MSc Individual Project

Imperial College London
Department of Computing

Q#: A Quantum Programming Language
by Microsoft

Author: Ahmed, Talha

Supervisor: Dr Mario Berta

Submitted in partial fulfillment of the requirements for the MSc degree in Computing Science of Imperial College London

September 7, 2018
Abstract

The multi-paradigm quantum programming language Q# was analysed and used to study and create novel programs that were able to go beyond the capabilities of any classical program. This language was built by Microsoft to succeed LIQUi|> and is a Domain-Specific Language (DSL) that can be used within Microsoft’s Quantum Development Kit (QDK). The quantum programs are run on a quantum simulator which possesses properties that a real quantum computer would from a behavioural aspect. It uses the .NET Code SDK, allowing for easy creation, building and running of quantum projects via the command line.

Initially, the main features and libraries available were studied and experimented with by analysing implementations of the Quantum Teleportation and Deutsch-Jozsa algorithms. Thereafter, an algorithm to solve an arbitrary $2 \times 2$ matrix system of linear equations was implemented, demonstrating a theoretical quantum speed-up with time complexities of order $O(\log N)$ compared to the classical $O(N)$ for sparse matrices. Running the algorithm for a particular matrix achieved results that were within the range of theoretical predictions.

Subsequently, as an extension to the project, concepts within Quantum Game Theory were explored. This led to the Mermin-Peres Magic Square game successfully being simulated in Q#; a game where no classical winning strategy exists, yet a quantum strategy is able to win in all possible cases of the game.
# Contents

1 Introduction .................................................. 1  
1.1 History of Quantum Mechanics ............................. 1  
1.2 Aims and Objectives of Project ............................. 3  
2 Background ................................................... 5  
2.1 Quantum Mechanics ......................................... 5  
   2.1.1 Wave-Particle Duality .................................. 6  
   2.1.2 Superposition and the Schrödinger Equation ........... 8  
   2.1.3 The Uncertainty Principle ............................. 10  
   2.1.4 Quantum Entanglement ................................ 11  
2.2 Quantum Computing ......................................... 12  
   2.2.1 Qubits .................................................. 12  
   2.2.2 Quantum Gates and Circuits ........................... 14  
   2.2.3 The No-Cloning Theorem ................................ 20  
2.3 Quantum Hardware Implementations ......................... 21  
   2.3.1 Ion Traps ................................................. 22  
   2.3.2 Niobium Loops ......................................... 23  
3 Quantum Algorithms .......................................... 26  
3.1 The Q# Programming Language .............................. 26  
   3.1.1 Primitive Types and Available Operations ............ 26  
   3.1.2 File Structure .......................................... 28  
3.2 Quantum Teleportation ...................................... 30  
   3.2.1 Algorithm Outline ...................................... 30  
   3.2.2 Q# Implementation .................................... 32  
3.3 Deutsch-Jozsa Algorithm .................................... 34  
   3.3.1 Algorithm Outline ...................................... 35  
   3.3.2 Q# Implementation .................................... 37  
3.4 Solving Linear Equations .................................... 40  
   3.4.1 Algorithm Outline ...................................... 41  
   3.4.2 Q# Implementation .................................... 43  
   3.4.3 Applications in Machine Learning ...................... 46  
4 Quantum Game Theory .......................................... 47  
4.1 What is Game Theory? ...................................... 47
### Table of Contents

4.1.1 Prisoner's Dilemma .......................... 47  
4.1.2 Bayesian Games ............................. 48  
4.2 The Mermin-Peres Game ....................... 49  
  4.2.1 The Classical Magic Square Game .......... 49  
  4.2.2 Quantum Pseudo-Telepathy ................. 51  
4.3 Q# Implementation ............................ 54  
  4.3.1 Building the Game Board Circuits .......... 54  
  4.3.2 Simulating and Testing the Game .......... 59  
4.4 Exploring Other Concepts ...................... 66  

5 Conclusion ............................. 67  
  5.1 Outcomes of Project ....................... 67  
  5.2 Future Improvements ...................... 67  

6 Ethical Considerations ...................... 69  

Bibliography .................................. 70  

A Ethics Checklist .............................. I  

B Q# Code for Linear Equation Solver .......... IV  

C Q# Code for Mermin-Peres Circuits ............ VII
Chapter 1

Introduction

1.1 History of Quantum Mechanics

The realm of Quantum Mechanics has been puzzling Physicists over the last decade, displaying results that are very counter-intuitive for even the brightest of minds to comprehend. Emerging as a result of earlier discoveries, including that of the photoelectric effect demonstrated by Heinrich Hertz in 1887 and the electron through cathode rays by J.J. Thomson in 1897 [1], Quantum Physics led to a revolution in the scientific world. In 1900, Max Planck hypothesized that atomic systems which radiate energy consist of discrete "energy levels", whereby the differences between these energy elements are proportional to the frequency with which each of them radiate energy, resulting in Planck’s famous constant, $h (= 6.63 \times 10^{-34} Js)$ [2]. This discreteness seems very profound, due to the fact that in the classical universe we observe physical units of measure, like distance and time, on a continuous scale. However, classical physics also leads to many absurd theories, such as the existence of an 'ultraviolet catastrophe' involving infinite energies, and electrons spiralling into an atomic nucleus (something which is never observed in stable atoms). More sound theories describing what is actually observed can be formulated by considering the consequences of this discreteness.

In 1905 Albert Einstein postulated that light could be represented as individual particles in the form of "energy packets" in order to explain the photoelectric effect [3] - with the now universally accepted name of "photon" proposed by Gilbert N. Lewis [4]. This phenomenon, whereby a current was observed in a circuit when certain frequencies of light were shone on a metal piece, was explained by the fact that electrons were ejected from the metal surface only if the incident photons had energy greater than a given threshold known as the work function (illustrated in Figure 1.1). The discovery awarded Einstein with his first Nobel Prize in Physics in 1921. Thereafter, the foundations of Quantum Mechanics were born in the early 1920s by the likes of Max Born, Werner Heisenberg, Wolfgang Pauli and many others [5].
Although Quantum Physics was initially used to explain phenomena in particle and atomic physics, its concepts and mathematical formalisms are beginning to be applied to a wide range of different fields. Of particular interest is Quantum Computing, whereby concepts such as superposition and entanglement can be utilised to create systems with immense computational capabilities that would otherwise not be possible with current hardware [7] [8]. For example, a quantum algorithm known as Shor’s Algorithm makes use of Quantum Fourier Transforms (QFTs) to perform integer factorisation in polynomial $\log N$ time complexities, as opposed to classical ones which take sub-exponential time [9]. The power of such an algorithm would allow an attacker to cause severe havoc on current public-key cryptosystems which takes advantage of the "factoring problem" associated with the product of two large prime numbers [10]. Much research is being conducted in the field of Quantum Cryptography to issue a new era of security across large networks, one that will be far superior to the current framework.

Practical limitations such as the ability to physically store and transmit quantum bits (qubits will be explained in Section 2.2.1) on quantum hardware means that we are still a few years away from fully embracing quantum computation. However, to unlock the groundbreaking promises of quantum technologies, not only quantum hardware but also quantum software will be required. Various programming languages and quantum simulators have already been open-sourced online, including, but not limited to, imperative and functional languages, as well as compilers. This project mainly focused on exploring the first multi-paradigm quantum programming language Q#, created by Microsoft as a successor to its functional quantum language LIQUi\textsuperscript{|> [11] [12].}

This report will begin by exploring the fundamental concepts of Quantum Mechanics. We will then apply these principles to devise basic Quantum Computing operations that can be used to manipulate data stored on such devices. Moreover, the various physical implementations of quantum computers currently being tested will
be brought to light, as well as the quantum programming facilities currently available. Thereafter, a few quantum algorithms will be studied along with their Q# implementations; including a few simulations which haven’t been performed in the language previously.

1.2 Aims and Objectives of Project

The main aims and objectives that were laid out for this project are as follows:

1. **Conducting Background Research**: The initial stages of the project involved doing thorough background reading into Quantum Computing itself. Topics such as quantum gates and circuits, and ultimately quantum algorithms were studied as these were vital for being able to later program using a quantum programming language. How such concepts can effectively be utilised to solve current issues in computing were explored.

2. **Learning Q# Programming Language**: After understanding the main concepts of Quantum Computing, this task saw how they were translated to Microsoft’s new programming language. The syntax rules as well as the various data types and structures available were studied and experimented with. Thereafter, readily available quantum algorithm implementations provided in Microsoft’s Quantum GitHub repository [13] were studied and simulated to gain a deeper level of appreciation for how processes are run within the language framework as well as judge what its limitations are.

3. **Comparing With Another Language**: Due to a shift in focus of the project, it was decided that it would prove more beneficial to study one language (Q#) instead of multiple. This is because focusing on a single language would allow for it to be learnt to a greater level of depth. Although comparing different features of multiple languages can help to assess the benefits and drawbacks of each, the amount of experimentation within each would be limited.

4. **Implementing New Example**: This goal was indeed achieved. A new quantum algorithm example which currently doesn’t exist on Microsoft’s Quantum GitHub repository was implemented in Q#. Namely, the algorithm is one which can be used to solve linear equations at time complexities that are significantly better than the current classical counterparts of the algorithm. Details outlining the key principles of this, as well as the Q# implementation, will be presented in Section 3.4. Moreover, we will see how such an algorithm is being applied to real world examples by researchers today.

The initial specifications also proposed that if all of these main objectives were met, then the project may go on to explore other concepts/implementations. In particular, the two choices were Quantum Game Theory or Creating a Quantum Game. It was decided that the former would be explored, and thus Chapter 4 is dedicated
1.2. AIMS AND OBJECTIVES OF PROJECT

Chapter 1. Introduction

to Quantum Game Theory, presenting key ideas and how a particular example was simulated in Q#.
Chapter 2

Background

Before delving into the quantum algorithms that were explored in this project, an understanding of the core principles of Quantum Mechanics needs to be obtained. This chapter gives an overview of this, before discussing how such principles can be applied in Quantum Computing. Finally, we end the chapter by briefly touching on how such machines are currently being created.

2.1 Quantum Mechanics

Quantum Mechanics is the study of physical interactions between light and matter at very small scales [2]. The field arose in the late 19th century when many observations by scientists at a microscopic level couldn't be explained using the knowledge of classical physics at the time. For example, electrons were found to occupy regions in a probabilistic manner as opposed to having a definite position in space and time, as would be expected for a macroscopic object such as a ball [14]. As a result, many new theories were devised that would explain the very counter-intuitive discoveries made. This eventually led to the creation and advancement of many new scientific fields. For instance, Solid State Physics studies how particles interact within crystals known as semiconductors, enabling pioneers to take full advantage by creating devices such as the smartphones of today [15].

As previously explained, atomic systems were found to consist of discrete energy levels by Max Planck. Such discreteness can also be witnessed in many other properties of small-scale objects, such as position and speed. The name commonly used by physicists to describe this is "quantisation of matter", and is where the name "Quantum Mechanics" arises from.
2.1.1 Wave-Particle Duality

Wave-particle duality refers to the fact that matter, such as electrons (or even light), can be described as being both a wave and a particle [16]. Einstein’s work on the photoelectric effect led to him postulating that light could be described as particles in the form of photon energy packets. He proposed that the energy, $E$, of these photons was proportional to the frequency, $f$, of the light as

$$E = hf = \frac{hc}{\lambda}$$  \hspace{1cm} (2.1)

where $h$ is Planck’s constant, and the speed of light equivalence $c = f\lambda$ was invoked to relate the energy to the wavelength, $\lambda$ [3]. However, later in the same year he published a paper introducing the world to the concept of special relativity, within which he treated light as a field of waves [17]. The clear contradiction in the nature of light between his two theories emphasised that he was ready to accept the strange concept of wave-particle duality.

The modern interpretation of light accepts this wave-particle duality, describing it as a collection of multiple photons which propagate through space as electromagnetic waves [18]. Moreover, it applies to all types of particles; matter was first observed to display wave-like properties when a beam of electrons passing through a slit was found to diffract (the bending of waves when passing through an aperture) - something which was previously thought to only occur with light. This demonstration was made by the scientist Louis de Broglie, who had predicted that matter behaved as waves, relating an object’s momentum, $p$, to its wavelength as

$$p = \frac{h}{\lambda_{dB}}$$  \hspace{1cm} (2.2)

where $\lambda_{dB}$ is the de Broglie wavelength of the object [19].

A very famous experiment which clearly demonstrates wave-particle duality is the double-slit experiment. This experiment was first carried out by Thomas Young in 1801 to describe the wave theory of light before the concept of wave-particle duality was first conceived [20]. However, the experiment has now been shown to also work for particles.

The set-up for the experiment involves a wall with two equally sized slits, through which light from a single source can be shined from one side. The other side of the wall has a screen a certain distance away, so that the incident light can be observed once it has traversed through the slits. The pattern that is observed on the screen is striped, usually referred to as an interference pattern [21]. This can be explained by considering the light to act as a wave. The wave from the singular source splits into two new waves when passing through the slits. This is illustrated in Figure 2.1 (left), where the blue lines represent the peaks of the wave (wave-fronts). These waves spread outwards and interfere with each other when they meet, causing constructive interference where a peak from each wave intersects and destructive interference where a peak from one meets a trough of the other, cancelling each other
out. Thus, the striped pattern on the screen is a result of the constructive parts of the interfering waves.

\[\text{Figure 2.1: Illustration of the double-slit experiment for light (left) and electrons (right). The blue lines represent wave-fronts, while the blue dots depict individual electron particles [21].}\]

Now, a similar experiment can be done using single electrons instead of light. By initially blocking one of the slits and firing single electrons at a time, it would be expected that a single stripe would form on the screen resembling the slit the electrons passed through. This is indeed what is observed. However, quantum effects begin to emerge when both slits are opened. One would expect this to build up two stripes on the screen as more electrons are fired through. Contrary to this, the screen in fact displays an interference pattern similar to the one seen with light, as depicted in Figure 2.1 (right). Figure 2.2 displays snapshots of a screen as more electrons are fired through the slits from image \(b\) to \(e\).

\[\text{Figure 2.2: A view of a screen as more electrons are fired in a double-slit experiment from image } b \text{ to } e. \text{ Each dot represents the measurement of a single electron. It can be seen that as more electrons are fired, a distinct interference pattern emerges [21].}\]

One possible explanation for this occurrence is that the electrons may be interfering with each other as they pass through the slit. However, this cannot be the case
as they are fired individually at the slits. It is as if the electron begins to act as a wave after it's fired, passing through both slits simultaneously, interfering with itself, then collapsing back to a single point when viewed on the screen. This experiment indicates that wave-particle duality is also observed in matter.

Interestingly, the experiment breaks down when a detector is placed at one of the slits, as the output on the screen produces two single stripes instead of the interference pattern. It is as if the actual acting of measuring the electron is causing it to collapse from its wave-like nature to its particle form [22]. This leads us onto the discussion of superposition and the measurement problem.

### 2.1.2 Superposition and the Schrödinger Equation

A wavefunction is a continuous mathematical function describing the state of a quantum system, such as a single particle or even a larger collection of particles [23]. Usually denoted by the Greek letter $\psi$, it encompasses all measurable information about the system and may also be a complex function. A very fundamental property of the wavefunction is that it may consist of linear combinations of basic configurations (also known as eigenstates) that the system can be in. This leads to the concept of superposition - multiple quantum states can be added together to create a new valid quantum state [24]. Ultimately, what this means is that a system can exist in multiple states at the same time.

Furthermore, the mere act of measuring the state of the system results in just one of the eigenstates. This is known as the measurement problem, which is a very fundamental concept in Quantum Mechanics - measuring the state of a quantum system results in its wavefunction collapsing into a single state in a probabilistic manner, and it is unknown how these linear combination of states transform into a single, well-defined outcome. Mathematically, the wavefunction after a collapse is usually denoted by a Dirac-Delta function which geometrically represents a single, vertical spike of unit volume. Figure 2.3 illustrates this concept by showing how the wavefunction of a particle in two dimensions, initially spread over a large region on the $x - y$ plane, collapses to a single set of coordinates after a measurement. Physically, this means that the probability of the particle being at a particular location was initially spread over a large region, but this reduced to a single, well-defined location after measurement as its position is now known with complete certainty. This interpretation of the wavefunction arises from what is known as the Copenhagen Interpretation of Quantum Mechanics [25].

In three spatial dimensions, the wavefunction consists of two variables, $r = (x, y, z)$ and time, $t$, such that $\psi(r, t)$ gives the probability amplitude for finding a particle at a given location and time [27]. This is because each space and time coordinate represents a particular state, with its own amplitude within the wavefunction. Multiplying this amplitude by its complex conjugate, $\psi^*(r, t)$, and integrating within a region with lower bound $r_1$ and upper bound $r_2$ gives the probability of measuring
Chapter 2. Background

2.1. QUANTUM MECHANICS

Figure 2.3: Illustration of the wavefunction collapse mechanism in the Copenhagen Interpretation of Quantum Mechanics. The $x \times y$ plane represents the spatial position of a particle, while the vertical axis depicts its wavefunction, $\psi$. It can be seen that the initial large spread of the wavefunction collapses to a single Dirac-Delta function after measurement [26].

The particle in this particular region:

$$\text{Prob}_{r_1<r<r_2}(t) = \int_{r_1}^{r_2} \psi^*(r,t)\psi(r,t) \, dr = \int_{r_1}^{r_2} |\psi(r,t)|^2 \, dr \quad (2.3)$$

where the definition $\psi^*(r,t)\psi(r,t) = |\psi(r,t)|^2$ has been used, and the integration is over all spatial dimensions.

Another property of the wavefunction is that it normalised over all states [27]. This means that the amplitudes across all states in the superposition must be such that their probabilities sum to unity over all space - the particle must be found somewhere. Mathematically, this can be represented as an integral over all space as

$$\int_{-\infty}^{\infty} |\psi(r,t)|^2 \, dr = 1 \quad (2.4)$$

The Schrödinger Equation

Newton’s second law of motion has played a pivotal role in classical mechanics. Describing the evolution of a physical system subject to external forces, the second-order differential equation $F = m \ddot{x}$ (where $\ddot{x} = (\ddot{x}, \ddot{y}, \ddot{z})$ is the second-order time derivative of the three Cartesian coordinates) can be solved to find not only an object’s position, but also its momentum, effectively being able to describe the entire state of the classical system at any given time [28].

Erwin Schrödinger devised the quantum analogue of this law in 1925, giving an equation that is able to describe the evolution of quantum systems over time [23].

This equation laid the foundations for a mathematical formalism that would go on to illustrate the dynamics of many complicated quantum systems. Now known as the Schrödinger equation, it enables one to analytically predict the distribution of probabilistic outcomes.

The Time Dependent Schrödinger Equation (TDSE) applied to a wavefunction $\psi(r, t)$ is given by

$$i\hbar \frac{\partial \psi(r, t)}{\partial t} = \hat{H}\psi(r, t)$$

(2.5)

where $\hbar = h/2\pi$ is the reduced Planck’s constant, $i = \sqrt{-1}$ is the imaginary unit and $\hat{H}$ is the Hamiltonian operator [23]. The Hamiltonian operator usually encompasses the total kinetic and potential energies of the quantum system. For a single particle of mass $m$, this is given by

$$i\hbar \frac{\partial \psi(r, t)}{\partial t} = \left[ -\frac{\hbar^2}{2m} \nabla^2 + V(r, t) \right] \psi(r, t)$$

(2.6)

where $\nabla^2$ is the Laplacian such that $-\frac{\hbar^2}{2m}\nabla^2\psi(r, t)$ gives the kinetic energy of the particle, and $V(r, t)$ is the potential experienced by the particle. Equation (2.6) is a linear partial differential equation representing the principle of conservation of energy. It can be solved for the wavefunction so that further information can be deduced about the state of the system, as well as how it may evolve over time.

The Time Independent Schrödinger Equation (TISE) can be used to describe the orbitals of atoms, by interpreting the wavefunctions that solve the equation as stationary states [23]. This equation is given by

$$E\psi(r, t) = \left[ -\frac{\hbar^2}{2m} \nabla^2 + V(r, t) \right] \psi(r, t)$$

(2.7)

where $E$ is the total energy of the quantum system. Such an equation is known as an eigenvalue equation, as the operations applied to the wavefunction on the RHS result in the same wavefunction multiplied by some constant factor, $E$, on the LHS, known as the eigenvalue. The wavefunctions that solve this equation are then referred to as eigenstates. The significance of this result is that any wavefunction can be represented as a linear combination of energy eigenstates, as these are also solutions to the Schrödinger equation.

### 2.1.3 The Uncertainty Principle

The uncertainty principle was originally devised by Werner Heisenberg in 1927, postulating that, at a quantum level, particular pairs of physical properties of a system cannot both be known with exact certainty [29]. For instance, if the position of a particle is measured with uncertainty $\Delta x$ and its momentum with uncertainty $\Delta p$, then it must hold that

$$\Delta x \Delta p \geq \frac{\hbar}{2}$$

(2.8)
In other words, if the position of the particle is known very precisely (i.e. its position uncertainty $\Delta x$ is very small), then its momentum cannot be known very well and must have a high uncertainty in $\Delta p$ such that equation (2.8) holds.

Classically, one can determine the position and momentum of a moving object with reasonable ease provided the right measuring apparatus is used. This seems to contradict what Heisenberg’s uncertainty principle states in equation (2.8). However, on a large scale the de Broglie wavelength as defined by equation (2.2) is very small for everyday objects, meaning that we don’t experience matter behaving as a wave. Moreover, the order of magnitude for the uncertainty levels dealt with on a macro level are far greater than that of Planck’s constant, but still small enough that they are negligible compared to the actual measured values of the properties.

On the other hand, the significantly smaller values that are dealt with at a quantum level make the consequences of the uncertainty principle considerably more impactful to the measurements made - the uncertainties have orders of magnitude very similar to the actual measurements being made. This is due to the fact that we cannot determine the property of a quantum system without interacting with it in one way or another, which introduces an unavoidable level of uncertainty in the measurement result such that all properties are impossible to measure exactly.

The Heisenberg Microscope Experiment

An interesting example which demonstrates this key principle is the Heisenberg microscope experiment [30]. This involves firing photons at an electron in order to determine the electron’s position and momentum. The incident photons are reflected off the electron via a process called Compton Scattering, and the reflection is observed on a screen using a microscope [31]. It can be shown that the lowest uncertainty in the electron’s position is achieved by using light with a short wavelength. A short wavelength implies that the photons have a large energy by equation (2.1) - they give a large ‘push’ to the electron when scattering off it. However, the side effect is that the electron recoils with a much larger momentum than before, making it difficult to accurately determine its momentum. Hence, one cannot determine both properties of the electron with exact certainty in this experiment.

2.1.4 Quantum Entanglement

Quantum entanglement occurs when multiple entities that have either been created from a single source or have interacted in a particular way share a common quantum state [32]. This means that their state cannot be decomposed into independent quantum states of the individual entities in the system. For example, two particles could be created with opposite spins (an intrinsic property of a particle related to its angular momentum, which can be oriented at different angles) from a single source with total spin zero, such that when one particle is measured along an axis to be in the spin up (+1) orientation, the other will always be in the opposite direction
(−1) when measured along the same axis. This ensures that angular momentum is conserved before and after the creation - their measurements are anti-correlated. Naturally, such a process of creating these particles could occur when a cosmic-ray particle decays.

Interestingly, when these particles are moved a large distance apart and one of them is measured to be in a particular state, the other still appears to always collapse into the opposite state along the same axis; it is as if each particle can identify the measurement made on its counterpart with no form of communication. Such a paradoxical phenomenon was the cause of much frustration to Einstein, Boris Podolsky and Nathan Rosen, who suggested that faster-than-light communication was impossible and therefore the view of Quantum Mechanics at the time needed further refinement. They suggested in a paper in 1935 that the entangled state contained fixed, hidden variables at the point of creation which already pre-defined the outcomes of the measurements such that they were always anti-correlated [32]. However, it was later found that such a formalism breaks down when spin measurements along varying axis are considered, violating what is known as Bell’s inequality (named after John Bell) [33]. Ultimately, this means that entanglement is inherently a quantum mechanical process.

As will be discussed in Chapter 3, two quantum bits can be prepared into a single entangled state known as a Bell State or EPR Pair. Such a state is widely used in quantum computing algorithms as a means of communicating information between several parties. A few examples of how this is done will be outlined in the chapters to follow.

## 2.2 Quantum Computing

One of the more recent applications of Quantum Mechanics has been in the field of Computer Science, where the counter-intuitive phenomena observed are being applied to various computational tasks. Quantum Computing has started to gain a lot more attraction outside of the laboratory, as not only the public, but also private corporations are beginning to invest into a future quantum era, seeking to enhance the security of their systems whilst also staying one step ahead of their competitors [34]. We will now discuss the main components which are used to build quantum computers before moving onto some of the current hardware implementations that are being researched and tested.

### 2.2.1 Qubits

Classically, the bit is the most fundamental form for storing information on a computer. No matter the program or language being used, all data is converted into binary to be processed by the hardware. These bits can only be in two distinct states - either 0 or 1 [35].
Chapter 2. Background

2.2. QUANTUM COMPUTING

Analogous to bits, qubits (short for quantum bit) are the fundamental building blocks of quantum computers [36]. These can also exist in states that correspond to those of the classical bit; namely $|0\rangle$ and $|1\rangle$, which in vector form are represented as

$$|0\rangle \equiv \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle \equiv \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

respectively [37]. The ket notation ($'|\rangle$) being used was originally invented by theoretical physicist Paul Dirac, and so is more commonly known as Dirac Notation [24]. These two states are referred to as computational basis states as they form an orthonormal basis within a vector space. This gives rise to a unique property of qubits that classical bits don’t possess - a qubit can be in any linear combination of states, $|\psi\rangle$, typically called superpositions. Mathematically, this can be represented as

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$$

(2.10)

where $\alpha$ and $\beta$ are the complex number amplitudes of the two states. A key property of this superposition state is that one cannot determine the values of these coefficients, as any measurement of its wavefunction (a mathematical function describing the state of a quantum system [23], as in equation (2.10)), $\psi$, results in it collapsing into the 0 state with probability $|\alpha|^2$, and likewise state 1 with probability $|\beta|^2$. These probabilities must sum to unity: $|\alpha|^2 + |\beta|^2 = 1$ [37]. Thus, the act of measuring a qubit results in it only giving either a 0 or 1 state in a probabilistic nature. It is this inherent superposition property of qubits that gives rise to many new possibilities for computations that would otherwise not be feasible classically.

Physically, the existence of such entities has been validated through various experiments over the years, and can be realised in many different forms. For example, an atomic model can be used whereby an electron orbiting a central nucleus can exist in either a ‘ground’ state ($|0\rangle$) or an ‘excited’ state ($|1\rangle$). Incident light on the atom of the correct frequency can cause the electron to shift from the ground state to the excited one, while the emission of light can result in it dropping back down to the ground state. However, the length of time for which the light is shined on the atom can be further limited to cause the electron to be in an intermediate state between $|0\rangle$ and $|1\rangle$. This represents the superposition state that qubits can exist in, as mathematically formalised in equation (2.10).

Multiple Qubits

Multiple qubits can also be illustrated using similar notation to that of a single qubit. Consider two qubits: this system would be comprised of four computational basis states, $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$, where the first and second digits represent the states of the first and second qubits respectively [37]. These states arise because of the fact that there are four corresponding states for two classical bits.

As before, the system can also exist in a superposition of these basis states, given by

$$|\psi\rangle = \alpha_{00} |00\rangle + \alpha_{01} |01\rangle + \alpha_{10} |10\rangle + \alpha_{11} |11\rangle$$

(2.11)
where the probability of state $|xy\rangle$ is $|\alpha_{xy}|^2$, and the sum over all probabilities is again unity.

### 2.2.2 Quantum Gates and Circuits

At a hardware level, classical computers are constructed using electrical circuits which mainly consist of wires to send bits across channels and logic gates to manipulate bit values. Analogous to this, a quantum computer can be constructed using quantum circuits, consisting of wires to carry qubits and quantum gates to convert the quantum information from one form to another. A few simple gates and circuits will be discussed in this section. These can be combined in different combinations via wires to create quantum circuits that can perform a wide variety of tasks.

#### Single Qubit Gates

For classical circuits, the single bit gate is the NOT gate. This simply flips the value of the bit from 0 to 1 or vice versa. A similar quantum gate which is not only able to convert a $|0\rangle$ state to a $|1\rangle$ and vice versa, but can also act on superpositions (i.e. linear combinations) of these two states in the form $\alpha |0\rangle + \beta |1\rangle$ is the quantum NOT gate [37]. This gate can be represented as a matrix, usually denoted by $X$, and is given by

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

(2.12)

This gate is able to convert a superposition state of a single qubit in the form $\alpha |0\rangle + \beta |1\rangle$ to $\beta |0\rangle + \alpha |1\rangle$, by simply interchanging the states $|0\rangle$ and $|1\rangle$. This can be seen by representing the state $\alpha |0\rangle + \beta |1\rangle$ in vector notation as

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

(2.13)

such that applying the $X$ gates gives the output

$$X \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}$$

(2.14)

The naming convention for this gate was chosen due to the fact that the matrix resembles that of the Pauli-X matrix used by physicists, and likewise for the Pauli-Y and Pauli-Z quantum gate representations.

As mentioned previously, a normalisation condition must hold for the wavefunction of the qubit. For example, the state given in equation (2.10) must obey $|\alpha|^2 + |\beta|^2 = 1$. This should remain true after the qubit has been passed through a quantum gate, say $U$, to form $|\psi\rangle = \alpha' |0\rangle + \beta' |1\rangle$ i.e. $|\alpha'|^2 + |\beta'|^2 = 1$. It can be shown that a sufficient property for this to hold is that the gate matrix must be unitary, such that

$$U^\dagger U = I$$

(2.15)
where \( I \) is the identity matrix and \( U^\dagger = (U^T)^* \) [38]. This property does indeed hold for the \( X \) gate and so is a unitary gate.

There are also many other single qubit gates that exist. For instance, the \( Z \) (Pauli-Z) gate changes the sign on the \(|1\rangle\) state while keeping the \(|0\rangle\) state intact, and is represented as
\[
Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}
\] (2.16)

Yet another gate is the Hadamard, \( H \), given by
\[
H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}
\] (2.17)

which is sometimes referred to as the ‘square-root of NOT’ gate, as it changes the \(|0\rangle\) to \((|0\rangle + |1\rangle)/\sqrt{2}\) and the \(|0\rangle\) to \((|0\rangle - |1\rangle)/\sqrt{2}\). This particular gate turns out to be extremely useful in many quantum algorithms, including quantum teleportation which will be discussed in the next section.

Figure 2.4 shows schematic diagrams of these three gates, illustrating how they would be represented on a typical quantum circuit.

\[
\begin{align*}
\alpha|0\rangle + \beta|1\rangle & \xrightarrow[\text{Time evolution}]{X} \beta|0\rangle + \alpha|1\rangle \\
\alpha|0\rangle + \beta|1\rangle & \xrightarrow[\text{Time evolution}]{Z} \alpha|0\rangle - \beta|1\rangle \\
\alpha|0\rangle + \beta|1\rangle & \xrightarrow[\text{Time evolution}]{H} \frac{\alpha|0\rangle + |1\rangle}{\sqrt{2}} + \frac{\beta|0\rangle - |1\rangle}{\sqrt{2}}
\end{align*}
\]

Quantum single qubit gates.

**Figure 2.4:** Schematic diagrams demonstrating the operations of the single qubit gates \( X, Z \) and \( H \) [39].

**Multi-qubit Gates**

An important multi-qubit gate is the controlled-NOT gate, also known as CNOT. It takes two input qubits: a control qubit and a target qubit. This gate is such that the target qubit is only flipped if the control is set to \(|1\rangle\). This action can be concisely written as
\[
|A, B\rangle \rightarrow |A, A \oplus B\rangle
\] (2.18)
where $A$ and $B$ are the control and target qubits respectively, and $\oplus$ is addition modulo two. The gate is reminiscent of the classical XOR gate, as the output of the target qubit is simply the two states XOR’d. For all possible permutations of input, the outputs are as follows:

- $|00\rangle \rightarrow |00\rangle$
- $|01\rangle \rightarrow |01\rangle$
- $|10\rangle \rightarrow |11\rangle$
- $|11\rangle \rightarrow |10\rangle$

Such a gate can be represented using two wires: the top for the control qubit $A$, and the lower for the target $B$. Figure 2.5 illustrates the CNOT gate in this form.

![CNOT gate diagram](image)

**Figure 2.5**: Schematic diagram illustrating the operation of the CNOT gate. Two input wires and are used [37].

The gate can also be represented in a unitary matrix fashion as

$$U_{\text{CN}} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0
\end{bmatrix} \quad (2.19)$$

An important property of unitary quantum gates is that they can always be inverted, due to the fact that the inverse of such a matrix is also unitary. This is in contrast to the irreversible nature of classical gates such as XOR or NAND gates. In these gates, the values of the two inputs cannot be determined given just the output, resulting in a loss of information.

An interesting aspect to note is that all other quantum gates can be constructed using a combination of the CNOT and single qubit gates. Thus, they encompass a universality set together, similar to how a NAND gate can be used as a prototype for all other classical gates [37].

For instance, one can create a two-qubit SWAP gate by simply using a combination of CNOT gates. This operation can be used to switch the states of two qubit registers, as the name suggests. Figure 2.6 illustrates this, where the top and bottom wire represent the first and second qubits respectively.

Each operation of the CNOT gate can be summarised as

$$|A, B\rangle \rightarrow |A, A \oplus B\rangle \quad (2.20)$$
2.2. QUANTUM COMPUTING

Figure 2.6: Schematic diagram illustrating the how a SWAP gate can be created from 3 CNOT gates. This gate simply switches the states of two qubits [37].

\[ \rightarrow |A \oplus (A \oplus B), A \oplus B \rangle = |B, A \oplus B \rangle \quad (2.21) \]
\[ \rightarrow |B, (A \oplus B) \oplus B \rangle = |B, A \rangle \quad (2.22) \]

where \( A \) and \( B \) are the top and bottom qubits respectively. Also, note that that property \( A \oplus A \equiv 0 \) of addition modulo two has been utilised to reduce some of the expressions in the intermediate stages.

In matrix form, the SWAP gate can be represented as

\[ U_{SWAP} = U_{CN_{AB}} U_{CN_{BA}} U_{CN_{AB}} \quad (2.23) \]

where the matrix \( U_{CN_{AB}} \) is simply that given in equation (2.19), while the matrix \( U_{CN_{BA}} \) represents a CNOT gate where \( B \) is the control qubit instead of \( A \), given by

\[ U_{CN_{BA}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \quad (2.24) \]

Thus, by multiplying equation (2.23) out, we find that

\[ U_{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.25) \]

Rotational Gates

An interesting and useful visualisation of a qubit state is in a spherical coordinates representation. Namely, two angles - a polar angle, \( \theta \), and an azimuthal angle, \( \varphi \) - can be used to define a point on a unit sphere, commonly referred to as a Bloch Sphere. Figure 2.7 shows this representation on a Cartesian axis. It can be seen that the positive \( z \) axis represents the \( |0\rangle \) qubit state, while the negative direction is the \( |1\rangle \) state.

Mathematically, this qubit state can be represented as

\[ |\psi\rangle = \cos \left( \frac{\theta}{2} \right) |0\rangle + e^{i\varphi} \sin \left( \frac{\theta}{2} \right) |1\rangle \quad (2.26) \]
Figure 2.7: A qubit can be represented on a Bloch Sphere using spherical polar coordinates [37]. The positive and negative $z$ axes represent the $|0\rangle$ and $|1\rangle$ qubit states respectively [37].

It can easily be verified that this state is normalised, as the amplitudes are such that $|\cos\left(\frac{\theta}{2}\right)|^2 + |e^{i\phi}\sin\left(\frac{\theta}{2}\right)|^2 = 1$. Such a representation can be very useful for quantum computations, as it allows a clear visualisation of how a qubit state is being manipulated along the sphere when different operations are performed.

In particular, there exist three primary rotation matrices, known as the Pauli rotational operators, that can be used to rotate a qubit state along a given axis of the Bloch Sphere [37]. Each matrix, $R_n(\theta)$, takes an angle parameter to specify the amount of rotation to be performed. These operators are laid out below in their matrix forms, as well as how they can be decomposed into an identity matrix, $I$, and their respective Pauli matrices.

$$R_x(\theta) = e^{-iX/2} = \cos\left(\frac{\theta}{2}\right) I - i \sin\left(\frac{\theta}{2}\right) X = \begin{bmatrix} \cos\frac{\theta}{2} & -i \sin\frac{\theta}{2} \\ -i \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}$$ (2.27)

$$R_y(\theta) = e^{-iY/2} = \cos\left(\frac{\theta}{2}\right) I - i \sin\left(\frac{\theta}{2}\right) Y = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix}$$ (2.28)

$$R_z(\theta) = e^{-iZ/2} = \cos\left(\frac{\theta}{2}\right) I - i \sin\left(\frac{\theta}{2}\right) Z = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}$$ (2.29)

A unique property of the Pauli matrices, $X, Y$ and $Z$, is that squaring them produces the identity matrix. Suppose we have a similar matrix, $A$, such that $A^2 = I$. Then
for any real number $x$:

$$e^{iAx} = \sum_{k=0}^{\infty} \frac{(iAx)^k}{k!}$$

$$= \sum_{\text{even } k} \frac{(ix)^k}{k!} A^k + \sum_{\text{odd } k} \frac{(ix)^k}{k!} A^k$$

$$= \sum_{\text{even } k} \frac{(ix)^k}{k!} I + \sum_{\text{odd } k} \frac{(ix)^k}{k!} A$$

$$= \cos(x) I + i \sin(x) A$$

The final equivalence was obtained by noting that the two series expansions are those of the trigonometric cosine and sine functions respectively. This derivation explains how the definitions of the rotation matrices above arise.

A fascinating property of these rotational operators is that any unitary matrix operation, $U$, can be formed by simply multiplying the Pauli-Y and -Z matrices together as

$$U = e^{i\alpha} R_z(\beta) R_y(\gamma) R_z(\delta) \quad (2.30)$$

where $\alpha, \beta, \gamma, \delta$ are all real. This is because the rows and columns of $U$ are orthonormal [37].

**Quantum Measurement**

As previously noted, the measurement of a qubit’s state results in its wavefunction collapsing into either the $|0\rangle$ or $|1\rangle$ state. This can also be represented with its own gate, as depicted in Figure 2.8. The double lines used at the output illustrate the fact that the qubit has collapsed into a probabilistic classical bit.

![Figure 2.8: Schematic diagram depicting the measurement of a qubit. The double lines represent the fact that the qubit has collapsed into a classical bit [40].](image)
2.2.3 The No-Cloning Theorem

There are also a few other key properties of quantum circuits to note that differ from classical, electrical circuits. One of them is that quantum circuits are acyclic - feedback from one section of a circuit to another cannot be done. Also, the FANIN property of regular circuits, where multiple wires meeting at a junction are combined to give a single wire with the bitwise OR of those wires, is impossible for qubit wires. This is because the operation is not reversible and therefore not unitary - we cannot determine the states of the qubits prior to meeting at a junction. Moreover, multiple copies of qubits as a result of a wire splitting into multiple wires, as is done classically, cannot be done. This is sometimes referred to as a FANOUT; the restriction is governed by the No-Cloning Theorem [37].

The No-Cloning Theorem states that a copy of a qubit cannot be created if it is in a superposition state. Classically, copying a bit is easy to achieve. One can simply use a classical CNOT gate with two inputs: the bit to copy, $x$, and a bit to store the duplicate, initialised to 0. Figure 2.9 (left) illustrates this, where the output has two copies of the $x$ bit.

\[
\begin{align*}
X \quad x \quad x \quad X \\
0 \quad y \quad x \oplus y \quad x
\end{align*}
\]

\[
\psi = a|0\rangle + b|1\rangle 
\]

\[
|0\rangle \quad a|00\rangle + b|11\rangle
\]

**Figure 2.9:** Left: Copying of a bit can easily be done using a classical CNOT gate and an extra bit to store the new copy. Right: The quantum attempt at copying a qubit state [37].

Figure 2.9 (right) shows a similar attempt at copying a qubit in the state $|\psi\rangle = a|0\rangle + b|1\rangle$ using a quantum CNOT gate with $|\psi\rangle$ as the control qubit, and $|0\rangle$ as the target qubit. The aim is to create the state $|\psi\rangle |\psi\rangle$, such that we have two copies of the original qubit. Multiplying the desired state out gives

\[
|\psi\rangle |\psi\rangle = a^2 |00\rangle + ab |01\rangle + ba |10\rangle + b^2 |11\rangle \tag{2.31}
\]

which is different to the actual state obtained, $a|00\rangle + b|11\rangle$. The only way that these two states will match is when $ab = 0$, meaning that it is possible to copy the original qubit only when it's in an eigenstate of either $|0\rangle$ or $|1\rangle$. Moreover, one can consider what happens when the 'copy' qubit measured. Measuring the state $a|00\rangle + b|11\rangle$ causes it to collapse into the state $|00\rangle$ ($|11\rangle$) with probability $|a|^2$ ($|b|^2$). This means that after measuring the 'copy', the original qubit also collapses into the same eigenstate, losing any of the hidden information contained within its previous superposition state. Thus, it is impossible to copy qubit states in general as measuring the copy shouldn't result in the original qubit also collapsing. It is this
fundamental property that lies at the heart of many quantum algorithms, especially quantum cryptosystems where the integrity, as well as the copying of data, is of significant importance.

Below is a formal proof of the No-Cloning Theorem, which highlights why in general qubit states cannot be copied [37].

**Formal Proof**

Suppose we possess two quantum data registers. The first contains an unknown pure quantum state that we wish to copy, denoted by $|\psi\rangle$. The latter is the target register within which we wish to store a copy of the first register state. This target is initialised into another pure state, $|s\rangle$. Thus, our machine is initially in the state $|\psi\rangle \otimes |s\rangle$.

We can now introduce a unitary operator, $U$, that transforms these registers into the desired state of $|\psi\rangle \otimes |\psi\rangle$. Moreover, this unitary operator is able to clone a different pure state, $|\phi\rangle$, in a similar fashion. These operations can be written as

$$U(|\psi\rangle \otimes |s\rangle) = |\psi\rangle \otimes |\psi\rangle \quad (2.32)$$
$$U(|\phi\rangle \otimes |s\rangle) = |\phi\rangle \otimes |\phi\rangle \quad (2.33)$$

To see the effect of this, the inner product of these two operations can be calculated as follows

$$\langle s | \otimes \langle \phi | U^\dagger U (|\psi\rangle \otimes |s\rangle) = (\langle \phi | \otimes \langle \phi |)(|\psi\rangle \otimes |\psi\rangle) \quad (2.34)$$
$$\langle \phi | \psi \rangle = (\langle \phi | \psi \rangle)^2 \quad (2.35)$$

where the property $U^\dagger U = I$ has been invoked. Thus, it can be seen that the only two solutions to this expression are $\langle \phi | \psi \rangle = 0$ or $\langle \phi | \psi \rangle = 1$. In other words, the two pure states being copied are either orthogonal or are identical to each other. Hence, a quantum cloner of this kind will not be able to clone states that are in different computational basis states; it is impossible to use unitary operations to clone unknown quantum states.

### 2.3 Quantum Hardware Implementations

Physical realisations of quantum computers are still in their infant stages as the major technology firms, including Microsoft, IBM and Intel, compete to create the first mass-producible quantum processing units. There are an immense number of hardware implementations that are currently being researched and tested for quantum computers. This section will briefly introduce two of the most recognised developments in the industry.
2.3.1 Ion Traps

An ion is an atom that has either gained or lost electrons so that it has an overall electrical charge that may be positive or negative. One way they can be created is by heating a metal to high temperatures inside a vacuum. The low pressure will enable the metal to evaporate into a vapour of metal atoms, which can then be ionised by firing a beam of free electrons at them, knocking out some of the orbital electrons in the outermost shell of the metal. The ions that are formed this way will have a positive charge. This process is indeed what was done by a group of researchers at Oxford University, who formed Calcium ions by heating it to approximately 800°C [41].

These ions where then captured inside of an ion trap. The ion trap consists of four electrodes that produce an electromagnetic field which interacts with the electric field produced by the ion. The electrodes are arranged in a square such that the ion can be confined within a particular axis. Figure 2.10 (top) shows this equipment, where the electrodes are highlighted in blue.

Figure 2.10 (bottom) shows the electrical potential that is experienced by an ion trapped in between the four electrodes across a two dimensional plane. Note that the red, flat regions are the electrodes. It can be seen that there is a saddle point with an increasing potential along the \( x \) axis, and a decreasing potential along the \( y \) axis. This means that the movement of the ion along the \( x \) direction is well confined, by it is unstable along the \( y \) direction. In order to keep the ion trapped at a single location, the current in the electrodes were alternated at a frequency of 6MHz such that the potential oscillated, giving the ion an average potential energy that is negligible. As the ion tends to move away from the centre of the trap, it begins to oscillate such that it falls back towards the centre.

These ions can be used as a physical manifestation of qubits. Their states can be changed by manipulating their kinetic energy. However, being able to control the amount of kinetic energy that is injected and removed from each ion requires little thermal energy i.e. energy from the surroundings that will cause the ions to move in an undesired fashion. This can be achieved by using laser cooling, whereby physical phenomena including resonance and the Doppler effect are used to reduce the (random) kinetic energy of the ions.

Thereafter, the states of the ions can be manipulated by illuminating them with controlled pulses of light of different frequencies - the ions will absorb and emit photons, representing a transition from one state to another. In order to represent multiple qubits, more ions can be added to a single trap, where each trap signifies a quantum register. Then, individual qubits can be operated on by targeting them with a laser beam that is only a few microns in diameter, smaller than the separation between ions.

So far, the experiment has been tested with three to ten qubits in a trap at a time [41]. One of the major problems this approach faces is the infeasibility of scaling it up, as the number of ion traps required for large register counts would make it very
impractical to physically build and store a quantum computer with this mechanism. Moreover, the amount of power required to constantly manipulate the qubits would reach astronomical levels.

2.3.2 Niobium Loops

D-Wave Systems is a Canadian company that is currently testing and producing quantum computing architectures, and was the first to sell a quantum computer [42]. The underlying process by which D-Wave’s quantum computers operate is through quantum annealing in order to solve optimisation problems, such as machine learning, Monte Carlo methods, financial analysis, etc [43]. Usually, optimisation problems involve reaching a global minimum of some function. Classically, this may lead to problems when starting off at an arbitrary position on the function, as one may end up in a local minimum instead of a global one. One way classical
programmers can deal with this is by introducing a random 'jump' while descending so that local minima can be escaped, however this still may not lead to the most optimal solution.

Quantum annealing seeks to remediate this issue by having a state that is in multiple starting positions along the function’s surface via the principle of superposition. Thereafter, each step of the algorithm causes the amplitudes of the locations where the function is deepest to increase such that the probability of measuring the system in one of these locations increases. Moreover, the concept of quantum tunnelling (where an entity has a small chance of traversing through a barrier) allows the state to traverse directly through local minima instead of having to climb them, while quantum entanglement can add a level of correlation between coordinates.

Physically, the qubit states are created using something known as a Superconducting QUantum Interference Device (SQUID) - a form of quantum transistor. It consists of a qubit 'loop' comprised of a metal known as Niobium, which is cooled to temperatures as low as 15 milliKelvin using liquid Helium as a coolant. This causes the loops to become superconducting such that they are able to carry current with zero electrical resistance. This current is able to flow in either direction, inducing a magnetic field representing the two basis states of the qubit. However, since the current is essentially moving electrons, these electrons can exhibit quantum effects due to their wave-like nature, allowing the current to flow in both directions of the wire simultaneously, creating a superposition state. Figure 2.11 below gives a depiction of this Niobium loop. Each qubit loop also has a Josephson junction attached, which weakly couples it to another superconductor so that the Quantum Processing Unit (QPU) can be programmed.

![Figure 2.11: Depiction of a Niobium qubit loop used by D-Wave in their SQUID quantum transistors. The current in the loop can be in a superposition of both directions simultaneously [44].](image-url)

D-Wave sees their computers as an aid to classical computers as opposed to completely replacing them. There has been a lot of controversy surrounding D-Wave's quantum hardware, as many have claimed that these computers don't truly possess
the quantum speed-up that has been promised by many theoretically laid out algorithms. Moreover, the fact that it can only solve certain, niche problems means that it isn’t an entirely universal quantum computing device. Despite this, companies like Lockheed Martin and Google seem keen to invest in these devices, suggesting that there is still a desire for ‘partially’ quantum computers [45].
Chapter 3

Quantum Algorithms

3.1 The Q# Programming Language

Q# is a multi-paradigm programming language that can be utilised to create quantum algorithms. It was built by Microsoft as a successor to LIQUi|> and is a Domain-Specific Language (DSL) that can be used within Microsoft’s Quantum Development Kit (QDK) [46]. The quantum programs are run on a quantum simulator which possesses properties that a real quantum computer would from a behavioural aspect. It uses the .NET Code SDK, allowing for easy creation, building and running of quantum projects via the command line [47].

3.1.1 Primitive Types and Available Operations

Primitive Types

The language contains the usual primitive types that can be found in most other programming languages. These are:

- **Int**: A signed, 64-bit integer.
- **Double**: A decimal or floating point number with double precision.
- **Bool**: A Boolean value which can take two values, namely false or true.
- **String**: A sequence of characters stored in a variable or as a literal.

On top of these are some new quantum primitives, listed below [48].

- **Qubit**: Stores a single qubit state. Specifying variables of this type assigns qubits in the state $|0\rangle$. Q# doesn’t allow the user direct access to the qubit’s state; they can only be manipulated using pre-defined operations.

- **Result**: The type that represents the values of measurements made on qubits. It is binary valued: Zero if the eigenvalue measured was $+1$, and One if it is $-1$.
• Pauli: Used to specify the quantum mechanical Pauli basis within which qubits are to be measured or even rotated. It can take the form of Pauli* where * is either X, Y, Z for each of the Pauli orientations, or even I for the identity version.

Access to arrays and tuples for all of the above types is also available, as well as control sequences such as if-else statements and for-loops [49].

Operations

Q# has also enabled users to create operations and functions, together called callables. Functions are subroutines which can contain classical code only, but are allowed to have qubits passed to them. On the other hand, operations allow quantum code to be written, which can perform tasks such as applying quantum gates to qubits [50].

Some of the most basic operations that can be applied to qubits are provided in a library called Microsoft.Quantum.Primitive [51]. Some of these are listed below, along with the parameters they take and the return type (after the colon) where a "()" indicates that nothing is returned.

• H (qubit : Qubit) : (): Performs the Hadamard transform to a single qubit as outlined by equation (2.17).

• X (qubit : Qubit) : (): Applies the Pauli-X gate to a single qubit. Pauli -Y and -Z versions also exist.

• Rx (theta : Double, qubit : Qubit) : (): Applies the $R_x(\theta)$ rotational gate to a single qubit at a specified angle. $R_y(\theta)$ and $R_z(\theta)$ versions also exist.

• CNOT (control : Qubit, target : Qubit) : (): Applies the controlled Pauli-X gate to two qubits. One qubit is the control, while the other is the target to be flipped.

• M (qubit : Qubit) : Result: Measures a single qubit in the Pauli-Z basis. Returns Zero if the eigenvalue is $+1$, One if it is $-1$.

• ResetAll (qubits : Qubit[]) : (): Measures an array of qubits, resetting them to the $|0\rangle$ state so they can safely be released. Using this is recommended as good practice at the end of each operation that assigns new qubits.

These are but some of the operations included in this API. Another library is the Microsoft.Quantum.Canon API, which contains many more operations and functions that are able to perform more complex calculations and manipulations [52]. These have all been built using the primitive operations. Some of these will be highlighted in the examples to come.
3.1. THE Q# PROGRAMMING LANGUAGE  Chapter 3. Quantum Algorithms

Functors

Moreover, Q# also has what are known as functors [48]. This consists of keywords that, when added within the definition of an operation, is able to automatically generate a new, variant implementation of that operation. This is especially handy when certain algorithms require more complex processing. One of these functors is Adjoint, which produces the complex conjugate and transpose, $A^\dagger$, of an operation $A$. For example, the adjoint form of the Pauli-X gate on a qubit $q$ would be invoked as $(\text{Adjoint} \ X)(q)$. Another useful functor is Controlled, which is able to create a controlled version of an operation based on specified control registers.

3.1.2 File Structure

As Microsoft mentions [53], the intended way of using the language is as a co-processor, reminiscent of how GPUs are used in Machine Learning for expensive tasks such as training. As a result, Q# projects have a Driver.cs C# file which contains the code to invoke the quantum program, stored in a .qs file. This design decision allows programmers to write classical and quantum code separately before bringing all the components together.

The template for a project can be automatically created [54]. First, the following command is used in any command line (like Bash) to download up-to-date Q# templates:

```
$ dotnet new -i "Microsoft.Quantum.ProjectTemplates::0.2.1806.3001-preview"
```

Thereafter, the command

```
$ dotnet new console -lang Q# --output HelloWorld"
```

creates a HelloWorld directory containing all files for the project with the pre-built template.

As an example to highlight this file structure, below is the code contained in the Operation.qs and Driver.cs files for a simple Hello World! project.

Operation.qs

```c
namespace HelloWorld
{
    open Microsoft.Quantum.Canon;
    open Microsoft.Quantum.Primitive;

    operation Greet (who : String) : (String) 
    {
        body
        {
```
This file defines a single operation `Greet(who : String) : (String)` which takes a parameter of type `String` and also returns a string literal. The string returned is simply "Hello, {who}!" which is formatted such that the "{who}" is replaced by the string literal passed to the operation. This is written inside a block known as the body. The reason for this is because operations may have certain properties about them defined outside of the actual processes being executed. For example, the keywords `controlled auto` may be written outside of this block, specifying that the operation has a controlled version which Q# automatically generates. Although this code is written in Q#, there are no quantum processes taking place. This is merely an example of the file structure.

Looking above the operation definition, there are a few things to note. Firstly, the use of the namespace `HelloWorld` ensures that other files in the same directory have access to this program. Below this, the preambles `open Microsoft.Quantum.*` can be seen for the Primitive and Canon libraries provided by Microsoft. This allows one to easily use these APIs by simply calling the functions and operations defined within them.

**Driver.cs**

```csharp
using Microsoft.Quantum.Simulation.Core;
using Microsoft.Quantum.Simulation.Simulators;

using System;
	namespace HelloWorld
{

class Driver
{

    static void Main(string[] args)
    {
        using (var sim = new QuantumSimulator())
        {
            var result = Greet.Run(sim, "World").Result;
            Console.WriteLine(result);
        }
    }
}
```

The classical code appears to be slightly more involved. The two `using Microsoft.Quantum.Simulation.*` statements allow this C# code to interface with the Q# code
in Operation.qs. It also uses an object-oriented structure by defining a class called Driver.

Within this is the Main function that invokes the quantum program. It does this within a using block by defining a new variable sim which is an instance of a QuantumSimulator class. This is required as Q# runs using a quantum simulator on the local machine as opposed to a real quantum computer. The operation defined in the Operation.qs file is called in the line Greet.Run(sim, "World").Result and stored in variable result. It can be seen that the simulator is passed into the operation as well as the string literal, and the final value returned by the operation is selected via its Result attribute. In fact, this same structure is used in all driver files when creating projects in this language. The final message is then printed and can be viewed in the command line.

For the remainder of this report, when code is shown, the following colour scheme will be used: blue for keywords; green for comments; red for strings; purple for API operations and functions; and cyan for reserved symbols like true or Zero.

### 3.2 Quantum Teleportation

We now have the tools to build a circuit which enables quantum teleportation - a means of transferring a qubit between a sender and receiver without the need for a quantum communications channel [37].

#### 3.2.1 Algorithm Outline

Two parties, who we will name Alice and Bob, are to communicate a qubit state \( |\psi \rangle = \alpha |0\rangle + \beta |1\rangle \) to each other. Alice will be the sender in possession of the said quantum state, while Bob will be the recipient. Before moving apart, the two create a Bell state [33]. A Bell state involves creating an entangled pair of qubits, in the form

\[
|\beta_{00}\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}}
\]  

i.e. a superposition of the \(|00\rangle\) and \(|11\rangle\) pair states. What's unique about this particular state is that after the measurement of the first qubit state, the measurement of the second qubit will always give the same result as the first - the measurements are highly correlated. Such a qubit pair is commonly referred to as an entangled pair, as this still holds even after moving the qubits light years apart. This state can be created using a combination of the Hadamard and CNOT gates on two qubits. Figure 3.1 demonstrates this.

After the creation of this Bell state pair, Alice and Bob separate such that they can only communicate via a classical communications channel. Moreover, Alice is unable to determine the state of the \( |\psi \rangle \) message that needs to be sent to Bob, due to the fact that measuring it will result in the wavefunction collapsing. Furthermore, the
laws of quantum mechanics dictate that copies of quantum states are impossible to create, as was discussed with the no-cloning theorem.

This is where quantum teleportation can be used, where Alice can take advantage of the principles of quantum mechanics to transmit the state to Bob. Firstly, an initial state is prepared in the form

$$|\psi_0\rangle = |\psi\rangle |\beta_{00}\rangle = \frac{1}{\sqrt{2}} \left[ \alpha |00\rangle + |11\rangle + \beta (|00\rangle + |11\rangle) \right]$$

where the first two qubits of each three-qubit state are possessed by Alice (the message and her qubit respectively), while the last is Bob’s. This is a combination of the message to be sent by Alice and the Bell State pair.

Subsequently, this state is passed as input into the quantum circuit outlined in Figure 3.2 below, where the top line represents the message to be sent, the middle Alice’s entangled qubit $|A\rangle$, and the lower being Bob’s entangled qubit $|B\rangle$.

It can be seen that Alice’s qubit has been entangled with Bob’s by the waved line between their wires [37].
3.2. QUANTUM TELEPORTATION

Chapter 3. Quantum Algorithms

gate, acquiring the state $|\psi_1\rangle$ given by

$$|\psi_1\rangle = \frac{1}{\sqrt{2}} \left[ \alpha |0\rangle (|00\rangle + |11\rangle) + \beta |1\rangle (|10\rangle + |01\rangle) \right]$$  \hspace{1cm} (3.3)

Thereafter, the message qubit is send through a Hadamard gate, resulting in

$$|\psi_2\rangle = \frac{1}{2} \left[ \alpha (|0\rangle + |1\rangle)(|00\rangle + |11\rangle) + \beta (|0\rangle + |1\rangle)(|10\rangle + |01\rangle) \right]$$

$$= \frac{1}{2} \left[ |00\rangle (\alpha |0\rangle + \beta |1\rangle) + |01\rangle (\alpha |1\rangle + \beta |0\rangle) + |10\rangle (\alpha |0\rangle - \beta |1\rangle) + |11\rangle (\alpha |1\rangle - \beta |0\rangle) \right]$$  \hspace{1cm} (3.4)

where the equivalence on the second line arises by rearranging Alice’s two qubits together and then grouping like terms. Thus, four different terms are obtained. The state of Bob’s qubit is paranthesised in each of these terms, such that Alice can deduce what state Bob’s qubit is in by simply measuring the states of her two qubits, then sending these classical results to Bob. The possible states of Bob’s qubit are summarised below, depending on the measurements made by Alice.

- $00 \rightarrow |\psi_3(00)\rangle \equiv [\alpha |0\rangle + \beta |1\rangle]$
- $01 \rightarrow |\psi_3(01)\rangle \equiv [\alpha |1\rangle + \beta |0\rangle]$
- $10 \rightarrow |\psi_3(10)\rangle \equiv [\alpha |0\rangle - \beta |1\rangle]$
- $11 \rightarrow |\psi_3(11)\rangle \equiv [\alpha |1\rangle - \beta |0\rangle]$

As can be seen, if Alice sends Bob the two bits 00, then Bob doesn’t need to perform any further manipulations as his qubit is already identical to the state of the original $|\psi\rangle$ message that was to be sent. On the other hand, if the first bit sent by Alice is 1, then Bob must apply a $Z$ gate to achieve the desired result, while the second qubit being 1 means an $X$ gate must be applied.

Thus, Alice and Bob have successfully transmitted a quantum message without the need of a quantum communications channel! It may at first appear that this communication was done faster than the speed of light. However, one needs to recall that Alice still needs to send two classical bits to Bob so that he can manipulate his qubit to obtain the desired end result, so the laws of relativity set out by Einstein still hold [37].

3.2.2 Q# Implementation

Below is the Q# code for the program TeleportationSample.qs as featured on Microsoft’s GitHub repository [55]. The message has been encoded as a parameter variable `msg`, while the qubits of Alice and Bob are the variables `here` and `there` respectively. The built-in quantum operations that have been used are highlighted in the quantum circuit in Figure 3.2.
operation Teleport(msg : Qubit, there : Qubit) : ()
{
    body
    {
        using (register = Qubit[1]) {
            // (1) Ask for an auxiliary qubit that we can use to prepare for teleportation.
            let here = register[0];
            // (2) Create some entanglement that we can use to send our message.
            H(here);
            CNOT(here, there);
            // (3) Move our message into the entangled pair.
            CNOT(msg, here);
            H(msg);
            // (4) Measure out the entanglement.
            if (M(msg) == One) { Z(there); }
            if (M(here) == One) { X(there); }
            // (5) Reset our "here" qubit before releasing it.
            Reset(here);
        }
    }
}

The format of the code has been written from the perspective of the sender (Alice in the example given in Section 3.2.1), whereby the message and sender qubits are passed into the operation. The main steps that define the Teleport(msg : Qubit, there : Qubit) operation are as follows:

1. A using block is used within with a single qubit register is allocated. This is so the qubit cannot be used outside of this block. The register is assigned to a variable here (Alice’s qubit) using the let keyword. This type of assignment is used where variables cannot be changed e.g. here cannot be assigned to a different qubit in another register. If a variable needs to be manipulated, then the mutable keyword is used instead.

2. The qubits here and there are prepared into an entangled Bell Pair creating the $|\psi_0\rangle$ state. It can be seen that the latter gate uses Alice’s qubit as the control, while Bob’s is the target. This process was illustrated in Figure 3.1.

3. The message to be transferred is moved into the Bell State pair by now using the msg qubit as a control, then applying a Hadamard gate to it. This results in the state $|\psi_2\rangle$, where Bob’s qubit is in a superposition of four possible variations.
of the original message’s state.

4. First, the message qubit is measured in the Pauli-Z basis using the primitive operation \( M(\text{qubit} : \text{Qubit}) : \text{Result} \). This is done inside an if-statement such that if the result is 1, then the Pauli-Z gate is applied to Bob’s qubit. A similar statement measures Alice’s qubit, then applies the Pauli-X gate Bob’s qubit if it is 1, as was outlined in the previous section. Thus, the implementation has successfully been able to translate the quantum circuit into code!

5. As a measure of good practice, the \( \text{Reset}(\text{target} : \text{Qubit}) : () \) operation is used to change the qubit register assigned within the block to the state \( |0\rangle \) before releasing it.

This program highlights some of the most basic processing that can be done with qubits using a handful of the Primitive API components. Although ground-breaking from a conceptual perspective when compared to classical algorithms, the quantum teleportation algorithm is of little use by itself. Its power can be fully harnessed by using it as a subroutine for other, more complex algorithms.

The next two algorithm implementations are substantially more complex in nature. As such, simulation results for each will also be laid out so that a greater appreciation for their behaviour can be gained.

3.3 Deutsch-Jozsa Algorithm

The Deutsch-Jozsa algorithm was one of the first algorithms to demonstrate a quantum speed-up compared to any classical algorithm aiming to solve the same problem [56]. It takes advantage of the fact that quantum amplitudes may also be negative, whilst a classical probabilistic approach can only use positive probabilities. The algorithm demonstrates two key concepts which quantum computers exhibit: quantum parallelism whereby a function \( f(x) \) can be evaluated for different inputs simultaneously, and the phenomena of interference as was seen in the double-slit experiment demonstrating wave-particle duality.

The problem this algorithm aims to tackle is the determination of whether a function \( f(x) \) is constant or balanced. It takes bit strings \( x \) of length \( n \) (where each bit \( \in \{0, 1\} \)) and returns a binary value - either zero or unity. By constant, it is meant that the function returns the same value for all possible inputs. On the contrary, a balanced function is where the output is 0 for exactly half of the inputs, but 1 for the rest. Classically, the worst-case time complexity to solve this is \( O(2^{n−1}) \). Since there are \( 2^n \) possible numbers the input \( x \) can represent, the function would need to be queried with half the possible inputs plus one (i.e. \( 2^n/2 + 1 \)), since one may receive \( 2^n/2 \) 0’s before getting a 1, in which case the function is balanced [57].

By introducing the principles of quantum parallelism and interference, the Deutsch-Jozsa algorithm can deterministically solve the problem using a single call to the function \( f(x) \).
3.3.1 Algorithm Outline

Figure 3.3 indicates the overall circuit that is executed in the algorithm. It begins

![Circuit Diagram]

by preparing a state consisting of \( n \) qubit registers in the state \(|0\rangle\), so that the length \( n \) query \( x \) for the function \( f(x) \) can be stored. An extra register in the state \(|1\rangle\) is also prepared, so that the final output of the function being evaluated can be stored. Such a state can easily be created by using a Pauli-X gate on a \(|0\rangle\) state qubit.

Mathematically, the prepared state can be written as

\[
|\psi_0\rangle = |0\rangle^\otimes n |1\rangle
\]  

where the ‘\( \otimes n \)’ denotes the fact that there are \( n \) qubits in the \(|0\rangle\) state representing \( x \). A Hadamard transform, \( H^\otimes n \), is then applied to the query qubits, while the answer qubit is passed through a single Hadamard gate, \( H \) [37].

A Hadamard transform (also known as a Walsh-Hadamard gate) is essentially where the Hadamard gate is applied to \( n \) qubits. For example, if \( n = 2 \), then

\[
H^\otimes 2 |0\rangle^\otimes 2 = \left( \frac{|0\rangle + |1\rangle}{\sqrt{2}} \right) \left( \frac{|0\rangle + |1\rangle}{\sqrt{2}} \right) = \frac{|00\rangle + |01\rangle + |10\rangle + |11\rangle}{2}
\]

(3.6)

gives the maximal superposition state across all computational basis states. In compact form, it can be written for an \( n \) qubit string as

\[
H^\otimes n |0\rangle^\otimes n = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle
\]

(3.7)

where the sum is over all possible binary values of \( x \).
Going back to the circuit, applying these gates to the state $|\psi_0\rangle$ results in

$$|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle \left[ |0\rangle - |1\rangle \right]$$

(3.8)

This operation has resulted in the query qubits collectively being in a superposition of all possible values of $x$, while the answer qubit is in an equally weighted superposition of the two Pauli-Z basis states. The function $f(x)$ can now be applied to this state. This is done by using a 'black-box' operation called $U_f$ which takes an input $|x,y\rangle$ (where $y$ is the answer qubit in this case) and manipulates it such that the output is $|x,y \oplus f(x)\rangle$. This means that it leaves the query qubits unchanged, but performs addition modulo 2 on the answer register with the output of the function $f(x)$. This results in the state

$$|\psi_2\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} |x\rangle \left[ |0\rangle - |1\rangle \right]$$

(3.9)

where the second equality comes from considering both possibilities of $f(x)$ being 0 or 1 for each $x$.

At this point, the answer register can be ignored as it is no longer required. This is because the evaluation of the function is now stored in the amplitude of the state represented by the query qubits. Applying the Hadamard transform again to these qubits gives the state

$$|\psi_3\rangle = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} \left[ \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y} |y\rangle \right] \left[ |0\rangle - |1\rangle \right]$$

(3.10)

where $x \cdot y = x_1y_1 \oplus \ldots \oplus x_ny_n$ is the inner product of $x$ and $y$ under modulo 2. A single measurement of the query register will now allow one to determine whether $f(x)$ is constant or balance. This is because the probability of measuring the state $|0\rangle^\otimes n$ is given by

$$Prob(|0\rangle^\otimes n) = \left| \frac{1}{2^n} \sum_{x} (-1)^{f(x)} \right|^2$$

(3.11)

as $x \cdot y = 0$ in this case. This probability can be probed for both cases of the problem. In the instance where $f(x)$ is balanced, equation (3.11) evaluates to 0. This is because half of the values in the sum will be $-1$, while the other half will be $+1$, causing an overall destructive interference effect as all amplitudes will cancel out. On the other hand, if the function is constant then the probability will be 1 due to
constructive interference, meaning that the final state will be measured as $|0\rangle^\otimes n$ with complete certainty [37].

Thus, we have been able to determine a key property of the function $f(x)$ using only a single evaluation of it. Deciding whether it is constant or balanced is simply a matter of applying the specified circuit, then checking if the measured query state contains any 1’s. If so, then it is a balanced function; otherwise constant.

### 3.3.2 Q# Implementation

#### The Code

The Q# code in the SimpleAlgorithms.qs program provided on Microsoft’s GitHub repository contains the implementations of multiple algorithms [58]. One of these is the Deutsch-Jozsa algorithm. The schematic for the algorithm shown in Figure 3.3 is implemented in the form of an operation `IsConstantBooleanFunction(Uf : (Qubit[] => ()), n : Int) : Bool`, which takes in a ‘black-box’ unitary operator $U_f$ and an integer $n$ representing the size of the input $x$ into the function $f(x)$. This operation is shown below (note that some of the comments from the repository have been altered/removed).

```qsharp
operation IsConstantBooleanFunction(Uf : (Qubit[]) => ()), n : Int) : Bool
{
    body
    {
        mutable resultArray = new Result[n];

        using (qubits = Qubit[n + 1]) {
            // (1) The last qubit needs to be flipped so that the function will actually be computed into the phase when $U_f$ is applied.
            X(qubits[n]);

            // (2) Now, a Hadamard transform is applied to each of the qubits.
            ApplyToEach(H, qubits);

            // (3) We now apply $U_f$ to the $n + 1$ qubits, computing $|x, y\rangle \rightarrow |x, y \text{ plus_mod2} f(x)\rangle$.
            Uf(qubits);

            // (4) Applying the Hadamard transform to the query (first $n$) qubits
        }
    }
}
```

37
3.3. DEUTSCH-JOZSA ALGORITHM

Chapter 3. Quantum Algorithms

```csharp
ApplyToEach(H, qubits[0..(n - 1)]);

// (5) Measuring and resetting all qubits
for (idx in 0..(n - 1)) {
    set resultArray[idx] = MResetZ(qubits[idx]);
}

Reset(qubits[n]);

// (6) Checking if all measurements are 0
return ForAll(IsResultZero, resultArray);
```

It returns `true` if a given function specified by the unitary operator $U_f$ is constant, but `false` when it isn’t (in which case it is considered balanced). A mutable array of size $n$ in which the final measurements result will be stored is initially created. Subsequently, $n + 1$ qubits are allocated in an array, where the last qubit will be used to store the output of the function $f(x)$. The steps that follow are outlined below:

1. The $(n + 1)th$ qubit in the array is flipped using a Pauli-X gate. This is because all qubits start off in the $|0\rangle$ state, but one of these needs to be in the $|1\rangle$ state for the algorithm to execute correctly.

2. The Hadamard transform is applied to all qubits (including the answer qubit). This could be done by using a for-loop to apply a Hadamard gate to each element in the qubit array. Conveniently, the Canon library provides a function `ApplyToEach (singleElementOperation : ('T => ()), register : 'T[]) : ()` which is able to apply the Hadamard gate to all array elements at once in a single statement.

3. The pre-defined unitary operator $U_f$ is applied to all qubits. This changes them from the state $|x, y\rangle$ to $|x, y \oplus f(x)\rangle$ where in this case, $x$ is the first $n$ qubits in the array, and $y$ is the $(n + 1)th$ element. Note that the operation definition specifies that $U_f$ must be of type `Qubit[] => ()`. This means that it must take a type `Qubit` array as input, but doesn’t need to return anything.

4. The Hadamard transform is now applied to the first $n$ qubits. This is done by using index notation, specifying the range of elements to cover within a lower and upper bound as `qubits[0..(n-1)]`.

5. The query qubits are finally measured and stored in the results array. This is done using the predicate `MResetZ (target : Qubit) : Result`, which measures a qubit in the Pauli-Z basis and subsequently resets the qubit before it is released.

6. The algorithm dictates that a function is constant if a measurement of the query
The overall purpose of this operation is to apply the unitary

$$U|k, z\rangle = |k, z \oplus x_k\rangle$$

(3.12)

where $x_k = 1$ only when the integer representation of $|k\rangle$ is in the markedElements array. The use of the ControlledOnInt predicate over each marked element is such that it applies the Pauli-X gate to the target (answer) qubit only if the $n$ qubit inputs correspond to the marked element. Since this predicate only accepts an array instead of a single qubit, and the Pauli-X gate only accepts a single qubit, the ApplyToEachCA($X$, _) operation has been used so that the target can be accepted in array form.

In order to give this implementation of $U_f$ a return type of (Qubit[] => ()), it can be called from a function of the same return type (not shown here).
3.4. **SOLVING LINEAR EQUATIONS**

Chapter 3. Quantum Algorithms

Simulation

Below is the output of this program when it is run in the command line. Two different examples have been input. The first passes in the marked elements $[1, 2]$ for a two-qubit input. Since this state can range from the integers 0 to 3, the program should return false, meaning the function is balanced. The second example passes in the elements $[0, 1, 2, 3, 4, 5, 6, 7]$ for a three-qubit input. Since this contains all possible values of the input state, the program should return true, indicating a constant function. It can be seen from the simulation that this is indeed the case; the Deutsch-Jozsa algorithm has been successfully implemented.

Press enter key to continue...

Measured that test case $\{1, 2\}$ was balanced!

Measured that test case $\{0, 1, 2, 3, 4, 5, 6, 7\}$ was constant!

Both constant and balanced functions measured successfully!

Unfortunately, although this algorithm demonstrates a significant quantum speed-up, it isn’t used very much to solve actual real-world problems. Moreover, there are classical algorithms which take advantage of probabilistic strategies, whereby a small amount of randomised inputs can quickly converge to the nature of $f(x)$ with a high level of confidence [37]. The next algorithm is one that can be used in a wide variety of applications.

### 3.4 Solving Linear Equations

The ability to solve a system of linear equations has come time and time again in a plethora of different applications. Whether solving a system of differential equations or performing some sort of regression, many problems within fields such as science and even economics usually stumble upon a set of linear equations [59].

In compact matrix form, a system of linear equations can be denoted as $Ax = b$, where $A$ is known as a matrix of coefficients. The problem is such that the unknown vector $x$ needs to be determined given a known vector $b$. Analytically, this system can be solved by finding $x = A^{-1}b$, which may be trivial to solve, but as the system size increases this task becomes cumbersome. Classical algorithms which aim to solve such equations may exhibit a worst-case time complexity of $O(N^3)$ if the system is dense, and complexity $O(Ns\kappa)$ for sparse systems, where $s$ is the sparsity and $\kappa$ the condition number [60].

A quantum mechanical algorithm proposed by Harrow, Hassidim and Lloyd (HHL) is able to compute the unknown vector in time complexity $O(\kappa^2 \log N)$ [61]. This algorithm is confined to solve systems of equations consisting of $N \times N$ square matrices that are Hermitian and invertible. By Hermitian, it is meant that the matrix is...
self-adjoint i.e. $A^\dagger = A$ [62]. The quantum form of the problem can be written as

$$A |x\rangle = |b\rangle$$ (3.13)

where the vectors $x$ and $b$ are represented by the quantum states $|x\rangle$ and $|b\rangle$ respectively. The task then becomes to find a solution for $|x\rangle$ as

$$|x\rangle = \frac{A^{-1} |b\rangle}{\|A^{-1} |b\rangle\|}$$ (3.14)

where the normalisation arises due to the fact that the squared amplitudes of a quantum state must sum to unity.

The quantum algorithm which solves this problem uses a technique known as phase estimation [63]. This involves finding the eigenvalues of a unitary function $U$ with eigenstates $|\psi\rangle$, where the eigen-equation is given by

$$U |\psi\rangle = e^{i2\pi\theta} |\psi\rangle$$ (3.15)

The phase estimation algorithm does this by finding the value of the phase $\theta$ of the eigenvalue with a high probability using a series of controlled gates of the unitary operator. After performing a series of rotations in a particular order, the quantum state $|\psi\rangle$ has the binary representation of this phase encoded within. A subroutine within phase estimation is the (inverse) Quantum Fourier Transform (QFT). This routine performs a linear transformation on multiple qubits, and is the quantum mirror of the classical discrete Fourier Transform. More on the mechanisms of the QFT can be found at [64].

### 3.4.1 Algorithm Outline

Given that the $N \times N$ matrix $A$ in equation (3.13) has eigenstates $|u_j\rangle$ with corresponding eigenvalues $\lambda_j$, it can be decomposed into a combination of these. This technique is commonly referred to as spectral decomposition, and can only be done for matrices which are diagonalisable i.e. can be written in the form $A = R\Lambda R^\dagger$ where $\Lambda$ is a diagonal matrix containing the eigenvalues of $A$ [60]. Expanding $A$ in terms of its eigenstates gives

$$A = \sum_{j=1}^{N} \lambda_j |u_j\rangle \langle u_j|$$ (3.16)

$$A^{-1} = \sum_{j=1}^{N} \lambda_j^{-1} |u_j\rangle \langle u_j|$$ (3.17)

where $\langle u_j|$ is the complex conjugate of the eigenstate $|u_j\rangle$ and the inverse $A^{-1}$ is also shown. The state $|b\rangle$ can also be expanded in the eigenbasis of $A$ as

$$|b\rangle = \sum_{j=1}^{N} \beta_j |u_j\rangle$$ (3.18)
where $\beta_j = \langle u_j | b \rangle$. Then the solution we seek, $| x \rangle \propto A^{-1} | b \rangle$, can be written as

$$| x \rangle \propto \left( \sum_{k=1}^{N} \lambda_j^{-1} | u_k \rangle \langle u_k | \right) I \left( \sum_{j=1}^{N} \beta_j | u_j \rangle \right) = \sum_{j=1}^{N} \frac{\beta_j}{\lambda_j} | u_j \rangle$$

(3.19)

where in the last equality we have used the fact that $\langle u_k | u_j \rangle = \delta_{kj}$ as the eigenstates are orthogonal to each other.

In order to obtain the desired state in equation (3.19), the following steps can be performed [60].

Initially, an $N$ qubit register in the state $| 0 \rangle \otimes N$ is prepared along with the vector qubit $| b \rangle$, where the former register will be denoted by $c$. This state expands as

$$| b \rangle | 0 \rangle \otimes N = \sum_{j=1}^{N} \beta_j | u_j \rangle | 0 \rangle \otimes N$$

(3.20)

Subsequently, phase estimation corresponding to the operator $A$ is performed on the register $c$. This transforms them into a superposition of states which contain the eigenvalues of the matching eigenstates $| u_j \rangle$, giving

$$\sum_{j=1}^{N} \beta_j | u_j \rangle | 0 \rangle \otimes N \xrightarrow{\text{Phase Estimation}} \sum_{j=1}^{N} \beta_j | u_j \rangle | \lambda_j \rangle$$

(3.21)

This phase estimation is done using a controlled unitary matrix of the form

$$U = \sum_{\tau=0}^{2^n-1} | \tau \rangle \langle \tau | \otimes e^{i2\pi A\tau / 2^n}$$

(3.22)

where $n$ is the qubit number of register $c$. Thereafter, another register containing a single ancillary qubit in the $| 0 \rangle$ state is introduced. This is rotated about the Pauli-Y axis using the $R_y(\theta_j)$ rotation gate, where $C$ is a constant and $\theta_j = 2 \arcsin(C/\lambda_j) \propto 1/\lambda_j$ is the angle to be rotated through, determined by the eigenvalues stored in register $c$. This rotation leads to the state

$$\sum_{j=1}^{N} \beta_j | u_j \rangle | \lambda_j \rangle | 0 \rangle \xrightarrow{\text{Rotation}} \sum_{j=1}^{N} \beta_j | u_j \rangle | \lambda_j \rangle \left( \sqrt{1 - \frac{C^2}{\lambda_j^2}} | 0 \rangle + \frac{C}{\lambda_j} | 1 \rangle \right)$$

(3.23)

The inverse phase estimation can now be performed to disentangle the eigenvalue register $c$, so that it returns back to the state $| 0 \rangle \otimes N$. This leaves the state of qubit $| b \rangle$ and the ancillary as

$$\sum_{j=1}^{N} \beta_j | u_j \rangle \left( \sqrt{1 - \frac{C^2}{\lambda_j^2}} | 0 \rangle + \frac{C}{\lambda_j} | 1 \rangle \right)$$

(3.24)

It can be seen that the ancillary qubit is in a superposition state after the rotation. If it is now measured and collapses into the state $| 1 \rangle$, the state of register $| b \rangle$ becomes

$$C \sum_{j=1}^{N} \frac{\beta_j}{\lambda_j} | u_j \rangle \propto | x \rangle$$

(3.25)
Thus, it can be seen that the state of qubit $|b\rangle$ has transformed into one that is proportional to the desired state $|x\rangle$.

Figure 3.4 shows the circuit depicting this algorithm for a general $2 \times 2$ matrix. It can be seen that just four qubits have been used: two to represent the eigenvalues, one for the state $|b\rangle$ and one ancillary qubit. The different stages of the algorithm have been highlighted in yellow boxes, starting with phase estimation, then $R_y(\lambda_j^{-1})$ rotation, followed by an inverse phase estimation.

![Figure 3.4: Schematic diagram depicting the circuit used to solve a $2\times2$ matrix equation.](image)

As such, 2 qubits are used in register $c$, a single register for qubit $|b\rangle$, and an ancillary qubit which is measured at the end. It can be seen that the algorithm consists of a phase estimation, $R_y(\lambda_j^{-1})$ rotation and an inverse phase estimation. The values used are $t_0 = 2\pi$ and $r = 2$ [59].

$S$ is known as the $\pi/4$ phase gate, given by

$$ S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \quad (3.26) $$

This circuit makes a few approximations to the general algorithm. One of these is that the reciprocal of the eigenvalues, $\lambda_j^{-1}$, is achieved by using a SWAP gate on the first two qubits. This is because these two qubits represent the eigenvalues by which the ancillary qubit is rotated. For example, if a matrix has the eigenvalues $\lambda_1 = 1$ and $\lambda_2 = 2$ and register $c$ is in the state $|01\rangle$ (representing $\lambda_1$ in binary), then applying the SWAP gate changes it to the state $|10\rangle$, which is $2\lambda_1^{-1}$. The opposite occurs when the swap is performed on the state $|10\rangle$, giving $2\lambda_2^{-1}$. Thus, a rotation of $R_y(\theta_j)$ is in fact through the angle $\theta_j = (2\pi/4)\lambda_j^{-1}$, which arises from the fact that the approximation $\sin(\theta_j/2) \approx \theta_j/2$ has been made. Therefore, this circuit is restricted to matrices which have the aforementioned eigenvalues. It can be shown that for this particular circuit, $C = 0.736$ gives the desired output [65].

### 3.4.2 Q# Implementation

An implementation of this algorithm has not been done before in the Q# programming language. Therefore, the implementation highlighted below breaks new ground by introducing a new simulation amongst the plethora of examples already provided by Microsoft.
This implementation is specific to a particular, non-unitary matrix, but follows the same procedure outlined by the circuit in Figure 3.4. It is given by

\[
A = \frac{1}{2} \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix} = R_y^\dagger(\pi/2) \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} R_y(\pi/2)
\] (3.27)

where the last equality diagonalizes the matrix \(A\) into its two eigenvalues \(\lambda_1 = 1\) and \(\lambda_2 = 2\).

**Simulating the algorithm**

A portion of the algorithm, specifically the *Hamiltonian Simulation* in the circuit, requires controlled operations of \(e^{i\pi A}\) and \(e^{i\pi A/2}\). It can be shown that the \(e^{i\pi A}\) operation is simply a Pauli-X gate so it is easy to adopt in Q#. However, the \(e^{i\pi A/2}\) operation is given by

\[
e^{i\pi A/2} = \frac{1}{2} \begin{bmatrix} -1 + i & -1 - i \\ -1 - i & -1 + i \end{bmatrix}
\] (3.28)

which cannot directly be decomposed into the gates provided in the Primitive library. As a consequence, an approximation was made using a series of rotational gates as follows:

\[
R_y^\dagger(-\pi/2)R_x^\dagger(\pi)R_x(\pi/2)R_y(\pi/2) = \frac{1}{2} \begin{bmatrix} -1 - i & -1 - i \\ -1 - i & +1 - i \end{bmatrix}
\] (3.29)

Once these two gates were implemented, the rest of the circuit was translated into a Q# operation called `SolveLinearEquation` (angle : Double) : Bool[]. This operation returns a Boolean array containing the results of measurements performed on the ancillary qubit (\(q_3\)) as well as the qubit in state \(|\propto|x\rangle(b)\) after the algorithm is performed. The code for this program has been included in the Appendix B, as it adds little to the discussion.

However, there are a few things to note. Firstly, the algorithm takes a single argument, `angle`. This is because the state \(|b\rangle\) is prepared by applying the rotational gate \(R_y(\theta)\) to a qubit in the initial state \(|0\rangle\). This essentially creates a superposition state \(|b\rangle = \cos(\theta/2)|0\rangle + \sin(\theta/2)|1\rangle\) representing the state vector

\[
b = \begin{bmatrix} \cos(\theta/2) \\ \sin(\theta/2) \end{bmatrix}
\] (3.30)

Secondly, the (inverse) Quantum Fourier transform circuit shown in portion 3 of the circuit schematic (orange box) could have been implemented using the series of gates shown. However, it was found that Q# provides a `QFT` (qs : BigEndian) : () operation in its Canon library, which is able to perform the transform on an array of qubits representing an integer in big-endian representation. This significantly simplified the implementation.
Results and Discussion

Theoretically, the register containing qubit $|b\rangle$ becomes proportional to the desired state $|x\rangle$ after performing the algorithm. This means that it is in a superposition state of $|0\rangle$ and $|1\rangle$ where the amplitudes of these two states gives the components of the normalised vector $x$. However, as previously mentioned, Q# makes these amplitudes opaque to the user. As a result, it was decided that the algorithm would be sampled multiple times for any given angle, and the ratio of measuring the qubit in the state 0 compared to 1 would be used to calculate the probability of each occurring. Thus, the amplitudes can be approximated by normalising these two probabilities together such that their magnitude is unity if represented in vector format.

Table 3.1 below shows the measured results (to three significant figures) of the vector $\hat{x} = x/\|x\|$ for a range of angles specifying the state of $|b\rangle$. The algorithm was sampled a total of 10,000 times to get a fairly accurate amplitude estimate. This was repeated 10 times so that an average could be computed, with the quoted error being the standard deviation across the 10 iterations. The theoretical results for each angle have also been provided as a comparison.

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>$x_{\text{theory}}$</th>
<th>$x_{\text{measured}}$</th>
<th>$\Delta \hat{x}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0$</td>
<td>$(+0.949, -0.316)$</td>
<td>$(+0.972 \pm 0.002, -0.234 \pm 0.009)$</td>
<td>$(0.023, 0.0082)$</td>
</tr>
<tr>
<td>$\pi/2$</td>
<td>$(+0.707, +0.707)$</td>
<td>$(+0.705 \pm 0.016, +0.708 \pm 0.016)$</td>
<td>$(0.002, 0.001)$</td>
</tr>
<tr>
<td>$2\pi/3$</td>
<td>$(+0.289, +0.957)$</td>
<td>$(+0.357 \pm 0.008, +0.934 \pm 0.003)$</td>
<td>$(0.068, 0.023)$</td>
</tr>
<tr>
<td>$2\pi/5$</td>
<td>$(+0.888, +0.461)$</td>
<td>$(+0.875 \pm 0.008, +0.484 \pm 0.015)$</td>
<td>$(0.013, 0.023)$</td>
</tr>
</tbody>
</table>

Table 3.1: The results obtained for the algorithm over various angles specifying the state of $|b\rangle$. The measurements appear to be fairly close to the theoretical solution.

It can be seen that the results of the simulation seem to be reasonably close to the theoretical result. As can be seen, the angle of $\pi/2$ gave the most accurate results, as the deviation $\Delta \hat{x}$ from the true solution is incredibly small. However, it also appears to be the least precise as it has the highest level of uncertainty in the measurements. On the other hand, the angle of $0$ appears to be the least accurate, but the most precise.

The slight deviations suggest that the algorithm isn’t completely perfect. This may be because of the approximation made to the Hamiltonian Simulation where the controlled $e^{i\pi A/2}$ gate could not be decomposed into the principle circuit gates. Moreover, the fact that the amplitudes of the final state couldn’t be seen meant only a probabilistic approach could be used to calculate them. This itself introduces an extra layer of uncertainty shrouding what may possibly be a more accurate result. Furthermore, the fact that the algorithm had to be run numerous times to get the desired result made the task more tedious than necessary. This could be remedied by carrying out the algorithm on an ensemble system, whereby multiple inputs are run through the circuit simultaneously so that an accurate result can be determine in a shorter period of time.
Should Q# allow arbitrary matrices to be used, the algorithm could be made even more accurate in the future. This is because one wouldn’t have to resort to finding a series of elementary gates which may or may not combine to give the exact matrix being considered. Even still, the results indicate that the implementation works very well up to about two decimal places.

### 3.4.3 Applications in Machine Learning

As mentioned earlier, the task of solving linear equations appears in many different applications across many different fields. The field of Machine Learning constantly requires the refinement of a set of weights $w$ when training Neural Networks to maximise the accuracy of predictions on unseen data. For instance, a Support Vector Machine (SVM) is a form of supervised machine learning used to perform binary classifications. When training, this algorithm requires that a hyperplane $w \cdot x + b = 0$ divides a series of data points such that $w \cdot x + b \geq 1$ when the classification is positive, and $w \cdot x + b \leq -1$ when negative. The vector $x$ is an $N$-dimensional vector representing the feature space of the data that is used to train the network, and $b$ is some constant.

The underlying linear equation that needs to be solved in this optimisation problem is given by

$$
\begin{bmatrix}
0 & 1^T \\
1 & K + \gamma^{-1}I
\end{bmatrix}
\begin{bmatrix}
b \\
\alpha
\end{bmatrix} =
\begin{bmatrix}
0 \\
y
\end{bmatrix}
$$

(3.31)

where $K$ is referred to as a kernel matrix where $K_{ij} = x_i \cdot x_j$, $\gamma$ is the tuning parameter, and $\alpha$ gives the coefficients of the weights such that $w = \sum_{j=1}^{M} \alpha_j x_j$ [60].

This is where a quantum linear solver would provide significant speed-ups. An algorithm, such as the one described in this section, could be used to find the unknown state $|b, \alpha \rangle$. This is but one of the many problems where the power of quantum algorithms could be harnessed to solve such systems.
Chapter 4

Quantum Game Theory

4.1 What is Game Theory?

Game Theory is where mathematical models are used to describe the actions of rational decision-makers, whether those decisions are cooperative or conflicting [66]. Used in a wide range of fields including economics, psychology and computer science, it can be used to justify which action(s) are the best to make in a given scenario.

4.1.1 Prisoner’s Dilemma

A famous example of Game Theory is the Prisoner’s Dilemma problem, which has been studied extensively to deduce that two prisoners may not cooperate even though it appears to be in their best interests to do so, by looking at the problem from first-person as opposed to a birds-eye-view [67].

Prisoner's Dilemma involves two (rational) individuals who have been isolated in solitary confinement for a crime they may have committed, unable to communicate with each other. Insufficient evidence is possessed by prosecutors in order to convict the pair for the crime. A compromise is therefore offered to each prisoner - they may defect by accusing the other prisoner of the crime or cooperate with them by remaining silent. Figure 4.1 demonstrates this using a pay-off matrix (sometimes called a normal form), where the prisoners are distinguished as $A$ and $B$.

The integers represent the number of years served in prison by $A$ and $B$. At first sight, it may seem that the optimal approach is for both to cooperate as this leads to a minimum total of 2 years served by the two. However, upon closer inspection it is realised that from a first-person perspective, such as prisoner $B$’s, defecting will always give the better outcome compared to cooperating, regardless of the decision made by the counterpart.

A key concept within Game Theory is Nash Equilibrium [69]. This states that the optimal result for a particular game is when each entity has no incentive to change
4.1. WHAT IS GAME THEORY?

Chapter 4. Quantum Game Theory

4.1. WHAT IS GAME THEORY?

4.1.2 Bayesian Games

Bayesian games are a set of games in Game Theory where each participant has incomplete information about other players of the same game session [70]. Unlike Prisoner’s Dilemma, where each prisoner knows that the counterpart has outcomes that are symmetric to their own, players of Bayesian games use probability distributions to decide on which strategy to adopt. Such games usually work by having some random element involved where each player is assigned parameter value(s) that they have no way of determining before the game begins. Moreover, no matter how much players converse about possible strategies depending on the values they each may obtain, they will not be able to bolster their pay-offs once the game commences, as they can no longer communicate with one another. The Nash Equilibrium of such games may be such that the combined highest possible pay-off of all players is lower than if the game was non-Bayesian i.e. there was no imbalance of information amongst participants.

Come Quantum Game Theory, many new decision routes enter the scene by introducing key quantum mechanical principles including superposed initial states, quantum entanglement of initial states, and even superposition of strategies [71]. The remainder of this chapter will focus on one particular Bayesian game that was explored, known as The Magic Square Game, and how it can be transformed to encompass a quantum strategy. Thereafter, we will see how the original version of

Figure 4.1: A pay-off matrix showing the number of years served in prison of two individuals A and B for all permutations of their combined actions - remain silent or betray [68].

their strategy after considering all of the possible choices of other players. This assumes that the strategies of other players remains constant. The game of Prisoner’s Dilemma has one Nash Equilibrium - both players betraying each other. This is because, although mutual cooperation leads to the best overall outcome, betraying is always the safer option from the perspective of any prisoner, as the other may choose to deviate from remaining silent.
the game can be simulated in Q#, something which has never been publicly done before.

## 4.2 The Mermin-Peres Game

The Mermin-Peres Magic Square game was originally devised by David Mermin and Asher Peres in order to demonstrate the concept of quantum contextuality [72] [73]. This is a theory which states that any measurement being made of a quantum observable is highly dependent on the measuring/detecting equipment used. David later went on to adopt the interpretation of quantum theory as tool for decision making not unlike classical probability theory instead of a physical formalism, now referred to as Quantum Bayesianism (QBism) [74].

### 4.2.1 The Classical Magic Square Game

The classical version of the game involves two players, who we will name Alice and Bob. They participate in what is known as a cooperative, non-local game i.e. they must cooperate with each other so that they both win the game, but have no means of communicating with one another once play commences. There is also an intermediary referee who randomly assigns parameters to each individual but does not disclose the information given to the other party. The referee also decides whether Alice and Bob have successfully won the game or not.

The steps involved in the game are outlined below:

1. Alice and Bob are separated from one another such that they have no way of communicating with each other.
2. The referee randomly selects a row, \( r \in [1, 2, 3] \), and gives this information to Alice (but doesn’t tell Bob).
3. Similarly, the referee randomly selects a column, \( c \in [1, 2, 3] \), and gives this information to Bob (but doesn’t tell Alice).
4. Alice and Bob are then each presented with their own 3 by 3 grid.
5. Alice is told to fill in the row \( r \) of her grid with \( +1 \)'s and \( -1 \)'s, such that the overall parity of the row is even i.e. the numbers multiply to a \( +1 \).
6. Bob is told to fill in the column \( c \) of his grid with \( +1 \)'s and \( -1 \)'s, such that the overall parity of the column is odd i.e. the numbers multiply to a \( -1 \).
7. The referee then checks that Alice has successfully filled the correct row with an even parity. Likewise, he checks if Bob has filled the correct column with an odd parity.
4.2. THE MERMIN-PERES GAME

8. Finally, the referee checks the number placed by Alice and Bob at the intersecting cell on the 3 by 3 grid with row $r$ and column $c$.

9. If both have filled this cell with the same number, they win the game. Otherwise, they lose the game.

As can be seen by these rules, this is a form of Bayesian game because Alice has no way of knowing which random column was assigned to Bob, and vice versa for Bob, giving them asymmetric imperfect information. Furthermore, it should be noted that Alice and Bob fill their respective row and column simultaneously and have no information about how the counterpart has filled in their grid until after the outcome of the game has been revealed.

What is of particular interest for this game is that there is no classical winning strategy that would allow Alice and Bob to win the game 100% of the time. In fact, it can be shown the maximum probability of winning the game is $\frac{8}{9}$ times. Moreover, no matter how much they discuss strategies prior to the game, this chance of winning cannot be surpassed.

In general, for a two-party game (denoted by $G$) employing a deterministic classical strategy, suppose that Alice is given a question $a$ for which she provides an answer using a deterministic function $f(a)$. Similarly, Bob is given a question $b$ and gives an answer using a similar function $g(b)$. Then, the proportion of legal questions $(a, b)$ for which there is a winning answer $(f(a), g(b))$ gives the success rate of the strategy. Mathematically, the maximum success proportion, $\hat{w}(G)$, is given by

$$\hat{w}(G) = \max_{f,g} \frac{\#\{(a,b) \in P \mid (a,b, f(a), g(b)) \in W\}}{\#P}$$ (4.1)

where $P$ is known as the promise (the set of all possible legal inputs) and $W$ is the winning condition (the set of all possible winning combinations).

On the contrary, a probabilistic classical strategy allows players to introduce and share random variables. The success of such a strategy is then determined by finding the probability $p$ with which a legal answer is found for all the possible legal questions that can be input into the game. The probability that a strategy $s$ gives a valid answer for a question $(a,b)$ can be denoted by $P_s(win \mid (a,b))$, such that the maximum chance of success across all probabilistic strategies, $w(G)$, is given by

$$w(G) = \max_s \min_{(a,b) \in P} P_s(win \mid (a,b))$$ (4.2)

It can be shown that for any game $G$, $\hat{w}(G) \geq w(G)$ i.e. the probability of winning the game with a probabilistic strategy cannot exceed the proportion of wins when employing a deterministic approach. Moreover, if the game has a set of legal questions $P$ but has no classical winning strategy (i.e. cannot be won 100% of the time), then it holds that

$$w(G) \leq \hat{w}(G) \leq \frac{\#P - 1}{\#P}$$ (4.3)
where $\#P$ is the total number of questions possible. Thus, it can be seen from equation (4.3) that for the Magic Square game, which has a total of $3 \times 3 = 9$ possible questions, the maximum chance of success is $8/9$ [75].

Perhaps a more intuitive explanation for this value is by attempting to construct a consistency table. Figure 4.2 shows one such example, illustrating how Alice and Bob may decide to strategise by pre-selecting the numbers input into the $3 \times 3$ grid so that all the rules of the Magic Square game are obeyed.

![Figure 4.2: An attempt at filling a 3 by 3 grid with +1’s and −1’s such that the Magic Sqaure game can always be won. It can be seen that a contradiction appears when the final cell is filled with either value [76].](image)

It can be seen that although eight of the cells have been filled with definite values, the final cell with a '?' cannot be filled without creating a contradiction; if it is filled with a $-1$, the row will have an even parity but the column won't have an odd parity and vice versa if a $+1$ is placed. Thus, one can see that this strategy can only be employed successfully $8/9$'s of the time.

### 4.2.2 Quantum Pseudo-Telepathy

Quantum pseudo-telepathy refers to the ability of parties to cooperate with each other in order to complete a task with no form of communication [75]. This is achieved by having them pre-share entangled states of particles, such that they can manipulate the states and then perform measurements that would yield the desired outcome. The name arises from the fact that for a classical observer who is oblivious to quantum effects it would appear as though the parties possess 'telepathic powers' since no apparent physical communication is taking place.

This technique can be applied to the Magic Square game, creating a quantum strategy that wins with complete certainty for all possible permutations of the game. The
4.2. THE MERMIN-PERES GAME

steps of the game play out the same way as the classical version with a few exceptions. Firstly, before moving apart Alice and Bob create an entangled state from four qubits, $|\psi\rangle$, given by

$$
|\psi_0\rangle = \frac{1}{2} \left[ |0000\rangle + |0101\rangle + |1010\rangle + |1111\rangle \right]
$$

where the first two qubits in each state belong to Alice, and the the final two belong to Bob. After this, Alice and Bob are separated (potentially even light years apart), each only having their two respective qubits of the entangled state at hand.

Thereafter, the referee selects a random row, $r$, and column, $c$, and tells them to Alice and Bob respectively, as usual. Now, instead of filling in their grids in a somewhat random fashion as would be done classically, they instead apply the unitary operations $A_r$ and $B_c$ to their qubits respectively. Subsequently, each measures their two qubits causing them to collapse into two bits of information each. The first bit tells Alice (Bob) how to fill in the leftmost (topmost) cell of their row (column), while the second indicates how the middle cell should be filled. The final cell can easily be determined by choosing the binary value which makes the overall row (column) even (odd) in parity.

For instance, if a bit has the value 0, then the corresponding cell is filled with a 0, and likewise if the value is 1. In this basis of using 0's and 1's instead of +1's and −1's, the parity is checked by simply adding the numbers together instead of multiplying, where an even (odd) sum represents an even (odd) parity. This change doesn’t affect the core mechanics of the game.

The $4 \times 4$ unitary matrices representing the operators which Alice and Bob apply to their qubits before making measurements are given by

$$
A_1 = \frac{1}{\sqrt{2}} \begin{bmatrix}
1 & 0 & 0 & i \\
0 & 1 & -i & 0 \\
i & -i & 1 & 0 \\
0 & i & 1 & 0
\end{bmatrix}, \\
A_2 = \frac{1}{2} \begin{bmatrix}
1 & i & i & 1 \\
1 & -i & i & -1 \\
i & i & -i & 1 \\
i & -i & -i & 1
\end{bmatrix}, \\
A_3 = \frac{1}{2} \begin{bmatrix}
1 & 1 & 1 & -1 \\
1 & 1 & -1 & 1 \\
1 & -1 & 1 & 1 \\
-1 & 1 & 1 & 1
\end{bmatrix}
$$

$$
B_1 = \frac{1}{2} \begin{bmatrix}
i & -i & 1 & 1 \\
-i & -i & 1 & -1 \\
1 & 1 & -i & i \\
i & i & 1 & 1
\end{bmatrix}, \\
B_2 = \frac{1}{2} \begin{bmatrix}
-1 & i & 1 & i \\
1 & i & -i & i \\
i & -i & 1 & i \\
-1 & -i & 1 & -i
\end{bmatrix}, \\
B_3 = \frac{1}{\sqrt{2}} \begin{bmatrix}
1 & 0 & 0 & 1 \\
-1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 \\
0 & 1 & -1 & 0
\end{bmatrix}
$$

It can be seen that there are six different matrices, where those in equation (4.5) are used by Alice according to the row $r$ she is given, while the bottom three in equation (4.6) are used by Bob according to the column $c$. In any run of the game, two of these matrices are applied to the entangled state. What is unique about these matrices is that they commute, meaning that one of Bob’s matrices can be applied before Alice’s or vice versa - the resulting state, $|\psi\rangle$, will give essentially the same desired outcome.
As an example, suppose the referee picks the row \( r = 2 \) and column \( c = 3 \). This would mean that Alice would apply the matrix \( A_2 \) to her qubits, and simultaneously Bob will use matrix \( B_3 \) on his two qubits. The net result of this will be the quantum state

\[
|\psi\rangle = (A_2 \otimes B_3) |\psi_0\rangle
\]  

(4.7)

\[
= \frac{1}{2\sqrt{2}} \left[ |0000\rangle + i |0010\rangle - |0101\rangle - i |0111\rangle \right. 
\]

(4.8)

\[
- |1001\rangle + i |1011\rangle + |1100\rangle - i |1110\rangle \right]
\]  

(4.9)

Thus it can be seen that the resulting state is a superposition of eight out of the sixteen possible configurations for the 4 qubits. Also, each of these states is equally likely to be measured with probability \( \frac{1}{8} \), which can be calculated by multiplying the amplitude of any state with its complex conjugate.

After obtaining this state, suppose that Alice and Bob measure their qubits such that the wavefunction collapses to give the bits \( 1001 \) (the fifth state in the equation above). Since the first two qubits belonged to Alice, she will only see the bits \( 10 \), while Bob will measure the bits \( 01 \). Subsequently, Alice will add a \( 1 \) to the final cell to give an overall even parity as \( a = 101 \). Similarly, Bob will add a \( 0 \) to his final cell to give a column of odd parity as \( b = 010 \). Thereafter, the referee can easily verify that they have won the game by the fact that \( a_3 = b_2 = 1 \) i.e. the intersecting cell contains the same integer. The same approach can be used to confirm that each of the seven other possible measurements made through this state also lead to a winning outcome.

Matrix Constraints

\[
\begin{pmatrix}
1 & 0 & 0 & 1 & i & 0 & 0 & i & i & 0 & 0 & i & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & -i & 0 & 0 & i & -i & 0 & 0 & i & -1 & 0 & 0 & 1 \\
0 & 1 & 1 & 0 & 0 & i & 0 & 0 & i & i & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & -1 & 0 & 0 & i & -i & 0 & 0 & i & -i & 0 & 0 & 1 & -1 \\
1 & 0 & 0 & 1 & -i & 0 & 0 & -i & i & 0 & 0 & i & -1 & 0 & 0 & -1 \\
-1 & 0 & 0 & 1 & i & 0 & 0 & -i & -i & 0 & 0 & i & 1 & 0 & 0 & -1 \\
0 & 1 & 1 & 0 & 0 & -i & 0 & 0 & i & i & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & -1 & 0 & 0 & -i & i & 0 & 0 & i & -i & 0 & 0 & -1 & 1 \\
1 & 0 & 0 & 1 & i & 0 & 0 & i & -i & 0 & 0 & -i & -1 & 0 & 0 & -1 \\
-1 & 0 & 0 & 1 & -i & 0 & 0 & i & i & 0 & 0 & -i & 1 & 0 & 0 & -1 \\
0 & 1 & 1 & 0 & 0 & i & 0 & 0 & -i & -i & 0 & 0 & 1 & -1 & 0 \\
0 & 1 & -1 & 0 & 0 & i & -i & 0 & 0 & -i & i & 0 & 0 & -1 & 1 \\
1 & 0 & 0 & 1 & -i & 0 & 0 & -i & -i & 0 & 0 & -i & 1 & 0 & 0 & 1 \\
-1 & 0 & 0 & 1 & i & 0 & 0 & -i & i & 0 & 0 & i & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & -i & -i & 0 & 0 & -i & i & 0 & 0 & 1 & 1 & 0 \\
0 & 1 & -1 & 0 & 0 & -i & i & 0 & 0 & -i & -i & 0 & 0 & 1 & -1 & 0
\end{pmatrix} = \frac{1}{2\sqrt{2}}
\]  

(4.10)
The $16 \times 16$ matrix represented by the tensor product $A_2 \otimes B_3$ is given in equation (4.10). Although this matrix looks very involved, we can inspect its properties to see exactly how it leads to a winning solution for the Magic Square game. Firstly, it can be noted that if the pre-shared entangled state between the players given in equation (4.4) is written as a $16 \times 1$ vector of the complex amplitudes of all possible states in the superposition, then only four of those entries will be non-zero. Therefore, when it is multiplied by this tensor product, only the 1st, 6th, 11th and 16th columns (highlighted in blue) have any effect on the wavefunction. This means that we can ignore all other columns of the matrix, significantly simplifying the problem.

Upon closer inspection of the rows, it is realised that those highlighted in red sum to zero when the relevant columns are added together. Relating these rows to the corresponding 4 qubit state, we find that they are those states which actually lead to a loss. Therefore, the constraint on this matrix is identified to be one where the amplitudes of the losing states always sum to zero. Thus, we can shift our attention from viewing winning in terms of the game to one where this constraint must be satisfied for all the possible matrix combinations in equations (4.5) and (4.6). Ultimately, this means that we don’t have to worry so much about what the amplitudes are on the winning states, as long as there are no losing states in the final superposition of the wavefunction $|\psi\rangle$.

As there are nine different matrix combinations possible depending on the row and column selected, and each combination has a total of eight rows corresponding to losing states, there are a total of 72 equations which must be satisfied. Thus, these constraints explain how the six matrices arise, and indeed may not be the only solution to this set of linear equations [77].

4.3 Q# Implementation

The Mermin-Peres Magic Square game was implemented and simulated in Q# successfully. Below are details of how this was achieved, followed by example runs of the simulation.

4.3.1 Building the Game Board Circuits

A large drawback of Q# is that it currently doesn’t support arbitrary unitary matrices to be used directly to manipulate qubits. As a result, the six matrix operators given in equations (4.5) and (4.6) needed to be translated into quantum circuit forms. Unfortunately, Q# currently doesn’t provide this translation functionality either. This led to the only other option: using a brute force approach to find a combination of quantum gates that gave the required matrix. This not only had to be done once but a total of six times, giving one circuit for each matrix.
Indeed, it was found that others in the quantum programming community had faced the same issue of circuit creation from unitary matrices. An open-sourced program written using a classical approach in C# which aimed to tackle this issue was found at [78]. The program contained a function called `FindPrintCircuitMatchingMatrix(ComplexMatrix target)` which took an instance of the struct `ComplexMatrix` and iterated through permutations of a vast array of matrices until it found a combination which gave the desired matrix `target` using at most four quantum gates.

Inspecting the program further, it was found that it contained a `Circuits` class which held all of the most fundamental quantum gates available as well as a few derived ones. It was decided that most of these gates would be omitted from the search so that only a total of six different ones were used: 2 single qubit gates and 4 double qubit gates. These gates are outlined in Tables 4.1 and 4.2, where some of them have been reproduced from previous sections for completeness.

### Table 4.1: Two single qubit gates were used in the Q# implementation. These were the Hadamard and Splitter gates.

<table>
<thead>
<tr>
<th>Name</th>
<th>Hadamard</th>
<th>Splitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol</td>
<td><img src="image" alt="Hadamard Symbol" /></td>
<td><img src="image" alt="Splitter Symbol" /></td>
</tr>
<tr>
<td>Matrix</td>
<td>[ \frac{1}{\sqrt{2}} \begin{bmatrix} 1 &amp; 1 \ 1 &amp; -1 \end{bmatrix} ]</td>
<td>[ \frac{1}{\sqrt{2}} \begin{bmatrix} 1 &amp; i \ i &amp; 1 \end{bmatrix} ]</td>
</tr>
</tbody>
</table>

As can be seen, the two single qubit gates used were the *Hadamard* and *Splitter* gates. The latter is reminiscent of the effects experienced by photons when passed through a beam splitter. This gate can be further decomposed into the primitive gates that Q# already provides as

\[
SPLIT = S^\dagger HXS
\]

where \( X \) is the Pauli-X gate and \( S \) is the \( \pi/4 \) phase gate.

The four double qubit gates used were the *CNOT*, *Swap*, *Increment* and *Decrement*. The last two acquire their names from the fact that they can increase/decrease the binary representation of two qubits by a single value, in which case the lower wire
4.3. Q# IMPLEMENTATION

Chapter 4. Quantum Game Theory

**Table 4.2:** Four double qubit gates were used in the Q# implementation. These were the CNOT, Swap, Increment and Decrement gates.

<table>
<thead>
<tr>
<th>Name</th>
<th>Swap</th>
<th>Controlled Not</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol</td>
<td><img src="image" alt="Symbol" /></td>
<td><img src="image" alt="Symbol" /></td>
</tr>
<tr>
<td>Matrix</td>
<td><img src="image" alt="Matrix" /></td>
<td><img src="image" alt="Matrix" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Decrement</th>
<th>Increment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol</td>
<td><img src="image" alt="Symbol" /></td>
<td><img src="image" alt="Symbol" /></td>
</tr>
<tr>
<td>Matrix</td>
<td><img src="image" alt="Matrix" /></td>
<td><img src="image" alt="Matrix" /></td>
</tr>
</tbody>
</table>

represents the least significant qubit. The four inputs and their corresponding outputs are given below for the Increment gate, where applying it to the $|11\rangle$ state simply resets it to the binary value of zero.

$|00\rangle \rightarrow |01\rangle$

$|01\rangle \rightarrow |10\rangle$

$|10\rangle \rightarrow |11\rangle$

$|11\rangle \rightarrow |00\rangle$

Such a gate can simply be implemented using a Pauli-X and CNOT gate as shown in Figure 4.3.

Once these gates were selected, the six matrices were input into the function, which was successfully able to return valid circuits for each. Below is an example output of the program showing the range of circuits returned for the $B_3$ matrix.
Figure 4.3: The unity incremental gate can be implemented using a Pauli-X and CNOT gate. The lower qubit, $|q_2\rangle$, is the least significant in the binary representation.

Searching for circuits that match:

```
| 0.707 | 0     | 0     | 0.707 |
| -0.707| 0     | 0     | 0.707 |
| 0     | 0.707 | 0.707 | 0     |
| 0     | 0.707 | -0.707| 0     |
```

----- (may take awhile)...

Gates.ControlledNot1When2.Then(Gates.H.OnWire2Of2())
Gates.Increment.Then(Gates.H.OnWire2Of2())
Gates.ControlledNot2When1.Then(Gates.H.OnWire1Of2()).Then(Gates.Swap)
Gates.ControlledNot1When2.Then(Gates.BeamSplit.OnWire2Of2()).Then(Gates.H.OnWire2Of2())
Gates.Swap.Then(Gates.Increment).Then(Gates.H.OnWire2Of2())
Gates.Increment.Then(Gates.BeamSplit.OnWire2Of2()).Then(Gates.H.OnWire2Of2())
Gates.ControlledNot2When1.Then(Gates.BeamSplit.OnWire1Of2()).Then(Gates.H.OnWire1Of2()).Then(Gates.Swap)
Gates.Swap.Then(Gates.Increment).Then(Gates.BeamSplit.OnWire2Of2()).Then(Gates.H.OnWire2Of2())
```

The eagled-eyed will find that some of these circuits, when multiplied out using the matrix forms of the gates, don't actually give the exact matrix $B_3$. This is because the program weakens some of the constraints further. In particular, it produces circuits where the resultant matrices may have certain rows with differing phases. This is because multiplying any row by, say, $-1$ or even $i$, doesn't change the net effect of the matrix. This enables the program to increase its effectiveness, producing a vast array of circuits as can be seen in the output above, with some more complicated than others. For this specific operator, the first circuit shown was implemented as it used the fewest number of gates.
Figure 4.4 shows the final circuits that were selected for the Q# implementation for all six matrix operators. The circuits for $A_{1,2,3}$ have been labelled as *Alice -Top, -Mid* and *-Bot* respectively, while the circuits for $B_{1,2,3}$ are labelled with *Bob -Left, -Mid* and *-Right* respectively.

**Figure 4.4:** The six circuits representing the matrix operations $A_{1,2,3}$ and $B_{1,2,3}$. These were implemented in Q#.

Where single qubit gates have been applied to one of the wires, the matrix across both wires is simply the tensor product of that gate with the identity matrix. For example, applying the Hadamard gate to the first wire can be represented as $H \otimes I$, while applying it to the second wire gives $I \otimes H$.

Figure 4.5 shows how these circuits are applied in a particular run of the game where Alice is given the row $r = 1$ to fill in, while Bob is given the column $c = 3$. 
Figure 4.5: A grid depicting the game board for the Mermin-Peres game in Q#. The instance where the referee picks the row $r = 1$ and column $c = 3$ is highlighted. Alice measures her qubits in the green regions, while Bob measures his in the blue and (first) purple cells.

4.3.2 Simulating and Testing the Game

The Code

Q# already provides built-in operations for three out of the six gates used. These are the Hadamard, CNOT and Swap gates. Implementations of the Increment (INC), Decrement (DEC) and Splitter (SPLIT) gates are shown below.

```qsharp
// Operation to increment the binary representation of two qubits, q2 is the least significant qubit
operation INC (q1 : Qubit, q2 : Qubit) : ()
```

4.3. Q# IMPLEMENTATION

The \texttt{INC} operation could have been implemented by using a combination of the built-in \texttt{CNOT} and \texttt{X} operations in a manner identical to that depicted in Figure 4.3. However, this need not be done since Q# supplies an operation \texttt{IntegerIncrementLE(increment : Int, target : LittleEndian)} in its Canon library, which is able to increment the integer encoded by the target qubit(s) state, $|x\rangle$, by a specified amount, $y$, as

$$|x\rangle \rightarrow |x + y\rangle$$

where the addition is done under modulo $2^N$ ($N$ is the length of the target state). It does this assuming that the qubit(s) represent an unsigned integer in the little-endian format i.e. the first qubit in the array is the least significant. The \texttt{LittleEndian} cast used converts an array of type \texttt{Qubit} into this format, where it can be seen that \texttt{q2} is the least significant as required.

Looking at the \texttt{DEC} implementation, it can be seen that the \texttt{adjoint} of the Increment gate was used. This is because the matrix which decrements the two qubits is in fact the transpose of the Increment gate (it contains no complex entries), so the adjoint
auto keyword can be utilised within the INC operation, allowing Q# to automatically apply the body of this operation in reverse.

Note that although the matrix multiplication of the gates which make up the Splitter operator are from left to right, the gates are implemented in the opposite direction within SPLIT, starting with \( S \) instead of its adjoint form \( S^\dagger \). Also, the \( S^\dagger \) gate has been applied by adding the prefix Adjoint before the \( S \) operation.

Below is the main operation, PseudoTelepathy(row : Int, col : Int) : Bool[], within the Operation.qs file which simulates the entire quantum pseudo-telepathy process, from entanglement to measurement. It is this operation which is called in the Driver.cs file (not shown).

```qsharp
operation PseudoTelepathy(row : Int, col : Int) : Bool[]
{
    body
    {
        // The result of the measured qubits will be stored in an array
        mutable measurements = new Result[4];

        using (qubits = Qubit[4])
        {
            // (1) Qubits a1 and b1 belong to Alice
            let a1 = qubits[0];
            let b1 = qubits[1];
            // Qubits a2 and b2 belong to Bob
            let a2 = qubits[2];
            let b2 = qubits[3];

            // (2) Alice and Bob entangle their qubits into two Bell State pairs before parting
            PrepareEntangledState([a1; b1], [a2; b2]);

            // (3) Each apply their circuit depending on the row/col given to them by the ref
            // They have no means of communicating with each other
            ApplyAliceCircuit(row, a1, b1);
            ApplyBobCircuit(col, a2, b2);

            // (4) Measuring all qubits and resetting them in the Z basis before exiting
            for (index in 0..3) {
                set measurements[index] = MResetZ(qubits[index]);
            }
        }
    }
}
```
4.3. Q# IMPLEMENTATION

Chapter 4. Quantum Game Theory

// (5) Converting the Result array into a Boolean array for better manipulation in the Driver

let boolMeasurements =
    BoolArrFromResultArr (measurements);

return boolMeasurements;

It can be seen that this operation takes a row and column Int parameter, which have been pre-determined using a random number generator in the driver code. The 5 main steps which take place (as highlighted in the comments of the code) within the using block are outlined below:

1. An array of 4 qubits are each assigned a variable name using the let keyword. The first two are labelled a1 and b1, and belong to Alice, while the latter two, a2 and b2, are Bob's.

2. The qubits are entangled into two Bell State pairs using a Canon library operation named PrepareEntangledState(left : Qubit[], right : Qubit[]), which requires two arrays of qubits (of equal length) where all qubits are in the state |0⟩. It then creates a Bell State pair by taking the pair of qubits at each index of the two arrays, and entangling them as (|00⟩ + |11⟩)/2. This essentially creates a single entangled state as given by equation (4.4). The pair (a1, a2) are entangled first, followed by (b1, b2).

3. From this point, the qubit pairs belonging to Alice and Bob are passed to separate operations, illustrating the idea that they have separated as there is no way these pairs can interact with each other. The ApplyAliceCircuit(row : Int, a1 : Qubit, b1 : Qubit) operation takes the randomly selected row as input, as well as Alice's two qubits. It then applies the appropriate circuit to the qubits based on the row value. A similar procedure is applied to Bob's qubits using the col variable. The code showing how the six circuits in the game were implemented have been included in Appendix C.

4. Once the circuits have been run by both, each qubit must be measured. This is done by iterating through all qubits, and using the MResetZ(q : Qubit) operation supplied by the Canon API, which simply returns a value of type Result in the Pauli-Z basis (i.e. either a One or Zero). It then resets the qubit into the |0⟩ state before releasing it. These measurements are stored in a pre-specified mutable array of the same type.

5. Finally, the measurements array of type Result is converted into a Boolean array using the function BoolArrFromResultArr(input : Result[]). This converts elements as Zero -> false, One -> true. Returning a Boolean array allows
for easier manipulation in the classical driver code.

Diagrammatically, these steps can be illustrated using an overall circuit showing the journey taken by all the qubits. Figure 4.6 below shows this for the case where row = 1 and col = 3.

<table>
<thead>
<tr>
<th>Entangle</th>
<th>Isolate</th>
<th>Play</th>
<th>Win</th>
</tr>
</thead>
</table>

![Circuit Diagram](image)

**Figure 4.6:** An overall schematic showing each stage of the Q# code. The instance where the referee picks the row \( r = 1 \) and column \( c = 3 \) is highlighted.

**Simulation**

Below is a sample simulation of the game, where the row and column 1 and 3 have been randomly selected by the referee, respectively.

This simulation demonstrates the power of Quantum Pseudotelepathy! Quantum mechanical circuits are applied in the Mermin Peres Magic Square Game. Such a game has no classical winning strategy!

Press enter to see the game rules...

Rules:
- The referee selects a random row and column of a 3 x 3 grid.
- Alice is told her ROW, and covers it with 0's and 1's such that it has an EVEN PARITY.
- Bob is told his COLUMN, and covers it with 0's and 1's such that it has an ODD PARITY.
- (They can't communicate, but they can have pre-shared entangled qubits.)
- Alice and Bob win if their common cell has the same number on it.
- (Without entangled qubits they can't win CONSISTENTLY. Can they do it with?)

Press enter to run a game...

+-----------------------------------------------
| Ref picked row = 1, col = 3                  |
| Ref tells Alice the row and Bob the col.    |
| They each pick a circuit and run their qubits through...|
| Alice applies her circuit:                  |
| a1 -----.---.---x-----                   |
|   |   |   |
| b1 -----⊕-------⊕---x-----               |
| Simultaneously, Bob applies his circuit:    |
| a2 --------⊕-------------                |
|   |   |
| b2 --------.---H----------            |
| They measure the outputs of their circuits:|
| Alice's qubit measurements: 1, 1          |
| Bob's qubit measurements: 0, 1            |
| Press enter to see how they filled their grids...|
| The resulting grid:                      |
| B1  B2  B3                                |
| +-----------------+-----------------+-------------|
| A1  1  | 1  | 0/0  |                         |
| +-----------------+-----------------+-------------|
| A2  |   | 1  |                         |
| +-----------------+-----------------+-------------|
| A3  |   | 0  |                         |
| +-----------------+-----------------+-------------|
+-----------------------------------------------
| They Won!
The simulation first starts off by introducing the user to the game. Subsequently, the rules of the game are described as was done in Section 4.2. The user is then prompted to hit the enter key to start the game. Thereafter, the randomly picked row and column picked by the referee are displayed. The circuits that Alice and Bob use are then printed so that the inner workings of the strategy can be studied. In this case, the two circuits *Alice-Top* and *Bob-Right* have been applied. Finally, the measurements of all four qubits are shown. As can be seen, Alice's qubits measure to be 1,1, which means that her third cell must be filled with a 0 for an overall even parity. Similarly, Bob's qubits collapse into 0,1, so his third cell contains a 0, giving an odd parity. The simulation then goes to show how Alice and Bob have filled in the grid, with the intersection '0/0' showing the values entered by Alice and Bob respectively. The game then checks to see if it has been won, simply by probing the intersecting cell for an equivalence in the two values entered.

Interestingly, if the referee picks the same row and column in another simulation, the measurements made on the qubits and hence the grid don't always match. The output of another simulation is given below, where the measurements on \( a_1, b_1, a_2, b_2 \) are now \([0, 1, 1, 0]\) instead of \([1, 1, 0, 1]\), but the game is still won. This highlights the fact that the final wavefunction of the entanglement consists of a superposition of the winning states after all circuits have been applied. The fact that Q# is able to exhibit this behaviour in the qubits while providing a high level of abstraction emphasises that the average programmer can begin creating programs without having to worry about the finer details.

```
| Ref picked row = 1, col = 3 |
| Ref tells Alice the row and Bob the col. |
| They measure the outputs of their circuits: |
| Alice's qubit measurements: 0, 1 |
| Bob's qubit measurements:  1, 0 |
| Press enter to see how they filled their grids... |

<p>| The resulting grid: |</p>
<table>
<thead>
<tr>
<th>B1</th>
<th>B2</th>
<th>B3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
```
Validating the Correctness of the Solution

Although the example simulations in the previous section appear to show a working solution to the Magic Square game by using quantum pseudo-telepathy, a more rigorous methodology needs to be used to validate that it works in all cases. One way to do this would be to check the final state of the four qubits which make up the wavefunction to see if it matches the theoretical result. However, since Q# doesn’t provide users which direct access to the amplitude of qubit states, this method cannot be used.

Therefore, an alternative would be to run the simulation a large number of times for each of the nine cases (combination of row and column). This is indeed what was done. The simulation was run over all combinations of row and column, checking each time if the game had been won. Moreover, since each combination has a total of eight possible winning states (i.e. qubit measurement results), each case was run a total of 80 times such that every possible winning state was measured, on average, eight times.

A single loss would invalidate the correctness of the implementation. Since no loss was encountered, it can be concluded with confidence that the implementation of the Quantum Mermin-Peres game in Q# was successful.

4.4 Exploring Other Concepts

The Mermin-Peres Magic Square game is just one example of a game where a quantum strategy trumps the classical counterpart. There are a large range of other concepts within game theory that could also be explored. For instance, a concept known as the Impossible Colouring Game tasks two players with ‘colouring’ a set of vectors given to them while obeying some orthogonality conditions [75]. This is yet another game where strategies within Quantum Game Theory can be used to enhance the returns from playing the game.
Chapter 5

Conclusion

5.1 Outcomes of Project

The aim of this project was to analyse and study the multi-paradigm quantum programming language Q#, then create novel software which would be able to go beyond the capabilities of a classical program.

Initially, the main features and libraries available were studied and experimented with by analysing implementations of the Quantum Teleportation and Deutsch-Jozsa algorithms. Thereafter, an algorithm to solve an arbitrary $2 \times 2$ matrix system of linear equations was implemented, demonstrating a theoretical quantum speed-up with time complexities of order $O(\log N)$ compared to the classical $O(N)$ for sparse matrices. Running the algorithm for a particular matrix achieved results that were within the range of theoretical predictions.

Subsequently, as an extension to the project, concepts within Quantum Game Theory were explored. This led to the Mermin-Peres Magic Square game successfully being simulated in Q#; a game where no classical winning strategy exists, while a quantum strategy is able to win in all possible cases of the game.

5.2 Future Improvements

Although many aspects of the project were successful, there were a few obstacles. The main issue faced was the fact that Q# doesn’t allow the creation of operations in the form of matrices, having to resort to other methods for translating them into the principle quantum gates. This feature could be added by Microsoft in the future, as it would significantly reduce the amount of time required to implement algorithms.

If the project were to be extended further, then other algorithms could be investigated and implemented in Q#. This could be in the form of alternate implementations of the same baseline theoretical algorithm, or new algorithms entirely. Addi-
tionally, since just a single concept in Quantum Game Theory was studied, further examples could be simulated to demonstrate various aspects of quantum pseudo-telepathy.
Chapter 6

Ethical Considerations

All sections of the ethics check-list have been ticked with a 'No', as they don't relate to the project. Under section 9 of the check-list, titled Misuse, the first question asks 'Does your project have the potential for malevolent/criminal/terrorist abuse?'. Since quantum computing was being studied during this project, within which a field on quantum cryptography exists that has the potential to break classical cryptosystems, this question may be of particular concern. However, since such quantum systems haven't currently been implemented on a mass scale, on top of the fact that this project didn't focus on quantum cryptography, the chance of criminal activity is negligible.
Bibliography


[27] Wavefunction and its properties, 2018. URL http://hyperphysics.phy-astr.gsu.edu/hbase/quantum/wvfun.html. pages 8, 9


[34] Quantum the ai summit london,. URL https://theaisummit.com/london/home/quantum/. pages 12


[41] Introduction to ion trap quantum computing | university of oxford department of physics. URL https://www2.physics.ox.ac.uk/research/ion-trap-quantum-computing-group/intro-to-ion-trap-qc#. pages 22, 23


[43] Richard Li. Quantum annealing versus classical machine learning applied to

[44] Introduction to the d-wave quantum hardware, . URL https://www.dwavesys.com/tutorials/background-reading-series/introduction-d-wave-quantum-hardware#h2-0. pages 24


[76] URL https://commons.wikimedia.org/wiki/File:Mermin-Peres_magic_square.svg. pages 51


# Appendix A

## Ethics Checklist

<table>
<thead>
<tr>
<th>Section 1: HUMAN EMBRYOS/FOETUSES</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does your project involve Human Embryonic Stem Cells?</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Does your project involve the use of human embryos?</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Does your project involve the use of human foetal tissues / cells?</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section 2: HUMANS</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does your project involve human participants?</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section 3: HUMAN CELLS / TISSUES</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does your project involve human cells or tissues? (Other than from “Human Embryos/Foetuses” i.e. Section 1)</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section 4: PROTECTION OF PERSONAL DATA</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does your project involve personal data collection and/or processing?</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)?</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Does it involve processing of genetic information?</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc.</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets?</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section 5: ANIMALS</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does your project involve animals?</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

| Section 6: DEVELOPING COUNTRIES | Yes | No |
Chapter A. Ethics Checklist

<table>
<thead>
<tr>
<th>Section 7: ENVIRONMENTAL PROTECTION AND SAFETY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does your project involve the use of elements that may cause harm to the environment, animals or plants? ✓</td>
</tr>
<tr>
<td>Does your project deal with endangered fauna and/or flora /protected areas? ✓</td>
</tr>
<tr>
<td>Does your project involve the use of elements that may cause harm to humans, including project staff? ✓</td>
</tr>
<tr>
<td>Does your project involve other harmful materials or equipment, e.g. high-powered laser systems? ✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section 8: DUAL USE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does your project have the potential for military applications? ✓</td>
</tr>
<tr>
<td>Does your project have an exclusive civilian application focus? ✓</td>
</tr>
<tr>
<td>Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items? ✓</td>
</tr>
<tr>
<td>Does your project affect current standards in military ethics – e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons? ✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section 9: MISUSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does your project have the potential for malevolent/criminal/terrorist abuse? ✓</td>
</tr>
<tr>
<td>Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery? ✓</td>
</tr>
<tr>
<td>Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied? ✓</td>
</tr>
<tr>
<td>Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity related project? ✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section 10: LEGAL ISSUES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Will your project use or produce software for which there are copyright licensing implications? ✓</td>
</tr>
</tbody>
</table>
### Chapter A. Ethics Checklist

<table>
<thead>
<tr>
<th>Question</th>
<th>✓</th>
</tr>
</thead>
<tbody>
<tr>
<td>Will your project use or produce goods or information for which there</td>
<td>✓</td>
</tr>
<tr>
<td>are data protection, or other legal implications?</td>
<td></td>
</tr>
<tr>
<td><strong>Section 11: OTHER ETHICS ISSUES</strong></td>
<td></td>
</tr>
<tr>
<td>Are there any other ethics issues that should be taken into considera-</td>
<td>✓</td>
</tr>
<tr>
<td>tion?</td>
<td></td>
</tr>
</tbody>
</table>

**Table A.1:** Checklist of the project’s potential ethical implications.
Appendix B

Q# Code for Linear Equation Solver

Below is the Q# code that was written for the Linear Equation Solver algorithm, as discussed in Section 3.4. Each of the 6 main stages highlighted in Figure 3.4 have been indicated in the comments.

```qsharp
operation SolveLinearEquation (angle : Double) : Bool[]
{
    body
    {
        mutable measurements = new Result[2];

        using(qubits = Qubit[4]) {

            // Assigning qubit variables
            let q0 = qubits[0];
            let q1 = qubits[1];
            let b = qubits[2];
            // q3 is the ancillary qubit
            let q3 = qubits[3];

            // Setting the qubit b in the state
            cos(angle / 2)|0> + sin(angle / 2)|1>
            Ry(angle, b);

            /////////// Phase Estimation ///////////
            // (1) Preparing eigenvalue qubits
            H(q0);
            H(q1);

            // (2) Hamiltonian Simulation
            (Controlled ExpPiA)([q0], (b));
            (Controlled ExpHalfPiA)([q1], (b));
        }
    }
}
```
Chapter B. Q# Code for Linear Equation Solver

// (3) Inverse Quantum Fourier Transform
(Adjoint QFT)(BigEndian(qubits[0..1]));

// (4) R(lambda^{-1}) rotation ////
SWAP(q0, q1);

(Controlled Ry)([q0], (PI() / 4.0, q3));
(Controlled Ry)([q1], (PI() / 2.0, q3));

SWAP(q0, q1);

// (5) Inverse Phase Estimation //////
// Quantum Fourier Transform
QFT(BigEndian(qubits[0..1]));

// Inverse Hamiltonian Simulation
(Controlled Adjoint ExpHalfPiA)([q1], (b));
(Controlled Adjoint ExpPiA)([q0], (b));

H(q0);
H(q1);

// (6) Measuring the ancilla qubit; if it is One, then the qubit b is the correct state
set measurements[0] = MResetZ(q3);
// Measuring the qubit b, which is in the \( |x> \) state that solves the linear equation
set measurements[1] = MResetZ(b);

// Resetting the qubits q0 and q1 before releasing
ApplyToEach(Reset, qubits[0..1]);
}

// Converting the measurements array to a Boolean array for easier manipulation
let boolMeasurements = BoolArrFromResultArr(measurements);
return boolMeasurements;
Appendix C

Q# Code for Mermin-Peres Circuits

Below are the Q# implementations of the six circuits that were used in the Mermin-Peres Magic Square game simulation.

// Operation which applies the A1 matrix circuit to Alice's two qubits
operation AliceTop (a1 : Qubit, b1 : Qubit) : ()
{
    body
    {
        // A1 Matrix:
        // [ 1 0 0 i
        // 0 1 -i 0
        // 0 -i 1 0
        // i 0 0 1 ] / sqrt(2)
        CNOT(a1, b1);
        SPLIT(a1);
        CNOT(a1, b1);
        SWAP(a1, b1);
    }
}

// Operation which applies the A2 matrix circuit to Alice's two qubits
operation AliceMid (a1 : Qubit, b1 : Qubit) : ()
{
    body
    {
        // A2 Matrix:
        // [ 1 i i 1
        // 1 -i i -1
        // 1 i -i -1
        // 1 i -i -1
    }
}
Chapter C. Q# Code for Mermin-Peres Circuits

// i -i -i 1 ] / 2
SPLIT(a1);
CNOT(b1, a1);
H(b1);
CNOT(a1, b1);
}
}

// Operation which applies the A3 matrix circuit to
Alice's two qubits
operation AliceBot (a1 : Qubit, b1 : Qubit) : ()
{
  body
  {
    // A3 Matrix:
    // [ 1 1 1 -1
    // 1 1 -1 1
    // 1 -1 1 1
    // -1 1 1 1 ] / 2
    H(a1);
    CNOT(a1, b1);
    H(a1);
  }
}

// Operation which applies the B1 matrix circuit to
Bob's two qubits
operation BobLeft (a2 : Qubit, b2 : Qubit) : ()
{
  body
  {
    // B1 Matrix:
    // [ i -i 1 1
    // -i -i 1 -1
    // 1 1 -i i
    // -i i 1 1 ] / 2
    SPLIT(a2);
    CNOT(a2, b2);
    SPLIT(a2);
    DEC(a2, b2);
  }
}
// Operation which applies the B2 matrix circuit to
// Bob's two qubits
operation BobMid (a2 : Qubit, b2 : Qubit) : ()
{
    body
    {
        // B2 Matrix:
        // [ -1  i  1  i
        //  1  i  1 -i
        //  1 -i  1 i
        // -1 -i  1 -i ] / 2

        H(a2);
        CNOT(b2, a2);
        SPLIT(b2);
        INC(a2, b2);
    }
}

// Operation which applies the B3 matrix circuit to
// Bob's two qubits
operation BobRight (a2 : Qubit, b2 : Qubit) : ()
{
    body
    {
        // B3 Matrix:
        // [  1  0  0  1
        // -1  0  0  1
        //  0  1  1  0
        //  0  1 -1  0 ] / sqrt(2)

        CNOT(b2, a2);
        H(b2);
    }
}