Quantum Computation: An Introduction

A Thesis presented

by

James Taylor Lewis Grimmelmann

To

Computer Science

in partial fulfillment of the honors requirements

for the degree of

Bachelor of Arts

Harvard College

Cambridge, Massachusetts

April 5, 1999

Contents

Chapter 1

Introduction

1.1 What is Quantum Computation?

In a literal sense, *all* computation is "quantum," in that it is carried out on physical devices which are subject to the laws of quantum mechanics. When we speak of "quantum computation," however, we generally mean something more specific: *the computation carried out by devices in which these laws are algorithmically explicit.* This attitude is similar to the way in which we speak of "randomized computation:" all computers are probabilistic to the extent that they are vulnerable to cosmic rays (for example), but by this phrase we usually mean computers whose use of randomness is explicit.

This analogy with randomized computation is useful in understanding the central problems of quantum computation as an emerging sub-field of computer science. Work on randomized computation generally falls into one of three categories:

1. "Where do we get it?" This category involves the description of architectural and algorithmic methods for obtaining random values for use in computation, and is the province of the disciplines of computer architecture and computer systems.

2. "How do we use it?" This category encompasses the description of algorithms which rely upon random choices for their efficiency or their effectiveness in solving computational problems, and is the domain of algorithm and protocol design.

3. "What can't it do?" This category includes the analysis of theoretical properties any randomized algorithms must satisfy and the setting of upper and lower bounds on the tractability of problems in models of computation that use randomness. Such analysis is is the realm of computability and complexity theory.

Nothing in this taxonomy, of course, depends on our restricting ourselves to randomized models of computation; we can equally well ask these questions about models in which other kinds of effects figure. Quantum computation is to quantum mechanics as randomized computation is to probability theory.

Quantum mechanics exhibits a number of phenomena which are alien to classical computing and to ordinary intuition. Quantum systems can exhibit *superposition*, in which the system behaves as though it were in multiple incompatible states simultaneously. The canonical example of superposition is the *two-slit* experiment, in which an electron is directed at a target through two small slits in an intervening barrier. Although each electron is observed to strike the detector at only one place, the distribution of locations where electrons strike indicate an *interference* pattern, as though a wave had traveled through both slits and interfered with itself (rather than being just the sum of the distributions from each slit independently). Such an outcome is impossible unless one posits that the electron travels through *both* slits. The electron is in a superposition of "traveled through the first slit" and "traveled through the second slit" states.

Superposition, as suggested in the above example, is also interesting because different terms in a superposition can interfere with each other: the information in the electron's "waveform" when it reaches the detector carries information about both possible paths. The act of *measuring* a quantum system can affect that system in radical ways: installing additional detectors to determine which slit the electron goes through destroys the interference pattern. Merely examining a system — no matter how non-intrusive the examination — can (and, under certain circumstances *must*) change that system drastically.

Quantum systems also display the effects of *entanglement*. Two particles can be placed in an "entangled" state such that they display phenomena which cannot be accounted for by any local theory. There is no way to model the particles as separate physical systems, even when they are physically separated: changes to one can affect the other, and the pattern of outcomes resulting from measurements on them displays non-local correlations. Although it runs counter to intuition (and to several "laws" of physics that have not yet been successfully reconciled with quantum mechanics), entanglement is real and has been repeatedly verified experimentally.

Quantum computation attempts to harness these various effects for computational purposes.

1.2 History

Physicists and philosophers have been interested in the strange realm of quantum mechanics for many years. Over the course of the twentieth century, physics has developed a reasonably complete description of quantum mechanical phenomena and the mathematical rules governing the behavior of quantum systems. There is little philosophical consensus on the correct relationship between these rules and the physical world (other than that the rules give predictions which can be observed to be correct), but philosophy has focused attention on the precise ways in which quantum physics differs from classical physics. The minimal examples developed by philosophers to highlight these differences are the lineal ancestors of the abstract systems considered in quantum computation.

Richard Feynman was among the first physicists to suggest that the computational aspects of quantum mechanics were worth exploring. He noted that simulating quantum systems is exponentially difficult classically because quantum systems can efficiently create superpositions with exponentially many terms. For this reason, he suggested, it would be worthwhile to develop quantum "computers" for the purpose of directly simulating quantum systems of interest to theoretical physicists. He also observed that precisely because they could efficiently create superpositions with

exponentially many terms, quantum computers would be capable of a form of "quantum parallelism." The difficulty in exploiting such parallelism consists in finding a way for the different "processors" to communicate with each other. Such communication is necessary if they are to combine information to make a single coherent answer, rather than an answer randomly chosen from an exponentially large sample.

The next major advance in quantum computation came at the end of the 1980s and in the early 1990s when David Deutsch developed a quantum algorithm that required fewer oracle queries to answer a question about an unknown function than did any classical algorithm. Following Deutsch's discovery, interest in quantum computation rose dramatically. A succession of algorithms for generalizations of Deutsch's problem were accompanied by an interest in methods for actually constructing quantum computers and in developing formal models to reason about them. In 1994, Peter Shor presented an algorithm for polynomial-time factorization on a quantum computer, demonstrating unexpected power for quantum algorithms. Shor's algorithm has been followed by several others, none quite as dramatic. Lov Grover's algorithm for unsorted database search, which achieves a quadratic speedup over any classical algorithm, is the most prominent, and the one with the most promising prospects for further discoveries.

1.3 From Hardware to Algorithms

In the tradition of abstraction within computer science, quantum algorithms have been specified with relatively little reference to specific physical models of computation. Traditional algorithm designers rely upon a certain (and somewhat fluid) set of primitive operations. The Shor and Grover algorithms, along with their extensions, depend on a "quantumized" set of such primitives, and their authors have been careful to express their algorithms so as to make those primitives which they require more or less explicit. In many respects, these primitives — by and large, simple operations on quantum bits, or "qubits" — seem to be reasonable algorithmic building blocks, much like the conventional primitives used to specify classical algorithms: the instruction-set of a von Neumann machine, the operations of basic set theory, or the manipulation of elementary data structures, for example.

In the quantum world, however, it is not *a priori* obvious that such abstract primitives necessarily map onto any implementable computational device. We understand reasonably well what a classical "bit" is: the voltage level along a wire can be effectively controlled, maintained in a way that is robust against likely errors, used to interact in computationally useful ways with other bits similarly stored, and scaled arbitrarily. Because of the small scale on which quantum effects dominate classical ones — and the quixotic nature of those very effects — virtually none of the traditional technology used to implement classical bits can be used to create qubits that manifest desirable quantum properties and satisfy the above constraints. The design of quantum computational devices requires techniques from almost every discipline of computer science. In many cases, these techniques draw upon both traditional results and more recent, seemingly unrelated, research. That quantum computation is even plausible is a major accomplishment, and one in which virtually all of computer science has participated.

1.4 About This Thesis

This thesis has been written as an introduction to quantum computation for computer scientists. It surveys, from the bottom up, the results which establish the possibility of quantum computation, the systems necessary to make quantum computation a reality, and the algorithms which make quantum computation desirable. The tools employed to surmount the obstacles faced by quantum computers come from many disciplines of computer science, and this thesis teases out the ways in which the traditional methods of these disciplines can be applied within the new context of quantum computation.

The specific contributions of this thesis are twofold. First, much of the material presented herein has not been previously gathered in one place. Although good surveys are available for some fields of quantum computation, there has been little cohesion to the literature in other areas. Specifically,

- The competing hardware implementations proposed for quantum computers are almost never compared with each other, and it is rare to find an article discussing more than one such proposal. This thesis presents all the salient hardware models in a single unified framework, emphasizing the relative feasibility of constructing actual quantum computers of each proposed type.
- Discussion of theoretical models of quantum computation is often not grounded in the relationship of those models to the quantum devices they are, in theory, abstractions of. The crucial distinctions between circuit and Turing machine models, in particular, seem to have been independently discovered on several occasions, but the importance and ramifications of these distinctions have rarely been noted. This thesis discusses formal models comparatively, with reference both to their underlying mathematical objects and to the ways in which these formal models abstract away from and reflect physical computational systems.
- The lack of cohesion in discussions of quantum programming languages is especially severe. Techniques borrowed from functional programming and reversible computing largely solve most of the problems posed by quantum programming, but this accomplishment has gone almost completely unnoticed, even by those who have participated in it. This thesis provides the first known overview of quantum programming as a single discipline. As a result, it is able to juxtapose the problems of quantum programming with their solutions, making explicit many connections that have remained implicit in the work of previous authors.

The other contributions of this thesis are expository ones. Beyond the advantages of discussing a wide variety of quantum computational issues in one unified context, this thesis has a number of specific features designed with pedagogical intent. In particular:

- It is the view of the author that much of the mathematical detail usually provided in discussing quantum algorithms is unnecessary, advancing the interests of neither precision nor elegance. The mathematical treatment of quantum effects in this thesis is considerably simplified from the usual treatment given in papers on the subject. Mathematical description is regarded as a means towards the end of providing the reader with a solid intuition for the computational aspects of quantum systems, not as an acceptable substitute for that intuition.

- As a corollary to the above, most discussions of quantum computation are designed for an audience of physicists, and therefore assume in their audience prior exposure to quantum mechanics and its mathematical formalisms. This thesis is relatively unique in that it is written for computer scientists. The necessary physics and linear algebra are motivated by consideration of computational systems, rather than being presented as abstract givens. Conversely, this thesis assumes in the reader a background in the ideas and terms of computer science, to about the level of an introductory or intermediate undergraduate curriculum.

- The extended treatment of programming language design for quantum devices is motivated by a belief that the study of methods for programming a system leads to a better understanding of that system's capabilities and limits. Understanding quantum computation and understanding how to specify quantum computation are closely related challenges.

- The quantum Fourier transform, despite its importance, is rarely presented clearly. Where proofs of correctness are offered, they are generally unilluminating. This thesis gives a new, inductive analysis of the quantum Fourier transform.

- This thesis provides an annotated bibliography to point out important papers and good review articles on quantum computation to the reader interested in exploring some aspect of quantum computation in greater depth.

1.5 Outline

Chapter 2 discusses the mathematical formalisms employed in later chapters, the basic constraints on quantum computation imposed by theoretical physics, and some useful ways of thinking about quantum systems. The subsequent chapters draw frequently, both implicitly and explicitly, on the definitions and results of chapter 2.

Chapter 3 surveys the physical implementations that have been proposed (and in some cases built) for quantum computers, with an emphasis on assessing the suitability of each as a platform for large-scale quantum computation. The key concern for each system is its ability to implement, scalably and reliably, a certain set of primitive operations.

Chapter 4 establishes that the primitive operations supported by proposed hardware for quantum computers are sufficient, in the sense that computers capable of carrying out such operations can carry out any operation available on any quantum computer. Part of this task involves pinning down the precise formal meaning of "any quantum computer," so chapter 4 investigates the principal theoretical models currently use for describing general-purpose quantum computers.

Chapter 5 studies the connections between quantum computation and *reversible computation*. Reversible computation emerged from research into information theory and low-power computation, but has established a number of results of importance to quantum computation. In particular, reversible computation provides answers to a number of problems faced in designing algorithms that will run on quantum computers and in reimplementing classically-computable functions on quantum architectures. Chapter 4 establishes the theoretical limits on the computations supported by devices described in chapter 3; chapter 5 fills in the gap by describing how those devices can compute functions more powerful than their primitive operations.

Chapter 6 discusses techniques for programming quantum computers, using the tools of programming language theory and design. A key challenge in programming computer systems is to find notations which communicate both the capabilities and the limitations of the devices being programmed; a number of disparate ideas from functional programming and other paradigms of programming are applicable to programming quantum computers.

Chapter 7 details some of the algorithms designed to take advantage of quantum computers. After starting with a sequence of toy problems which show off the power of quantum computers on certain kinds of problems, the chapter concludes with a discussion of Shor's fast factoring algorithm and Grover's unsorted database search algorithm, both of which achieve dramatic speedups over the best known classical algorithms for problems of general interest. The results of chapters 2 through 6 provide the "how" of quantum computation. Chapter 7 provides the "why."

Finally, chapter 8 briefly surveys other fields of interest within quantum computation before concluding with a discussion of the future prospects of quantum computation.

Chapter 2

Mathematical Preliminaries

2.1 Introduction

We wish to describe and to create computations that produce answers to questions of interest. The abstract existence of some value is not good enough: we require a procedure which terminates by providing that value in intelligible form. For example, any formula of first-order logic has a definite truth-value, but this *mathematical* guarantee is not a *computational* guarantee, and there exist families of formulae whose truth-values are not calculable by any computational process.

First-order logic is an example of a mathematical system that has certain properties which are not directly observable. Quantum mechanics postulates that only certain properties of physical systems are directly observable. These properties can take on values from some specified set; when we observe such a property, it has exactly one of those values. Further, if we observe some given value for an observable property of a system, then — in the absence of some explicit intervention — that property will continue to have that value on future observations.

These three facts —

1. that only certain properties of a system are observable,
2. that observations have definite outcomes,
3. and that identical repeated observations will have the same outcome —

are all we require to give an overview of quantum mechanics as it applies to computation.

2.2 The Classical Realm

2.2.1 A Bit

"Classical" systems, for our purposes, are those in which every physical property is observable. As an example of a classical system, consider a bit. A bit has a single observable property and that property can have one of two possible values. Let us call these values $|0\rangle$ and $|1\rangle$.

To be meaningfully observable, $|0\rangle$ and $|1\rangle$ must have some particular physical implementations. If the bit is to be stored in a transistor switch, $|0\rangle$ could designate a transistor with output voltage at or near five volts, and $|1\rangle$ a transistor whose output voltage is set near zero volts. If the bit is stored on a punch card, $|0\rangle$ could designate the absence of a hole in a specified position, and $|1\rangle$ the presence of a hole in that position. Other implementations would assign different physical states to $|0\rangle$ and $|1\rangle$.

On the other hand, the symbols $|0\rangle$ and $|1\rangle$ are typically used to represent abstract mathematical values according to some formal mapping. In a transistor switch with the active-low convention, $|0\rangle$ denotes logical FALSE and $|1\rangle$ denotes logical TRUE. More typically, $|0\rangle$ and $|1\rangle$ denote the *numbers* 0 and 1, respectively. In a register storing a binary number, a register with kth bit $|1\rangle$ denotes a number which is 2^k greater than the number denoted by the same register with kth bit $|0\rangle$, but the precise denotation of this bit cannot be established independently of denotations of the other bits.

$|0\rangle$ and $|1\rangle$ provide a convenient middle layer between physical implementations and the abstract systems these implementations model. These names allow us to speak about the computations carried out by computational hardware without becoming bogged down in the details of any specific implementation. They also allow us to use the same hardware to carry out computations on different kinds of mathematical objects. In one context $|1\rangle|0\rangle$ may stand for the ordered pair $(1, 0)$; in another it may stand for the binary number 10. It is easier to specify the mathematical properties of these objects in terms of *names* for particular physical states than in terms of the physical states themselves.

2.2.2 States and Observables

A usual view of computational systems — and of systems in general — is that we specify some space of possible states of the system, and the system's actual state is then represented by a point within that space. For each observable property of a system (usually called *observables*), we also specify what value that property will have if the system is observed (such an observation is also called a *measurement*) while in any given state.

Since a classical bit stores the distinct values $|0\rangle$ and $|1\rangle$, and these two states suffice to explain the behavior of the system, we define its state space B_c to be

$$B_c = \{|0\rangle, |1\rangle\}. \tag{2.1}$$

The notation B_c is meant to suggest an abstract classical bit.

To be fully precise, we now specify, for each possible state in the state space, the value the bit's observable will have if that observable is measured. If the system is in state $|0\rangle$, the measurement will produce the value $|0\rangle$; if the system is in state $|1\rangle$, the measurement will produce $|1\rangle$. This claim seems trivial because the set of observable values is the same as the set of possible states. We will soon see systems for which this equivalence does not hold, and we will need to give more complicated rules for determining the values that result from measurements.

2.2.3 Transformations

The state of a system is not static. There are usually many possible actions that will change the results of future observations made on the system. If a bit is observed to have the value $|0\rangle$ and is then fed through an inverter, the bit will have the value $|1\rangle$ if it is observed again. Abstractly, we specify such *operations* or *transformations* by giving a function from the state space into itself. For example, feeding a stored bit through an inverter is equivalent to applying the mapping F given by[1]

$$F|0\rangle = |1\rangle \quad F|1\rangle = |0\rangle. \tag{2.2}$$

Initializing the bit to represent the constant value 1 is equivalent to applying the mapping G given by

$$G|0\rangle = |1\rangle \quad G|1\rangle = |1\rangle. \tag{2.3}$$

The values of measurements taken before and after a transformation will, in general, be different. Some systems will (under certain physical conditions) change state with the passage of time. These changes can also be specified as such a transformation.

To review, let us think about this single-bit system in terms of our three principles. A generic bit has one physical property, and one observable. A measurement of the system results in either $|0\rangle$ or $|1\rangle$. Measurements do not produce some other value, say $|fish\rangle$, nor do they produce both $|0\rangle$ and $|1\rangle$ simultaneously. Further, in the absence of some external altering force, once a measurement has produced $|0\rangle$, the value of a future measurement will also be $|0\rangle$; similarly, a $|1\rangle$ remains a $|1\rangle$.

2.3 The Probabilistic Realm

2.3.1 States and Observations

Consider now a *probabilistic* bit. This system consists of a single, possibly unfair, coin. Again, this system has a single observable: the outcome of the coin flip. This observable can take on the values $|0\rangle$ and $|1\rangle$. However, B_c is not a large enough state space to explain the behavior of a random bit, since it has no way to represent "a bit which will have value $|0\rangle$ with probability p" for any p other than zero or one. Thus, we define the state space B_p of a probabilistic bit by attaching coefficients to $|0\rangle$ and $|1\rangle$ that indicate the probability that a measurement of the bit will produce the value $|0\rangle$ or $|1\rangle$, respectively. Formally,

$$B_p = \{a_0|0\rangle + a_1|1\rangle : a_0, a_1 \in \mathbb{R}; a_0^2 + a_1^2 = 1\}. \tag{2.4}$$

Measuring a bit in state $a_0|0\rangle + a_1|1\rangle$ produces the value $|0\rangle$ with probability a_0^2 and the value $|1\rangle$ with probability a_1^2. We have specified the state of the system in terms of square roots of

[1] In quantum mechanics, such transformations are usually specified using the applicative notation of linear algebra, because such transformations, in general, are linear transformations.

probabilities (instead of probabilities themselves) for reasons that will be clearer later. The side condition that $a_0^2 + a_1^2 = 1$ ensures that the probabilities sum to 1.[2]

Suppose a measurement produces the value $|0\rangle$. Because repeated measurements must be consistent, future measurements must also produce the value $|0\rangle$ with certainty. The only state in B_p which, when measured, yields $|0\rangle$ with certainty is $1|0\rangle + 0|1\rangle$. Therefore, we must conclude, that the state of the system after the first measurement is $1|0\rangle + 0|1\rangle$. The act of measuring a system *changes* the state of the system.

This is not as strange as it may sound. Examining the result of a random coin flip yields information about that coin flip. The coin cannot possibly have a different value unless it is flipped again. The bit *had* some probability of having the value $|1\rangle$, but now it *has* the value $|0\rangle$. A random bit is only random once.

Our three principles are also still in force. Observations still have definite outcomes, since seeing $|0\rangle$ and $|1\rangle$ are mutually exclusive events with probabilities summing to 1. Repeated observations produce the same value. And not every property of the system is directly observable: there is no way to directly observe a probability[3]. Gathering information about a system is an active, intrusive, process: once we know that the bit is $|1\rangle$, we have "destroyed" the possibility that it will be $|0\rangle$ when we look at it next.

2.3.2 Transformations

What does it mean to apply a transformation to this system? Consider the action of a bit-inverter on the state $a_0|0\rangle + a_1|1\rangle$, followed by an observation of the bit's state. The following explanation is tempting:

"With probability a_0^2, the bit initially had value $|0\rangle$, was inverted to $|1\rangle$, and observed as such. With probability a_0^2, the value $|1\rangle$ is observed at the end. Note that the probabilities for each outcome are the same as they would have been if the state $a_1|0\rangle + a_0|1\rangle$ was observed directly."

The problem with this reasoning is that the statement "with probability a_0^2, the bit was initially a $|0\rangle$," requires a measurement. Thus, the above procedure is really a measurement followed by an operation on the measured bit, whereas we were trying to capture the idea of an operation on an *unknown* bit. It is not possible to "look into" the unknown bit in order to apply a transformation to it. The result of of applying some operator to this state is another state of the system, one which is equally opaque. The only way to extract information about the value of the system is to measure it, which forces it to "collapse" into one value or the other. Until a measurement has taken place, we have no legitimate basis for speaking about the bit as "having value" $|0\rangle$ or $|1\rangle$. It is merely in a state from which we will eventually be able to extract a value; that value will be $|0\rangle$ with a certain probability and $|1\rangle$ with some other probability.

[2]It would be possible to remove this side condition, but then we would need to introduce a normalization term to the rules for measurement to make the probabilities of the different outcomes sum to 1. This normalization term makes certain states, like $5|0\rangle$ and $|0\rangle$ completely indistinguishable to any measurement, even under any possible transformation to the system. Doing the normalization at the level of the state space, rather than at the level of measurements, removes this redundancy.

[3]The best we can do is to *infer* those probabilities, with some chance of error and some degree of approximation, based on the outcomes of measurements. For coin flips, these measurements are Bernoulli trials.

This said, the "incorrect" heuristic for carrying out the bit-flip transformation gives us the right answer: the bit-flip carried out really does swap the probabilities associated with each value. If T acts some way on $|0\rangle$ and some other way on $|1\rangle$, then we have all the information we need to figure out how it acts on an arbitrary element of the state space:

$$T\left(a_0|0\rangle + a_1|1\rangle\right) = a_0 T|0\rangle + a_1 T|1\rangle. \tag{2.5}$$

We can reason carefully about the effects of changes to the system (in terms of the values that observations will take), even under uncertainty about the status of the system.

2.3.3 The Linear Algebra of a Probabilistic Bit

Our definitions from above can be restated in more general and concise forms in the notation of linear algebra.

B_p is a subset of the vector space $\mathbb{R}^2$, and $\{|0\rangle, |1\rangle\}$ is a basis for this space. Let us impose on $\mathbb{R}^2$ the inner product

$$\left(a_0|0\rangle + a_1|1\rangle\right) \cdot \left(b_0|0\rangle + b_1|1\rangle\right) = a_0 b_0 + a_1 b_1, \tag{2.6}$$

which induces the norm

$$\|a_0|0\rangle + a_1|1\rangle\| = \sqrt{(a_0^2 + a_1^2)} \tag{2.7}$$

B_p is the subset of $\mathbb{R}^2$ consisting of all vectors with norm 1. We denote by P_i the projection onto the subspace spanned by $|i\rangle$, i.e.

$$\begin{aligned} P_0(a_0|0\rangle + a_1|1\rangle) &= a_0|0\rangle \\ P_1(a_0|0\rangle + a_1|1\rangle) &= a_1|1\rangle. \end{aligned} \tag{2.8}$$

Finally, we use the notation

$$\text{prob}[|\phi\rangle \hookrightarrow |i\rangle] \tag{2.9}$$

to stand for the probability of obtaining the value $|i\rangle$ from a measurement of state $|\phi\rangle$. Given these definitions, the probability of obtaining a given value upon a measurement be rewritten in the simpler form

$$\text{prob}[|\phi\rangle \hookrightarrow |i\rangle] = \|P_i|\phi\rangle\|^2. \tag{2.10}$$

To ensure consistency of future measurements, we would like to say that after measuring value $|i\rangle$ from state $|\phi\rangle$, the new state of the system is $P_i|\phi\rangle$. However, this state does not have norm 1, so it needs to be properly normalized to

$$\frac{P_i|\phi\rangle}{\|P_i|\phi\rangle\|}. \tag{2.11}$$

By equation (2.5), we can characterize the possible transformations on B_p as those linear transformations which preserve norm.[4]

Note that there is in general no way to distinguish between the states $|\phi\rangle$ and $-|\phi\rangle$. Measuring either of these states directly will give exactly the same probability distribution. Applying any transformation T to these states yields the states $T|\phi\rangle$ and $-T|\phi\rangle$, which still produce identical distributions of values when observed. We allow B_p to contain these two distinct names, with the understanding that they represent the same underlying physical state.

2.4 The Quantum Realm

2.4.1 States, Measurements, and Transformations

We now have all of the formal framework necessary to give an account of a quantum system. Let us now consider the state space of the quantum analogue to a bit, the *qubit*. Instead being the norm-1 subset of the real vector space $\mathbb{R}^2$, the state space is now the norm-1 subset of the complex vector space $\mathbb{C}^2$. That is,

$$B_q = \{a_0|0\rangle + a_1|1\rangle : a_0, a_1 \in \mathbb{C}, a_0 a_0^* + a_1 a_1^* = 1\}. \tag{2.12}$$

The inner product here is the familiar

$$(a_0|0\rangle + a_1|1\rangle) \cdot (b_0|0\rangle + b_1|1\rangle) = a_0 b_0^* + a_1 b_1^*, \tag{2.13}$$

which induces the norm

$$\|a_0|0\rangle + a_1|1\rangle\| \quad = \quad \sqrt{|a_0|^2 + |a_1|^2} \tag{2.14}$$

Any state other than a basis element is said to be the *superposition* of those basis elements of which it is a linear combination.

As in the probabilistic case, we specify a measurement in terms of projections onto the subspaces spanned by $|0\rangle$ and $|1\rangle$. Equations (2.10) and (2.11) carry over unchanged. For B_q, we allow one observable, which can have the value $|0\rangle$ or $|1\rangle$. If the state $a_0|0\rangle + a_1|1\rangle$ is measured, the value observed will be $|0\rangle$ with probability $a_0 a_0^* = |a_0|^2$ and $|1\rangle$ with probability $a_1 a_1^* = |a_1|^2$.

Physicists would maintain that B_p has other observables (in fact, physicists would claim that B_p has an uncountably infinite number of observables). This is a case in which our practical restrictions on what constitutes an "observable" property of a system are stronger than those on the a physicist's formal definition of "observable." $|0\rangle$ and $|1\rangle$ are the *only* states of B_p to which

[4] The properties of this measurement are entirely determined by the projection operators P_0 and P_1. More generally, if V is a vector space which can be written as the direct sum of subspaces $U_0, U_1, \ldots U_n$, and P_i is the projection from V onto U_i, then these projections determine a possible measurement, which satisfies all the above equations. For our purposes, this will be important principally in discussing measurements on pieces of larger systems: looking at one piece of a system need not necessarily cause other pieces of the system to "collapse," as well. Technically, one specifies a self-adjoint operator; these subspaces are then the distinct eigenspaces of that operator. But this interpretation is more general than we require. With a single canonical basis in which all our measurements are carried out, it is more convenient to specify only the subspaces and not worry about a self-adjoint operator, which will be independent of basis.

we can give a meaningful extensional description. Thus we insist that only the measurement which produces $|0\rangle$ or $|1\rangle$ can be considered. This restriction does not affect the computational power of our formalism (to carry out a measurement in another basis, it suffices to rotate into that basis and then carry out a measurement in the basis $\{|0\rangle, |1\rangle\}$). We call $\{|0\rangle, |1\rangle\}$ the *computational basis.*

The permitted transformations on this system are those which are linear and preserve inner product[5]. Such transformations are called *unitary* and satisfy the equation

$$|\phi\rangle \cdot |\psi\rangle = U|\phi\rangle \cdot U|\psi\rangle \tag{2.15}$$

for all $|\phi\rangle, |\psi\rangle \in B_q$. Unitary transformations preserve norm and are invertible. Further, a transformation is unitary if and only if it is a bijection between two orthonormal bases. The identity transformation I is unitary: it takes the computational basis to itself. The negation transformation N, which takes $|0\rangle$ to $|1\rangle$ and vice-versa, is also unitary, as it permutes the computational basis.

Any linear transformation can be given in matrix form (with respect to a given basis); we will frequently find it convenient to do so. For example, with respect to the computational basis, I and N have matrix representations

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad N = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \tag{2.16}$$

There are also more exotic unitary transformations which have no classical counterparts. For example, consider the *conditional phase shift*, which leaves $|0\rangle$ unchanged but shifts the phase of $|1\rangle$ by π:

$$S_\pi = \begin{pmatrix} 1 & 0 \\ 0 & e^{\pi i} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{2.17}$$

For any state $|\phi\rangle$, measuring (in the computational basis) $|\phi\rangle$ and $S_\pi|\phi\rangle$ will produce completely identical distributions of values. Nonetheless, $|\phi\rangle$ and $S_\pi|\phi\rangle$ are different physical states, and applying other transformations after S_π *can* produce observably different states. We shall see an example of such a situation in the next section.

Another important unitary transformation is the *Hadamard rotation* H given by

$$H = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{2.18}$$

The Hadamard rotation has been called the quantum "fair-coin flip," since it takes $|0\rangle$ to the state $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, which has equal probability of yielding $|0\rangle$ and $|1\rangle$ if it is then measured. However, H has a number of other interesting properties, as we shall see.

2.4.2 A Quantum Example

We now examine an example that will demonstrate that the quantum picture of the world is not reducible to a classical one. Consider the two states

$$|X\rangle = \tfrac{1}{\sqrt{2}}|0\rangle + \tfrac{1}{\sqrt{2}}|1\rangle \quad |Y\rangle = \tfrac{1}{\sqrt{2}}|0\rangle - \tfrac{1}{\sqrt{2}}|1\rangle. \tag{2.19}$$

[5]This constraint follows from the dynamics of quantum systems, governed by the Schroedinger equation. time-evolution of a quantum system.

Measuring either of these states yields the values $|0\rangle$ and $|1\rangle$ with equal probability. In terms of this measurement, $|X\rangle$ and $|Y\rangle$ are indistinguishable.

On the other hand, applying the Hadamard rotation H to these states yields

$$\begin{aligned} H|X\rangle &= H\left(\tfrac{1}{\sqrt{2}}|0\rangle + \tfrac{1}{\sqrt{2}}|1\rangle\right) \\ &= \tfrac{1}{\sqrt{2}}\left(\tfrac{1}{\sqrt{2}}|0\rangle + \tfrac{1}{\sqrt{2}}|1\rangle\right) + \tfrac{1}{\sqrt{2}}\left(\tfrac{1}{\sqrt{2}}|0\rangle - \tfrac{1}{\sqrt{2}}|1\rangle\right) \\ &= |0\rangle \\ \text{and} & \\ H|Y\rangle &= H\left(\tfrac{1}{\sqrt{2}}|0\rangle + \tfrac{1}{\sqrt{2}}|1\rangle\right) \\ &= \tfrac{1}{\sqrt{2}}\left(\tfrac{1}{\sqrt{2}}|0\rangle + \tfrac{1}{\sqrt{2}}|1\rangle\right) - \tfrac{1}{\sqrt{2}}\left(\tfrac{1}{\sqrt{2}}|0\rangle - \tfrac{1}{\sqrt{2}}|1\rangle\right) \\ &= |1\rangle \end{aligned} \tag{2.20}$$

Measuring $H|X\rangle$ produces the value $|0\rangle$ with certainty; measuring $H|Y\rangle$ produces the value $|1\rangle$ with certainty. These two states, indistinguishable to direct measurement, nevertheless can be told apart by first applying an operation and then measuring. This is typical for quantum systems: often it is only the *relative* phase of two components of the state which carries information.

Although information is neither created nor destroyed by unitary transformations in a quantum system (because quantum systems are reversible) the information contained within a system may be more or less accessible depending on how that information is stored and how the system is measured. Measuring the $|X\rangle$ or $|Y\rangle$ states directly provides no information about the state of the system. Measuring a large number of copies of this state (either all $|Y\rangle$ or all $|X\rangle$) does not even produce any statistical information as to whether they are $|X\rangle$ or $|Y\rangle$.

Another way of characterizing this equivalence is that the "collapse" of the system induced by our measurement "destroys" one bit of information. On the other hand, applying H and then measuring extracts the full one bit of information present in the system. Since measuring $|0\rangle$ yields the value $|0\rangle$, the measurement causes no further "collapse" and no information is destroyed.

2.5 The Hadamard Rotation

The transformation H introduced in the previous section has a number of very interesting properties. First, it is self-inverse:

$$\begin{aligned} H^2 &= \left(\frac{1}{\sqrt{2}}\right)^2 \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\ &= I, \end{aligned} \tag{2.21}$$

a fact which "explains" the previous example. In terms of this identity, we can rewrite (2.20) as

$$\begin{aligned} H|X\rangle = H^2|0\rangle &= I|0\rangle = |0\rangle \\ H|Y\rangle = H^2|1\rangle &= I|1\rangle = |1\rangle. \end{aligned} \tag{2.22}$$

We already know that any unitary transformation takes a basis to another basis; the Hadamard rotation has the useful property of mapping back and forth between two especially useful bases. H takes the computational basis

$$|0\rangle \quad |1\rangle \tag{2.23}$$

to the *Fourier basis*[6]

$$\left\{ \ \tfrac{1}{\sqrt{2}}|0\rangle + \tfrac{1}{\sqrt{2}}|1\rangle \quad , \quad \tfrac{1}{\sqrt{2}}|0\rangle - \tfrac{1}{\sqrt{2}}|1\rangle. \ \right\} \tag{2.24}$$

and vice-versa. It is customary to refer to this mapping between bases as a *rotation* into and out of the Fourier basis.

The only difference between the two elements of the Fourier basis is the relative phase of the $|0\rangle$ and $|1\rangle$ components. As we have seen, relative phase differences are not directly measurable, but this does not mean that they are irrelevant (as global phase shifts are). In many quantum algorithms, almost all the information in a given state will be encoded in relative phase differences; Hadamard rotations or other changes of basis will then convert the phase differences into a directly observable form.

For example, consider the controlled-phase shift operator S_π on the Fourier basis states:

$$\begin{aligned} S_\pi\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right) &= \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle \\ S_\pi\left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle\right) &= \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \end{aligned} \tag{2.25}$$

S_π thus interchanges the two basis elements in the Fourier basis, even though it is diagonal in the computational basis. Thus, another way to implement N (the negation operator, which swaps $|0\rangle$ and $|1\rangle$) is to rotate into the Fourier basis, use S_π to interchange the elements of the Fourier basis, and then rotate back into the computational basis. Since H converts between computational and Fourier basis, this observation yields the identity

$$HS_\pi H = N. \tag{2.26}$$

In practice, this identity is more useful in the alternate form $S_\pi = HNH$, obtained by exploiting the self-inverse property of H. Phase shifts are computationally useful because they can contribute to non-trivial interference effects. A bit flip can be converted to a phase shift by applying it to a qubit in the state $\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$, because

$$N\left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle\right) = -\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle. \tag{2.27}$$

A bit-flip has been converted into an overall multiplication of the phase by -1 [7]. When combined with superposition, such phase shifts are central to quantum algorithms. To see how superposition and phase shifts interact in these algorithms, we will need to examine quantum systems larger than a single qubit.

[6] The Fourier basis is so named because it is the natural basis in which to carry out discrete Fourier analysis using quantum algorithms. We shall meet such algorithms later, in chapter 7.

[7] Technically a "phase shift of π," because $e^{i\pi} = -1$, a perspective that arises by thinking of quantum states in terms of polar coordinates.

2.6 Larger Quantum Systems

Consider the following characterization of B_q: "B_q is the set of norm-1 vectors in the complex vector space with basis B_c." Generalizing this construction yields a characterization of more complicated quantum systems in terms of their classical counterparts.

Specifically, consider an n-bit register, constructed from n single-bit registers, i.e. copies of B_c. Classically, this register can have any of 2^n possible values, one corresponding to each choice of the value $|0\rangle$ or $|1\rangle$ for each of the individual registers. Thus, the state space $B_c^{(n)}$ of a classical n-bit register is the set

$$B_c^{(n)} = \left\{ \begin{array}{c} |0\rangle \ldots |0\rangle|0\rangle \\ |0\rangle \ldots |0\rangle|1\rangle \\ |0\rangle \ldots |1\rangle|0\rangle \\ |0\rangle \ldots |1\rangle|1\rangle \\ \vdots \\ |1\rangle \ldots |1\rangle|0\rangle \\ |1\rangle \ldots |1\rangle|1\rangle \end{array} \right\} = \{|x_0\rangle|x_1\rangle \ldots |x_{n-1}\rangle : x_i \in \{0,1\}\} \tag{2.28}$$

For the sake of brevity, and to indicate that these states are states of a larger system, we will sometimes use the notation

$$|x\rangle = |x_0 \ldots x_{n-2}x_{n-1}\rangle \quad \text{for} \quad |x_0\rangle \ldots |x_{n-2}\rangle|x_{n-1}\rangle. \tag{2.29}$$

The state space $B_q^{(n)}$ of an n-bit quantum register, then, is the set of norm-1 vectors in the complex vector space with basis $B_c^{(n)}$.[8]. Formally,

$$B_q^{(n)} = \left\{ \sum_{x\in\{0,1\}^n} a_x|x\rangle : a_x \in \mathbb{C}, \sum_{x\in\{0,1\}^n} |a_x|^2 = 1 \right\}. \tag{2.30}$$

Since each qubit in this system has an observable value, the observables for the register as a whole are *the value of any specified qubit* In measuring qubit i, the projection P_0 is the projection onto the subspace of $B_q^{(n)}$ spanned by those elements of $B_c^{(n)}$ in which qubit i has value $|0\rangle$; P_1 is defined similarly. At this point, the usual equations (2.10) and (2.11) apply to determine the probability of each outcome and the state of the system after a measurement. More complicated measurements can be built up by measuring several qubits in succession. It is easy to check that measuring every qubit gives outcome $|x\rangle$ with probability $|a_x|^2$.

$B_q^{(n)}$ is a 2^{n+1}-dimensional vector space, so the possible transformations on it are the 2^{n+1}-dimensional unitary transformations. Of particular interest, however, is the subclass of those transformations which can be induced by acting on one bit at a time.

Consider the act of flipping a single bit (for notational convenience, the leftmost) in an n-bit classical register. This transformation is an action that involves one bit, so it can be considered as a

[8]More formally, $B_q^{(n)}$ is the *tensor product* of n copies of B_q.

mapping $N : B_c \to B_c$. However, N also induces a map $N_0 : B_c^{(n)} \to B_c^{(n)}$, obtained by computing the effect on the register of applying N to the leftmost bit:

$$N_0 = \begin{cases} |0\ldots00\rangle & \to & |1\ldots00\rangle \\ |0\ldots01\rangle & \to & |1\ldots01\rangle \\ & \vdots & \\ |1\ldots10\rangle & \to & |0\ldots10\rangle \\ |1\ldots11\rangle & \to & |0\ldots11\rangle. \end{cases} \tag{2.31}$$

Because N_0 is defined for every element of a basis for $B_q^{(n)}$, we can linearly extend N_0 into a map $N_0 : B_q^{(n)} \to B_q^{(n)}$:

$$N_0\left(\sum_x a_x|x\rangle\right) = \sum_x a_x N_0|x\rangle \tag{2.32}$$

More generally, it is possible to extend a transformation $T : B_q \to B_q$ which acts on a single qubit to a transformation $T^{(n)} : B_q^{(n)} \to B_q^{(n)}$ which acts on an entire quantum register. The action of $T^{(n)}$ on an element of $B_c^{(n)}$ can be determined by applying T to the appropriate qubit of that element and leaving the other qubits unchanged. The action of $T^{(n)}$ on an arbitrary element of $B_q^{(n)}$ follows by linearly extending its action on $B_c^{(n)}$.

As an example, consider a two-qubit system $B_q^{(2)}$, with basis

$$\{|00\rangle,|01\rangle,|10\rangle,|11\rangle\}. \tag{2.33}$$

A Hadamard rotation applied to the first qubit induces the map

$$\begin{array}{lcl} |00\rangle & \to & \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle \\ |01\rangle & \to & \frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|11\rangle \\ |10\rangle & \to & \frac{1}{\sqrt{2}}|00\rangle - \frac{1}{\sqrt{2}}|10\rangle \\ |11\rangle & \to & \frac{1}{\sqrt{2}}|01\rangle - \frac{1}{\sqrt{2}}|11\rangle. \end{array} \tag{2.34}$$

Single-qubit transformations are not the only ones which can be extended to work on entire registers in this way; it is also possible to specify a transformation which acts on two more more qubits and apply it to an entire register. In this case, just as before, the action of the transformation on a basis element is calculated by examining the action of the transformation on those qubits it explicitly affects.

An example of a transformation which cannot be reduced to a single-bit transformation is C, the controlled-NOT transformation, which flips the state of one qubit if and only if another qubit (the "control" qubit) is in state $|1\rangle$. That is, C acts on $B_c^{(2)}$ as

$$\begin{array}{lcl} |00\rangle & \to & |00\rangle \\ |01\rangle & \to & |01\rangle \\ |10\rangle & \to & |01\rangle \\ |11\rangle & \to & |10\rangle, \end{array} \tag{2.35}$$

where the left qubit is the control qubit. Notice that C permutes the elements of $B_c^{(2)}$, and thus is unitary. More importantly, C would not make sense as a transformation on either of its input qubits alone.

With multiple-qubit transformations such as C, it is possible to create *entangled* states, in which the correlation between two pieces of a system is important, not just their individual states. Consider the following sequence of transformations. Starting from the state $|00\rangle$, applying H produces the state $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle$. Applying the negation gate N to the second qubit, of this state, by (2.31), would then yield the state

$$\frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|11\rangle. \tag{2.36}$$

This state is not entangled: it can be written as

$$\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right)|1\rangle. \tag{2.37}$$

The first qubit is in state $H|0\rangle$, the second qubit is in state $|1\rangle$, and the state of the entire system can be "factored" into the states of the two individual qubits. Applying C to this state, however, yields the state

$$\frac{1}{\sqrt{2}}|01\rangle + \frac{1}{\sqrt{2}}|10\rangle, \tag{2.38}$$

which is entangled.

Measuring the first qubit produces values $|0\rangle$ and $|1\rangle$ with equal probability. If the value is $|0\rangle$, then the overall state of the system, projected onto the vector space with basis

$$\{|00\rangle, |01\rangle\}, \tag{2.39}$$

and then normalized, will be $|01\rangle$. The state of the system has become $|01\rangle$, so that any measurement of the second qubit returns $|1\rangle$ with certainty. Contrariwise, measuring $|1\rangle$ for the first qubit means that the second qubit has value $|0\rangle$ with certainty. Measuring either qubit fixes the value of the other, as well.

Examined independently, each qubit has equal probability of being $|0\rangle$ or $|1\rangle$, but this view does not describe the *correlation* between the two qubits: their values upon measurement are not independent random choices. This correlation is *non-local*, in that the two qubits could be spatially separated and will still display this perfect anti-correlation no matter when they are measured relative to each other.

2.7 Measurements

Entanglement provides another perspective on measurement. Consider what happens if a system is entangled with another system that is then permanently isolated from the original system. The

first qubit is the system of interest; the second qubit will be a "measuring device" qubit. Having created the entangled quantum state

$$\frac{1}{\sqrt{2}}|0\rangle|0\rangle + \frac{1}{\sqrt{2}}|1\rangle|1\rangle, \tag{2.40}$$

suppose that the second qubit is moved to an isolated location. At this point, no "local" transformation affecting only the first qubit can remove the entanglement. Applying a negation transformation N to (2.40) produces the state

$$\frac{1}{\sqrt{2}}|1\rangle|0\rangle + \frac{1}{\sqrt{2}}|0\rangle|1\rangle. \tag{2.41}$$

Applying the Hadamard rotation H to (2.40) produces the state

$$\left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle\right)\frac{1}{\sqrt{2}}|0\rangle + \left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle\right)\frac{1}{\sqrt{2}}|1\rangle. \tag{2.42}$$

Only states of the form

$$X \cdot \frac{1}{\sqrt{2}}|0\rangle + Y \cdot \frac{1}{\sqrt{2}}|1\rangle \tag{2.43}$$

can ever result from (2.40) by any sequence of transformations on only the first qubit. Fixing the second qubit means that the two terms of the superposition can never interfere with each other. Any eventual measurement of the first qubit will produce some state drawn from either X or Y, with equal probability, which will then induce the second qubit to "collapse" to $|0\rangle$ or $|1\rangle$, as though it had been measured directly.

Because there can be no interference between X and Y from the separation until the measurement, it is mathematically and physically equivalent to regard the measurement of the second qubit as taking place at the moment of separation. It is easy to check that exactly the same distribution of outcomes results from the post-transformation measurements if the qubits are also measured at the moment of separation or if they are not. The act of irrevocable entanglement is effectively equivalent to an immediate measurement.

Because quantum systems evolve in a unitary manner, no transformation creating entanglement is ever technically "irrevocable." But it is very easy to create certain transformations which are irreversible for all practical purposes. Macroscopic objects have enormous numbers of particles, each a quantum system that interacts with its many other quantum systems in complicated ways. The individual quantum effects of particles in macroscopic systems are dwarfed by the classical statistical effects of large numbers of particles interacting. Once a quantum system has become entangled with a macroscopic system, the transformation undoing that entanglement is prohibitively difficult to compute. Thus, entangling a quantum system with a classical one is, for all practical purposes, an irreversible entanglement, and therefore is tantamount to an immediate measurement. Put another way, if one wishes to measure a quantum system, entangling it with a macroscopic classical system suffices.

The flip side of this characterization of measurement is that it is important to avoid inadvertently entangling a quantum system with its environment during the course of its computation. The "interesting" quantum effects we have discussed at length are dependent upon the system's ability

to maintain a coherent superposition of states: measurement, by collapsing the state into one value, destroys such superpositions. Thus, because entanglement with the environment is tantamount to measurement, such entanglement must be avoided when measurements are not being deliberately taken. The hardest part of constructing a quantum computer is usually isolating the qubits from their (classical) environment well enough to prevent them from becoming entangled with that environment.

2.8 A Bit of Philosophy

This section is not strictly necessary for an understanding of the material in subsequent chapters. It is provided for the reader who is unsatisfied with the accounts of measurement given above, or who is curious about the philosophy of quantum mechanics.

Everyday experience appears to claim that the universe is not in the state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, but is most definitely in $|0\rangle$ or $|1\rangle$. In our first account of measurement (in order to ensure that measurements had consistent outcomes) we postulated that measurements are accompanied by a "collapse" into the measured state. These collapses are fundamentally non-unitary — they map states to probability distributions of states. Reconciling this view with the physical claim that quantum systems — which here include the entire universe, as the composition of many smaller, interacting, quantum systems — always evolve according to unitary constraints, is a major topic of heated debate in the philosophy of physics.

Much attention has been given, therefore, to attempting to explain precisely what must constitute a "measurement," why collapses do not really take place, why our everyday experience is in error, or some other resolution of this antinomy.

From the point of view of quantum computation, it is not necessary to resolve this question to carry out interesting computations. We have seen two accounts of measurement. In one, collapses take place when measurements occur, and those measurements take place when a sufficiently "large" device carries them out. In the other, no collapses take place, "everyday experience" is in error, and the universe exists in a superposition of the outcomes of an unimaginably large number of interactions. Although neither answer is philosophically acceptable, at least as presented here, they provide mathematically equivalent predictions of the results of "computations," considered as some sequence of unitary operations and measurements on an otherwise isolated quantum system. To the extent that any coherent philosophical point of view gives a definition of a "system" and a "measurement," the predictions of both our accounts will be consonant with that philosophy's predictions. In keeping with the pragmatic spirit of our discussion of quantum computation, these accounts are good enough.

Chapter 3

Quantum Hardware

3.1 Overview

There are certain requirements generally agreed to be the minimal set of operations a quantum computer must meet. They fall into two categories. First, there are the "gates" such a computer must implement: the mathematical operations that it promises to carry out on encoded qubits. As discussed in chapter 4, a set of gates which includes all one-qubit operations and a controlled-NOT gate will be universal. Thus, a quantum computer should be able to apply arbitrary rotations to single qubits and to alter the state of one qubit when a second qubit is in its $|1\rangle$ state.

Second, there are the physical requirements necessary for a quantum computer to implement in practice the transformations in its theoretical repertoire. These requirements are:

- Opacity: It must be possible to apply the above basic transformations to qubits without inspecting their state. If this requirement is not obeyed, the classical apparatus carrying out the transformations will become entangled with the quantum system, negating its uniquely quantum aspects.
- Size: The system must contain a number of distinct qubits. The estimate of number required for useful applications varies, but an implementation of Shor's factoring algorithm for numbers of interesting size would require several thousand qubits.
- Addressability It must be possible to operate individually on arbitrary qubits without applying the same transformation to other qubits in the system.
- Stability: The qubits must, within some "acceptable" error bounds, retain their current state when they are left alone, and the implementations of the transformations must not differ too much from their theoretical specifications. Unless a computer actually "does what it claims to," it is practically useless. This is a particular issue for quantum computers because their scale is so small and sensitive, and because traditional error-correction techniques are inapplicable to them.
- Measurement: Although the system should be opaque during operation, when a measurement is required, it must be possible to carry one out. It should ideally be possible to measure the state of (almost) any qubit whether directly or indirectly.

In those implementations under current experimental consideration, stability is the chief obstacle. The rapid decoherence of quantum systems interacting with their environments acts as an upper bound to the size of those systems. Addressability is less of a problem currently, but could become considerably more of one if the frontier of the scale of systems that can be stabilized advances.

Different implementations have different schemes for meeting these requirements, and these different techniques are better and worse at some of them. In this chapter we shall survey the leading proposed mechanisms for creating quantum computers in the laboratory, and discuss how well they have met the various criteria so far, in theory and in practice. Ion trap methods and those based on nuclear magnetic resonance technologies have enjoyed the most success and appear to be the most promising avenues of research, so we shall discuss them at greater length than some of the more exotic proposals.

3.2 Ion Trap

Consider a set of identical ions of precisely known mass and charge. These ions will respond in predictable ways to electric fields. In particular, if we can create a (two-dimensional) potential well, such that energy is required for them to move away from a certain preferred position, it will be possible to "trap" ions in that position. If the ions are then cooled to the point at which quantum effects become dominant, each individual ion can be used to hold one qubit.

The experimental construction of ion trap devices has been principally concerned with *linear* ion traps, in which the "bottom" of the potential well takes the form of a linear region in space. The electric field necessary to make ions gravitate to this line can be created by paralleling it with four electrodes (at the four corners of a square of which the preferred axis is the center). By putting an alternating-current potential through the electrodes at a high frequency Ω_v, a cylindrically-symmetric electric field is created, in which positive ions are drawn to the central axis[1] A small static (DC) voltage is then applied to the ends of the electrodes to prevent the ions from escaping through the "ends" of the trap. Since the ions are all positively charged, they repel each other and will space themselves out roughly evenly along the axis.

The above procedure suffices to confine the ions in an axial region of at most about a millimeter in diameter (starting off with "hot" ions with a great deal of energy). For quantum effects to dominate, they must be cooled to a temperature at which their possible energy levels are both quantized and predictable. Two distinct energy states of the ion are used, one representing $|0\rangle$ and the other representing $|1\rangle$. In practice, this means the ions must be cooled to very near their "ground state," the lowest admissible energy state, for the states to be stable and distinguishable.

This cooling can be carried out by means of laser pulses of very carefully-controlled radiation frequency, which encourage the trapped ions to release energy. Forced-evaporation techniques similar to those used to produce Bose-Einstein condensates are also theoretically attractive, but have had less success in practice thus far. In any case, once the ions are cooled, they are individually localized [2] within non-overlapping regions of diameter a micrometer or less.

[1] The particular choice of Ω_v depends on the charge and mass of the ion to be used. For $^{43}Ca^{+}$, for example, a few megahertz at an amplitude of about ± 9 volts would be used.

[2] At this scale, ions display significant wave-like features, so this localization is really a bound on the wave-function of the ion.

With this separation, it is possible to address any individual ion with laser pulses; these pulses can implement any desired unitary transformation on the single qubit represented by that ion.

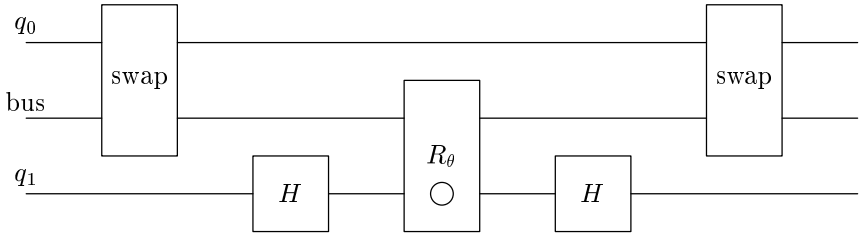
The trickier case is enabling qubits to interact; it is not possible, in general, to bring an arbitrary pair of qubits into proximity, since their locations are more or less fixed during the course of the computation. Instead, the usual approach is to create an additional qubit capable of interacting with any of the ion-stored qubits we choose. Because this qubit carries information between arbitrary qubits, it is referred to as a "bus" qubit.

Since the bus must be able to interact with any of the ions, the standard technique is to encode it using a property of the ions collectively: the vibrational motion (i.e. energy level) of the center-of-mass of the ions. The lowest energy level (the "ground state") is $|0\rangle$ and a particular higher-energy state serves as $|1\rangle$.

Because it is implemented so differently from the other qubits, the bus suffers from some limitations: it cannot be directly addressed in the same way the others can, nor can it interact with the others in very many ways. The options available, however, suffice for our purpose of conveying information back and forth between qubits stored in the ions.

It is possible to issue a laser pulse to an arbitrary ion which swaps the value of the qubits held by that ion and by the bus. More precisely, if it takes energy at a laser frequency ω for the ion to switch between its energy states and it takes energy at frequency ν to raise the center of mass from its ground state (representing $|0\rangle$) to its excited state (representing $|1\rangle$), then a laser pulse at frequency $\omega - \nu$ allows only two kinds of transition[3]. Either the ion goes from its $|1\rangle$ state to its $|0\rangle$ state and the center of mass goes from $|0\rangle$ to $|1\rangle$, or vice-versa.

It is also possible to make the phase rotations applied to qubits conditional on the state of the bus: only if the bus is in state $|1\rangle$ does the rotation take place. By using the standard trick for converting between bit-flips and rotations, these conditional rotations, together with unconditional rotations and the swaps described above, suffice to implement a controlled-NOT gate between two qubits stored in ions.



(3.1)

Since a controlled-NOT gate plus the complete set of one-qubit gates is universal for quantum computation, the ion trap is universal in simulating quantum circuits.

A qubit can be measured by illuminating its ion with radiation of a wavelength which would cause the ion to jump from its $|1\rangle$ state to a considerably higher energy state which does not otherwise participate in the computation. If the ion is indeed in the $|1\rangle$ state, it will resonate with the applied radiation and will fluoresce. If it is in the $|0\rangle$ state, no such resonance will take place.

[3]The duration of the pulses is also important, because it affects the amount of time under which the system evolves according to the induced Hamiltonian. For our purposes, though, we only need to consider the cases in which the pulses are time to cause evolution through π radians.

Thus, by seeing whether light is emitted by the ion in response to a particular applied wavelength, it is possible to measure the state of the qubit stored in that ion.

The main source of error for an ion trap device, at present, is experimental uncertainty. Small errors in the laser frequencies, vibrational motion of the ions, and imprecision in controlling the electro-magnetic fields within the apparatus all have the effect of "heating" the ions and raising their energy. If an ion gets too "hot" it may enter a state other than $|0\rangle$ or $|1\rangle$, thus introducing an essentially irreparable error into the computation.

It is these errors, which scale with the number of qubits and the number of operations carried out, which currently place limits on the scale and duration of computations using ion trap devices. Theoretical estimates, based on the degree of control available with present technology, are that for a ten-qubit system, no more than about 200 operations can be carried out. Experimental implementations, to date, have achieved substantially fewer operations than even this modest number.

The good news is that most of these issues are primarily ones of precise control over wavelengths and of the quality of the isolation of the experimental apparatus from its surroundings. With improvements in these technologies, the above limits may well be loosened. The next serious theoretical obstacle faced by ion trap computation is spontaneous photon emission: the odds that a cooled ion will spontaneously emit a photon (and thereby change energy level) depend principally on the length of time that the ion has remained in that state. The particular choice of ion and trapping frequency, as well as several other more-or-less fixed parameters set an upper limit on the number of computational operations that can be carried out in a given amount of time. Numerically, for computations taking a few hundred basic operations, the odds that one of the ions will emit a photon are negligible, but for computations on the order of a million operations, these odds become significant.

3.3 NMR

The nucleus of an atom that is part of a larger molecule will have certain resonant frequencies at which it is able to vibrate. These frequencies are determined by the structure of the molecule near the atom; they depend on factors that include the weights of the various nuclei to which it is bonded and the electrostatics of the molecule in the region near this particular nucleus. In general, for a molecule with asymmetries, the different nuclei will have different resonant frequencies.

Organic chemists have long exploited this phenomenon by employing the techniques of nuclear magnetic resonance (NMR) to determine molecular structure. A sample of an unknown organic molecule is placed in a solvent whose resonant frequencies are few, known, and unlikely to lie near those of any nucleus in the mystery molecule. Magnetic pulses over a range of frequencies are applied and those frequencies at which some constituent of the molecule resonates are noted. This information is then used to reconstruct the structure of the molecule: different components of organic molecules have different "signature" patterns of resonant peaks.

It is also possible to regard the different nuclei in a molecule as individual quantum systems each capable of storing a qubit. Each nucleus has a slight magnetic polarization μ. Information is encoded in the direction in which the nucleus is oriented. A stable magnetic field is applied to the

NMR chamber as part of the overall NMR process: a nucleus oriented towards the field is in the $|0\rangle$ state, while a nucleus oriented away from the field is in the $|1\rangle$ state.

Precisely applied radio-frequency pulses of the proper orientation, frequency, and duration will cause nuclei that resonate at that frequency to rotate their magnetic orientation Given our encoding for individual qubit states, most important single-qubit gates can be implemented with one to three pulses aligned with the x, y, or z axes (making them especially convenient to implement). Because the different nuclei in a molecule have different resonant frequencies, they are individually addressable; a nucleus will not respond to pulses off its resonant frequency.

Multiple qubit interactions are somewhat more complicated to describe. In outline form, the spins of two adjacent nuclei can be "coupled" to each other, so that *in the absence of pulses* their states will evolve in a way that depends on the value of both. By controlling the amount of time they are allowed to evolve in this way, two-qubit rotations can be implemented.

Measurement of a nucleus consists in applying a pulse at the resonant frequency along the $|0\rangle$-$|1\rangle$ axis: nuclei in these two states will produce resonance lines of opposite sign, so that they can be distinguished. This is the stage at which the bulk nature of NMR computation enters: in order to produce a detectable output, a large number of molecules are required. Typical experimental implementations use solutions with molecular concentrations on the order of 100 millimoles per liter. There has been some thought that smaller sample sizes might have certain advantageous properties (more easily cooled to under 1 Kelvin, smaller experimental apparatus, etc.), but at present sample size does not pose a serious obstacle to NMR computation.

Instead, the real difficulties with NMR computation arise from trying to scale up the number of qubits involved in a computation. Experimental work has focused on molecules with only two or three nuclei of interest. The resonant frequencies and the relevant coupling frequencies for these nuclei are easily determined, and the rest of the molecule essentially does not interact with them during the computation.

To support a hundred-qubit system, a molecule with well over a hundred atoms is required (since not every atom will have suitable resonance properties). Further, the resonant frequencies for each of these atoms (as well as a great many precise details about their interactions with each of the other atoms) must be known in order to calibrate properly the pulses, which will need to be at many different frequencies. To make matters even worse, there is no way to couple two nuclei from remote parts of the molecule directly: two-qubit gates may be applied only to adjacent nuclei, meaning that any computation will have to be arranged in a way that intimately depends upon the topology of the molecule being used.

The most promising response to this difficulty has been the idea of using polymers as the molecules in NMR computing. The advantage of using a polymer is that many of its nuclear sites are locally indistinguishable, so that the proliferation of resonant frequencies is limited and the topology of the molecule is radically simplified. The disadvantage of using a polymer is that many of its nuclear sites are locally indistinguishable, so that they are no longer individually addressable.

The way around this limitation proposed by advocates of the polymer strategy is to use a polymer with some distinguished nuclei that are addressable by different frequency pulses than the nuclei in the unadorned polymer units are. The question then becomes how to move arbitrary qubits to the nuclei where they can be individually manipulated.

One answer to this question is to use a polymer that has some pattern of individually adjacent nuclei which repeat, i.e. $\ldots ABCABCABC\ldots$, and then to issue pulses which will cause *all* of the nuclei of the same type to respond in the same way. For example, a set of pulses which swap the values of the qubit at an A nucleus and at its B neighbor, followed by a set which swap B with C and then C with A, will send every qubit at an A site to the next A site along the chain. This technique makes it possible to "rotate" different qubits into and out of an active site — a nucleus which can participate in the ABC resonances, but also has some other resonant frequency at which it can be individually addressed. This model is in some ways like a cellular automaton[4], in which the cells update in parallel in response to their local situations. It is also in some ways like a Turing machine, in which an active "head" moves along a passive "tape." Here it is the "tape" which moves, while the "head" remains stationary.

Although such a technique does promise truly scalable quantum computation, in the sense that it can be extended to an essentially arbitrary number of qubits, there are some enormous technical hurdles to this project. Finding a suitable polymer is a very difficult prospect.

Experimenters have had more success to date with NMR than with any other quantum computing technology: devices of two and three qubits have been constructed and have successfully carried out several of the important quantum algorithms, including Deutsch's and Grover's. These devices typically employ either a molecule with two atoms, or a molecule with two or three nuclei with resonance properties very distinct from those of the other nuclei in the molecule. Extremely little progress has been made towards realizing experimentally any of the polymer-based NMR schemes.

3.4 Photonics

The above proposals have taken the circuit model as a metaphor: the circuit becomes a sequence of operations applied to more-or-less stationary qubits. Optical quantum computation takes the circuit model at face value: the circuit is a physical device through which qubits travel from input towards output.

In optical quantum computation, qubits are stored in the physical state of individual photons, in particular, the *position* of those photons. Since photons will most definitely not sit still, they are instead fed through a fixed (and macroscopic) set of circuit elements which act on them in predictable ways. In the appealing *dual-rail* model, each photon is allowed two possible paths along which to travel: one of those paths is $|0\rangle$ and the other is $|1\rangle$.

The gates which act on a qubit are along one (or both) of its paths. A NOT gate, for example, just crosses the paths. Inserting a phase delay on one path but not the other implements a conditional phase-shift. A Hadamard rotation is a beam-splitter that evenly splits an incoming waveform along its two output paths. Fredkin and Toffoli gates are also straightforward in principle to create, making a photonic device a universal quantum computer.

Optical devices have several strengths as quantum computers. It is very easy to place photons in the superpositions needed for such devices. Single photons (as chargeless and massless particles) do not become entangled with their environments as easily as larger systems do. Also, the

[4]Lloyd, picking up on this idea, uses it to develop a proposal not just for quantum computation but for massively parallel quantum computation, which could be a way around the inefficiencies of micro-scale quantum computation.

devices involved have been successfully showing off simple quantum phenomena for years: optical experiments have been the prime source of the laboratory confirmations of quantum-mechanical wierdness which populate the pages of popular science magazines.

Most of these advantages, however, carry explicit tradeoffs. It is considerably more difficult to cause photons (as massless and chargeless particles) to interact than it is to cause larger systems to interact. Because photons travel at the speed of light, if minor imprecisions in the spacing of experimental apparatus cause two photon paths to the same point to have different lengths, their times of arrival can be so different as to prevent the necessary interference between them. The precision necessary to create larger quantum circuits will also require advances in single-photon technology significantly beyond that presently available, since these small errors cascade in unfortunate ways,[5], and current technology introduces a number of approximations to some of the gates.

One interesting proposal is to use a single photon to carry *multiple* qubits by taking any of 2^n paths if it simulates n qubits. This construction, since it grows exponentially with the size of the system, is clearly unsuitable for general large-scale quantum computation. However, as *simulation* technology for analyzing small quantum circuits, it avoids this exponential blowup by only considering small values of n. For small values of n, as well, the limits of optical technology are less restrictive, so that photonics might well be the technology of choice for confirming small-scale phenomena of quantum algorithms.

3.5 Quantum Dots

Quantum dots, if successfully implemented, would represent the ultimate achievement of solid-state physics: the successful creation of semiconductors that operate entirely at the level of single electrons. A quantum dot is a small region of semi-conducting material doped precisely so as to have room for exactly one extra electron. Qubits are then encoded by the states of the electrons on individual quantum dots. Of all proposals for quantum computation, quantum dots come closest to the theoretical model of a cellular automaton.

The physical descriptions of such systems are quite complicated, more so even than for the other models we have been discussing, nor is there general agreement on the best ways in which to implement specific transformations on the qubits stored in quantum dots. The presence or absence of an electrostatic barrier between two dots, it has been suggested, could selectively permit or prevent them from coupling, thus allowing for some multiple-qubit gates. The application of *very* precisely-controlled electro-magnetic pulses could also cause various evolution to take place in the state of an individual quantum dot. Given the extremely small tolerances for such transformations and the nanoscale at which they take place (since the dots must be close enough to each other to interact), it has been suggested that these barriers and pulses could be administered by auxiliary quantum dots which are themselves classically controlled.

There are some extremely serious obstacles to the implementation of actual quantum-dot computers, most having to do with the small scale at which it must be carried out and the stringent

[5]Error-correction schemes for quantum computers strive to correct errors in which one quantum state is substituted for another: $|0\rangle$ for $|1\rangle$, or $-|0\rangle$ for $|0\rangle$ for example. But the errors that accumulate in an optical device are errors of timing, for which no error-correction scheme can correct.

requirements on the physical properties of the substrate at the near-molecular level. The problem of measuring qubits is a tricky one: proposals involve exploiting phenomena like electron tunneling, in which an electron "jumps" from one location to another *through* an intervening barrier. Current semiconductor fabrication technology is not even close to good enough for the demands of quantum computation. Advances are also probably necessary in the technology required to effectively control quantum dots (by creating very carefully modulated and located electro-magnetic fields).

On the positive side, quantum dot technology is the direction in which semiconductor research is headed, of its own accord. The circuitry in use in present computers is pressing up against the scale at which quantum effects — especially electron tunneling — begin to become significant enough to appear as errors in classical computations. From the perspective of quantum dots, these "errors" are the real computation. Whether or not quantum dot technology will ever succeed in realizing quantum computation, it does mark an interesting frontier between more traditional computing technology and quantum computation.

3.6 Anyons

The usual fault-tolerance model for quantum computation is to take the errors from the "hardware" level as a given and to impose a level (or levels) of error-correction in "software." Kitaev's proposal for quantum computation using anyons is an attempt to create a quantum computer which is intrinsically fault-tolerant.

Schematically, the problem with encoding information in "ordinary" quantum properties like location or orientation of spin is that uncertainty effects and environmental interactions perturb quantum systems in ways which cause large variations in those properties. Anyons encode information in "topological" properties of quantum systems, which are exponentially insensitive to errors in position.

Anyons exist in a two-dimensional system, in which their motion is confined to a plane. To change the information carried by a particle, that particle must be carried entirely *around* another particle.[6] If the two particles are separated after this carrying-around, a great deal of decoherence can take place in their positions without affecting the information that the one has been carried around the other.

The constructions of qubits and gates from these details are rather technical — in particular, multiple particles are required to store a single qubit and the operations required to carry out the basic transformations are somewhat complicated — but an anyonic system can implement the necessary primitive gates to make it a universal quantum computer.

What is more of a concern is that an anyon-based quantum computer is, for all practical purposes, impossible to realize physically. First, because space is three-dimensional, the two-dimensional existance of anyons is something of an issue. More importantly, the number of spin dimensions required by an anyon carrying out "interesting" computation (a Toffoli gate) is very high: 60, in comparison with the more convenient two possessed by an electron. There is little

[6] Hence the description "topological": what matters is a topological invariant of a particle's history — its winding number — rather than being dependent on position alone

likelihood that anyons will be found in the laboratory any time soon, and there is even less hope of building a device capable of controlling and measuring them.

What may save anyon computation, however, are the techniques of *quantum simulation* in which one quantum system simulates another. Such techniques are not especially useful for other quantum devices: the simulator introduces error, much as the system it simulates would have done. With care, however, the fault-tolerance of anyons can be exploited to make this quantum bootstrapping work for simulating them. All that is necessary is that the errors introduced by the quantum system carrying out the simulation be the kind of errors to which anyons are insensitive. In particular, if a simulation system represents the position of an anyon by some error-prone property, the intrinsic insensitivity of anyons to errors in position minimizes the effects of the simulator's error. The errors introduced by the computational device are not errors to which the computation is sensitive.

Of the technologies described here, anyons are the furthest from reality at present, nor is it clear that quantum computation with anyons has much of a future. If quantum simulation of anyon systems can be made to work, then such systems have the potential to solve several other major problems of quantum computation in one fell swoop. This possibility alone makes them worth considering.

Chapter 4

Quantum Theory

4.1 Theoretical Models of Quantum Computation

In the previous chapter, we have characterized what information a quantum system can *store*. Now, we take up the related question of what functions a quantum device can *compute*. Specifically, given a quantum system with state space S, we ask which unitary transformations

$$T : S \to S \tag{4.1}$$

can be computed on one of the hardware models considered in the previous chapter by applying some sequence of primitive operations it supports. Since we would like these results to be, as far as possible, independent of the particular hardware model, we need first to pin down a good theoretical model of general-purpose quantum computation.

4.1.1 Quantum Turing Machines

Classically, the Turing machine has served as a reasonable universal model of computation. The basic description of a quantum Turing machine is due to Deutsch. In his view the classical Turing machine's infinite supply of tape squares, initially blank, corresponds to an infinite supply of qubits, initially in some "blank" state, say $|0\rangle$. A finite control, in general a quantum system, interacts with those qubits by engaging in carefully specified interactions with a finite number of them at a time — without loss of generality, one. This finite control can change which qubits it interacts with, alter their state (possibly while altering some of its own state) in a unitary manner, and make decisions about what instructions to carry out next.

Because the finite control is constrained to unitary evolution, it is not immediately clear how to implement such primitives as conditional control-flow. Nor is it immediately obvious how to represent an arbitrary quantum Turing machine "program" in a format that a quantum Turing machine itself could read and implement. A classical Turing machine expresses the idea of locality of individual operations by constraining the head to operate on one square at a time; a quantum Turing machine's interactions are similarly constrained. However, the constructions and reductions involved in implementing what seem like fairly elementary operations involve some more "global"

restrictions on the the transition function of a quantum Turing machine. Informally, the transition function is a mapping from configurations to configurations; for this mapping to be unitary, it must respect orthogonality across all possible configurations, something it is clear a great many "possible" transition functions will fail to do. The hardest problem is preventing multiple configurations from being mapped to the same configuration while still allowing different control-flow paths to reconverge, a necessary condition for any looping or reentrant control-flow.

The solution to this problem — and to a number of other problems — can be found in Bernstein and Vazirani's proof of the existence of a universal quantum Turing machine. Their proof revolves around establishing that quantum Turing machines can carry out the complete set of traditional Turing machine primitives in a manner susceptible to concise description. They give a sequence of reductions showing how quantum Turing machines can implement non-trivial control-flow, the execution of other quantum Turing machines as "subroutines," changes of basis, and even arbitrary unitary transformations with concise (polynomially-bounded) representations provided as input [1]. By developing a framework for quantum Turing machine designers to decompose problems into smaller subtasks and then compose the actions of machines which solve those subtasks, they effectively give a reduction from classical Turing machines to quantum Turing machines. The cost their solution imposes, however, is a substantial increase in the intricacy of the actual transition function used to implement simple-seeming primitives.

In the other direction, a quantum Turing machine can be represented as a state in a complex vector space evolving according to a specified unitary operator. Although this space is infinite-dimensional, only a finite number of these dimensions ever simultaneously have non-zero components. Thus, a classical Turing machine can simulate a quantum Turing machine to arbitrarily good precision, by reducing the problem of simulating a quantum Turing machine to carrying out a series to complex matrix multiplications, at times expanding the matrix as necessary. As a consequence, quantum Turing machines are Turing-equivalent devices.[2]

Although quantum Turing machines are universal by inheritance from the universality of classical Turing machines, they are also universal for quantum computation in a more direct sense. To the extent that quantum Turing machines are "programmable," they are programmable with the descriptions (in matrix form) of other quantum Turing machines. Further, any quantum system can be represented by a unitary transformation in matrix form. Such a transformation can be simulated on a quantum Turing machine. This argument makes quantum Turing machines universal for any quantum devices whose actions can be computably described, which is to say, any reasonable quantum model of computation.

4.1.2 Quantum Cellular Automata

Quantum cellular automata are cellular automata whose transition functions can be continuous (with complex coefficients). They correspond reasonably naturally to to quantum-dot-based imple-

[1] The polynomial bound is crucial only to the efficiency of the implementation. The transformations can easily be allowed to have exponentially-large descriptions, which suffices to admit arbitrarily good approximations to arbitrary unitary transformations, at the cost of the polynomial time-bound on the quantum Turing machine executing the transformation

[2] Given that quantum Turing machines are, because of the rules of measurement, probabilistic devices, this statement is slightly simplistic. Quantum Turing machines are Turing-equivalent in the same sense that traditional Turing machines with access to a source of random bits are Turing-equivalent: a traditional Turing machine can describe the probability distribution of their outputs with arbitrarily good accuracy.

mentations. They are also a massively parallel model, combining the parallelism of a large number of processors with "quantum parallelism," making them able to carry out certain operations very efficiently.

The above list exhausts the strengths of the quantum cellular automaton model. The same issue that initially appeared to plague quantum Turing machines — determining whether a specified transition function actually induces a unitary evolution of the system as a whole — is also a problem for quantum cellular automata. Unfortunately, while Bernstein and Vazirani gave an affirmative answer to the problem for quantum Turing machines, the corresponding answers for quantum cellular automata have thus far been mostly negative. For example, in one dimension, no homogeneous (having the same transition function at each cell) linear cellular automata can exist. If the homogeneity conditions are weakened, nontrivial quantum cellular automata become possible, but the issues involved in programming them are not theoretically well-understood.

Perhaps more seriously, there are not yet any interesting *uses* for quantum cellular automata. There exist several compelling and innovative algorithms for other general-purpose models, but quantum cellular automata have done nothing to distinguish themselves in this way. Classically, massively parallel models enjoy certain advantages over sequential ones; massively parallel quantum models enjoy those advantages over their sequential analogues, but they have yet to demonstrate that they possess advantages unique to the combination of "quantum" with "parallel." Since quantum cellular automata have demonstrated neither wide applicability nor especially pleasant theoretical properties, we turn our attention instead to a model which has shown both.

4.1.3 Acyclic Quantum Circuits

The basic idea of a quantum circuit formalism is that we regard a unitary operation which only (directly) affects n qubits as a *gate* that takes n inputs to n outputs. The *wires* in this model represent the values of qubits between the application of transformations that directly operate on them and are allowed to freely cross. The transformations are given a total linear order; the output of a gate may only be connected to the input of a gate which follows it in that order.

Further, each wire must connect exactly one output and exactly one input (including as "outputs" the inputs to the overall circuit, and vice-versa). Neither may there be any input or output not connected to a wire. These latter two constraints are necessary to ensure that the action of the circuit is unitary. To duplicate a qubit (by splitting a wire to multiple inputs) or destroy it (by failing to connect it to an input) would be a mapping between vector spaces of different dimension, which *a priori* cannot be unitary. Even if a destruction and a duplication are carried out simultaneously, it is easy to see that the transformation represented by these actions is not unitary, as it has neither a right and nor a left inverse. We will consider these constraints again later, in discussing connections to reversible computation and issues in quantum programming language design.

The formal semantics of an acyclic circuit are straightforward. If the gates of the circuit implement the transformations $G_1, G_2, \ldots G_k$ to the qubits specified as their inputs (extended to transformations on the full 2^n-dimensional space in the usual way), and $\sigma_0, \sigma_2, \ldots, \sigma_k$ are the permutation matrices [3] which carry the appropriate qubits to each gate, then the overall action of the

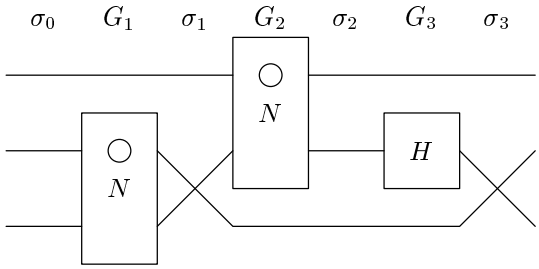
[3] In an actual implementation, the σ_i could physically interchange qubits, or could implement transformations to swap their values, or the G_i could act directly on the appropriate qubits "in place." All three possibilities have equivalent matrix forms

circuit is the transformation

$$\sigma_k G_k \sigma_{k-1} G_{k-1} \ldots G_2 \sigma_1 G_1 \sigma_0, \tag{4.2}$$

Conversely, more complicated unitary transformations can be carried out by factoring them into simpler ones which become the gates of the circuit.

Quantum circuits are also commonly specified in diagrammatic format. The following diagram is representative of the notation we will use for circuit diagrams:



(4.3)

Horizontal lines represent qubits, to which gates (boxes) and permutations (rearrangements of the lines) are applied; inputs enter at the left and outputs leave at the right. Here, the first two gates are controlled-NOT gates, and the final gate is a Hadamard rotation. The circles inside the two controlled-NOT gates indicate the qubit to which the rotation is applied (the qubit with no circle over it inside the gate is the control qubit).

4.1.4 Circuits and Turing Machines Compared

We will use the circuit model as our preferred theoretical model of quantum computation. While the advantages this model over the cellular automaton model are clear, the decision to consider quantum circuits instead of quantum Turing machines requires some justification.

The principal objection to such a decision is that circuits are inherently a *straight-line* model, in which the sequence of actions to be carried out on a given input does not depend on the input. A Turing machine, on the other hand, can take actions conditionally, based on the input or on intermediate values computed during the computation. This allows the Turing machine to handle inputs of different sizes, and to carry out nontrivial control-flow.

The reply to this objection takes two forms. First, quantum circuits are less powerful than quantum Turing machines only in the same weak sense that classical circuits are less powerful than classical Turing machines. Classically, if we allow families of circuits for problems, we can simulate any Turing machine M by a family of circuits $C_M(s,t)$, where s is the size of inputs to be considered, and t is the time-bound on the computation of the Turing machine. Thus, for any computation actually carried out by a Turing machine to produce an answer from some input, there exists a circuit which computes the same answer from the same input.

A similar result holds for quantum Turing machines and quantum acyclic circuits; Yao has shown that this simulation can be carried out with only polynomial slowdown for any function computable in polynomial time on a quantum Turing machine. Any problem (efficiently) computable on a quantum Turing machine is (efficiently) computable by acyclic quantum circuits. Thus, although

there is no individual "universal quantum circuit," no loss in intrinsic computational power results from dropping back to circuits.

The second reply is perhaps more important, given the present state of quantum computation. Quantum circuits are currently feasible to implement, whereas quantum Turing machines are not. Of the models discussed in the previous chapter, only quantum dots, even in principle, are capable of actually implementing a quantum Turing machine. The hurdles, theoretical and practical, to implementing quantum Turing machines in their full generality, are almost unimaginable.

The quantum Turing machine model requires that the finite-state control must not only contain some quantum state but in fact *be* a fully quantum system. The "head" in a quantum Turing machine must be able (by being in a superposition of distinct states) to act on spatially separated qubits simultaneously, and to carry out different kinds of interactions with them. On the other hand, the reality of present technology is that quantum systems require macroscopic classical control systems: large magnetic fields, precision lasers, optical beamsplitters, and the like. Creating quantum systems capable of controlling other quantum systems in the complicated ways required by the quantum Turing machine model is a goal that will remain out of reach for quite some time.

The bad news for quantum Turing machines, however, is not such bad news for quantum computation. The features of quantum Turing machines which make them so difficult to implement are not actually necessary for most quantum algorithms. The main quantum algorithms of interest are almost all straight-line algorithms: they use (implicitly) data parallelism, but not instruction parallelism. It is not necessary for a quantum computer to be able to make control-flow decisions in order to carry out interesting computations on it.

Although there have been a few algorithms proposed which rely upon such techniques,[4], in general they are neither necessary nor fully intuitive. In some respects, computations using such models are not entirely well-defined: particular, the question of when a computation that involves such a superposition of head states, some of which may be final while others are not, can be said to have "ended" is a very thorny one. It is also generally easier to reason formally about the finite-dimensional unitary transformation induced by a quantum circuit than it is to reason about the infinite-dimensional induced by a quantum Turing machine.

From the algorithm-design point of view, it is difficult to make effective use of the doors opened by allowing control flow based on the values of qubits in superpositions. As a general principle, some degree of similarity between the actions of different parts of the superposition of states of the finite control is necessary in order for them to interfere in useful ways. Without this interference, the quantum case devolves to the classically-implementable probabilistic one. Thus, we shall use acyclic quantum circuits as our model of general-purpose quantum computation: they have almost all of the good features of quantum Turing machines, but almost none of the poor ones.

[4]Kondacs and Watrous, for example, have given an algorithm for the quantum equivalent of a bi-directional finite-state automaton that allows it to recognize the language $a^n b^n$ by placing its tape head in a superposition of states at different places along the input. When restricted to one-way motion (which precludes the possibility of spatial separation of the pieces of a superposition), quantum finite-state automata are provably *weaker* than their classical counterparts. Therefore, in this case, allowing a quantum rather than a classical control system does increase the power of the model.

4.2 Quantum Computability Theory

4.2.1 Approximating Transformations

We now turn our attention to the matter of determining which unitary transformations can be computed by acyclic quantum circuits with given primitive operations. We seek quantum analogues to the result that any Boolean function can be computed by a circuit composed entirely of NAND gates, or that AND and OR gates together suffice to compute any monotone function. More precisely, given some finitely-generable set of gates $\mathcal{G}$, what (possibly) larger class of transformations $\mathcal{T}$ can be implemented by some circuit with gates drawn from $\mathcal{G}$?

A quick counting argument shows that it is impossible for any finitely-generable $\mathcal{G}$ to implement every element of U^S exactly. Such a $\mathcal{G}$ can give rise to only a countably infinite number of distinct circuits. On the other hand, the set of unitary transformations, even on a 1-dimensional complex vector space, has uncountable cardinality.

It *is* possible, however, to implement every element of U^S to arbitrarily good accuracy with certain very simple gate sets $\mathcal{G}$. That such approximations suffice follows because U^S has *metric topological* structure. That is, given a reasonable *metric* on U^S (a way of assigning to any two transformations a non-negative real "distance" between them), the actions of group multiplication (composing two transformations) and group inverse (taking the inverse of a transformation) are continuous with respect to that metric (do not map transformations which are close together to transformations which are far apart). We can specify the worst-case error with which one transformation approximates another as a metric [5] on U^S; we can then be confident that as long as the individual error of a sequence of transformations is bounded tightly enough, the error of the overall transformation they implement will also be bounded as closely as we require. Thus, the error in the state of the system after applying the overall transformation can also be bounded as closely as necessary, which effectively bounds the degree to which the probability distribution of actual outputs can differ from the distribution of outputs resulting from the "correct" transformation.

4.2.2 A Universal Quantum Gate

Given this broader definition of what it means for a gate set $\mathcal{G}$ to implement a class of transformations, it turns out that there exist $\mathcal{G}$ which suffice to implement *any* transformation in U^S. Such $\mathcal{G}$ are called *universal*. David Deutsch found a single three-qubit gate which, by itself, is universal

Consider N, the negation gate. Note that this gate leaves the vector

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \tag{4.4}$$

[5] Given two n-dimensional transformations T and V, one good metric regards them as matrices and lets

$$d(T,V) = \max\{|T_{ij} - V_{ij}|0 \leq: i,j \leq n\}.$$

We could also equally well let

$$d(T,V) = \sum_{0 \leq i,j \leq n} |T_{ij} - V_{ij}|,$$

or use a metric more specifically tailored to the manifold structure of U^S. In any event, these metrics are equivalent in that they induce the same topology on U^S

unchanged [6] We can thus regard N as a *rotation* about the axis defined by $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. Since $N^2 = I$, N rotates its input by an angle of π radians. We can also consider rotations by other angles; let R_θ be a rotation about this axis by the angle θ. [7]

Deutsch's insight was that to generate the *Lie algebra* of unitary transformations on a three-qubit space, it suffices to be able to apply appropriate R_θs in an appropriately controllable way. Let θ be any angle such that π and θ are *incommensurable*, that is, such that $\frac{\pi}{\theta}$ is irrational. Deutsch's universal gate D is the gate of three inputs which applies iR_θ to its third input if the first two inputs are both $|1\rangle$. We call this gate a "controlled-controlled-iR_θ," to indicate that the first two qubits *control* the application of iR_θ to the third.

The first step in Deutsch's construction is to show that with D, it is possible, to arbitrarily good accuracy, create a controlled-controlled-R_α gate for any angle α. Start by taking D^4, which can be rewritten as

$$D^4 = (iR_\theta)^4 = i^4(R_\theta)^4 = R_{4\theta}, \tag{4.6}$$

because R_θ is a rotation. Because θ was incommensurable with π, so is 4θ. By taking successive powers of D^4, it is possible to approximate arbitrary rotations, because the angles they rotate by will be densely distributed through $[0, 2\pi]$. [8]

In particular, it is possible to generate an arbitrarily good approximation to $N = R_\pi$. The controlled-controlled-N gate is called a *Toffoli gate*, and it is fundamental to universal reversible and quantum computation. As we shall discuss below, the Toffoli gate is universal for reversible Boolean functions. In particular, with Toffoli gates the eight basis states of the three-qubit system can be arbitrarily permuted, so that it is possible to bring any pair of basis states into the $|0\rangle_3$ and $|1\rangle_3$ positions where the iR_θ can be applied to them. Once there, they can be phase-shifted and rotated as necessary, then swapped back to their original "places." The details are slightly tedious, but not especially complicated.

Given the ability to implement arbitrary unitary transformations on three qubits at a time, it is not difficult to show how to make arbitrary transformations on n qubits. The easiest method is to use Toffoli gates to make versions of D which have more than two control bits. This version, for

[6] This vector is an eigenvalue of of this operator, with eigenvalue 1. Its existence is guaranteed by the spectral theorem, which in fact shows that every unitary transformation can be regarded as a rotation about some axis.

[7] It is possible to give R_θ in matrix form, but the derivation is more illuminating than the result. R_θ rotates about its axis $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ by an angle of θ. However, the subspace of vectors perpendicular to this axis is one dimensional. The rotation in this subspace takes the form of a phase shift (the only degree of freedom available), so that R_θ multiples the phase component of $\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$ (which spans this subspace) by $e^{i\theta}$. These two vectors are the Fourier basis, so that R_θ is diagonal in the Fourier basis with matrix

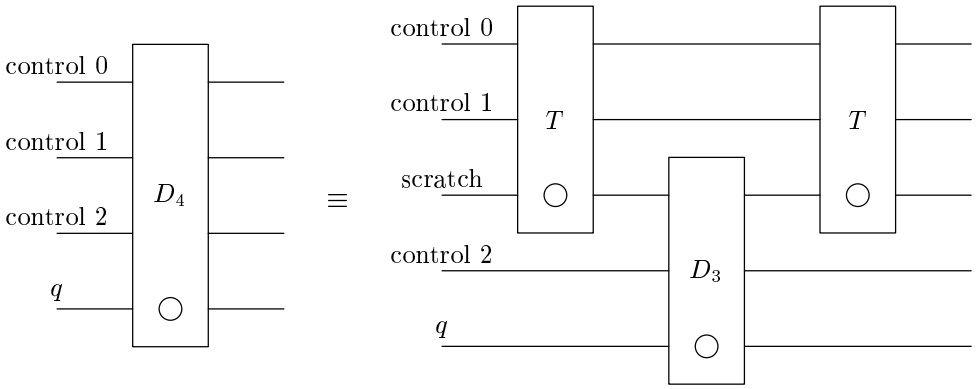
$$R'_\theta = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}.$$

To return to the computational basis, we conjugate R'_θ by the Hadamard matrix, which converts between computational and Fourier bases:

$$R_\theta = HR'_\theta H = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \begin{pmatrix} 1+e^{i\theta} & 1-e^{i\theta} \\ 1-e^{i\theta} & 1+e^{i\theta} \end{pmatrix}. \tag{4.5}$$

[8] This fact follows from the Weyl equidistribution theorem.

example (using a scratch qubit which starts at $|0\rangle$ and is returned to $|0\rangle$), has three control bits:



(4.7)

Slight modifications of the proof give above show that an n-bit Deutsch gate suffices to implement any unitary transformation on n qubits.

4.2.3 Other Universal Gates

Deutsch's result can be extended in two interesting directions. One is to follow through on the promise made in the previous chapter to show that a gate set $\mathcal{G}$ which includes a controlled-NOT gate and all one-qubit gates is universal. The other is something of a mathematical curiosity, but illuminating about the distinctive nature of quantum computation.

The reduction of a Deutsch gate to a controlled-NOT plus one-qubit rotations proceeds in several stages. First, the three-qubit gate D_3 can be optimized into a two-qubit version D_2. The key insight is that D_3 treats its first two inputs identically, suggesting that there might be a way to make the construction work with only one controlling bit instead of two. In fact, a controlled-R_θ gate that imposes a phase shift of $\frac{\pi}{4}$ radians (so that applying the gate eight times causes no net phase shift) instead of $\frac{\pi}{2}$ radians works. D_2 is a controlled S_θ gate, where

$$S_\theta = e^{i\frac{\pi}{4}} R_\theta = \sqrt{iR_{2\theta}}. \tag{4.8}$$

The proof that D_2 is universal is generally analogous to the proof for D_3, except that the construction of a Toffoli gate is more intricate.

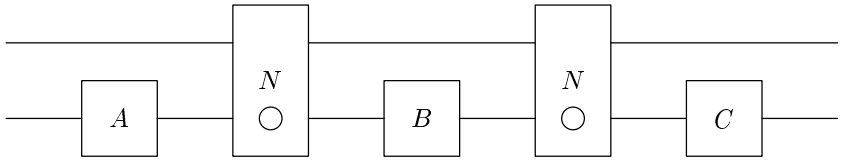
To complete the reduction, we show that a controlled-NOT gate and arbitrary single-qubit rotations suffice to implement a controlled-U gate, where U is *any* unitary transformation on one qubit. N can be considered as a self-inverse rotation, so that conjugating some other rotation B by N will change B into a different rotation $\overline{B}$. An arbitrary rotation U can be decomposed [9] into ABC such that

$$ABC = U \quad \text{but} \tag{4.9}$$

$$ANBNC = I. \tag{4.10}$$

[9] The proof relies on geometric properties of the group of unitary transformations on $\mathbb{C}^2$, considered as a group of rotations in three (real-valued) spatial dimensions. In essence, B is chosen so that $\overline{B}$ will be B^{-1}; A and C then either cancel with $\overline{B}$ or act together with B to carry out a full rotation by U.

The following circuit implements a controlled-U gate:



(4.11)

If the first qubit is $|0\rangle$, the controlled-NOT gates act trivially on the second qubit and the overall rotation $ABC = U$ is applied to the second qubit. If the first qubit is $|1\rangle$, the controlled-NOT gates flip the state of the second qubit, imposing the overall rotation $ANBNC = I$. This sequence of gates, therefore, acts as a controlled-U gate.[10] Since D_2 is a controlled-S_θ gate, a controlled-NOT gate and one-qubit rotations constitute a universal set of gates. [11] This result justifies the choice of this set of primitive gates as the goal of quantum computing hardware schemes.

The other extension of Deutsch's construction is to show that the Deutsch gate D_2 is not unique in being universal: *almost all* two-qubit gates are universal.[12] This fact suggests that the space of quantum transformations is indeed considerably "larger" than the space of classical ones, and that any journey into the space of quantum operations takes us decidedly beyond the power of classical operations. It also suggests that implementing all quantum operations is equivalent in difficulty to implementing any appropriately chosen operation of sufficient complexity, an argument with both reductive and constructive overtones. Some interesting mathematical glosses on Grover's and Shor's algorithms start from this observation about the usefulness of "almost any" quantum transformation.

[10] Technically, the first qubit must be flipped before and after this circuit, but this is trivial with one-qubit rotations.

[11] Even this result can be improved on: two appropriately-chosen rotations suffices. Given rotations A and B which do not commute, it is possible to generate, to arbitrary accuracy, any single-qubit rotation (by a similar argument to that used to show that a single R_θ suffices to generate, to arbitrary accuracy, any R_α).

[12] Technically, the set of non-universal two-qubit gates has measure zero in the set of all two-qubit gates. The proof, due to Deutsch, Barenco, and Ekert, proceeds along somewhat analogous lines to those in the proof for the Deutsch gate. They consider a "generic" two-qubit gate: one which has four eigenvalues which are incommensurate with π and with each other. They then show that its powers are dense in a torus of higher dimension than the one-dimensional $[0, 2\pi]$ considered for Deutsch's gate. Then, by switching the inputs before and after applying the gate, they generate a second set of points dense in that torus. At this point, new "gates" can be generated as linear combinations of the old ones or through taking their commutator; the proof then involves checking that these operations yield enough linearly independent gates to generate (up to arbitrarily small approximation) the full sixteen-dimensional space of possible transformations.

Chapter 5

Quantum Circuits

5.1 Overview

The decomposition of D_2 into controlled-NOT gates and single-qubit rotations, among its other features, is interesting because it breaks a quantum computation down into two sets of primitives: one that is superficially quasi-classical and one that is purely quantum. This kind of decomposition is a general feature of quantum algorithms: the computation can be divided into quasi-classical and quantum phases. The quasi-classical phases are often quite extensive, involving long sequences of operations that compute quite non-trivial functions. The quantum phases, on the other hand, are almost exclusively changes of basis. One very general pattern is that a quantum phase sets up a superposition of computational basis states, a quasi-classical phase operates (implicitly) in parallel on each of the states in this superposition, and a final quantum phase recombines the different elements of the superposition into a single answer.

In these algorithms, the changes of basis are usually algorithmically straightforward. The analysis of how they act on different states can be mathematically sophisticated, but the underlying sequence of gates is usually very simple. On the other hand, the intervening quasi-classical phase can involve the computation of functions with decidedly non-trivial circuits, even classically. In creating actual gate arrays to carry out such algorithms, it is crucial to know not just that these quasi-classical stages can be implemented with quantum gates, but that they can be implemented *efficiently*.

To be more precise, the quasi-classical transformations are those transformations which map elements of the computational basis only to elements of the computational basis. Such transformations define permutations of the computational basis. When applied to a superposition of basis states, they do not combine or divide elements of the superposition or cause them to interfere: the number of elements in the superposition and their relative phases and amplitude are unchanged by such a stage. The set of all such transformations on n qubits is isomorphic to the set of logically-reversible Boolean functions on n bits. *Reversible computing*, the study of such functions and their implementation, is older than quantum computation and has generated a number of results applicable to quantum computation.

Specifically, results from reversible computing provide affirmative answers to the following questions raised by quantum computation:

1. Can any Boolean function, reversible or not, be computed by a circuit built from reversible gates? If so, then any Boolean function classically computable will also be computable entirely within the quasi-classical fragment of quantum circuits. Up to some very mild restrictions on what it means for a reversible circuit to "compute" an irreversible function, the answer to this question is "yes." Further, the overhead (in time and space) of such simulations is, even in the worst case, not unseasonable.

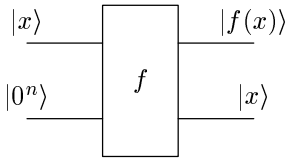
2. Are there (reversible) classical gates which are universal for the reversible Boolean functions? If so, then such gates could be used as building-blocks for quasi-classical quantum circuits in the same way that NAND gates are used as building-blocks for classical Boolean circuits. As mentioned above, the Toffoli gate is indeed universal for reversible functions.

3. Where an efficient algorithm for a reversible function exists — even if that algorithm requires irreversible steps — does there exist a fully reversible algorithm of comparable efficiency? If so, then such algorithms can be carried over to quantum architectures with little loss of efficiency. We shall see several explicit reversible constructions for familiar arithmetic functions which preserve the asymptotic running time of the irreversible algorithms they are based on.

We will now examine each of these questions in more detail.

5.2 General Reversible Simulations

A quick brute-force argument shows that any Boolean function — reversible or not — can be reversible evaluated with no time overhead. The construction is simple: keep copies of all arguments to the subfunctions (and to their subfunctions, recursively down to the gate level) as they are evaluated. Each n-bit gate we thus become a $2n$-bit gate. Such gates leave their first n inputs unchanged. The second n inputs are assumed to start off $|0\rangle$ and are then used to hold the "original" outputs of the function computed by the old, n-bit gate.



(5.1)

Given the pair $\langle x, f(x)\rangle$, the application of f to x can be "uncomputed" by resetting the second n bits to $|0\rangle$.[1] The space complexity of this algorithm, however, is $O(n)$ for an n-step computation, which is unacceptably high for general-purpose simulation. Fortunately, an algorithm due to Bennett, shows by example that, with more substantial cleverness, it is possible to do much better.

[1] To be picky, this is an insufficient specification, as it is only the correct behavior when the second n bits hold $f(x)$, where x is the value held in the first n bits. Technically, the "reversed" action of this gate needs to be specified for the cases when the second n bits hold some other value. The following argument, although slightly inelegant, takes care of those cases The gate will have some action (in the forward direction) on inputs when the second n bits hold values other than $|0\rangle$, even though the gate would not normally (by convention) be executed on such input. The gate thus determines a mapping from its possible inputs to some output values; this mapping will be bijective because the gate (being inductively built from reversible gates) is reversible. The inverse image of this mapping supplies the proper behavior of the gate run backwards, even if it is never used in such a way.

To simplify the analysis, suppose that an irreversible computer alters no more than a constant amount of its storage in each elementary step (one "unit"), and that an algorithm operates as a series of steps each of which takes one input to one output (any classical straight-line computation can be put in this form by appropriate currying). This assumption is reasonable for circuits, because within a given circuit, there is some constant upper bound k on the fan-in of gates in the circuit. One unit of storage is then k bits, since no gate can consume more values than it sees as inputs.

The key insight behind the algorithm is that running reversible gates in reverse releases the storage being used to hold their outputs. By backing up a portion of the computation, it is possible to reclaim unneeded storage space. Consider the following scheme for computing a two-stage function $g \circ f$, where g and f are each functions from one unit of input to one unit of output.

Action	Stored Values
(Initially)	x
Compute $f(x)$	$x \quad f(x)$
Compute $g(f(x))$	$x \quad f(x) \quad g(f(x))$
Run the gate for f in reverse	$x \qquad\quad g(f(x))$

At this point, only the values of x and $g(f(x))$ are being stored, which is the amount of storage we would require if we could compute $g(f(x))$ directly. Thus, the space needed to hold $f(x)$ is only required during the actual computation of $g(f(x))$ form x.

Note that the ability to uncompute $g(f(x))$ has now been lost, since the value of $f(x)$ is no longer cached. If the algorithm were to go on to compute $h(g(f(x)))$, it could not reclaim the space storing $g(f(x))$ in the same way that it reclaimed the space storing $f(x)$. With the values x, $g(f(x))$, and $h(g(f(x)))$ on hand, it appears to be stuck caching $g(f(x))$.

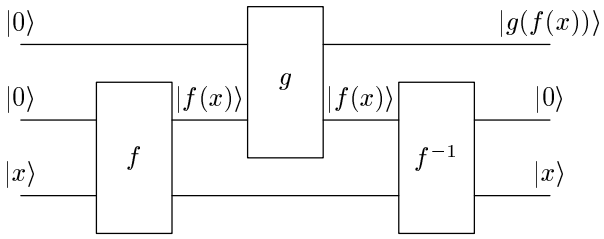
However, by paying the one-register space overhead involved in recomputing $f(x)$, the algorithm can, at any point it decides $g(f(x))$ is no longer useful, perform the following steps:

Action	Stored Values
(Initially)	$x \qquad\quad g(f(x))$
Compute $f(x)$	$x \quad f(x) \quad g(f(x))$
Run the gate for g in reverse	$x \quad f(x)$
Run the gate for f in reverse	x

This procedure is the exact reversal of the procedure used to calculate $g \circ f$ and exactly undoes that computation (with the same amount of temporary overhead). To emphasize this point, we might call this second procedure one for $(g \circ f)^{-1}$. The key is that this technique provides a way of packaging up procedures that compute and uncompute larger portions of a computation than individual gates.

The general case then, involves applying the "two steps forward, one step back" philosophy recursively. The computation is split into two equal-length halves. The final value of the first half is computed (using this recursive decomposition if it has length of more than a single step). From this final value of the first half, the final value of the second half can then be computed (again, using the recursive breakdown if necessary). Finally, the the "halfway" value is reclaimed by running

backwards the procedure used to compute it.



(5.2)

Now, let us analyze the behavior of this algorithm by considering $T(N)$, the time that a reversible simulation of an irreversible N-step computation takes, and $S(N)$, the space overhead involved. The time taken is the time to run two computations of length $N/2$ in the forward direction, and one in the reverse direction. Since computations take the same amount of time forwards as backwards, we have

$$T(N) = 3T(N/2), \tag{5.3}$$

a recurrence which solves explicitly (given a base case of $T(1) = 1$) to

$$T(N) = 3^{\log_2 N}. \tag{5.4}$$

On the other hand, the space taken by this recursive stage is at its greatest while we are hanging on to the intermediate value *and* also computing a computation of length $N/2$, so that

$$S(N) = 1 + S(N/2), \tag{5.5}$$

which solves explicitly to

$$S(N) = \log_2 n. \tag{5.6}$$

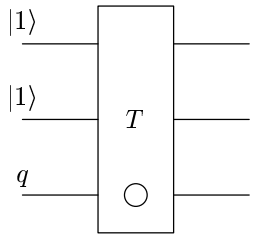
This is not the only tradeoff possible between time and space. To achieve a slowdown that is better than a change of exponent requires exponential amounts of space; to achieve better-than-logarithmic space consumption is impossible. Within these bounds, we can improve the base of the exponent in our expression for $T(N)$ to any value arbitrarily close to 2 while still keeping $S(N) \in O(\log_2 n)$ (just with increasingly large constants as the base approaches 2). To do so, we modify the above algorithm to compute n steps and then uncompute the first $n - 1$, making the algorithm run faster at the cost of more temporary storage space. In general, though, the above algorithm, unmodified, suffices for most purposes. Applied to a function computable in polynomial time, the construction provides a polynomial-time reversible algorithm for the same function, using only polylogarithmic extra space. From the point of view of complexity theory, this is generally good enough.

5.3 Universal Reversible Gates

Just as the two-bit gate NAND is universal for classical irreversible computation, there exist multiple three-bit gates which are universal for reversible computation (if given access to $O(1)$ "scratch" bits, initially fixed at zero or one, and which we are allowed to modify at will). One such gate is

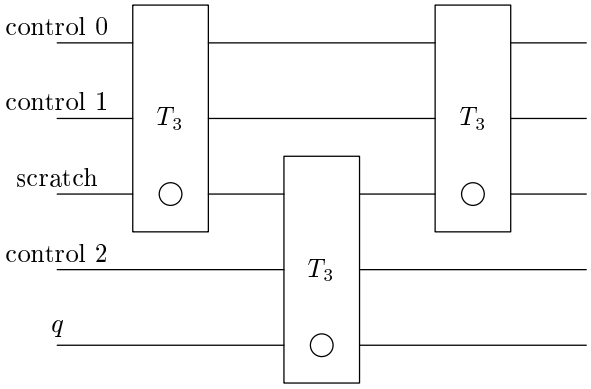
the Toffoli gate, a controlled-controlled-NOT, which inverts its third input if its first two inputs are both $|1\rangle$.

The proof of universality proceeds in several stages. First, several other useful gates can be built from 3-bit Toffoli gates. An unconditional NOT gate is easy, given two bits known to be $|1\rangle$:



(5.7)

It is also a straightforward matter to construct n-bit Toffoli gates, for arbitrary n (such gates flip their nth input if the other $n-1$ gates are all $|1\rangle$). The following circuit creates a four-bit Toffoli gate from three-bit Toffoli gates, using one bit of temporary space (assumed to be zero initially, and returned to zero at the end of the circuit):



(5.8)

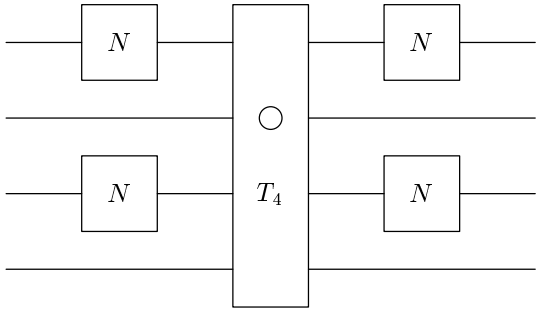
This construction clearly generalizes to allow us to construct an $n+1$-bit Toffoli gate from two 3-bit Toffoli gates and one n-bit Toffoli gate.

By applying NOT gates to the inputs and outputs of an n-bit Toffoli gate, it is possible to flip a single bit from an n-bit vector if and only if the other $n-1$ bits have a specific value. If we consider the computational basis as a set of 2^n elements,

$$\begin{array}{c} |0\ldots00\rangle \\ |0\ldots01\rangle \\ |0\ldots10\rangle \\ |0\ldots11\rangle \\ \vdots \\ |1\ldots11\rangle \end{array} \tag{5.9}$$

such an operation interchanges any two elements which differ only in a single bit, while leaving all

other elements unchanged. For example



(5.10)

interchanges $|0101\rangle$ and $|0001\rangle$. From here, the next goal is to show that any two elements of the computational basis can be transposed, no matter how many elements they differ in. Regarding the elements of the basis as the vertices of an n-dimensional hypercube, those elements connected by an edge are precisely those which differ in exactly one bit. The standard routing algorithm provides a path from any element to any other along the edges of the hypercube. A sequence of transpositions along these edges and back will exchange the two endpoints. For example, the following sequence interchanges $|0010\rangle$ and $|1100\rangle$ while leaving unchanged all other elements of the basis fixed:

$$\begin{aligned} |0010\rangle &\leftrightarrow |1010\rangle \\ |1010\rangle &\leftrightarrow |1110\rangle \\ |1110\rangle &\leftrightarrow |1100\rangle \\ |1110\rangle &\leftrightarrow |1010\rangle \\ |1010\rangle &\leftrightarrow |0010\rangle \end{aligned} \tag{5.11}$$

Because arbitrary permutations can be written as a product of transpositions and this last construction can generate any transposition using Toffoli gates, Toffoli gates can generate any permutation of the computational basis. Since the reversible functions on n bits are the permutations of the 2^n possible bit vectors, Toffoli gates are universal for reversible computation.

Although no two-bit gate is universal for reversible computation, the Toffoli gate is not the only three-bit gate that is universal. The *Fredkin gate*, which swaps the values of its second and third inputs if its first input is $|1\rangle$, is also universal (if given access to two supplies of bits initially known to be $|0\rangle$ and $|1\rangle$, respectively).

5.4 Reimplementing Familiar Functions

The construction of an arbitrary reversible function from Toffoli gates alone is a profligate one: as stated, it requires $O(n2^n)$ Toffoli gates. However, this construction builds circuits for some functions which have no polynomially-sized circuits, reversible or irreversible. For functions which are polynomially computable irreversibly, the Bennett simulation algorithm builds polynomially-sized reversible circuits, but increases both the running time and the space required by any algorithm. For many functions already known to be reversible, however, it is possible to do considerably better than this upper bound.

Where conventional algorithms for these problems involve a mixture of reversible and irreversible steps, the reversible algorithms for the same problems implement the reversible steps directly,

thereby avoiding paying the overhead involved in preserving around inputs to reversible steps. Only for those steps which are intrinsically irreversible are inputs retained. A number of other tricks, depending on the specific structure of the problem, are also employed.

5.4.1 Addition

Consider the standard sequential algorithm for adding two n-bit numbers $a_{n-1} \ldots a_1 a_0$ and $b_{n-1} \ldots b_1 b_0$ to make another n-bit number $s_{n-1} \ldots s_1 s_0$ (along with an overflow bit). A series of 1-bit additions are chained together with carries. Given a_i, b_i, and c_i, the carried bit from the previous addition, it is easy to s_i and c_{i+1} using only a few gates. The algorithm starts with $c_0 =$ and c_n becomes the overflow bit. Left over are the the carry bits, which we erase when we need them next.

The only problem involved in making this algorithm reversible is finding a way to reclaim the space taken by the carry bits when we no longer need them. But this is trivial; the logic that computed c_i can be run backwards. Thus c_n is erased first, then c_{n-1}, working backwards to c_1. The time overhead is $O(n)$, which is a substantial improvement on the general construction. The space overhead is $O(n)$ during the computation, but 0 after it.

Several improvements are possible to this algorithm. At the cost of overwriting one of the inputs during the course of the computation, the space overhead can be slashed even further. Rather than taking $|a\rangle|b\rangle|0\rangle$ to $|a\rangle|b\rangle|a+b\rangle$, a more space-efficient algorithm takes $|a\rangle|b\rangle$ to $|a\rangle|a+b\rangle$.

This technique is a very common one in the construction of reversible and quantum algorithms. In computing $f(x)$, where f is a reversible phase of some larger computation, it is common to overwrite x with $f(x)$, since it is easy to recover x from $f(x)$ if it is needed again. This technique even works when the algorithm for f contains irreversible steps; such steps can be replaced with reversible simulations, as in the adding circuit above. It also works when f takes multiple inputs and the value of one of them is reconstructible from the computed value of f and the rest of the inputs. In this case, the currying of f to function of one argument yields a reversible function, and we are justified in calling f reversible. The $+=$ operator in many programming languages is an example of such an f curried to become reversible.

5.4.2 Modular Addition and Beyond

Another algorithm illustrative of the techniques involved in designing efficient quantum analogues of classical algorithms is one for modular addition: given a and b, calculate $a + b \pmod N$ for some fixed $N \geq 2$. Considered as a curried function, modular addition is reversible. The problem is that the traditional algorithm — add a and b, and then subtract N if the result is too large — involves a reversible stage followed by an irreversible one. The challenge is to write a circuit that subtracts N if necessary while keeping enough information around to determine whether N needs to be added back in, if run backwards.

Vedral et al. solved this problem with a variant of the following algorithm, which assumes the availability of a binary encoding of N in a register which we will call r, and also a spare register, z, set to 0:

1. Given a and b, replace a with $a + b$ (just an in-place addition).

2. Compare N with $a+b$, storing the result in a temporary bit c (letting c be 1 if N is greater than $a+b$, and 0 otherwise).

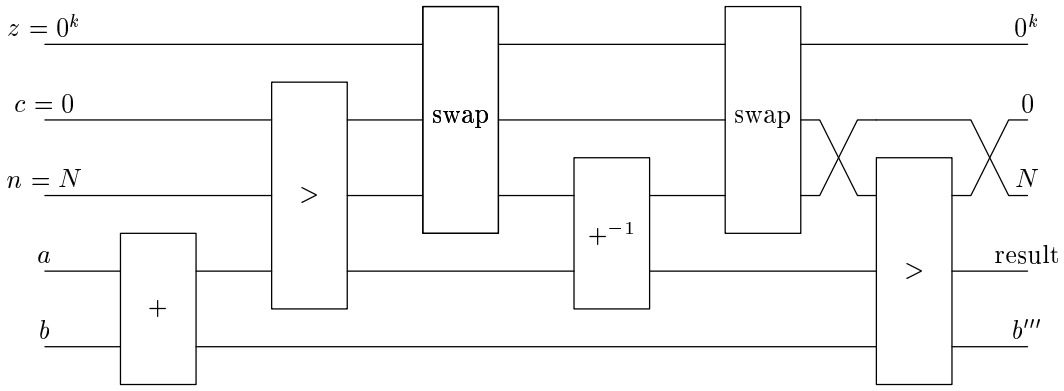
3. If $c=1$, then swap the contents of r with the contents of z. r now holds N if and only if $a+b \geq N$.

4. Subtract the contents of r from $a+b$, by running the addition circuit backwards.

5. Repeat step 3, swapping r and z if $c=1$. This undoes the effects of step 3, so that r now holds N and z holds 0, as they have been swapped either twice or not at all.

6. Compare b with $a+b$ and invert c iff $b > a+b$. This last step restores c to be 0.

The register that formerly held a now holds $a+b$ if we had $a+b < N$, and holds $a+b-N$ if $a+b \geq N$. Since we have returned all the auxiliary registers to their initial values, this algorithm works as advertised in computing $a+b \pmod{N}$. Subtractions can be implemented by running addition circuits backwards. Comparisons can be carried out by subtracting one value from another and checking the most significant bit for underflow. The following circuit diagram illustrates the above algorithm:



(5.12)

The trickery in this algorithm is that the *subtraction* cannot be a conditional operation: the algorithm cannot choose whether or not to subtract, as this would lead to an irreversible choice of computation path based on the result of a comparison. Instead, it makes the subtraction unconditional, but make the *swap* the conditional operation; the swap can easily be implemented with Toffoli gates, since it is effectively data-independent. [2]

More complicated operations are also within our inventory. Multiplication, in binary, is a series of controlled additions; for each digit in a, either add a bit-shifted version of b to the running total or add 0. Exponentiation is similarly a series of controlled multiplications. Both of these operations can be performed modulo N with analogous modifications to those used for addition modulo N. There are many possible optimizations, especially for space. The use of "carry-save" techniques to parallelize $O(n)$ n-bit additions, the use of the Fast Fourier Transform to speed up multiplications, and the use of multiplexing techniques to improve the efficiency of modular exponentiation are all possible.

[2] This statement is slightly unfair to subtraction and other "complicated" operations. By using 4-bit Toffoli gates with an extra "control" line, it is always possible to make any reversible program into a controlled program, which acts trivially if the extra control line is 0. This procedure does not scale so well to nested conditional statements, however, since it requires using Toffoli gates with more and more control lines. In general, it is usually more efficient to perform minor surgery on the algorithms we use so that they are amenable to straight-line execution.

Chapter 6

Quantum Programming

6.1 Problems of Quantum Programming

Given the kinds of computations quantum circuits can carry out, the next important challenge is determining how best to *express* the circuit for a given computation. This is the task programming languages have traditionally carried out, by providing a human-intelligible form for specifying computational actions. The benefits of high-level programming languages are well-established; the question is whether quantum architectures can also enjoy these benefits. As we shall see, the answer is largely yes.

6.1.1 Programming for Circuits

Classically, programming for circuits (instead of for a general-purpose von Neumann machine) is reasonably well-understood. Hardware designers are accustomed to using high-level description languages for encoding their designs.[1] Typically, these languages go far beyond merely enumerating the interconnections between individual gates in a complicated design. Languages like VHDL include support for abstracting away from the details of implementing individual components of a circuit in the same way that traditional languages include support for abstracting away from the details of implementing functions. The circuit-like aspect of quantum circuit models does not present any fundamentally new challenges.

6.1.2 Programming Unitary Transformations

Instead, it is the quantum aspects of quantum circuits that are more challenging. There are quantum computations on even a small number of qubits which have no classical counterparts. Such operations, like the Hadamard rotation and the conditional phase shift, are the basic building-blocks of most interesting quantum computations. Such actions generally involve

[1] Modern hardware-deign languages actually attack considerably harder problems than the ones at issue in specifying quantum circuits, because classical circuits can be, and frequently are, cyclic.

- manipulations of state (like phase rotations) that are classically undetectable,
- manipulations of superposition (like the Hadamard rotation) that are classically incoherent,[2] or
- the correlated manipulation of multiple qubits (like the Fredkin gate) in a way that does not lend itself to the multiple-input/one-input paradigm of many formal models of classical computation.

A good quantum programming language should make using fundamentally quantum operations convenient and encourage the programmer to use those operations the architecture supports.

These problems have also been largely solved. That a conditional phase rotation alters an individual qubit's state in an "undetectable" way is only a problem if one insists that all variables must be transparent and all their properties and fields must be visible at all times. Object-oriented programming rejects such demands, establishing a notion of opaque objects whose contents can only be inspected or altered through certain interface functions. An object-oriented approach to quantum programming treats qubits as objects: their contents can only be accessed by applying unitary transformations to them or by measuring them. A conditional phase shift, for example, alters the internal state of a qubit; there is no *a priori reason* why such an action should necessarily change the value of a subsequent measurement (which is just another action acting on a qubit). This approach matches our circuit metaphor very well; the "gates" which are placed across one or more "lines" become functions which operate on qubits, where the qubits are thought of as fixed in place

That certain quantum operations are classically bizarre in their effects is an issue that no programming language can entirely avoid. By being precise about what we really want from a quantum programming language, however, it is possible to make some progress. Where the "programming" approach breaks down is in the *semantics* of a quantum program. Due to effects of superposition and correlation between different qubits, the status of a program cannot be represented merely by the status of all the variables in the program. Further, each operation acts as a unitary transformation on the vector space of the entire computation, not just on the space of the qubits it directly names. On the other hand, the individual *operations* of the language cannot (and indeed must not) inspect the global status of the computation, in particular, information about different terms in a superposition.

That said, the real power of quantum computation springs from these non-local effects: by carrying out a series of straightforward and localized transformations, it is possible to implement a more interesting unitary transformation on the system as a whole. Programming languages are among the tools useful for specifying these local transformations in a clear and concise way. A good semantic description of a quantum programming language will allow us to reason precisely about the effects of a program. However, the programming language itself succeeds if it allows us to produce useful programs without having to explicitly write down the high-dimensional transformations they implement, in the same way that classical programming languages do not require us to explicitly list their effects on every possible input.

[2]The Hadamard rotation is classically absurd because can cause a computation in one state to split into a superposition of two distinct states which can nonetheless still affect each other.

Thus, to return to the issue of representing primitive transformations, such transformations can ultimately be given in the form of the matrix specifying their effects on the qubits they directly name. Every programming language has a level of primitive operations whose implementation is no longer expressed in the language. System calls, syntactic primitives like +, declarations of externally-defined functions, and assembly-language segments are all typical primitives for traditional languages.

For a quantum language, the primitives are those basic unitary operations permitted by the particular quantum implementation at hand. The sequence of NMR pulses implementing a controlled-NOT or a Hadamard rotation are effectively atomic units of computation from an programmer's point of view, just as the qubit is the atomic unit of storage.[3]

The internals of these primitives would be implementation-specific. On an NMR-based computer, such primitive would be translated into sequences of pulses. On an anyon-based computer, they would be translated into sequences of conjugations. For those doing formal semantics of such languages, the primitive operations would be encoded directly as matrices. For quantum simulators, operation each could also be associated with a range of possible errors. The point is that these primitives are the point at which the "machine-independence" of a quantum programming language stops. It is even possible to imagine implementations in which these programming primitives would automatically generate error-correction operations along with the higher-level functions they implement.

The final problem named above, that quantum functions, by affecting multiple qubits, require multiple return values, is the least serious. Numerous classical programming paradigms have been devised to enable multiple return values. Functions in languages with lists (or ad hoc lists built with pointers) can return a list of return values which is then destructured by the calling function: in languages with strong typing and pattern-matching, this process is particularly straightforward. Functions in languages with side-effects can "return" multiple values by modifying variables (or locations) passed to them. Declarative languages avoid the problem altogether, since a $2n$-place relation easily expresses the proper connection between n input qubits and n output qubits. We shall see examples in all of these styles below.

6.2 Preventing Non-Unitary Transformations

So far, we have considered computations possible on quantum computers that are impossible on classical architectures. However, there are irreversible classical computations on even a small number of bits which have no direct quantum counterparts. Such operations, like assignment, are often the basic building-blocks of many interesting classical computations. Although it is always possible to "compile" irreversible operations into a reversible form, the core of a quantum programming

[3]It is not currently necessary to pin down precisely which primitives they should include; these are details for the designers of particular languages for particular implementations (or for languages portable across several implementations). Our concern here is with establishing that the tools exist for such designers to design languages when the need arises, that there are no more fundamental problems that stand in the way. A reasonable set of primitives for a hardware-independent quantum language would probably include a controlled-NOT gate, a family of conditional rotations, and a family of conditional phase shifts, each of which would be defined differently, depending on the specific quantum computer being targeted. Other gates, like Toffoli gates or unconditional Hadamard rotations, could be implemented as syntactic sugar atop this primitive layer.

language should respect the limits of the architecture it runs on and discourage the programmer from even thinking in non-unitary terms. Ideally, these constraints should be *syntactic*: non-unitary operations should not correspond to valid programs.

6.2.1 Reversibility as Typing

Let us focus more closely on what causes a program to specify non-unitary actions. It fails to be reversible when it destructively overwrites a value. More generally, any step that consumes one or more values without producing other values sufficient to reconstruct the original ones cannot be undone. Provided these computed values are themselves conserved in a similar fashion, the original ones will be, also. A more subtle point is that it is impossible to produce *independent* copies of qubits. It is possible to produce copies, true, but these copies are entangled with the originals. Viewed as a constraint on operations, this fact is merely the time-reversed view of the previous constraint: a set of inputs cannot produce outputs with greater information content than the inputs. It is impossible both to throw away the values of qubits and to produce them *ex nihilo*. In both directions, if each step of a program is unitary, the program as a whole will be.

What this analysis suggests is that the requirement that quantum state evolve in a unitary manner can be rephrased as a condition on the permissible operations on qubits. The requirements of reversibility are a *type-based* constraint on the way in which qubits can be combined and manipulated. This way of phrasing the second constraint points the way. It reflects the classical approach usually taken in dealing with typing. The ML philosophy that "Well-typed programs cannot go wrong," suggests a similar dictum that "Locally-unitary programs cannot go wrong," for the quantum domain.

The approach taken in functional languages of considering variables as *names* rather than as storage locations turns out to be the most fruitful way of formalizing the right constraints. In a quantum context, variables become meaningful when they are associated with the contents of a qubit, i.e. when they are bound to the return values or to formal parameters of functions. Variables become meaningless again once the qubits to which they refer have been altered or possibly been entangled with other qubits, i.e. when they have been used as inputs to functions or been returned from a function. In the circuit metaphor, each piece of "wire" running from one gate to another is given a distinct name. Several functional languages — Linear LISP and Concurrent Clean — feature such constraints, although only Concurrent Clean's implements them as typing rules, using so-called *uniqueness types*.

6.2.2 Uniqueness Types

The Concurrent Clean approach is to allow, as a type modifier, a uniqueness attribute, denoted by prepending an asterisk to the type to be marked as unique. Stricter rules apply to variables flagged as having unique type than to ordinary variables. They may not be used as rvalues more than once within the scope in which they are defined, unless their uses are provably mutually exclusive (on two separate arms of an "if-then-else" statement, for example). Further, when their values are passed into another context, whether through let-binding or as arguments to a function, the variables which are bound to hold their values must also be declared unique, so that a unique value does not become non-unique through renaming.

Uniqueness types were originally introduced into Concurrent Clean, a lazily evaluated typed functional programming language, as a way of incorporating stateful objects into a purely functional context without assignment. The classic example is interfacing to a file; after calling

write(*file*,'*a*')

file no longer has the same value that it did before the write call. The traditional approach in functional implementations is to separate store locations from the values held in them, and to modify those values with destructive updates, at the cost of losing referential transparency. The idea behind uniqueness typing is that after the write, *file* no longer exists: what we have now is a different object, which requires a different *name*. The following code fragment expresses this idea:

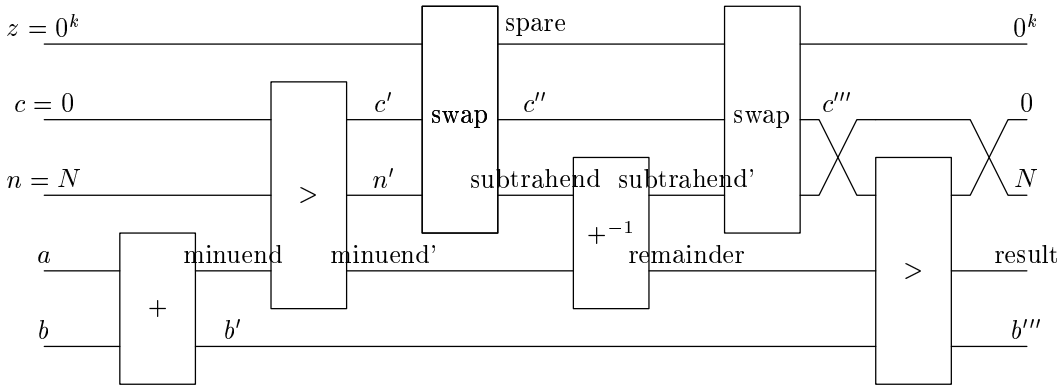
```
let  fileA = write(File,'a')
  in   ...
```

Here, *file* and *fileA* — and the first parameter and the return value of write — must be flagged as unique. The compiler can statically enforce this requirement as part of a unification-based type check stage. Although some care must be taken in getting exactly right the correct propagation of uniqueness values when components of higher-order types are marked as unique, consistent and coherent algorithms for checking uniqueness are generally straightforward. Concurrent Clean, using uniqueness types in place of assignment, has been used to implement several systems, including most of a small operating system.

The constraints of unitary time-evolution can be enforced by making qubits unique types and subjecting them to the additional restriction that they must be used *exactly* once in the scope in which they are defined, rather than *at most* once. [4] At this point, the correct behavior falls out more or less automatically. Consider the following functional pseudocode for the modular addition algorithm from the previous chapter:

```
function in_place_modular_add( qreg a,  qreg b) =
  let  n = prepare_reg(classical_n),
    z = prepare_reg(0),
    c = prepare_bit(0) in
     let (minuend, b') = in_place_add(a,b) in
        let (n', minuend', c') = compare_gt(n, minuend, c) in
           let (c", subtrahend, spare) = controlled_swap(c', n, z) in
              let (remainder, subtrahend') = undo in_place_add(minuend',subtrahend) in
                 let (c"', release_reg(classical_n), release_reg(0)) = controlled_sawp(c", subtrahend', spare) in
                    let (result, b", release_bit(0)) = compare_gt(remainder, b', c"') in
                       (result, b");
```

[4] This distinction reveals the fact, which we have heretofore swept beneath the rug, that uniqueness types, as they are used in Concurrent Clean, are really built on a framework of *affine* logic, in which resources cannot be duplicated, but can be discarded. Linear LISP incorporates true linear semantics, requiring exactly one use per definition. For purposes of discussion, though, Concurrent Clean is more useful because it allows both linear and unrestricted variables and formulates the distinction in terms of type.



(6.1)

The direct correspondence between this "program" and the circuit it denotes illustrates a number of desirable features of quantum languages. First, as promised, the one-to-one correspondence of names with "wires," combined with the constraints of uniqueness, automatically ensures the conservation of qubits. Second, pattern-matching suffices to bind the multiple values returned from a gate. The operations "prepare" and "release" handle the flow of values associated with temporary storage returned to its original state at the end of the modular addition function; we discuss their semantics below. Finally, note that it is a relatively simple matter to give the above language a formal semantics that specifies the action of each program on the Hilbert space of the system, since the language "compiles" so cleanly into circuit form, and the semantics of acyclic circuits are quite straightforward.

6.2.3 Linear Logic

Uniqueness types can be placed on a firm theoretical footing through the use of *linear logic* to give semantics to typing systems that employ uniqueness constraints. In linear logic, explicit accounting is made of the resources used in generating proofs; it is called "linear" because each resource must be used exactly once in the course of a proof. The contrast between a linear proof system and an ordinary proof system is most apparent in a sequent calculus setting.

Introduced by Gentzen, sequent calculi employ judgments of the form $\Gamma \vdash A$ to mean that A is provable using the resources contained in Γ. When used for typing, A usually has the form $x : \alpha$, meaning that x has type α, and Γ takes the form of a sequence of such statements, i.e.,

$$\Gamma = u_1 : \gamma_1 \quad , \quad u_2 : \gamma_2 \quad , \ldots \quad , \quad u_k : \gamma_k. \tag{6.2}$$

In traditional typing systems, Γ is a set of assumptions about the types of subexpressions, so the following additional rules of inference are generally used:

$$\frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta}\mathsf{exch} \qquad \frac{A, \Gamma \vdash B \quad \Delta \vdash A}{\Delta, \Gamma \vdash B}\mathsf{subst}$$

$$\frac{A, A, \Gamma \vdash B}{A, \Gamma \vdash B}\mathsf{contract} \qquad \frac{\Gamma \vdash B}{A, \Gamma \vdash B}\mathsf{weaken} \tag{6.3}$$

The exchange rule (exch) allows assumptions to be used in any order in proving a goal. The substitution rule (subst) means that the assumptions used to prove a subgoal A, plus the additional assumptions used to prove the main goal B from A, are together sufficient to prove B. The

contraction rule (contract) allows an assumption to be used multiple times in proving a goal. The weakening rule (weaken) allows the introduction of extra assumptions that are not used in the proof.

In a linear proof system, the contraction and weakening rules are omitted, so that assumptions must be discharged *exactly once* in the proof of a goal. Without contraction, a given assumption can only be used in one place; without weakening, it must be used in at least one place.

For example, consider the following fragment of a simple typing system for quantum expressions. Here α and β are type variables, and the primitive type Q represents a qubit.

$$\frac{}{\vdash |0\rangle : \mathrm{Q}} \qquad \frac{}{\vdash |1\rangle : \mathrm{Q}} \qquad \frac{}{\vdash \mathsf{cnot} : \mathrm{Q} \times \mathrm{Q} \to \mathrm{Q} \times \mathrm{Q}}$$

$$\frac{}{x : \alpha \vdash x : \alpha} \qquad \frac{\Gamma \vdash u : \alpha \quad \Delta \vdash v : \beta}{\Gamma, \Delta \vdash (u, v) : \alpha \times \beta} \qquad \frac{\Gamma \vdash u : \alpha \to \beta \quad \Delta \vdash v : \alpha}{\Gamma, \Delta \vdash uv : \beta} \qquad \frac{\Gamma, x : \alpha, \vdash u : \beta \quad \Delta \vdash y : \alpha}{\Gamma, \Delta \vdash \mathsf{let}\ x = y\ \mathsf{in}\ u : \beta} \tag{6.4}$$

In this system, a program u will be well-typed with type α iff

$$\vdash u : \alpha. \tag{6.5}$$

Then the expression

```
let  x = |0⟩ in
  let  y = |1⟩ in
    cnot(x, y)
```

will be well-typed with type Q × Q, according to the following proof tree:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x, \mathrm{Q} \vdash x : \mathrm{Q}} \quad \dfrac{}{y : \mathrm{Q} \vdash y : \mathrm{Q}}}{x : \mathrm{Q}, y : \mathrm{Q} \vdash (x, y) : \mathrm{Q} \times \mathrm{Q}} \quad \dfrac{}{\mathsf{cnot} : \mathrm{Q} \times \mathrm{Q} \to \mathrm{Q} \times \mathrm{Q}}}{x : \mathrm{Q}, y : \mathrm{Q} \vdash \mathsf{cnot}(x, y) : \mathrm{Q} \times \mathrm{Q}} \quad \dfrac{}{\vdash |1\rangle : \mathrm{Q}}}{y : \mathrm{Q}, x : \mathrm{Q} \vdash \mathsf{cnot}(x, y) : \mathrm{Q} \times \mathrm{Q}} \quad \dfrac{}{\vdash |0\rangle : q}}{x : \mathrm{Q} \vdash \mathsf{let}\ y = |1\rangle\ \mathsf{in} \cdots : \mathrm{Q} \times \mathrm{Q}}}{\vdash \mathsf{let}\ x = |0\rangle\ \mathsf{in} \cdots : \mathrm{Q} \times \mathrm{Q}} \tag{6.6}$$

On the other hand, the program fragment

```
let  x = |1⟩ in
  cnot(x, x)
```

will be ill-typed: the proof tree can get as far as

$$\dfrac{\dfrac{\dfrac{\overline{x, \mathrm{Q} \vdash x : \mathrm{Q}} \quad \overline{x : \mathrm{Q} \vdash x : \mathrm{Q}}}{x : \mathrm{Q}, x : \mathrm{Q} \vdash (x, x) : \mathrm{Q} \times \mathrm{Q}} \quad \overline{\mathsf{cnot} : \mathrm{Q} \times \mathrm{Q} \to \mathrm{Q} \times \mathrm{Q}}}{x : \mathrm{Q}, x : \mathrm{Q} \vdash \mathsf{cnot}(x, x) : \mathrm{Q} \times \mathrm{Q}} \quad \overline{\vdash |0\rangle : \mathrm{Q}}}{x : \mathrm{Q} \vdash \mathsf{let}\ x = |0\rangle\ \mathsf{in} \cdots : \mathrm{Q} \times \mathrm{Q}} \tag{6.7}$$

but the final expression has the undischarged hypothesis $x : \mathrm{Q}$. Without contraction, this proof fails. Since this program, because it attempts to make two distinct copies of x, is impossible to implement, the typing system rejects it. The following expression (in a context which names d as a qubit) is also ill-typed:

```
let  z = d in
  let  x = |0⟩ in
    let  y = |1⟩ in
      cnot(x, y)
```

This time, the tree gets as far as

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{x, \mathrm{Q} \vdash x : \mathrm{Q}} \quad \overline{y : \mathrm{Q} \vdash y : \mathrm{Q}}}{x : \mathrm{Q}, y : \mathrm{Q} \vdash (x, y) : \mathrm{Q} \times \mathrm{Q}} \quad \overline{\mathsf{cnot} : \mathrm{Q} \times \mathrm{Q} \to \mathrm{Q} \times \mathrm{Q}}}{x : \mathrm{Q}, y : \mathrm{Q} \vdash \mathsf{cnot}(x, y) : \mathrm{Q} \times \mathrm{Q}} \quad \overline{\vdash |1\rangle : \mathrm{Q}}}{y : \mathrm{Q}, x : \mathrm{Q} \vdash \mathsf{cnot}(x, y) : \mathrm{Q} \times \mathrm{Q}} \quad \overline{\vdash |0\rangle : q}}{x : \mathrm{Q} \vdash \mathsf{let}\ y = |1\rangle\ \mathsf{in} \cdots : \mathrm{Q} \times \mathrm{Q}}}{\vdash \mathsf{let}\ x = |0\rangle\ \mathsf{in} \cdots : \mathrm{Q} \times \mathrm{Q}} \tag{6.8}$$

before it blocks because the let rule needs to discharge the assumption $z : \mathrm{Q}$. There is no way to introduce that assumption into this proof tree without the weakening rule, as z is named in no subexpression of the let where it is bound. Since this expression attempts to discard the qubit represented by d, the typing system catches the leak and rejects it as ill-typed.

Linear logic has all the properties necessary to make it a solid foundation upon which to build a typing system. Its explicitly linear hypothesis-tracking can be embedded in proof systems without undue strain. Various theoretical results establish the consistency, completeness, and soundness of various systems of linear logic. It has also shown applicability to areas of computer science ranging from complexity theory to natural language semantics.

6.2.4 Preparation, Measurement, and Reversal

Another advantage of seeing the restrictions placed on qubits as type constraints is that such a view provides reasonably natural ways to describe the preparation and measurement of qubits. Preparing an initially-blank qubit is a function of no arguments which returns a qubit:

$$\mathsf{prepare} : \mathrm{unit} \to \mathrm{Q} \tag{6.9}$$

Note that the functionality of prepare functionality is essentially that of a heap allocator: we draw on an implicit pool of fresh qubits. As such, prepare is implicitly present in Deutsch's quantum Turing

machine model, with its infinite supply of blank qubits. prepare can also be seen (if wrapped with appropriate transformations) as a way of taking a classical value and embedding it in a quantum storage location.

The corresponding primitive for measurement takes a qubit as input and returns 0 or 1 to a classical variable:

$$\mathsf{measure} : \mathrm{Q} \to \{0, 1\} \tag{6.10}$$

The measurement operation "destroys" the quantum properties of the qubit in the process of entangling the qubit with a classical, implicitly macroscopic, system[5]. Its functionality, looking at the set of quantum values in a program, mimics that of an operation returning memory to the heap. Note that this operation is the *only* way to "promote" a quantum value into a classical one, since direct coercions are ruled out by the constraint that unique values cannot be stored in non-unique variables. This typing system preserves the distinction between $|0\rangle$ and 0.

As a consequence, it is impossible to use quantum values, which could be in a superposition of states, in boolean or integral contexts. This restriction immediately rules out making control-flow decisions based on quantum values without measuring those values (and thereby destroying any superposition they may be in). As control flow is always determined by purely classical values, so that the finite-state control which implicitly implements the program need never be in a superposition itself. Therefore, purely quantum programs written using formalisms implementing type-based constraints are automatically equivalent in computational power to acyclic quantum circuits. Classical programs with quantum subroutines retain their full power, but the quantum subroutines provably do not introduce or require quantum-mechanical behavior in the classical fragment. We shall return to this idea below.

Ensuring that the syntax and semantics of a quantum language guarantee logical reversibility has an advantage beyond merely preventing non-unitary operations. Any subroutine defined in such a language can be run in reverse; since many efficient quantum algorithms require the backwards computation of certain steps (as does the general simulation algorithm for irreversible computations), it would be convenient for the language to allow the execution in reverse of arbitrary quantum subroutines. This is easily accomplished: whatever construct a language would normally have for executing (purely quantum) code in a forwards direction can be supplemented by a parallel construct one for executing such code backwards.

6.3 Other Programming Paradigms

We have focused on embedding quantum operations in a functional context; there are also reasonable ways of understanding quantum operations in programming paradigms usually classified as imperative or object-oriented.

[5] Aharonov et al. have shown that any computation with intermediate measurements can always be converted into an equivalent computation with a single measurement at the end, even if the measured qubits are later used in the computation. Thus, whether "measurement" operations destroy qubits or not is an issue of expressive convenience, rather than affecting the strength of our programming model.

6.3.1 Object-Oriented Programming

Qubits support only a very limited range of operations when taken one at a time, and most "interesting" quantum operations involve multiple qubits. In a multiple-qubit transformation, the qubits are equal simultaneous participants; there is in general no way to assign to a transformation a primary qubit upon which it acts, or to decompose a transformation into a sequence of single-qubit actions. For this reason, a pure object-oriented system based on method invocation or message-passing does not work particularly well if qubits are treated as first-class objects.

However, as we have seen, the "object" metaphor itself is applicable. In this metaphor, the data objects in a program are equated with concrete physical objects which may have internal properties that are not directly observable, are difficult to move, can be difficult to duplicate, and can only be interacted with through interfaces on their "surface." For qubits, the requirements of storage and of quantum-mechanical constraints give this metaphor teeth: we can only interact with qubits in certain restricted ways. [6] By applying these primitives — generally quantum gates — to qubits, we can change their internal state in ways that are not directly observable. For example, here is the addition example from above, rewritten in a more object-oriented style:

```
procedure in_place_modular_add(qreg a, qreg b)
{
  qreg n(classical_n), z(0);
  qubit c(0);

  in_place_add(a,b);
  compare_gt(n, a, c);
  controlled_swap(c,n,z);
  undo in_place_add(n,a);
  controlled_swap(c,n,z);
  compare_gt(a,b,c);
}
```

In this style, names refer to qubits instead of to wires. Such a convention makes explicit that the same storage locations hold both the outputs and the inputs of a gate. We are focusing now on gates which do not compute values, but instead apply changes to a set of qubits: we have switched from *functions* to *procedures*. [7]

6.3.2 Imperative Programming

This change is also one of the key changes in going from a functional language to an imperative one: the transition in emphasis from the *values* functions compute to the *side effects* of functions. Here is the same example, rewritten in imperative style:

[6] For example, object-oriented languages are frequently hostile to direct copying of complex objects: by default, only references to them may be duplicated. This constraint is quite appropriate for qubits.

[7] If we consider being "classical" as an abstract class from which all classical objects inherit, then object-oriented notions of inheritance and subclassing neatly enforce the typing constraints forbidding promotion of quantum to classical objects. The semantics for Concurrent Clean develop a notion of subtyping between unique types and their non-unique counterparts.

```
procedure in_place_modular_add(qreg a, qreg b)
{
  qreg n = prepare_reg(classical_n);
  qreg z = prepare_reg(0);
  qubit c = prepare_bit(0);

  in_place_add(a,b);
  compare_gt(n, a, c);
  controlled_swap(c,n,z);
  undo in_place_add(n,a);
  controlled_swap(c,n,z);
  compare_gt(a,b,c);
}
```

Viewed in this way, qubits become, ironically, *anti-functional* datatypes: they can *only* be operated on procedurally, as opposed to classical datatypes which support functional access. In this respect, qubits are second-class objects.

This suggests another interesting way to think of quantum architectures in the computational models we have been using: to a classical computer we attach a "quantum coprocessor" which can store qubits and implement on them operations it is instructed to perform by the main processor. The main CPU and the coprocessor communicate along classical channels, and all control decisions are made by the main CPU. On the other hand, the coprocessor is the only piece which can actually "look inside" the qubits in the system or manipulate their state coherently. In this respect, the key difference between a quantum device and a floating-point coprocessor is that the latter (with current technology) lives on the motherboard, whereas the former requires large and specialized laboratory apparatus.

Shor's quantum algorithm for factoring involves carrying out a quantum computation which depends intimately on the number n to be factored. The metaphor Shor uses is that we "construct" a quantum circuit given n. Although Shor is careful to establish that this construction is efficiently computable, given n, the implication is that the construction of the circuit is an offline activity, while its execution is an online one. On a software level, we see a similar idea if we think of writing a program that involves some classical parameters to a program with a quantum aspect. The "construction" of the quantum circuit amounts to partially evaluating the program with respect to the classical parameters. In the case of Shor's algorithm, this partial evaluation results in a purely quantum program embedded within a small classical control loop that repeatedly executes the quantum program until it succeeds.

6.3.3 Declarative Programming

Declarative programming is already capable of expressing reversible functions in a natural manner: if a function is specified in terms of reversible functions, it is guaranteed to be reversible. The advantage of a declarative style for quantum programming is that there is no need for an additional language construct for reverse execution: calling predicates with only the "output" values instantiated automatically reverses the circuit the predicate represents. Here is the same example from above, rewritten this time in declarative style:

```
modular_add( a, b, result) : –
  add(a,b,raw_sum),
  compare_gt(prepare(classical_n),raw_sum, c),
  conditional_swap(c, prepare(classical_n), prepare(0), subtrahend,spare),
  add(result,subtrahend,raw_sum),
  conditional_swap(c, subtrahend, spare, prepare(classical_n), prepare(0)),
  compare_gt(result,b,c).
```

Some of the typical advantages of declarative programming do not carry over to the quantum domain. For example, the ability to specify functions logically and rely upon backtracking to find a solution does not fit well with the straight-line requirements of quantum circuits. Additionally, the limits of name-equivalence can be somewhat surprising. In the example above, b is both an input to and an output of the circuit, a fact the declarative-style version exploits by using only one name/parameter to represent that value. However, this technique cannot be used indiscriminately, or two distinct qubits may be unified to the same uninstantiated name, a violation of the no-cloning restriction. It is more useful to treat the declarative style as a style for expressing quantum circuits, rather than expecting all the useful features of declarative programming in the classical domain to carry over to the quantum setting.

Chapter 7

Quantum Algorithms

7.1 Quantum Oracles

Before we can launch into our discussion of quantum algorithms, we should first take care of one last important definition, that of a *quantum oracle*. A classical oracle, in its simplest circuit-accessible form, is a black box with n inputs and m outputs which computes some function

$$f : \{0,1\}^n \to \{0,1\}^m. \tag{7.1}$$

We are usually concerned with the number of queries to the oracle and the amount of additional processing time required to extract some piece of information about f.

To take a traditional example, f might be a function which treats its input as a truth assignment to n variables and returns 1 if and only if that assignment satisfies a boolean formula hard-coded in f. One of the major questions of interest in computational learning theory has been determining for which classes of boolean formulae some algorithm can reconstruct the hidden formula hidden inside the black-box, while using only a polynomial number of queries.

A quantum oracle differs from a classical oracle in two ways. First, in order that the oracle can be reversible, it will be an $n+m$-input (and thus $n+m$-output) gate. By convention, the first n bits will be input, and the last m will be output. The oracle acts as

$$|x\rangle|y\rangle \Rightarrow |x\rangle|y \oplus f(x)\rangle. \tag{7.2}$$

The oracle will be self-inverse, because two successive bit flips cancel.

Second, the quantum oracle, as its name implies, accepts quantum queries. For classical queries — inputs which are elements of the computational basis — it will of course respond just as the classical oracle would have. For quantum queries — inputs which are superpositions of more than one element of the computational basis — the oracle will act linearly on that superposition, following the usual rule (equation (2.5)). We can thus consider the oracle as a gate implementing a (possibly unknown) unitary transformation O_f.

The internal operations of the oracle do not concern us, except so far as there are certain requirements these operations must satisfy. The oracle, although it may have mutable state, must

not expose that state to the outside world. The oracle may not accidentally retain state or have internals that are observable, even in principle. By the standards we have been applying to quantum devices, this means that the oracle itself must also be a fully quantum device. It will not suffice to connect a quantum computer up to a classical oracle; the oracle must be as precisely and carefully controlled as the computer that queries it. It will be useful to keep this fact in mind during the following discussions of algorithms, as a check on the classes of problems that may or may not be effectively solvable.

7.2 Deutsch's Problem

We begin our discussion of quantum algorithms by considering *Deutsch's problem.* Although it is something of a toy problem, it illustrates, the features that our later, more complicated, algorithms will exploit.

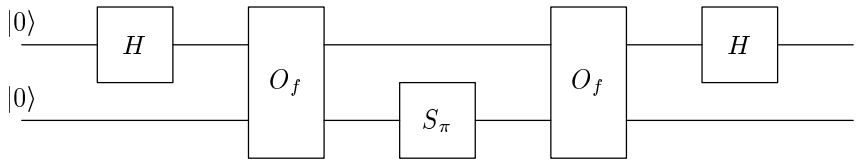
In this problem, we are given a function $f : \{0,1\} \to \{0,1\}$, and we are asked to determine whether $f(0) = f(1)$. Classically, two queries are both necessary and sufficient. This problem, however, can be solved with only one quantum query. We will first give a two-query algorithm, and then show how those two queries can be combined into one.

7.2.1 The Two-Query Algorithm

The following is a two-query algorithm for Deutsch's problem, using two qubits, one for input to the oracle and the other for output:

1. Starting from $|0\rangle|0\rangle$, apply a Hadamard rotation to the first (input) qubit.
2. Pass the two qubits to the oracle.
3. Apply the conditional phase shift operator S_π to the second (output) qubit.
4. Invert step (2) by running the oracle on the the two qubits again.
5. Invert step (1) by applying another Hadamard rotation to the first qubit.

This algorithm terminates with $|0\rangle$ in the first register if f is constant, and $|1\rangle$ if f is balanced. The following circuit illustrates its actions:

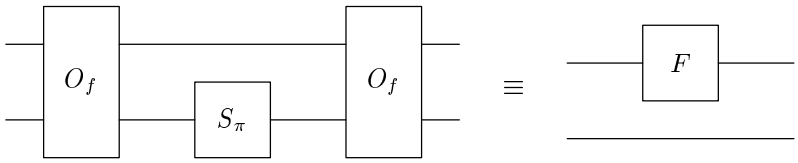


(7.3)

To see why this algorithm works, it is easiest to work inside-out. Let F be the the transformation implemented by steps (2) through (4):

$$|x\rangle|0\rangle \xrightarrow{O_f} |x\rangle|f(x)\rangle \xrightarrow{S_\pi} (-1)^{f(x)}|x\rangle|f(x)\rangle \xrightarrow{O_f} (-1)^{f(x)}|x\rangle|0\rangle. \tag{7.4}$$

Provided the second qubit starts in state $|0\rangle$ (which it does in the above algorithm), F acts only on the first qubit.

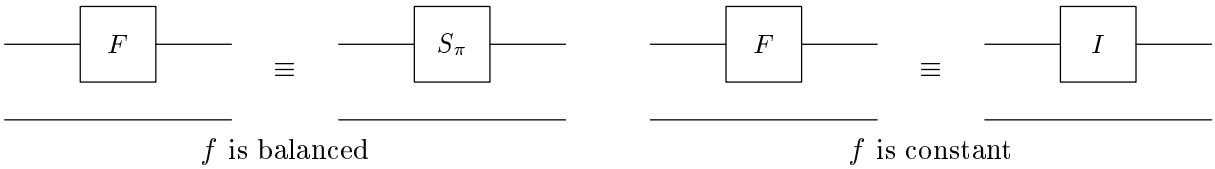


(7.5)

Further, F is diagonal in the computational basis: it imposes a phase shift based on the value of $f(x)$ but otherwise leaves untouched both $|0\rangle$ and $|1\rangle$. In matrix form,

$$F = \begin{pmatrix} (-1)^{f(0)} & 0 \\ 0 & (-1)^{f(1)} \end{pmatrix} = (-1)^{f(0)} \begin{pmatrix} 1 & 0 \\ 0 & (-1)^{f(0)+f(1)} \end{pmatrix}. \tag{7.6}$$

The leading constant is irrelevant: it applies equally to $|0\rangle$ and $|1\rangle$ and therefore constitutes a global phase shift. If f is balanced, then $f(0) + f(1) = 1$, and F simplifies to S_π. If f is constant, then $F = I$. F therefore either leaves its input untouchede or imposes a relative phase shift of π on the $|1\rangle$ component of its input.



(7.7)

If $F = I$, then the two Hadamard rotations cancel each other, making the entire algorithm act trivially on its input. If $F = S_\pi$, the Hadamard rotations convert this conditional phase shift into a bit flip, so the entire algorithm inverts its input. Symbolically,

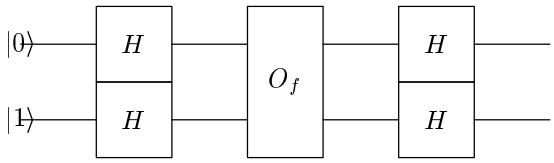
$$HFH|0\rangle = \begin{cases} HIH|0\rangle = H^2|0\rangle = |0\rangle, & f \text{ is constant} \\ HS_\pi H|0\rangle = N|0\rangle = |1\rangle, & f \text{ is balanced.} \end{cases} \tag{7.8}$$

7.2.2 The One-Query Algorithm

F is the heart of the above algorithm, but that algorithm requires two oracle queries in order to translate the oracle's bit-flip into a phase shift. By using a Hadamard rotation to convert the bit-flip into a phase shift, however, the following algorithm requires only one query to the oracle:

1. Starting from $|0\rangle|1\rangle$, apply a Hadamard rotation to the first (input) qubit.
2. Apply a Hadamard rotation to the second (output) qubit.
3. Pass the two qubits to the oracle.
4. Invert step (2) by applying another Hadamard rotation to the second qubit. again.
5. Invert step (1) by applying another Hadamard rotation to the first qubit.

This algorithm has the following circuit representation:



(7.9)

Steps (2) through (4) are another implementation of F, using only one oracle query.

Another form of the bit-flip/phase-shift equivalence expressed in equation (2.26) is that $NH = HS_\pi$. Therefore

$$NH|1\rangle = HS_\pi|1\rangle = -H|1\rangle. \tag{7.10}$$

Consider, as in (7.4) the actions of the transformation HO_fH acting on input $|x\rangle|1\rangle$:

$$|x\rangle|1\rangle \xrightarrow{H} |x\rangle(H|1\rangle) \xrightarrow{O_f} (-1)^{f(x)}|x\rangle(H|1\rangle) \xrightarrow{H} (-1)^{f(x)}|x\rangle|1\rangle. \tag{7.11}$$

Provided that the second qubit starts in the state $|1\rangle$ (which it does in the single-query algorithm), this transformation is therefore a faithful implementation of F.

7.2.3 Features of the Algorithm

There are a few essential features of this algorithm it is worthwhile to tease out. First, from an information-theoretic point of view, solving Deutsch's problem requires extracting one bit of information about f: whether it is constant or balanced. The problem with classical queries is that the information they provide is useful for answering questions about the individual values of $f(0)$ and $f(1)$, not just whether than whether $f(0) = f(1)$. On the other hand, the query in the single-query quantum algorithm provides no information as to the actual value of $f(0)$, only whether it is the same as $f(1)$. The quantum algorithm can issue a query that corresponds more precisely to the question being asked.

Speaking informally, we might say that the query issued by this algorithm assigns equal amplitudes to the terms $|0\rangle$ and $|1\rangle$. Since each query represents one bit of information, the algorithm obtains half a bit of information about the value of $f(0)$ and half a bit of information about $f(1)$. This information is stored in the first qubit in the phases of $|0\rangle$ and $|1\rangle$, so that each term in the superposition holds half a bit of information. The Hadamard transformation then causes the information about whether $f(0) = f(1)$ to interfere constructively, and the information about the specific values of $f(0)$ and $f(1)$ to interfere destructively.

Every other quantum algorithm we will encounter will use a variant on this technique. The specific pattern of interference is usually the most interesting part of a quantum algorithm. Usually, the complicated part of designing a quantum algorithm is finding a transformation which causes constructive interference for exactly the right term of a superposition.

Deutsch's algorithm is also our first example of an *amplification* algorithm. The two possible answers — "f is constant" and "f is balanced" — are encoded as $|0\rangle$ and $|1\rangle$, respectively. The state $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ is a uniform approximation to both $|0\rangle$ and $|1\rangle$. The oracle query amplifies that

term in the superposition which is an approximation to the correct answer. In Deutsch's algorithm, one oracle query suffices to increase the amplitude of the preferred term to 1; in Grover's algorithm, multiple queries are required.

Another way to express the advantage enjoyed by the quantum algorithms is to say that they exploit "quantum parallelism." By creating the superposition $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$, a quantum algorithm uses superposition to calculate every value of f "in parallel." The final Hadamard rotation then "adds" together phase components without explicit calculation, so that interference effects can be used to carry out computation.

7.3 The Bernstein-Vazirani Problem

Bernstein and Vazirani, following up on an idea of Deutsch and Josza, generalized the Deutsch problem to functions of more than one qubit.

A *parity function* f_a on n inputs is specified by giving an n-bit vector a. For any input x,

$$f_a(x) = x \cdot a, \tag{7.12}$$

the mod-2 dot product of x and a. In the Bernstein-Vazirani problem, the mystery function f one of the 2^n parity functions on n inputs. The task of the algorithm is to determine a. Observe that the Deutsch problem is. more or less, the one-bit case of this problem: f is constant for $a = 0$ and balanced for $a = 1$.[1] In the following discussion, we will use $x \oplus y$ to indicate the bitwise addition (modulo 2) of two vectors.

7.3.1 The Bernstein-Vazirani Algorithm

Classically, this problem can be solved with n queries. Examine determine each bit of a, one-by-one, by examining

$$\begin{array}{c} f(100\ldots0), \\ f(010\ldots0), \\ \vdots \\ f(000\ldots1) \end{array} \tag{7.13}$$

A quantum algorithm can produce an exact solution to this problem, however, with only one quantum query. The algorithm is the n-bit extension of the single-query algorithm for Deutsch's problem.

First, we require an n-bit version of the Hadamard rotation, for the first and last stages of the modified algorithm. Applying the single-bit Hadamard rotation to each qubit of an n-bit register

[1] For true symmetry with the Deutsch problem, we would need to allow negations of parity functions. The one-bit version of the Bernstein-Vazirani problem would require $f(0) = 0$, and can thus be solved classically in one query (on the value of $f(1)$.

gives a transformation with the right properties. Denote this n-bit transformation $H^{(n)}$. Then

$$\begin{aligned} H^{(n)}|x\rangle &= H|x_0\rangle H|x_1\rangle \ldots H|x_{n-1}\rangle \\ &= \prod_{i=0}^{n-1} H|x_i\rangle \\ &= \prod_{i=0}^{n-1} \frac{1}{\sqrt{2}}(|0\rangle + (-1)^{x_i}|1\rangle) \\ &= \frac{1}{\sqrt{2^n}} \prod_{i=1}^{n} \sum_{y_i \in \{0,1\}} (-1)^{x_i \wedge y_i}|y_i\rangle \\ &= \frac{1}{\sqrt{2^n}} \sum_{y \in \{0,1\}^n} (-1)^{x \cdot y}|y\rangle. \end{aligned} \tag{7.14}$$

Using this definition, we can now give an algorithm to solve the Bernstein-Vazirani problem.

1. Starting from $|0^n\rangle|1\rangle$, apply the n-bit Hadamard rotation $H^{(n)}$ to the first register.
2. Apply a Hadamard rotation to the second qubit.
3. Pass the register and the second qubit to the oracle.
4. Apply a Hadamard rotation to the second qubit.
5. Apply the n-bit Hadamard rotation to the first register.

This algorithm will terminate in the state $|a\rangle|1\rangle$, so that a can be determined with certainty by examining the first register.

Steps (2) through (4) implement an n-bit version of F, which we call $F^{(n)}$. Provided the second qubit begins in state $|1\rangle$ (which it does in this algorithm), $F^{(n)}$ acts on the first register as

$$|x\rangle \xrightarrow{F^{(n)}} (-1)^{x \cdot a}|x\rangle. \tag{7.15}$$

$F^{(n)}$ multiplies by -1 the phase of any term in the superposition for which $f(x) = 1$.

7.3.2 Proof of Correctness

The algorithm begins by applying $H^{(n)}$ to a register containing $|0^n\rangle$. Since $x \cdot y = 0$ for all y, the state of the register becomes

$$|0^n\rangle \xrightarrow{H^{(n)}} \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle.^2 \tag{7.16}$$

$F^{(n)}$ then imposes a relative phase shift on some of the components of the superposition in the first register:

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle \xrightarrow{F^{(n)}} \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{x \cdot a}|x\rangle. \tag{7.17}$$

Finally, a second n-bit Hadamard rotation converts this phase shift information back into bit-flip form:

$$\begin{aligned}\frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}(-1)^{x\cdot a}|x\rangle &\stackrel{H^{(n)}}{\longrightarrow} \frac{1}{\sqrt{2^n}}\sum_{x\in\{0,1\}^n}(-1)^{x\cdot a}\left(\frac{1}{\sqrt{2^n}}\sum_{y\in\{0,1\}^n}(-1)^{x\cdot y}|y\rangle\right)\\ &= \sum_{x\in\{0,1\}^n}\left(\frac{1}{2^n}\sum_{y\in\{0,1\}^n}(-1)^{x\cdot a+x\cdot y}\right)|y\rangle\\ &= \sum_{y\in\{0,1\}^n}\left(\frac{1}{2^n}\sum_{x\in\{0,1\}^n}(-1)^{x\cdot(a\oplus y)}\right)|y\rangle \qquad (7.18)\end{aligned}$$

The coefficient of $|y\rangle$ in this sum can be rewritten in terms of a. Consider first the case $y = a$. Since

$$a \oplus a = 00\ldots0, \qquad (7.19)$$

the phase component of every term inside the sum over x in (7.18) is 1 and x vanishes from inside the summation:

$$\frac{1}{2^n}\sum_{x\in\{0,1\}^n}(-1)^{x\cdot(a\oplus y)} = \frac{1}{2^n}\sum_{x\in\{0,1\}^n}1 = 1 \qquad (7.20)$$

Thus, the component of $|a\rangle$ in the summation over y in (7.18) is 1. Measuring the first register yields $|a\rangle$ with probability 1.

We could stop the analysis at this point, since we automatically know that the amplitude of $|y\rangle$ in $|\phi\rangle$ must be 0 for any $y \neq a$. If it were some nonzero value k, then the probability of observing $|y\rangle$ would be $|k|^2$, and the total probability would fail to sum to 1. It is instructive, however, to carry out the computation for these other y, to see precisely why their coefficients vanish.

Suppose that $y \neq a$. Then y and a must differ in at least one bit. Without loss of generality, suppose that it is the leftmost bit, so that $y_0 \neq a_0$. Then the leftmost bit of $a \oplus y$ will be 1. Consider the values of $x \cdot (a \oplus y)$ for two values of x (let us call them $\tilde{x_0}$ and $\tilde{x_1}$) which differ only in the first digit:

$$\begin{aligned}\tilde{x_1}\cdot(a\oplus y) &= (\tilde{x_0}\oplus 10\ldots0)\cdot(a\oplus y)\\ &= (\tilde{x_0}\cdot(a\oplus y)) + (10\ldots0\cdot(a\oplus y))\\ &= \tilde{x_0}\cdot(a\oplus y)+1. \qquad (7.21)\end{aligned}$$

Of any such pair of x, therefore, one will raise -1 to an even power, and the other to an odd power.

The coefficient of $|y\rangle$ from equation (7.18) can therefore be written as

$$\begin{aligned}
\frac{1}{2^n}\sum_{x\in\{0,1\}^n}(-1)^{x\cdot(a\oplus y)} &= \frac{1}{2^n}\sum_{x_i\in\{0,1\}}(-1)^{xc\dot{(}a\oplus)} \\
&= \frac{1}{2^n}\sum_{x_i\neq 0\in\{0,1\}}\left(\sum_{x_0\in\{0,1\}}(-1)^{x\cdot(a+y)}\right) \\
&= \frac{1}{2^n}\sum_{x_i\neq 0\in\{0,1\}}((1)+(-1)) \\
&= \frac{1}{2^n}\sum_{x_i\neq 0\in\{0,1\}}0 \qquad (7.22)\\
&= 0. \qquad (7.23)
\end{aligned}$$

7.3.3 Commentary

Any difference between y and a means that the contributions to the amplitude of $|y\rangle$ from the various $|x\rangle$ can be arranged into pairs which cancel.

Another way of looking at this problem, inspired by Fourier analysis, is to note that the parity functions $f_a : \{0,1\}^n \to \{0,1\}$ are the Fourier basis functions $\hat{f}_a : \{0,1\}^n \to \{1,-1\}$, by different names[3] The Fourier basis $\{\chi_a\}_{a\in\{0,1\}^n}$ is closely associated with the Fourier basis function in several ways, including that $\hat{f}_a|a\rangle = |\chi_a\rangle$. Simon's algorithm exploits the mutual orthogonality of the functions $\hat{f}_a$.

The algorithm starts by preparing the state $|\chi_0\rangle$. The oracle query implements a group action on the Fourier basis which takes $|\chi_0\rangle$ to $|\chi_a\rangle$. The final Hadamard rotation is a Fourier transform on $\{0,1\}^n$, which sends $|\chi\rangle$ to $|y\rangle$ with coefficient given by $\hat{f}_y(|\chi\rangle)$. If $|\chi\rangle$ is the state corresponding to the parity/Fourier function $\hat{f}_a$, then $f_a(|\chi\rangle) = 1$ but $f_y(|\chi\rangle) = 0$ for any other y.

7.4 Simon's Problem

The above examples do actually not represent a major gain for quantum algorithms. Although the quantum algorithms are better, they are faster at solving problems that were nonetheless classically easy. Bernstein and Vazirani, modifying the above problem into a recursive construction, produced the first problem solvable in polynomial time in a quantum model, but not in a classical one. Simon developed the following simpler problem, which still gives the quantum algorithm an exponential speedup.

We say that a function f is *invariant under XOR-mask* if there is some n-bit value a such that

$$f(x) = f(x \oplus a) \qquad (7.24)$$

[3]The difference between f_a and $\hat{f}_a$ is the difference between $\mathbb{Z}/2$ and the multiplicative group of square roots of unity over the complex numbers: two ways of describing the same group structure, one additive and one multiplicative.

for all x. With respect to the additive group structure on Z_2^n, such functions are periodic with period a. In Simon's problem, the unknown function is some

$$f : \{0,1\}^n \to \{0,1\}^n \tag{7.25}$$

which is invariant under some XOR-mask a. Further, if $f(x) = f(y)$ then $x = y$ or $x \oplus a = y$. The problem is to determine the XOR-mask a. [4]

Classically, any algorithm for this problem, even a probabilistic one, requires time exponential in n. Two distinct queries x and y such that $f(x) = f(y)$ suffice to determine a, but any set of queries which fails to provide such a pair provides no information about a. The optimal strategy for any algorithm is to maximize the number of distinct values of $x \oplus y$ in the set of queries it issues. But k queries can generate at most k such pairs, so that the worst-case number of queries required to deterministically try all 2^n possible a is $\Omega(2^{n-1})$. Probabilistically, the expected time is $\Omega(2^{n-2})$, which is still exponential.

7.4.1 Simon's Algorithm

On the other hand, an expected $O(n)$ iterations of the following algorithm solve Simon's problem:

1. Prepare two registers in the state $|0^n\rangle|0^n\rangle$.
2. Apply $H^{(n)}$ to the first register to obtain the even superposition of all basis states.
3. Call the oracle on the two registers.
4. Measure the second register.
5. Apply $H^{(n)}$ to the first register.
6. Measure the first register.

Let Y be the set

$$\{y : y \cdot a = 0\}. \tag{7.26}$$

This algorithm terminates with $|y\rangle$ in the first register, where y is randomly drawn from Y. Note that $0^n \in Y$, that Y is closed under $\oplus$, and that $|Y| = 2^{n-1}$. Thus Y forms a subspace of $\{0,1\}^n$ and has dimension $n-1$. From any $n-1$ linearly independent elements of Y we will be able to determine both Y and a.

Let us examine the actions of the algorithm more closely to see why it works. Steps (1) through (3) prepare the state

$$\frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle|f(x)\rangle. \tag{7.27}$$

[4] We will neglect the degenerate case $a = 0$ in our analysis. Handling this case requires only a constant number of additional queries; ruling it out removes a corner case from the analysis of the algorithm.

Since f has period a, measuring the second register yields

$$\frac{1}{\sqrt{2}}\left(|x_0\rangle + |x_0 \oplus a\rangle\right)|f(x_0)\rangle \tag{7.28}$$

for some x_0 randomly chosen from X. At this point, $H^{(n)}$ will extract some information about a. Since the second qubit no longer affects the state of the first qubit, we drop it from the analysis.

$$\begin{aligned}\frac{1}{\sqrt{2}}|x_0\rangle + \frac{1}{\sqrt{2}}|x_0 \oplus a\rangle &\xrightarrow{H^{(n)}} \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2^n}}\sum_{y\in\{0,1\}^n}(-1)^{x\cdot y}|y\rangle\right) + \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2^n}}\sum_{y\in\{0,1\}^n}(-1)^{(x\oplus a)\cdot y}|y\rangle\right) \\ &= \frac{1}{\sqrt{2}}\frac{1}{\sqrt{2^n}}\sum_{y\in\{0,1\}^n}(-1)^{x\cdot y}\left(1+(-1)^{a\cdot y}\right)|y\rangle \end{aligned} \tag{7.29}$$

If $a \cdot y = 1$, then the coefficient of $|y\rangle$ vanishes. If $a \cdot y = 0$, the two terms inside the summation add constructively and $|y\rangle$ has coefficient

$$\frac{(-1)^{x\cdot y}}{\sqrt{2^{n-1}}}. \tag{7.30}$$

7.4.2 Features of the Algorithm

The real magic in this algorithm takes place during the second Hadamard rotation. Before this rotation, the first register contains a superposition of two elements, each of which contains no useful information by itself. The information in this state is in the correlation between these elements — they are separated by exactly a. The Hadamard rotation exposes this correlation because it is the Fourier transform for the additive group of n-bit vectors.

Put another way, f is a periodic function, and we wish to determine its period. After sampling f at a great many points in parallel, we apply the Fourier transform to convert the sampled points into a set of Fourier coefficients. Of these coefficients, $\hat{f}_a$ will be large and the rest will vanish. Both the sampling process and the computation of the Fourier transform are efficient for a quantum system, because superposition allows sampling at an exponential number of points and interference allows for rapid computation of the Fourier coefficients.

On another note, although we measured the second register in step (4), the algorithm functions identically if this step is dropped. The second register does not participate in the algorithm after this step, but it is in a superposition of many basis states. These terms of the superposition cannot interfere with each other, as the contents of the second register remain distinct between them. Permanently isolating a piece of a system from the rest of the system is, from the perspective of the system, equivalent to measuring that piece of the system.

7.5 Shor's Algorithm

Simon's algorithm, given a function with period a, determines a. So does Shor's algorithm. The difference between the two is the structure of the underlying group. Simon's algorithm treats

$\{0,1\}^n$ as the group $(\mathbb{Z}/2)^n$, with a group law of exclusive-OR. Shor's algorithm treats $\{0,1\}^n$ as the group $\mathbb{Z}/2^n$, with a group law of binary addition, an attitude we can emphasize by writing $\{0,1\}^n$ as $\{0,1,\ldots,2^n-1\}$.

A function f is *periodic with period* r iff $f(x) = f(y)$ whenever $y = x + r$, and f is not periodic with period smaller than r[5]. Shor's algorithm determines the value of r for periodic functions that are efficiently computable.

7.5.1 The Quantum Fourier Transform

The Hadamard rotation $H^{(n)}$ carries out a Fourier transform on $\{0,1\}^n$. For $\{0,1,\ldots,2^n-1\}$, the appropriate analogue is a quantum version of a traditional discrete Fourier transform, generally referred to as (the) Quantum Fourier Transform (QFT), which acts as

$$|x\rangle \xrightarrow{\text{QFT}} = \frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} e^{2\pi i x y/2^n} |y\rangle \tag{7.31}$$

where $x \in \{0,1,\ldots,2^n-1\}$ (so that $|x\rangle$ belongs to the computational basis). The amplitude of the $|y\rangle$ term in the final state is independent of x and y, but the phase of this term depends on both x and y. We introduce the notation

$$\Phi(t) = e^{\pi i t}, \tag{7.32}$$

so that $\text{QFT}|x\rangle$ can be rewritten as

$$\frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} \Phi\left(\frac{xy}{2^{n-1}}\right) |y\rangle. \tag{7.33}$$

We will also use the identity $\Phi(x)\Phi(y) = \Phi(x+y)$.

The classical discrete Fourier transform can be efficiently computed; so can the quantum discrete Fourier transform. The easiest way to think about the circuit that implements it is as a binary multiplication circuit which computes bitwise products incrementally, starting from the most-significant-bit and working towards the least-significant bit. Classically, an incremental calculation repeatedly left-shifts previous partial products; the QFT algorithm uses successively smaller phase rotations for less significant bits.

nitialize the register The circuit itself is made up from two kinds of gates. First, there are one-bit Hadamard gates. For consistency with Shor's notation, we call R_k the Hadamard gate which acts on qubit k. Second, there are also conditional rotation gates $S_{j,k}$, which multiply the phase by $\Phi(2^{k-j})$ iff both of their inputs (the jth and kth qubit of a register) are $|1\rangle$. Note that all the $S_{j,k}$ commute with each other, as phase multiplications commute.

The algorithm to compute the QFT works as follows:

[5]This condition can also be restated in the equivalent form $f(x) = f(y)$ iff $y = x + kr$ for some integer k such that $x, y \in \{0,1,2,\ldots,2^n-1\}$.

1. Load into the register the state to which the QFT will be applied, according to the convention that if the register is in the state

$$|x_0\rangle|x_1\rangle\ldots|x_{n-1}\rangle, \tag{7.34}$$

then x_0 is the *most* significant digit of x and x_{n-1} is the *least* significant digit.

2. For each k from 1 to n, carry out the following steps
   - Apply the gates $S_{j,k}$ for all $j < k$.
   - Apply the gate R_k.

The register will now be in a superposition of a number of terms, each of the form

$$|y_0\rangle|y_1\rangle\ldots|y_{n-1}\rangle. \tag{7.35}$$

These terms are the *bit-reversals* of the answer: in each, y_0 is the *least* significant digit of y and y_{n-1} is the *most* significant digit.

To be precise about this bit-reversal. we use X_k and Y_k as shorthand for the binary numbers encoded by the k leftmost qubits in any term in a superposition. Formally

$$X_k = 2^{k-1}x_0 + \cdots + 2x_{k-2}x_{k-1} = \sum_{i=0}^{k-1} 2^{k-1-i}x_i \tag{7.36}$$

$$Y_k = 2^{k-1}y_{k-1} + \cdots + 2y_1 + y_0 = \sum_{i=0}^{k-1} 2^i y_i \tag{7.37}$$

X_{k+1} and Y_{k+1} can be expressed incrementally as

$$X_{k+1} = 2X_k + x_k \quad Y_{k+1} = 2^k y_k + Y_k. \tag{7.38}$$

We now examine in detail the action of the transform on $|x\rangle$ for an arbitrary $x \in \{0, 1, \ldots, 2^n - 1\}$. Each stage of the algorithm extends the transform to include one more qubit, so that after k stages the quantum Fourier transform has been calculated for the leftmost k qubits and the results stored in those qubits in bit-reversed form. Formally, we assume that the first k stages have produced the state

$$\frac{1}{\sqrt{2^k}} \sum_{y_0\ldots y_{k-1}\in\{0,1\}^k} \Phi\left(\frac{X_k Y_k}{2^{k-1}}\right)|y_0\rangle\ldots|y_{k-1}\rangle|x_k\rangle\ldots|x_{n-1}\rangle. \tag{7.39}$$

(For $k = 0$, this state is $|x_0\rangle|x_1\rangle\ldots|x_{n-1}\rangle$) At the end of the $k+1$th stage, the system has the state

$$\frac{1}{\sqrt{2^{k+1}}} \sum_{y_0\ldots y_k\in\{0,1\}^{k+1}} \Phi\left(\frac{X_{k+1} Y_{k+1}}{2^k}\right)|y_0\rangle\ldots|y_k\rangle|x_{k+1}\rangle\ldots x_{n-1} \tag{7.40}$$

For $n = k$, this final state is (7.33).

To see that the gates $S_{j,k}$ followed by R_k correctly implement one stage of the quantum Fourier transform, consider first the effect of the $S_{j,k}$ gates on the state (7.39). The kth qubit has state

$|x_k\rangle$. In each term of the sum, for each $j < k$, the jth qubit from the left is $|y_j\rangle$. qubit from the left is $|y_j\rangle$. If both are 1 (that is, if $y_j x_k = 1$), then the phase of that term is multiplied by $\Phi(2^{j-k} = \Phi(2^j/2^k)$. Taken across all possible values of j, the phase is multiplied by

$$\Phi\left(\sum_{j=0}^{k-1} 2^{j-k} x_k\right) = \Phi\left(\frac{x_k}{2^k}\sum_{j=0}^{k-1} 2^j y_j\right) = \Phi\left(\frac{1}{2^k} x_k Y_k\right), \tag{7.41}$$

taking the overall state from (7.39) to

$$\frac{1}{\sqrt{2^k}} \sum_{y_0 \ldots y_{k-1} \in \{0,1\}^k} \Phi\left(\frac{2X_k Y_k + x_k Y_k}{2^k}\right) |y_0\rangle \ldots |y_{k-1}\rangle |x_k\rangle \ldots |x_0\rangle. \tag{7.42}$$

Next, the gate R_k takes each term in this superposition to two terms, one each for $|y_k\rangle = |0\rangle$ and $|y_k\rangle = |1\rangle$. In each of these terms, the amplitude is multiplied by $\frac{1}{\sqrt{2}}$; the phase of a term is multiplied by -1 if and only if both $|x_k\rangle$ and $|y_k\rangle$ are $|1\rangle$. Put another way, the phase is multiplied by $\Phi(x_k y_k)$. Thus, the gate R_k takes (7.42) to the state

$$\frac{1}{\sqrt{2^{k+1}}} \sum_{y_0 \ldots y_k \in \{0,1\}^{k+1}} \Phi\left(\frac{2X_k Y_k + x_k Y_k + 2^k x_k y_k}{2^k}\right) |y_0\rangle \ldots |y_k\rangle |x_{k+1}\rangle \ldots |x_{n-1}\rangle. \tag{7.43}$$

This state can be recognized as (7.33) with the phase factored. More precisely

$$\begin{aligned}
\Phi\left(\frac{X_{k+1} Y_{k+1}}{2^k}\right) &= \Phi\left(\frac{(2X_k + x_k)(2^k y_k + Y_k)}{2^k}\right) \\
&= \Phi\left(\frac{2X_k Y_k + x_k Y_k + 2^k x_k y_k}{2^k}\right) \Phi\left(\frac{2^{k+1} X_k y_k}{2^k}\right) \\
&= \Phi\left(\frac{2X_k Y_k + x_k Y_k + 2^k x_k y_k}{2^k}\right),
\end{aligned} \tag{7.44}$$

because the right-hand term must always be a multiple of 2, and therefore causes a rotation that is a multiple of 2π, which vanishes.

7.5.2 The Period-Finding Algorithm

Structurally, Shor's algorithm follows Simon's algorithm very closely.

1. Prepare two registers in the state $|0^n\rangle|0^n\rangle$.
2. Appply $H^{(n)}$ to the first register to obtain the even superposition of all basis states.
3. Call the oracle on the two registers.
4. Measure the second register.
5. Apply the quantum Fourier transform to the first register.
6. Measure the first register.

Only step (5) differs from the corresponding step in Simon's algorithm. A different kind of Fourier transform extracts different information about the periodicity of f.

Shor's algorithm, as does Simon's, begins by preparing the state

$$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle. \tag{7.45}$$

The algorithm then applies the oracle for f, producing the state

$$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle|f(x)\rangle. \tag{7.46}$$

Shor's algorithm now measures the second register (as in Simon's algorithm, this step is not strictly necessary, but makes the analysis simpler to follow), and observes some value; particular $f(x_0)$. The state of the first register after this measurement is slightly more complicated: it contains the equal superposition of all x such that $f(x) = f(x_0)$. Let us call the number of such values A. Without loss of generality, the number we name x_0 is the smallest such value, so that the others are given by $x_0 + r, x_0 + 2r, \ldots, x_0 + (A-1)r$. The state of the overall system after this measurement is therefore

$$\frac{1}{\sqrt{A}} \sum_{j=0}^{A-1} |x_0 + jr\rangle. \tag{7.47}$$

At this point, a quantum Fourier transform is used to extract information about this state.

$$\begin{aligned} \mathrm{QFT}\left(\frac{1}{\sqrt{A}} \sum_{j=0}^{A-1} |x_0 + jr\rangle\right) &= \frac{1}{\sqrt{A}} \sum_{j=0}^{A-1} \frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} \Phi\left(\frac{(x_0 + jr)y}{2^{n-1}}\right) |y\rangle \\ &= \frac{1}{\sqrt{2^n A}} \sum_{y=0}^{2^n-1} \left(\Phi\left(\frac{x_0 y}{2^{n-1}}\right) \sum_{j=0}^{A-1} \Phi\left(\frac{jry}{2^n}\right)\right) |y\rangle \end{aligned} \tag{7.48}$$

Finally, the first register is observed. The probability of seeing $|y\rangle$ is given by the square of the absolute value of the amplitude of $|y\rangle$, that is

$$\begin{aligned} \left|\frac{1}{\sqrt{2^n A}} \Phi\left(\frac{x_0 y}{2^{n-1}}\right) \sum_{j=0}^{A-1} \Phi\left(\frac{jry}{2^{n-1}}\right)\right|^2 &= \left|\frac{1}{\sqrt{2^n A}}\right|^2 \cdot \left|\Phi\left(\frac{x_0 y}{2^{n-1}}\right)\right|^2 \cdot \left|\sum_{j=0}^{A-1} \Phi\left(\frac{jry}{2^{n-1}}\right)\right|^2 \\ &= \frac{1}{2^n A} \cdot 1 \cdot \left|\sum_{j=0}^{A-1} \Phi\left(\frac{jry}{2^{n-1}}\right)\right|^2 \\ &= \frac{A}{N} \left|\frac{1}{A} \sum_{j=0}^{A-1} \Phi\left(\frac{jry}{2^{n-1}}\right)\right|^2 \end{aligned} \tag{7.49}$$

This value will be large when the terms inside the sum interfere constructively. Constructive interference will occur when each addition of another $2\pi i \frac{yr}{2^n}$ does not much affect the phase, i.e.

when yr is close to a multiple of 2^n. A good approximation to y allows a reasonable-probability possibility of reconstructing r (by taking greatest common denominators).

Shor's algorithm can fail because r is almost certain not to divide 2^n, so that even the correct solution is not perfectly measured by the QFT, and also because values close to r will also display constructive interference in the final distribution. Nonetheless, it can be shown that with arbitrarily high probability, $O(\log \log r)$ repetitions suffice to find r. The probability of success on any individual trial actually is at least $\phi(r)/3r$.

7.5.3 Factoring As Period-Finding

One standard periodic function is exponentiation modulo a fixed value: given some fixed a and n that are relatively prime to each other, $f(x) = a^x \pmod n$[6]. The goal of an algorithm is to determine the *order* of a modulo n, the smallest nonzero x for which $a^x \equiv 1(\mathrm{mod} n)$.

Order-finding is useful because there is a randomized reduction from factorization to order-finding. Given an efficient way of computing the order of arbitrary elements $x(\bmod n)$, it is possible to factor n efficiently with high probability of success.

Choose a random $1 < x < n$ and compute its order r. Thus

$$x^r - 1 \equiv 0(\mathrm{mod} n), \tag{7.50}$$

can be factored as

$$(x^{\frac{r}{2}} - 1)(x^{\frac{r}{2}} + 1) \equiv 0(\mathrm{mod} n), \tag{7.51}$$

provided that r is even. Because r is the order of x, it is not the case that $x^{\frac{r}{2}} - 1 \equiv 0(\mathrm{mod} n)$. Thus the first factor must be nontrivial. So long as $(x^{\frac{r}{2}} + 1) \not\equiv 0(\mathrm{mod} n)$, the second factor is also nontrivial, so that if we take these two factors and compute their greatest common divisors with n, we will have two nontrivial factors of n. It can be shown that this procedure fails with probability at most 2^{-k+1}, where k is the number of distinct prime factors of n. For composite numbers, $k \geq 2$, so that this procedure succeeds with probability at least $\frac{1}{2}$. Shor's algorithm is therefore a polynomial-time probabilistic factoring algorithm.

7.6 Grover's Algorithm

Grover's algorithm extends some of the ideas inherent in Simon's algorithm in very different directions than Shor's algorithm does. The algorithm finds an element in an unsorted database, and also allows a number of related algorithms related to finding one distinguished element from a collection: determining the minimum or the median of such a database, for example.

Formally, Grover's problem provides an oracle O_{x_0} that evaluates a function

$$f : \{0,1\}^n \to \{0,1\} \tag{7.52}$$

[6]Shor's algorithm can be generalized to find the order of elements over more complicated Abelian groups, not just the multiplicative group modulo n.

such that $f(x_0) = 1$ for some single $x_0 \in \{0,1\}^n$, and $f(x) = 0$ for all $x \neq x_0$. The task of an algorithm is to determine the value of x_0. Note that, unlike previous problems, there are a great many "interesting" computational problems which can be reduced to this one. For a problem of database search, for example, the oracle, on input x would query the database in place x, evaluate the predicate "does x contain the requested value?", and return 1 if and only if that predicate was satisfied. Of course, the database need not be a black box, either. One could, for example, place a circuit within the database to directly compute a function. The same exponential speedup as seen for Simon's problem would allow a quantum device to solve any problem in NPin polynomial time. This possibility is not the case — in fact, Grover's quadratic speedup is provably close the best possible in general — but nevertheless, it is a much more broadly applicable algorithm than others we have been considering.

7.6.1 The Grover Iteration

The oracle transformation O_{x_0} is half the algorithm. In Grover's algorithm, O_{x_0} is always used with the output qubit in state $H|1\rangle$, so that O_{x_0} acts by phase inversion.

The other half of the algorithm is transformation, which we will call R (for "reflection," as will become clear shortly). It makes the algorithm simpler to pretend that the algorithm has access to a second oracle O_0, one which computes a known function: $f(x) = 1$ if $x = 0$ and $f(x) = 0$ otherwise. This oracle is just an n-bit controlled-NOT, but regarding it as a second oracle makes clear a formal symmetry in the structure of Grover's algorithm. R is carried out in the following steps:

1. Apply the n-bit Hadamard rotation $H^{(n)}$ to the n input qubits.
2. Feed the n input qubits and the output qubit to O_0.
3. Multiply the phase of the system by -1[7]
4. Apply $H^{(n)}$ to the n input qubits again.

We call the transformation $RO_{x_0} = H^{(n)}O_0H^{(n)}O_{x_0}$ one *iteration*, or I. Grover's algorithm for searching an unsorted database consists in applying I^k (for some k we will derive in a moment) to the usual even superposition of all basis states, and then measuring the system. With good probability, the result of this measurement will be $|x_0\rangle$.

Conceptually, under the right conditions, one iteration I will increase the amplitude of $|x_0\rangle$ while decreasing the amplitude of every other term.

The iteration starts with some distribution of amplitudes $a_1, \ldots a_{2^n-1}$ across the basis states in the computational basis. The Hadamard rotation shifts this distribution into its Fourier representation. Steps 2 and 3 leave untouched the coefficient of $|0^n\rangle$ while inverting all other coefficients. A second Hadamard rotation then translates these changes back into the computational basis.

[7]This step is not necessary, since it clearly does not affect any observable quantity of the system in any way. This inversion is included for clarity of exegesis, as it makes a superfluous minus sign disappear

Because H takes $|0^n\rangle$ state equally to (or from, since $H^2 = I$) all states in the computational basis, the coefficient of $|0^n\rangle$ in the Fourier basis is the average of the a_i. Every other non-zero component in the Fourier basis specifies that certain a_i differ from this average value. value. Flipping every coefficient other than that on $|0^n\rangle$, therefore, inverts all information about the a_i except their average.

Thus, after the rotation back into the computational basis, all of the a_i have been inverted about their average value, which we notate $\bar{a}$. If we write $a_i = \bar{a} + (a_i - \bar{a})$, this transformation sends

$$a_i \quad \rightarrow \quad \bar{a} - (a_i - \bar{a}) \quad = \quad 2\bar{a} - a_i. \tag{7.53}$$

The transformation O_{x_0}, on the other hand, inverts the value of g at x_0, and leaves all other values of g untouched. Consider, then, the effects of one full iteration on the initial state $H^{(n)}|0^n\rangle$, the even superposition of all basis in terms of this function g. The algorithm begins with

$$a_i = \frac{1}{\sqrt{2^n}} \tag{7.54}$$

for all i. After O_{x_0} is applied, $a_{x_0} = \frac{1}{\sqrt{2^n}}$ and all other a_i are unchanged. The average of these 2^n values of a_i is

$$\frac{-\frac{1}{\sqrt{2^n}} + \sum_{x \neq x_0} \frac{1}{\sqrt{2^n}}}{2^n} \quad = \quad \left(1 + \frac{1}{2^{n-1}}\right) \frac{1}{\sqrt{2^n}} \tag{7.55}$$

Thus, by applying the identity above, the value of a_{x_0} becomes

$$2 \cdot \left(1 - \frac{1}{2^{n-1}}\right) \frac{1}{\sqrt{2^n}} - \frac{(-1)}{\sqrt{2^n}} \quad = \quad \frac{1}{\sqrt{2^n}} \left(3 - \frac{1}{2^{n-2}}\right) \tag{7.56}$$

while the value of each other a_i becomes

$$2 \cdot \left(1 - \frac{1}{2^{n-1}}\right) \frac{1}{\sqrt{2^n}} + \frac{(-1)}{\sqrt{2^n}} \quad = \quad \frac{1}{\sqrt{2^n}} \left(1 - \frac{1}{2^{n-1}}\right) \tag{7.57}$$

The amplitude of the selected state has jumped significantly, while the amplitude of each of the unselected states has declined slightly. In general, if the marked state begins the iteration with amplitude $0 < a_{x_0} < \frac{1}{\sqrt{2}}$ and the other states have the appropriate amplitude $l > 0$, then the change in a_{x_0} as a result of applying one iteration of Grover's algorithm is bounded below by $\frac{1}{\sqrt{2^n+2}}$ (and l becomes smaller, but still > 0).

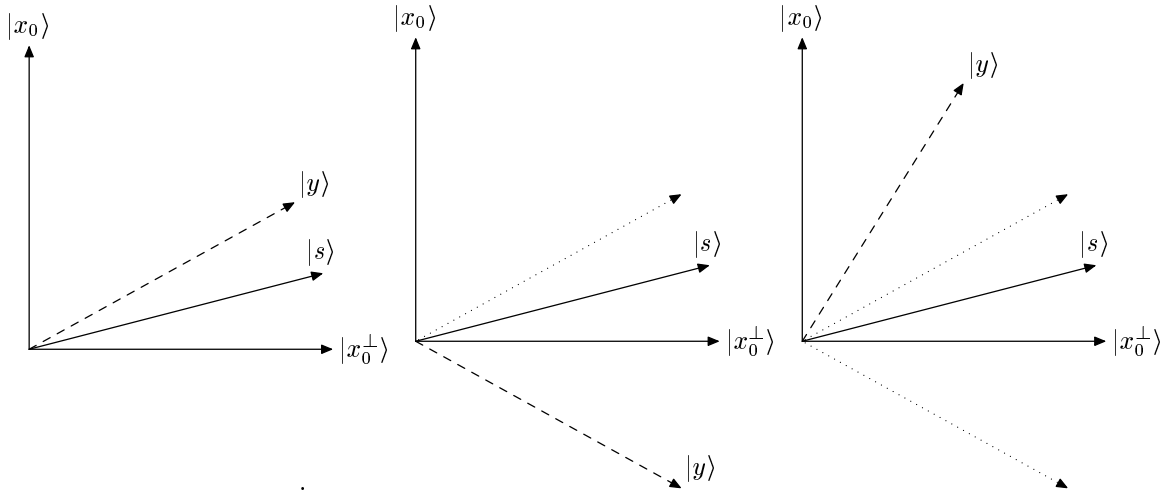
7.6.2 The Full Algorithm

Thus, repeated applications of the Grover iteration will eventually result in some iteration after which $k > \frac{1}{\sqrt{2}}$. By the bound above, this takes place after at most $\frac{\sqrt{2^n+2}}{\sqrt{2}} = \sqrt{2^n + 1}$ repetitions. Since $a_{x_0} > \frac{1}{\sqrt{2}}$, a measurement of the system at this point, will produce the value $|x_0\rangle$ with probability at least $\frac{1}{2}$. One additional query suffices to determine whether the $|y\rangle$ produced by the algorithm really is such that $f(y) = 1$ (and thus $y = x_0$). If not, the overall process can be repeated

as necessary. To achieve probability of success greater than $1 - 2^{-r}$ requires at most r repetitions of the overall algorithm, for total time (and queries) $O(r\sqrt{2^n})$.

No classical algorithm can do as well. There is no strategy better than to guess at x_0, feeding each guess to the oracle. It is easy to show that this algorithm succeeds in expected time 2^{n-1}, or with probability greater than $1 - 2^{-r}$ in time $(1 - 2^{-r})(2^{n-1})$. It is also easy to show that this strategy is optimal. The quantum algorithm thus achieves a quadratic speedup over the classical algorithm for this problem.

It is possible to be more precise about the number of iterations required to maximize the probability of finding $|x_0\rangle$. Let us consider the actions of an iteration on vectors lying in the plane spanned by $|x_0\rangle$ and $|s\rangle$, the initial state. Suppose the system is currently in the state $|y\rangle$. The oracle stage flips the $|x_0\rangle$ component of $|y\rangle$, i.e. flips $|y\rangle$ about the axis perpendicular to $|x_0\rangle$, which we call $|x_0^\perp\rangle$ The rotation stage preserves only the $|s\rangle$ component of $|y\rangle$, i.e. flips $|y\rangle$ about the $|s\rangle$ axis. Taken together, then, since the overall parity of the two transformations is non-inverting, they must define a rotation. With what angle does this rotation act? As the following diagrams illustrate, an arbitrary vector is rotated by twice the angle between $|x_0^\perp\rangle$ and $|s\rangle$:



(7.58)

Let us call this angle θ. Since $|x_0\rangle$ and $|x_0^\perp\rangle$ are perpendicular,

$$\sin\theta = \sin\angle(|x_0^\perp\rangle, |s\rangle) = \cos\angle(|x_0\rangle, |s\rangle) = \frac{(|x_0\rangle)\cdot(|s\rangle)}{||x_0\rangle||s\rangle|} = \frac{1}{\sqrt{2^n}}. \tag{7.59}$$

In order to rotate a state which begins as $|s\rangle$ (i.e., rotated by θ from $|x_0^\perp\rangle$) so that it becomes $|x_0\rangle$, the single-iteration rotation needs to repeated until the overall rotation equals $\frac{\pi}{4}$ radians. A quick numerical argument shows that approximately

$$\frac{\pi}{4}\sqrt{2^n} \tag{7.60}$$

rotations are required. Continued applications of the Grover iteration will continue to rotate $|y\rangle$ by 2θ each time. After $\frac{\pi}{2}\sqrt{2^n}$ iterations $|y\rangle$ will be very close to $-|s\rangle$. In general, for any given n, there is an optimal number of rotations to perform, after which further rotations will reduce the amplitude of the desired state.

7.6.3 Extensions to Grover's Algorithm

It could be the case that there are multiple solutions in the database. For the case where there are known to be exactly r solutions to $f(x) = 1$, Grover's algorithm can be modified to work by changing the number of iterations required. Let $|X\rangle$ be the even superposition of all r solutions to $f(x) = 1$. It is easy to check that

$$|X\rangle = \frac{1}{\sqrt{r}} \left(\sum_{i=1}^{r} |x_i\rangle \right), \tag{7.61}$$

so that

$$(|X\rangle) \cdot (|s\rangle) = \sqrt{\frac{r}{2^n}}. \tag{7.62}$$

As long as r remains relatively small compared with n, we can use the same small-sine approximation to conclude that close to

$$\frac{\pi}{4}\sqrt{\frac{2^n}{r}} \tag{7.63}$$

iterations are necessary. As the number of solutions increases, the quantum algorithm continues its quadratic speedup over the classical one, which requires $O(\frac{2^n}{r})$ queries.

The algorithm can even be modified to work for an *unknown* number of solutions. In this case, a random number of iterations from 0 to $\frac{pi}{4}\sqrt{2^n}$ are carried out, followed by a measurement. This procedure boosts boosts the expected probability of success — for ever possible r — to about $\frac{1}{2}$.

In another direction, the algorithm can also be optimized to work faster when information about the set of possible solutions is available. From the perspective of rotations, the virtue of a reflection about $|s\rangle$ is that, for every possible $|x_0\rangle$, $|s\rangle$ is not orthogonal to $|x_0\rangle$, so that the two reflections combined are nontrivial. Further, $\angle(|s\rangle, |x_0\rangle)$ is the same for every x_0. Thus, if it is known that only some subset $X \subseteq \{0,1\}^n$ of possible x could be solutions, the algorithm can be tweaked to favor those X. There are two such optimizations available.

First, the algorithm can start with a vector that is the evenly weighted superposition of the elements of X, rather than of every element of $\{0,1\}^n$ (assuming that X can be computed efficiently). This change gives the desired states larger initial amplitude, so that fewer iterations are required to amplify them. Second, the reflection should be about a vector which has a large (and ideally, equal) angle with each of the $|x^\perp\rangle, x \in X$)at the cost of having smaller angle with the $|x^\perp\rangle, x \notin X$). This can be done by using the function that supplies the various values of $x \in X$ to prepare a vector $|X\rangle$ which is close to the evenly weighted superposition of all the X. In place of the Hadamard rotation, a rotation to a basis in which $|X\rangle$ is one of the elements is used. Then, the inversion-about-average instead becomes an inversion about $|X\rangle$. Again, the speedup over a similarly "structured" classical search problem is quadratic.

This general idea of iterated amplification has also been used to develop algorithms to find the median of an unsorted set of 2^n items, again with quadratic speedup.

Quadratic speedup for an exponential problem is not a fully encouraging result. From a computational complexity point of view, Grover's algorithm does not cross any major tractability

boundaries. It makes difficult problems easier, but not sufficiently easier to make them easy. Pragmatically, given the head start that conventional computers have on quantum ones, it would be surprising if quantum computers were to "catch up" sufficiently with classical ones to make their use practical for Grover-type problems, even with this quadratic advantage.

What is more, Christof Zalka has proven that Grover's algorithm is asymptotically optimal. No algorithm requires fewer queries than Grover's for the general case. This result effectively gives us a limit on the performance gains of quantum computers for unstructured problems. To do better than a quadratic speedup, a quantum computer must exploit some sort of structure in the problem, structure that is classically difficult to locate. This is the approach taken by Simon's algorithm.

Chapter 8

Conclusion

8.1 Other Directions

8.1.1 Quantum Information Theory

Quantum information theory is a rich field, and one that is the focus of much current research. The question of how information can be stored in and extracted from quantum states has been subjected to several quite fruitful formalizations. Quantum information theory deals with the degree to which states display entanglement, the amount of information that can be extracted by making measurements on portions of entangled systems, and the relative information content of different sets of measurements, among other issues. Quantum information theory also allows a rigorous proof of the *No Cloning Theorem*, that it is impossible to make a copy of the state of an arbitrary quantum system which is not entangled with the system being copied.

One exciting area of research in quantum information theory is quantum communication complexity. Because entanglement is non-local, quantum communication has properties that can be strikingly different from classical communication. Many of the algorithms we have considered can be rephrased as communication problems: considering them as problems in which the goal is to minimize the number of qubits exchanged between two parties reveals interesting features of these algorithms. There are also a number of results on quantum measures of communication complexity for various functions, including a number of lower-bound results.

8.1.2 Quantum Cryptography

The distinctive features of quantum computation also have interesting implications for cryptographic schemes. If Alice has sent Bob a qubit entangled with a qubit she retains, it is possible for her to change some properties of measurements made on Bob's qubit at any time until he measures it, even after she has physically "given" it to him. Manipulations of entanglement make some cryptographic applications simpler: it is possible to create quantum channels which are provably safe from eavesdropping, and to use "quantum teleportation" to transmit quantum states more accurately than classical channels allow. On the other hand, other cryptographic challenges become

much harder: it is possible for dishonest parties to cheat in ways that would have been classically impossible. For example, quantum bit commitment is provably impossible.

Quantum cryptography generally involves less interaction between qubits than quantum computation, and the requirements on long-term storage are generally much looser. Usually, what matters is getting a qubit from one place to another intact. For such reasons, quantum cryptography has enjoyed more experimental success to date than quantum computation. Quantum channels have been created which transmit information over distances of many meters, and their theoretically useful properties have been experimentally confirmed.

8.1.3 Quantum Error Correction

It is generally agreed that quantum hardware is unlikely to improve in precision and stability by the many orders of magnitude necessary to make reliable large-scale quantum computation directly feasible. For this reason, a great deal of research attention is being directed to making quantum computation fault-tolerant at the "software" or "circuit" level: creating schemes which can detect and correct errors in stored qubits as they occur.

Quantum error correction is much harder than classical error correction for a number of reasons. First, the transconductance nonlinearities of semiconductors, classically used to regenerate degraded signals and restore voltages to their nominal levels, are not applicable to quantum systems. Unitary time-evolution means that it is not possible to "compress" an "error" region inside state space into a smaller "nominal" region. Second, the nature of entanglement is such that errors can propagate to syndrome qubits used to detect errors. Error-correction schemes must be designed to prevent errors from being inadvertently spread by the error-correction process itself. Most importantly, it is not entirely clear what the right error models for quantum systems are. Conflicting error models differ with respect to what kinds of errors must be prevented (phase shifts, amplitude errors, bit flips, etc.) and how these errors are correlated (independent errors, concentrated in specific qubits, uniform errors in all qubits, etc.). Different error-correction schemes defend against different kinds of errors.

Nonetheless, a rich literature devoted to quantum error correction is developing. Some schemes draw upon classical coding theory to create quantum codes which are resistant to bit flips and can correct against detected errors. Other techniques exploit the "quantum watchdog" effect and measure values that should be known with certainty, thus (with very high probability) collapsing away the small component of the state which is in error. Still other techniques involve making multiple copies of a quantum computation and repeatedly projecting these copies onto a small subspace which they share, so that errors (differences) are eliminated while the computation (their common component) is preserved.

8.1.4 Continuous Quantum Computation

We have focused, as has the vast majority of research into quantum computation, on quantum systems which encode discrete-valued information. Some proposals, however, see as quantum computation's greatest strength as ability to work directly with continuously-valued systems. Feynman's original motivation for quantum computation was precisely such an application: modeling quantum systems — a computationally difficult task classically, but one of great interest to physicists.

Since then, a number of authors have developed more specific algorithmic methods for simulating quantum systems on other quantum systems.

8.1.5 Quantum Language Theory

For almost any formal class of objects in the theory of computation, it is possible to define a quantum version of that class. We have discussed some of these new classes (such as quantum Turing machines and quantum circuits), but by no means all. There remains a great deal of work to be done in mapping out such classes and determining their relationship to conventional classes. Quantum finite-state automata, in particular, are more powerful than traditional finite-state automata in some respects, but less powerful in others. The right mathematical formalization of quantum pushdown automata and quantum grammars has thus far remained elusive, but attempts at producing good descriptions of their properties have suggested that interesting results may lie in store once the right formalizations have been worked out.

8.2 The Future of Quantum Computation

In theory, quantum computers are better at certain operations than classical ones. In practice, it is difficult to predict whether quantum computers large enough to carry out those operations on a non-trivial scale will ever be feasible. Present technology is inadequate to the task, but there do not appear to be any intrinsic physical restrictions which would block the development of quantum computers. The best that can be safely said is that within ten or fifteen years it will probably be known whether quantum computers will ever be possible.

On the other end of the spectrum, the applications for quantum computers remain uncertain. Grover's algorithm, although suggestive, does not correspond well to any major open problems. Indexing technology is too well-developed for a quadratic speedup of unordered search to be useful. Shor's algorithm has more immediate applications, but period-finding techniques are only applicable within certain limited domains. Quantum computation has yet to turn up a reasonably-sized class of general-purpose algorithms useful in multiple circumstances. Until it does so, the motivation for building quantum computers remains more one of curiosity (and codebreaking) than of practical utility.

In between these extremes, however, the picture is far less murky. Suppose, for the sake of imagination, that tomorrow a physics research laboratory were to announce the successful creation of a ten-thousand-qubit quantum device capable of carrying out tens of millions of elementary operations before the onset of decoherence. Such a computer could carry out Shor's algorithm, factoring in seconds numbers that would take years to factor on classical computers. Algorithm designers and complexity theorists could carry out experiments on quantum search heuristics and average-case complexity of quantum problems. If one of them were to discover a new highly-efficient quantum algorithm for a ubiquitous problem in computer science, it could be implemented on this hypothetical computer with very little difficulty. Such a computer could be used "out of the box," as it were, *even though many of its underlying principles differ radically from those of any computer now in existence.*

The problems associated with making quantum computation possible. at every level between hardware and algorithms, are not trivial problems. They are interesting problems, but they are also solved problems. In the last ten years, in the near-complete absence of actual quantum computers, the computer science and physics communities have formulated and resolved the major issues of quantum computation.

The problems have come from many areas of computer science (and many related disciplines), but the solutions have come from even more. Algorithm design has supplied applications for quantum computers; complexity theory has commented on and refined those algorithms. Parallel and scientific computation have supplied techniques for conceptualizing and simulating quantum computers. Physics has provided the descriptions of reality on which quantum computation builds, and suggested ideas for the construction of quantum computers. Linear algebra and automata theory have developed abstract models of quantum computation. Computer architecture has explained how to connect basic operations into complicated higher-level operations. Reversible computing has strayed from its low-power computation origins to explain key characteristics of quantum computers. Formal logic and programming language design have provided tools for describing and reasoning about quantum systems and for expressing quantum computations abstractly. Coding theory has shown how to improve the stability of quantum computations, even in the presence of unreliable hardware. Game theory, learning theory, language theory, databases, and artificial intelligence have all begun to explore their connections to quantum computation.

Quantum computation is a microcosm of computer science. Many of the defining problems of traditional computation reappear in quantum contexts in subtle yet new forms. It is a measure of how well developed "traditional" computer science is that quantum computation has been able to draw upon so many distinct techniques and receive so many answers. It is a measure of how much remains to be done in "traditional" computer science that quantum computation has been able to open so many doors and pose so many questions.

Bibliography

General

The canonical repository for research in quantum computation is the `quant-ph` electronic preprint archive at:

> `http://xxx.lanl.gov`

John Preskill has taught a course at Caltech on quantum computation from the perspective of physics. His (unpublished) lecture notes from that course are the best available "textbook" for the field. At present, these notes are available at:

> `http://www.theory.caltech.edu/people/preskill/ph229/`

The philosophical issues raised by quantum mechanics are quite complicated. In particular it is very difficult to give an account of measurement that is both physically accurate and philosophically reasonable. For an accessible and highly readable overview of these issues and the of principal competing theories, see

> D. Albert, *Quantum Mechanics and Experience*, Cambridge: Harvard University Press, 1992.

Hardware

A paper that discusses the use of NMR techniques to create quantum gates (although assuming some knowledge of NMR in the reader) is

> J. Jones et al., "Quantum Logic Gates and Nuclear Magnetic Resonance Pulse Sequences," *Journal of Magnetic Resonance, 135,* pp. 353-360, 1998.
> `quant-ph/9805070`

The polymer model of scalable NMR quantum computation is proposed in

S. Lloyd, "A Potentially Realizable Quantum Computer," *Science 261,* pp. 1569-1571, 1993.

A successful experimental realization of an NMR quantum computer carrying out out a two-bit version of Grover's search algorithm is described in

I. Chuang et al., "Experimental Implementation of Fast Quantum Searching" *Physical Review Letters 80,* 15, pp. 3408-3411, 1998.

A successful experimental realization of an NMR quantum computer carrying out Deutsch's algorithm is described in

J. Jones and M. Mosca, "Implementation of a Quantum Algorithm to Solve Deutsch's Problem on a Nuclear Magnetic Resonance Quantum Computer," *Journal of Chemical Physics 109,* pp 1648-1653, 1998.
`quant-ph/9801027`

A slightly technical but highly readable review article on ion trap quantum computation is

A. Steane, "The Ion Trap Quantum Information Processor," *Applied Physics B 64,* 6, pp. 623-643, 1997.
`quant-ph/9608011`

Quantum computation with photons is proposed in

I. Chuang and Y. Yamamoto, "A Simple Quantum Computer."
`quant-ph/9505011`

The use of a single photon to store multiple qubits is proposed in

N. Cerf et al., "Optical Simulation of Quantum Logic," *Physical Review A 57,* 3, pp R1477-R1480, 1998.
`quant-ph/9706022`

The literature on quantum dots is notoriously dense. One of the more accessible articles on the subject is

D. Loss and D. DiVincenzo, "Quantum Computation with Quantum Dots," *Physical Review A 57,* 1, pp. 120-126, 1998.
`cond-mat/9701055`

A non-technical introduction to quantum computation with anyons is contained in

J. Preskill, "Fault-Tolerant Quantum Computation," Caltech/USC/MIT Institute for Quantum Information and Computation Technical Report QUIC-97-034, 1997.
`quant-ph/9712048`

A discussion of some of the issues involved in simulating quantum computers with classical ones, along with simulation results on the effects of decoherence and experimental inaccuracy on quantum computations, can be found in

K. Obenland and A. Despain, "Models to Reduce the Complexity of Simulating a Quantum Computer."
`quant-ph/9712004`

Theory

The ability of quantum circuits to simulate quantum Turing machines is shown in

A. Yao, "Quantum circuit complexity," *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science*, pp. 352-361, 1993.

The definitive paper on quantum Turing machines is

E. Bernstein and U. Vazirani, "Quantum Complexity Theory," *SIAM Journal on Computing, 26,* 5, pp. 1411-1473, 1997.

Deutsch's three-bit universal gate is described in

D. Deutsch, "Quantum Computational Networks," *Proceedings of the Royal Society of London A, 425*, p. 73, 1989.

The first construction of a universal two-bit gate is given in

A. Barenco "A Universal Two–Bit Gate for Quantum Computation."
`quant-ph/9505016`

A full proof that almost all two-bit gates are universal is contained in

D. Deutsch et al. "Universality in Quantum Computation."
`quant-ph/9505018`

A comprehensive paper on the construction of larger quantum gates from smaller ones is

A. Barenco et al. "Elementary Gates for Quantum Computation."
`quant-ph/9503016`

Circuits

A general characterization of the use of quantum circuits as subroutines, along with several useful equivalences, is proven in

> D. Aharonov et al., "Quantum Circuits with Mixed States," *Proceedings of the 30th Annual ACM Symposium on Theory of Computation* pp. 20-30, 1997.
> `quant-ph/9806029`

Our result on the simulation of irreversible computations with reversible ones, along with a wealth of other results on time/space/reversibility trade-offs can be found in

> M. Li and P. Vitanyi, "Reversibility and Adiabatic Computation: Trading Time and Space for Energy," *Proceedings of the Royal Society of London A, 452,* pp. 769-789, 1996.
> `quant-ph/9703022`

A number of elementary constructions for reversible circuits computing arithmetical functions are presented in

> V. Vedral et al., "Quantum Networks for Elementary Arithmetic Operations."
> `quant-ph/9511018`

More advanced techniques can be found in

> D. Beckman et al., "Efficient Networks for Quantum Factoring," CALT-68-2021, 1996.
> `quant-ph/9602016`
>
> P. Gossett, "Quantum Carry-Save Arithmetic," SGI-98-5433-3b, 1998.
> `quant-ph/9808061`
>
> C. Zalka, "Fast Versions of Shor's Quantum Factoring Algorithm," 1998
> `quant-ph/9806084`

Languages

The insight that uniqueness types can express the restrictions imposed by reversibility is due to Greg Baker, and can be found in his thesis from Macquarie University (currently unpublished).

The programming language QCL, being developed by Bernhard Ömer for his doctoral thesis at the Technical University of Vienna, incorporates a number of the techniques described in chapter 6, although not all. It is documented at

> `http://tph.tuwien.ac.at/~oemer/qc/index.html/`

Linear logic was first developed by Girard in

> J.-Y. Girard, "Linear Logic," *Theoretical Computer Science 50,* pp. 1-102, 1987.

A detailed presentation of the use of linear logic to implement uniqueness types can be found in

> E. Barendsen and S. Smetsers, "Uniqueness Typing for Functional Languages with Graph Rewriting Semantics," *Mathematical Structures in Computer Science 6,* pp. 579-612, 1996.

Linear LISP is described, to a first approximation, in

> H. Baker, " 'Use-Once' Variables and Linear Objects — Storage Management, Reflection, and Multi-Threading," *ACM Sigplan Notices 30,* 1, pp. 45-52, 1995.

Algorithms

Simon's problem is described in

> D. Simon, "On the Power of Quantum Computation," *Proceedings of the 35th Annual IEEE Symposium on the Foundations of Computer Science,* pp. 116-123, 1994.

Shor's algorithm was first presented at the 1994 Annual IEEE Symposium on Foundations of Computer Science , along with a closely related algorithm for efficiently computing discrete logarithms. An expanded version of the conference paper is

> P. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithm on a Quantum Computer," *SIAM Journal on Computing 26,* 5, p. 1484-1509, 1997.
> `quant-ph/9508027`

The circuit efficiently implementing the quantum Fourier transform is presented in

> D. Coppersmith "An Approximate Fourier Transform Useful in Quantum Factoring," IBM Research Report RC19642.

An important optimization of the quantum Fourier transform to require only a linear number of gates is described in

> R. Griffiths and C.-S. Niu, "Semiclassical Fourier Tranform for Quantum Computation," *Physical Review Letters 76*, pp. 3228-3231, 1996.
> `quant-ph/9511007`

Grover's algorithm is described in

L. Grover "A Fast Quantum Mechanical Algorithm for Database Search," *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing,* pp. 212-219, 1996.
quant-ph/9605043

A proof that the algorithm is optimal (adapted from Christof Zalka's original lower bound) is contained in

L. Grover "How Fast Can a Quantum Computer Search?"
quant-ph/9809029

A unified treatment of a number of important quantum algorithms as applications of Fourier analysis can be found in

R. Josza, "Quantum Algorithms and the Fourier Transform," *Proceedings of the Royal Society of London A 454,* 1969, pp. 323-337, 1998
quant-ph/9707033

A unified treatment of a number of important quantum algorithms as applications of phase estimation techniques can be found in

R. Cleve et al., "Quantum Algorithms Revisited," *Proceedings of the Royal Society of London A 454,* 1969, pp. 339-354, 1998
quant-ph/9708016

Other Directions

Preskill's survey article on fault-tolerant quantum computation is also a good introduction to the literature on quantum error correction.

A good survey of quantum information theory is

C. Bennett and P. Shor, "Quantum Information Theory," *IEEE Transactions on Information Theory 44*, p. 2724-2743, 1998.

A brief introduction to quantum cryptography, along with an extensive bibliography in the field, can be found in

S. Lomonaco, "A Quick Glance at Quantum Cryptography," *Cryptologia, 23*,1,pp.1-41, 1999
quant-ph/9811056

An explanation of the impossibility of quantum bit commitment can be found in

> H. Chau and H.-K. Lo, "Making An Empty Promise With A Quantum Computer," *Fortschritte der Physik 46,* pp. 507-520, 1998.
> `quant-ph/9709053`

A reasonable introduction to quantum communication complexity is

> H. Buhrman et al., "Quantum vs. Classical Communication and Computation," *Proceedings of the 30th Annual ACM Symposium on the Theory of Computing*, 1998.

A proof that quantum finite-state automata are inequivalent to classical finite-state automata can be found in

> A. Kondacs and J. Watrous, "On the Power of Quantum Finite State Automata," *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science,* pp. 66-75, 1997.

An interesting, if somewhat questionable, set of definitions for quantum context-free grammars and quantum push-down automata are proposed in

> C. Moore and J. Crutchfield, "Quantum Automata and Quantum Grammars."
> `quant-ph/9707031`