Marshall University
Marshall Digital Scholar

Theses, Dissertations and Capstones

2019

Universal Quantum Computation

Junya Kasahara
kasahar1@marshall.edu

Follow this and additional works at: https://mds.marshall.edu/etd

 Part of the Mathematics Commons, Numerical Analysis and Computation Commons, and the Numerical Analysis and Scientific Computing Commons

Recommended Citation
Kasahara, Junya, "Universal Quantum Computation" (2019). *Theses, Dissertations and Capstones*. 1222.
https://mds.marshall.edu/etd/1222

This Thesis is brought to you for free and open access by Marshall Digital Scholar. It has been accepted for inclusion in Theses, Dissertations and Capstones by an authorized administrator of Marshall Digital Scholar. For more information, please contact zhangj@marshall.edu, beachgr@marshall.edu.

UNIVERSAL QUANTUM COMPUTATION

A thesis submitted to
the Graduate College of
Marshall University
In partial fulfillment of
the requirements for the degree of
Master of Arts
in
Mathematics
by
Junya Kasahara
Approved by
Dr. Carl Mummert, Committee Chairperson
Dr. JiYoon Jung
Dr. Huong Nguyen

MARSHALL UNIVERSITY
MAY 2019

APPROVAL OF THESIS/DISSERTATION

We, the faculty supervising the work of Junya Kasahara, affirm that the thesis, *Universal Quantum Computation*, meets the high academic standards for original scholarship and creative work established by the Department of Mathematics and the College of Science. This work also conforms to the editorial standards of our discipline and the Graduate College of Marshall University. With our signatures, we approve the manuscript for publication.

5/10/19

Dr. Carl Mummert, Department of Mathematics — Committee Chairperson — Date

5-8-2019

Dr. JiYoon Jung, Department of Mathematics — Committee Member — Date

05/06/2019

Dr. Huong Nguyen, Department of Physics — Committee Member — Date

ACKNOWLEDGEMENTS

I appreciate all the help to accomplish the thesis. First, I thank my family: my mother, father, and brother. Also, I thank Dr. Carl Mummert for his support. His discussion is always inspiring. I thank Dr. JiYoon Jung and Dr. Huong Nguyen for their participation in the committee. I thank all the faculty members who trained me for mathematics, and friends for their association and help.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

We study quantum computers and their impact on computability. First, we summarize the history of computer science. Only a few articles have determined the direction of computer science and industry despite the fact that many works have been dedicated to the present success. We choose articles by A. M. Turing and D. Deutsch, because A. M. Turing proposed the basic architecture of modern computers while D. Deutsch proposed an architecture for the next generation of computers called quantum computers. Second, we study the architecture of modern computers using Turing machines. The Turing machine has the basic design of modern computers despite its simple structure. Then we study quantum computers. Quantum computers are believed to be the next generation, and expected to have a breakthrough in processing speed. We study what makes quantum computers have such a high processing speed. Third, we study how quantum computers gain such a processing speed with an example of Shor's algorithm. This algorithm allows quantum computers to factor natural numbers into primes. Finally, we discuss a possible impact of quantum computers on the notion of computability.

CHAPTER 1

INTRODUCTION

This thesis studies quantum computers and their possible impacts on computability. Quantum computing is a technology that is expected to be one of the biggest breakthroughs in computer science and industry. Many techniques and approaches are being tested and developed concurrently. The cutting edge technologies of quantum computers are presently having their beginning [5].

1.1 Thesis structure

The thesis structure is as follows. Chapter 1 surveys the history of computer science with two articles. One was written by A. M. Turing [10], and the other was written by D. Deutsch [3]. We also study quantum mechanics, since a unification of computers proposed by A. M. Turing and quantum mechanics leads to the next generation of computers, called quantum computers, proposed by D. Deutsch.

Chapter 2 studies the design of computers with the article by A. M. Turing [10]. We call the present computers we are using now as *classical* computers to make a contrast between our present computers and the next generation of quantum computers throughout the dissertation. Then, we study how classical computers are reborn to be the next generation of computers called quantum computers, using the article by D. Deutsch [3]. The chapter gives analysis of how quantum computers change the classical computers' process mechanism.

Chapter 3 illustrates how quantum computers achieve an incomparable speed with the example of Shor's algorithm [8]. This algorithm uses a quantum computer to factor a natural number into primes. There, quantum computers show their strength. Classical computers take a long time to factor a large number; however, quantum computers take much less time to complete the task because of the superposition technique. We study how the algorithm uses the superposition method.

Chapter 4 argues that quantum computers will alter the concept of computability and discusses their application to cryptography.

1.2 Brief survey of computer history and quantum mechanics

Now, we survey the history of computers. The success of computers and the industry dates back to a mathematics article published in 1936. The mathematician A. M. Turing proposed an elementary model of classical computers called a Turing machine [10]. The concept of computability was proposed as well. He claimed that if a goal and an algorithm are given, then Turing machines can execute the same tasks as humans.

Around the same period, quantum mechanics was being born as well. Quantum mechanics is a subject describing the dynamics of elementary particles, the most fundamental building blocks and constituents of the universe. Unlike classical mechanics, quantum mechanics describes the tiny particles as not points, but waves. The waves are called wave functions. Now particle dynamics is not a movement of a point or dot, but a sum or combination of all possible states represented by wave functions. Each state is assigned a certain probability that tells how likely the state is to occur. The sum of all the wave functions of a particle is called a superposition. The unification of quantum mechanics and classical computers yields another breakthrough in the human history, called quantum computers [3].

Quantum computers are the next generation of computers with an incomparable processing speed. The concept originated from an unification of classical computers and quantum mechanics, proposed by D. Deutsch [3]. His concept was as follows. Quantum mechanics describes particle dynamics as a sum of all possible particle states. Such a sum is called a superposition. On the other hand, classical computers change their CPU and memory states as they execute programs. Deutsch proposed to use quantum mechanics to describe not only particle dynamics, but also CPU and memory dynamics as a superposition of all possible CPU and memory states, and compute the superposition. Thus, quantum computers calculate not individual cases one by one, but all possible cases and scenarios together in one computation.

That unique computation method is how quantum computers achieve an incomparable speed. The most fundamental theory describing nature is again quantum mechanics. Since they simulate quantum mechanics in such a precise way by an enormous speed, quantum computers can simulate any finite physical system. Here, simulation means that quantum computers can

compute all possible evolutions of the physical system and present the outcomes. If a quantum computer is given a proper algorithm and all initial conditions including a number of particles in the universe, then it is possible to simulate the entire universe.

CHAPTER 2

QUANTUM COMPUTERS

2.1 The history of computers

The computer industry is accelerating its development, and the next generation of computers, called quantum computers, is being studied. One of the primary developments of computers is processing speed. Quantum computers are expected to show an incomparable processing speed compared to the present ones in use. This section gives an overview of the history of computers and studies possible impacts quantum computers are about to bring.

Despite the successes and developments of the computer industry, only a certain number of mathematics articles have determined the direction of the industry in the past. This thesis studies two of the landmark works characterizing the history of computer science and the industry.

The first article was written by A. M. Turing [10]. The article proposed Turing machines, which are considered the first model of modern computers. Despite its simple structure, the Turing machine is a great model to study the design and framework of modern computers. Therefore, we study the Turing machines as a simple model of modern computers. We call modern computers as *classical computers* from now on.

The second article was written by D. Deutsch [3]. The article proposed a computer with a new concept of parallel computing, called a quantum computer. While classical computers perform calculations one by one, quantum computers use a superposition to perform all calculations together. Therefore, the processing speed is incomparable.

Since these two articles are the cornerstones of computer developments, this thesis studies the two works and discusses possible impacts generated by quantum computers.

2.2 The definition of quantum computers

We summarize how we construct quantum computers by three steps. First, we set up preliminaries formulating classical computer CPU and memory states in section 2.3. Second, we model the classical computer CPU and memory states dynamics in section 2.4. Third, we develop

classical computers into quantum computers in section 2.5. We employ the quantum mechanics formulation for two reasons. First, it is convenient to use these notations to represent activities of classical computers. Second, the quantum mechanics superposition formulation is vital for the new concept of parallel computing [3].

2.3 Quantum computer construction and preliminaries

We first model the complex activities of classical computers by a Turing machine and focus on CPU and memory only. Then, we formulate the CPU and memory activities by quantum mechanics. That approach is the exact path paving the way for the next generation of quantum computers. Before we discuss construction of a quantum computer, we review some historical background to justify our approach.

The historical background tells that A. M. Turing proposed a basic machine that computes with a given algorithm, called a Turing machine [10]. The architecture was simple compared to computers we are using currently, because it has only a head (CPU) and long paper tape (memory). Those elements are, however, the cores of the present computers, and the Turing machine is considered a basic model and birth of modern classical computers. Therefore, it is convenient to model classical computers as Turing machines since the machine has the important elements of CPU and memory; at the same time, it is simple and manageable enough to model.

Since we construct quantum computers by that approach, we need to establish three preliminaries: Turing machines, linear algebra, and quantum mechanics. Those are reviewed in the following sections.

2.3.1 Preliminary 1: Turing machines

A Turing machine is a basic model of classical computers. Despite its simple structure, the Turing machine can execute any computer program we are using for classical computers. Thus, the machine has all essential elements of classical computers [10].

A Turing machine consists of several elements. First, the machine has a tape. The tape is a memory and acts as a recording device. Thus, the tape records a sequence of zeros and ones. Each location, which has zero or one, is called a cell. Each zero or one represents one bit of information. Second, the machine has a head. The head models a CPU, and moves left or right

and writes down zero or one on each cell of a long tape of paper. Third, the machine has a state table. The table is equivalent with programs of classical computers. The table lists a finite number of predetermined mechanical steps or instructions that tell the machine how to proceed inputs. Fourth, the machine has a state register. The register indicates a state of the machine. For instance, if the state register shows the special value of 1, then the machine has completed a task and halted already. The machine is always in one of a finite number of states. The state table tells how to change that cell (or leave it unchanged), and whether to move the head one cell left or right, and how to change one state to another [10].

2.3.2 Preliminary 2: Linear algebra

We employ the concepts of basis and eigenvectors to define quantum and classical computers. Suppose that we have the infinite dimensional vector space called a Hilbert space, and there we need to construct a vector $\vec{a}$ representing a position of a particle in the 3-D space. The following elements formulate the position in the Hilbert space [4].

The first preliminary is *basis vectors*. Suppose that we have a vector $\vec{a}$ in 3-D space as

$$\vec{a} = a_1\hat{x} + a_2\hat{y} + a_3\hat{z},$$

where a_1, a_2, and a_3 are real number coefficients, and $\hat{x} = (1,0,0), \hat{y} = (0,1,0)$, and $\hat{z} = (0,0,1)$ are basis vectors. The basis vectors have the usual orthonormal relations, which means that,

$$\hat{x}\cdot\hat{x} = \hat{y}\cdot\hat{y} = \hat{z}\cdot\hat{z} = 1 \text{ and} \tag{2.1}$$

$$\hat{x}\cdot\hat{y} = \hat{x}\cdot\hat{z} = \hat{y}\cdot\hat{z} = 0. \tag{2.2}$$

Here, recall that we have the freedom to choose any basis vector set to represent the vector, $\vec{a}$, as long as the vector set spans the 3-D space. Thus, it is possible to choose another unit vector set, $\{\hat{\alpha}, \hat{\beta}, \hat{\gamma}\}$, with a new coefficient set, $\{b_1, b_2, b_3\}$ to represent the same vector, $\vec{a}$, as

$$\vec{a} = b_1\hat{\alpha} + b_2\hat{\beta} + b_3\hat{\gamma}. \tag{2.3}$$

A choice of basis vectors is one important decision in quantum mechanics; thus we discuss some more of the perspective next. Here, we change the notation from the linear algebra fashion to the quantum mechanical fashion as

$$\vec{x} \to |x\rangle. \tag{2.4}$$

If we have $|x\rangle = (x_1, x_2, \cdots, x_n)$ and $|y\rangle = (y_1, y_2, \cdots, y_n)$, then the inner product of $|x\rangle$ and $|y\rangle$ is defined as

$$\langle y|x\rangle \equiv \sum_{i=1}^{n} y_i^* x_i,$$

where y_i^* is a complex conjugate of y_i. Suppose that $|x\rangle$, $|y\rangle$, and $|z\rangle$ have the length of one, then, the orthonormal property remains still for the basis vectors as

$$\hat{x} \cdot \hat{x} \equiv \langle x|x\rangle = 1, \quad \hat{y} \cdot \hat{y} \equiv \langle y|y\rangle = 1, \quad \hat{z} \cdot \hat{z} \equiv \langle z|z\rangle = 1, \tag{2.5}$$

where $(|x\rangle)^*$ is a complex transpose of $|x\rangle$, and a complex conjugate relation holds as $\langle x| \equiv (|x\rangle)^*$. The vector $\langle x|$ is called a bra-vector and $|x\rangle$ is called a ket-vector [4].

The next preliminary is *eigenvectors*. If we have a freedom to choose basis vectors, then it is beneficial to choose a basis of eigenvectors. Recall eigenvectors and eigenvalues in linear algebra. For instance, if we apply an eigenmatrix H for an eigenvector $|E_1\rangle$, then we have an eigenvalue E_1. The formulation is described as

$$H|E_1\rangle = E_1|E_1\rangle. \tag{2.6}$$

We label an eigenvector by its eigenvalue; i.e. we denote the eigenvector as $|E_1\rangle$, because it has the eigenvalue E_1. Also,we change our notation from $\vec{E_1}$ to $|E_1\rangle$ as is usual usage of quantum mechanics.

The next preliminary is the concept of *state* in quantum mechanics. We construct a quantum mechanics state now. Suppose that our state $|E\rangle$ has an unknown energy E, however, we know that the quantum state has, say two possible energies E_1 and E_2. Then we express the

quantum state $|E\rangle$ by a superposition of the two eigenvectors $|E_1\rangle$ and $|E_2\rangle$ as

$$|E\rangle = \lambda_1|E_1\rangle + \lambda_2|E_2\rangle. \tag{2.7}$$

The coefficients λ_1, λ_2 are associated with the eigenvectors $|E_1\rangle, |E_2\rangle$ respectively. Here, the coefficients must satisfy the property

$$\sum_m |\lambda_m|^2 = 1. \tag{2.8}$$

This is because we would like to use quantum state vectors as a probability density function later, as the usual convention of probability theory. Thus, the length of the vector needs to be 1. Such a coefficient adjustment calculation is called normalization. After normalization, each λ_m represents the square root of the probability of $|E_m\rangle$. We will present this process in more detail later.

We now give an example of how the Hamiltonian operator is used. If we would like to know a mean value or an expectation value of our quantum state energy μ, then we apply the energy operator H called the *Hamiltonian* for the quantum state vector from equation (2.7).

$$H|E\rangle = \lambda_1 H|E_1\rangle + \lambda_2 H|E_2\rangle. \tag{2.9}$$

Since $|E_1\rangle$, $|E_2\rangle$ are eigenvectors of H as discussed in equation (2.6), we obtain the eigenvalues, E_1 and E_2 as

$$H|E\rangle = \lambda_1 E_1|E_1\rangle + \lambda_2 E_2|E_2\rangle. \tag{2.10}$$

That is one advantage to choosing eigenvectors $|E_1\rangle$, $|E_2\rangle$ to represent $|E\rangle$. As soon as we apply the Hamiltonian operator, we can tell that the eigenvalues are E_1 and E_2. Now we apply a ket vector for equation (2.10), then we have

$$\langle E|H|E\rangle = \langle E|(\lambda_1 E_1|E_1\rangle + \lambda_2 E_2|E_2\rangle). \tag{2.11}$$

Since a dot product of a pair of the eigenvectors is one or zero, we calculate an expectation value of energy μ as

$$\mu = \langle E|H|E\rangle = |\lambda_1|^2 E_1 + |\lambda_2|^2 E_2. \tag{2.12}$$

Again, equation (2.12) is a mean value or an expectation value μ as we see in a probability and statistics course. Here, the bra-ket vectors play the role of a probability density function to compute a mean value of the operator: H.

2.3.3 Preliminary 3: Quantum mechanics

Quantum mechanics describes evolutions and interactions of particles; however, the field plays two important additional roles in this thesis. First, we use its notations to describe the CPU and memory activities of classical computers. Second, its concept of superposition is the key for parallel quantum computing.

We would like to introduce some examples of quantum mechanics briefly so that the readers can familiarize themselves with the field. One example is that particle energies are allowed to be certain discrete numbers only. Such a discreteness is called energy quantization.

Here, we present two examples to facilitate understanding of the subject before we start a mathematical formalization. Meanings and implications of quantum mechanics are controversial still; thus, we use only arguments agreeable by majority of the physics community [9].

The first example is an electron energy state in a hydrogen atom. The atomic composition of hydrogen is as follows. An electron is orbiting around the atomic nucleus that consists of a proton. Then, the electron is restricted to certain orbits represented by n and energies E_n, which have the formula, with the unit of electron volt [eV],

$$E_n = -\frac{13.6}{n^2} \quad [eV], \quad n = 1, 2, 3, \cdots . \tag{2.13}$$

The orbit is counted with $n = 1$ from the closest to the nucleus. The enegies are negative numbers, because an energy input is necessary to free the electron.

The second example is a particle trapped between two potential walls, and its image is as follows. Suppose there are two high potential energy walls. Since a particle does not have kinetic energy bigger than those potential energies, the particle is trapped and bouncing back and forth between them. These particle dynamics can be viewed as a standing wave.

Recall a standing wave generated by a string vibration. If we hold both ends of the string and shake it fast enough, then the string looks stationary. The ends don't move at all in your

hands, while the middle of string keeps the biggest and constant vertical displacement. Since this kind of vibration has one antinode in the middle and two nodes at both ends, the standing wave has the lowest frequency and energy. If you increase the frequency, the next standing wave will have two antinodes and three nodes. This illustration of antinodes and nodes in a standing wave is one way to understand that the particle energies are discretized between potential walls [4].

Suppose that we have the Hilbert space for a particle energy trapped between two potential barriers as discussed in the second example above, and there we need to construct a function $\psi_i(x)$ representing a quantum mechanics state of the particle. We now describe how to formulate it in the Hilbert space.

The first step in the formulation is the choice of a *basis*. If we consider a position vector as discussed previously, then the most useful convention is to choose the basis vectors of $\hat{x} = (1, 0, 0), \hat{y} = (0, 1, 0)$, and $\hat{z} = (0, 0, 1)$ to span the 3-D space in linear algebra. Thus, we need basis functions to span a quantum mechanical state. Here, we present one example regarding how to establish basis functions.

We start with the Schrödinger equation

$$-\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V\psi = E\psi, \tag{2.14}$$

where $\hbar$ is the Planck constant divided by 2π, m is a particle mass, x is a particle position, V is a potential energy, E is an energy of the particle, and ψ is a wave function of the particle. If a particle is free between two potential barriers, then we set $V = 0$ and solve the equation. We have the solution

$$\psi_i(x) = A\exp\left(-\frac{i\sqrt{2mE_i}}{\hbar}x\right) + B\exp\left(\frac{i\sqrt{2mE_i}}{\hbar}x\right), \quad i = 1, 2, 3, \cdots, \tag{2.15}$$

where A, B are unknown coefficients. As discussed previously, the energy, E_i, is discretized and equation (2.15) is our basis function.

Our primary interest is a mean or expectation value as an experimental result in quantum mechanics. Thus, it is a more convenient formalization to choose *eigenfunctions* to construct a quantum mechanics state. One example is that equation (2.15) is an eigenfunction to construct a

quantum mechanics state. If we apply the Hamiltonian operator H for equation (2.15), then we have the eigenvalue E_i. The energy E_1 is from the ground state $\psi_1(x)$ that has one antinode and two nodes in the second example above, while E_2 is from the second state $\psi_2(x)$ with two antinodes and three nodes.

Now, we discuss how to construct a *quantum state* with a superposition of the eigenfunctions $\psi_i(x)$. A concept of that superposition is one of the exact keys toward quantum computers, and we will discuss more details after this section. The following is a process we need.

First, choose possible eigenfunctions and build up a linear combination. Suppose that we have two possible quantum mechanics states $i = 1, 2$ in this example. Then we assemble a quantum mechanics state by a linear combination of the eigenfunctions of $\psi_1(x), \psi_2(x)$ as

$$\psi(x) = C\psi_1(x) + D\psi_2(x), \tag{2.16}$$

where the coefficients C and D are yet to be determined.

Second, determine the unknown coefficients C and D. We will use a square of the quantum mechanics state function (2.16) as a probability density function. In other words, if we multiply $\psi(x)$ with its complex conjugate $\psi^*(x)$, then we have a probability density function

$$\text{a probability density function} = \psi^*(x) \cdot \psi(x) \tag{2.17}$$

Equation (2.17) shows a property of the function $\psi(x)$. The property is that the function needs to satisfy the condition

$$\int_{all} \psi^*(x) \cdot \psi(x) dx = 1. \tag{2.18}$$

Equation (2.18) detemines the unknown coefficients of C, D. The process is called *normalization*.

Third, we have the new function of quantum mechanical state

$$\psi(x) = \lambda_1\psi_1(x) + \lambda_2\psi_2(x), \quad \sum_i |\lambda_i|^2 = 1. \tag{2.19}$$

Now, the state function is ready to use as a probability density function.

We use an *evolution operator* to formulate a system evolution in discrete time steps. Suppose that a physical system is at the initial condition of $t = 0$, which is denoted as $|\psi(t = 0)\rangle$. A quantum mechanical system at an arbitrary time t is represented by an evolution operator U with the initial state $|\psi(t = 0)\rangle$ as

$$|\psi(t)\rangle = U(t, t_0)|\psi(0)\rangle. \tag{2.20}$$

Here, $U(t, t_0)$ is the evolution operator with the form

$$U(t, t_0) = \exp\left(\frac{-iH(t - t_0)}{\hbar}\right), \tag{2.21}$$

where H is a Hamiltonian, t_0 is an initial time ($t_0 = 0$ in this example), and $\hbar$ is the Planck constant divided by 2π. Since the evolution operator is unitary, the following property will hold

$$U^+U = UU^+ = \mathbb{I}, \tag{2.22}$$

where U^+ is a complex transpose of U with the form

$$U^+(t, t_0) = \exp\left(\frac{iH(t - t_0)}{\hbar}\right), \tag{2.23}$$

and $\mathbb{I}$ is the identity.

A mean or expectation value is of primary interest as an observable result in physics. For instance, a mean value of energy in the quantum state above is evaluated by the Hamiltonian operator H for the quantum state and integrate over all the domain

$$\mu = \int_{all} \psi^*(x) H \psi(x) dx. \tag{2.24}$$

Equation (2.24) gives a mean value of energy μ.

2.4 Construction step 2: Modeling classical computers

In this section, we construct a classical computer and describe how to execute a program. The quantum mechanical formulation will be used. The following is the outline of this section. The computer construction relies on the approach of Deutsch [3], which this thesis follows closely.

First, we define three parameters identifying a classical computer state. The parameters will represent a scanning head location x, a finite processor state $\vec{n}$, and an infinite memory state $\vec{m}$. Second, we define computer eigenstates with the three parameters studied at the first subsection. Third, we define a computer eigenstate evolution. The evolution describes a classical computer state program execution. Fourth, we define a state initialization. The initialization step must be completed before a program execution. Fifth, we describe a program execution and how a classical computer returns to the initial state.

2.4.1 The three variables identifying a classical computer state

We need three parameters determining our computer state: the finite processor $\vec{n}$, the infinite memory (infinitely long tape) $\vec{m}$, and the currently scanned tape location x. We have illustrated how to formulate a position vector in linear algebra and a state function in quantum mechanics. Now, we need a state vector to represent a classical computer state, and three variables to define an eigenvector to construct the state vector. We discuss the variables as follows.

The first variable is $\vec{n}$ for the *finite processor*. The processor consists of M cells, and each has two possible states, 0 and 1 which each represents a one bit signal. Each cell is labelled n_i as

$$\text{finite processor:} \quad \vec{n} = (n_0, n_1, n_2, \cdots, n_{M-1}). \tag{2.25}$$

The vector notation of $\vec{n}$ represents the sequence of n_i.

The head and state register of a Turing machine models the finite processor. The state register stores the state of the computation. It moves left or right along an infinitely long tape, and writes down 0 and 1 onto cells of the tape. The bits of zeros and ones represent a status of the finite processor.

The second variable is $\vec{m}$ for the *infinite memory*. The infinite memory consists of

infinitely many cells, and each cell is labelled m_i as

$$\text{memory:} \quad \vec{m} = (\cdots, m_{-2}, m_{-1}, m_0, m_1, m_2, \cdots). \tag{2.26}$$

The infinitely long tape of a Turing machine models the infinite memory, which is subdivided into cells. The cells are labelled as

$$(\cdots, m_{-2}, m_{-1}, m_0, m_1, m_2, \cdots). \tag{2.27}$$

Each cell, m_i, has two possible states 0 and 1, which correspond to a digital signal of on and off respectively. That there are only two states is analogous with the two values of fermion spin in quantum mechanics. Fermions have a spin up $+1$ or down -1. The fact that the fermions have only ± 1 spin states can be formulated by the spin rotation matrices, and the matrices are used for quantum computers as well. We explain some more detail later. Therefore, each cell of memory m_i has the state as

$$m_i = 0 \quad \text{or} \quad m_i = 1, \quad \forall i \in \mathbb{Z}. \tag{2.28}$$

As a result, the infinite memory will be an infinitely long sequence of 0 and 1, written on an infinitely long tape. The vector notation of $\vec{m}$ represents the sequence of m_i.

The third variable is x for the *currently scanned tape location*. The currently scanned tape location is represented by the variable x as

$$\text{currently scanned tape position } x: \qquad (\cdots, -2, -1, 0, 1, 2, \cdots) \tag{2.29}$$

The variable x corresponds to the index i of m_i. Thus, if $x = 4$, then the scanned tape location is the cell of m_4 currently in the infinitely long tape. Since the tape has an infinite length, the variable x can be any integer representing each cell in the infinitely long tape modeling the infinite memory.

We represent eigenvectors of a classical computer state with the above three variables as

$$|x; \vec{n}; \vec{m}\rangle \equiv |x; n_0, n_1, \cdots, n_{M-1}; \cdots, m_{-1}, m_0, m_1, \cdots\rangle, \tag{2.30}$$

with the orthonormal property of

$$\langle x_2; \vec{n}_2; \vec{m}_2 | x_1; \vec{n}_1; \vec{m}_1 \rangle \equiv \delta_{x_1 x_2} \delta_{\vec{n}_1 \vec{n}_2} \delta_{\vec{m}_1 \vec{m}_2}, \tag{2.31}$$

where δ is a Kronecker delta symbol. Recall that $|x_1; \vec{n}_1; \vec{m}_1\rangle$ is an unit vector identified by its labeling of $x, \vec{n}, \vec{m}$; thus, its inner product with itself will be one. That its own inner product is an unity is the same requirement we impose onto the basis vectors as discussed in equation (2.1).

We define a classical computer eigenstate evolution. The evolution of the eigenstates over time is formulated by an evolution operator. As the evolution operator is discussed in the quantum mechanics preliminary, a dot product of eigenvectors define an evolution operator.

2.4.2 Evolution operators

A classical computer state evolution is formulated by an operator called the evolution operator U. The evolution operator has certain properties as follows.

First, the evolution operator is a constant operator, and its time interval is constant. That time interval is the exact time length that the scanning head moves one cell to the next cell left or right. If an evolution is a n-step process, then we need to use the operator n times. Thus, we will have

$$\underbrace{UUU \dots U}_{n \text{ applications}} = U^n. \tag{2.32}$$

The evolution operator can be understood as follows. Applying the evolution operator n times means that time elapses by the amount of nT, where T is the minimum time interval by which the scanning head moves. Thus, a computer state $|\psi(t = nT)\rangle$ shows that the time amount of nT has passed since the initial state $|\psi(t = 0)\rangle$, which can be formulated with the evolution operator as

$$|\psi(t = nT)\rangle = U^n |\psi(t = 0)\rangle \quad (n \in \mathbb{Z}^+). \tag{2.33}$$

Second, the evolution operator is an unitary operator. Thus, the following property holds

$$U^+ U = U U^+ = \mathbb{I}, \tag{2.34}$$

where $\mathbb{I}$ is the identity matrix, and U^+ is the complex transpose of U.

Third, the evolution operator changes a currently scanned tape location in a computer state. Since the scanning head is in motion always, if time T elapses, then the head will be in an adjacent cell either right or left. Thus, conversely speaking, if the evolution operator is applied, then time elapses with the same amount of T, by which the scanning head moves. Therefore, the operator changes a location of the scanning head forward or backward on the infinitely long tape.

Since the evolution operator moves the scanning head right or left, there are two possible representations for the operator,

$$U \rightarrow \ U^F \ \text{ or } \ U^B, \tag{2.35}$$

where the operator U^F moves the scanning head forward (right) while the operator U^B moves the scanning head backward (left). Since the scanning head is in motion always in the infinitely long tape, the evolution operator has no sub-operator representing the head static. The operator has only two sub-operators representing two motions.

We consider how the sub-operators U^F, U^B change a computer eigenstate $|x;\vec{n};\vec{m}\rangle$. The evolution operator U^F moves the scanning head of computer forward. Thus, we have

$$\text{Forward motion:} \quad U^F|x;\vec{n};\vec{m}\rangle = |x+1;\vec{n}';\vec{m}'\rangle. \tag{2.36}$$

The evolution operation U^B moves the scanning head of computer backward. Thus, we have

$$\text{Backward motion:} \quad U^B|x;\vec{n};\vec{m}\rangle = |x-1;\vec{n}';\vec{m}'\rangle. \tag{2.37}$$

Fourth, the evolution operator can be represented as a matrix. If the evolution operator is applied for a computer state $|x;\vec{n};\vec{m}\rangle$ once, then a new computer state will be

$$U|x;\vec{n};\vec{m}\rangle. \tag{2.38}$$

Now, it is evaluated the probability that the new computer state: $U|x;\vec{n};\vec{m}\rangle$ is $|x';\vec{n'};\vec{m'}\rangle$. Then,

the bra vector $\langle x'; \vec{n'}; \vec{m'}|$ is applied for the new state, thus, we obtain

$$|\langle x'; \vec{n'}; \vec{m'}|U|x; \vec{n}; \vec{m}\rangle|^2. \tag{2.39}$$

Thus, the dot product of the two computer states is the matrix representation of evolution operator as

$$U^{F,B}(x'; \vec{n'}; \vec{m'}|x; \vec{n}; \vec{m}) = \frac{1}{2}\delta_{\vec{n'}}^{A(\vec{n},\vec{m})}\delta_{\vec{m'}}^{B(\vec{n},\vec{m})}[1 \pm C(\vec{n}, \vec{m})], \tag{2.40}$$

where A, B, C are functions with the ranges of $(\mathbb{Z}_2^M), \mathbb{Z}_2, \{-1, 1\}$ respectively. We discuss the conversion factor $\frac{1}{2}[1 \pm C(\vec{n}, \vec{m})]$ more. We have the motivation to incorporate the factor as follows.

We would like to use the spin rotation operator of quantum mechanics to formulate one bit signal of zero and one so that a matrix element $U^{F,B}(x'; \vec{n'}; \vec{m'}|x; \vec{n}; \vec{m})$ will be zero or one at equation (2.40). For that purpose, we reset the fermions' spin $+\frac{1}{2}\hbar$ to be $+1$, and $-\frac{1}{2}\hbar$ to be -1. The conversion factor $C(\vec{n}, \vec{m})$ will have those ± 1 as inputs, and $U^{F,B}(x'; \vec{n'}; \vec{m'}|x; \vec{n}; \vec{m})$ will be zero or one. At the same time, the spin rotation operators will formulate a switch between ± 1.

2.4.3 Initializing a computer eigenstate

Once the computer is activated, then it starts running the program, and the process is formulated by repeatedly applying the evolution operator U to the computer state.

Before the processor starts proceeding any information, a computer state $\psi(t)$ must be initialized. Then, the following steps must be proceeded for initialization. First, the time is set to be zero $t = 0$. Second, the currently scanned tape location will be at the initial starting position $x = 0$. Third, the finite processor is ready to start $\vec{n} = \vec{0}$. Fourth, the memory state is set to be $\vec{m} = \vec{0}$. Thus, the initial state of the computer is a linear superposition of the eignvectors of (2.30)

$$|\psi(t = 0)\rangle = \sum_m \lambda_m |x = 0; \vec{n} = \vec{0}; \vec{m} = \vec{0}\rangle, \tag{2.41}$$

Be aware again that a complex square of $\psi(t)$ is a probability function. Thus, the following

condition must be satisfied,

$$\sum_m |\lambda_m|^2 = 1. \tag{2.42}$$

2.4.4 Classical computer program executions

Now, after the initialization process is complete, the computer executes a program. Recall that a computer state is identified by the three parameters $x, \vec{n}, \vec{m}$ defined in section 2.4.1, of which each is separated by a semicolon in the ket vector as

$$| \underbrace{x}_{position} ; \underbrace{\vec{n}}_{processor} ; \underbrace{\vec{m}}_{memory} \rangle. \tag{2.43}$$

Classical computers receive inputs and a program, then yield outputs in a program execution. Suppose that a computer executes a computable function f. We can make a program executing the computable function f. Let's call the program $\pi(f)$. The program $\pi(f)$ receives the inputs i, and generates outputs $f(i)$. Recall that the finite processor consists of the cells labelled between n_0 and n_{M-1}, while the infinite memory consists of an infinite number of cells m_i as in equation (2.26).

We need to *initialize* the computer eigenstate before a program execution. Thus, we follow the discussion in section 2.4.3. First, we set the currently scanned tape location x to be 0 as

$$x : x \to 0. \tag{2.44}$$

Second, we set the finite processor to be the initial condition as

$$\vec{n} : \vec{n} \to \vec{0}. \tag{2.45}$$

Here, $\vec{0}$ represents a sequence of zeros.

Third, we store the program $\pi(f)$ and input i onto the infinite memory. We have the

initial condition that all the infinite memory cells have zeros, i.e.

$$\vec{m} = \vec{0} = (\cdots, \underbrace{0}_{m_{-2}}, \underbrace{0}_{m_{-1}}, \underbrace{0}_{m_0}, \underbrace{0}_{m_1}, \underbrace{0}_{m_2} \cdots). \tag{2.46}$$

Then, we need to allocate some part of the infinite memory $\vec{m}$ to the program $\pi(f)$ and input i. Suppose that the program $\pi(f)$ is recorded at the cells between m_{-100} and m_0, and the inputs i have the cells between m_1 and m_{10}. Then, the memory $\vec{m}$ has the new state as

$$\begin{aligned} \vec{m} &= (\cdots, \underbrace{0}_{m_{-101}}, \underbrace{m_{-100}, \cdots, m_0}_{\text{program:}\pi(f)}, \underbrace{m_1, \cdots, m_{10}}_{\text{input:}i}, \underbrace{0}_{m_{11}}, \cdots) \\ &\equiv (\pi(f), i, \vec{0}). \end{aligned} \tag{2.47}$$

Be aware that the rest cells of the infinite memory have a null value represented by $\vec{0}$. Now, we record the program $\pi(f)$ and input i onto the infinite memory, and we have the new memory state as

$$\vec{m} : \vec{0} \to (\pi(f), i, \vec{0}). \tag{2.48}$$

We modify the eigenstate (2.43) with the above change as

$$|x; \vec{n}; \vec{m}\rangle \to |\underbrace{0}_{x}; \underbrace{\vec{0}}_{\vec{n}}; \underbrace{\pi(f), i, \vec{0}}_{\vec{m}}\rangle. \tag{2.49}$$

Now, equation (2.49) is the eigenstate when the program $\pi(f)$ starts with the input i.

Once the computer has been initialized, we can execute the program. The remainder of this section summarizes how the computer state changes at each step of execution.

As the program $\pi(f)$ is executed, the computer changes its CPU and memory states step by step. Its constant change is described as an evolution of the computer state (2.49) with the evolution operator U. Suppose that the outputs $f(i)$ are recorded on the cells between m_{11} and

m_{20} in the infinite memory. Then, the new memory state is as

$$\begin{aligned}\vec{m} &= (\cdots, \underbrace{0}_{m_{-101}}, \underbrace{m_{-100}, \cdots, m_0}_{\text{program:}\pi(f)}, \underbrace{m_1, \cdots, m_{10}}_{\text{input:}i}, \underbrace{m_{11}, \cdots, m_{20}}_{output:j}, \cdots) \\ &\equiv (\pi(f), i, f(i), \vec{0}). \end{aligned} \tag{2.50}$$

Then, the execution of the program is represented as an evolution of the computer state (2.49) with applying the evolution operator U n times. Thus, we have the equation as

$$U^n \, |\underbrace{\underbrace{0}_{x}; \underbrace{\vec{0}}_{\vec{n}}; \underbrace{\underbrace{\pi(f)}_{program}, \underbrace{i}_{inputs}}_{\vec{m}}}_{\text{Initial state}}, \vec{0}\rangle = |\underbrace{\underbrace{0}_{x}; \underbrace{1, \vec{0}}_{\vec{n}}; \underbrace{\underbrace{\pi(f)}_{program}, \underbrace{i}_{inputs}, \underbrace{f(i)}_{outputs}}_{\vec{m}}}_{\text{Final state}}, \vec{0}\rangle \, . \tag{2.51}$$

Recall that the inputs, program, and outputs are recorded as a sequence of one bit signals, 0 and 1, on the infinite memory $\vec{m}$. Also, be aware at equation (2.51) that the CPU state $\vec{n}$ changes as

$$\vec{n} = (0, \cdots, 0) \rightarrow \vec{n} = (1, 0, 0, \cdots, 0) \tag{2.52}$$

In other words, the new CPU state has 1 at the first cell. That is a signal indicating that the program execution is complete.

2.4.5 Issues in constructing quantum computers

We would like to build a quantum computer; however, we need to resolve some technical issues.

- Issue 1: We need to make a program for quantum computers to execute. We construct the program $\pi(f)$ that uses a computable function f, receives the input i at slot 2, and generates its output $f(i)$ to slot 3 in this example.

- Issue 2: Each cell changes its stored value as the program is executed. We use the spin rotation matrices to formulate the process.

- Issue 3: A computer state must return to the initial state after a program execution. In

other words, the state must be in a sequence of null values except a new result of the program execution.

For issue 1, we need to make a program for quantum computers to execute. We will discuss two programs. One is the executable program $\pi(f)$ executing the computable function f. The other is its subprogram ϕ which main role is to take a value at a slot, use it as an input for a computable function g, and replace it with the function's outcome.

First, we discuss the executable program $\pi(f)$ and the computable function f. It accepts an input and generates an output. Now, we suppose that the program has the input i at slot 2, and generates the output $f(i)$ at slot 3. Here, we name a group of memory cells a slot. Each cell holds one bit of information zero or one; however, one cell does not have enough capacity to store all information necessarily. For instance, if you would like to store the value of 3, then the number is shown as 11 in a binary expression. Thus, we need two cells to store 3. To store information, CPU regroups cells of $2^1, 2^2, 2^3$ and so on at an infinite amount of cells, and such a group of memory cells is called a slot. In the previous example, we assigned the input i to the memory cells between m_1 and m_{10}. We call the memory cells as slot 2, and slot 3 is a group of the memory cells between m_{11} and m_{20}. Slot 1 is a group of the memory cells, labelled between m_{-100} and m_0, and stores the program $\pi(f)$. The executable program has the notation $\pi(f)$, and it is modified with the new information of inputs and outputs as

$$\pi(f) \to \pi(f, 2, 3) \tag{2.53}$$

Now, the program has the new notation $\pi(f, 2, 3)$, where 2 represents the input slot while 3 represents the output slot.

The computer eigenstate of equation (2.51) has the new notation, with the new program notation $\pi(f, 2, 3)$, as

$$|\underbrace{0}_{x};\underbrace{1, \vec{0}}_{\vec{n}};\underbrace{\pi(f, 2, 3), i, f(i), \vec{0}}_{\vec{m}}\rangle \tag{2.54}$$

Equation (2.54) has all the information of $x, \vec{n}, \vec{m}$ explicitly. For now, we abbreviate the notation of equation (2.54) since we are focusing on activities of the infinite memory only. Thus, we have

the notation

$$|0;1,\vec{0};\underbrace{\pi(f),i,f(i),\vec{0}}_{\vec{m}}\rangle = |\overbrace{\underbrace{\pi(f,2,3)}_{\text{slot 1}},\overbrace{i}^{\text{slot 2}},\overbrace{f(i)}^{\text{slot 3}}}^{\vec{m}}\rangle. \tag{2.55}$$

The program $\pi(f)$ receives an input i, executes a computable function f, yields an output $f(i)$, and resets $x, \vec{n}, \vec{m}$ to the initial condition. If it is a n-step process, then the evolution operator can express the program execution as

$$U^n|0;\vec{0};\pi(f),i,\vec{0}\rangle = |0;1,\vec{0};\pi(f),i,f(i),\vec{0}\rangle. \tag{2.56}$$

The program $\pi(f,2,3)$ receives an input i at slot 2, executes the computable function f, generates an output $f(i)$ to slot 3, leaves slot 2 unchanged, and resets the rest of computer state to the initial state. Since the function f is computable, the program output $f(i)$ is added on a previously stored value at slot 3. For instance, if slot 3 has some non-zero sequence j initially, then the initial value j should not be just replaced by, but combined with a new output $f(i)$. Thus, the output result is expressed as

$$\text{output slot } j \to j \oplus f(i), \tag{2.57}$$

where $\oplus$ is any associative and commutative operator following the algebra rule

$$\oplus \text{ algebra rule: } \left.\begin{array}{r} i \oplus i = 0, \\ \\ i \oplus 0 = i, \end{array}\right\}. \tag{2.58}$$

Thus, a computer eigenstate evolves with the program execution as

$$|\overbrace{\pi(f,2,3)}^{slot1},\overbrace{i}^{slot2},\underbrace{\overbrace{j}^{slot3}}_{output}\rangle \to |\overbrace{\pi(f,2,3)}^{slot1},\overbrace{i}^{slot2},\underbrace{\overbrace{j\oplus f(i)}^{slot3}}_{new\ output}\rangle, \tag{2.59}$$

where we abbreviate x and $\vec{n}$. Here, the $\oplus$ algebra represents a binary operation. Recall that a slot is a group of cells in the infinite memory. The cells show a binary number, i.e. a sequence of 0 and 1. There, a combination of $j \oplus f(i)$ will be a binary operation following the algebra rule of

equation (2.58).

Second, we define the subprogram $\phi(g, a)$ as a part of $\pi(f)$. Its role is to transform a stored information, integer i, into $g(i)$ at slot a with a computable function g. A bijective computable function g has a program $\phi(g, a)$, and its role is to replace an integer i in slot a by $g(i)$. The subprogram $\phi(g, a)$ consists of four parts, and we explain them as follows.

1. The first part is $\phi_1(g, a, b)$. Suppose that slot a has x and slot b has zero initially. Then $\phi_1(g, a, b)$ changes the values of slots a and b as follows,

$$\text{slot } a : i \to i, \tag{2.60}$$

$$\text{slot } b : 0 \to g(i) \oplus 0 \to g(i). \tag{2.61}$$

   Slot b stores the value of $g(i)$ temporarily.

2. The second part is $\phi_2(g^{-1}, b, a)$. This portion resets the value of slot a to be zero as follows,

$$\text{slot } a : g^{-1}(g(i)) \oplus i \to i \oplus i \to 0, \tag{2.62}$$

$$\text{slot } b : g(i) \to g(i). \tag{2.63}$$

3. The third part is $\phi_3(I, b, a)$. This portion adds $g(i)$ into the value of 0 at slot a as follows,

$$\text{slot } a : g(i) \oplus 0 \to g(i), \tag{2.64}$$

$$\text{slot } b : g(i) \to g(i). \tag{2.65}$$

4. The fourth part if $\phi_4(I, a, b)$. This portion adds $g(i)$ into the value at slot b and resets the value to be zero at slot b as follows,

$$\text{slot } a : g(i) \to g(i), \tag{2.66}$$

$$\text{slot } b : g(i) \oplus g(i) \to 0. \tag{2.67}$$

$$V_0 = \begin{pmatrix} \cos\alpha & \sin\alpha \\ \text{-sin } \alpha & \cos\alpha \end{pmatrix} \quad V_1 = \begin{pmatrix} \cos\alpha & i\sin\alpha \\ i\sin\alpha & \cos\alpha \end{pmatrix} \tag{2.70}$$

$$V_2 = \begin{pmatrix} \text{Exp(i}\alpha) & 0 \\ 0 & 1 \end{pmatrix} \quad V_3 = \begin{pmatrix} -1 & 0 \\ 0 & \text{Exp(i}\alpha) \end{pmatrix} \tag{2.71}$$

Figure 1: Spin rotation matrices
The matrices formulate a change between zero and one at each cell. A combination of the four matrices can construct a unitary operator arbitrary close to any given unitary operator.

We combine the four portions and obtain the subprogram $\phi(g, a)$ as

$$\phi(g, a) = \phi_1(g, a, b)\phi_2(g^{-1}, b, a)\phi_3(I, b, a)\phi_4(I, a, b). \tag{2.68}$$

Again, the overall effect of ϕ is to take a value i at slot a and put $g(i)$ into slot a. Here, I is the *perfect measurement* function [2], and this function is defined as,

$$|\pi(I, 2, 3), i, j\rangle \rightarrow |\pi(I, 2, 3), i, j \oplus i\rangle. \tag{2.69}$$

For issue 2, we use the spin rotation matrices to formulate one bit of information at each cell. The bit information changes between zero and one at each cell of the infinite memory. We use the spin rotation matrices in quantum mechanics to formulate the process. The matrices formulate a change between zero and one at each cell. A combination of the four matrices can construct a unitary operator arbitrary close to any given unitary operator. The matrices are shown in Figure 1.

The formulation provides another advantage. Classical computers are based on one bit information, 0 and 1; however, the spin rotation operator formulation allows any weighted combination of 0 and 1. Thus, the approach not only describes a bit information in classical computer, but also has the advantage to formulate a superposition state in quantum computer.

Recall the subprogram $\phi(g, a)$ discussed at issue 1. We consider an executable program $\phi(V_i, a)$. It performs the rotation operator V_i for $i \in \{0, 1, 2, 3\}$ on the least significant digit in slot a. We illustrate the program now. Suppose a computer state, which has the least significant digit

j, which is zero or one, at slot 2. Then, we apply the rotation operator V_i and that computer state evolution is discribed as

$$\begin{aligned} |\phi(V_i,2),j\rangle &\rightarrow \sum_{k=0}^{1}\langle k|V_i|j\rangle|\phi(V_i,2),k\rangle \\ &= \langle k|V_i|j\rangle|\phi(V_i,2),k\rangle|_{k=0} + \langle k|V_i|j\rangle|\phi(V_i,2),k\rangle|_{k=1}. \end{aligned} \tag{2.72}$$

We add some more information for the equation (2.72). First, the initial number j will be k after the operation. Since j,k represent one bit, their possible numbers are zero or one again.

Second, the initial computer state $|\phi(V_i,2),j\rangle$ will evolve after we execute the program $\phi(V_i,2)$ and apply the rotation operator V_i. There are two possible outcomes $|\phi(V_i,2),k\rangle$ with $k=0$ and $k=1$. We would like to know a probability that the initial computer state $|\phi(V_i,2),j\rangle$ ends up with the final computer state $|\phi(V_i,2),k\rangle$ with $k=0$ and $k=1$. Therefore, we construct a probability amplitude $\langle k|V_i|j\rangle$ for $k=0,1$ respectively, and create the new computer state;

$$\sum_{k=0}^{1}\langle k|V_i|j\rangle|\phi(V_i,2),k\rangle. \tag{2.73}$$

Be aware that we obtain a probability for $k=0,1$ each if we square their associating probability amplitudes $\langle k|V_i|j\rangle$.

For issue 3, we need to ensure that the computer state $|\psi(t)\rangle$ returns to the initial state after the program execution. Suppose that the computer consists of a sequence of L cells which stores L-bit information and a program $\rho(|\psi\rangle)$. The program $\rho(|\psi\rangle)$ returns a computer state $|\psi(t)\rangle$ to the basis state $|0_L\rangle$ in which all L bits are zero.

For instance, a computer state $|\psi(t)\rangle$ has two possibilities depending on its first bit, 0 or 1. Thus, the state is expressed by a linear superposition of those two possibilities as

$$|\psi(t)\rangle = c_0|0\rangle|\psi_0\rangle + c_1|1\rangle|\psi_1\rangle, \tag{2.74}$$

where the first term represents that the state's first bit is zero while the second term represents that the state's first bit is one. The ket vectors $|\psi_0\rangle$ and $|\psi_1\rangle$ are states of the $L-1$ bits that are stored in the cells between m_2 and m_L. By the induction hypothesis, there exists programs ρ_0

and ρ_1 which returns $|\psi_0\rangle$ and $|\psi_1\rangle$ to the $(L-1)$-fold product $|0_{L-1}\rangle$. Therefore there exists a program which executes the following step. If the first bit is zero, then execute ρ_0 and return $|\psi_0\rangle$ to $|0_{L-1}\rangle$. If the first bit is one, then execute ρ_1 and return $|\psi_1\rangle$ to $|0_{L-1}\rangle$. This operation, acting on the cells between m_2 and m_L, returns the physical state (2.74) to

$$(c_0|0\rangle + c_1|1\rangle)|0_{L-1}\rangle. \tag{2.75}$$

2.4.6 In what sense is our quantum computer universal?

We discuss what universal computers mean here. Certain computers are designed and built for specific purposes such as optimization calculations or payment calculations at grocery stores. They are calculators rather than universal computers.

Universal computers serve general purposes. If a specific computable algorithm is given with arbitrary inputs, then a universal computer can present results. Universal computers execute computable functions. So, a computer is called universal if it generates an output based on a given algorithm (or a program) to execute a provided mathematical function.

Therefore, in addition to optimization or price calculation, universal computers can calculate an economic growth rate, an average score of examinees, or annual interest rate of a bank as long as an algorithm is given.

Quantum computers have the following capabilities. They have a finite size of CPU and memory. The memory is called qubit, which stores a superposition of CPU and memory states. Quantum computers execute quantum algorithms step by step, generate outputs, and halt. The execution process is formulated theoretically by an unitary operator.

Our quantum computer can simulate any quantum computer with any input. Given a specific algorithm, a quantum computer can execute it and generate outputs. Therefore, it will be a perfect physical simulator such as the entire universe with an appropriate input.

2.5 Construction step 3: Quantum computers from classical ones

A new approach for parallel computing, called a superposition, is a leap from classical to quantum computers. Classical computers employ parallel computing techniques already. Here are

some methods. One computer uses not just one CPU, but some CPUs together for one task. The idea is that many tasks can be sudivided into smaller ones. Therefore, the parallel computing technique subdivides one big task into pieces and assigns them into all CPUs. Concurrent computing assigns a main task to many CPUs and proceeds simultaneously. One advantage is that the concurrent technique can connect as many CPUs as needed and use all together for one task. Classical computers have techniques for parallel computing, and all the techniques have the same approach to use as many CPUs as possible so that a task can be completed soon. Therefore, the processing speed depends on the number of CPUs. Unlike that approach, quantum computers use a superposition of states. We illustrate that difference conceptually.

Classical computers proceed a task one step at a time. For instance, if a classical computer calculates a function $f(x)$ with the input $x = 0, 1, 2, \cdots, n$, then the computer calculates the function n times once with each input to obtain the output $\{f(0), f(1), f(2), \cdots f(n)\}$. Therefore, the processing speed depends on the number of CPUs.

Quantum computers use the idea of superposition. For instance, if a quantum computer calculates the same function $f(x)$, then it defines a new parameter α by a superposition. Suppose that we define the input $x = 0$ in two cells as $|0, 0\rangle$, the input $x = 1$ in two cells as $|0, 1\rangle$, and the input $x = 2$ in two cells as $|1, 0\rangle$. The superposition of all the three inputs will be as

$$\alpha = c_0|0, 0\rangle + c_1|0, 1\rangle + c_2|1, 0\rangle, \tag{2.76}$$

and a quantum computer computes the computable function f as

$$f(x = \alpha). \tag{2.77}$$

The superposition enables three calculations to be completed by one computation. Here, a linear superposition is the key concept and a big advantage of quantum computers. It is not a number, but a superposition of each state, $|0, 0\rangle, |0, 1\rangle$, and $|1, 0\rangle$, that a quantum computer calculates. The concept needs two developments, *qubit* and a new quantum algorithm. Now, we discuss how the classical computer's concept, one bit signal of 0 and 1 leaps to the concept *qubit* next, and an example of quantum algorithm in the next chapter. We need a new memory cell system for a

superposition parallel computing, and it is called *qubit*. We discussed the memory consisting of cells. A conceptual contrast between classical and quantum computings is as follows. Suppose that classical computers have three cells. Each cell can have 0 or 1. Classical computers receive the input x, and compute the computable function f. For instance, if $x = 0$, then 0 is represented as a bit signal of three cells as $(0, 0, 0)$. The program $\pi(f)$ computes $f : (0, 0, 0)$, then it moves to the next calculation $x = 1$ with $(0, 0, 1)$. The process is repeated eight times until the program finishes the computation of $(1, 1, 1)$.

On the other hand, quantum computers have *qubits*. The qubits can have a number, 0 or 1. It is represented as a superposition of two states as equation (2.74). Then, instead of calculating each case, $(0, 0, 0), (0, 0, 1), \cdots, (1, 1, 1)$ respectively as the classical computers do, the qubit computes all the eight cases together as a superposition of $(0, 0, 0), (0, 0, 1), \cdots, (1, 1, 1)$. Such a superposition can be represented as;

$$\alpha = c_1(0, 0, 0) + c_2(0, 0, 1) + c_3(0, 1, 0) + \cdots + c_8(1, 1, 1). \tag{2.78}$$

The quantum computers compute not each state of $(0, 0, 0), (0, 0, 1), \cdots, (1, 1, 1)$ separately, but α so that all the cases $2^3 = 8$ can be completed by one calculation. The coefficients $c_1, c_2, \cdots, c_8$ represent probability amplitudes, but only one can be observed. The qubit technology has only to use one cell once since the technology uses a superposition of the bit signals zero and one both simultaneously, while a classical computer uses one cell twice for zero and one respectively. So, the quantum computers are faster than the classical computers twice if both have only one cell. If the qubit technology uses q cells, then the quantum computers are, in principle, faster by 2^q times. Quantum mechanics is the most fundamental theory describing the nature. The nature evolves with a probability. Therefore, quantum computers can simulate and reproduce any finite physical system evolution. Quantum computers can simulate even the universe itself if they are given a proper computable algorithm and all the data as an initial condition. Such a computation process, however, needs a new algorithm differing from classical codes we are using currently.

CHAPTER 3

QUANTUM PROGRAMS

3.1 Shor's algorithm

We discussed that quantum computers have a revolutionary system of parallel computing, which generates an incomparable processing speed so far. We, however, need a quantum program to maximize the potential of quantum computer. Here, we study how a quantum program achieves such a fast processing speed with the example of Shor's algorithm.

The mathematician Peter Shor proposed a quantum computer program called *Shor's algorithm*, and its purpose is integer factorization, i.e. finding prime factors for a given integer n [8].

Shor's quantum algorithm has a root of the idea in number theory that finding the period of a function is related to finding the factors of a large composite interger [12].

If an integer n is given and we would like to factor it into primes, then we construct the mod function $f_n(a) = x^a \mod n$, where x is an integer chosen at random, and coprime with respect to n.

The next step is to substitute a with $0, 1, 2, \cdots, Q-1$, where $Q = 2^q$ and $n^2 \leq Q \leq 2n^2$, at the mod function and obtain their outputs. The parameter q is a size of qubits the quantum computer has. The outputs will show a certain pattern. The number of values contained in that pattern is called the period r, that is a function of x. Number theory predicts that the greatest common divisor $\gcd(x^{r/2} - 1, n)$ is likely to be a factor of n.

3.2 Pseudocode for Shor's algorithm

Now, we illustrate how the idea is used for Shor's algorithm with $n = 15$, since the algorithm was tested to factor the number $n = 15$ with quantum computers and the experiment was successful for the first time in the history [11].

First, choose a number x that is coprime with respect to n. So, $x = 8$ in this example.

Second, define the mod function $f_n(a) = x^a \mod n$ with $x = 8$ and $n = 15$, that is

$f_{15}(a) = 8^a \mod 15$.

Third, substitute a with $0, 1, 2, \cdots$ and obtain outputs of the mode function. There we have,

$$f_{15}(0) = 1, \quad f_{15}(1) = 8, \quad f_{15}(2) = 4, \quad f_{15}(3) = 2, \quad f_{15}(4) = 1, \quad f_{15}(5) = 8, \quad f_{15}(6) = 4, \cdots \tag{3.1}$$

Fourth, analyze the outputs. The outputs form the clear pattern of $1, 8, 4, 2, 1, 8, 4, 2, ...$, and the pattern consists of four numbers 1, 8, 4, and 2. Thus, the period of this sequence is r=4. The period $r = 4$ is what the algorithm is looking for.

Fifth, compute the greatest common divisor $\gcd(x^{r/2} - 1, n)$ of $x^{\frac{r}{2}} - 1, n$. Suppose that we have $x^r \equiv 1 \mod n$, where r is a period. Then, its mathematical equivalence is $x^r - 1 = 0 \mod n$. So, we can rewrite it as;

$$n | x^r - 1 \rightarrow n | (x^{\frac{r}{2}} - 1)(x^{\frac{r}{2}} + 1). \tag{3.2}$$

If the equation $n | (x^{\frac{r}{2}} - 1)(x^{\frac{r}{2}} + 1)$ holds, but $n \not| x^{\frac{r}{2}} - 1$ and $n \not| x^{\frac{r}{2}} + 1$, then verification shows that there are suitable values of x. Now, if $f = \gcd(x^{r/2} - 1, n)$ is less than n, is not 1, and is a divisor of n, then 4 is a factor of n.

The greatest common divisor of $\gcd(x^{r/2} - 1, n)$ will be $\gcd(63, 15) = 3$. The number 3 is a non-trivial divisor of the number 15. If you have one factor of n, then the other is calculated by a simple division as $15/3 = 5$. Now, we have $15 = 3 \times 5$.

3.3 Implementing Shor's algorithm on a quantum computer

The key is to find the period r of the mod function $f_n(a) = x^a \mod n$ for a factor of given number n. The quantum computer can find a period r more quickly than a classical computer. The following is an outline of the process.

First, the quantum computer reserves slot 2 and 3 at memory as discussed in section 2.4.4,

and sets the initial condition as

$$|x;\vec{n};\vec{m}\rangle = |\underbrace{0}_{x};\underbrace{\vec{0}}_{\vec{n}};\underbrace{\pi(f),\underbrace{0}_{\text{slot 2}},\underbrace{0}_{\text{slot 3}},\vec{0}}_{\text{memory } \vec{m}}\rangle. \tag{3.3}$$

Note that f is the mod function and $\pi(f)$ is its executable program.

Second, at slot 2, the quantum computer creates a superposition of the integers $a = 0, 1, 2, 3, \cdots, Q-1$, where $Q = 2^q$ and q is the size of qubit. The superposition will be a linear combination of all those inputs to the mod function. Then a quantum computer computes the mod function with the superposition of a as an input. Since the superposition of a includes all possible inputs of a, one calculation of the superposition of a finishes all possible cases, and writes out a superposition of all the possible outputs b into slot 3 as

$$|\pi(f),\underbrace{0}_{\text{slot 2}},\underbrace{0}_{\text{slot 3}},\vec{0}\rangle \rightarrow |\pi(f),\underbrace{a}_{\text{slot 2}},\underbrace{b}_{\text{slot 3}},\vec{0}\rangle. \tag{3.4}$$

We abbreviate the quantum computer state noting the memory state only.

Third, the quantum computer observe b, the state of slot 3, however, if you measure the state, then you lose the superposition of slot 3. Before our observation, slot 3 has a superposition of all possible outputs b from the superposition of all possible inputs of a. Such a superposition of b will disappear once it is observed. The slot 3 state reduces to one specific state only. It is called a wave function collapse in quantum mechanics. The quantum computer still obtains some value k from slot 3, which is an output of the mod function. The wave function collapse changes the quantum computer state as

$$|\pi(f),\underbrace{a}_{\text{slot 2}},\underbrace{b}_{\text{slot 3}},\vec{0}\rangle \rightarrow |\pi(f),\underbrace{a}_{\text{slot 2}},\underbrace{k}_{\text{slot 3}},\vec{0}\rangle. \tag{3.5}$$

Fourth, the quantum computer changes the superposition of a at slot 2 due to the wave function collapse in slot 3. Observing slot 3 will cause a change in the state of slot 2. Quantum mechanics says that an observation itself changes a physical state. Slot 2 and 3 are parts of the same quantum computer memory. Thus, observing slot 3 and extracting the information of k will

cause the wave function collapse at slot 2 as well. Therefore, the slot 2 state reduces from the superposition of all possible inputs of a to just a superposition of some values of a that satisfy the mod function with the observed k at slot 3. Precisely speaking, the superposition of a means that all possible values of $a, a+r, a+2r, \cdots$ satisfying the mod function with the specific value of k observed at slot 3. Recall that r is a period of the mod function. Now, the quantum computer state changes as

$$|\pi(f), \underbrace{a}_{\text{slot 2}}, \underbrace{k}_{\text{slot 3}}, \vec{0}\rangle \rightarrow |\pi(f), \underbrace{\alpha}_{\text{slot 2}}, \underbrace{k}_{\text{slot 3}}, \vec{0}\rangle. \tag{3.6}$$

Note that α is the superposition of the values of $a, a+r, a+2r, \cdots$.

Fifth, the quantum computer extracts the period r from slot 2 storing the superposition of the values of $a, a+r, a+2r, \cdots$ by the Quantum Fourier Transform and returns the result into the same slot. The Quantum Fourier Transform [7] transforms the state $|\alpha\rangle$ at slot 2 as

$$|\alpha\rangle \rightarrow \frac{1}{\sqrt{2^m}} \sum_{c=0}^{2^m-1} e^{\frac{2\pi i c x}{2^m}} |c\rangle. \tag{3.7}$$

After the execution of the Quantum Fourier Transform, slot 2 will have a periodic function with the peak of multiples of $1/r$ consequently. In other words, if we plot the periodic function, then the first peak is at $1/r$, and the second peak is at $2 \times 1/r$, and we see a peak at (integer)$\times 1/r$. Thus, if the slot 2 state is observed, it is likely to obtain a period r.

Sixth, the quantum computer calculates factors of given n with x, r. The rest of the task can be done by a classical computer as well. Since the period r is obtained already, quantum computers can find the factors of given n by $\gcd(x^{r/2}-1, n)$ and $\gcd(x^{r/2}+1, n)$.

Seventh, if the period r is not an even number, then we choose another value for x and repeat the process. As seen, finding the period r is the key to prime-factor a given n. Shor's algorithm uses a superposition for that purpose. Although some portion of the algorithm is executable by classical computers as well, only quantum computers can execute the superposition calculation thanks to the qubit system. The unique process of superposition achieves such a high processing speed of quantum computers.

CHAPTER 4

CONCLUSION: QUANTUM COMPUTABILITY

We have discussed a universal quantum computer. Here, we will summarize what the word *universal* means once again. Certain machines are designed and built for specific purposes. For instance, cash registers scan items and add prices at grocery stores. The machines are capable of just one task. Unlike such cash registers, universal computers can perform a variety of tasks. If specific programs are given, then the universal computers execute the programs and complete a variety of tasks such as word processing, music composition, or spreadsheet calculations.

Two important elements distinguish universal computers from machines for specific purposes. The two elements are algorithm and a goal of the task. The algorithm is a finite set of instructions to perform a computable function. A goal of the task is what to achieve or when to finish. For instance, repeating the same calculation one hundred times can be a goal of task.

For a task to be computable, computers must know two elements before they start the task. First, computers need to know a finite step by step instruction regarding how to do the task. Second, computers need to know when they are allowed to stop. The halting problem studies whether a task is computable with a given input and algorithm by an intrinsic reason.

Classical and quantum computers are alike in that both need an algorithm and a goal to complete a task, but different in terms of computability.

In our daily lives, our information transmission is encrypted by the RSA algorithm over the internet. An unauthorized access is possible theoretically, but impossible practically because the decoding process takes a long time. Therefore, the task is not effectively computable by classical computers, and we are safe at present.

It may be, however, possible to decode the encrypted information for quantum computers by using their high speed. Quantum computers will have the capability to make some of the present practically incomputable tasks computable. Therefore, the decoding process may be efficiently computable for quantum computers.

The RSA algorithm uses prime factorization of large numbers, and protects our

information by the difficulty of factoring, however, Shor's algorithm can reduce that difficulty with quantum computers.

Shor's algorithm was used to factor 15 into 3×5 by a quantum computer with 7 qubits in 2001 [11]. Quantum computers and the algorithm achieved the factorization of 21 in 2012 [6]. Various methods of quantum computing and algorithm are factorizing larger numbers one after another, 143 in April 2012 [13], and 56153 in November 2014 [1].

The steps of success of Shor's algorithm and quantum computers are possibly, at the same time, the steps toward the end of our safety. The success is ironic, because quantum computers and Shor's algorithm will facilitate the factorization and decoding process. Again, our information security is based on the difficulty of a large number prime-factorization, and that is the exact point quantum computers are trying to break. We are exposing ourselves to danger by our own efforts. When we lose information security, quantum computers and Shor's algorithm, as we studied in the thesis, may be responsible for it. Technology is always a double-edged sword.

REFERENCES

[1] N. S. Dattani and N. Bryans, *Quantum factorization of 56153 with only 4 qubits*, ArXiV quant-ph/1411.6758D, 2014.

[2] D. Deutsch, *Quantum theory as a universal physical theory*, Internat. J. Theoret. Phys. **24** (1985), no. 1, 1–41. MR 791830

[3] ———, *Quantum theory, the Church-Turing principle and the universal quantum computer*, Proc. Roy. Soc. London Ser. A **400** (1985), no. 1818, 97–117. MR 801665

[4] David Griffiths, *Introduction to quantum mechanics*, second ed., Pearson Prentice Hall, 2014.

[5] Laszlo Gyongyosi and Sandor Imre, *A survey on quantum computing technology*, Comput. Sci. Rev. **31** (2019), 51–71. MR 3902792

[6] Martin-Lopez et al., *Experimental realization of Shor's quantum factoring algorithm using qubit recycling*, Nature Photonics (2012), no. 6, 773–776.

[7] E. Rieffel, *An introduction to quantum computing for non-physicists*, ACM Computing Surveys (2000), no. 32, 300–335.

[8] Peter W. Shor, *Algorithms for quantum computation: discrete logarithms and factoring*, 35th Annual Symposium on Foundations of Computer Science (Santa Fe, NM, 1994), IEEE Comput. Soc. Press, Los Alamitos, CA, 1994, pp. 124–134. MR 1489242

[9] M. Tegmark and A. Wheeler, *100 years of quantum mechanics*, Scientific American **284** (2001), no. 2, 72–79.

[10] A. M. Turing, *On computable numbers, with an application to the Entscheidungsproblem*, Proc. London Math. Soc. (2) **42** (1936), no. 3, 230–265. MR 1577030

[11] Lieven M. K. Vandersypen, Mathias Steffen, Gregory Breyta, Constantino S. Yannoni, Mark H. Sherwood, and Isaac L. Chuang, *Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance*, Nature (2001), no. 414, 883–887.

[12] Colin P. Williams and Scott H. Clearwater, *Ultimate zero and one computing at the quantum frontier*, first ed., Copernicus, 2000.

[13] Nanyang Xu et al., *Quantum factorization 143 on a dipolar-coupling nuclear magnetic resonance system*, Physical Review Letters (2012), no. 108, 130501.

APPENDIX A

LETTER FROM INSTITUTIONAL RESEARCH BOARD

MARSHALL UNIVERSITY
www.marshall.edu

Office of Research Integrity

March 15, 2019

Junya Kasahara
Department of Mathematics
532 Smith Hall
Marshall University

Dear Junya:

This letter is in response to the submitted thesis abstract entitled *"Universal Quantum Computation."* After assessing the abstract, it has been deemed not to be human subject research and therefore exempt from oversight of the Marshall University Institutional Review Board (IRB). The Code of Federal Regulations (45CFR46) has set forth the criteria utilized in making this determination. Since the information in this study does not involve human subjects as defined in the above referenced instruction, it is not considered human subject research. If there are any changes to the abstract you provided then you would need to resubmit that information to the Office of Research Integrity for review and a determination.

I appreciate your willingness to submit the abstract for determination. Please feel free to contact the Office of Research Integrity if you have any questions regarding future protocols that may require IRB review.

Sincerely,

Bruce F. Day, ThD, CIP
Director