9})$$

A.4.5 Conjugation

$$c = a + bi, \quad \bar{c} = a - bi \quad (\text{A.10})$$

$$\bar{c}_1 = \overline{(2 + 3i)} = 2 - 3i \quad (\text{A.11})$$

A.4.6 Modulus (Magnitude)

$$\begin{aligned} |c_1| &= |a + bi| = \sqrt{a^2 + b^2} \\ &= |2 + 3i| = \sqrt{13} \end{aligned} \quad (\text{A.12})$$

When describing the normalisation of a complex number c , we are referring to $|c|^2$:

$$\begin{aligned} |c_1|^2 &= c_1 \times \bar{c}_1 \\ |2 + 3i|^2 &= (2 + 3i) \times (2 - 3i) \\ \sqrt{13}^2 &= (4 + 9) + (6 - 6)i = 13. \end{aligned} \quad (\text{A.13})$$

It will become apparent how the normalisation of complex numbers is extremely important as we come to manipulate quantum bits.

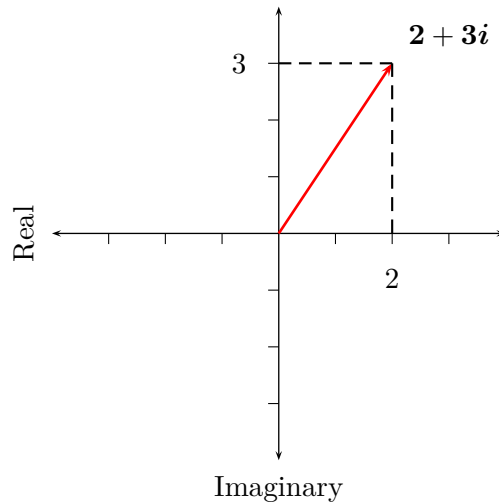


Figure A.1: Cartesian representation of complex number $2 + 3i$

A.5 Representation of Complex Numbers

Beyond the algebraic applications of complex numbers, their significance is also apparent in geometry. With this, we see how it opens up the advantages of their use in physics.

A.5.1 Cartesian Representation

We can see in Figure A.1 how a complex number can be represented as a vector on the Argand plane (complex plane), with the axes determining the real and imaginary parts of the number.

Defining a complex number z as a pair of real numbers representing the real and imaginary parts, a and b respectively, of the complex number gives us

$$z \mapsto (a, b), \quad (\text{A.14})$$

the mapping of a complex number to the complex plane described. Effectively this provides us with a point in the Cartesian two-dimensional coordinate system. In our example it would give us

$$c_1 \mapsto (2, 3). \quad (\text{A.15})$$

As this effectively allows us to ignore the presence of i it is very useful and we will concentrate upon this representation of complex numbers.

A.5.2 Polar Representation

We can also determine a complex number z from its modulus (magnitude) ρ and angle (phase) θ :

$$z \mapsto (\rho, \theta). \quad (\text{A.16})$$

This form, as shown in Figure A.2, is described as the polar representation of complex numbers. The values of ρ and θ are computed from real and imaginary parts a and b by

$$\rho = \sqrt{a^2 + b^2} \quad \text{and} \quad \theta = \tan^{-1} \left(\frac{b}{a} \right), \quad (\text{A.17})$$

or reversed using

$$a = \rho \cos(\theta) \quad \text{and} \quad b = \rho \sin(\theta). \quad (\text{A.18})$$

Our example gives

$$c_1 \mapsto \left(\sqrt{4 + 9}, \tan^{-1} \left(\frac{3}{2} \right) \right) = \left(\sqrt{13}, 0.983 \right). \quad (\text{A.19})$$

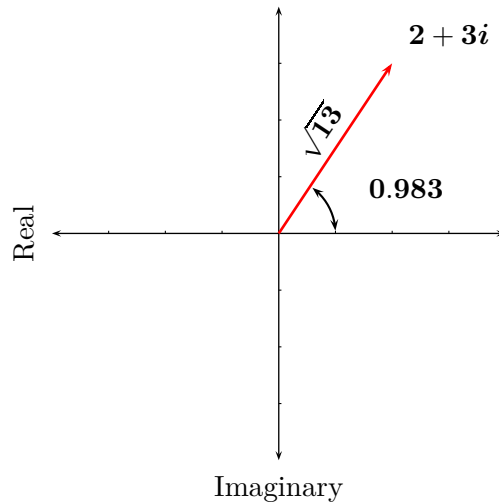


Figure A.2: Polar representation of complex number $2 + 3i$

Considering the geometry of complex numbers, we can perform many of the operations described previously (A.4) through geometric transformations or techniques. We will not cover this here; however any textbook on linear algebra would cover this topic.

A.6 Complex Vector Spaces

Taking complex numbers a step further, we will provide a very brief introduction to the vast subject of complex vector spaces. Our motive for covering this topic is that the state of a quantum system corresponds to a vector in a complex vector space. We will use vectors to describe states and matrices, covered later (A.8), as a way to describe dynamics.

In computing terms, a vector can be considered to be a one-dimensional array. For the majority of our examples we shall use the n -dimensional complex vector space \mathbb{C}^n . This vector space includes all vectors of some fixed dimension containing complex numbers. The specific case we will begin with is an ordered set of dimension 6, denoted $\mathbb{C}^6 = \mathbb{C} \times \mathbb{C} \times \mathbb{C} \times \mathbb{C} \times \mathbb{C} \times \mathbb{C}$, with an example of such a complex vector represented as

$$V = \begin{bmatrix} 2 + 3i \\ 17 + 5i \\ 3 - 2i \\ 11 - 7i \\ 4.4 + 4i \\ -i \end{bmatrix}. \tag{A.20}$$

However, everything we will show for vectors of size 6 can be applied to vectors of size n , and so the set \mathbb{C}^n for a fixed (but arbitrary) size n therefore has the structure of a complex vector space.

With our example vector termed V we would represent the j th term of this vector as $V[j]$, and following the convention of programming with arrays, we will take the first (top) term as 0. For example $V[2]$ would refer to $3 - 2i$.

As a special case, the vector $\mathbf{0}$, defined as $\mathbf{0}[j] = 0$, has the property

$$V + \mathbf{0} = V = \mathbf{0} + V, \quad \forall V \in \mathbb{C}^n. \tag{A.21}$$

A.7 Operations on Complex Vectors - \mathbb{C}^n

We will look at the definitions of the various operations we can carry out with such vectors. These operations should be familiar as they are simply a combination of the operations applied to vectors and the operations applied to complex numbers discussed above. We will show examples of each operation using complex vectors V and U ,

$$V = \begin{bmatrix} 2 + 3i \\ 17 + 5i \\ 3 - 2i \\ 11 - 7i \\ 4.4 + 4i \\ -i \end{bmatrix}, \quad U = \begin{bmatrix} 8 \\ 9.1i \\ 5 - i \\ -2i \\ 1 + i \\ 1.5 \end{bmatrix} \quad (\text{A.22})$$

taken from the set \mathbb{C}^6 , as well as the complex number $z = 4 - 2i$.

A.7.1 Addition

$$(V + U)[j] = V[j] + U[j] \quad (\text{A.23})$$

$$\begin{aligned} V + U &= \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix} + \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} v_0 + u_0 \\ v_1 + u_1 \\ v_2 + u_2 \\ v_3 + u_3 \\ v_4 + u_4 \\ v_5 + u_5 \end{bmatrix} \\ &= \begin{bmatrix} 2 + 3i \\ 17 + 5i \\ 3 - 2i \\ 11 - 7i \\ 4.4 + 4i \\ -i \end{bmatrix} + \begin{bmatrix} 8 \\ 9.1i \\ 5 - i \\ -2i \\ 1 + i \\ 1.5 \end{bmatrix} = \begin{bmatrix} 10 + 3i \\ 17 + 14.1i \\ 8 - 3i \\ 11 - 9i \\ 5.4 + 5i \\ 1.5 - i \end{bmatrix} \end{aligned} \quad (\text{A.24})$$

A.7.2 Subtraction

$$(V - U)[j] = V[j] - U[j] \quad (\text{A.25})$$

$$V - U = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix} - \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} v_0 - u_0 \\ v_1 - u_1 \\ v_2 - u_2 \\ v_3 - u_3 \\ v_4 - u_4 \\ v_5 - u_5 \end{bmatrix}$$

$$= \begin{bmatrix} 2 + 3i \\ 17 + 5i \\ 3 - 2i \\ 11 - 7i \\ 4.4 + 4i \\ -i \end{bmatrix} - \begin{bmatrix} 8 \\ 9.1i \\ 5 - i \\ -2i \\ 1 + i \\ 1.5 \end{bmatrix} = \begin{bmatrix} -6 + 3i \\ 17 - 4.1i \\ -2 - i \\ 11 - 5i \\ 3.4 + 3i \\ -1.5 - i \end{bmatrix} \quad (\text{A.26})$$

A.7.3 Multiplication by Scalar

$$(z.V)[j] = z \times V[j] \quad (\text{A.27})$$

$$\begin{aligned} z.V &= z \cdot \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix} = \begin{bmatrix} z \times v_0 \\ z \times v_1 \\ z \times v_2 \\ z \times v_3 \\ z \times v_4 \\ z \times v_5 \end{bmatrix} \\ &= \begin{bmatrix} (4 - 2i) \times (2 + 3i) \\ (4 - 2i) \times (17 + 5i) \\ (4 - 2i) \times (3 - 2i) \\ (4 - 2i) \times (11 - 7i) \\ (4 - 2i) \times (4.4 + 4i) \\ (4 - 2i) \times (0 - i) \end{bmatrix} = \begin{bmatrix} 14 + 8i \\ 78 - 14i \\ 8 - 14i \\ 30 - 50i \\ 25.6 + 7.2i \\ -2 - 4i \end{bmatrix} \end{aligned} \quad (\text{A.28})$$

A.7.4 Additive Inverse (Negative)

$$(-V)[j] = -(V[j]) \quad (\text{A.29})$$

$$-V = \begin{bmatrix} -v_0 \\ -v_1 \\ -v_2 \\ -v_3 \\ -v_4 \\ -v_5 \end{bmatrix} = \begin{bmatrix} -2 - 3i \\ -17 - 5i \\ -3 + 2i \\ -11 + 7i \\ -4.4 - 4i \\ i \end{bmatrix} \quad (\text{A.30})$$

A.8 Operations on Complex Matrices - $\mathbb{C}^{m \times n}$

We will also require operations upon complex matrices, whose definitions will expand upon those of vectors. The definitions will be made for the complex vector space $\mathbb{C}^{m \times n}$, where this is the set of m by n matrices containing complex numbers. Again, these matrices will be treated as arrays and referenced as such to aid the transition to implementation. Given a matrix $A \in \mathbb{C}^{m \times n}$, entries are

referenced as follows:

$$A = \begin{matrix} & \mathbf{0} & \mathbf{1} & \cdots & \mathbf{n-2} & \mathbf{n-1} \\ \mathbf{0} & c_{0,0} & c_{0,1} & \cdots & c_{0,n-2} & c_{0,n-1} \\ \mathbf{1} & c_{1,0} & c_{1,1} & \cdots & c_{1,n-2} & c_{1,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \mathbf{m-2} & c_{m-2,0} & c_{m-2,1} & \cdots & c_{m-2,n-2} & c_{m-2,n-1} \\ \mathbf{m-1} & c_{m-1,0} & c_{m-1,1} & \cdots & c_{m-1,n-2} & c_{m-1,n-1} \end{matrix} . \tag{A.31}$$

This is equivalent to the multidimensional array

```

1 T matrixA[m][n];
2 for (int j = 0; j < m; ++j)
3     for (int k = 0; k < n; ++k)
4         matrixA[j][k] = c_{j,k};

```

where the notation `c_{j,k}` refers to the relevant complex number entry, and `T` is the type (such as `std::complex<double>`).

As our implementation will be in C++ we have chosen to use the Boost C++ libraries, specifically the uBLAS library which provides basic linear algebra constructions [12]. Instead of working with multidimensional arrays, we use the provided `ublas::matrix` type which provides us with many of following operations. As such, our above example is replaced by the following:

```

1 ublas::matrix<T> matrixA(m, n);
2 for (unsigned int j = 0; j < matrixA.size1(); ++j)
3     for (unsigned int k = 0; k < matrixA.size2(); ++k)
4         matrixA(j, k) = c_{j,k};

```

As you can see, references to entries are made by row and then by column. For example $A[j, k]$ refers to $c_{j,k}$. We use $A[j, -]$ ($A_{j,-}$) to refer to the j th row, and $A[-, k]$ ($A_{-,k}$) refers to the k th column. The operations below will be defined using two matrices $A, B \in \mathbb{C}^{m \times n}$.

A.8.1 Addition

$$(A + B)[j, k] = A[j, k] + B[j, k] \tag{A.32}$$

$$A + B = \begin{bmatrix} a_{0,0} + b_{0,0} & a_{0,1} + b_{0,1} & \cdots & a_{0,n-1} + b_{0,n-1} \\ a_{1,0} + b_{1,0} & a_{1,1} + b_{1,1} & \cdots & a_{1,n-1} + b_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m-1,0} + b_{m-1,0} & a_{m-1,1} + b_{m-1,1} & \cdots & a_{m-1,n-1} + b_{m-1,n-1} \end{bmatrix} \tag{A.33}$$

A.8.2 Subtraction

$$(A - B)[j, k] = A[j, k] - B[j, k] \tag{A.34}$$

A.8.3 Multiplication by Scalar

$$(z.A)[j, k] = z \times A[j, k] \tag{A.35}$$

A.8.4 Matrix Multiplication

Matrix multiplication requires matrices A and B where $A \in \mathbb{C}^{m \times n}$, $B \in \mathbb{C}^{n \times p}$ and forms the matrix $AB \in \mathbb{C}^{m \times p}$. It is defined as

$$(AB)[j, k] = \sum_{x=0}^{n-1} (A[j, x] \times B[x, k]). \quad (\text{A.36})$$

It must be noted that matrix multiplication is not commutative, and so

$$AB \neq BA. \quad (\text{A.37})$$

Despite this, if the dimensions of the matrices are not square then the matrix multiplication BA would not be possible.

To begin with we will show a generalised example where we multiply the row vectors of A by the column vectors of B :

$$\begin{aligned} AB &= \begin{bmatrix} A_{0,-} \\ A_{1,-} \\ \vdots \\ A_{m-1,-} \end{bmatrix} \begin{bmatrix} B_{-,0} & B_{-,1} & \cdots & B_{-,p-1} \end{bmatrix} \\ &= \begin{bmatrix} A_{0,-} \cdot B_{-,0} & A_{0,-} \cdot B_{-,1} & \cdots & A_{0,-} \cdot B_{-,p-1} \\ A_{1,-} \cdot B_{-,0} & A_{1,-} \cdot B_{-,1} & \cdots & A_{1,-} \cdot B_{-,p-1} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m-1,-} \cdot B_{-,0} & A_{m-1,-} \cdot B_{-,1} & \cdots & A_{m-1,-} \cdot B_{-,p-1} \end{bmatrix}. \end{aligned} \quad (\text{A.38})$$

If we reverse the use of vectors to the column vectors of A and row vectors of B , we can also have

$$\begin{aligned} AB &= \begin{bmatrix} A_{-,0} & A_{-,1} & \cdots & A_{-,n-1} \end{bmatrix} \begin{bmatrix} B_{0,-} \\ B_{1,-} \\ \vdots \\ B_{n-1,-} \end{bmatrix} \\ &= A_{-,0} \otimes B_{0,-} + A_{-,1} \otimes B_{1,-} + \cdots + A_{-,n-1} \otimes B_{n-1,-}, \end{aligned} \quad (\text{A.39})$$

where \otimes represents the tensor product described in A.8.12.

Looking at a concrete example of the first generalisation we will use matrices A and B defined as

$$A = \begin{bmatrix} 2 + 3i & 11 - 7i \\ 17 + 5i & 4 + 4i \\ 3 - 2i & -i \end{bmatrix}, \quad B = \begin{bmatrix} 8 & 9i & 5 - i & -2i \\ 1 + i & 2 & 3 - i & 8i \end{bmatrix}; \quad (\text{A.40})$$

$$AB[0, 0] = ((2 + 3i) \times 8) + ((11 - 7i) \times (1 + i)) \quad (\text{A.41})$$

$$AB[0, 1] = ((2 + 3i) \times 9i) + ((11 - 7i) \times 2) \quad (\text{A.42})$$

$$AB[1, 1] = ((17 + 5i) \times 9i) + ((4 + 4i) \times 2) \quad (\text{A.43})$$

...

$$AB = \begin{bmatrix} (16 + 24i)+ & (18i - 27)+ & (13 + 13i)+ & (6 - 4i)+ \\ (18 + 4i) & (22 - 14i) & (8 - 6i) & (56 + 88i) \\ (136 + 40i)+ & (153i - 45)+ & (90 + 8i)+ & (10 - 34i)+ \\ 8i & (8 + 8i) & (16 + 8i) & (32i - 32) \\ (24 - 16i)+ & (18 + 27i)+ & (13 - 13i)+ & (-4 - 6i)+ \\ (1 - i) & (-2i) & (-1 - 3i) & 8 \end{bmatrix}$$

$$= \begin{bmatrix} 34 + 28i & 4i - 5 & 21 + 7i & 62 + 84i \\ 136 + 48i & 161i - 37 & 106 + 16i & -22 - 2i \\ 25 - 17i & 18 + 25i & 12 - 16i & 4 - 6i \end{bmatrix} \quad (\text{A.44})$$

A.8.4.1 Identity Matrix

We define an n by n square matrix termed the identity matrix, I , which acts as a unit of matrix multiplication, as

$$I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \quad (\text{A.45})$$

for any size n , and has the property

$$IA = A = AI. \quad (\text{A.46})$$

The 3 by 3 identity matrix is

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (\text{A.47})$$

A.8.5 Additive Inverse (Negative)

$$(-A)[j, k] = -(A[j, k]) \quad (\text{A.48})$$

A.8.6 Multiplicative Inverse

Where an n by n matrix A is invertible, we have some n by n matrix A^{-1} such that

$$AA^{-1} = A^{-1}A = I \quad (\text{A.49})$$

where I is the identity matrix of size n by n . If such an A^{-1} does not exist, A is described as singular. Determining the inverse of a matrix is a computationally difficult task.

A.8.7 Transposition

Transposition is a function from $\mathbb{C}^{m \times n}$ to $\mathbb{C}^{n \times m}$:

$$A^T[j, k] = A[k, j]. \quad (\text{A.50})$$

A.8.8 Conjugation

Conjugation is a function from $\mathbb{C}^{m \times n}$ to $\mathbb{C}^{n \times m}$ and the notation is overloaded as an operation on complex numbers and matrices:

$$\overline{A}[j, k] = \overline{A[j, k]}. \quad (\text{A.51})$$

A.8.9 Dagger (Adjoint)²

The dagger operator is a combination of the above transpose and conjugate operations, so is also a function from $\mathbb{C}^{m \times n}$ to $\mathbb{C}^{n \times m}$:

$$A^\dagger[j, k] = \overline{A[k, j]}. \quad (\text{A.52})$$

For an example of the use of the dagger operator see Equation B.18 in B.3.

²The “dagger” operator notation is universally used over “adjoint” in quantum mechanics.

A.8.9.1 Hermitian Matrix

An n by n matrix A that satisfies

$$\begin{aligned} A^\dagger &= A \\ A[j, k] &= \overline{A[k, j]} \end{aligned} \tag{A.53}$$

is termed a Hermitian matrix with the operator such a matrix represents termed self-adjoint. From the definition we see that the diagonal entries must be real. An example of such a matrix is

$$\begin{bmatrix} -3.5 & 3 - 2i & i \\ 3 + 2i & 6 & -4 \\ -i & -4 & 2 \end{bmatrix}. \tag{A.54}$$

A.8.9.2 Unitary Matrix

An n by n matrix U is described as unitary if

$$UU^\dagger = U^\dagger U = I, \tag{A.55}$$

where I is the identity matrix also of size n by n . You can see an example of such a matrix in B.3, shown in Equation B.16. We also see that by definition, U^\dagger is also the multiplicative inverse of U . As we mentioned, determining the inverse of a matrix is computationally expensive, so using unitary matrices is advantageous in the case where the inverse matrix is commonly required.

A.8.10 Inner Product (Dot Product)

The inner product operates on square matrices in the complex vector space $\mathbb{C}^{n \times n}$ and produces a real number. It is defined as

$$\langle A, B \rangle = \sum_{j=0}^{n-1} (A^\dagger B)[j, j]. \tag{A.56}$$

A.8.11 Eigenvalues and Eigenvectors

If there is some complex number c and a vector $V \in \mathbb{C}^n$ and $V \neq 0$ for a matrix $A \in \mathbb{C}^{n \times n}$ there is

$$AV = c.V, \tag{A.57}$$

then c is an eigenvalue and V is an eigenvector of A . The matrix A can effectively be replaced by c with only a change to the length of the vector and not the direction. An interesting note is that eigenvalues for Hermitian matrices are always real.

A.8.12 Tensor Product (Kronecker Product)

The tensor product of two vector spaces (quantum systems) provides a description of both spaces (systems) as one. It is described as the fundamental building operation of quantum systems [62]. The tensor product is the matrix holding every element of the first matrix scalar multiplied with the entirety of the second matrix. We can define the tensor product of two matrices A and B where $A \in \mathbb{C}^{m \times n}$, $B \in \mathbb{C}^{p \times q}$ forming the a matrix $A \otimes B \in \mathbb{C}^{mp \times nq}$ as

$$(A \otimes B)[j, k] = A[j/p, k/q] \times B[j \bmod p, k \bmod q] \tag{A.58}$$

$$\begin{aligned}
 A \otimes B &= \begin{bmatrix} a_{0,0} \cdot B & \cdots & a_{0,n-1} \cdot B \\ \vdots & \ddots & \vdots \\ a_{m-1,0} \cdot B & \cdots & a_{m-1,n-1} \cdot B \end{bmatrix} \\
 &= \begin{bmatrix} a_{0,0} \cdot \begin{bmatrix} b_{0,0} & \cdots & b_{0,q-1} \\ \vdots & \ddots & \vdots \\ b_{p-1,0} & \cdots & b_{p-1,q-1} \end{bmatrix} & \cdots \\ \vdots & \ddots \\ a_{m-1,0} \cdot \begin{bmatrix} b_{0,0} & \cdots & b_{0,q-1} \\ \vdots & \ddots & \vdots \\ b_{p-1,0} & \cdots & b_{p-1,q-1} \end{bmatrix} & \cdots \end{bmatrix} \\
 &= \begin{bmatrix} a_{0,0} \times b_{0,0} & a_{0,0} \times b_{0,1} & \cdots & a_{0,n-1} \times b_{0,q-2} & a_{0,n-1} \times b_{0,q-1} \\ a_{0,0} \times b_{1,0} & a_{0,0} \times b_{1,1} & \cdots & a_{0,n-1} \times b_{1,q-2} & a_{0,n-1} \times b_{1,q-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{1,0} \times b_{0,0} & a_{1,0} \times b_{0,1} & \cdots & a_{1,n-1} \times b_{0,q-2} & a_{1,n-1} \times b_{0,q-1} \\ a_{1,0} \times b_{1,0} & a_{1,0} \times b_{1,1} & \cdots & a_{1,n-1} \times b_{1,q-2} & a_{1,n-1} \times b_{1,q-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m-1,0} \times b_{p-2,0} & a_{m-1,0} \times b_{p-2,1} & \cdots & a_{m-1,n-1} \times b_{p-2,q-2} & a_{m-1,n-1} \times b_{p-2,q-1} \\ a_{m-1,0} \times b_{p-1,0} & a_{m-1,0} \times b_{p-1,1} & \cdots & a_{m-1,n-1} \times b_{p-1,q-2} & a_{m-1,n-1} \times b_{p-1,q-1} \end{bmatrix} \quad (\text{A.59})
 \end{aligned}$$

Looking at matrices A and B where $m = p = 2$ and $n = q = 2$ gives

$$\begin{aligned}
 A \otimes B &= \begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix} \otimes \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} \\
 &= \begin{bmatrix} a_{0,0} \cdot \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} & a_{0,1} \cdot \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} \\ a_{1,0} \cdot \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} & a_{1,1} \cdot \begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix} \end{bmatrix} \\
 &= \begin{bmatrix} a_{0,0} \times b_{0,0} & a_{0,0} \times b_{0,1} & a_{0,1} \times b_{0,0} & a_{0,1} \times b_{0,1} \\ a_{0,0} \times b_{1,0} & a_{0,0} \times b_{1,1} & a_{0,1} \times b_{1,0} & a_{0,1} \times b_{1,1} \\ a_{1,0} \times b_{0,0} & a_{1,0} \times b_{0,1} & a_{1,1} \times b_{0,0} & a_{1,1} \times b_{0,1} \\ a_{1,0} \times b_{1,0} & a_{1,0} \times b_{1,1} & a_{1,1} \times b_{1,0} & a_{1,1} \times b_{1,1} \end{bmatrix}. \quad (\text{A.60})
 \end{aligned}$$

Using the tensor product we can sometimes rewrite a vector, for example

$$\begin{bmatrix} 15 & 6 & 9 & 5 & 2 & 3 \end{bmatrix}^T \in \mathbb{C}^6 = \mathbb{C}^2 \times \mathbb{C}^3, \quad (\text{A.61})$$

as a simplified pair of vectors,

$$\begin{bmatrix} 3 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 5 \\ 2 \\ 3 \end{bmatrix}. \quad (\text{A.62})$$

If this is possible the vector is called separable. Where we cannot rewrite a vector in this way, those that can be written as the non-trivial sum of tensors are called entangled.

A.8.13 Powers of Matrices

We briefly look at powers of matrices, specifically so we can remind the reader of the value of A^0 . Generally we have

$$A^x = \prod_{y=1}^x A. \quad (\text{A.63})$$

For the two cases we are likely to see, we have

$$A^0 = I \quad \text{and} \quad A^1 = A, \quad (\text{A.64})$$

where I is the identity matrix.

A.9 Further Reading

As we have touched on a number of expansive and advanced topics there is a large amount of additional material available to expand on what has been introduced. Any textbook on linear algebra would cover the majority of topics and provide enough examples and illustrations for most readers. More advanced material on linear algebra may be found in [6], and complex vector spaces are covered in [29].

Appendix B

Quantum Mechanics

B.1 Introduction

The topic of quantum mechanics may appear the most unfamiliar to many readers; as a scientific model, quantum mechanics describes various concepts and phenomena, some of which directly oppose ideas ingrained as common sense. Ideas of the macroscopic world must be put aside; when considering things at the atomic and subatomic level, these rules no longer apply. As we will describe, the primary difference between classical and quantum computers is the utilisation of quantum mechanics. The fact that they are bound by quantum laws offers advantages, but also requires the reader to gain an understanding of these unfamiliar notions.

As previously mentioned, we will focus as little as possible upon the advanced physics and mathematics relating to quantum computing due to the computer science bias of this project. However, an understanding of some basic quantum mechanics is required to comprehend the main ideas and problems associated with emulating a quantum computer and justifying its correctness.

One of our main restrictions is to constrict our investigation to finite dimensional quantum mechanics, alleviating us of much of the more complex mathematics and physics. This restriction will allow us to represent system states using vector spaces of finite dimension, which consist of finite vectors with complex entries. We can change these vectors using multiplication by operators or matrices (which are themselves finite and contain complex entries).

Throughout this section we will develop different systems to aid our explanations, incorporating descriptions of the various quantum mechanical phenomena at each stage. By the end we should arrive at a usable representation of a quantum system which will provide the basis of our main topic: quantum computing.

B.2 Deterministic vs. Probabilistic Systems

The first thing to appreciate in regards to quantum mechanics is that it goes against many of the ideas we are brought up believing in, on both a scientific and a human scale. These ideas must be put on hold if the reader is to successfully grasp this unnatural new way of thinking required to comprehend quantum physics. We will help to bridge this gap by first discussing the differences between classical deterministic and classical probabilistic systems.

B.2.1 Classical Deterministic Systems

We will begin by modelling a deterministic system. Figure B.1 shows six positions at which coins can be stacked, with the number of coins currently placed at each. The starting state of the system can be represented by a vector $S = [2 \ 7 \ 5 \ 1 \ 3 \ 11]^T$ of size 6. This state indicates there are 2 coins at position 0, 7 coins at position 1, 5 coins at position 2, and so on.

The next part of the model we require are the dynamics, referring to how the system (states) will change. We will create a graph with directed edges (Figure B.2) to show how coins will move from

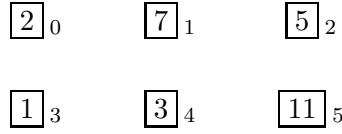


Figure B.1: Deterministic coin stacks model

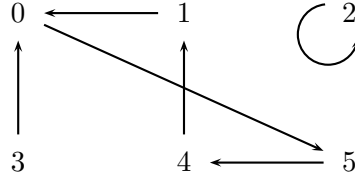


Figure B.2: Dynamics of deterministic coin stacks model

one stack to another, as this is what we wish to model. Each edge from one vertex, v_x , to another, v_y , shows that during one time click, all coins at v_x will move to v_y . There is one restriction made for the systems we will be modelling; for all vertices there should be exactly one outgoing edge, corresponding to a classical deterministic system.

As mentioned previously, we now have vector S representing the coin stack state, and we can create a matrix M from the dynamics determined in Figure B.2:¹

$$M[j, k] = \begin{cases} 1 & \text{if there is an arrow from vertex } k \text{ to vertex } j \\ 0 & \text{otherwise} \end{cases} \tag{B.1}$$

$$M = \begin{matrix} & & & & & & \text{from vector} \\ & & & & & & \mathbf{0} \ \mathbf{1} \ \mathbf{2} \ \mathbf{3} \ \mathbf{4} \ \mathbf{5} \\ \mathbf{0} & & & & & & \left[\begin{array}{cccccc} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{array} \right] \\ \mathbf{1} & & & & & & \\ \mathbf{2} & & & & & & \\ \mathbf{3} & & & & & & \\ \mathbf{4} & & & & & & \\ \mathbf{5} & & & & & & \end{matrix} \cdot \tag{B.2}$$

With this Boolean matrix, we can see that our earlier restriction has left us with a class of Boolean matrices with a single 1 entry in each column.

As stated, the vector X represents the current state, say at time t , now the dynamics of the system are defined as M we can determine the state of the system after one time click at time $t + 1$. We will represent the new state as a vector Y using matrix multiplication:²

$$Y[j] = (MX)[j] = \sum_{z=0}^5 (M[j, z] \times X[z]) \tag{B.3}$$

¹Note the definition of M states an entry is 1 where there is an arrow from vertex k to j , which may be the opposite of some texts and the reader's intuition.

²For a reminder of matrix multiplication see A.8.4.

$$Y = MX = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 7 \\ 5 \\ 1 \\ 3 \\ 11 \end{bmatrix} = \begin{bmatrix} 7+1 \\ 3 \\ 5 \\ 0 \\ 11 \\ 2 \end{bmatrix} = \begin{bmatrix} 8 \\ 3 \\ 5 \\ 0 \\ 11 \\ 2 \end{bmatrix}. \quad (\text{B.4})$$

To describe this new state, we can see that each vertex in Y contains the sum of all coins at vertices in X which had arrows pointing to it. In our new state, $Y[3] = 0$ as there are no arrows pointing to vertex 3 in our model.

From this we will eventually move towards a quantum system which works in the same way; states are represented by vectors, matrices represent system dynamics, and multiplying a state by a dynamic provides a new state one time click later.

Continuing with our coin stack model, we are able to calculate a dynamic based on M , but which gives us the state two time clicks later instead of one. We do this by multiplying M by itself: $MM = M^2$. We must be careful that as we are working with Boolean matrices we use Boolean operators here.

$$M^2[j, k] = \bigvee_{z=0}^5 M[j, z] \wedge M[z, k] \quad (\text{B.5})$$

$$M^2[j, k] = \begin{cases} 1 & \text{if there is a path from vertex } k \text{ to vertex } j \text{ of length 2} \\ 0 & \text{otherwise} \end{cases} \quad (\text{B.6})$$

$$M^2 = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}. \quad (\text{B.7})$$

We can generalise this for a path of arbitrary length n :

$$M^n[j, k] = \begin{cases} 1 & \text{if there is a path from vertex } k \text{ to vertex } j \text{ of length } n \\ 0 & \text{otherwise} \end{cases}. \quad (\text{B.8})$$

Looking back at our interaction between states and dynamics we can see that from the current state X we can define the state in n time steps as

$$Y = M^n X. \quad (\text{B.9})$$

Where we have two or more dynamics that act upon states, the action of one followed by another is described by their matrix product.

B.2.2 Classical Probabilistic Systems

Now we have gained an understanding of deterministic behaviour and created a working model, we will move on to probabilistic systems. We cannot be sure of the next state of a system or its dynamics as in our deterministic model, we must instead base our calculations upon probabilities. Generalising our previous model to use one coin and decreasing the number of stacks to four, we can represent the systems state as a column vector $X = \left[\frac{1}{14} \quad \frac{3}{7} \quad \frac{2}{7} \quad \frac{3}{14} \right]^T$ of size 4. Instead of representing the

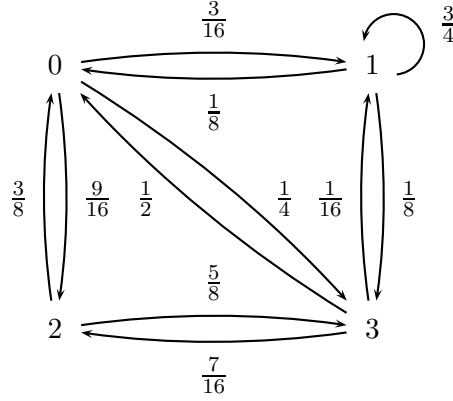


Figure B.3: Dynamics of probabilistic coin stacks model

number of coins, we now have one coin with the state X representing the probability that the coin will be in each stack. As the coin must always be at one of the positions the sum of probabilities is 1.

The dynamics of the model must also be generalised. We will allow several arrows to leave each vertex, adding weights (probabilities) between 0 and 1 to each arrow.³ To ensure we do not lose or gain any coins from the system we will ensure probabilities leaving a vector sum to 1, and those arriving at a vector sum to 1. Our new dynamics are shown in Figure B.3 and the matrix M for this graph.

$$M = \begin{bmatrix} 0 & \frac{1}{8} & \frac{3}{8} & \frac{1}{2} \\ \frac{3}{16} & \frac{3}{4} & 0 & \frac{1}{16} \\ \frac{9}{16} & 0 & 0 & \frac{7}{16} \\ \frac{1}{4} & \frac{1}{8} & \frac{5}{8} & 0 \end{bmatrix}. \tag{B.10}$$

To verify we have maintained our restrictions we see all of the rows and columns in M sum to 1. As such, M is described as a doubly stochastic matrix.

As before, if we wish to determine the state (probabilities) at the next time step we get

$$Y = MX = \begin{bmatrix} 0 & \frac{1}{8} & \frac{3}{8} & \frac{1}{2} \\ \frac{3}{16} & \frac{3}{4} & 0 & \frac{1}{16} \\ \frac{9}{16} & 0 & 0 & \frac{7}{16} \\ \frac{1}{4} & \frac{1}{8} & \frac{5}{8} & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{14} \\ \frac{3}{7} \\ \frac{2}{7} \\ \frac{3}{14} \end{bmatrix} = \begin{bmatrix} \frac{3}{56} + \frac{3}{28} + \frac{3}{28} \\ \frac{3}{224} + \frac{9}{28} + \frac{3}{224} \\ \frac{9}{224} + \frac{3}{32} \\ \frac{1}{56} + \frac{3}{56} + \frac{5}{28} \end{bmatrix} = \begin{bmatrix} \frac{15}{56} \\ \frac{39}{112} \\ \frac{15}{112} \\ \frac{1}{4} \end{bmatrix}. \tag{B.11}$$

Maintaining the requirement that the coin is at one of the four positions, the sum of Y 's vector entries is 1. Again we can generalise this to the state in n time steps by taking $Y = M^n X$ where X is the current state.

In general, quantum systems are in a probabilistic state which is manipulated via matrix multiplication as we have been doing here. To bring us closer to our goal we will create a probabilistic model to represent the most important experiment in quantum mechanics: the double slit experiment.

Instead of moving from stack to stack, our coins will now be pushed through two gaps in a wall and land in one of five target areas as shown in Figure B.4. It is safe to assume that the person pushing the coins is skilled enough to always get them through one of the gaps in the wall and to land on a target.

We will also assume that coins are equally likely to go through either gap in the wall, and from there, land on either of the three targets in front of each hole. Namely this is t_0, t_1 or t_2 from the top gap (we will call g_0), and t_2, t_3 or t_4 from the bottom gap (g_1). This means that the central target

³To simplify calculations we will use fractions to represent probabilities; however any real number may be used.

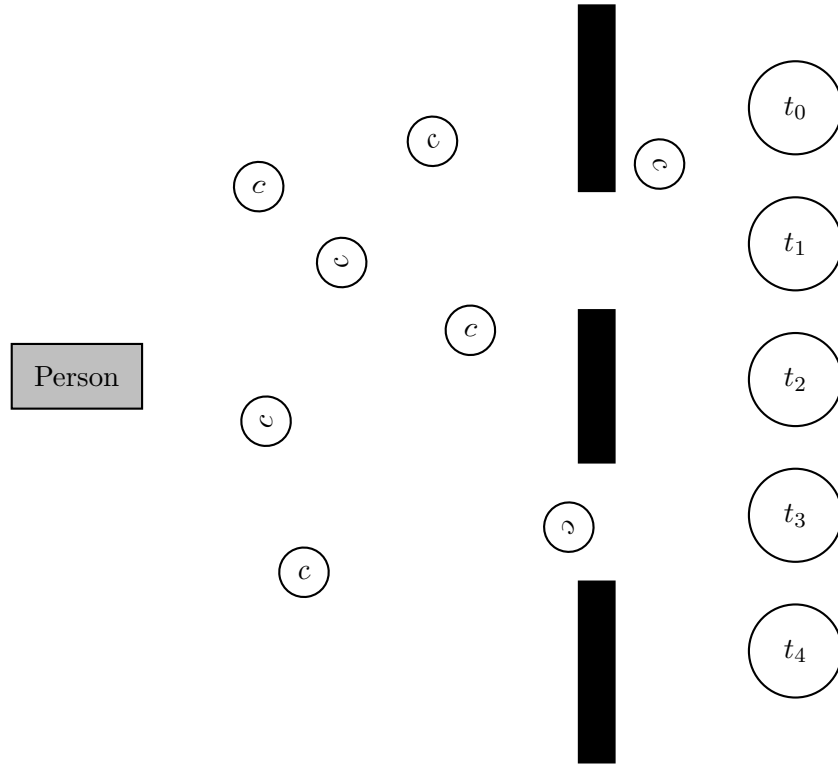


Figure B.4: Probabilistic double slit coin push model

t_2 can be reached from either g_0 or g_1 . From this information we can construct a graph (Figure B.5) stating the probabilities of reaching either gap, and from each gap, the probabilities of reaching each target. In order to easily create our dynamics matrix we will take the Person as point 0, g_0 as 1, g_1 as 2, and t_0 to t_4 as 3 to 7 respectively.

As before, our dynamics matrix, M , is made up of the probabilities for moving from one point to another, as shown in Figure B.5:

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{B.12}$$

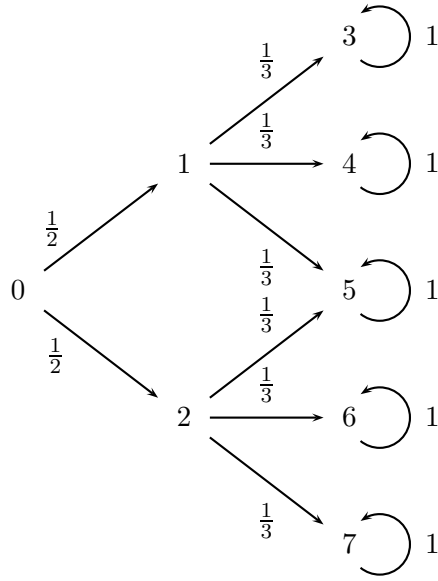


Figure B.5: Dynamics of probabilistic double slit coin push model

As before we can see where the coins would move after two time clicks:

$$M^2 = MM = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{6} & \frac{1}{3} & 0 & 1 & 0 & 0 & 0 & 0 \\ \frac{1}{6} & \frac{1}{3} & 0 & 0 & 1 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 1 & 0 & 0 \\ \frac{1}{6} & 0 & \frac{1}{3} & 0 & 0 & 0 & 1 & 0 \\ \frac{1}{6} & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (\text{B.13})$$

An important thing to notice here is that entry $M^2[5,0] = \frac{1}{6} + \frac{1}{6} = \frac{1}{3}$ as point 5 can be reached from either gap (points 1 and 2).

We can now follow the state of a single coin which we are sure starts at 0, represented by state $X = [1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]^T$. After one time click it is in state

$$Y_1 = MX = [0 \ \frac{1}{2} \ \frac{1}{2} \ 0 \ 0 \ 0 \ 0 \ 0]^T, \quad (\text{B.14})$$

or two time clicks it is in state

$$Y_2 = M^2X = [0 \ 0 \ 0 \ \frac{1}{6} \ \frac{1}{6} \ \frac{1}{3} \ \frac{1}{6} \ \frac{1}{6}]^T. \quad (\text{B.15})$$

B.3 Invertibility and Reversibility

With an understanding of the classical deterministic and probabilistic models we will take a step into the quantum world. We will return to the models we have already outlined so the reader is able to see the important differences. Modelling quantum systems requires us to move away from real numbers, and to utilise complex numbers.

This presents us with the core of quantum theory: probabilities are real numbers between 0 and 1, whereas with complex numbers, c , we use their modulus squared, $|c|^2$, as the probability, as this

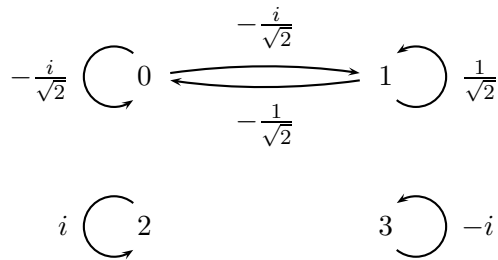


Figure B.6: Dynamics of complex number coin stacks model

is always a real number. Adding two real number probabilities together can only ever equal a larger probability: $p_1 \leq (p_1 + p_2) \geq p_2$. However, adding two complex numbers together can cancel each other out, and in fact produce a lower probability: $|c_1 + c_2|^2$ is not always bigger than $|c_1|^2$. We can see that taking $c_1 = 6 + 2i$ and $c_2 = -5 - i$ gives $|c_1|^2 = 40$, $|c_2|^2 = 26$ and $|c_1 + c_2|^2 = |1 + i|^2 = 2$, so in this case it is clear that $|c_1|^2 > |c_1 + c_2|^2 < |c_2|^2$.

In quantum mechanics, the scenario we have just described is called interference; one complex number may interfere with another. We will return to this important phenomenon in B.4.

We will return to our previous model and generalise our states and dynamics to work with our new complex universe. First we must modify the requirement that the sum of entries in each column is 1. As we are now using the modulus squared of each entry as the probability, we require that the sum of entries modulus squared in each column is 1. We must also modify our graphs to be weighted with complex numbers, and our adjacency matrix to be unitary instead of doubly stochastic.⁴

Our first example will modify the probabilistic coin stacks model presented in B.2.2. Figure B.6 shows the dynamics of the new complex system.

We create our unitary matrix made up of the values in our graph:⁵

$$M = \begin{bmatrix} -\frac{i}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \\ -\frac{i}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & 0 & 0 & -i \end{bmatrix}. \quad (\text{B.16})$$

Taking the modulus squared of the entries in M we have the matrix of probabilities

$$N = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (\text{B.17})$$

which we can see is doubly stochastic.

We can return to the unitary matrix definition introduced in A.8.9.2 and show our model satisfies this. Firstly let us find M^\dagger and I :

$$M^\dagger = \begin{bmatrix} \frac{i}{\sqrt{2}} & \frac{i}{\sqrt{2}} & 0 & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & -i & 0 \\ 0 & 0 & 0 & i \end{bmatrix} \quad (\text{B.18})$$

⁴For a reminder of unitary matrices see A.8.9.2. We defined a unitary matrix as $UU^\dagger = U^\dagger U = I$.

⁵We have chosen simple values here to make our example clearer.

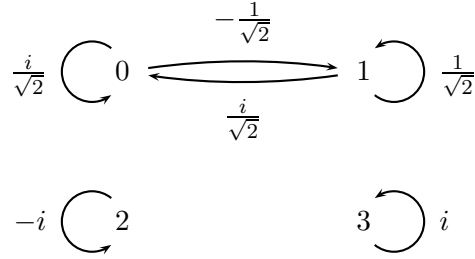


Figure B.7: Reverse dynamics of complex number coin stacks model

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (\text{B.19})$$

We wish to show

$$MM^\dagger = M^\dagger M = I \quad (\text{B.20})$$

$$\begin{aligned} MM^\dagger &= M^\dagger M = \begin{bmatrix} -\frac{i}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 \\ -\frac{i}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & 0 & 0 & -i \end{bmatrix} \begin{bmatrix} \frac{i}{\sqrt{2}} & \frac{i}{\sqrt{2}} & 0 & 0 \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & -i & 0 \\ 0 & 0 & 0 & i \end{bmatrix} \\ &= \begin{bmatrix} \frac{-i}{\sqrt{2}} \times \frac{i}{\sqrt{2}} + \frac{-1}{\sqrt{2}} \times \frac{-1}{\sqrt{2}} & \frac{-i}{\sqrt{2}} \times \frac{i}{\sqrt{2}} + \frac{-1}{\sqrt{2}} \times \frac{1}{\sqrt{2}} & 0 & 0 \\ \frac{-i}{\sqrt{2}} \times \frac{i}{\sqrt{2}} + \frac{1}{\sqrt{2}} \times \frac{-1}{\sqrt{2}} & \frac{-i}{\sqrt{2}} \times \frac{i}{\sqrt{2}} + \frac{1}{\sqrt{2}} \times \frac{1}{\sqrt{2}} & 0 & 0 \\ 0 & 0 & i \times -i & 0 \\ 0 & 0 & 0 & -i \times i \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{2} + \frac{1}{2} & \frac{1}{2} - \frac{1}{2} & 0 & 0 \\ \frac{1}{2} - \frac{1}{2} & \frac{1}{2} + \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = I. \quad (\text{B.21}) \end{aligned}$$

As we know that multiplication of our state, X , by M represents one time step forwards, and multiplying by I maintains the previous state ($XI = X$), we can see that when we combine (multiply) M , the step forwards, with M^\dagger , the outcome is no change in state: $MM^\dagger = I$. From this we can deduce that M^\dagger must be the exact inverse of M , so when the state is multiplied by M^\dagger it must represent one time step backwards. From this we see the invertibility property of quantum mechanics. We will see how useful such a straightforward reverse (undo) operation is later on. Many quantum dynamics are invertible, with one obvious exception to this being measurement.

Out of curiosity, let us look at the graphical representations of M^\dagger and I . Figure B.7 shows M^\dagger , and when compared to M , we can see each arrow is reversed in direction and the weights are now conjugated: $M^\dagger[j, k] = \overline{M[k, j]}$. If we take a state, X , and multiply it by M , then multiply it by M^\dagger , our new state will be equal to X with a probability of 1. As for I , we see in Figure B.8 this gives a probability of 1 to all coins staying in the stacks they are currently in.

B.4 Interference and Superposition

Our models thus far have been based in the macro world. To see examples of quantum mechanical phenomena we must instead model the quantum world. We will revisit our double slit experiment,

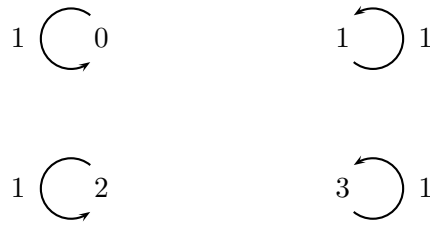


Figure B.8: Identity matrix dynamics of complex number coin stacks model

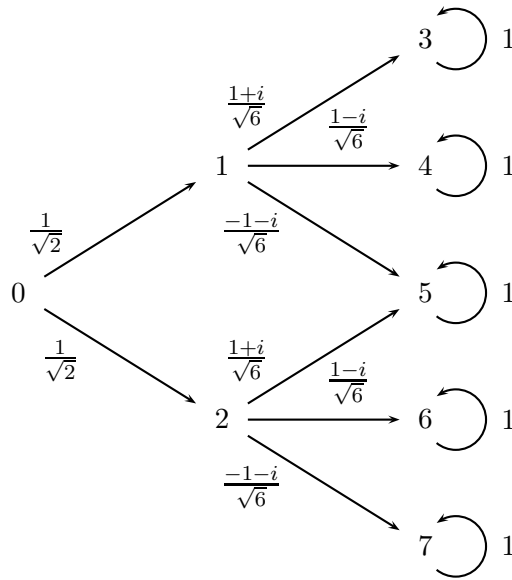


Figure B.9: Dynamics of photon double slit model

however we will replace our coins bound by the laws of classical physics with the photons of light bound by those of quantum physics. Instead of a person pushing coins, we will model a laser firing photons through our two (much smaller) slits.

With our new model we will maintain our previous assumptions that a photon will always make its way through one of the slits to one of three targets per gap, with the same, even probabilities as previously. The physical positioning and dimensions of the various objects are irrelevant for our purposes; however, we will assume the basic physical setup as shown in Figure B.4.

In Figure B.9 we define our complex number dynamics. As previously, our probabilities are calculated by calculating the modulus squared, $|c|^2$, of our complex number, c . One requirement we will drop for this model is that of M being an entirely unitary matrix. A true model of our experiment would create a graph with a huge numbers of arrows on, including those from right to left. As the intention of our model is to solely demonstrate interference, we do not require a unitary matrix. In the same way our choice of complex numbers is simply such that their modulus squared equals the correct probability, and no specific selection was made.

Placing this into an adjacency matrix we have

$$M = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1+i}{\sqrt{6}} & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{1-i}{\sqrt{6}} & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & \frac{-1-i}{\sqrt{6}} & \frac{1+i}{\sqrt{6}} & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1-i}{\sqrt{6}} & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{-1-i}{\sqrt{6}} & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (\text{B.22})$$

From this we generate our matrix of probabilities from the modulus squared of entries,

$$N = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad (\text{B.23})$$

which we can see is in fact identical to our probabilistic double slit coin push dynamics in Equation B.12. As with the coins we will create a matrix M^2 representing the dynamics of two time clicks:

$$M^2 = MM = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1+i}{\sqrt{12}} & \frac{1+i}{\sqrt{6}} & 0 & 1 & 0 & 0 & 0 & 0 \\ \frac{1-i}{\sqrt{12}} & \frac{1-i}{\sqrt{6}} & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & \frac{-1-i}{\sqrt{6}} & \frac{1+i}{\sqrt{6}} & 0 & 0 & 1 & 0 & 0 \\ \frac{1-i}{\sqrt{12}} & 0 & \frac{1-i}{\sqrt{6}} & 0 & 0 & 0 & 1 & 0 \\ \frac{-1-i}{\sqrt{12}} & 0 & \frac{-1-i}{\sqrt{6}} & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad (\text{B.24})$$

and from this, again create a matrix of normalised entries (probabilities):

$$N^2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{6} & \frac{1}{3} & 0 & 1 & 0 & 0 & 0 & 0 \\ \frac{1}{6} & \frac{1}{3} & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 1 & 0 & 0 \\ \frac{1}{6} & 0 & \frac{1}{3} & 0 & 0 & 0 & 1 & 0 \\ \frac{1}{6} & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \quad (\text{B.25})$$

Our intuition here would tell us that the probabilities we have taken from M^2 should be the same as those we had found in the probabilistic double slit coin push dynamics for two clicks, M_p^2 , as in Equation B.13. On comparison we have one glaring difference. In our probabilistic model we have $M_p^2[5, 0] = \frac{1}{3}$ where both targets could be reached from either gap. In our quantum model our result is $N^2[5, 0] = 0$. Here we have an example of interference.

We calculate

$$\begin{aligned} N^2[5, 0] &= \left(\frac{1}{\sqrt{2}} \times \frac{-1-i}{\sqrt{6}} \right) + \left(\frac{1}{\sqrt{2}} \times \frac{1+i}{\sqrt{6}} \right) = \frac{1}{\sqrt{2}} \left(\frac{-1-i}{\sqrt{6}} + \frac{1+i}{\sqrt{6}} \right) \\ &= \frac{-1-i}{\sqrt{12}} + \frac{1+i}{\sqrt{12}} = \frac{0}{\sqrt{12}} = 0. \end{aligned} \tag{B.26}$$

Here we return to an idea presented at the start of B.3. Where we would expect real number probabilities to create a larger probability, our complex numbers have interfered with each other such that their sum comes to 0. In words, although there are two routes for the photon to travel from the laser (point 0) to the third target (point 5), the probability of a photon being there is 0.

The explanation that quantum physics provides for this is that the photons act as waves. Similar to ripples formed on the surface of water, sometimes photons interfere with each other such that they may reinforce each other, or, as in this case, they are able to cancel each other out. The reason why the double slit experiment is so important in quantum physics is that it clearly demonstrates photons of light behaving as waves, whereas previously they were thought to act as particles.

Expanding on this further, if we complete the experiment with only one photon we will witness exactly the same dynamics. Our intuition would be to assume interference cannot occur with only one photon as there is no other photon (wave) to interfere with. To explain this we reveal a big difference between the probabilistic and quantum models. Instead of understanding that there is a certain probability, specified by the system state, that the photon is in a specific point, we instead must think of the photon as being at all points in the system at once. It is only when we measure the position of the photon that we discover the point at which it is situated. Therefore the probabilities offered by the system state only apply when we take a measurement of the system.

In our case, the photon will travel simultaneously through both the top and bottom slits, and where it travels through to the other side, it is then when it can cancel itself out through interference. The idea that a photon is not simply at a single point, but is in fact at many different points at the same time is described by quantum mechanics as superposition.

This may be the point at which quantum mechanics leaves readers slightly doubtful of its correctness, as it is not possible for us to ever see (measure) this behaviour as it happens. As we have said, as soon as we measure the position of the photon it is only ever in one place. We can never observe this superposition, as, by the action of measurement, the observable collapses to a single, classical state; the photon is seen in one position. Our photon sometimes behaves as a particle, and sometimes as a wave, depending on how it is being observed. This concept is termed wave-particle duality.

The field of quantum physics has no explanation of this phenomena, simply the description of it. Fortunately we will not require any further understanding or explanation of it, and the reader must simply accept that this is the behaviour we shall be emulating. As we refer back to quantum computing, it is the concept of superposition that presents much of the power, intrigue, and excitement of the subject. Classical computers are always in one state, at all times. Quantum computers allow us to be in any number of classical states and, as such, complete computations across all states like no classical computer can.

B.5 Composite Systems

The next obvious expansion to our understanding is to discuss combining multiple systems together. We will provide an explanation of how we will assemble our models such that we can exploit the concept of superposition. As described previously, we represent multiple states and multiple dynamics

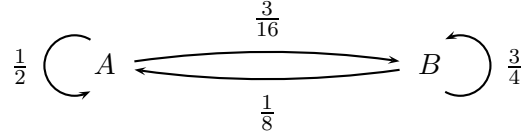


Figure B.10: Dynamics of probabilistic coin stacks model - silver coins

by using the tensor product. For simplicity we will describe this using our probabilistic coin stacks model; however everything we discuss can be equally applied to quantum models.

For our example we will take two different coloured coins, gold and silver. We will take our previously defined probabilistic dynamics for the gold coins. We define the start states X_G and X_S for gold coins and silver coins respectively. Our dynamics are shown in M_G (as previously in Figure B.3) for gold, and M_S (Figure B.10).

$$X_G = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{4} \\ \frac{1}{4} \\ 0 \end{bmatrix}, \quad X_S = \begin{bmatrix} \frac{1}{4} \\ \frac{3}{4} \end{bmatrix} \quad (\text{B.27})$$

$$M_G = \begin{bmatrix} 0 & \frac{1}{8} & \frac{3}{8} & \frac{1}{2} \\ \frac{3}{16} & \frac{3}{4} & 0 & \frac{1}{16} \\ \frac{9}{16} & 0 & 0 & \frac{7}{16} \\ \frac{1}{4} & \frac{1}{8} & \frac{5}{8} & 0 \end{bmatrix}, \quad M_S = \begin{bmatrix} \frac{1}{4} & \frac{3}{4} \\ \frac{3}{4} & \frac{1}{4} \end{bmatrix}. \quad (\text{B.28})$$

Our task is to combine both states and both sets of dynamics. We will first take the tensor products of our states X_G and X_S to describe the state of the combined system:

$$\begin{aligned} X &= X_G \otimes X_S \\ X &= \begin{bmatrix} \frac{1}{2} \\ \frac{1}{4} \\ \frac{1}{4} \\ 0 \end{bmatrix} \otimes \begin{bmatrix} \frac{1}{4} \\ \frac{3}{4} \end{bmatrix} = \begin{bmatrix} \frac{1}{8} \\ \frac{3}{8} \\ \frac{1}{16} \\ \frac{3}{16} \\ \frac{1}{16} \\ \frac{3}{16} \\ 0 \\ 0 \end{bmatrix}. \end{aligned} \quad (\text{B.29})$$

Secondly we wish to combine our dynamics so take the tensor product of these:

$$\begin{aligned} M &= M_G \otimes M_S \\ M &= \begin{bmatrix} 0 & \frac{1}{8} & \frac{3}{8} & \frac{1}{2} \\ \frac{3}{16} & \frac{3}{4} & 0 & \frac{1}{16} \\ \frac{9}{16} & 0 & 0 & \frac{7}{16} \\ \frac{1}{4} & \frac{1}{8} & \frac{5}{8} & 0 \end{bmatrix} \otimes \begin{bmatrix} \frac{1}{4} & \frac{3}{4} \\ \frac{3}{4} & \frac{1}{4} \end{bmatrix} \end{aligned}$$

$$= \begin{bmatrix} 0 & 0 & \frac{1}{32} & \frac{3}{32} & \frac{3}{32} & \frac{9}{32} & \frac{1}{8} & \frac{3}{8} \\ 0 & 0 & \frac{3}{32} & \frac{1}{32} & \frac{9}{32} & \frac{3}{32} & \frac{3}{8} & \frac{1}{8} \\ \frac{3}{64} & \frac{9}{64} & \frac{3}{16} & \frac{9}{16} & 0 & 0 & \frac{1}{64} & \frac{3}{64} \\ \frac{9}{64} & \frac{3}{64} & \frac{9}{16} & \frac{3}{16} & 0 & 0 & \frac{3}{64} & \frac{1}{64} \\ \frac{9}{64} & \frac{27}{64} & 0 & 0 & 0 & 0 & \frac{7}{64} & \frac{21}{64} \\ \frac{27}{64} & \frac{9}{64} & 0 & 0 & 0 & 0 & \frac{21}{64} & \frac{7}{64} \\ \frac{1}{16} & \frac{3}{16} & \frac{1}{32} & \frac{3}{32} & \frac{5}{32} & \frac{15}{32} & 0 & 0 \\ \frac{3}{16} & \frac{1}{16} & \frac{3}{32} & \frac{1}{32} & \frac{15}{32} & \frac{5}{32} & 0 & 0 \end{bmatrix}. \quad (\text{B.30})$$

As a side note, we must be careful to combine states and dynamics in the same way as the tensor product is not commutative, and so in general $A \otimes B \neq B \otimes A$.

We should next take a moment to describe what our new state, X , and dynamics, M , represent. The state $0A$ will describe the gold coin at point 0 and the silver coin at point A . The dynamic $0A \rightarrow 1B$ describes the gold coin moving from point 0 to 1, and the silver coin moving from point A to B . With this we have the probability entries in the state and dynamics matrices described by

$$X_{state} = \begin{bmatrix} 0A \\ 0B \\ 1A \\ 1B \\ 2A \\ 2B \\ 3A \\ 3B \end{bmatrix} \quad (\text{B.31})$$

$$M_{state} = \begin{bmatrix} 0A \rightarrow 0A & 0B \rightarrow 0A & 1A \rightarrow 0A & 1B \rightarrow 0A & \dots \\ 0A \rightarrow 0B & 0B \rightarrow 0B & 1A \rightarrow 0B & 1B \rightarrow 0B & \dots \\ 0A \rightarrow 1A & 0B \rightarrow 1A & 1A \rightarrow 1A & 1B \rightarrow 1A & \dots \\ 0A \rightarrow 1B & 0B \rightarrow 1B & 1A \rightarrow 1B & 1B \rightarrow 1B & \dots \\ 0A \rightarrow 2A & 0B \rightarrow 2A & 1A \rightarrow 2A & 1B \rightarrow 2A & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}. \quad (\text{B.32})$$

It is at this point that one of the main challenges of creating a simulation of a quantum computer becomes apparent. Where we had two states of size 4 and 2, we now have the new state represented by a vector of size $2 \times 4 = 8$. Our dynamics of size 4 by 4 and 2 by 2, combined create an 8 by 8 matrix. We move from having a state represented by 6 vector entries to 8, and dynamics from 20 matrix entries to 64.

If we generalise this growth, we start with two systems with n points and m points respectively. Our composite system will have $n \times m$ points. If we consider the adjacency matrix for this combined graph, it will be n^m by n^m in size, with n^{2m} entries. The growth we see is exponential. Considering this problem with a view to simulate quantum systems, as we incorporate more systems to our simulation, the demands on resources required to represent the state and dynamics grow rapidly.

B.6 Quantum Mechanics and Ket Notation

B.6.1 Quantum States

When described as an n dimensional quantum system, we are saying that system may be observed in any one of n possible states. Although the world is a continuous physical system, computers cannot

deal with infinities, and, as such, we must turn our systems into a discrete one.

If we look at a potential quantum system we may consider the different positions a particle may be in: one of n positions. As previously, we would represent this state (position) as a column vector of dimension n . The standard notation for describing quantum states used in quantum mechanics is ket notation (from bra-ket notation $\langle | \rangle$):

$$|\psi\rangle = \begin{bmatrix} c_0 & c_1 & c_2 & \dots & c_{n-1} \end{bmatrix}^T. \quad (\text{B.33})$$

Where we have a single entry 1 we define a pure state where we are certain the particle is in position 1:

$$|\psi\rangle = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \end{bmatrix}. \quad (\text{B.34})$$

To determine the length of $|\psi\rangle$ we normalise (modulus squared) each entry:

$$L = |c_0|^2 + |c_1|^2 + |c_2|^2 + \dots + |c_{n-1}|^2. \quad (\text{B.35})$$

Our assumptions of the state are the same as described in the previous subsections. We find the probability of a particle being in position j , if measured, to be $|c_j|^2$. One thing we must be aware of is that the entries in our length vector, L , may not sum to 1 as we will encounter non-normalised vectors.⁶ As such, we must modify our probability for j to

$$\frac{|c_j|^2}{L}. \quad (\text{B.36})$$

Where the system is not in a pure state, the state described by our ket notation is a superposition of states. If the particle were to be measured, our state vector describes the probability that it would be in each of the possible positions. Before being measured, the particle is in a superposition of the possible positions; it is in all positions at once.

As we interact with multiple superpositions, we must look at how to perform operations upon them. Given two superpositions $|\psi\rangle = \begin{bmatrix} c_0 & c_1 & c_2 & \dots \end{bmatrix}^T$ and $|\phi\rangle = \begin{bmatrix} d_0 & d_1 & d_2 & \dots \end{bmatrix}^T$, we can add them together as such:

$$|\psi\rangle + |\phi\rangle = \begin{bmatrix} c_0 + d_0 & c_1 + d_1 & c_2 + d_2 & \dots & c_{n-1} + d_{n-1} \end{bmatrix}^T. \quad (\text{B.37})$$

Multiplying a state by some complex number $z \in \mathbb{C}$ gives

$$z|\psi\rangle = \begin{bmatrix} z \times c_0 & z \times c_1 & z \times c_2 & \dots & z \times c_{n-1} \end{bmatrix}^T. \quad (\text{B.38})$$

We will look at an interesting property of the states we have defined. Adding a superposition to itself, which can also be considered as multiplying by an integer, gives

$$|\psi\rangle + |\psi\rangle = 2|\psi\rangle = \begin{bmatrix} 2c_0 & 2c_1 & 2c_2 & \dots & 2c_{n-1} \end{bmatrix}^T. \quad (\text{B.39})$$

Finding the length of this state, the sum of the modulus squared of each entry, gives

$$\begin{aligned} N &= |2c_0|^2 + |2c_1|^2 + |2c_2|^2 + \dots + |2c_{n-1}|^2 \\ &= 2^2(|c_0|^2 + |c_1|^2 + |c_2|^2 + \dots + |c_{n-1}|^2) = 2^2L, \end{aligned} \quad (\text{B.40})$$

which we use to calculate the probability a particle is found in position j as per Equation B.36:

$$\frac{|2c_j|^2}{N} = \frac{2^2|c_j|^2}{N} = \frac{2^2|c_j|^2}{2^2(|c_0|^2 + |c_1|^2 + |c_2|^2 + \dots + |c_{n-1}|^2)}$$

⁶Our use of the word normalised here implies a vector of length 1. The quantum systems we will describe will always be normalised, so we include this purely for completeness.

$$\begin{aligned}
 &= \frac{|c_j|^2}{|c_0|^2 + |c_1|^2 + |c_2|^2 + \dots + |c_{n-1}|^2} \\
 &= \frac{|c_j|^2}{L}.
 \end{aligned} \tag{B.41}$$

We see our result is that we find identical probabilities that a particle is found at point j in both systems $|\psi\rangle$ and $2|\psi\rangle$; we have described the same system. Generalising this, we can say that for any complex number, $z \in \mathbb{C}$, $|\psi\rangle$ and $z|\psi\rangle$ describe the same system state.

By adding a division to our probability definition in Equation B.36 we have removed the length as a differentiating factor between states. As we have shown, the direction is the only important factor in defining the state of a system. States with the same direction and different lengths describe the same system. We can now work with non-normalised vectors. To normalise a vector $|\psi\rangle$, we take

$$\frac{|\psi\rangle}{\| |\psi\rangle \|}. \tag{B.42}$$

The last thing we shall mention on our newly found notation is on how to depict composite systems. There are a number of ways to show combined states using ket notation:

$$|\psi\rangle \otimes |\phi\rangle = |\psi\rangle|\phi\rangle = |\psi, \phi\rangle = |\psi\phi\rangle. \tag{B.43}$$

States in $\mathbb{C}^n \otimes \mathbb{C}^m$ which we are unable to easily represent as $|\psi\rangle \in \mathbb{C}^n$ and $|\phi\rangle \in \mathbb{C}^m$ are described as entangled, another phenomenon of quantum physics.

B.6.2 Dynamics

We will briefly describe the role of ket notation in our discussion of dynamics. As we have shown, dynamics of a quantum system are defined by n by n unitary adjacency matrices. Given a state $|\psi\rangle$ defining the system at time t and a unitary matrix M representing the system dynamics, the state at time $t + 1$ is provided by

$$M|\psi\rangle. \tag{B.44}$$

B.6.3 Observables and Measurement

As we have described earlier, when we take a measurement of an observable, the state of the system collapses down to a pure state.⁷ At this point the state is no longer a superposition. We shall represent this in ket notation as

$$|\psi\rangle \Rightarrow |\varphi\rangle \tag{B.45}$$

where $|\psi\rangle = [c_0 \ c_1 \ c_2 \ \dots]^T$ and $|\varphi\rangle = [0 \ 1 \ 0 \ \dots]^T$.⁸

The pure state to which the system collapses to is determined probabilistically as we have described in B.6.1. The observable (measurement) is represented by an n by n Hermitian matrix.⁹ For a matrix representing an observable of an n dimensional system, there are n distinct eigenvectors and eigenvalues.

B.7 Further Reading

What we have presented only scrapes the surface of the expansive field of quantum mechanics. The reader should be armed with enough knowledge for our purposes; however there is a substantial amount of literature available to learn more. One of the key texts in this field is [18], but a good introduction can also be found in [30].

⁷As described in B.6.1, a pure state refers to a column vector containing a single entry 1, with the rest of the vector made up of 0 entries. In this state we are sure that the measurement of the system will result in a specific outcome.

⁸Here we use the notation \Rightarrow to imply the collapse of a superposition state to a pure state.

⁹For a reminder of Hermitian matrices see A.8.9.1.

Appendix C

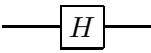

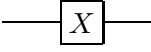
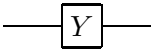
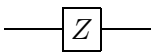
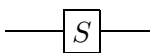
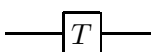
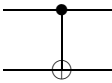
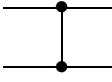
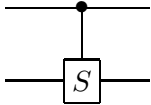
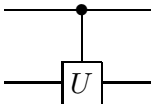
Symbols

We make use of a number of symbols in this report related to quantum circuits, measurement patterns, and quantum mechanics in general.

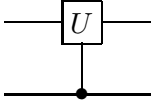
C.1 Mathematics and Physics

Transpose	A^T	See A.8.7.
Conjugate	\bar{A}	See A.4.5.
Adjoint	A^\dagger	See A.8.9.
Tensor product	$U \otimes V$	See A.8.12.
XOR / modulo 2	$x \oplus y$	
Normalisation / modulus squared	$ c ^2$	See A.4.6.
Bra-ket	$\langle \phi \psi \rangle$	See B.6
Ket	$ \psi\rangle$	
Bra	$\langle \phi $	
Inner product	$ \psi\rangle \cdot \langle \phi $	

C.2 Quantum Gates and Quantum Circuits

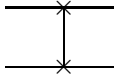
Hadamard / H		$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Identity / I		$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
Pauli-X / NOT / X		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y / Y		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z / Z		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
phase / $P(\frac{\pi}{2})$ / S		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$P(\frac{\pi}{8})$ / T		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix}$
controlled- X / controlled-NOT / CX		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
controlled- Z / CZ		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$
controlled-phase / CS		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & i \end{bmatrix}$
controlled- U / CU		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & c & d \end{bmatrix}$ $U = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$

swapped controlled- U / UC



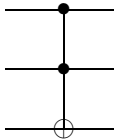
$$U = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & a & 0 & b \\ 0 & 0 & 1 & 0 \\ 0 & c & 0 & d \end{bmatrix}$$

swap



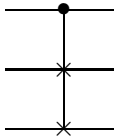
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Toffoli



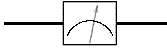
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Fredkin



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

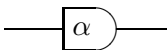
measurement in standard basis



$$M_0 = |0\rangle \cdot \langle 0| = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$M_1 = |1\rangle \cdot \langle 1| = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

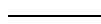
measurement in non-standard basis



$$M_{+\alpha} = |+\alpha\rangle \cdot \langle +\alpha| = \frac{1}{2} \begin{bmatrix} 1 & e^{i\alpha} \\ e^{i\alpha} & e^{2i\alpha} \end{bmatrix}$$

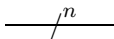
$$M_{-\alpha} = |-\alpha\rangle \cdot \langle -\alpha| = \frac{1}{2} \begin{bmatrix} 1 & -e^{i\alpha} \\ -e^{i\alpha} & e^{2i\alpha} \end{bmatrix}$$

qubit



$$|\psi\rangle = a|0\rangle + b|1\rangle = \begin{bmatrix} a \\ b \end{bmatrix}$$

n qubits



$$|\psi_1\rangle|\psi_2\rangle \dots |\psi_n\rangle$$

For the diagrams of quantum circuits in this report we have used a modified version of Isaac Chuang's qasm2circ package [9].