Stabilizer Codes and Quantum Error Correction

Thesis by
Daniel Gottesman

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California

2002
(Submitted May 21, 1997)

Acknowledgements

I would like to thank my advisor John Preskill for his guidance, and the members of the QUIC collaboration, particularly David Beckman, John Cortese, Jarah Evslin, Chris Fuchs, Sham Kakade, Andrew Landahl, and Hideo Mabuchi, for many stimulating conversations. My graduate career was supported by a National Science Foundation Graduate Fellowship, by the U. S. Department of Energy under Grant No. DE-FG03-92-ER40701, and by DARPA under Grant No. DAAH04-96-1-0386 administered by the Army Research Office.

Abstract

Controlling operational errors and decoherence is one of the major challenges facing the field of quantum computation and other attempts to create specified many-particle entangled states. The field of quantum error correction has developed to meet this challenge. A group-theoretical structure and associated subclass of quantum codes, the stabilizer codes, has proved particularly fruitful in producing codes and in understanding the structure of both specific codes and classes of codes. I will give an overview of the field of quantum error correction and the formalism of stabilizer codes. In the context of stabilizer codes, I will discuss a number of known codes, the capacity of a quantum channel, bounds on quantum codes, and fault-tolerant quantum computation.

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Preliminary Material

1.1 Quantum Computers

Computers have changed the world in many ways. They are ubiquitous, running air-traffic control and manufacturing plants, providing movie special effects and video games, and serving as a substrate for electronic mail and the World Wide Web. While computers allow us to solve many problems that were simply impossible before their advent, a number of problems require too much computation to be practical even for relatively simple inputs, and using the most powerful computers.

The field of classical complexity theory has developed to classify problems by their difficulty. A class of problems is generally considered tractable if an algorithm exists to solve it with resources (such as time and memory) polynomial in the size of the input. Two well-known classically intractable problems are factoring an n-bit number and the Traveling Salesman problem (finding the minimum cyclic path connecting n cities with specified distances between them). Both of these problems are in the complexity class NP (for "non-deterministic polynomial"):[1] given a black box that solves the problem (an *oracle*), we can check in polynomial time that the solution is correct. The Traveling Salesman problem is an NP-complete problem; that is, any problem in NP can be transformed into an instance of the Traveling Salesman problem in polynomial time. If we can solve the Traveling Salesman problem in polynomial time, we can solve any NP problem in polynomial time. Factoring may or may not be NP-complete, but so much work has been done attempting to solve it that the consensus is that it is classically intractable, and RSA public-key cryptography, which is used, for instance, to send credit-card numbers in Web browsing software, depends on the difficulty of factoring large numbers.

[1] Strictly speaking, it is the associated decision problems that are in NP.

As computer hardware develops over time, the underlying technology continually changes to become faster, smaller, and generally better. What was impossible on yesterday's computers may be quite possible today. A problem that was intractable on the earlier hardware might become tractable with the new technology. However, the strong Church-Turing Thesis [1] states that this is not the case, and that every physical implementation of universal computation can simulate any other implementation with only a polynomial slowdown.[2] In this way, the Church-Turing Thesis protects complexity theory from obsolescence as computer technology improves. While a new computer may be able to factor larger numbers, the difficulty of factoring numbers will still scale the same way with the size of the input on the new hardware as on the old hardware.

Another problem that has proven to be classically intractable is simulating quantum systems. A single spin-1/2 particle, such as an electron trapped in a quantum dot, has a two-dimensional space of states, which can be considered to describe the direction of its spin. A similar classical particle such as a Heisenberg spin would also have a two-dimensional space of states. However, n quantum particles have a 2^n-dimensional state space, while n classical Heisenberg spins would only have a $2n$-dimensional space of states. The extra states in the quantum system come from the presence of entangled states between many different particles. Note that while an n-bit classical digital computer has 2^n possible states, they only form an n-dimensional state space, since a state can be described by an n-component binary vector. To describe a state in a quantum computer with n qubits requires a complex vector with 2^n components. I give a basic introduction to quantum mechanics in section 1.2. Quantum systems are difficult to simulate classically because they generically utilize the full 2^n-dimensional Hilbert space as they evolve, requiring exponential classical resources.

This fact led Feynman to conjecture that a quantum computer which used quantum mechanics intrinsically might be more powerful than a computer mired in the classical world [2]. While this seems a sensible suggestion when just looking at quantum mechanics, it is in fact quite revolutionary in that it suggests that the strong Church-Turing Thesis is wrong![3] This opens up the possibility that classical complexity classes might not apply for quantum computers, and that some classically intractable problems might become tractable. The most spectacular instance of this is Shor's discovery of an algorithm to factor numbers on a quantum computer in a polynomial time in the number of digits [3]. Another impressive algorithm is Grover's algorithm [4], which can find a single object in an unsorted database of N objects in $O(\sqrt{N})$ time on a quantum computer, while the same task would require an exhaustive search on a classical computer, taking $O(N)$ time. It has been shown that $O(\sqrt{N})$ time is the best possible speed for this task [5], which tends to suggest that NP-complete problems are still intractable on a quantum computer, although this has not

[2] The original Church-Turing thesis only states that any universal computer can simulate any other computer, but the requirement of polynomial resources is a useful strengthening.

[3] A classical computer can simulate a quantum computer, but only with exponential resources, so the weak Church-Turing Thesis does still hold.

been shown (note that a proof of this would also show P $\neq$ NP for a classical computer).

However, declaring by theoretical fiat the basic properties of a quantum computer is a far cry from actually building one and using it to factor large numbers. Nevertheless, the first steps in building a quantum computer have been taken. Any quantum computer requires a system with long-lived quantum states and a way to interact them. Typically, we consider systems comprised of a number of two-state subsystems, which are called *qubits* (for "quantum bits"). There are many proposals for how to build a quantum computer. Some possible physical realizations of qubits are:

- the ground and excited states of ions stored in a linear ion trap, with interactions between ions provided through a joint vibrational mode [6, 7].
- photons in either polarization, with interactions via cavity QED [8].
- nuclear spin states in polymers, with interactions provided by nuclear magnetic resonance techniques [9].

While these implementations are seemingly very different, it is possible to simulate the computational process of one system on any of the others, providing a quantum analogue to the Church-Turing Thesis (although there are difficult technical or theoretical problems with scaling up the size of these implementations).

These suggested implementations of quantum computers all share a much higher susceptibility to errors than modern classical computers. While further development may reduce the size of errors by orders of magnitude, it is unlikely that quantum computers will ever reach the incredible reliability of classical computers. Modern classical computers guard against error largely by being digital instead of analog — instead of allowing each bit of the computer to vary continuously between 0 and 1, at each time step the hardware kicks the bit back to the nearer of 0 and 1. This prevents small errors from building up into large errors, which are therefore drastically reduced. The same technique cannot be used in a quantum computer, because continually measuring each qubit would destroy the entangled states that distinguish a quantum computer from a classical computer.

Entangled states are in general very delicate, and making a measurement on one will typically collapse it into a less entangled state. Small interactions with the environment provide a sort of continuous measurement of a system, and as the system grows in size, these become harder and harder to ignore. The system will *decohere* and begin to look like a classical system. Decoherence is why the world looks classical at a human scale. Reducing interactions with the environment can reduce the effects of decoherence, but not eliminate them entirely.

Even if the basal error rate in a quantum computer can be reduced to some small value ϵ per unit time, after N time steps, the probability of surviving without an error is only $(1-\epsilon)^N$, which decreases exponentially with N. Even

if an algorithm runs in polynomial time on an error-free computer, it will require exponentially many runs on a real computer unless something can be done to control the errors.

The same problem occurs for classical computers. There, the problem can be solved in principle by the use of error-correcting codes. In practice, they are not usually necessary for normal computer operation, but they are essential to overcome noise in communications channels. I give a basic introduction to the theory of classical error-correcting codes in section 1.3.

Classical error-correction techniques cannot be directly carried over to quantum computers for two reasons. First of all, the classical techniques assume we can measure all of the bits in the computer. For a quantum computer, this would destroy any entanglement between qubits. More importantly, a classical computer only needs to preserve the bit values of 0 and 1. A quantum computer also needs to keep phase information in entangled states. Thus, while quantum error-correcting codes are related to classical codes, they require a somewhat new approach.

The first quantum error-correcting codes were discovered by Shor [10] and Steane [11]. I discuss Shor's original code and some basics of quantum error-correcting codes in chapter 2. I then go on to describe the formalism of stabilizer codes in chapter 3, along with some simple examples and methods for creating new codes from old ones. Chapter 4 describes how to build networks to encode and decode stabilizer codes. Because we will want to use these codes in the operation of quantum computers, in chapter 5, I will discuss how to perform operations on states encoded using a quantum error-correcting code without losing the protection against errors. Chapter 6 describes how to use concatenated codes to do arbitrarily long calculations as long as the basic error rate is below some threshhold value, and presents a rough calculation of that threshhold. Chapter 7 discusses known upper and lower bounds on the existence of stabilizer codes and the channel capacity. Finally, in chapter 8, I will give a partial list of known quantum error-correcting codes and their properties. Appendix A contains a brief discussion of quantum gates and a list of symbols for them used in figures. Appendix B contains a glossary of useful terms for discussing quantum error-correcting codes.

Since the promise of quantum computation has attracted scientists from a number of fields, including computer science, mathematics, and physics, some of the background one group takes for granted may be alien to others. Therefore, in the following two sections, I have provided basic introductions to quantum mechanics and classical coding theory. People familiar with one or both fields should skip the appropriate section(s). For a more complete treatment of quantum mechanics, see [12]. For a more complete treatment of classical error-correcting codes, see [13].

1.2 Introduction to Quantum Mechanics

The state of a classical computer is a string of 0s and 1s, which is a vector over the finite field $\mathbf{Z}_2$. The state of a quantum computer (or any quantum system) is instead a vector over the complex numbers $\mathbf{C}$. Actually, a quantum state lies in a Hilbert space, since there is an inner product (which I will define later). The state is usually written $|\psi\rangle$, which is called a *ket*. A classical computer with n bits has 2^n possible states, but this is only an n-dimensional vector space over $\mathbf{Z}_2$. A quantum computer with n qubits is a state in a 2^n-dimensional complex vector space. For a single qubit, the standard basis vectors are written as $|0\rangle$ and $|1\rangle$. An arbitrary single-qubit state is then

$$\alpha|0\rangle + \beta|1\rangle. \tag{1.1}$$

α and β are complex numbers, with $|\alpha|^2 + |\beta|^2 = 1$. This is a *normalized* state. With multiple qubits, we can have states that cannot be written as the product of single-qubit states. For instance,

$$\frac{1}{\sqrt{2}}\left(|00\rangle + |11\rangle\right) \tag{1.2}$$

cannot be decomposed in this way. Such a state is said to be *entangled*. Entangled states are what provide a quantum computer with its power. They will also play a major role in quantum error correction. The particular state (1.2) is called an Einstein-Podalsky-Rosen pair (or EPR) pair, and serves as a useful basic unit of entanglement in many applications.

If we make a measurement on the qubit in equation (1.1), we get a classical number corresponding to one of the basis states. The measurement disturbs the original state, which collapses into the basis state corresponding to the measurement outcome. If we measure the state (1.1), the outcome will be 0 with probability $|\alpha|^2$, and it will be 1 with probability $|\beta|^2$. The normalization ensures that the probability of getting some result is exactly 1. Through most of this thesis, I will instead write down unnormalized states. These states will stand for the corresponding normalized states, which are formed by multiplying the unnormalized states by an appropriate constant. The overall phase of a state vector has no physical significance.

The measurement we made implements one of two projection operators, the projections on the basis $|0\rangle$, $|1\rangle$. This is not the only measurement we can make on a single qubit. In fact, we can project on any basis for the Hilbert space of the qubit. If we have multiple qubits, we can measure a number of different qubits independently, or we can measure some joint property of the qubits, which corresponds to projecting on some entangled basis of the system. Note that the projection on the basis $|0\rangle$, $|1\rangle$ for either qubit destroys the entanglement of the state (1.2), leaving it in a tensor product state.

A particularly fruitful way to understand a quantum system is to look at the behavior of various operators acting on the states of the system. For instance, a nice set of operators to consider for a single qubit is the set of Pauli spin

matrices

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \ \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \text{ and } \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{1.3}$$

The original measurement I described corresponds to measuring the eigenvalue of σ_z. The corresponding projection operators are $\frac{1}{2}(I \pm \sigma_z)$. If we have a spin-1/2 particle, this measurement is performed by measuring the spin of the particle along the z axis. We could also measure along the x or y axis, which corresponds to measuring the eigenvalue of σ_x or σ_y. The projections are $\frac{1}{2}(I \pm \sigma_x)$ and $\frac{1}{2}(I \pm \sigma_y)$.

We can also make measurements of more general operators, provided they have real eigenvalues. A matrix A has real eigenvalues iff it is Hermitian: $A^\dagger = A$, where $A^\dagger$ is the *Hermitian adjoint* (or just *adjoint*), equal to the complex conjugate transpose. Note that all of the Pauli spin matrices are Hermitian.

The Pauli matrices also satisfy an important algebraic property — they *anticommute* with each other. That is,

$$\{\sigma_i, \sigma_j\} = \sigma_i\sigma_j + \sigma_j\sigma_i = 0 \tag{1.4}$$

whenever $i \neq j$ (with $i, j \in \{x, y, z\}$). Another possible relationship between two operators A and B is for them to *commute*. That is,

$$[A, B] = AB - BA = 0. \tag{1.5}$$

It is possible for two matrices to neither commute nor anticommute, and, in fact, this is the generic case. Two commuting matrices can be simultaneously diagonalized. This means that we can measure the eigenvalue of one of them without disturbing the eigenvectors of the other. Conversely, if two operators do not commute, measuring one will disturb the eigenvectors of the other, so we cannot simultaneously measure non-commuting operators.

There is a natural complex inner product on quantum states. Given an orthonormal basis $|\psi_i\rangle$, the inner product between $|\alpha\rangle = \sum c_i|\psi_i\rangle$ and $|\beta\rangle = \sum d_i|\psi_i\rangle$ is

$$\langle\alpha|\beta\rangle = \sum c_i^* d_j \langle\psi_i|\psi_j\rangle = \sum c_i^* d_i. \tag{1.6}$$

Each ket $|\psi\rangle$ corresponds to a *bra* $\langle\psi|$ and the Hermitian adjoint is the adjoint with respect to this inner product, so $U|\psi\rangle$ corresponds to $\langle\psi|U^\dagger$. The operator $\sum |\psi\rangle\langle\phi|$ acts on the Hilbert space as follows:

$$\left(\sum |\psi\rangle\langle\phi|\right)|\alpha\rangle = \sum \langle\phi|\alpha\rangle \ |\psi\rangle. \tag{1.7}$$

The inner product can reveal a great deal of information about the structure of a set of states. For instance, $\langle\psi|\phi\rangle = 1$ if and only if $|\psi\rangle = |\phi\rangle$.

Eigenvectors of a Hermitian operator A with different eigenvalues are automatically orthogonal:

$$\langle\psi|A|\phi\rangle = \langle\psi|\big(A|\phi\rangle\big) = \lambda_\phi\langle\psi|\phi\rangle \tag{1.8}$$

$$= \big(\langle\psi|A\big)|\phi\rangle = \lambda_\psi^*\langle\psi|\phi\rangle. \tag{1.9}$$

Since the eigenvalues of A are real, it follows that $\langle\psi|\phi\rangle = 0$ whenever $\lambda_\phi \neq \lambda_\psi$. Conversely, if $\langle\psi|\phi\rangle = 0$, there exists a Hermitian operator for which $|\psi\rangle$ and $|\phi\rangle$ are eigenvectors with different eigenvalues.

We often want to consider a subsystem $\mathcal{A}$ of a quantum system $\mathcal{B}$. Since $\mathcal{A}$ may be entangled with the rest of the system, it is not meaningful to speak of the "state" of $\mathcal{A}$. If we write the state of $\mathcal{B}$ as $\sum |\psi_i\rangle|\phi_i\rangle$, where $|\psi_i\rangle$ is an orthonormal basis for $\mathcal{B} - \mathcal{A}$, and $|\phi_i\rangle$ are possible states for $\mathcal{A}$, then to an observer who only interacts with the subsystem $\mathcal{A}$, the subsystem appears to be in just one of the states $|\phi_i\rangle$ with some probability. $\mathcal{A}$ is said to be in a *mixed state* as opposed to the *pure state* of a closed system in a definite state.

We can extend the formalism to cover mixed states by introducing the *density matrix* ρ. For a pure system in the state $|\psi\rangle$, the density matrix is $|\psi\rangle\langle\psi|$. The density matrix for the subsystem for the entangled state above is $\sum |\phi_i\rangle\langle\phi_i|$. Density matrices are always positive and have $\operatorname{tr}\rho = 1$. To find the density matrix of a subsystem given the density matrix of the full system, simply trace over the degrees of freedom of the rest of the system.

Given a closed quantum system, time evolution preserves the inner product, so the time evolution operator U must be unitary. That is, $U^\dagger U = UU^\dagger = I$. An open system can be described as a subsystem of a larger closed system, so the evolution of the open system descends from the global evolution of the full system. Time evolution of the subsystem is described by some *superoperator* acting on the density matrix of the subsystem.

One fact about quantum states that has profound implications for quantum computation is that it is impossible to make a copy of an arbitrary unknown quantum state. This is known as the "No Cloning Theorem," [14] and is a consequence of the linearity of quantum mechanics. The proof is straightforward: Suppose we wish to have an operation that maps an arbitrary state

$$|\psi\rangle \to |\psi\rangle \otimes |\psi\rangle. \tag{1.10}$$

Then arbitrary $|\phi\rangle$ is mapped by

$$|\phi\rangle \to |\phi\rangle \otimes |\phi\rangle \tag{1.11}$$

as well. Because the transformation must be linear, it follows that

$$|\psi\rangle + |\phi\rangle \to |\psi\rangle \otimes |\psi\rangle + |\phi\rangle \otimes |\phi\rangle. \tag{1.12}$$

However,

$$|\psi\rangle \otimes |\psi\rangle + |\phi\rangle \otimes |\phi\rangle \neq (|\psi\rangle + |\phi\rangle) \otimes (|\psi\rangle + |\phi\rangle), \tag{1.13}$$

so we have failed to copy $|\psi\rangle + |\phi\rangle$. In general, if we pick an orthonormal basis, we can copy the basis states, but we will not have correctly copied superpositions of those basis states. We will instead have either measured the original system and therefore destroyed the superposition, or we will have produced a state that is entangled between the original and the "copy." This means that to perform quantum error correction, we cannot simply make backup copies of the quantum state to be preserved. Instead, we must protect the original from any likely error.

1.3 Introduction to Classical Coding Theory

Classical coding theory tends to concentrate on *linear codes*, a subclass of all possible codes with a particular relation between codewords. Suppose we wish to encode k bits using n bits. The data can be represented as a k-dimensional binary vector v. Because we are dealing with binary vectors, all the arithmetic is mod two. For a linear code, the encoded data is then Gv for some $n \times k$ matrix G (with entries from $\mathbf{Z}_2$), which is independent of v. G is called the *generator matrix* for the code. Its columns form a basis for the k-dimensional coding subspace of the n-dimensional binary vector space, and represent basis codewords. The most general possible codeword is an arbitrary linear combination of the basis codewords; thus the name "linear code."

Given a generator matrix G, we can calculate the dual matrix P, which is an $(n-k) \times n$ matrix of 0s and 1s of maximal rank $n-k$ with $PG = 0$. Since any codeword s has the form Gv, $Ps = PGv = 0v = 0$, and P annihilates any codeword. Conversely, suppose $Ps = 0$. Since P has rank $n-k$, it only annihilates a k-dimensional space spanned by the columns of G, and s must be a linear combination of these columns. Thus, $s = Gv$ for some v, and s is a valid codeword. The matrix P is called the *parity check matrix* for the code. It can be used to test if a given vector is a valid codeword, since $Ps = 0$ iff s is a codeword. The *dual code* is defined to be the code with generator matrix P^T and parity matrix G^T.

In order to consider the error-correcting properties of a code, it is useful to look at the *Hamming distance* between codewords. The Hamming distance between two vectors is the minimum number of bits that must be flipped to convert one vector to the other. The distance between a and b is equal to the *weight* (the number of 1s in the vector) of $a+b$. For a code to correct t single-bit errors, it must have distance at least $2t+1$ between any two codewords. A t bit error will take a codeword exactly distance t away from its original value, so when the distance between codewords is at least $2t+1$, we can distinguish errors on different codewords and correct them to the proper codewords. A code to encode k bits in n bits with minimum distance d is said to be an $[n, k, d]$ code.

Now suppose we consider a t bit error. We can write down a vector e to describe this vector by putting ones in the places where bits are flipped and zeros elsewhere. Then if the original codeword is s, after the error it is $s' = s + e$. If we apply the parity check matrix, we get

$$Ps' = P(s+e) = Ps + Pe = 0 + Pe = Pe, \tag{1.14}$$

so the value of Ps' does not depend on the value of s, only on e. If Pe is different for all possible errors e, we will be able to determine precisely what error occurred and fix it. Pe is called the *error syndrome*, since it tells us what the error is. Since $Pe = Pf$ iff $P(e-f) = 0$, to have a code of distance d, we need $Pe \neq 0$ for all vectors e of weight $d-1$ or less. Equivalently, any $d-1$ columns of P must be linearly independent.

We can place upper and lower bounds on the existence of linear codes to correct t errors. Each of the 2^k codewords has a *Hamming sphere* of radius t.

All the words inside the Hamming sphere come from errors acting on the same codeword. For a code on n bits, there are n one-bit errors, $\binom{n}{2}$ two-bit errors, and in general $\binom{n}{j}$ j-bit errors. The Hamming spheres cannot overlap, but they must all fit inside the vector space, which only has 2^n elements. Thus,

$$\sum_{j=0}^{t} \binom{n}{j} 2^k \leq 2^n. \tag{1.15}$$

This is called the *Hamming bound* on $[n, k, 2t+1]$ codes. As n, k, and t get large, this bound approaches the asymptotic form

$$\frac{k}{n} \leq 1 - H\left(\frac{t}{n}\right), \tag{1.16}$$

where $H(x)$ is the *Hamming entropy*

$$H(x) = -x \log_2 x - (1-x) \log_2 (1-x). \tag{1.17}$$

We can set a lower bound on the existence of $[n, k, 2t+1]$ linear codes as well, called the *Gilbert-Varshamov* bound. Suppose we have such a code (if necessary with $k = 0$) with

$$\sum_{j=0}^{2t} \binom{n}{j} 2^k < 2^n. \tag{1.18}$$

Then the spheres of distance $2t$ around each codeword do not fill the space, so there is some vector v that is at least distance $2t+1$ from each of the other codewords. In addition, $v + s$ (for any codeword s) is at least distance $2t+1$ from any other codeword s', since the distance is just $(v+s)+s' = v+(s+s')$, which is the distance between v and the codeword $s+s'$. This means that we can add v and all the vectors $v+s$ to the code without dropping the distance below $2t+1$. This gives us an $[n, k+1, 2t+1]$ code. We can continue this process until

$$\sum_{j=0}^{2t} \binom{n}{j} 2^k \geq 2^n. \tag{1.19}$$

Asymptotically, this becomes

$$\frac{k}{n} \geq 1 - H\left(\frac{2t}{n}\right). \tag{1.20}$$

Another case of great interest is the capacity of a classical channel. This is equal to the *efficiency* k/n of the most efficient code on an asymptotically large block that corrects measure one of the errors occuring. For instance, a common channel is the *binary symmetric channel*, where an error occurs independently on each bit with probability p for both 0 and 1. Shannon showed that channel capacity is just equal to one minus the entropy introduced by the channel [15]. For the binary symmetric channel, the entropy is just the Hamming entropy

$H(p)$, so the capacity is $1 - H(p)$, coinciding with the Hamming bound for the expected number of errors $t = pn$. Shannon also showed that the capacity of a channel can be achieved by choosing codewords at random, then discarding only a few of them (measure zero asymptotically).

Chapter 2

Basics of Quantum Error Correction

2.1 The Quantum Channel

Now we turn to the quantum channel. A noisy quantum channel can be a regular communications channel which we expect to preserve at least some degree of quantum coherence, or it can be the passage of time as a set of qubits sits around, interacting with its environment, or it can be the result of operating with a noisy gate on some qubits in a quantum computer. In any of these cases, the input of a pure quantum state can produce a mixed state as output as the data qubits become entangled with the environment. Even when a pure state comes out, it might not be the same state as the one that went in.

At first it appears that trying to correct a mixed state back into the correct pure state is going to be harder than correcting an erroneous pure state, but this is not the case. The output mixed state can be considered as an ensemble of pure states. If we can correct each of the pure states in the ensemble back to the original input state, we have corrected the full mixed state. Another way of phrasing this is to say the channel applies a superoperator to the input density matrix. We can diagonalize this superoperator and write it as the direct sum of a number of different matrices acting directly on the possible input pure states with various probabilities. If the code can correct any of the possible matrices, it can correct the full superoperator. A key point is that the individual matrices need not be unitary. From now on, I will only consider the effects of a (possibly non-unitary) matrix acting on a pure state.

2.2 A Simple Code

For the moment, let us consider only channels which cause an error on a single qubit at a time. We wish to protect a single logical qubit against error. We

cannot send it through the channel as is, because the one qubit that is affected might be the one we want to keep. Suppose we send through nine qubits after encoding the logical qubit as follows:

$$|0\rangle \rightarrow |\overline{0}\rangle = (|000\rangle + |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle) \quad (2.1)$$
$$|1\rangle \rightarrow |\overline{1}\rangle = (|000\rangle - |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle). \quad (2.2)$$

The data is no longer stored in a single qubit, but instead spread out among nine of them. Note that even if we know the nine qubits are in one of these two states, we cannot determine which one without making a measurement on at least three qubits. This code is due to Shor [10].

Suppose the channel flips a single qubit, say the first one, switching $|0\rangle$ and $|1\rangle$. Then by comparing the first two qubits, we find they are different, which is not allowed for any valid codeword. Therefore we know an error occurred, and furthermore, it flipped either the first or second qubit. Note that we do not actually measure the first and second qubits, since this would destroy the superposition in the codeword; we just measure the difference between them.

Now we compare the first and third qubits. Since the first qubit was flipped, it will disagree with the third; if the second qubit had been flipped, the first and third would have agreed. Therefore, we have narrowed down the error to the first qubit and we can fix it simply by flipping it back. To handle possible bit flips on the other blocks of three, we do the same comparisons inside the other blocks.

However, this is not the only sort of error that could have occurred. The channel might have left the identity of the 0 and 1 alone, but altered their relative phase, introducing, for instance, a relative factor of -1 when the first qubit is $|1\rangle$. Then the two basis states become

$$|\overline{0}\rangle \rightarrow (|000\rangle - |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle) \quad (2.3)$$
$$|\overline{1}\rangle \rightarrow (|000\rangle + |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle). \quad (2.4)$$

By comparing the sign of the first block of three with the second block of three, we can see that a sign error has occurred in one of those blocks. Then by comparing the signs of the first and third blocks of three, we narrow the sign error down to the first block, and flip the sign back to what it should be. Again, we do not want to actually measure the signs, only whether they agree. In this case, measuring the signs would give us information about whether the state is $|\overline{0}\rangle$ or $|\overline{1}\rangle$, which would destroy any superposition between them.

This does not exhaust the list of possible one qubit errors. For instance, we could have both a bit flip and a sign flip on the same qubit. However, by going through both processes described above, we will fix first the bit flip, then the sign flip (in fact, this code will correct a bit flip and a sign flip even if they are on different qubits). The original two errors can be described as the operation of

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \text{ and } \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \quad (2.5)$$

The simultaneous bit and sign flip is

$$\sigma_y = i\sigma_x\sigma_z = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}. \tag{2.6}$$

Sometimes I will write σ_{xi}, σ_{yi}, or σ_{zi} to represent σ_x, σ_y, or σ_z acting on the ith qubit.

The most general one-qubit error that can occur is some 2×2 matrix; but such a matrix can always be written as the (complex) linear combination of σ_x, σ_y, σ_z, and the 2×2 identity matrix I. Consider what happens to the code when such an error occurs:

$$|\psi\rangle = \alpha|\overline{0}\rangle + \beta|\overline{1}\rangle \rightarrow a\sigma_{xi}|\psi\rangle + b\sigma_{yi}|\psi\rangle + c\sigma_{zi}|\psi\rangle + d|\psi\rangle. \tag{2.7}$$

Suppose we perform the process above, comparing bits within a block of three, and comparing the signs of blocks of three. This acts as a measurement of which error (or the identity) has occurred, causing the state, originally in a superposition, to collapse to $\sigma_{xi}|\psi\rangle$ with probability $|a|^2$, to $\sigma_{yi}|\psi\rangle$ with probability $|b|^2$, to $\sigma_{zi}|\psi\rangle$ with probability $|c|^2$, and to $|\psi\rangle$ with probability $|d|^2$. In any of the four cases, we have determined which error occurred and we can fix it.

2.3 Properties of Any Quantum Code

Now let us consider properties of more general codes. A code to encode k qubits in n qubits will have 2^k basis codewords corresponding to the basis of the original states. Any linear combination of these basis codewords is also a valid codeword, corresponding to the same linear combination of the unencoded basis states. The space T of valid codewords (the *coding space*) is therefore a Hilbert space in its own right, a subspace of the full 2^n-dimensional Hilbert space. As with Shor's nine-qubit code, if we can correct errors E and F, we can correct $aE + bF$, so we only need to consider whether the code can correct a basis of errors. One convenient basis to use is the set of tensor products of σ_x, σ_y, σ_z, and I. The *weight* of an operator of this form is the number of qubits on which it differs from the identity. The set of all these tensor products with a possible overall factor of -1 or $\pm i$ forms a group $\mathcal{G}$ under multiplication. $\mathcal{G}$ will play a major role in the stabilizer formalism. Sometimes I will write it $\mathcal{G}_n$ to distinguish the groups for different numbers of qubits. $\mathcal{G}_1$ is just the quaternionic group; $\mathcal{G}_n$ is the direct product of n copies of the quaternions modulo all but a global phase factor.

In order for the code to correct two errors E_a and E_b, we must always be able to distinguish error E_a acting on one basis codeword $|\psi_i\rangle$ from error E_b acting on a different basis codeword $|\psi_j\rangle$. We can only be sure of doing this if $E_a|\psi_1\rangle$ is orthogonal to $E_b|\psi_2\rangle$; otherwise there is some chance of confusing them. Thus,

$$\langle\psi_i|E_a^\dagger E_b|\psi_j\rangle = 0 \tag{2.8}$$

when $i \neq j$ for correctable errors E_a and E_b. Note that we normally include the identity in the set of possible "errors," since we do not want to confuse an error

on one qubit with nothing happening to another. If we have a channel in which we are certain *some* error occurred, we do not need to include the identity as a possible error. In any case, the set of correctable errors is unlikely to be a group — it does not even need to be closed under multiplication.

However, (2.8) is insufficient to guarantee a code will work as a quantum error-correcting code. When we make a measurement to find out about the error, we must learn nothing about the actual state of the code within the coding space. If we did learn something, we would be disturbing superpositions of the basis states, so while we might correct the basis states, we would not be correcting an arbitrary valid codeword. We learn information about the error by measuring $\langle\psi_i|E_a^\dagger E_b|\psi_i\rangle$ for all possible errors E_a and E_b. This quantity must therefore be the same for all the basis codewords:

$$\langle\psi_i|E_a^\dagger E_b|\psi_i\rangle = \langle\psi_j|E_a^\dagger E_b|\psi_j\rangle. \tag{2.9}$$

We can combine equations (2.8) and (2.9) into a single equation:

$$\langle\psi_i|E_a^\dagger E_b|\psi_j\rangle = C_{ab}\delta_{ij}, \tag{2.10}$$

where $|\psi_i\rangle$ and $|\psi_j\rangle$ run over all possible basis codewords, E_a and E_b run over all possible errors, and C_{ab} is independent of i and j. This condition was found by Knill and Laflamme [16] and Bennett *et al.* [17].

The above argument shows that (2.10) is a necessary condition for the code to correct the errors $\{E_a\}$. It is also a sufficient condition: The matrix C_{ab} is Hermitian, so it can be diagonalized. If we do this and rescale the errors $\{E_a\}$ appropriately, we get a new basis $\{F_a\}$ for the space of possible errors, with either

$$\langle\psi_i|F_a^\dagger F_b|\psi_j\rangle = \delta_{ab}\delta_{ij} \tag{2.11}$$

or

$$\langle\psi_i|F_a^\dagger F_b|\psi_j\rangle = 0, \tag{2.12}$$

depending on a. Note that this basis will not necessarily contain operators that are tensor products of one-qubit operators. Errors of the second type actually annihilate any codeword, so the probability of one occuring is strictly zero and we need not consider them. The other errors always produce orthogonal states, so we can make some measurement that will tell us exactly which error occurred, at which point it is a simple matter to correct it. Therefore, a code satisfies equation (2.10) for all E_a and E_b in some set $\mathcal{E}$ iff the code can correct all errors in $\mathcal{E}$.

Another minor basis change allows us to find a basis where any two errors acting on a given codeword either produce orthogonal states or exactly the same state. The errors F_a that annihilate codewords correspond to two errors that act the same way on codewords. For instance, in Shor's nine-qubit code, σ_{z1} and σ_{z2} act the same way on the code, so $\sigma_{z1} - \sigma_{z2}$ will annihilate codewords. This phenomenon will occur iff C_{ab} does not have maximum rank. A code for which C_{ab} is singular is called a *degenerate* code, while a code for which it is not is *nondegenerate*. Shor's nine-qubit code is degenerate; we will see many

examples of nondegenerate codes later. Note that whether a code is degenerate or not depends on the set of errors it is intended to correct. For instance, a two-error-correcting degenerate code might be nondegenerate when considered as a one-error-correcting code.

In equation (2.10), $E = E_a^\dagger E_b$ is still in the group $\mathcal{G}$ when E_a and E_b are in $\mathcal{G}$. The weight of the smallest E in $\mathcal{G}$ for which (2.10) does *not* hold is called the *distance* of the code. A quantum code to correct up to t errors must have distance at least $2t + 1$. Every code has distance at least one. A distance d code encoding k qubits in n qubits is described as an $[n, k, d]$ code. Note that a quantum $[n, k, d]$ code is often written in the literature as $[[n, k, d]]$ to distinguish it from a classical $[n, k, d]$ code. I have chosen the notation $[n, k, d]$ to emphasize the similarities with the classical theory; when I need to distinguish, I will do so using the words "quantum" and "classical."

We can also consider variations of the usual error-correction problem. For instance, suppose we only want to detect if an error has occurred, not to correct it. This could, for instance, be used to prevent errors using the quantum Zeno effect [18]. In this case, we do not need to distinguish error E_a from E_b, only from the identity. We can use the same argument to find (2.10), only now $E_b = I$ always. This means a code to detect s errors must have distance at least $s + 1$. Another variation is when we know in which qubit(s) an error has occurred, as in the quantum erasure channel [19]. In this case, we only need distinguish E_a from those E_b affecting the same qubits. This means that $E_a^\dagger E_b$ has the same weight as E_a, and to correct r such located errors, we need a code of distance at least $r + 1$. We can also imagine combining all of these tasks. A code to correct t arbitrary errors, r additional located errors, and detect a further s errors must have distance at least $r + s + 2t + 1$.

2.4 Error Models

In this thesis, I will mostly assume that errors occur independently on different qubits, and that when an error occurs on a qubit, it is equally likely to be a σ_x, σ_y, or σ_z error. If the probability ϵ of error per qubit is fairly small, it is often useful to simply ignore the possibility of more than t errors, since this only occurs with probability $O(\epsilon^{t+1})$. Thus, I will typically deal with codes that correct up to t arbitrary errors. Such a code will handle any error on up to t qubits that leaves the data somewhere in the normal computational space (although moving it outside of the space of valid codewords).

In some systems, there will be errors that move the system outside of the computational space. For instance, if the data is stored as the ground or metastable excited state of an ion, the electron might instead end up in a different excited state. If the data is stored in the polarization of a photon, the photon might escape. In both of these cases, the normal error correction networks will not function properly, since they assume that the qubit is either in the state $|0\rangle$ or $|1\rangle$. However, by performing some measurement that distinguishes between the computational Hilbert space and other possible states, we can determine

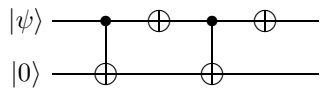

Figure 2.1: Network to detect leakage errors.

not only that this sort of *leakage error* has occurred, but also on which qubit it has occurred. Then we can cool the atom to the ground state or introduce a new photon with random polarization, and the error becomes a located error, which was discussed at the end of the previous section. One possible network of gates to detect a leakage error is given in figure 2.1 (see appendix A for a description of the symbols used in this and later figures). This network asssumes that states outside the normal computational space do not interact at all with other qubits. If the data state $|\psi\rangle$ is either $|0\rangle$ or $|1\rangle$, the ancilla qubit will flip and become $|1\rangle$. If the data state is neither $|0\rangle$ nor $|1\rangle$, the ancilla will remain $|0\rangle$, thus signalling a leakage error on this data qubit.

Another possible difficulty arises when correlated errors on multiple qubits can occur. While this can in principle be a severe problem, it can be handled without a change in formalism as long as the chance of a correlated error drops rapidly enough with the size of the blocks of errors. Since a t-qubit error will occur with probability $O(\epsilon^t)$ when the probability of uncorrelated single-qubit errors is ϵ, as long as the probability of a t-qubit correlated error is $O(\epsilon^t)$, the correlated errors cause no additional problems.

In real systems, the assumption that errors are equally likely to be σ_x, σ_y, and σ_z errors is a poor one. In practice, some linear combinations of σ_x, σ_y, and σ_z are going to be more likely than others. For instance, when the qubits are ground or excited states of an ion, a likely source of errors is spontaneous emission. After some amount of time, the excited state will either decay to the ground state, producing the error $\sigma_x + i\sigma_y$ with probability ϵ, or it will not, which changes the relative amplitudes of $|0\rangle$ and $|1\rangle$, resulting in the error $I - \sigma_z$ with probability $O(\epsilon^2)$. A channel that performs this sort of time evolution is known as an *amplitude damping* channel. Since the only $O(1)$ effect of time evolution is the identity, this sort of error can be protected against to lowest order by a code to correct an arbitrary single error. However, codes that take account of the restricted possibilities for errors can be more efficient than codes that must correct a general error [20], and understanding the physically likely sources of error will certainly be an important part of engineering quantum computers.

Chapter 3

Stabilizer Coding

3.1 The Nine-Qubit Code Revisited

Let us look more closely at the procedure we used to correct errors for the nine-qubit code. To detect a bit flip error on one of the first three qubits, we compared the first two qubits and the first and third qubits. This is equivalent to measuring the eigenvalues of $\sigma_{z1}\sigma_{z2}$ and $\sigma_{z1}\sigma_{z3}$. If the first two qubits are the same, the eigenvalue of $\sigma_{z1}\sigma_{z2}$ is $+1$; if they are different, the eigenvalue is -1. Similarly, to detect a sign error, we compare the signs of the first and second blocks of three and the first and third blocks of three. This is equivalent to measuring the eigenvalues of $\sigma_{x1}\sigma_{x2}\sigma_{x3}\sigma_{x4}\sigma_{x5}\sigma_{x6}$ and $\sigma_{x1}\sigma_{x2}\sigma_{x3}\sigma_{x7}\sigma_{x8}\sigma_{x9}$. Again, if the signs agree, the eigenvalues will be $+1$; if they disagree, the eigenvalues will be -1. In order to totally correct the code, we must measure the eigenvalues of a total of eight operators. They are listed in table 3.1.

The two valid codewords $|\overline{0}\rangle$ and $|\overline{1}\rangle$ in Shor's code are eigenvectors of all eight of these operators with eigenvalue $+1$. All the operators in $\mathcal{G}$ that fix both $|\overline{0}\rangle$ and $|\overline{1}\rangle$ can be written as the product of these eight operators. The set of operators that fix $|\overline{0}\rangle$ and $|\overline{1}\rangle$ form a group S, called the *stabilizer* of the code, and M_1 through M_8 are the generators of this group.

When we measure the eigenvalue of M_1, we determine if a bit flip error has

M_1	σ_z	σ_z	I	I	I	I	I	I	I
M_2	σ_z	I	σ_z	I	I	I	I	I	I
M_3	I	I	I	σ_z	σ_z	I	I	I	I
M_4	I	I	I	σ_z	I	σ_z	I	I	I
M_5	I	I	I	I	I	I	σ_z	σ_z	I
M_6	I	I	I	I	I	I	σ_z	I	σ_z
M_7	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	I	I	I
M_8	σ_x	σ_x	σ_x	I	I	I	σ_x	σ_x	σ_x

Table 3.1: The stabilizer for Shor's nine-qubit code

occurred on qubit one or two, i.e., if σ_{x1} or σ_{x2} has occurred. Note that both of these errors anticommute with M_1, while σ_{x3} through σ_{x9}, which cannot be detected by just M_1, commute with it. Similarly, M_2 detects σ_{x1} or σ_{x3}, which anticommute with it, and M_7 detects σ_{z1} through σ_{z6}. In general, if $M \in S$, $\{M, E\} = 0$, and $|\psi\rangle \in T$, then

$$ME|\psi\rangle = -EM|\psi\rangle = -E|\psi\rangle, \tag{3.1}$$

so $E|\psi\rangle$ is an eigenvector of M with eigenvalue -1 instead of $+1$ and to detect E we need only measure M.

The distance of this code is in fact three. Even a cursory perusal reveals that any single-qubit operator σ_{xi}, σ_{yi}, or σ_{zi} will anticommute with one or more of M_1 through M_8. Since states with different eigenvalues are orthogonal, condition (2.10) is satisfied when E_a has weight one and $E_b = I$. We can also check that every two-qubit operator E anticommutes with some element of S, except for those of the form $\sigma_{za}\sigma_{zb}$ where a and b are in the same block of three. However, the operators of this form are actually in the stabilizer. This means that $\sigma_{za}\sigma_{zb}|\psi\rangle = |\psi\rangle$ for any codeword $|\psi\rangle$, and $\langle\psi|\sigma_{za}\sigma_{zb}|\psi\rangle = \langle\psi|\psi\rangle = 1$ for all codewords $|\psi\rangle$, and these operators also satisfy equation (2.10). Since $\sigma_{za}\sigma_{zb}$ is in the stabilizer, both σ_{za} and σ_{zb} act the same way on the codewords, and there is no need to distinguish them. When we get to operators of weight three, we do find some for which (2.10) fails. For instance, $\sigma_{x1}\sigma_{x2}\sigma_{x3}$ commutes with everything in S, but

$$\langle\overline{0}|\sigma_{x1}\sigma_{x2}\sigma_{x3}|\overline{0}\rangle = +1 \tag{3.2}$$

$$\langle\overline{1}|\sigma_{x1}\sigma_{x2}\sigma_{x3}|\overline{1}\rangle = -1. \tag{3.3}$$

3.2 The General Stabilizer Code

The stabilizer construction applies to many more codes than just the nine-qubit one [21, 22]. In general, the stabilizer S is some Abelian subgroup of $\mathcal{G}$ and the coding space T is the space of vectors fixed by S. Since σ_y has imaginary components, while σ_x and σ_z are real, with an even number of σ_y's in each element of the stabilizer, all the coefficients in the basis codewords can be chosen to be real; if there are an odd number of σ_y's, they may be imaginary. However, Rains has shown that whenever a (possibly complex) code exists, a real code exists with the same parameters [23]. Therefore, I will largely restrict my attention to real codes.

For a code to encode k qubits in n, T has 2^k dimensions and S has 2^{n-k} elements. S must be an Abelian group, since only commuting operators can have simultaneous eigenvectors, but provided it is Abelian and neither i nor -1 is in S, the space $T = \{|\psi\rangle \text{ s.t. } M|\psi\rangle = |\psi\rangle \ \forall M \in S\}$ does have dimension 2^k. At this point it will be helpful to note a few properties of $\mathcal{G}$. Since $\sigma_x^2 = \sigma_y^2 = \sigma_z^2 = +1$, every element in $\mathcal{G}$ squares to ± 1. Also, σ_x, σ_y, and σ_z on the same qubit anticommute, while they commute on different qubits. Therefore, any two elements of $\mathcal{G}$ either commute or they anticommute. σ_x, σ_y, and σ_z are all

Hermitian, but of course $(iI)^\dagger = -iI$, so elements of $\mathcal{G}$ can be either Hermitian or anti-Hermitian. In either case, if $A \in \mathcal{G}$, $A^\dagger \in G$ also. Similarly, σ_x, σ_y, and σ_z are all unitary, so every element of $\mathcal{G}$ is unitary.

As before, if $M \in S$, $|\psi_i\rangle \in T$, and $\{M, E\} = 0$, then $ME|\psi_i\rangle = -E|\psi_i\rangle$, so

$$\langle\psi_i|E|\psi_j\rangle = \langle\psi_i|ME|\psi_j\rangle = -\langle\psi_i|E|\psi_j\rangle = 0. \tag{3.4}$$

Therefore the code satisfies (2.8) whenever $E = E_a^\dagger E_b = \pm E_a E_b$ anticommutes with M for some $M \in S$. In fact, in such a case it also satisfies (2.9), since $\langle\psi_i|E|\psi_i\rangle = \langle\psi_j|E|\psi_j\rangle = 0$. Therefore, if $E_a^\dagger E_b$ anticommutes with some element of S for all errors E_a and E_b in some set, the code will correct that set of errors.

Of course, strictly speaking, this is unlikely to occur. Generally, I will be an allowed error, and $E = I^\dagger I$ commutes with everything. However, S is a group, so $I \in S$. In general, if $E \in S$,

$$\langle\psi_i|E|\psi_j\rangle = \langle\psi_i|\psi_j\rangle = \delta_{ij}. \tag{3.5}$$

This will satisfy equation (2.10) also.

Now, there generally are many elements of $\mathcal{G}$ that commute with everything in S but are not actually in S. The set of elements in $\mathcal{G}$ that commute with all of S is defined as the centralizer $C(S)$ of S in $\mathcal{G}$. Because of the properties of S and $\mathcal{G}$, the centralizer is actually equal to the normalizer $N(S)$ of S in $\mathcal{G}$, which is defined as the set of elements of $\mathcal{G}$ that fix S under conjugation. To see this, note that for any $A \in \mathcal{G}$, $M \in S$,

$$A^\dagger M A = \pm A^\dagger A M = \pm M. \tag{3.6}$$

Since $-1 \notin S$, $A \in N(S)$ iff $A \in C(S)$, so $N(S) = C(S)$. Note that $S \subseteq N(S)$. In fact, S is a normal subgroup of $N(S)$. $N(S)$ contains $4 \cdot 2^{n+k}$ elements. The factor of four is for the overall phase factor. Since an overall phase has no effect on the physical quantum state, often, when considering $N(S)$, I will only really consider $N(S)$ without this global phase factor.

If $E \in N(S) - S$, then E rearranges elements of T but does not take them out of T: if $M \in S$ and $|\psi\rangle \in T$, then

$$ME|\psi\rangle = EM|\psi\rangle = E|\psi\rangle, \tag{3.7}$$

so $E|\psi\rangle \in T$ also. Since $E \notin S$, there is some state in T that is not fixed by E. Unless it differs from an element of S by an overall phase, E will therefore be undetectable by this code.

Putting these considerations together, we can say that a quantum code with stabilizer S will detect all errors E that are either in S or anticommute with some element of S. In other words, $E \in S \cup (\mathcal{G} - N(S))$. This code will correct any set of errors $\{E_i\}$ iff $E_a E_b \in S \cup (\mathcal{G} - N(S))$ $\forall E_a, E_b$ (note that $E_a^\dagger E_b$ commutes with $M \in \mathcal{G}$ iff $E_a E_b = \pm E_a^\dagger E_b$ does). For instance, the code will have distance d iff $N(S) - S$ contains no elements of weight less than d. If S has elements of weight less than d (except the identity), it is a degenerate

code; otherwise it is a nondegenerate code. For instance, the nine-qubit code is degenerate, since it has distance three and $\sigma_{z1}\sigma_{z2} \in S$. A nondegenerate stabilizer code satisfies

$$\langle\psi_i|E_a^\dagger E_b|\psi_j\rangle = \delta_{ab}\delta_{ij}. \tag{3.8}$$

By convention, an $[n, 0, d]$ code must be nondegenerate. When $E_aE_b \in S$, we say that the errors E_a and E_b are degenerate. We cannot distinguish between E_a and E_b, but there is no need to, since they have the same effect on the codewords.

It is sometimes useful to define the *error syndrome* for a stabilizer code. Let $f_M : \mathcal{G} \to \mathbf{Z}_2$,

$$f_M(E) = \begin{cases} 0 & \text{if } [M, E] = 0 \\ 1 & \text{if } \{M, E\} = 0 \end{cases} \tag{3.9}$$

and $f(E) = (f_{M_1}(E), \ldots, f_{M_{n-k}}(E))$, where $M_1, \ldots, M_{n-k}$ are the generators of S. Then $f(E)$ is some $(n-k)$-bit binary number which is 0 iff $E \in N(S)$. $f(E_a) = f(E_b)$ iff $f(E_aE_b) = 0$, so for a nondegenerate code, $f(E)$ is different for each correctable error E.

In order to perform the error-correction operation for a stabilizer code, all we need to do is measure the eigenvalue of each generator of the stabilizer. The eigenvalue of M_i will be $(-1)^{f_{M_i}(E)}$, so this process will give us the error syndrome. The error syndrome in turn tells us exactly what error occurred (for a nondegenerate code) or what set of degenerate errors occurred (for a degenerate code). The error will always be in $\mathcal{G}$ since the code uses that error basis, and every operator in $\mathcal{G}$ is unitary, and therefore invertible. Then we just apply the error operator (or one equivalent to it by multiplication by S) to fix the state. Note that even if the original error that occurred is a nontrivial linear combination of errors in $\mathcal{G}$, the process of syndrome measurement will project onto one of the basis errors. If the resulting error is not in the correctable set, we will end up in the wrong encoded state, but otherwise, we are in the correct state. In chapter 5, I describe a few ways of measuring the error syndrome that are tolerant of imperfect component gates.

Since the elements of $N(S)$ move codewords around within T, they have a natural interpretation as encoded operations on the codewords. Since S fixes T, actually only $N(S)/S$ will act on T nontrivially. If we pick a basis for T consisting of eigenvectors of n commuting elements of $N(S)$, we get an automorphism $N(S)/S \to \mathcal{G}_k$. $N(S)/S$ can therefore be generated by i (which we will by and large ignore) and $2k$ equivalence classes, which I will write $\overline{X}_i$ and $\overline{Z}_i$ $(i = 1 \ldots k)$, where $\overline{X}_i$ maps to σ_{xi} in $\mathcal{G}_k$ and $\overline{Z}_i$ maps to σ_{zi} in $\mathcal{G}_k$. They are encoded σ_x and σ_z operators for the code. If $k = 1$, I will write $\overline{X}_1 = \overline{X}$ and $\overline{Z}_1 = \overline{Z}$. The $\overline{X}$ and $\overline{Z}$ operators satisfy

$$[\overline{X}_i, \overline{X}_j] = 0 \tag{3.10}$$

$$[\overline{Z}_i, \overline{Z}_j] = 0 \tag{3.11}$$

$$[\overline{X}_i, \overline{Z}_j] = 0 \ (i \neq j) \tag{3.12}$$

$$\{\overline{X}_i, \overline{Z}_i\} = 0. \tag{3.13}$$

M_1	σ_x	σ_z	σ_z	σ_x	I
M_2	I	σ_x	σ_z	σ_z	σ_x
M_3	σ_x	I	σ_x	σ_z	σ_z
M_4	σ_z	σ_x	I	σ_x	σ_z
$\overline{X}$	σ_x	σ_x	σ_x	σ_x	σ_x
$\overline{Z}$	σ_z	σ_z	σ_z	σ_z	σ_z

Table 3.2: The stabilizer for the five-qubit code.

3.3 Some Examples

I shall now present a few short codes to use as examples. The first encodes one qubit in five qubits [17, 24] and is given in table 3.2. I have also included $\overline{X}$ and $\overline{Z}$, which, along with M_1 through M_4, generate $N(S)$. Note that this code is *cyclic* (i.e., the stabilizer and codewords are invariant under cyclic permutations of the qubits). It has distance three (for instance, $\sigma_{y1}\sigma_{z2}\sigma_{y3} \in N(S) - S$) and is nondegenerate. We can take the basis codewords for this code to be

$$|\overline{0}\rangle = \sum_{M \in S} M \ |00000\rangle \tag{3.14}$$

and

$$|\overline{1}\rangle = \overline{X}|\overline{0}\rangle. \tag{3.15}$$

That is,

$$\begin{aligned} |\overline{0}\rangle \;=\; & |00000\rangle + M_1|00000\rangle + M_2|00000\rangle + M_3|00000\rangle + M_4|00000\rangle \\ & + M_1M_2|00000\rangle + M_1M_3|00000\rangle + M_1M_4|00000\rangle \\ & + M_2M_3|00000\rangle + M_2M_4|00000\rangle + M_3M_4|00000\rangle \qquad (3.16) \\ & + M_1M_2M_3|00000\rangle + M_1M_2M_4|00000\rangle + M_1M_3M_4|00000\rangle \\ & + M_2M_3M_4|00000\rangle + M_1M_2M_3M_4|00000\rangle \\ = \; & |00000\rangle + |10010\rangle + |01001\rangle + |10100\rangle \\ & + |01010\rangle - |11011\rangle - |00110\rangle - |11000\rangle \\ & - |11101\rangle - |00011\rangle - |11110\rangle - |01111\rangle \qquad (3.17) \\ & - |10001\rangle - |01100\rangle - |10111\rangle + |00101\rangle, \end{aligned}$$

and

$$\begin{aligned} |\overline{1}\rangle \;=\; & \overline{X}|\overline{0}\rangle \\ = \; & |11111\rangle + |01101\rangle + |10110\rangle + |01011\rangle \\ & + |10101\rangle - |00100\rangle - |11001\rangle - |00111\rangle \\ & - |00010\rangle - |11100\rangle - |00001\rangle - |10000\rangle \qquad (3.18) \\ & - |01110\rangle - |10011\rangle - |01000\rangle + |11010\rangle. \end{aligned}$$

Since multiplying by an element of the stabilizer merely rearranges the sum $\sum M$, these two states are in T. When these are the encoded 0 and 1, $\overline{X}$ is the

M_1	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x
M_2	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z
M_3	I	σ_x	I	σ_x	σ_y	σ_z	σ_y	σ_z
M_4	I	σ_x	σ_z	σ_y	I	σ_x	σ_z	σ_y
M_5	I	σ_y	σ_x	σ_z	σ_x	σ_z	I	σ_y
$\overline{X}_1$	σ_x	σ_x	I	I	I	σ_z	I	σ_z
$\overline{X}_2$	σ_x	I	σ_x	σ_z	I	I	σ_z	I
$\overline{X}_3$	σ_x	I	I	σ_z	σ_x	σ_z	I	I
$\overline{Z}_1$	I	σ_z	I	σ_z	I	σ_z	I	σ_z
$\overline{Z}_2$	I	I	σ_z	σ_z	I	I	σ_z	σ_z
$\overline{Z}_3$	I	I	I	I	σ_z	σ_z	σ_z	σ_z

Table 3.3: The stabilizer for the eight-qubit code.

encoded bit flip operator σ_x and $\overline{Z}$ is the encoded σ_z. This code also has the property that every possible error syndrome is used by the single-qubit errors. It is therefore a *perfect* code. There are a number of other perfect codes [25, 26], which will be discussed in chapter 8.

A code encoding three qubits in eight qubits [21, 22, 27] appears in table 3.3. Again, M_1 through M_5 generate the stabilizer, and generate $N(S)$ with $\overline{X}_i$ and $\overline{Z}_i$. This is also a nondegenerate distance three code. The codewords are

$$|\overline{c_1 c_2 c_3}\rangle = \overline{X}_1^{c_1}\overline{X}_2^{c_2}\overline{X}_3^{c_3}\sum_{M\in S} M|00000000\rangle. \tag{3.19}$$

The operators $\overline{X}_i$ and $\overline{Z}_i$ are the encoded σ_x and σ_z on the ith encoded qubit. This code is one of an infinite family of codes [21, 28], which I present in chapter 8.

A particularly useful class of codes with simple stabilizers is the Calderbank-Shor-Steane (or *CSS*) class of codes [29, 30]. Suppose we have a classical code with parity check matrix P. We can make a quantum code to correct just σ_x errors using a stabilizer with elements corresponding to the rows of P, with a σ_z wherever P has a 1 and I's elsewhere. The error syndrome $f(E)$ for a product of σ_x errors E is then equal to the classical error syndrome for the same set of classical bit flip errors. Now add in stabilizer generators corresponding to the parity check matrix Q of a second classical code, only now with σ_x's instead of σ_z's. These generators will identify σ_z errors. Together, they can also identify σ_y errors, which will have a nontrivial error syndrome for both parts. In general, a code formed this way will correct as many σ_x errors as the code for P can correct, and as many σ_z errors as the code for Q can correct; a σ_y error counts as one of each.

We can only combine P and Q into a single stabilizer in the CSS form if the generators derived from the two codes commute. This will be true iff the rows of P and Q are orthogonal using the binary dot product. This means that the dual code of each code must be a subset of the other code. The minimum distance of the quantum code will be the minimum of the distances of P and

M_1	σ_x	σ_x	σ_x	σ_x	I	I	I
M_2	σ_x	σ_x	I	I	σ_x	σ_x	I
M_3	σ_x	I	σ_x	I	σ_x	I	σ_x
M_4	σ_z	σ_z	σ_z	σ_z	I	I	I
M_5	σ_z	σ_z	I	I	σ_z	σ_z	I
M_6	σ_z	I	σ_z	I	σ_z	I	σ_z
$\overline{X}$	I	I	I	I	σ_x	σ_x	σ_x
$\overline{Z}$	I	I	I	I	σ_z	σ_z	σ_z

Table 3.4: The seven-qubit CSS code.

Q. An example of a code of this sort is given in table 3.4. It is based on the classical $[7, 4, 3]$ Hamming code, which is self-dual. For this code, the codewords are

$$\begin{aligned} |\overline{0}\rangle &= |0000000\rangle + |1111000\rangle + |1100110\rangle + |1010101\rangle \\ &\quad + |0011110\rangle + |0101101\rangle + |0110011\rangle + |1001011\rangle \end{aligned} \tag{3.20}$$

and

$$\begin{aligned} |\overline{1}\rangle &= |0000111\rangle + |1111111\rangle + |1100001\rangle + |1010010\rangle \\ &\quad + |0011001\rangle + |0101010\rangle + |0110100\rangle + |1001100\rangle. \end{aligned} \tag{3.21}$$

The encoded $|0\rangle$ state is the superposition of the even codewords in the Hamming code and the encoded $|1\rangle$ state is the superposition of the odd codewords in the Hamming code. This behavior is characteristic of CSS codes; in general, the various quantum codewords are superpositions of the words in subcodes of one of the classical codes.

CSS codes are not as efficient as the most general quantum code, but they are easy to derive from known classical codes and their simple form often makes them ideal for other purposes. For instance, the seven-qubit code is particularly well suited for fault-tolerant computation (as I will discuss in chapter 5).

3.4 Alternate Languages for Stabilizers

There are number of possible ways of describing the stabilizer of a quantum code. They each have advantages and are useful in different circumstances. The description I have used so far uses the language of finite group theory and is particularly useful for making contact with the usual language of quantum mechanics. This is the form presented in [21].

We can instead write the stabilizer using binary vector spaces, as in [22], which emphasizes connections with the classical theory of error-correcting codes. To do this, we write the stabilizer as a pair of $(n-k) \times n$ binary matrices (or often one $(n-k) \times 2n$ matrix with a line separating the two halves). The rows correspond to the different generators of the stabilizer and the columns

correspond to different qubits. One matrix has a 1 whenever the generator has a σ_x or a σ_y in the appropriate place, the other has a 1 whenever the generator has a σ_y or σ_z. Overall phase factors get dropped. For instance, the five-qubit code in this form becomes

$$\left(\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right). \tag{3.22}$$

Other elements of $\mathcal{G}$ get converted to two n-dimensional vectors in the same way. We can convert back to the group theory formalism by writing down operators with a σ_x if the left vector or matrix has a 1, a σ_z if the right vector or matrix has a 1, and a σ_y if they are both 1. The generators formed this way will never have overall phase factors, although other elements of the group might. Multiplication of group elements corresponds to addition of the corresponding binary vectors.

In the binary formalism, the condition that two operators commute with each other becomes the condition that the following inner product is 0:

$$Q(a|b,c|d) = \sum_{i=1}^{n} (a_i d_i + b_i c_i) = 0, \tag{3.23}$$

using binary arithmetic as usual. a_i, b_i, c_i, and d_i are the ith components of the corresponding vectors. Therefore the condition that the stabilizer be Abelian converts to the condition that the stabilizer matrix $(A|B)$ satisfy

$$\sum_{l=1}^{n} (A_{il} B_{jl} + B_{il} A_{jl}) = 0. \tag{3.24}$$

We determine the vectors in $N(S)$ by evaluating the inner product (3.23) with the rows of $(A|B)$. To get a real code (with an even number of σ_y's), the code should also satisfy

$$\sum_{l=1}^{n} A_{il} B_{il} = 0. \tag{3.25}$$

Another formalism highlights connections with the classical theory of codes over the field GF(4) [26]. This is a field of characteristic two containing four elements, which can be written $\{0, 1, \omega, \omega^2\}$. Since the field has characteristic two,

$$1 + 1 = \omega + \omega = \omega^2 + \omega^2 = 0. \tag{3.26}$$

Also, $\omega^3 = 1$ and $1 + \omega = \omega^2$. We can rewrite the generators as an n-dimensional "vector" over GF(4) by substituting 1 for σ_x, ω for σ_z, and ω^2 for σ_y. The multiplicative structure of $\mathcal{G}$ becomes the additive structure of GF(4). I put vector in quotes because the code need not have the structure of a vector space over GF(4). If it does (that is, the stabilizer is closed under multiplication by

ω), the code is a *linear* code, which is essentially a classical code over GF(4). The most general quantum code is sometimes called an *additive* code, because the stabilizer is only closed under sums of its elements. In this formalism, the five-qubit code appears as

$$\begin{pmatrix} 1 & \omega & \omega & 1 & 0 \\ 0 & 1 & \omega & \omega & 1 \\ 1 & 0 & 1 & \omega & \omega \\ \omega & 1 & 0 & 1 & \omega \end{pmatrix}. \tag{3.27}$$

Note that the five-qubit code is a linear quantum code.

Again, there is an additional condition for a quantum code. Define the "trace" operator by Tr ω = Tr $\omega^2 = 1$, Tr 1 = Tr 0 = 0. Two operators in $\mathcal{G}$ commute iff their images, the vectors u and v over GF(4), satisfy

$$\text{Tr } u \cdot \overline{v} = \text{Tr} \left(\sum_{j=1}^{n} u_j \overline{v}_j \right) = 0, \tag{3.28}$$

where $\overline{v}_j$ is conjugation on the jth component of v, switching ω and ω^2, and leaving 0 and 1 alone.

3.5 Making New Codes From Old Codes

Using old codes to find new ones can simplify the task of finding codes, which can otherwise be quite a difficult problem. There are a number of simple modifications we can make to existing codes to produce new codes with different parameters [25, 26].

One trivial change is to perform a permutation of σ_x, σ_y, and σ_z on each qubit. This leaves the distance and size of the code the same, although it may be useful for codes that can correct different numbers of σ_x, σ_y, and σ_z errors. A slightly less trivial manipulation is to add a new qubit and a new generator which is σ_x for the new qubit. The other generators are tensored with the identity on the new qubit to form the generators of the new code. This makes an $[n, k, d]$ code (degenerate or nondegenerate) into an $[n + 1, k, d]$ degenerate code: Any operator acting as σ_y or σ_z on the new qubit will anticommute with the new generator, and any operator with the form $M \otimes \sigma_{x(n+1)}$ will be equivalent to the operator $M \otimes I$. Therefore, an operator must have at least weight d when restricted to the first n qubits to be in $N(S) - S$.

A less trivial manipulation is to remove the last qubit, converting an $[n, k, d]$ code into an $[n-1, k+1, d-1]$ code. To do this, we choose the $n-k$ generators of S so that M_1 ends σ_x, M_2 ends σ_z, and M_3 through M_{n-k} end I. We can always do this when $d > 1$ by picking the first two and then multiplying by combinations of them to make the others end appropriately.[1] Then the new

[1] If the code has been formed by adding a single σ_x (or σ_y or σ_z) generator, as above, we may not be able to do this for a given qubit, but there will always be at least one qubit for which we can.

M_1'	σ_x	σ_z	σ_z	σ_x
M_2'	σ_y	σ_x	σ_x	σ_y
$\overline{X}_1$	σ_x	σ_x	σ_x	σ_x
$\overline{X}_2$	σ_x	I	σ_x	σ_z
$\overline{Z}_1$	σ_y	σ_z	σ_y	I
$\overline{Z}_2$	I	σ_x	σ_z	σ_z

Table 3.5: A $[4, 2, 2]$ code derived from the $[5, 1, 3]$ code.

code has a stabilizer formed from the last $n - k - 2$ generators, dropping M_1 and M_2. Suppose we have an operator A on the first $n - 1$ qubits of weight w that commutes with M_3 through M_{n-k}. There are four possibilities, all of which lead to an operator of weight at most $w + 1$ that commutes with the original stabilizer:

1. A commutes with both M_1 and M_2.
2. A commutes with M_1, but not M_2. Then $A \otimes \sigma_{xn}$ commutes with M_1 and M_2.
3. A commutes with M_2, but not M_1. Then $A \otimes \sigma_{zn}$ commutes with M_1 and M_2.
4. A anticommutes with both M_1 and M_2. Then $A \otimes \sigma_{yn}$ commutes with M_1 and M_2.

Since the original code had distance d, w must be at least $d-1$, which is therefore the distance of the new code. The stabilizer has $n-k-2$ generators, so the code encodes $(n-1)-(n-k-2) = k+1$ qubits. The new $\overline{X}$ and $\overline{Z}$ operators are M_1 and M_2 (in either order), restricted to the first $n-1$ qubits. An example of this construction is to remove the last qubit from the $[5, 1, 3]$ code of figure 4.2 to produce a $[4, 2, 2]$ code: the generators of the new code are M_1 and M_3M_4, both without the last qubit. The new stabilizer is given in figure 3.5. Note that the $\overline{Z}_1$ operator is equal to $M_3\overline{Z}$ for the five-qubit code. I have multiplied by M_3 so that $\overline{Z}_1$ anticommutes with $\overline{X}_1$.

Another way to make new codes is by *pasting* together old codes. Suppose we have four stabilizers R_1, R_2, S_1, and S_2, with $R_1 \subset S_1$ and $R_2 \subset S_2$. Let R_1 define an $[n_1, l_1, c_1]$ code, R_2 be an $[n_2, l_2, c_2]$ code, S_1 be an $[n_1, k_1, d_1]$ code, and S_2 be an $[n_2, k_2, d_2]$ code. Then $k_i < l_i$ and $c_i \leq d_i$. We require $l_1 - k_1 = l_2 - k_2$ and for S_1 and S_2 to be nondegenerate.[2] Let generators of R_1 be $\{M_1, \ldots, M_{n_1-l_1}\}$, the generators of S_1 be $\{M_1, \ldots, M_{n_1-k_1}\}$, the generators of R_2 be $\{N_1, \ldots, N_{n_2-l_2}\}$, and the generators of S_2 be $\{N_1, \ldots, N_{n_2-k_2}\}$. We form a new stabilizer S on $n_1 + n_2$ qubits generated by

$$\begin{aligned}\{&M_1 \otimes I, \ldots, M_{n_1-l_1} \otimes I, I \otimes N_1, \ldots, I \otimes N_{n_2-l_2},\\ &M_{n_1-l_1+1} \otimes N_{n_2-l_2+1}, \ldots, M_{n_1-k_1} \otimes N_{n_2-k_2}\}.\end{aligned} \tag{3.29}$$

[2] We can actually allow S_1 and S_2 to be degenerate, as long as all the degenerate operators are confined to R_1 and R_2

M_1	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	I	I	I	I	I
M_2	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	I	I	I	I	I
M_3	I	I	I	I	I	I	I	I	σ_x	σ_z	σ_z	σ_x	I
M_4	I	σ_x	I	σ_x	σ_y	σ_z	σ_y	σ_z	I	σ_x	σ_z	σ_z	σ_x
M_5	I	σ_x	σ_z	σ_y	I	σ_x	σ_z	σ_y	σ_x	I	σ_x	σ_z	σ_z
M_6	I	σ_y	σ_x	σ_z	σ_x	σ_z	I	σ_y	σ_z	σ_x	I	σ_x	σ_z

Table 3.6: The thirteen-qubit code formed by pasting together the five- and eight-qubit codes.

The code has $(n_1 - l_1) + (n_2 - l_2) + (l_i - k_i)$ generators, and therefore encodes $l_1 + k_2 = l_2 + k_1$ qubits. For instance, if S_1 is the eight-qubit code and S_2 is the five-qubit code, with R_1 generated by $\sigma_x\sigma_x\sigma_x\sigma_x\sigma_x\sigma_x\sigma_x\sigma_x$ and $\sigma_z\sigma_z\sigma_z\sigma_z\sigma_z\sigma_z\sigma_z\sigma_z$ and R_2 generated by $\sigma_x\sigma_z\sigma_z\sigma_x I$, we can make the $[13, 7, 3]$ code given in table 3.6.

In general, the distance of the new code will be $\min\{d_1, d_2, c_1 + c_2\}$. This is because an operator acting on just the first n_1 qubits can only commute with S if it commutes with S_1, an operator acting on the last n_2 qubits can only commute with S if it commutes with S_2, and an operator acting on both parts must commute with both $R_1 \otimes I$ and $I \otimes R_2$.

Another very useful way of producing new codes is to *concatenate* two codes to produce a code of greater total distance. Suppose we have an $[n_1, k, d_1]$ code (stabilizer S_1) and we encode each of its n_1 qubits again using an $[n_2, 1, d_2]$ code (stabilizer S_2). The result is an $[n_1 n_2, k, d_1 d_2]$ code. Its stabilizer S is n_1 copies of S_2, acting on the physical qubits in blocks of size n_2, plus an additional $n_1 - k$ generators corresponding to the generators of S_1. However, these generators are encoded to act on the second code. That is, a σ_x acting on the first code must be replaced by an $\overline{X}$ for the second code. For instance, the code resulting from concatenating the five-qubit code with itself has the stabilizer given in table 3.7. The concatenated code has distance $d_1 d_2$ because operators in $N(S) - S$ must have distance at least d_2 on at least d_1 blocks of n_2 qubits, so have weight at least $d_1 d_2$. Note that it is not strictly necessary to use the same code to encode each qubit of S_1.

There are two possible ways to concatenate when S_2 encodes multiple qubits. Suppose S_1 is an $[n_1, k_1, d_1]$ code and S_2 is an $[n_2, k_2, d_2]$ code. Further, suppose n_1 is a multiple of k_2. Then we can encode blocks of S_1 of size k_2 using S_2. This will result in a code using $n_1 n_2 / k_2$ qubits to encode k_1 qubits. It still takes an operator of distance at least d_2 to cause an error on an n_2-qubit block, but such an error can cause up to k_2 errors on S_1, so the resulting code need only have distance $\lceil d_1/k_2 \rceil d_2$. However, the k_2 errors that result are not a general set of k_2 errors, so the code may actually be better. Suppose S_1 has distance d_1' ($d_1' \geq \lceil d_1/k_2 \rceil$) for blocks of k_2 errors, i.e., d_1' such blocks must have errors before the code fails. Then the concatenated code has distance $d_1' d_2$.

Another way to concatenate codes encoding multiple qubits is to add additional blocks of S_1 to fill the spaces in S_2. That is, we actually encode k_2 copies

M_1	σ_x	σ_z	σ_z	σ_x	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
M_2	I	σ_x	σ_z	σ_z	σ_x	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
M_3	σ_x	I	σ_x	σ_z	σ_z	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
M_4	σ_z	σ_x	I	σ_x	σ_z	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
M_5	I	I	I	I	I	σ_x	σ_z	σ_z	σ_x	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
M_6	I	I	I	I	I	I	σ_x	σ_z	σ_z	σ_x	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
M_7	I	I	I	I	I	σ_x	I	σ_x	σ_z	σ_z	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
M_8	I	I	I	I	I	σ_z	σ_x	I	σ_x	σ_z	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
M_9	I	I	I	I	I	I	I	I	I	I	σ_x	σ_z	σ_z	σ_x	I	I	I	I	I	I	I	I	I	I	I
M_{10}	I	I	I	I	I	I	I	I	I	I	I	σ_x	σ_z	σ_z	σ_x	I	I	I	I	I	I	I	I	I	I
M_{11}	I	I	I	I	I	I	I	I	I	I	σ_x	I	σ_x	σ_z	σ_z	I	I	I	I	I	I	I	I	I	I
M_{12}	I	I	I	I	I	I	I	I	I	I	σ_z	σ_x	I	σ_x	σ_z	I	I	I	I	I	I	I	I	I	I
M_{13}	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	σ_x	σ_z	σ_z	σ_x	I	I	I	I	I	I
M_{14}	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	σ_x	σ_z	σ_z	σ_x	I	I	I	I	I
M_{15}	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	σ_x	I	σ_x	σ_z	σ_z	I	I	I	I	I
M_{16}	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	σ_z	σ_x	I	σ_x	σ_z	I	I	I	I	I
M_{17}	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	σ_x	σ_z	σ_z	σ_x	I
M_{18}	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	σ_x	σ_z	σ_z	σ_x
M_{19}	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	σ_x	I	σ_x	σ_z	σ_z
M_{20}	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	σ_z	σ_x	I	σ_x	σ_z
M_{21}	σ_x	σ_x	σ_x	σ_x	σ_x	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_x	σ_x	σ_x	σ_x	σ_x	I	I	I	I	I
M_{22}	I	I	I	I	I	σ_x	σ_x	σ_x	σ_x	σ_x	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_x	σ_x	σ_x	σ_x	σ_x
M_{23}	σ_x	σ_x	σ_x	σ_x	σ_x	I	I	I	I	I	σ_x	σ_x	σ_x	σ_x	σ_x	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z
M_{24}	σ_z	σ_z	σ_z	σ_z	σ_z	σ_x	σ_x	σ_x	σ_x	σ_x	I	I	I	I	I	σ_x	σ_x	σ_x	σ_x	σ_x	σ_z	σ_z	σ_z	σ_z	σ_z

Table 3.7: Result of concatenating the five-qubit code with itself.

of S_1, encoding the ith qubit of each copy in the same S_2 block. This produces an $[n_1n_2, k_1k_2, d_1d_2]$ code, since any failure of an S_2 block only produces one error in each S_1 block.

3.6 Higher Dimensional States

So far, we have only considered systems for which the Hilbert space is the tensor product of two-state systems. However, it may turn out that a good physical implementation of quantum computation uses three- or four-level atoms, or spin-one particles, or some other system where it makes more sense to consider it as the tensor product of d-dimensional systems, where $d > 2$. I will call the fundamental unit of such a system a *qudit*. In such a case, we will want to consider error correcting codes where a single qudit error can occur with reasonable probability. For these systems, the stabilizer code formalism needs to be modified to deal with the extra dimensions.

Fundamental to the success of the stabilizer formalism was the use of the Pauli spin matrix basis for possible errors. The algebraic properties of this basis allowed a straightforward characterization of errors depending on whether they commuted or anticommuted with elements of an Abelian group. Knill [31] has codified the properties necessary for this construction to generalize to d-dimensional spaces. Suppose we have a set of d^2 unitary operators $E_1, \ldots, E_{n^2}$ (including the identity) acting on a single qudit such that the E_i's form a basis for all possible $d \times d$ complex matrices. If $E_iE_j = w_{ij}E_{i*j}$ for all i, j (where $*$ is some binary group operation), then the E_i's are said to form a *nice* error basis. The values w_{ij} will then have modulus one. Given a nice error basis, we form the group $\mathcal{G}_n$ for this basis as the tensor product of n copies of the error basis, with possible overall phases generated by the w_{ij}'s. Then an Abelian subgroup S of $\mathcal{G}_n$ that does not contain any nontrivial phase times the identity will have a nontrivial set T of states in the Hilbert space in the $+1$ eigenspace of every operator in S. The code T can detect any error E for which $EM = cME$ for some $M \in \mathcal{G}_n$ and some $c \neq 1$.

One interesting complication of codes over d-dimensional spaces is that when S has $n - k$ generators, T need not encode k qudits. This can only occur when d is composite and the order of a generator of S is a nontrivial factor of d. It is still true that if S has r elements, then T will be (d^n/r)-dimensional. If all the generators of S have order d, T does encode k qudits.

One particularly convenient error basis for any d is generated by D_ω and C_n, where $(D_\omega)_{ij} = \delta_{ij}\omega^i$ and $(C_n)_{ij} = \delta_{j,(i+1 \bmod n)}$. ω is a primitive nth root of unity. For $d = 2$, this just reduces to the usual Pauli basis, since $C_2 = \sigma_x$ and $D_{-1} = \sigma_z$. For higher d, D_ω maps $|i\rangle \to \omega^i|i\rangle$ and C_n adds one modulo n. This is a nice error basis, with

$$C_nD_\omega = \omega D_\omega C_n. \tag{3.30}$$

The elements of the basis can be written $C_n^a D_\omega^b$, and

$$\left(C_n^a D_\omega^b\right)\left(C_n^c D_\omega^d\right) = \omega^{ad-bc}\left(C_n^c D_\omega^d\right)\left(C_n^a D_\omega^b\right). \tag{3.31}$$

Codes for higher-dimensional systems have not been as extensively studied as those for two-dimensional systems, but some constructions are given in [31, 32, 33, 34, 35].

Chapter 4

Encoding and Decoding Stabilizer Codes

4.1 Standard Form for a Stabilizer Code

To see how to encode a general stabilizer code [36], it is helpful to describe the code in the language of binary vector spaces (see section 3.4). Note that the specific choice of generators is not at all unique. We can always replace a generator M_i with $M_i M_j$ for some other generator M_j. The corresponding effect on the binary matrices is to add row j to row i in both matrices. For simplicity, it is also helpful to rearrange qubits in the code. This has the effect of rearranging the corresponding columns in both matrices. Combining these two operations, we can perform Gaussian elimination on the first matrix, putting the code in this form:

$$\begin{array}{r} \\ r\{ \\ n-k-r\{ \end{array} \left(\begin{array}{cc|cc} \overbrace{I}^{r} & \overbrace{A}^{n-r} & \overbrace{B}^{r} & \overbrace{C}^{n-r} \\ 0 & 0 & D & E \end{array} \right). \tag{4.1}$$

Here, r is the rank of the σ_x portion of the stabilizer generator matrix.

Then we perform another Gaussian elimination on E to get

$$\begin{array}{r} \\ r\{ \\ n-k-r-s\{ \\ s\{ \end{array} \left(\begin{array}{ccc|ccc} \overbrace{I}^{r} & \overbrace{A_1}^{n-k-r-s} & \overbrace{A_2}^{k+s} & \overbrace{B}^{r} & \overbrace{C_1}^{n-k-r-s} & \overbrace{C_2}^{k+s} \\ 0 & 0 & 0 & D_1 & I & E_2 \\ 0 & 0 & 0 & D_2 & 0 & 0 \end{array} \right). \tag{4.2}$$

The rank of E is $n-k-r-s$. However, the first r generators will not commute with the last s generators unless $D_2 = 0$, which really implies that $s = 0$. Thus we can always put the code into the standard form

$$\begin{array}{r} \\ r\{ \\ n-k-r\{ \end{array} \left(\begin{array}{ccc|ccc} \overbrace{I}^{r} & \overbrace{A_1}^{n-k-r} & \overbrace{A_2}^{k} & \overbrace{B}^{r} & \overbrace{C_1}^{n-k-r} & \overbrace{C_2}^{k} \\ 0 & 0 & 0 & D & I & E \end{array} \right). \tag{4.3}$$

For instance, the standard form for the five-qubit code of table 3.2 is

$$\left(\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 \end{array}\right). \tag{4.4}$$

Suppose we have an $\overline{X}$ operator which in this language is written $(u|v) = (u_1u_2u_3|v_1v_2v_3)$, where u_1 and v_1 are r-dimensional vectors, u_2 and v_2 are $(n-k-r)$-dimensional vectors, and u_3 and v_3 are k-dimensional vectors. However, elements of $N(S)$ are equivalent up to multiplication by elements of S. Therefore, we can also perform eliminations on $\overline{X}$ to force $u_1 = 0$ and $v_2 = 0$. Then, because $\overline{X}$ is in $N(S)$, we must satisfy (3.23), so

$$\begin{aligned} \left(\begin{array}{cccccc} I & A_1 & A_2 & B & C_1 & C_2 \\ 0 & 0 & 0 & D & I & E \end{array}\right) \left(\begin{array}{c} v_1^T \\ 0 \\ v_3^T \\ 0 \\ u_2^T \\ u_3^T \end{array}\right) &= \left(\begin{array}{c} v_1^T + A_2v_3^T + C_1u_2^T + C_2u_3^T \\ u_2^T + Eu_3^T \end{array}\right) \\ &= \left(\begin{array}{c} 0 \\ 0 \end{array}\right). \end{aligned} \tag{4.5}$$

Suppose we want to choose a complete set of k $\overline{X}$ operators. We can combine their vectors into two $k \times n$ matrices $(0U_2U_3|V_10V_3)$. We want them to commute with each other, so $U_3V_3^T + V_3U_3^T = 0$. Suppose we pick $U_3 = I$. Then we can take $V_3 = 0$, and by equation (4.5), $U_2 = E^T$ and $V_1 = E^TC_1^T + C_2^T$. The rest of the construction will assume that this choice has actually been made. Another choice of U_3 and V_3 will require us to perform some operation on the unencoded data to compensate. For the five-qubit code, the standard form of the $\overline{X}$ generator would be (00001|10010). We can see that this is equivalent (mod S) to the $\overline{X}$ given in table 3.2.

We can also pick a complete set of k $\overline{Z}$ operators, which act on the code as encoded σ_z operators. They are uniquely defined (up to multiplication by S, as usual) given the $\overline{X}$ operators. $\overline{Z}_i$ is an operator that commutes with $M \in S$, commutes with $\overline{X}_j$ for $i \neq j$, and anticommutes with $\overline{X}_i$. We can bring it into the standard form $(0U_2'U_3'|V_1'0V_3')$. Then

$$U_3'V_3^T + V_3'U_3^T = I. \tag{4.6}$$

When $U_3 = I$ and $V_3 = 0$, $V_3' = I$. Since equation (4.5) holds for the $\overline{Z}$ operators too, $U_2' = U_3' = 0$ and $V_1' = A_2^T$. For instance, for the five-qubit code, the standard form of the $\overline{Z}$ generator is (00000|11111), which is exactly what is given in table 3.2.

4.2 Network for Encoding

Given a stabilizer in standard form along with the $\overline{X}$ operators in standard form, it is straightforward to produce a network to encode the corresponding code. The operation of encoding a stabilizer code can be written as

$$|c_1 \dots c_k\rangle \rightarrow \left(\sum_{M\in S} M\right) \overline{X}_1^{c_1} \cdots \overline{X}_k^{c_k} |0 \dots 0\rangle \tag{4.7}$$

$$= (I + M_1) \cdots (I + M_{n-k}) \overline{X}_1^{c_1} \cdots \overline{X}_k^{c_k} |0 \dots 0\rangle, \tag{4.8}$$

where M_1 through M_{n-k} generate the stabilizer, and $\overline{X}_1$ through $\overline{X}_k$ are the encoded σ_x operators for the k encoded qubits. This is true because, in general, for any $N \in S$,

$$N \left(\sum_{M\in S} M\right) |\psi\rangle = \left(\sum_{M\in S} NM\right) |\psi\rangle = \left(\sum_{M'\in S} M'\right) |\psi\rangle, \tag{4.9}$$

so $\sum M|\psi\rangle$ is in the coding space T for any state $|\psi\rangle$. If we define the encoded 0 as

$$|\overline{0}\rangle = \sum_{M\in S} M|\overbrace{0 \dots 0}^{n}\rangle, \tag{4.10}$$

then by the definition of the $\overline{X}$'s, we should encode

$$|c_1 \dots c_k\rangle \rightarrow \overline{X}_1^{c_1} \cdots \overline{X}_k^{c_k} \left(\sum_{M\in S} M\right) |0 \dots 0\rangle. \tag{4.11}$$

Since $\overline{X}_i$ commutes with $M \in S$, this is just (4.7). Naturally, to encode this, we only need to worry about encoding the basis states $|c_1 \dots c_k\rangle$.

The standard form of $\overline{X}_i$ has the form $Z^{(r)} X^{(n-k-r)} \sigma_{x(n-k+i)}$ ($Z^{(r)}$ is the product of σ_z's on the first r qubits and $X^{(n-k-r)}$ is the product of σ_x's on the next $n-k-r$ qubits). Suppose we put the kth input qubit $|c_k\rangle$ in the nth spot, following $n-1$ 0s. The state $\overline{X}_k^{c_k}|0 \dots 0\rangle$ therefore has a 1 for the nth qubit iff $|c_k\rangle = |1\rangle$. This means we can get the state $\overline{X}_k^{c_k}|0 \dots 0\rangle$ by applying $\overline{X}_k$ (without the final σ_{xn}) to the input state conditioned on the nth qubit. For instance, for the five-qubit code, $\overline{X} = Z \otimes I \otimes I \otimes Z \otimes X$. The corresponding operation is illustrated in figure 4.1. In this case $r = n - k = 4$, so there are no bit flips, only controlled σ_z's.

In the more general case, we also need to apply $\overline{X}_1$ through $\overline{X}_{k-1}$, depending on c_1 through c_{k-1}. Since the form of the $\overline{X}$'s ensures that each only operates on a single one of the last k qubits, we can substitute $|c_i\rangle$ for the $(n-k+i)$th qubit and apply $\overline{X}_i$ conditioned on it, as with $|c_k\rangle$. This produces the state $\overline{X}_1^{c_1} \cdots \overline{X}_k^{c_k} |0 \dots 0\rangle$.

Further, note that the $\overline{X}$ operators only act as σ_z on the first r qubits and as σ_x on the next $n-k-r$ qubits. Since σ_z acts trivially on $|0\rangle$, we can just

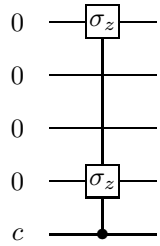

Figure 4.1: Creating the state $\overline{X}|00000\rangle$ for the five-qubit code.

ignore that part of the $\overline{X}$'s when implementing this part of the encoder, leaving just the controlled NOTs. The first r qubits automatically remain in the state $|0\rangle$ after this step of encoding. This means that for the five-qubit code, this step of encoding is actually trivial, with no operations. In general, this step is only necessary if $r < n - k$.

For the next step of the encoding, we note that the standard form of the first r generators only applies a single bit flip in the first r qubits. This means that when we apply $I + M_i$, the resulting state will be the sum of a state with $|0\rangle$ for the ith qubit and a state with $|1\rangle$ for the ith qubit. We therefore apply the Hadamard transform

$$R = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{4.12}$$

to the first r qubits, putting each in the state $|0\rangle + |1\rangle$. Then we apply M_i (for $i = 1, \ldots, r$) conditioned on qubit i (ignoring the factor of σ_{xi}). While these operators may perform phase operations on the first r qubits, they do not flip them, so there is no risk of one operation confusing the performance of another one. The one possible complication is when M_i has a factor of σ_{zi}. In this case, σ_{zi} only introduces a minus sign if the qubit is $|1\rangle$ anyway, so we do not need to condition it on anything. Just performing σ_{zi} after the Hadamard transform is sufficient. For the five-qubit code, the full network for encoding is given in figure 4.2.

For more general codes, $r < n-k$, and there are $n-k-r$ generators that are formed just of the tensor product of σ_z's. However, we do not need to consider such generators to encode. Let M be such a generator. Since M commutes with all the other generators and every $\overline{X}$, we can commute $I + M$ through until it acts directly on $|0 \ldots 0\rangle$. However, σ_z acts trivially on $|0\rangle$, so $I+M$ fixes $|0 \ldots 0\rangle$, and in equation (4.8), we can skip any M_i that is the tensor product of σ_z's. The effect of these operators is seen just in the form of the $\overline{X}$ operators, which must commute with them.

Applying each of the $\overline{X}$ operators requires up to $n - k - r$ two-qubit operations. Each of the first r qubits must be prepared with a Hadamard transform and possibly a σ_z, which we can combine with the Hadamard transform. Then

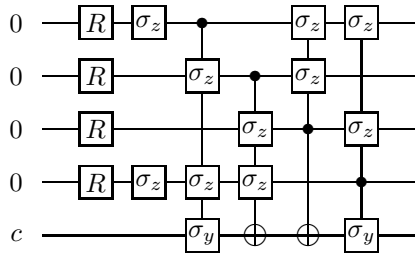

Figure 4.2: Network for encoding the five-qubit code.

applying each of the first r generators requires up to $n-1$ two-qubit operations. The whole encoder therefore requires up to r one-qubit operations and at most

$$k(n-k-r)+r(n-1) \leq (k+r)(n-k) \leq n(n-k) \tag{4.13}$$

two-qubit operations.

4.3 Other Methods of Encoding and Decoding

We can decode a code by performing the above network in reverse. In order to do this, we should first perform an error correction cycle, since the network will not necessarily work properly on an encoded state. Note that in principle we can build a decoder that corrects while decoding. We can form a basis for the Hilbert space from the states $A|\psi_i\rangle$, where $A \in \mathcal{G}$ and $|\psi_i\rangle$ is a basis state for the coding space T. The combined corrector/decoder would map $A|\psi_i\rangle$ to $|i\rangle \otimes |f(A)\rangle$, where $f(A)$ is the error syndrome for A. If A is not a correctable error, $|i\rangle$ will not necessarily be the state encoded by $|\psi_i\rangle$, but if A is correctable, it will be. It is not usually worthwhile using a quantum network that does this, since the error correction process is usually dealt with more easily using classical measurements. However, some proposed implementations of quantum computation cannot be used to measure a single system [9], so this sort of network would be necessary. The decoding method presented in [37] can easily be adapted to produce networks that simultaneously correct and decode.

One good reason not to decode by running the encoder backwards is that most of the work in the encoder went into producing the encoded 0. There is no actual information in that state, so we might be able to save time decoding if we could remove the information without dealing with the structure of the encoded 0. We can do this by using the $\overline{X}$ and $\overline{Z}$ operators. If we want to measure the ith encoded qubit without decoding, we can do this by measuring the eigenvalue of $\overline{Z}_i$. If the eigenvalue is $+1$, the ith encoded qubit is $|0\rangle$; if it is -1, the ith encoded qubit is $|1\rangle$. In standard form, $\overline{Z}_i$ is the tensor product of σ_z's. That means it will have eigenvalue $(-1)^P$, where P is the parity of the

qubits acted on by $\overline{Z}_i$. Therefore, if we apply a controlled-NOT from each of these qubits to an ancilla qubit, we have performed a controlled-NOT from the ith encoded qubit to the ancilla — we will flip the ancilla iff the ith encoded qubit is $|1\rangle$.

If the original state of the code is $|\overline{0}\rangle|\psi\rangle + |\overline{1}\rangle|\phi\rangle$ (with the first ket representing the ith *logical* qubit) and the ancilla begins in the state $|0\rangle$, after applying this CNOT operation, we have

$$|\overline{0}\rangle|\psi\rangle|0\rangle + |\overline{1}\rangle|\phi\rangle|1\rangle. \tag{4.14}$$

Now we apply $\overline{X}_i$ conditioned on the ancilla qubit. This will flip the ith encoded qubit iff the ancilla is $|1\rangle$. This produces the state

$$|\overline{0}\rangle|\psi\rangle|0\rangle + |\overline{0}\rangle|\phi\rangle|1\rangle = |\overline{0}\rangle\left(|\psi\rangle|0\rangle + |\phi\rangle|1\rangle\right). \tag{4.15}$$

The ith encoded qubit has been set to 0 and the ancilla holds the state that the ith encoded qubit used to hold. The rest of the code has been left undisturbed. We can repeat this operation with each of the encoded qubits, transferring them to k ancilla qubits. Each such operation requires at most $2(n-k+1)$ two-qubit operations (since $\overline{Z}$ requires at most $r+1$ operations and $\overline{X}$ could require $n-k+1$ operations). Therefore, the full decoder uses at most $2k(n-k+1)$ operations, which is often less than is required to encode. At the end of the decoding, the original n qubits holding the code are left in the encoded 0 state.

We can run this process backwards to encode, but we need an encoded 0 state to begin with. This could be a residue from an earlier decoding operation, or could be produced separately. One way to produce it would be to use the network of section 4.2, using $|0\ldots0\rangle$ as the input data. Alternately, we could produce it by performing an error correction cycle on a set of n $|0\rangle$'s for the stabilizer generated by $M_1, \ldots M_{n-k}, \overline{Z}_1, \ldots, \overline{Z}_k$. This stabilizer has n generators, so there is only one joint $+1$ eigenvector, which is just the encoded 0 for the original code.

Chapter 5

Fault-Tolerant Computation

5.1 Encoded Computation and Fault-Tolerance

I have shown how to encode qubits in blocks to protect them from individual errors. This, by itself, is useful for transmitting quantum data down a noisy communications line, for instance — we can encode the data using the code, send it, correct the errors, and decode it. Then we can process the data normally. However, the framework so far is insufficient for performing computations on a realistic quantum computer. If we need to decode the data in order to perform quantum gates on it, it is vulnerable to noise during the time it is decoded. Even if we know how to perform gates on the data while it is still encoded, we must be careful to make sure that a single error does not cause us to accidentally perform the wrong computation.

For instance, suppose a single qubit has been flipped and we apply a controlled-NOT from it to another qubit. Then the second qubit will flip exactly when it is supposed to stay the same. In consequence, now both the first and the second qubits have bit flip errors. If both qubits are part of the same block, we now have two errors in the block instead of one. Before very much of this occurs, we will have too many errors in the block to correct. If we correct errors often enough, we can salvage the situation [34], but in the process we lose a lot of the power of the error-correcting code. Therefore, I will define a fault-tolerant operation as one for which a single error introduces at most one error per block of the code. In a large computer, we have many encoded blocks of data, and a given operation may introduce one error in a number of them. However, each block retains its ability to correct that single error.

In the example above, an error propagated forward from the control qubit to the target qubit of the CNOT. In a quantum computer, errors can also propagate backwards. For instance, suppose we have the state

$$(\alpha|0\rangle + \beta|1\rangle)(|0\rangle \pm |1\rangle) \tag{5.1}$$

and perform a CNOT from the first qubit to the second. The resulting state is

$$\alpha|0\rangle(|0\rangle \pm |1\rangle) + \beta|1\rangle(\pm 1)(|0\rangle \pm |1\rangle) = (\alpha|0\rangle \pm \beta|1\rangle)(|0\rangle \pm |1\rangle). \tag{5.2}$$

Initially flipping the sign on the second qubit will result in a sign flip on the first qubit after the CNOT. In a CNOT, amplitude (bit flip) errors propagate forwards, and phase errors propagate backwards.

This means that not only must we make sure not to perform operations from one qubit to another within a block, we must also be sure not to perform multiple CNOTs from a block onto the same target qubit, even if it is a disposable ancilla qubit. Otherwise, a single phase error in the ancilla qubit can produce multiple errors within a block. Operations for which each qubit in a block only interacts with the corresponding qubit, either in another block or in a specialized ancilla, will be called *transversal* operations. Any transversal operation is automatically fault-tolerant, although there are some fault-tolerant operations which are not transversal.

5.2 Measurement and Error Correction

Suppose we want to measure the operator $\sigma_{z1}\sigma_{z2}$, as with Shor's nine-qubit code. The eigenvalue is $+1$ if both qubits are the same and -1 if they are different. One natural way to do this is perform a CNOT from both qubits to a third ancilla qubit, initially in the state $|0\rangle$. If both qubits are $|0\rangle$, the ancilla is left alone, and if both are $|1\rangle$, the ancilla gets flipped twice, returning to the state $|0\rangle$. If only one of the two qubits is $|1\rangle$, the ancilla only flips once, ending up in the state $|1\rangle$. Measuring the ancilla will then tell us the eigenvalue of $\sigma_{z1}\sigma_{z2}$.

However, this procedure is not a transversal operation. Both qubits interact with the same ancilla qubit, and a single phase error on the ancilla qubit could produce phase errors in both data qubits, producing two errors in the block (actually, this particular example does not have this problem, since a phase error on the ancilla qubit is meaningless until after it has interacted with the first data qubit; but if we were measuring $\sigma_{z1}\sigma_{z2}\sigma_{z3}$ instead, the problem would be a real one). One possible solution to the problem is to use two ancilla qubits, both initially $|0\rangle$, instead of one. Then we perform CNOTs from the first data qubit to the first ancilla qubit and from the second data qubit to the second ancilla qubit. Then we measure the ancilla qubits and determine their parity. This will again tell us the eigenvalue of $\sigma_{z1}\sigma_{z2}$, and we do not run the risk of introducing two phase errors into the data.

However, we have instead done something worse. By measuring both ancilla qubits, we have, in effect, measured the original data qubits, which destroys any superposition of the $+1$-eigenstates of $\sigma_{z1}\sigma_{z2}$. To make this work, we need to be able to measure the ancilla without finding out anything about the data. Since we are only interested in the parity of the data qubits, we could have just as well started the ancilla in the state $|11\rangle$ as $|00\rangle$. If both or neither ancilla qubits are flipped, the parity is still even, and if only one is flipped, the parity

is odd, as it should be. However, measuring the ancilla still tells us what states the data qubits were in. The state of a data qubit is equal to the reverse of the measured state of the corresponding ancilla qubit.

This means if we start the ancilla in the superposition $|00\rangle+|11\rangle$ and perform CNOTs from the data qubits to the ancilla qubits, measuring the ancilla will again tell us the parity of the data qubits. However, we do not know whether the state we measure originally corresponded to the ancilla state $|00\rangle$ or $|11\rangle$, which means we cannot deduce the state of the data. The two ancilla states correspond to the two possible states of the data qubits with the same parity. This means that measuring the ancilla does not destroy a superposition of these two states of the data. This is what we desired.

Because we interact each data qubit with a separate ancilla qubit, a single phase error in the ancilla will only produce a single phase error in the data. Of course, if a single qubit in the ancilla flips so we start in the state $|01\rangle + |10\rangle$, we will measure the wrong parity. We can circumvent this problem by simply preparing multiple ancillas in the same state, performing the CNOTs to each of them, and measuring each. If we prepare three such ancillas and determine the parity as the majority result, the answer will be correct unless two errors have occurred. If the chance of a single error is ϵ, the chance of getting two errors in the data or getting the wrong measurement result is $O(\epsilon^2)$.

We can use this trick on products of more than two σ_z operators [38] by preparing the ancilla in a state which is the sum of all even parity states. Such a state can be made by preparing a "cat" state $|0\ldots0\rangle + |1\ldots1\rangle$ (named after Schrödinger's cat) and performing a Hadamard transform (4.12) on each qubit. Again, we perform a CNOT from the data qubits to corresponding qubits in the ancilla and measure the ancilla. The result will have even parity iff the selected data qubits have even parity, but the measurement does not destroy superpositions of the possible data states with that parity. Again, a single error in the ancilla could give the wrong parity, so we should repeat the measurement. Also, the preparation of the "cat" state is not at all fault-tolerant, so we could easily have multiple bit flip errors in the "cat" state, which will result in multiple phase errors in the ancilla state. Since phase errors will feed back into the data, we should carefully verify the "cat" state to make sure that we do not have multiple amplitude errors.

Suppose we want to measure a more general operator in $\mathcal{G}$, such as $M_1 = \sigma_x \otimes \sigma_z \otimes \sigma_z \otimes \sigma_x \otimes I$, the first generator for the five-qubit code. Note that under the Hadamard transform

$$\begin{aligned} |0\rangle &\leftrightarrow |0\rangle + |1\rangle \\ |1\rangle &\leftrightarrow |0\rangle - |1\rangle, \end{aligned} \tag{5.3}$$

so the eigenvectors of σ_z transform to the eigenvectors of σ_x and vice-versa. This means to measure M_1, we should perform the Hadamard transform on qubits one and four and instead measure $\sigma_z \otimes \sigma_z \otimes \sigma_z \otimes \sigma_z \otimes I$. We know how to do this from the above discussion. Then we should perform the Hadamard transform again to return to the original state (modulo any collapse caused by

the measurement). In a similar way, we can rotate σ_y into σ_z (exactly how is discussed in more detail in section 5.3), and therefore measure any operator in $\mathcal{G}$.

From the ability to make measurements, we can easily perform error correction for any stabilizer code [39]. Recall that to correct errors, we measure the eigenvalue of each generator of the stabilizer. This we now know how to do fault-tolerantly. This tells us the error syndrome, which tells us the error (or class of degenerate errors). This error is some operator in $\mathcal{G}$, and to correct it, we just apply the operator to the code. Since it is the tensor product of single qubit operators, this is a transversal operation, and is therefore fault-tolerant.

Because a full measurement of the error syndrome takes a fair amount of time, the possibility of an error in the data while measuring the syndrome cannot be ignored. An error in the data in the middle of the syndrome measurement will result in the wrong syndrome, which could correspond to a totally different error with nothing in common with the actual error. Therefore, we should measure the syndrome multiple times, only stopping when we have sufficient confidence that we have determined the correct current error syndrome. Since we are measuring the syndrome multiple times, we only need to measure each bit once per overall syndrome measurement; repetitions of the syndrome measurement will also protect against individual errors in the syndrome bits. The true error syndrome will evolve over the course of repeated measurements. Eventually, more errors will build up in the data than can be corrected by the code, producing a real error in the data. Assuming the basic error rate is low enough, this occurance will be very rare, and we can do many error correction cycles before it happens. However, eventually the computation will fail. In chapter 6, I will show how to avoid this result and do arbitrarily long computations provided the basic error rate is sufficiently low.

5.3 Transformations of the Stabilizer

Now I will begin to discuss how to perform actual operations on encoded states. We already know how to perform encoded σ_x, σ_y, and σ_z operations on stabilizer codes. These operations all commute with the stabilizer and therefore leave the generators of the stabilizer alone. A more general unitary operation U will not necessarily do this. If $M \in S$, then $|\psi\rangle = M|\psi\rangle$ for $|\psi\rangle \in T$, and

$$U|\psi\rangle = UM|\psi\rangle = UMU^\dagger U|\psi\rangle, \tag{5.4}$$

so $UMU^\dagger$ fixes $U|\psi\rangle$. Even if we have an operator N which is not in S, U will take the eigenvectors of N to eigenvectors of $UNU^\dagger$, effectively transforming $N \to UNU^\dagger$. Suppose $UMU^\dagger \in \mathcal{G}$. Then if we want an operation that takes an encoded codeword to another valid codeword, we need $UMU^\dagger \in S$. If this is true for all $M \in S$, then $U|\psi\rangle \in T$ as well, and U is a valid encoded operation. If it is also transversal, we know it will be fault-tolerant as well.

The set of U such that $UAU^\dagger \in \mathcal{G}$ for all $A \in \mathcal{G}$ is the normalizer $N(\mathcal{G})$ of $\mathcal{G}$ in $U(n)$. It turns out that $N(\mathcal{G})$ is generated by the single qubit operations R

(the Hadamard transform) and

$$P = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}, \tag{5.5}$$

and the controlled NOT [17, 22]. The set of U such that $UMU^\dagger \in S$ for all $M \in S$ is the normalizer $N_{U(n)}(S)$ of S in $U(n)$, which need not be a subset of $N(\mathcal{G})$. Any transversal operator in $N_{U(n)}(S)$ is a valid fault-tolerant operation. However, operators outside of $N(\mathcal{G})$ are much more difficult to work with and analyze. Therefore, I will restrict my attention to operators in the intersection of $N(\mathcal{G})$ and $N_{U(n)}(S)$.

The operators in $N(\mathcal{G})$ acting on $\mathcal{G}$ by conjugation permute tensor products of σ_x, σ_y, and σ_z. For instance,

$$R\sigma_x R^\dagger = \tfrac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \sigma_z \tag{5.6}$$

$$R\sigma_z R^\dagger = \tfrac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \sigma_x. \tag{5.7}$$

Also,

$$R\sigma_y R^\dagger = -iR\sigma_x\sigma_z R^\dagger = -iR\sigma_x R^\dagger R\sigma_z R^\dagger = -i\sigma_z\sigma_x = -\sigma_y. \tag{5.8}$$

R switches σ_x and σ_z. Similarly,

$$P\sigma_x P^\dagger = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix} = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} = \sigma_y \tag{5.9}$$

$$P\sigma_z P^\dagger = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & -i \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \sigma_z. \tag{5.10}$$

P switches σ_x and σ_y. These two operations generate all possible permutations of σ_x, σ_y, and σ_z. Operators in $N(\mathcal{G}_1)$ can be viewed as transformations of the Bloch sphere which permute the coordinate axes.

The third generator of $N(\mathcal{G})$ is the controlled NOT. It acts on two qubits, and therefore permutes the elements of $\mathcal{G}_2$. Its action is as follows:

$$\begin{aligned} \sigma_x \otimes I &\rightarrow \sigma_x \otimes \sigma_x \\ I \otimes \sigma_x &\rightarrow I \otimes \sigma_x \\ \sigma_z \otimes I &\rightarrow \sigma_z \otimes I \\ I \otimes \sigma_z &\rightarrow \sigma_z \otimes \sigma_z. \end{aligned} \tag{5.11}$$

Amplitudes are copied forwards and phases are copied backwards, as I described before. In the same way, any element of $N(\mathcal{G})$ gives a permutation of $\mathcal{G}$. These permutations of $\mathcal{G}$ always preserve the group structure of $\mathcal{G}$, so are actually automorphisms of $\mathcal{G}$.

Given an automorphism of $\mathcal{G}$, we can always find an element of $N(\mathcal{G})$ that produces that automorphism [40], modulo the automorphism $iI \rightarrow -iI$. We

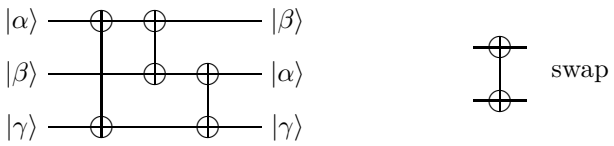

Figure 5.1: Network to swap $|\alpha\rangle$ and $|\beta\rangle$ using ancilla $|\gamma\rangle$.

can find the matrix of a given transformation U corresponding to some automorphism by determining the action of U on basis states. $|0\rangle$ is an eigenvector of σ_z, so it is mapped to an eigenvector of $U\sigma_z U^\dagger$. $|1\rangle = \sigma_x|0\rangle$, so it becomes $(U\sigma_x U^\dagger)U|0\rangle$. For instance, the automorphism $T : \sigma_x \to \sigma_y,\ \sigma_z \to \sigma_x$ maps $|0\rangle \to (1/\sqrt{2})\,(|0\rangle + |1\rangle)$ and $|1\rangle \to \sigma_y T|0\rangle = -(i/\sqrt{2})\,(|0\rangle - |1\rangle)$. Thus, the matrix of T is

$$T = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -i \\ 1 & i \end{pmatrix}. \tag{5.12}$$

Another useful operation is to swap two qubits in a block. This is not a transversal operation, and it is not fault-tolerant by itself. An error during the swap gate can produce errors in the two qubits to be swapped, producing two errors in the same block. However, we do not need to worry about error propagation because the swap gate swaps the errors along with the correct states. Therefore, to get a fault-tolerant swap gate, we only need to produce a circuit to swap qubits that does not directly interact them. Such a circuit is given in figure 5.1.

In order to produce a valid fault-tolerant encoded operation, we may combine swap operations within a block of an error-correcting code and transversal operations on the block to get something that permutes the elements of the stabilizer. The set of such operations is the automorphism group $\mathcal{A}(S)$ of S. Codes with a large automorphism group are therefore better suited for performing fault-tolerant operations. For instance, the seven-qubit code of table 3.4 is invariant under any single-qubit operation in $N(\mathcal{G})$ performed bitwise. There are also a number of permutations of its qubits in the automorphism group, although they turn out to be unimportant in this case. The five-qubit code of table 3.2 has fewer automorphisms. The only transversal operations in its automorphism group are

$$T : \sigma_x \to \sigma_y,\ \sigma_z \to \sigma_x \tag{5.13}$$

and T^2. Note that in the language of GF(4) codes, the operation T corresponds to multiplication by ω^2. Therefore it is a valid transversal operation for any linear quantum code. The five-qubit code is also invariant under cyclic permutations of the five component qubits, although these operations turn out to leave the encoded data unchanged, so are not very useful.

Once we have a possible encoded operation U, we must discover what it actually does to the encoded states. We can do this by analyzing the behavior of $N(S)/S$ under the operation. Because U is in $N(\mathcal{G}) \cap N_{U(n)}(S)$, it also has a natural action on $N(S)/S \cong \mathcal{G}_k$. This action on $\mathcal{G}_k$ is equivalent to some operation in $N(\mathcal{G}_k)$. This is the operation that is performed on the k encoded qubits. For instance, the Hadamard transform R applied bitwise to the seven-qubit code switches $\overline{X} = \sigma_{x5}\sigma_{x6}\sigma_{x7}$ and $\overline{Z} = \sigma_{z5}\sigma_{z6}\sigma_{z7}$. This is just R applied to the $\mathcal{G}_1$ group for the single encoded qubit. In the same way, P bitwise for the seven-qubit code converts $\overline{X}$ into $-\overline{Y}$ ($\overline{Y}$ is the encoded σ_y), and thus performs an encoded $P^\dagger$. The minus sign for $\overline{Y}$ occurs because $\overline{Y} = -i\overline{X}\overline{Z} = -i(i^3)\sigma_{y5}\sigma_{y6}\sigma_{y7} = -\sigma_{y5}\sigma_{y6}\sigma_{y7}$.

For the five-qubit code, $\overline{X} = \sigma_{x1}\sigma_{x2}\sigma_{x3}\sigma_{x4}\sigma_{x5}$ and $\overline{Z} = \sigma_{z1}\sigma_{z2}\sigma_{z3}\sigma_{z4}\sigma_{z5}$, so T bitwise transforms $\overline{X}$ to $\overline{Y}$ and $\overline{Z}$ to $\overline{X}$, and therefore acts as an encoded T operation. For both the five- and seven-qubit codes, the qubit permutations in $\mathcal{A}(S)$ produce the identity operation on the encoded qubits. For a block encoding k qubits, an operation in the automorphism group might perform any multiple-qubit operation in $N(\mathcal{G}_k)$.

We can also do multiple-qubit operations interacting two blocks by applying multiple-qubit operations transversally between the blocks. For instance, we can apply a CNOT from the ith qubit in the first block to the ith qubit in the second block. We can interact r blocks by applying transversally any operation in $N(\mathcal{G}_r)$. We can even apply different operations to different qubits within a block. However, we should not also apply swaps within a block unless we can perform error correction afterwards, since otherwise errors could spread from one qubit in a block to the corresponding qubit in a different block, then back to a different qubit in the first block, producing two errors in the first block.

The stabilizer of two blocks of a code is just $S \times S$. Therefore, the operation, to be valid, must permute the elements of this group. For instance, bitwise CNOT applied between two blocks of the seven-qubit code is a valid operation, because

$$\begin{aligned} M_i \otimes I &\rightarrow M_i \otimes M_i \ (i = 1, 2, 3) \\ M_i \otimes I &\rightarrow M_i \otimes I \ (i = 4, 5, 6) \\ I \otimes M_i &\rightarrow I \otimes M_i \ (i = 1, 2, 3) \\ I \otimes M_i &\rightarrow M_i \otimes M_i \ (i = 4, 5, 6). \end{aligned} \tag{5.14}$$

Since this also takes

$$\begin{aligned} \overline{X} \otimes I &\rightarrow \overline{X} \otimes \overline{X} \\ I \otimes \overline{X} &\rightarrow I \otimes \overline{X} \\ \overline{Z} \otimes I &\rightarrow \overline{Z} \otimes I \\ I \otimes \overline{Z} &\rightarrow \overline{Z} \otimes \overline{Z}, \end{aligned} \tag{5.15}$$

it acts as a CNOT on the encoded qubits. On the other hand, bitwise CNOT applied to the five-qubit code is not a valid operation, because, for instance,

$M_1 = \sigma_x \otimes \sigma_z \otimes \sigma_z \otimes \sigma_x \otimes I$, so $M_1 \otimes I \rightarrow M_1 \otimes (\sigma_x \otimes I \otimes I \otimes \sigma_x \otimes I)$ and $\sigma_x \otimes I \otimes I \otimes \sigma_x \otimes I$ is not in S.

The CSS codes are those for which the stabilizer is the direct product of a part where the elements are tensor products of σ_{xi}'s and a part where the elements are tensor products of σ_{zi}'s. We can also pick the $\overline{X}$ and $\overline{Z}$ operators to be tensor products of σ_{xi}'s and σ_{zi}'s, respectively. This means that just as with the seven-qubit code, bitwise CNOT will be a valid operation for any CSS codes, and will perform the CNOT between corresponding encoded qubits in the two blocks.

Conversely, if bitwise CNOT is a valid operation for a code, that means it is a CSS code: Let $M = XY$ be an arbitrary element of the stabilizer S, where X is the tensor product of σ_{xi}'s and Z is the tensor product of σ_{zi}'s. Then, under CNOT, $M \otimes I \rightarrow M \otimes X$ and $I \otimes M \rightarrow Z \otimes M$. Thus, X and Z are themselves elements of S. The stabilizer therefore breaks up into a σ_x part and a σ_z part, which means it is a CSS code.

5.4 The Effects of Measurements

We are not strictly limited to unitary operations in a quantum computation. We can also make measurements, which correspond to randomly applying one of a set of complete projection operators, usually labeled by eigenvalues of a Hermitian operator. Based on the classical measurement result, we can then apply one of a number of possible operators to the resulting quantum state. This process can be converted into a purely quantum process, but in the idealization where classical computation is error-free while quantum computation is not, there is a distinct advantage in converting as much as possible to classical information. Even in a more realistic situation, classical computation is likely to be much more reliable than quantum computation and classical error-correction methods are simpler than quantum ones. In addition, we may know how to perform operations conditioned on classical information fault-tolerantly even when we do not know how to perform the corresponding quantum operations fault-tolerantly. As we shall see, ancilla preparation and measurement are powerful tools for expanding the available set of fault-tolerant quantum operations.

Suppose we wish to measure operator A, with $A^2 = I$. Measuring A for a state $|\psi\rangle$ will typically give one of two results $|\psi_+\rangle$ or $|\psi_-\rangle$, corresponding to the two eigenvalues ± 1 of A. In order to keep the description of our algorithm under control, we would like a way to convert $|\psi_-\rangle$ to $|\psi_+\rangle$ for any possible input state $|\psi\rangle$. This will not be possible unless we know something more about the possible states $|\psi\rangle$. Suppose we know that there is a unitary operator M, with $M|\psi\rangle = |\psi\rangle$ and $\{M, A\} = 0$. Then

$$\begin{aligned} M^\dagger|\psi_-\rangle &= M^\dagger \frac{1}{2}(I - A)|\psi\rangle = M^\dagger \frac{1}{2}(I - A)M|\psi\rangle \\ &= M^\dagger M \frac{1}{2}(I + A)|\psi\rangle = \frac{1}{2}(I + A)|\psi\rangle \\ &= |\psi_+\rangle. \end{aligned} \tag{5.16}$$

If we make the measurement, then apply $M^\dagger$ if the result is -1 and do nothing if the result is $+1$, then we have applied the nonunitary operator $P_+ = \frac{1}{2}(I+A)$. We can then continue the computation with the assurance that the computer is in the state $|\psi_+\rangle$. In order to perform this nonunitary operator, we have taken advantage of the fact that $|\psi\rangle$ is a $+1$-eigenstate of M. This trick cannot be used if we do not know anything about the state of $|\psi\rangle$.

We know how to measure operators in $\mathcal{G}$ fault-tolerantly. If we prepare an ancilla in a known state and apply a known set of operations in $N(\mathcal{G})$, the resulting state can be at least partially described by a stabilizer S. This stabilizer is not the stabilizer of a quantum error-correcting code, but simply a way of describing the information we have about the state. In many of the applications below, there will be one stabilizer for the error-correcting code, and another which describes the restricted state of the data due to our preparation of the ancilla in a known state. We can fault-tolerantly measure (fault-tolerant with respect to the error-correcting code) an operator $A \in \mathcal{G}$ that anticommutes with some $M \in S$ (the stabilizer describing the data) and correct the result as above to perform the operation P_+. Any operators in S that commute with A will still fix the state of the system after the measurement and correction. Hereafter, in the context of performing operations on encoded states, I will usually speak of "measuring" A when I mean applying P_+ for A.

If $A \in S$, there is no need to measure A to perform P_+, since the state is already an eigenstate of A with eigenvalue $+1$. If A commutes with everything in S but is not in S itself, then measuring A will give us information about which state we had that was fixed by S. However, we do not have an M that anticommutes with A, so we cannot fix P_- to P_+. If A anticommutes with some element of S, say M_1, then we can choose the remaining $n-k-1$ generators of S to commute with A (if M_i anticommutes with A, $M_1 M_i$ will commute with A). The stabilizer S' after applying P_+ will then be generated by A and $M_2, \ldots, M_{n-k}$.

We can better understand the operator P_+ by looking at the transformation it induces from $N(S)/S$ to $N(S')/S'$. Half of the representatives of each coset in $N(S)/S$ will commute with A and half will anticommute, since of N and $M_1 N$, one will commute and one will anticommute. If $N \in N(S)$ commutes with A, its eigenvectors and eigenvalues are left unchanged by measuring A. Therefore the coset represented by N in $N(S')/S'$ will act on $P_+|\psi\rangle$ in the same way as the coset in $N(S)/S$ acted on $|\psi\rangle$. Any representative of the same coset in $N(S)/S$ will produce the same coset in $N(S')/S'$ as long as it commutes with A. We therefore have a map from $N(S)/S \cong \mathcal{G}$ to $N(S')/S' \cong \mathcal{G}$, which is an operation in $N(\mathcal{G})$. Using selected ancilla preparation and existing tranversal operations, we can use this process to create new transversal operations.

A nice example of this formalism, which can be applied independently of quantum error correction, is a description of quantum teleportation [41]. We start with three qubits, the first in an arbitrary state $|\psi\rangle$, the other two in the Bell state $|00\rangle + |11\rangle$. This state can be described by the stabilizer S_1 generated by $I \otimes \sigma_x \otimes \sigma_x$ and $I \otimes \sigma_z \otimes \sigma_z$. The cosets of $N(S_1)/S_1$ can be represented by $\overline{X} = \sigma_x \otimes I \otimes I$ and $\overline{Z} = \sigma_z \otimes I \otimes I$. The third qubit is far away, so we cannot

perform any quantum gates interacting it with the other two qubits. However, we can make measurements on the first two qubits and send the information to be used to perform conditional quantum gates just on the third qubit.

First, we apply a CNOT from the first qubit to the second qubit. This produces stabilizer S_2 generated by $I \otimes \sigma_x \otimes \sigma_x$ and $\sigma_z \otimes \sigma_z \otimes \sigma_z$, with $\overline{X} = \sigma_x \otimes \sigma_x \otimes I$ and $\overline{Z} = \sigma_z \otimes I \otimes I$. Now measure σ_x for the first qubit. This produces stabilizer S_3 generated by $\sigma_x \otimes I \otimes I$ and $I \otimes \sigma_x \otimes \sigma_x$. The coset representative $\sigma_x \otimes \sigma_x \otimes I$ commutes with the measured operator, so it still represents the new coset. Multiplying by the first generator of S_3 still gives a coset representative of $\overline{X}$ in $N(S_3)/S_3$, so $\overline{X} = I \otimes \sigma_x \otimes I$. $\sigma_z \otimes I \otimes I$ does not commute with the measured operator, but $(\sigma_z \otimes \sigma_z \otimes \sigma_z)(\sigma_z \otimes I \otimes I) = I \otimes \sigma_z \otimes \sigma_z$ represents the same coset in $N(S_2)/S_2$ and does commute with the measured operator, so it represents the $\overline{Z}$ coset in $N(S_3)/S_3$. The measurement potentially requires an application of $\sigma_z \otimes \sigma_z \otimes \sigma_z$ if it is necessary to correct P_-. This provides one of the sets of conditional operations used in quantum teleportation.

Now we measure σ_z for the second qubit. This produces the stabilizer S_4 generated by $\sigma_x \otimes I \otimes I$ and $I \otimes \sigma_z \otimes I$. This time, the representative of $\overline{Z}$ commutes with the measured operator, so $\overline{Z}$ for $N(S_4)/S_4$ is $I \otimes \sigma_z \otimes \sigma_z \cong I \otimes I \otimes \sigma_z$. $I \otimes \sigma_x \otimes I$ does not commute, but $(I \otimes \sigma_x \otimes \sigma_x)(I \otimes \sigma_x \otimes I) = I \otimes I \otimes \sigma_x$ does, so in $N(S_4)/S_4$, $\overline{X} = I \otimes I \otimes \sigma_x$. The operation to correct P_- this time is $I \otimes \sigma_x \otimes \sigma_x$. This provides the second set of conditional operations in teleportation.

Note that S_4 completely determines the state of the first two qubits and does not restrict the state of the third qubit at all. In fact, the $\overline{X}$ operator in $N(S_1)/S_1$, which started as σ_x for the first qubit, has been transformed into σ_x for the third qubit, and $\overline{Z}$, which began as σ_z for the first qubit, has become σ_z for the third qubit. This means the final state is $(|0\rangle + |1\rangle) \otimes |0\rangle \otimes |\psi\rangle$, and we have teleported the state as desired.

After we measure σ_x, σ_y, or σ_z for a qubit, we have completely determined the state of that qubit, so its contribution to the stabilizer will just be the operator just measured, and it will not contribute to standard representatives of the cosets in $N(S')/S'$ at all. Therefore, when describing how to produce new transversal operations, I will drop qubits from the notation after they have been measured.

5.5 Producing New Operations in $N(\mathcal{G})$

The group $N(\mathcal{G})$ can be generated by just the operations R, P, and CNOT applied to arbitrary qubits and pairs of qubits. I will now show that, by using measurements, we can, in fact, generate $N(\mathcal{G})$ using just CNOT. Then I will demonstrate that for most known codes, we can apply an encoded CNOT transversally.

First, note that by preparing an ancilla in an arbitrary state and measuring σ_x, σ_y, or σ_z, we can always prepare that ancilla qubit in the $+1$ eigenstate of any of these three operators. Also, there are only six interesting operators in

$N(\mathcal{G}_1)$: I, R, P (and $P^\dagger$), Q (and $Q^\dagger$), T, and T^2 (and $T^\dagger$ and $(T^2)^\dagger$), where $Q = P^\dagger RP$ switches σ_y and σ_z, and $T = RP^\dagger$ is the cyclic permutation of σ_x, σ_y, and σ_z. I have only counted this as six operators, since the adjoints produce the same permutations, but with different signs distributed among σ_x, σ_y, and σ_z. This effect can also be produced by applying σ_x, σ_y and σ_z themselves. Any two non-identity operators in this set, other than T and T^2, will suffice to generate all of them.

Suppose we have an arbitrary single-qubit state $|\psi\rangle$. Let us prepare an ancilla qubit in the $+1$ eigenstate of σ_z, then apply a CNOT from the data qubit to the ancilla qubit. This produces the stabilizer $\sigma_z \otimes \sigma_z$, with $\overline{X} = \sigma_x \otimes \sigma_x$ and $\overline{Z} = \sigma_z \otimes I$. Now measure σ_y for the ancilla qubit and discard the ancilla. This leaves the first qubit with $\overline{X} = -\sigma_y$ and $\overline{Z} = \sigma_z$, which means we have applied $P^\dagger$.

Now prepare the ancilla in the $+1$ eigenstate of σ_x and apply a CNOT from the ancilla qubit to the data qubit. This produces stabilizer $\sigma_x \otimes \sigma_x$, with $\overline{X} = \sigma_x \otimes I$ and $\overline{Z} = \sigma_z \otimes \sigma_z$. Measure σ_y for the ancilla and discard it, leaving $\overline{X} = \sigma_x$ and $\overline{Z} = -\sigma_y$. We have applied $Q^\dagger$. Along with P from above, this suffices to generate $N(\mathcal{G}_1)$ and therefore $N(\mathcal{G}_n)$ for any n.

We can also produce T directly by preparing the ancilla in the $+1$ eigenstate of σ_y and applying a CNOT from the ancilla qubit to the data qubit. This produces a stabilizer of $\sigma_x \otimes \sigma_y$, with $\overline{X} = \sigma_x \otimes I$ and $\overline{Z} = \sigma_z \otimes \sigma_z$. Measure σ_y for the *data* qubit and discard it, leaving $\overline{X} = \sigma_y$ and $\overline{Z} = \sigma_x$, both on the former ancilla qubit. The net result is to apply T, but to move the data from the data qubit to what began as the ancilla qubit.

Now let us turn our attention to transversal operations on quantum error-correcting stabilizer codes. Consider the following four-qubit transformation:

$$\begin{aligned}
\sigma_x \otimes I \otimes I \otimes I &\rightarrow \sigma_x \otimes \sigma_x \otimes \sigma_x \otimes I \\
I \otimes \sigma_x \otimes I \otimes I &\rightarrow I \otimes \sigma_x \otimes \sigma_x \otimes \sigma_x \\
I \otimes I \otimes \sigma_x \otimes I &\rightarrow \sigma_x \otimes I \otimes \sigma_x \otimes \sigma_x \\
I \otimes I \otimes I \otimes \sigma_x &\rightarrow \sigma_x \otimes \sigma_x \otimes I \otimes \sigma_x \\
\sigma_z \otimes I \otimes I \otimes I &\rightarrow \sigma_z \otimes \sigma_z \otimes \sigma_z \otimes I \\
I \otimes \sigma_z \otimes I \otimes I &\rightarrow I \otimes \sigma_z \otimes \sigma_z \otimes \sigma_z \\
I \otimes I \otimes \sigma_z \otimes I &\rightarrow \sigma_z \otimes I \otimes \sigma_z \otimes \sigma_z \\
I \otimes I \otimes I \otimes \sigma_z &\rightarrow \sigma_z \otimes \sigma_z \otimes I \otimes \sigma_z.
\end{aligned} \tag{5.17}$$

Given an element M of an arbitrary stabilizer, this operation applied bitwise maps

$$\begin{aligned}
M \otimes I \otimes I \otimes I &\rightarrow M \otimes M \otimes M \otimes I \\
I \otimes M \otimes I \otimes I &\rightarrow I \otimes M \otimes M \otimes M \\
I \otimes I \otimes M \otimes I &\rightarrow M \otimes I \otimes M \otimes M \\
I \otimes I \otimes I \otimes M &\rightarrow M \otimes M \otimes I \otimes M.
\end{aligned} \tag{5.18}$$

Each of these images is in the group $S \times S \times S \times S$, so this is a valid transversal

operation for *any* stabilizer code. Because of (5.18), this operation just applies itself to the encoded qubits. When the code has multiple qubits per block, (5.17) applies itself to all of the corresponding sets of encoded qubits.

This is very useful, since if we have two logical qubits and prepare two more ancilla logical qubits each in the $+1$ eigenstate of σ_z, and then apply (5.17) to these four qubits, we get a stabilizer with generators $\sigma_z \otimes I \otimes \sigma_z \otimes \sigma_z$ and $\sigma_z \otimes \sigma_z \otimes I \otimes \sigma_z$, and

$$\begin{aligned} \overline{X}_1 &= \sigma_x \otimes \sigma_x \otimes \sigma_x \otimes I \\ \overline{X}_2 &= I \otimes \sigma_x \otimes \sigma_x \otimes \sigma_x \\ \overline{Z}_1 &= \sigma_z \otimes \sigma_z \otimes \sigma_z \otimes I \\ \overline{Z}_2 &= I \otimes \sigma_z \otimes \sigma_z \otimes \sigma_z. \end{aligned} \tag{5.19}$$

Measure σ_x for both ancilla qubits and discard them. This leaves us with

$$\begin{aligned} \overline{X}_1 &= \sigma_x \otimes \sigma_x \\ \overline{X}_2 &= I \otimes \sigma_x \\ \overline{Z}_1 &= \sigma_z \otimes I \\ \overline{Z}_2 &= \sigma_z \otimes \sigma_z. \end{aligned} \tag{5.20}$$

This we can recognize as the CNOT from the first data qubit to the second data qubit. As the CNOT suffices to get every operation in $N(\mathcal{G})$, we can therefore perform any such operation transversally for any stabilizer code encoding a single qubit.

There are other operations like (5.17) that work for any stabilizer code. The condition they must satisfy [35] is for σ_x tensor any number of copies of the identity to map to the tensor product of some number of copies of σ_x and I, and σ_z in the same position must map to the same tensor product of σ_z and I. This means any such automorphism can be fully described by an $n \times n$ binary matrix (for an n-qubit operation). The image of σ_{xi} must commute with the image of σ_{zj} for $i \neq j$. This means that the binary dot product of two different rows of the matrix must be 0. Also, the image of σ_{xi} must anticommute with the image of σ_{zi}. This means that the binary dot product of any row with itself must be 1. These two conditions combine to say that the matrix must be an element of $O(n, \mathbf{Z}_2)$, the orthogonal group over $\mathbf{Z}_2$. The smallest n for which this group has an element other than a permutation is $n = 4$. If we were working with d-dimensional states instead of qubits, we would instead need a matrix in $O(n, \mathbf{Z}_d)$. Note that the straightforward generalization of (5.18) is in $O(n, \mathbf{Z}_d)$ for $n = d + 2$.

Codes which have single-qubit tranversal operations other than the identity will in general have a larger available space of multiple-qubit operations. Any n-qubit automorphism that maps σ_x to the tensor product of I with $U_i(\sigma_x)$ and σ_z to the same tensor product of I with $U_i(\sigma_z)$ will be an automorphism of n copies of S if U_i is an automorphism of S for all i. Note that U_i may be the identity. It may also be possible for U_i to not be an automorphism of $\mathcal{G}_1$ at all,

although this will depend on the code. For instance, for a CSS code, we can have $U_i(\sigma_x) = \sigma_x$, $U_i(\sigma_z) = I$ or $U_i(\sigma_x) = I$, $U_i(\sigma_z) = \sigma_z$.

5.6 Codes With Multiple Encoded Qubits

For codes encoding more than one qubit per block, we have more work to do. We only know how to perform (5.17) between corresponding qubits in different blocks, and furthermore, we must perform the operation between *all* the encoded qubits in both blocks.

The solution to the second problem is straightforward. If we prepare an ancilla qubit in the $+1$ eigenstate of σ_x and apply a CNOT from the ancilla to a single data qubit, we get the stabilizer $\sigma_x \otimes \sigma_x$, with $\overline{X} = \sigma_x \otimes I$ and $\overline{Z} = \sigma_z \otimes \sigma_z$. Then if we measure σ_z for the data qubit, we are left with $\overline{X} = \sigma_x$ and $\overline{Z} = \sigma_z$, both for the ancilla qubit. We have transferred the data qubit to the ancilla qubit without changing it. On the other hand, if we had prepared the ancilla qubit in the $+1$ eigenstate of σ_z and applied the CNOT, nothing in the data qubit would have changed.

We can use this fact to switch individual encoded qubits out of a storage block into a temporary holding block. Prepare the holding block with all the encoded qubits in the $+1$ eigenstate of σ_z, except the jth encoded qubit, which is in the $+1$ eigenstate of σ_x. Then use (5.17) to apply a CNOT from the holding block to the storage block and measure σ_z for the jth encoded qubit in the storage block. This switches the jth encoded qubit from the storage block to the holding block while leaving the other qubits in the storage block undisturbed. The jth encoded qubit in the storage block is left in the state $|0\rangle$, as are all the encoded qubits in the holding block but the jth one.

To perform operations between just the jth encoded qubits in two (or more) different blocks while leaving the other qubits in those blocks alone, we can switch both jth qubits into new, empty blocks, as above. Then we interact them. If necessary, we again clear all but the jth encoded qubit in each temporary block by measuring σ_z. Then we can switch the qubits back into the initial blocks by applying a CNOT from the holding block to the appropriate storage block and measuring $\overline{X}_j$ for the holding block.

This leaves the questions of interacting the jth encoded qubit in one block with the ith encoded qubit in another block, and of interacting two encoded qubits in the same block. We can partially solve either problem by switching the two qubits to be interacted into separate holding blocks. If we know how to swap the jth encoded qubit with the first encoded qubit, we can then swap both qubits into the first position, interact them as desired, then swap them back to their initial positions and switch them back to their storage block or blocks.

One way to swap qubits within a block is to perform some nontrivial action on a single block. For a code with trivial automorphism group, this will not exist. However, almost any automorphism will suffice to swap encoded qubits as desired. This is because there are so few two-qubit operations in $N(\mathcal{G})$. Any automorphism of the code will produce some element of $N(\mathcal{G}_k)$ on the k encoded

qubits, typically (although certainly not always) interacting all of them. If we perform some measurement on all of the encoded qubits in the block except the first and the jth, we are left with a two-qubit operation between those two encoded qubits.

We can always perform single-qubit operations on any encoded qubit in a block by switching the qubit into a fresh block, applying the operation to every encoded qubit in the new block, clearing unneccesary qubits and switching the qubit back to the first block. Using this freedom, any operation in $N(\mathcal{G}_2)$ can be transformed to map $\sigma_x \otimes I$ to one of $\sigma_x \otimes I$, $\sigma_x \otimes \sigma_x$, and $I \otimes \sigma_x$. There is still a remaining freedom to switch σ_y and σ_z on either qubit, and we may also switch either with σ_x for any qubit where the image of $\sigma_x \otimes I$ acts as the identity. We treat the three possibilities as separate cases:

- $\sigma_x \otimes I \to \sigma_x \otimes I$

  The operation preserves the group structure of $\mathcal{G}_2$, so the image of $I \otimes \sigma_x$ must commute with $\sigma_x \otimes I$. Up to single-qubit operations, the possibilities are

  1. $I \otimes \sigma_x$: The image of $\sigma_z \otimes I$ must be either $\sigma_z \otimes I$ or $\sigma_z \otimes \sigma_x$. In the first case, the image of $I \otimes \sigma_z$ is $I \otimes \sigma_z$ and the operation is the identity. In the second case, the image of $I \otimes \sigma_z$ must be $\sigma_x \otimes \sigma_z$. If we apply R to the first qubit before the operation and again after it, this produces a CNOT from the first qubit to the second qubit.
  2. $\sigma_x \otimes \sigma_x$: The image of $\sigma_z \otimes I$ must be $\sigma_z \otimes \sigma_z$ and the image of $I \otimes \sigma_z$ may be either $I \otimes \sigma_z$ or $\sigma_x \otimes \sigma_y$. If it is $I \otimes \sigma_z$, the operation is exactly CNOT from the second qubit to the first. If it is $\sigma_x \otimes \sigma_y$, we can again get CNOT from the second qubit to the first by simply applying Q to the second qubit, followed by the operation.

- $\sigma_x \otimes I \to I \otimes \sigma_x$

  This case is related to the first one by simply swapping the two qubits. Therefore, the possibilities can be reduced to a simple swap, and a CNOT either way followed by a swap.

- $\sigma_x \otimes I \to \sigma_x \otimes \sigma_x$

  Now there are three possibilities for the image of $I \otimes \sigma_x$: $I \otimes \sigma_x$ again, $\sigma_x \otimes I$, or $\sigma_z \otimes \sigma_z$.

  1. $I \otimes \sigma_x$: The image of $I \otimes \sigma_z$ must be $\sigma_z \otimes \sigma_z$. The image of $\sigma_z \otimes I$ may be either $\sigma_z \otimes I$ or $\sigma_y \otimes \sigma_x$. As with case two above, if it is $\sigma_z \otimes I$, this is a CNOT from the first qubit to the second; if it is $\sigma_y \otimes \sigma_x$, we can apply Q to the first qubit and then this operation to get a CNOT from the first qubit to the second.
  2. $\sigma_x \otimes I$: This case can be produced from the previous one by swapping the two qubits. Thus, the operation can be converted into a CNOT from the first qubit to the second followed by a swap.

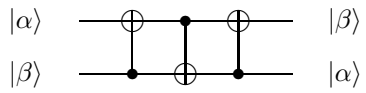

Figure 5.2: Network to swap two qubits using CNOT.

3. $\sigma_z \otimes \sigma_z$: In this case, the image of $\sigma_z \otimes I$ can be $\sigma_z \otimes I$, $I \otimes \sigma_z$, $\sigma_y \otimes \sigma_x$, or $\sigma_x \otimes \sigma_y$. If the image of $\sigma_z \otimes I$ is $\sigma_z \otimes I$, the image of $I \otimes \sigma_z$ must be $I \otimes \sigma_x$ or $\sigma_z \otimes \sigma_y$. If it is $I \otimes \sigma_x$ and we apply R to the second qubit and then this operation, it performs a CNOT from the first qubit to the second. If it is $\sigma_z \otimes \sigma_y$, we can apply $T\sigma_z$ to the second qubit, followed by the operation in order to get a CNOT from the first qubit to the second. If the image of $\sigma_z \otimes I$ is $I \otimes \sigma_z$, we can get it from last case by swapping the qubits, so it can be reduced to a CNOT from the first qubit to the second followed by a swap.

   If the image of $\sigma_z \otimes I$ is $\sigma_y \otimes \sigma_x$, then the image of $I \otimes \sigma_z$ may again be either $I \otimes \sigma_x$ or $\sigma_z \otimes \sigma_y$. If it is $I \otimes \sigma_x$, we can perform Q on the first qubit and R on the second qubit, followed by the two-qubit operation. This produces a CNOT from the first qubit to the second one. If it is $\sigma_z \otimes \sigma_y$, we can perform Q on the first qubit and $T\sigma_z$ on the second qubit, followed by the two-qubit operation. This again produces a CNOT from the first qubit to the second qubit.

   Finally, if the image of $\sigma_z \otimes I$ is $\sigma_x \otimes \sigma_y$, we can produce the previous case by applying a swap, so the two-qubit operation can be converted to a CNOT from the first qubit to the second qubit followed by a swap.

Also, note that R applied to both qubits, followed by a CNOT in one direction, followed by R on both qubits, produces a CNOT in the other direction. Therefore, up to application of single-qubit operations, the only possible two-qubit operations in $N(\mathcal{G})$ are the identity, a CNOT, a swap, or a CNOT followed by a swap. We can make a swap out of three CNOTs using the simple network from figure 5.2.

We cannot make a general swap out of CNOT followed by swap. However, if the control qubit of the CNOT begins in the state $|0\rangle$, the operation does swap the two qubits. This is all that is necessary to get all of $N(\mathcal{G})$, since we only need to move a single data qubit around within an otherwise empty block.

Even if we have no automorphism to switch the jth qubit and the first qubit, we can still do it using quantum teleportation [41]. To do this, we will need an EPR pair entangled between the first and jth encoded qubits. We can make an unencoded EPR pair and then encode it normally. However, a single error during the encoding can destroy the pair. Therefore, we will need to make a number of EPR pairs and purify good ones using an entanglement purification

protocol (EPP) [17, 42]. We can interact corresponding qubits in the EPR pair using operations in $N(\mathcal{G})$, which is all that is necessary. For instance, we could make five EPR pairs and use the one-way EPP derived from the five-qubit code to purify a single good EPR pair. It would take two independent errors to get an error in this pair. An easier way to make the EPR pair is to start with the $+1$ eigenstate of both $\overline{Z}_1$ and $\overline{Z}_j$, then to measure $\overline{X}_1\overline{X}_j$, which is an operator in $N(S)$ just like any other. This leaves the ancilla block in the $+1$ eigenstate of $\overline{Z}_1\overline{Z}_j$ and $\overline{X}_1\overline{X}_j$, which is just an EPR pair.

Once we have a reliable EPR pair, the teleportation process requires only operations in $N(\mathcal{G})$ between corresponding encoded qubits. This allows us to move the jth encoded qubit in one otherwise empty block to the first encoded qubit in the block that previously held the EPR pair. This allows us to do *any* operation in $N(\mathcal{G})$ for *any* stabilizer code. Essentially the same procedures will work when the basic unit is the qudit instead of the qubit [43].

5.7 The Toffoli Gate

The group $N(\mathcal{G})$ is insufficient to allow universal quantum computation. In fact, Knill [44] has shown that a quantum computer using only elements from $N(\mathcal{G})$ and measurements can be simulated efficiently on a classical computer. The argument follows easily from the results of the preceding sections. If we begin with a state initialized to $|0\cdots 0\rangle$, the stabilizer is $\sigma_{z1}, \sigma_{z2}, \ldots$. Each operation in $N(\mathcal{G})$ produces a well-defined transformation of the stabilizer, which can be classically tracked efficiently. Any measurement will also transform the stabilizer in a well-defined way, which is again easy to keep track of on a classical computer. Therefore, we can store and evolve complete information on the state of the quantum computer with only polynomial classical overhead.

In order to perform truly universal quantum computation, even a single gate outside of $N(\mathcal{G})$ can be sufficient. For instance, the Toffoli gate (a three-qubit gate which flips the third qubit iff both of the first two qubits are $|1\rangle$) along with $N(\mathcal{G})$ suffices for universal computation. Shor gave an implementation of the Toffoli gate [38] which can be easily adapted to any code allowing $N(\mathcal{G})$. Since this is any stabilizer code, we can do universal computation for any stabilizer code. Note that there are a number of other gates outside $N(\mathcal{G})$ that we could add to get a universal set of gates (such as the single-qubit $\pi/8$ rotation), and for some codes, it may be easier to perform these gates than the Toffoli gate [45]. However, I will just discuss the implementation of the Toffoli gate.

The Toffoli gate can be expanded using $\mathcal{G}$ as a basis as follows:

$$\frac{1}{4}\left(3I + \sigma_{z1} + \sigma_{z2} - \sigma_{z1}\sigma_{z2} + (I - \sigma_{z1})(I - \sigma_{z2})\sigma_{x3}\right). \tag{5.21}$$

Applying the Toffoli gate to a state therefore produces the following transformation on the elements of $\mathcal{G}_3$:

$$\sigma_{x1} \quad \rightarrow \quad \frac{1}{16}\left(3I + \sigma_{z1} + \sigma_{z2} - \sigma_{z1}\sigma_{z2} + (I - \sigma_{z1})(I - \sigma_{z2})\sigma_{x3}\right)$$

$$
\begin{aligned}
& \times (3I - \sigma_{z1} + \sigma_{z2} + \sigma_{z1}\sigma_{z2} + (I + \sigma_{z1})(I - \sigma_{z2})\sigma_{x3})\,\sigma_{x1} \\
= \;& \frac{1}{2}\,(I + \sigma_{z2} + (I - \sigma_{z2})\sigma_{x3})\,\sigma_{x1} \\
\sigma_{x2} \;\rightarrow\; & \frac{1}{2}\,(I + \sigma_{z1} + (I - \sigma_{z1})\sigma_{x3})\,\sigma_{x2} \\
\sigma_{x3} \;\rightarrow\; & \sigma_{x3} \\
\sigma_{z1} \;\rightarrow\; & \sigma_{z1} \\
\sigma_{z2} \;\rightarrow\; & \sigma_{z2} \\
\sigma_{z3} \;\rightarrow\; & \frac{1}{16}\,(3I + \sigma_{z1} + \sigma_{z2} - \sigma_{z1}\sigma_{z2} + (I - \sigma_{z1})(I - \sigma_{z2})\sigma_{x3}) \\
& \times (3I + \sigma_{z1} + \sigma_{z2} - \sigma_{z1}\sigma_{z2} - (I - \sigma_{z1})(I - \sigma_{z2})\sigma_{x3})\,\sigma_{z3} \\
= \;& \frac{1}{2}\,(I + \sigma_{z1} + (I - \sigma_{z1})\sigma_{z2})\,\sigma_{z3}.
\end{aligned}
\tag{5.22}
$$

This means σ_{z1}, σ_{z2}, and σ_{x3} stay the same, σ_{x1} becomes σ_{x1} tensor a CNOT from qubit two to qubit three, σ_{x2} becomes σ_{x2} tensor a CNOT from qubit one to qubit three, and σ_{z3} becomes σ_{z3} tensor a conditional sign for qubits one and two.

Suppose we can make the ancilla

$$|A\rangle = \frac{1}{2}(|000\rangle + |010\rangle + |100\rangle + |111\rangle). \tag{5.23}$$

This state is fixed by the three operators

$$
\begin{aligned}
M_1 &= \frac{1}{2}\,(I + \sigma_{z2} + (I - \sigma_{z2})\sigma_{x3})\,\sigma_{x1} \\
M_2 &= \frac{1}{2}\,(I + \sigma_{z1} + (I - \sigma_{z1})\sigma_{x3})\,\sigma_{x2} \\
M_3 &= \frac{1}{2}\,(I + \sigma_{z1} + (I - \sigma_{z1})\sigma_{z2})\,\sigma_{z3}.
\end{aligned}
\tag{5.24}
$$

Now suppose we have three data qubits (numbers four, five, and six) that we wish to perform a Toffoli gate on. We simply apply CNOTs from qubit one to qubit four, qubit two to qubit five, and from qubit six to qubit three. This produces the following "stabilizer":

$$
\begin{aligned}
M_1' &= \frac{1}{2}\,(I + \sigma_{z2} + (I - \sigma_{z2})\sigma_{x3})\,\sigma_{x1}\sigma_{x4} \\
M_2' &= \frac{1}{2}\,(I + \sigma_{z1} + (I - \sigma_{z1})\sigma_{x3})\,\sigma_{x2}\sigma_{x5} \\
M_3' &= \frac{1}{2}\,(I + \sigma_{z1} + (I - \sigma_{z1})\sigma_{z2})\,\sigma_{z3}\sigma_{z6}.
\end{aligned}
\tag{5.25}
$$

Then measure σ_{z4}, σ_{z5}, and σ_{x6} and discard qubits 4–6. As we can see, this produces the transformation (5.22) on the three data qubits while moving them to what were formerly the ancilla qubits. Note that correcting for measured eigenvalues of -1 will require applying M_1, M_2, or M_3, which are not elements of $\mathcal{G}$. They are, however, elements of $N(\mathcal{G})$.

Therefore, in order to perform the Toffoli gate on encoded states, we must produce an encoded version of the ancilla $|A\rangle$. Then we need only perform measurements and encoded operations in $N(\mathcal{G})$ to produce the effect of a Toffoli gate. Below, I will assume $\mathcal{G}$ only encoded one qubit per block. If it encodes more, we can still do the same thing by moving the qubits to be interacted into the first encoded qubit in otherwise empty blocks. The $\overline{X}$ and $\overline{Z}$ operators used to create the ancilla are just $\overline{X}_1$ and $\overline{Z}_1$.

To produce the encoded ancilla $|A\rangle$, we start with the encoded version of the state $|A\rangle + |B\rangle$, where

$$|B\rangle = \frac{1}{2}(|001\rangle + |011\rangle + |101\rangle + |110\rangle). \tag{5.26}$$

Note that $|B\rangle$ is related to $|A\rangle$ by applying σ_x to the third qubit. Since

$$|A\rangle + |B\rangle = \sum_{a=000}^{111} |a\rangle = (|0\rangle + |1\rangle)^3, \tag{5.27}$$

we can easily prepare it by measuring $\overline{X}$ for each block. Henceforth, $|A\rangle$ and $|B\rangle$ will denote the encoded versions of themselves. Now we take an ancilla in a "cat" state $|0\ldots0\rangle + |1\ldots1\rangle$, where the number of qubits in the cat state is equal to the number of qubits in a single block of the code. Then we will perform an operation that takes

$$\begin{aligned} |0\ldots0\rangle|A\rangle &\rightarrow |0\ldots0\rangle|A\rangle \\ |1\ldots1\rangle|A\rangle &\rightarrow |1\ldots1\rangle|A\rangle \\ |0\ldots0\rangle|B\rangle &\rightarrow |0\ldots0\rangle|B\rangle \\ |1\ldots1\rangle|B\rangle &\rightarrow -|1\ldots1\rangle|B\rangle. \end{aligned} \tag{5.28}$$

Then under (5.28),

$$(|0\ldots0\rangle+|1\ldots1\rangle)(|A\rangle+|B\rangle) \rightarrow (|0\ldots0\rangle+|1\ldots1\rangle)|A\rangle+(|0\ldots0\rangle-|1\ldots1\rangle)|B\rangle. \tag{5.29}$$

If we measure $\sigma_x \otimes \cdots \otimes \sigma_x$ for the cat state, if we get $+1$, the rest of the ancilla is in the state $|A\rangle$. If we get -1, the rest of the ancilla is in the state $|B\rangle$. One complication is that a single qubit error in the cat state can cause this measurement result to be wrong. Luckily,

$$(|0\ldots0\rangle + |1\ldots1\rangle)|A\rangle \rightarrow (|0\ldots0\rangle + |1\ldots1\rangle)|A\rangle \tag{5.30}$$

$$(|0\ldots0\rangle + |1\ldots1\rangle)|B\rangle \rightarrow (|0\ldots0\rangle - |1\ldots1\rangle)|B\rangle. \tag{5.31}$$

Therefore, if we prepare another cat state and apply (5.28) again, we should again get $+1$ if the ancilla was actually in the state $|A\rangle$ after the first measurement and -1 if it was actually in the state $|B\rangle$. We can therefore get any desired level of reliability for the ancilla state by repeating (5.28) a number of times. Finally, once we are confident we have either $|A\rangle$ or $|B\rangle$, we apply $\overline{X}$ to

the third ancilla qubit if it is $|B\rangle$. This means we will always have prepared the state $|A\rangle$.

To perform (5.28), we will have to perform the operation $|A\rangle \to |A\rangle$ and $|B\rangle \to -|B\rangle$ if and only if the qubits of the cat state are $|1\ldots1\rangle$. If the qubits of the cat state are $|0\ldots0\rangle$, then we do nothing to the rest of the ancilla. I will show that we can apply $|A\rangle \to |A\rangle$ and $|B\rangle \to -|B\rangle$ using a series of transversal operations and measurements. If we apply these operations and measurements conditioned on the corresponding qubit from the cat state being $|1\rangle$, then we have actually performed (5.28). Conditioning the operations on the cat state bit will generally involve using Toffoli gates and possibly other gates outside $N(\mathcal{G})$, but they are all gates on *single* qubits rather than blocks. We assume we know how to perform universal computation on individual qubits, so these gates are available to us.

The state $|A\rangle$ is a $+1$-eigenvector of M_3, from equation (5.24). $|B\rangle$ is a -1-eigenvector of the same M_3, so applying M_3 does, in fact, transform $|A\rangle \to |A\rangle$ and $|B\rangle \to -|B\rangle$. M_3 is just a conditional sign on the first two qubits (i.e. an overall sign of -1 iff both qubits are $|1\rangle$) times σ_z on the third qubit. Therefore it is in $N(\mathcal{G})$ and can be performed transversally for any stabilizer code. Therefore, we can perform universal computation using any stabilizer code.

5.8 Construction of Gates in $N(\mathcal{G})$

In order to use the general fault-tolerant protocols, we need to apply three- or four-qubit gates. Suppose our basic gates are limited to one- and two-qubit gates. These gates are sufficient to give us any gates in $N(\mathcal{G})$. I will now give a construction for any gate in $N(\mathcal{G})$ using one- and two-qubit gates.

The construction will be inductive. In section 5.6, I showed that any one- or two-qubit gate could be made using R, P, and CNOT. Suppose we can construct any n-qubit gate using one- and two-qubit gates, and let U be an $(n+1)$-qubit gate. Using swaps and one-qubit gates, we can guarantee that

$$M = U\sigma_{z1}U^\dagger = \sigma_{x1} \otimes M' \tag{5.32}$$

and

$$N = U\sigma_{x1}U^\dagger = I \otimes N' \text{ or } \sigma_{z1} \otimes N'. \tag{5.33}$$

Note that $\{M, N\} = 0$. Suppose

$$U(|0\rangle \otimes |\psi\rangle) = |0\rangle \otimes |\psi_1\rangle + |1\rangle \otimes |\psi_2\rangle, \tag{5.34}$$

where $|\psi\rangle$, $|\psi_1\rangle$, and $|\psi_2\rangle$ are states of the last n qubits. The results of section 5.4 tell us that if we measure σ_z for the first qubit after applying U and apply $M^\dagger$ (which anticommutes with σ_{z1}) if the result is -1, we will get $|0\rangle \otimes |\psi_1\rangle$. This means that $|\psi_2\rangle = M'|\psi_1\rangle$. Define U' by $U'|\psi\rangle = |\psi_1\rangle$. Then

$$U|0\rangle \otimes |\psi\rangle = (I + M)(|0\rangle \otimes U'|\psi\rangle). \tag{5.35}$$

Now,

$$\begin{aligned} U(|1\rangle \otimes |\psi\rangle) &= U\left[(\sigma_x|0\rangle) \otimes |\psi\rangle\right] && (5.36)\\ &= NU(|0\rangle \otimes |\psi\rangle) && (5.37)\\ &= N(I+M)(|0\rangle \otimes U'|\psi\rangle) && (5.38)\\ &= (I-M)N(|0\rangle \otimes U'|\psi\rangle) && (5.39)\\ &= (I-M)(|0\rangle \otimes N'U'|\psi\rangle). && (5.40) \end{aligned}$$

Therefore, if we first apply U' to the last n qubits, followed by applying N' to the last n qubits conditioned on the first qubit, followed by a Hadamard transform R on the first qubit, followed by M' on the last n qubits conditioned on the first qubit, we have applied U:

$$\begin{aligned} |0\rangle \otimes |\psi\rangle + |1\rangle \otimes |\phi\rangle &\rightarrow |0\rangle \otimes U'|\psi\rangle + |1\rangle \otimes U'|\phi\rangle && (5.41)\\ &\rightarrow |0\rangle \otimes U'|\psi\rangle + |1\rangle \otimes N'U'|\phi\rangle && (5.42)\\ &\rightarrow (|0\rangle + |1\rangle) \otimes U'|\psi\rangle + (|0\rangle - |1\rangle) \otimes N'U'|\phi\rangle && (5.43)\\ &\rightarrow (|0\rangle \otimes U'|\psi\rangle + |1\rangle \otimes M'U'|\psi\rangle) \\ &\quad + (|0\rangle \otimes N'U'|\phi\rangle - |1\rangle \otimes M'N'U'|\phi\rangle) && (5.44)\\ &= [|0\rangle \otimes U'|\psi\rangle + M(|0\rangle \otimes U'|\psi\rangle)] \\ &\quad + [|0\rangle \otimes N'U'|\phi\rangle - M(|0\rangle \otimes N'U'|\phi\rangle)] && (5.45)\\ &= (I+M)(|0\rangle \otimes U'|\psi\rangle) \\ &\quad + (I-M)(|0\rangle \otimes N'U'|\phi\rangle) && (5.46)\\ &= U(|0\rangle \otimes |\psi\rangle) + U(|1\rangle \otimes |\phi\rangle) && (5.47)\\ &= U(|0\rangle \otimes |\psi\rangle + |1\rangle \otimes |\phi\rangle). && (5.48) \end{aligned}$$

U' is an n-qubit gate in $N(\mathcal{G})$, which, by the inductive hypothesis, we can perform using one- and two-qubit gates. Both M' and N' are in $\mathcal{G}$, so applying them conditioned on the first qubit requires only two-qubit gates in $N(\mathcal{G})$. Therefore, this construction allows us to perform any U in $N(\mathcal{G})$ using only one- and two-qubit gates. The construction is summarized in figure 5.3.

To get M and N in the correct form requires only identifying a single qubit on which M does not act as the identity and N acts differently from M. From there, a single one-qubit gate and a swap between that qubit and the first puts M and N in the desired form. It is not really necessary for the construction that the selected qubit be in the first position, so we can actually put M and N in the right form using just one one-qubit gate. We also need to perform R on that qubit in the middle of the operation. Applying M' and N' conditioned on the selected qubit uses up to $2n$ two-qubit gates. Therefore, this construction of U uses the gates in U' plus up to two one-qubit gates and $2n$ two-qubit gates. Thus, by induction, an $(n+1)$-qubit gate ($n \geq 2$) can use up to $2(n-2)$

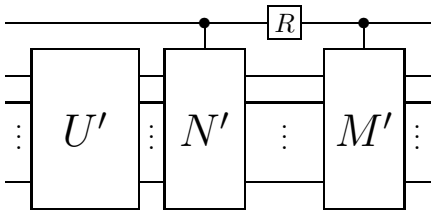

Figure 5.3: Recursive construction of gates in $N(\mathcal{G})$.

one-qubit gates and

$$1 + \sum_{j=3}^{n+1} 2(j-1) = 1 + (n+2)(n-1) = n^2 + n - 1 \tag{5.49}$$

two-qubit gates.

Note that this construction can also be used for encoding data into a stabilizer code. The map U will map $\sigma_{xi} \rightarrow \overline{X}_i$ and $\sigma_{zi} \rightarrow \overline{Z}_i$ $(i = 1, \ldots, k)$ for the k data qubits. The remaining $n-k$ qubits start out as $|0\rangle$, so for $i = k+1, \ldots, n$, we map $\sigma_{zi} \rightarrow M_{i-k}$, where M_j $(j = 1, \ldots, n-k)$ are generators of S. Any remaining freedom for the choice of the image of σ_{xi} for $i = k+1, \ldots, n$ is unimportant. This produces an encoding for any stabilizer code using any $\overline{X}$ and $\overline{Z}$ operators in $N(\mathcal{G})$. In some cases, it may be more efficient than the construction given in chapter 4, but the upper bound for efficiency is higher.

5.9 Refining the Error Correction Algorithm

Since errors occur while we are measuring the error syndrome, we are inevitably led to a race between the errors that are constantly occuring and our ability to correct them. Therefore it is desireable to be able to perform error correction as efficiently as possible. In this section, I will discuss a few ways of speeding up error correction.

One significant improvement is to do classical error correction on the syndrome bits [46]. The most basic form of error correction described in section 5.2 measures the eigenvalues of the $n-k$ generators of S. If we treat these as classical bits, we can encode them using a classical $[m, n-k, d']$ linear code. The bits of the classical codeword will be linear combinations of the original syndrome bits, which means they will correspond to eigenvalues of products of the generators of the stabilizer. This means we need only measure these m new elements of the stabilizer. Then we can do classical error correction on the result to extract the actual $(n-k)$-bit syndrome. If there were less than d' errors on the measured syndrome bits, we can still determine the real syndrome. This protects very well against ancilla errors that produce the wrong measurement

result for a single syndrome bit. It protects less well against data errors that cause the syndrome to change in the middle of measurement, but there is a good chance it will warn us when such an error has occurred. If no errors are detected using the classical code, it is quite likely we have measured the correct syndrome. There is still a chance that we have not, so we may want to repeat the measurement, but we will not have to do it as many times to produce the same level of confidence in the result.

Another possible improvement is to reduce the number of qubits needed to perform error correction. Below, I present a method due to Steane [47]. This method puts more effort into preparing the ancilla, allowing a reduction in the number of operations performed on the data. In some situations, this results in an improvement in error tolerance; in other situations, the effort spent in preparing the ancilla is too large, and this results in worse tolerance for errors.

Steane's ancilla state uses $2n$ qubits, which are prepared in the sum of the states of a classical code. The specific classical code is formed by taking the two matrices in the binary vector space representation of S (section 3.4) and tacking them together into a single $(n-k)\times 2n$ matrix. The matrix for the σ_z's is first. This is the parity check matrix of the classical code. The ancilla state can be described by a stabilizer S_A on $2n$ qubits. The first $n-k$ generators of the stabilizer are the rows of the parity check matrix with σ_z's for the 1s. The remaining $n+k$ generators of the stabilizer are the $n+k$ independent tensor products of σ_x's that commute with the first $n-k$ generators. Note that the fact that S is Abelian means that $n-k$ of the new generators will also be formed directly from the generators of the stabilizer, this time by combining the σ_x and σ_z matrices with the σ_x one first and replacing 1s with σ_x's. There is only a single state in the Hilbert space fixed by all $2n$ of these generators, and that is the desired ancilla state.

For instance, if the original code is a CSS code such as the seven-qubit code, the resulting ancilla state is the tensor product of two ancilla states, each in the superposition of all the states in one of the two classical codes that make up the CSS code. For the seven-qubit code, that means two copies of $|\overline{0}\rangle + |\overline{1}\rangle$, where $|\overline{0}\rangle$ and $|\overline{1}\rangle$ are the encoded 0 and 1 states for the seven-qubit code. In general, the classical code will be able to identify as many errors as the quantum code can, counting errors in both bits j and $j+n$ (for $j \le n$) as a single error.

Once we have this ancilla, we should again verify it, as we did for the "cat" states in sections 5.2 and 5.7. Then we apply a CNOT from data qubit i to ancilla qubit i, followed by a Hadamard transform R on the data qubit and a CNOT from the ith data qubit to the $(n+i)$th ancilla qubit, followed by a final Hadamard transform on the data qubit. Assuming no phase errors in the ancilla, the data qubit ends up in its original state. We can see this by looking at the stabilizer of the ancilla. The last $n+k$ generators M of S_A are all tensor products of σ_x's, so the CNOTs simply map $I\otimes M \to I\otimes M$, which is obviously still in $S\times S_A$. The first $n-k$ generators are tensor products of σ_z's, say $M_1\otimes M_2$ (with M_1 and M_2 n-qubit operators). The CNOTs then map

$$I\otimes(M_1\otimes M_2) \to M_1(RM_2R^\dagger)\otimes(M_1\otimes M_2). \tag{5.50}$$

But M_1 has a σ_z anywhere some element $M \in S$ does and $RM_2R^\dagger$ has a σ_x anywhere the same M does, so $M_1(RM_2R^\dagger) = M$, and $M_1(RM_2R^\dagger)\otimes(M_1\otimes M_2)$ is in $S \times S_A$.

The effect of the CNOTs on the generators M of S is to copy the σ_x's forward into the first n qubits of the ancilla and the σ_z's forward into σ_x's in the last n qubits of the ancilla. That is, $M \otimes I \rightarrow M \otimes (M_1 \otimes M_2)$, where M_1 and M_2 are the product of σ_x's, and $M_1 \otimes M_2$ is one of the second set of $n-k$ generators of S_A. Therefore a correct codeword will have no effect on the ancilla.

Measuring σ_z on each of the $2n$ ancilla qubits will therefore give us a random codeword from the classical code without disturbing the data or the quantum code. A bit flip error in the jth qubit of the quantum code will carry forward to a bit flip error in the jth qubit of the ancilla, and a phase error in the jth qubit of the quantum code will produce a bit flip error in the $(n+j)$th qubit of the ancilla. Therefore, errors in the quantum code will produce bit flip errors in the measured classical codeword. The actual codeword tells us nothing, but the error syndrome will identify the error in the quantum code. As with the cat state method, an incorrect ancilla qubit can result in the wrong error syndrome, but repeating the error syndrome measurement can give an arbitrarily high confidence level to the result. Single-qubit phase errors in the ancilla will just feed back to single-qubit phase or bit flip errors in the data.

Chapter 6

Concatenated Coding

6.1 The Structure of Concatenated Codes

Encoding data using a quantum error-correcting code and applying fault-tolerant operations to it may or may not actually improve the basic error rate for the computation. Since the gates involved in error correction are themselves noisy, the process of error correction introduces errors at the same time it is fixing them. If the basic gate error rate is low enough, the error correction will fix more errors than it introduces on the average, and making a fault-tolerant computation will help rather than harm. If the error rate is too high, attempting to correct errors will introduce more errors than are fixed, and error correction is actively doing harm. Even if error correction helps rather than harms, statistical fluctuations will eventually produce more errors than the code can correct, resulting in a real error in the data. Furthermore, the extra computational overhead required to do fault-tolerant operations may counteract the additional resistance to errors provided by the code, so the encoded computer may not be able to do longer computations than the original computer.

Nevertheless, if the basic error rate in the quantum computer is low enough, we *will* be able to do longer computations using quantum codes and fault-tolerance than we could without them. Suppose we can get a certain amount of improvement by using a specific code, say the seven-qubit code. We might imagine that by using a code that corrects more errors, we could do a longer computation yet, and by increasing the number of errors the code corrects indefinitely, we could do arbitrarily long computation. However, for arbitrary families of codes, the number of steps required to do error correction may increase rapidly with the number of errors corrected. Therefore, the time required to do error correction may eventually overwhelm the capability of the code to deal with errors, and the performance of the computer will start to decrease again. To solve this problem, we need to find a class of codes where the time to measure the error syndrome increases only slowly with the error-correcting capabilities of the code.

The desired class of codes is concatenated codes [34, 45, 48, 49]. For a concatenated code, the data is encoded using some $[n, k, d]$ code, then each qubit in a block is again encoded using an $[n_1, 1, d_1]$ code. The qubits making up blocks in the new code may be further encoded using an $[n_2, 1, d_2]$ code, and so on indefinitely. The result is an $[nn_1n_2 \cdots n_{l-1}, k, dd_1d_2 \cdots d_{l-1}]$ code. We can find the error syndrome of such a code rather rapidly. We measure the error syndrome for the $[n_{l-1}, 1, d_{l-1}]$ code (the *first level* of the code) for all of the blocks of n_{l-1} qubits at once. To do this, we must make the assumption that we can do parallel computation on different qubits. Note that we need this assumption anyway, or storage errors will always build up on some block while we are correcting errors on the other blocks. Similarly, we measure the error syndrome for the $[n_{l-2}, 1, d_{l-2}]$ code at the second level of the code in parallel for different blocks, and so on, for all l levels of the code. Therefore, we can measure the error syndrome for the whole code in only the sum of the number of steps required to measure each constituent code, instead of something like the product, which would be a more typical complexity for a code of the same parameters.

In order to analyze concatenated codes, it is useful to make a few simplifying assumptions. One assumption is that we are using the same code at every level. One particularly good code for this purpose is the $[7, 1, 3]$ code, because any operation in $N(\mathcal{G})$ can be immediately performed transversally, keeping the overhead for fault-tolerant computation small. In addition, it is a small code, so the complexity of error correction is not too large. Allowing varying codes at different levels may improve the space efficiency of the code, but it will not change the basic results. The other simplifying assumption is that the operations at level j are basically similar to operations at level $j + 1$. Each level feeds information about error rates for different gates and storage errors and relative times for the different operations to the next lower level, but nothing else. Error correction at each level is an independent process. Note that this will impair the error-correction properties of the code, since the full minimum distance of the code assumes that we combine information about the error syndrome from all the different levels. However, even with this assumption, we will find that for low enough basic error rates, we can do arbitrarily long computations with arbitrarily low real error rates by using sufficiently many levels of concatenation (the basic error rate is the rate of errors in actual physical qubits due to gates or storage errors; the real error rate is the rate of errors in the encoded data). When the basic error rate is low enough, adding an extra level of concatenation further reduces the real error rate; if the basic error rate is too high, adding an extra layer increases the real error rate because of the extra time spent on error correction and calculation.

In this chapter, I will present a rough calculation of the error threshhold below which arbitrarily long computation is possible. In my discussion, the zeroth level of the code consists of the individual physical qubits making it up. These qubits form the blocks of a $[7, 1, 3]$ code. Each block of seven physical qubits forms a qubit at the first level of the code. In general, qubits at the jth level of the code consist of 7^j physical qubits. There are a total of l levels

in the code. The qubits at the lth level are the real data qubits. We wish to keep the effective error rate on these qubits as low as possible. For this calculation, I will assume that storage errors occur independently on different physical qubits with rate p_{stor}. The error rate for any one- or two-qubit gate in $N(\mathcal{G})$ will be p_g, and the error rate for the Toffoli gate will be p_{Tof}. I assume any gate may produce correlated errors on the qubits affected by the gate, but will produce no errors on any other qubits. There will be an additional storage error on qubits unaffected by the gate, but the storage error is included in the gate error for qubits that are affected by the gate. All the errors are assumed to be stochastically distributed, so the error probabilities for different qubits will add instead of the error amplitudes in the quantum states. In addition, the error rates for state preparation and state measurement will be important. I will denote them by p_{prep} and p_{meas}, respectively.

The computation will call for various operations performed on the qubits encoded at various different levels. After any operation at level j, I will perform error correction at level j. This means we can give an effective error rate to each operation at level j. The fact that a given error rate refers to a gate at level j will be noted by a superscript (j). Thus, $p_{stor}^{(0)}$ is the storage error rate on the physical qubits, while $p_g^{(l)}$ is the effective error rate on the data qubits from performing an operation in $N(\mathcal{G})$. Only allowing one gate per error correction will typically reduce the performance of the code. Errors created during error correction will dominate; an optimized code would perform error correction when the expected accumulated chance of errors was roughly equal to the chance of errors during error correction. However, the assumption of one gate per error correction is another very useful simplifying assumption because it preserves the self-similar character of the concatenated code, allowing a relatively straightforward recursive calculation of the real error rates.

Some logical operations, such as the Toffoli gate, will require more and more physical operations as the level increases. The basic time required to perform a physical operation will be 1, and the storage error rate (at any level) is the error rate per unit time. The time to perform a Toffoli gate at level j will be denoted $t_{Tof}^{(j)}$. Because operations in $N(\mathcal{G})$ can be performed at any level just by performing a single operation from $N(\mathcal{G})$ in parallel at the next lower level, the time to perform an operation in $N(\mathcal{G})$ at any level is just 1. The time to prepare a state encoded at the jth level is $t_{prep}^{(j)}$ and the time to measure a qubit at the jth level is $t_{meas}^{(j)}$. $t_{prep}^{(0)} = 0$ and $t_{meas}^{(0)} = 1$.

6.2 Threshhold for Storage Errors and Gates From $N(\mathcal{G})$

To determine $p_g^{(j)}$ in terms of quantities at level $j-1$, we note that a gate in $N(\mathcal{G})$ at level j consists of a single gate in $N(\mathcal{G})$ on each of the constituent qubits at level $j-1$ followed by a full error correction cycle at level $j-1$. In order for the level j gate to have an error, there must be two errors at level

$j-1$, either in the $N(\mathcal{G})$ gate or in the error correction. I will assume that there is no residual error that was missed in an earlier error correction step. A more careful calculation should consider such leftover errors, which can be significant. Suppose the chance of an error occuring in a single data qubit during a single measurement of the error syndrome is p_{EC}. There are a few possible situations that result in an error at level j. Two errors at level $j-1$ could occur in any of $\binom{7}{2} = 21$ choices of two qubits. This could occur from two $N(\mathcal{G})$ gates going wrong, with probability $(p_g^{(j-1)})^2$. We repeat the error syndrome measurement until we get the same result twice. If there is one error from an $N(\mathcal{G})$ gate and one from either of these measurements of the error syndrome, there will be an error at level j. The probability of this is $4p_g^{(j-1)}p_{EC}$. Finally, both errors could come from the error correction. This could be two errors in the first or second syndrome measurement, with probability $2p_{EC}^2$. Given one error in a syndrome measurement, we will need to do three syndrome measurements total. If two of those go wrong, it will also produce an error at level j. This has probability $6p_{EC}^2$. There are also a number of possibilities involving an error in the ancilla state producing an incorrect syndrome and requiring more measurements. However, I assume the error rates involved are all fairly low, so the probability of this situation producing an error at level j is smaller by $O(p)$, which I will assume is negligable. Thus, the total gate error rate at level j is

$$p_g^{(j)} = 21\left((p_g^{(j-1)})^2 + 4p_g^{(j-1)}p_{EC} + 8p_{EC}^2\right). \tag{6.1}$$

Similarly, a single time step at level j without a gate involves a single time step without a gate at level $j-1$ followed by error correction. Therefore,

$$p_{stor}^{(j)} = 21\left((p_{stor}^{(j-1)})^2 + 4p_{stor}^{(j-1)}p_{EC} + 8p_{EC}^2\right). \tag{6.2}$$

The salient aspect of these equations is that the probability of error at level j is of the order of the square of the error rate at level $j-1$. This means that $p_g^{(l)}$ will scale roughly as

$$p_g^{(0)}(p_g^{(0)}/p_{thresh})^{2^l} \tag{6.3}$$

for some threshhold error rate p_{thresh} and similarly for $p_{stor}^{(l)}$. This is a very rapid decrease in $p_g^{(l)}$ as a function of l when $p_g^{(0)} < p_{thresh}$. We will thus only need a few levels, of order $\log(\log p)$ to bring the real error rate down to $O(p)$ per step. Thus, the number of extra qubits necessary for a fault-tolerant computation is only polylog p times the original number, which is a very good scaling. However, while the asymptotic scaling is quite good, for vaguely reasonable p, the actual number of extra qubits needed is quite large.

In order to determine the threshhold p_{thresh}, let us calculate p_{EC}. I will assume we are using Shor's cat state method to correct errors, although another method (such as Steane's) might ultimately lead to better performance. We have to measure six syndrome bits, so we will need to prepare six cat states, each using four qubits. I will assume a limited ability to plan ahead in the

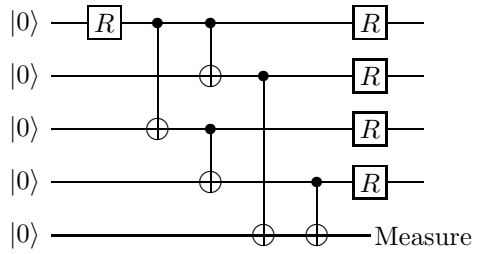

Figure 6.1: Cat state construction and verification.

calculation, so the data qubits will have to wait for the first cat state in a single measurement of the error syndrome, but the other cat states are being prepared at the same time, so they will be ready just when they are needed. To prepare a cat state, we start with all four qubits in the state $|0\rangle$ (encoded using the code at level $j-1$), perform a Hadamard rotation R on the first qubit, then a CNOT from the first qubit to the third qubit, and then two more CNOTs, from the first qubit to the second and from the third to the fourth, as shown in figure 6.1. Bit flip errors at this point will become phase errors after the final Hadamard transform, so we need to ensure that there is at most one. Every way a single gate error earlier in the construction can produce two bit flip errors here makes the second and fourth qubits different. Therefore, we perform CNOTs from the second and fourth qubits to an additional ancilla test qubit and measure the test qubit. If it is $|0\rangle$, we can use the ancilla; if it is $|1\rangle$, there is at least one error in the cat state, possibly two. We throw the cat state out and construct another one. Finally, we must perform a Hadamard transform on each of the four qubits in the cat state to get the actual ancilla used in error correction.

An examination of the circuit shows that any bit flip errors before the cycle in which there are two CNOTs will cause the test qubit to flip. Therefore, only errors at this stage or later will have a chance of affecting the actual ancilla used. For the second and fourth qubits, the error must actually occur after (or during) the CNOT to the test qubit. Therefore, the chance of an important error in any single ancilla qubit is $2p_g + p_{stor}$ (for qubits two and four) or $p_g + 2p_{stor}$ (for qubits one and three). Although only phase errors can feed back, the fault-tolerant network does not treat σ_x and σ_z errors symmetrically, so in order to be safe, I will consider the worst case where every error is of the most dangerous type. However, in no case can an error in the test qubit feed back into the data qubits, so I have not included errors from this source.

Now, we can construct a network for error syndrome measurement such that each data qubit contributes to at most four syndrome bits. In addition, two Hadamard rotations are necessary. Therefore, the process of syndrome measurement introduces at most an additional probability $6p_g + 2p_{stor}$ of error. To this, we must add the probability of an error feeding back, plus the accumulation

of storage errors while we prepare the cat state and measure the ancilla. There is only waiting time for the preparation of the first cat state and measurement of the last one, since preparation and measurement of the other cat states is taking place in parallel. Feedback is a more serious problem, but we can arrange it so that no data qubit interacts with more than two ancilla qubits with error rate $2p_g + p_{stor}$, so the total feedback is at most $6p_g + 6p_{stor}$. Therefore,

$$\begin{aligned} p_{EC} &= (6p_g + 6p_{stor}) + (6p_g + 2p_{stor}) + (6 + t_{prep} + t_{meas})p_{stor} \quad (6.4) \\ &= 12p_g + (14 + t_{prep} + t_{meas})p_{stor}. \quad (6.5) \end{aligned}$$

Now, in order to measure a qubit encoded at some level, it is sufficient to measure all of the constituent qubits. At level one, this gives us some seven-bit string which is a codeword of the classical Hammng code (possibly with some errors). Whether it is a codeword of even or odd parity will tell us whether the corresponding level one qubit is $|0\rangle$ or $|1\rangle$. We can continue to do this at all levels, using classical error correction at each level to correct any errors in individual bits. This will, in general, require a fair amount of classical computation. However, I will assume that classical computation is much faster than quantum computation when it can perform the same task, and that in the regime of interest, $t_{meas} = 1$. No matter what the speed of the classical computer, eventually $t_{meas}^{(j)}$ will become greater than one, but due to the rapid convergence of the double exponential, this will have a very small effect on the threshhold.

Preparing encoded $|0\rangle$ states at level $j-1$ does take a fair amount of time, however. Furthermore, the amount of time will increase with level. One way to prepare encoded 0 states reliably is by performing a full error correction cycle for the code with the addition of the $\overline{Z}$ operator $\sigma_{z5}\sigma_{z6}\sigma_{z7}$. The input state can be anything. The time to do this is at most $4(t_{EC} + 1)$. Recall that we must get the same error syndrome twice before we trust it. If there is an error in the second syndrome measurement, we may have to measure the syndrome twice more, for a total of four times. The chance of two errors is lower order, and therefore we ignore it.

The time for one error correction cycle is $t_{EC} = 14 + t_{prep} + t_{meas}$, so $t_{prep}^{(j)} = 64 + 4t_{prep}^{(j-1)}$. In order to cut down the growth rate with level, I will assume we can plan ahead enough to prepare the ancillas for later syndrome measurements while measuring the earlier syndromes. Then $t_{prep}^{(j)} = 43 + t_{prep}^{(j-1)}$. Recalling that $t_{prep}^{(0)} = 0$, we then get $t_{prep}^{(j)} = 43j$. One benefit of preparing states using error correction is that the chance of residual error is minimal. I will take $p_{prep}^{(j)} = 0$ where it matters.

Finally, we get the result for p_{EC}. The t_{prep} that contributes is actually $t_{prep}^{(j-1)}$, so

$$p_{EC}^{(j)} = 12p_g^{(j-1)} + [15 + 43(j-1)]\, p_{stor}^{(j-1)}. \quad (6.6)$$

Therefore,

$$p_g^{(j)} = 21\left[(p_g^{(j-1)})^2 + 4p_g^{(j-1)}p_{EC} + 8p_{EC}^2\right] \quad (6.7)$$

$$= 25221\,(p_g^{(j-1)})^2 + [61740 + 176988(j-1)]\,p_g^{(j-1)}p_{stor}^{(j-1)} + \left[37800 + 216720(j-1) + 310632(j-1)^2\right](p_{stor}^{(j-1)})^2 \quad (6.8)$$

and

$$p_{stor}^{(j)} = 21\left[(p_{stor}^{(j-1)})^2 + 4p_{stor}^{(j-1)}p_{EC} + 8p_{EC}^2\right] \quad (6.9)$$

$$= 24192\,(p_g^{(j-1)})^2 + [61488 + 173376(j-1)]\,p_g^{(j-1)}p_{stor}^{(j-1)} + \left[39081 + 220332(j-1) + 310632(j-1)^2\right](p_{stor}^{(j-1)})^2. \quad (6.10)$$

Note a number of things here. If we perform error correction after every time step, whether it has a gate or not, the storage error rate and gate error rate at the next level will actually be dominated by the error rate of error correction, so they will be very close. Also, at levels beyond the first, the error rate is dominated by storage errors occuring while we wait around encoding the ancilla qubits for error correction. Therefore, the algorithm will benefit greatly from a more rapid preparation algorithm, a better ability to plan ahead, or both.

First, consider the limit in which storage errors are negligable. In this case, we do not perform error correction after a step without a gate. Therefore, $p_{stor}^{(j)} = 0$ at all levels. Then, $p_g^{(j)} = 25221\,(p_g^{(j-1)})^2$, and the threshhold for a computation involving only operations from $N(\mathcal{G})$ is $p_{thresh} = 1/25200 = 4.0 \times 10^{-5}$. A second limit would be when $p_g^{(0)} = p_{stor}^{(0)}$, so there are no gate errors beyond the simple storage error in the same time step. Then they should be equal at all other levels, as well. Then

$$p_{stor}^{(j)} = \left[124761 + 393708(j-1) + 310632(j-1)^2\right](p_{stor}^{(j-1)})^2. \quad (6.11)$$

Then $p_{stor}^{(1)} = 124800\,(p_{stor}^{(0)})^2$, $p_{stor}^{(2)} = 8.3 \times 10^5\,(p_{stor}^{(1)})^2$, and $p_{stor}^{(3)} = 2.2 \times 10^6\,(p_{stor}^{(2)})^2$. For higher j, we approximate

$$p_{stor}^{(j)} = 3.1{\times}10^5\,(j{-}1)^2(p_{stor}^{(j-1)})^2 = \left[(j-1)^2 p_{stor}^{(j-1)}/(3.2 \times 10^{-6})\right] p_{stor}^{(j-1)}. \quad (6.12)$$

To get continual improvement, it is sufficient for $p_{stor}^{(j)}/p_{stor}^{(j-1)} < (j-1)^2/j^2$. This will mean $p_{stor}^{(j)} \leq \frac{9}{j^2}p_{stor}^{(3)}$. It suffices for $p_{stor}^{(4)} = \frac{9}{16}p_{stor}^{(3)}$, so $p_{stor}^{(3)} = \frac{1}{16}(3.2 \times 10^{-6})$. Following this back, we find that for only storage errors, the threshhold is roughly $p_{thresh} = 2.2 \times 10^{-6}$, or slightly more than an order of magnitude worse than for just gate errors.

Let us consider another case. Suppose we can plan ahead well, and prepare ancillas for error correction just in time for when they are needed. Then $p_{EC} = 12p_g + 9p_{stor}$, and

$$p_g^{(j)} = 25221\,(p_g^{(j-1)})^2 + 37044\,p_g^{(j-1)}p_{stor}^{(j-1)} + 13608\,(p_{stor}^{(j-1)})^2 \quad (6.13)$$

$$p_{stor}^{(j)} = 24192\,(p_g^{(j-1)})^2 + 37296\,p_g^{(j-1)}p_{stor}^{(j-1)} + 14385\,(p_{stor}^{(j-1)})^2. \quad (6.14)$$

For all practical purposes, for $j > 1$, $p_g^{(j)} = p_{stor}^{(j)} = p^{(j)} = 75873\,(p^{(j-1)})^2$. This means that the threshhold occurs at $p^{(1)} = 1/75873 = 1.3 \times 10^{-5}$. At the limit

$p_{stor}^{(0)} = 0$, we get a threshhold for p_g of $p_{thresh} = 2.3 \times 10^{-5}$. At the limit $p_g^{(0)} = p_{stor}^{(0)}$, we get a threshhold $p_{thresh} = 1.3 \times 10^{-5}$.

Finally, suppose we do not do error correction after every step, but instead attempt to optimize the number of steps N between error corrections. Then the chance of error in N steps is $Np_g^{(j-1)}$ or $Np_{stor}^{(j-1)}$, and equations (6.1) and (6.2) become

$$Np_g^{(j)} = 21\left[N^2(p_g^{(j-1)})^2 + 4Np_g^{(j-1)}p_{EC} + 8p_{EC}^2\right] \tag{6.15}$$

$$Np_{stor}^{(j)} = 21\left[N^2(p_{stor}^{(j-1)})^2 + 4Np_{stor}^{(j-1)}p_{EC} + 8p_{EC}^2\right]. \tag{6.16}$$

The values $p_g^{(j)}$ and $p_{stor}^{(j)}$ now represent average error rates, rather than strict error rates per step. As long as we do gates from $N(\mathcal{G})$ only or storage only, these values will be accurate representations, but if we mix and match, the story will be a bit different. Optimizing with respect to N gives us

$$-\frac{21}{N^2}\left[N^2(p_g^{(j-1)})^2 + 4Np_g^{(j-1)}p_{EC} + 8p_{EC}^2\right] \tag{6.17}$$

$$+\frac{21}{N}\left[2N(p_g^{(j-1)})^2 + 4p_g^{(j-1)}p_{EC}\right] = 0 \tag{6.18}$$

$$N^2(p_g^{(j-1)})^2 + 4Np_g^{(j-1)}p_{EC} + 8p_{EC}^2 = 2N^2(p_g^{(j-1)})^2 + 4Np_g^{(j-1)}p_{EC} \tag{6.19}$$

$$N^2(p_g^{(j-1)})^2 - 8p_{EC}^2 = 0 \tag{6.20}$$

$$N = \sqrt{8}\,(p_{EC}/p_g^{(j-1)}). \tag{6.21}$$

The same is true for storage steps. The optimum number of steps makes the accumulated chance of error during gates $\sqrt{8}$ times the chance of error during error correction. Plugging in this value for N gives us

$$p_g^{(j)} = \frac{21}{N}(16 + 8\sqrt{2})p_{EC}^2. \tag{6.22}$$

Assuming no storage errors, $p_{EC} = 12p_g^{(j-1)}$, so $N = 34$ and $p_g^{(j)} = 2.4 \times 10^3\,(p_g^{(j-1)})^2$, so the threshhold is $p_{thresh} = 4.1 \times 10^{-4}$. In practice, we will not be able to perform error correction after exactly 34 gates, since there will be Toffoli gates occuring at possibly inconvenient times, but if we get close to the right frequency of error correction, the actual threshhold will not be too much worse than this.

6.3 Toffoli Gate Threshhold

To figure out the recursion relation for the Toffoli gate, look at figure 6.2, which summarizes the construction in section 5.7. I will follow each qubit individually

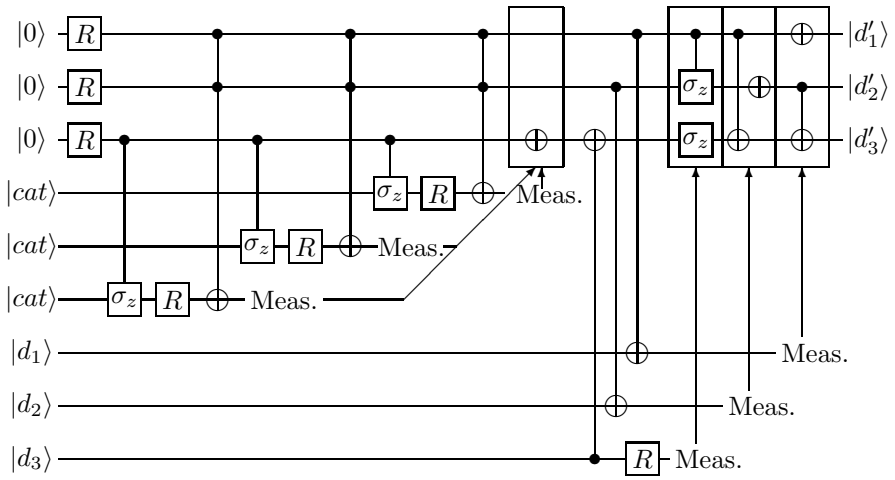

Figure 6.2: The Toffoli gate construction. Each line represents seven qubits at the next lower level.

in order to figure out the final chance of error for that qubit. This is a construction for the Toffoli gate at level $j+1$. I will assume we do error correction on all three ancilla qubits only after the Toffoli gate is completed. All three ancilla qubits start out with $p_{prep}^{(j+1)}$ chance of error from preparing encoded $|0\rangle$'s. There are actually two types of relevant encoding errors. There can be errors remaining at lower levels. Since we have just done an error correction cycle, I assume that the number of residual errors is negligable. There is also a chance that the qubit will not be an encoded $|0\rangle$, but some other encoded state. This would count as a complete failure of the Toffoli gate, since it would produce a real error at level $j+1$. However, I will assume that the chance of this happening is also zero.

Assume the chance of a remaining bit flip error in a cat state is p_{cat} and the time to make a cat state is t_{cat}. Only bit flip errors feed back from the cat states to the ancilla qubits in this network. Let A_1, A_2, and A_3 be the accumulated chances of error in the three ancilla qubits. First we have a Hadamard transform on all three of these qubits. After the first gate in the ancilla construction, $A_3 = t_{cat}\, p_{stor}^{(j)} + 2p_g^{(j)} + p_{cat}$. It will have to sit around an additional $1 + t_{Tof}^{(j)}$ time steps before the interaction with the next cat state begins. Thus, after the first cat state is finished, $A_3 = (t_{cat} + t_{Tof}^{(j)} + 1)p_{stor}^{(j)} + 2p_g^{(j)} + p_{cat}$. By the time of the Toffoli gate with the first two ancilla qubits, the chance of errors in the cat state which can feed back into the main ancilla is at most $p_{cat} + 2p_g^{(j)}$. The first two ancilla qubits have already waited a time $t_{cat} + 2$, so the overall chance

of errors in the first two ancilla qubits is

$$A_1 = A_2 = (t_{cat} + 2)p^{(j)}_{stor} + p_{cat} + 3p^{(j)}_g + p^{(j)}_{Tof}. \tag{6.23}$$

We repeat the cat state interaction two more times with new cat states, which we have been preparing in parallel with the first cat state. Therefore, we only need $2 + t^{(j)}_{Tof}$ more time steps for each interaction, introducing the same amount of error as the equivalent steps in the first interaction. We must also measure the cat states. We can do it in the basis they end up in; we check for odd or even parity. If two of the three cat states have odd parity, we decide the ancilla is in the state $|B\rangle$, and we perform σ_x on the third ancilla qubit. This process will take an additional $t^{(j)}_{meas} + 1$ time units. After the ancilla creation is completed, the chances of error on the three qubits are

$$\begin{aligned}
A_1 &= \left(t_{cat} + t^{(j)}_{meas} + 7\right) p^{(j)}_{stor} + 3p_{cat} + 7p^{(j)}_g + 3p^{(j)}_{Tof} && (6.24)\\
A_2 &= \left(t_{cat} + t^{(j)}_{meas} + 7\right) p^{(j)}_{stor} + 3p_{cat} + 7p^{(j)}_g + 3p^{(j)}_{Tof} && (6.25)\\
A_3 &= \left(t_{cat} + t^{(j)}_{meas} + 3t^{(j)}_{Tof} + 3\right) p^{(j)}_{stor} + 3p_{cat} + 5p^{(j)}_g. && (6.26)
\end{aligned}$$

The whole ancilla construction has taken a time $t_{cat} + t^{(j)}_{meas} + 3t^{(j)}_{Tof} + 7$, during which time the data qubits have been accumulating storage errors. I assume here that $t_{cat} \geq t^{(j)}_{prep} + 1$.

Now we perform the CNOTs between the data qubits and the ancilla qubits. Again we make the conservative assumption that all of the accumulated chance of error on the data qubits feeds into the ancilla qubits. Thus,

$$\begin{aligned}
A_1 &= \left(2t_{cat} + 2t^{(j)}_{meas} + 3t^{(j)}_{Tof} + 14\right) p^{(j)}_{stor} + 3p_{cat} + 8p^{(j)}_g + 3p^{(j)}_{Tof} && (6.27)\\
A_2 &= \left(2t_{cat} + 2t^{(j)}_{meas} + 3t^{(j)}_{Tof} + 14\right) p^{(j)}_{stor} + 3p_{cat} + 8p^{(j)}_g + 3p^{(j)}_{Tof} && (6.28)\\
A_3 &= \left(2t_{cat} + 2t^{(j)}_{meas} + 6t^{(j)}_{Tof} + 10\right) p^{(j)}_{stor} + 3p_{cat} + 6p^{(j)}_g. && (6.29)
\end{aligned}$$

Now we measure σ_z for the first two data qubits and σ_x for the third data qubit. We will add one time step for the Hadamard rotation on the third data qubit, plus $t^{(j)}_{meas}$ to measure. We should include a chance of the Toffoli gate failing because of the wrong result on one of these measurements, but I will assume that chance is small compared to the accumulated errors on the ancilla qubits. Before we start doing the conditional operations to convert the ancilla states to complete the transfer of the data, the chances of error are

$$\begin{aligned}
A_1 &= \left(2t_{cat} + 3t^{(j)}_{meas} + 3t^{(j)}_{Tof} + 15\right) p^{(j)}_{stor} + 3p_{cat} + 8p^{(j)}_g + 3p^{(j)}_{Tof} && (6.30)\\
A_2 &= \left(2t_{cat} + 3t^{(j)}_{meas} + 3t^{(j)}_{Tof} + 15\right) p^{(j)}_{stor} + 3p_{cat} + 8p^{(j)}_g + 3p^{(j)}_{Tof} && (6.31)\\
A_3 &= \left(2t_{cat} + 3t^{(j)}_{meas} + 6t^{(j)}_{Tof} + 11\right) p^{(j)}_{stor} + 3p_{cat} + 6p^{(j)}_g. && (6.32)
\end{aligned}$$

I will now assume that all three operations are necessary; this is the worst case, and usually there will be fewer gate errors. The first conditional operation interacts ancilla qubits one and two, giving

$$A_1 = \left(4t_{cat} + 6t_{meas}^{(j)} + 6t_{Tof}^{(j)} + 30\right) p_{stor}^{(j)} + 6p_{cat} + 17p_g^{(j)} + 6p_{Tof}^{(j)} \quad (6.33)$$

$$A_2 = \left(4t_{cat} + 6t_{meas}^{(j)} + 6t_{Tof}^{(j)} + 30\right) p_{stor}^{(j)} + 6p_{cat} + 17p_g^{(j)} + 6p_{Tof}^{(j)} \quad (6.34)$$

$$A_3 = \left(2t_{cat} + 3t_{meas}^{(j)} + 6t_{Tof}^{(j)} + 11\right) p_{stor}^{(j)} + 3p_{cat} + 7p_g^{(j)}. \quad (6.35)$$

The second conditional operation interacts ancilla qubits one and three, so

$$A_1 = \left(6t_{cat} + 9t_{meas}^{(j)} + 12t_{Tof}^{(j)} + 41\right) p_{stor}^{(j)} + 9p_{cat} + 25p_g^{(j)} + 6p_{Tof}^{(j)} \quad (6.36)$$

$$A_2 = \left(4t_{cat} + 6t_{meas}^{(j)} + 6t_{Tof}^{(j)} + 30\right) p_{stor}^{(j)} + 6p_{cat} + 18p_g^{(j)} + 6p_{Tof}^{(j)} \quad (6.37)$$

$$A_3 = \left(6t_{cat} + 9t_{meas}^{(j)} + 12t_{Tof}^{(j)} + 34\right) p_{stor}^{(j)} + 9p_{cat} + 25p_g^{(j)} + 6p_{Tof}^{(j)}. \quad (6.38)$$

The third operation interacts the second and third ancilla qubits. Much of the error from the first and second ancilla qubits has already been introduced into the third qubit, so there is no need to add it again. In fact, much of it may cancel out instead. However, I assume it remains. The only new error for the third ancilla qubit is the gate error on the second qubit from the previous operation plus the gate error for this operation. Thus,

$$A_1 = \left(6t_{cat} + 9t_{meas}^{(j)} + 12t_{Tof}^{(j)} + 41\right) p_{stor}^{(j)} + 9p_{cat} + 26p_g^{(j)} + 6p_{Tof}^{(j)} \quad (6.39)$$

$$A_2 = \left(6t_{cat} + 9t_{meas}^{(j)} + 12t_{Tof}^{(j)} + 41\right) p_{stor}^{(j)} + 9p_{cat} + 27p_g^{(j)} + 6p_{Tof}^{(j)} \quad (6.40)$$

$$A_3 = \left(6t_{cat} + 9t_{meas}^{(j)} + 12t_{Tof}^{(j)} + 41\right) p_{stor}^{(j)} + 9p_{cat} + 27p_g^{(j)} + 6p_{Tof}^{(j)}. \quad (6.41)$$

The overall chance of error on a single one of the new data qubits after the full Toffoli gate construction is thus

$$\left(6t_{cat} + 9t_{meas}^{(j)} + 12t_{Tof}^{(j)} + 41\right) p_{stor}^{(j)} + 9p_{cat} + 27p_g^{(j)} + 6p_{Tof}^{(j)}. \quad (6.42)$$

The time taken to perform this Toffoli gate is

$$t_{Tof}^{(j+1)} = t_{cat} + 2t_{meas}^{(j)} + 3t_{Tof}^{(j)} + 12. \quad (6.43)$$

After error correction, the chance of a real error at level $j+1$ is

$$p_{Tof}^{(j+1)} = 21 \left\{ \left[\left(6t_{cat} + 9t_{meas}^{(j)} + 12t_{Tof}^{(j)} + 41\right) p_{stor}^{(j)} + 9p_{cat} + 27p_g^{(j)} + 6p_{Tof}^{(j)} \right]^2 + 4\left[\left(6t_{cat} + 9t_{meas}^{(j)} + 12t_{Tof}^{(j)} + 41\right) p_{stor}^{(j)} + 9p_{cat} + 27p_g^{(j)} + 6p_{Tof}^{(j)} \right] p_{EC} + 8p_{EC}^2 \right\}. \quad (6.44)$$

In order to simplify the recursion relation so that it is easily solvable, I will only investigate the limit where there are no storage errors. In this case, it makes sense to verify the cat state used in the construction until the chance of errors in it is negligable. Therefore, I will also assume that $p_{cat} = 0$. Then the recursion relation for the Toffoli gate becomes

$$\begin{aligned} p_{Tof}^{(j+1)} &= 21\left[(27p_g^{(j)} + 6p_{Tof}^{(j)})^2 + 4\,(27p_g^{(j)} + 6p_{Tof}^{(j)})\,p_{EC} + 8p_{EC}^2\right] && (6.45)\\ &= 66717\,(p_g^{(j)})^2 + 12852\,p_g^{(j)}p_{Tof}^{(j)} + 756\,(p_{Tof}^{(j)})^2. && (6.46)\end{aligned}$$

Recall that in this limit, $p_g^{(j)} = 25221\,(p_g^{(j-1)})^2$, so

$$p_g^{(j)} = 25200^{a(j)}(p_g^{(0)})^{2^j}, \tag{6.47}$$

where $a(j+1) = 1 + 2a(j)$, with $a(1) = 1$. Therefore, $a(j) = 2^j - 1$, and

$$\begin{aligned} p_g^{(j)} &= 4.0\times10^{-5}\left[p_g^{(0)}/(4.0\times10^{-5})\right]^{2^j} && (6.48)\\ &= p_{thresh}\left(p_g^{(0)}/p_{thresh}\right)^{2^j}. && (6.49)\end{aligned}$$

Writing $\epsilon = p_g^{(0)}/p_{thresh}$, we have

$$p_{Tof}^{(j+1)} = 1.1\times10^{-4}\,\epsilon^{2^{j+1}} + 0.51\,\epsilon^{2^j}p_{Tof}^{(j)} + 756\,(p_{Tof}^{(j)})^2. \tag{6.50}$$

The first term is often negligable compared to the second term, in which case

$$p_{Tof}^{(j+1)} = \left(0.51\,\epsilon^{2^j} + 756\,p_{Tof}^{(j)}\right)p_{Tof}^{(j)}. \tag{6.51}$$

In the limit where ϵ is small, we find a threshhold value of $p_{Tof}^{(0)} = 1/756 = 1.3\times10^{-3}$.

Even when ϵ is fairly large, the presence of Toffoli gates does not present much of a problem for the threshhold. For instance, if we demand that $p_{Tof}^{(0)} = p_g^{(0)} = \epsilon p_{thresh}$, then

$$\begin{aligned} p_{Tof}^{(1)} &= 1.1\times10^{-4}\,\epsilon^2 + [12852\,p_{thresh}\epsilon + 756\,p_{thresh}\epsilon]\,p_{Tof}^{(0)} && (6.52)\\ &\approx 1.3\times10^{-4}\,\epsilon^2, && (6.53)\\ p_{Tof}^{(2)} &= 1.1\times10^{-4}\,\epsilon^4 + \left[0.51\,\epsilon^2 + 756\,(1.3\times10^{-4})\,\epsilon^2\right]p_{Tof}^{(1)} && (6.54)\\ &= 1.9\times10^{-4}\,\epsilon^4, && (6.55)\\ p_{Tof}^{(3)} &= 1.1\times10^{-4}\,\epsilon^8 + \left[0.51\,\epsilon^4 + 756\,(1.9\times10^{-4})\,\epsilon^4\right]p_{Tof}^{(2)}, && (6.56)\\ &= 2.3\times10^{-4}\,\epsilon^8. && (6.57)\end{aligned}$$

If we let $\epsilon^4 = 1.9/2.3$, so $\epsilon \approx 0.95$, then $p_{Tof}^{(3)} = p_{Tof}^{(2)}$, and as we add levels of concatenation, the Toffoli gate error gate will begin to improve. Therefore, the presence of Toffoli gates with the same physical error rate as other gates causes less than a 5% reduction in the threshhold.

Chapter 7

Bounds on Quantum Error-Correcting Codes

7.1 General Bounds

The question of how efficient an error-correcting code of a given block size can be made in terms of both encoded qubits and distance is an interesting and important question in the theories of both classical and quantum error correction. In the classical theory, only upper and lower bounds exist on the efficiency of codes that must have a given minimum distance between all codewords. The true, achievable bounds on such codes are unknown. Better understood in the classical case is the asymptotic efficiency of coding (where we only require that the code correct all likely errors). In the limit of infinite bits sent, we usually require the code to correct measure one of the errors occuring using some probability measure associated with the channel. Classically, Shannon's theorem tells us what the achievable capacity of a channel is. No real quantum analogue of Shannon's theorem is known, despite extensive work on the subject [50, 51, 52].

One simple upper bound on the efficiency of quantum codes is the quantum Hamming bound [53]. For a nondegenerate code with basis codewords $|\psi_i\rangle$ and possible errors E_a, all of the states $E_a|\psi_i\rangle$ are linearly independent for all a and i. If the code uses n qubits, there can only be 2^n linearly indepedent vectors in the Hilbert space, so the number of errors times the number of codewords must be less than or equal to 2^n. If the code corrects all errors of weight t or less and encodes k qubits, this means

$$\sum_{j=0}^{t} 3^j \binom{n}{j} 2^k \leq 2^n. \tag{7.1}$$

There are $\binom{n}{j}$ ways to choose j qubits to be affected by j errors and 3^j ways these errors can be tensor products of σ_x, σ_y, and σ_z. This bound is completely analogous to the classical Hamming bound, with two differences: the quantum

bound has a factor of 3^j reflecting the additional quantum-mechanical degrees of freedom; and the quantum bound only applies to nondegenerate codes. The distinction between degenerate and nondegenerate codes is a purely quantum-mechanical distinction; there are no classical degenerate codes. It is unknown whether there are any degenerate codes that exceed the quantum Hamming bound (7.1).

If we let the block size n grow arbitrarily large, we should also increase the expected number of errors. Consider the depolarizing channel, which is equally likely to have σ_x, σ_y, and σ_z errors. Suppose there is a probability p of having one of these errors on a given qubit and $1-p$ of having no error. The expected number of errors on a block of size n is $t = np$. The number of likely errors will be about the number of errors of length t, so the quantum Hamming bound becomes

$$3^{np} \binom{n}{np} 2^k \leq 2^n. \tag{7.2}$$

Taking the logarithm and rearranging gives us

$$\frac{k}{n} \leq 1 - p \log_2 3 - H(p). \tag{7.3}$$

Again, $H(x) = -x \log_2 x - (1-x) \log_2 (1-x)$, as with the asymptotic form of the classical Hamming bound (1.16). As with the classical case, we can achieve the quantum Hamming bound by using random codes. Unlike the classical case, this is not always the most efficient use of the channel, so (7.3) does not give the actual channel capacity of the quantum channel. I will discuss this question in greater detail in section 7.6.

For minimum distance codes, it is not in general possible to achieve the quantum Hamming bound. We can set a lower bound, the quantum Gilbert-Varshamov bound. Recall that

$$\langle \psi_i | E_a^\dagger E_b | \psi_j \rangle = C_{ab} \delta_{ij} \tag{7.4}$$

for a quantum code correcting errors $\{E_a\}$ with basis states $|\psi_i\rangle$. The matrix C_{ab} is Hermitian, but is further constrained by the algebraic relationships of the operators $E_a^\dagger E_b$. It is better to consider C_{ab} as a function of operators $O = E_a^\dagger E_b$. When the possible errors are all operators of up to weight t, O can be any operator of weight $\leq 2t$. Slightly more generally, for a code of distance d, O is any operator of weight less than d. Therefore, the statement

$$\langle \psi | E_a^\dagger E_b | \psi \rangle = C_{ab} \tag{7.5}$$

is actually

$$N = \sum_{j=0}^{d-1} 3^j \binom{n}{j} \tag{7.6}$$

constraints on the state $|\psi\rangle$. For generic C_{ab} (satisfying the appropriate algebraic constraints) and generic linear subspace V with dimension larger than N, there will be states $|\psi\rangle$ satisfying equation (7.5).

Suppose we choose generic C_{ab} and a generic state $|\psi_1\rangle$ satisfying (7.5). Now restrict attention to the subspace orthogonal to $|\psi_1\rangle$ and to all $O|\psi_1\rangle$ for operators O of weight less than d. For an n-qubit Hilbert space, this subspace has dimension $2^n - N$. Choose a generic state $|\psi_2\rangle$ in this subspace satisfying (7.5). Now restrict attention to the subspace orthogonal to both $O|\psi_1\rangle$ and $O|\psi_2\rangle$. We can again pick $|\psi_3\rangle$ in this subspace satisfying (7.5), and so on. Choose $|\psi_i\rangle$ orthogonal to all $O|\psi_j\rangle$ ($j \leq i-1$) and satisfying (7.5). We can continue doing this as long as

$$\sum_{j=0}^{d-1} 3^j \binom{n}{j} i < 2^n. \tag{7.7}$$

Therefore, we can always find a distance d quantum code encoding k qubits in n qubits satisfying

$$\sum_{j=0}^{d-1} 3^j \binom{n}{j} 2^k \geq 2^n. \tag{7.8}$$

This is the quantum Gilbert-Varshamov bound. In the limit where $t = pn = d/2$, with n large, this becomes

$$\frac{k}{n} \geq 1 - 2p \log_2 3 - H(2p). \tag{7.9}$$

The quantum Hamming bound only limits the efficiency of nondegenerate codes. For degenerate codes, we can still set a bound, but it will not be as restrictive. For an $[n, k, d]$ code, we can choose any $d-1$ qubits and remove them. The remaining $n - d + 1$ qubits must contain enough information to reconstruct not only the 2^k possible codewords, but the state of the missing qubits as well. Because the missing qubits can be any qubits, we can choose them to have maximum entropy. Then

$$n - d + 1 \geq d - 1 + k \tag{7.10}$$

$$n \geq 2(d-1) + k. \tag{7.11}$$

This is the Knill-Laflamme bound [16, 54]. It is a quantum analog of the classical Singleton bound. A code to correct t errors must have distance $d = 2t + 1$, so for such a code, $n \geq 4t + k$. This bound holds for any code with a given minimum distance, whether it is degenerate or nondegenerate. For instance, this bound demonstrates that the smallest one-error-correcting quantum code uses five qubits.

7.2 Weight Enumerators and Linear Programming Bounds

In the classical theory of error-correcting codes, the distribution of codeword weights contains a great deal of information about the code. This distribution

is often encoded in the coefficients of a polynomial, and algebraic relationships between these polynomials, known as *weight enumerators*, can be very useful for setting bounds on classical codes. Many of the same ideas can be adapted for use with quantum error-correcting codes [23, 55, 56, 57].

Let A_d be the number of elements of the stabilizer S with weight d, and let B_d be the number of elements of $N(S)$ with weight d (ignoring overall phases). Note that $B_d \geq A_d \geq 0$. Define polynomials

$$A(z) = \sum_{d=0}^{n} A_d z^d \tag{7.12}$$

$$B(z) = \sum_{d=0}^{n} B_d z^d. \tag{7.13}$$

$A_0 = B_0 = 1$ always. For a code of distance d, $B_{d'} = A_{d'}$ for all $d' < d$. For a nondegenerate code, $B_{d'} = A_{d'} = 0$ for $d' < d$. A degenerate code has $B_{d'} = A_{d'} > 0$ for at least one $d' < d$. $A(z)$ and $B(z)$ are the weight enumerators of S and $N(S)$.

The polynomials $A(z)$ and $B(z)$ satisfy the quantum MacWilliams identity [55]:

$$B(z) = \frac{1}{2^{n-k}}(1+3z)^n A\left(\frac{1-z}{1+3z}\right). \tag{7.14}$$

In other words,

$$\sum_{d=0}^{n} B_d z^d = \frac{1}{2^{n-k}} \sum_{d=0}^{n} A_d (1-z)^d (1+3z)^{n-d}. \tag{7.15}$$

Matching coefficients of z^d, we find

$$B_d = \frac{1}{2^{n-k}} \sum_{d'=0}^{n} \left[\sum_{s=0}^{d} (-1)^s 3^{d-s} \binom{d'}{s} \binom{n-d'}{d-s} \right] A_{d'}. \tag{7.16}$$

To prove this, note that an operator $E \in \mathcal{G}$ of weight d will either commute with every operator $M \in S$ or it will commute with exactly half of the operators in S. Therefore, if we sum

$$\sum_{M \in S} (-1)^{f_M(E)}, \tag{7.17}$$

we will get zero if $E \notin N(S)$ and 2^{n-k} if $E \in N(S)$ (recall that $f_M(E)$ is 0 if M and E commute and 1 if they do not). Therefore, we can write B_d as follows:

$$B_d = \frac{1}{2^{n-k}} \sum_{E} \sum_{M \in S} (-1)^{f_M(E)}, \tag{7.18}$$

where the sum over E is taken over all $E \in \mathcal{G}$ of weight d. We reverse the order of summation and break up the sum over M to the sum over d' and the sum

over $M \in S$ of weight d' to get

$$B_d = \frac{1}{2^{n-k}} \sum_{d'=0}^{n} \sum_{M} \sum_{E} (-1)^{f_M(E)}. \tag{7.19}$$

Now, any given M and E will both act nontrivially on some set of s qubits. Of those s, they will act as different Pauli matrices on t qubits and as the same Pauli matrix on $s-t$ qubits. Now,

$$(-1)^{f_M(E)} = (-1)^t. \tag{7.20}$$

The number of operators E that agree with M on $s-t$ qubits and disagree on t qubits is

$$1^{s-t} 2^t 3^{d-s} \binom{s}{t} \binom{d'}{s} \binom{n-d'}{d-s}. \tag{7.21}$$

Note that this does not depend on M. Thus,

$$\begin{aligned}
B_d &= \frac{1}{2^{n-k}} \sum_{d'=0}^{n} \sum_{M} \sum_{s=0}^{d} \sum_{t=0}^{s} \left[1^{s-t} (-2)^t \binom{s}{t} \right] 3^{d-s} \binom{d'}{s} \binom{n-d'}{d-s} && (7.22)\\
&= \frac{1}{2^{n-k}} \sum_{d'=0}^{n} \sum_{M} \sum_{s=0}^{d} (1-2)^s 3^{d-s} \binom{d'}{s} \binom{n-d'}{d-s} && (7.23)\\
&= \frac{1}{2^{n-k}} \sum_{d'=0}^{n} \sum_{M} \sum_{s=0}^{d} (-1)^s 3^{d-s} \binom{d'}{s} \binom{n-d'}{d-s} && (7.24)\\
&= \frac{1}{2^{n-k}} \sum_{d'=0}^{n} \left[\sum_{s=0}^{d} (-1)^s 3^{d-s} \binom{d'}{s} \binom{n-d'}{d-s} \right] A_{d'}. && (7.25)
\end{aligned}$$

This proves the quantum MacWilliams identity (7.14) for stabilizer codes. The coefficients A_d and B_d can also be defined for non-stabilizer codes, and equation (7.14) will still hold, so any bounds derived strictly from the quantum MacWilliams identity will hold for any quantum code, not just stabilizer codes. For any code of distance d, the coefficients A_d and B_d satisfy the additional constraints

$$\begin{aligned}
B_0 &= A_0 = 1 && (7.26)\\
B_{d'} &= A_{d'} \ (d' < d) && (7.27)\\
B_{d'} &\geq A_{d'} \geq 0 \ (\forall\, d'). && (7.28)
\end{aligned}$$

For a nondegenerate code, $A_{d'} = B_{d'} = 0$ for $d' < d$. These constraints along with equation (7.14) restrict the allowed values of A_d and B_d. The constraints are all linear, so standard linear programming techniques will find solutions. If there are no possible integer values of A_d and B_d satisfying all of the constraints, there is no $[n, k, d]$ code. Otherwise, the possible solutions

will give us parameters of possible codes. For instance, applying the constraints for a $[5,1,3]$ code produces the unique solution $A_i = (1,0,0,0,15,0)$ and $B_i = (1,0,0,30,15,18)$ [55]. Therefore, the usual five-qubit code is essentially the only $[5,1,3]$ code. There are thus no degenerate five-qubit codes.

Even tighter linear programming bounds than those produced by the quantum MacWilliams identity are possible. This can be done using the quantum shadow enumerator [23]. The *shadow* $Sh(S)$ of a code S is defined as the set of $E \in \mathcal{G}$ satisfying

$$f_M(E) \equiv \mathrm{wt}(M) \pmod 2 \tag{7.29}$$

for all $M \in S$ (where $\mathrm{wt}(M)$ is the weight of M). Define S_d to be the number of elements of $Sh(S)$ of weight d (again, ignoring overall phases), and

$$S(z) = \sum_{d=0}^{n} S_d z^d. \tag{7.30}$$

$S(z)$ is the *shadow enumerator* of S. Then

$$S(z) = \frac{1}{2^{n-k}}(1+3z)^n A\left(\frac{z-1}{1+3z}\right). \tag{7.31}$$

If S contains only operators of even weight, then $E \in Sh(S)$ iff $f_M(E) = 0$ for all $M \in S$, so $Sh(S) = N(S)$, and $S_d = B_d$. Furthermore, in this case, $A(z)$ is an even function, so

$$\begin{aligned} S(z) &= B(z) = \frac{1}{2^{n-k}}(1+3z)^n A\left(\frac{1-z}{1+3z}\right) && (7.32)\\ &= \frac{1}{2^{n-k}}(1+3z)^n A\left(\frac{z-1}{1+3z}\right). && (7.33) \end{aligned}$$

If S contains an element of odd weight, consider the subset $S' \subset S$ of even weight operators. Then S' has exactly 2^{n-k-1} elements. This is true because in order for $M, M' \in S$ to commute, they must overlap and disagree only on an even number of qubits. Thus, $\mathrm{wt}(MM') \equiv \mathrm{wt}(M) + \mathrm{wt}(M') \pmod 2$. The shadow of S is just $Sh(S) = N(S') - N(S)$. Let $B'(z)$ and $A'(z)$ be the weight enumerators of S' and $N(S')$. Then

$$\begin{aligned} S(z) &= B'(z) - B(z) && (7.34)\\ &= \frac{1}{2^{n-k-1}}(1+3z)^n A'\left(\frac{1-z}{1+3z}\right) - \frac{1}{2^{n-k}}(1+3z)^n A\left(\frac{1-z}{1+3z}\right) && (7.35)\\ &= \frac{1}{2^{n-k}}(1+3z)^n \left[2A'\left(\frac{1-z}{1+3z}\right) - A\left(\frac{1-z}{1+3z}\right)\right]. && (7.36) \end{aligned}$$

Now, $A'_d = A_d$ for even d and $A'_d = 0$ for odd d, so $A(z) + A(-z) = 2A'(z)$, and

$$S(z) = \frac{1}{2^{n-k}}(1+3z)^n A\left(\frac{z-1}{1+3z}\right). \tag{7.37}$$

Again, the shadow enumerator can be defined for non-stabilizer codes and satisfies the same relationship with $A(z)$ as for stabilizer codes. In both the stabilizer and non-stabilizer case, $S_d \geq 0$. Along with (7.31), this provides additional constraints for the linear programming bound restricting the parameters of any code. These bounds have been applied to all possible codes with $n \leq 30$ [23, 26]. Among other things, they show that the smallest possible distance five code is an $[11, 1, 5]$ code and that degenerate codes in this region all fall below the quantum Hamming bound. The shadow enumerator can also be used to show that any nondegenerate code on n qubits can correct at most $\lfloor \frac{n+1}{6} \rfloor$ errors [23].

7.3 Bounds on Degenerate Stabilizer Codes

It is still unknown whether there are any degenerate codes that exceed the limits set by the quantum Hamming bound, but for certain restricted cases, we can show that there are not. For codes using fewer than 30 qubits, the linear programming bounds of the previous section show this. In this section, I will show that the statement also is true for all stabilizer codes that correct one or two errors. The results can be extended slightly beyond stabilizer codes, but do not apply to the most general possible code.

For a one-error-correcting degenerate code, the stabilizer S will contain one or more operators of weight one or two. Weight one operators totally constrain a qubit and both the operator and the qubit can be eliminated, converting an $[n, k, d]$ code into an $[n-1, k, d]$. If the latter satisfies the quantum Hamming bound, the former will as well. Suppose there are l independent weight two operators $M_1, \ldots, M_l$ in S. Let D be the group generated by $M_1, \ldots, M_l$. Note that $S - D$ will contain no operators of weight less than three. The weight two operators in D tell us which errors produce the same states. For instance, if $M_1 = \sigma_{z1}\sigma_{z2}$, $\sigma_{z1}|\psi\rangle = \sigma_{z2}|\psi\rangle$ for any codeword $|\psi\rangle$.

Any operator in $N(D)$ will take states fixed by D to states fixed by D. The total dimensionality of the subspace fixed by D is 2^{n-l}. Suppose that none of the operators in D acts on some qubit j. Then all of the three operators σ_{xj}, σ_{yj}, and σ_{zj} are in $N(D)$, and they are not degenerate. Therefore, they must produce orthogonal states in the subspace fixed by D for each basis codeword. There are always at least $n - 2l$ qubits not affected by D, since each generator of D can add at most two qubits. Therefore,

$$[1 + 3(n-2l)]\, 2^k \leq 2^{n-l} \tag{7.38}$$

$$k \leq n - l - \log_2[1 + 3(n-2l)]. \tag{7.39}$$

Recall that the quantum Hamming bound says that

$$k \leq n - \log_2(1 + 3n), \tag{7.40}$$

so (7.39) is more restrictive when

$$l + \log_2[1 + 3(n-2l)] \geq \log_2(1 + 3n) \tag{7.41}$$

$$l \geq \log_2 \left[\frac{1+3n}{1+3(n-2l)} \right] \tag{7.42}$$

$$= \log_2 \left[1 + \frac{6l}{1+3(n-2l)} \right]. \tag{7.43}$$

Assuming $n \geq 2l$, we see that the quantum Hamming bound will still hold if $l \geq \log_2(1+6l)$. This is true for $l \geq 5$. For $l = 4$, (7.43) holds for $n \geq 9$; for $l = 3$, it holds for $n \geq 7$. For $l = 2$, (7.43) holds for $n \geq 5$, and for $l = 1$, it holds for $n \geq 4$. The remaining possibilities with $n \geq 2l$ are ruled out by the linear programming bounds of section 7.2. On the other hand, if $l > n/2$, then $k \leq n - l \leq n/2$. For $n \geq 13$, the quantum Hamming bound is less restrictive than this, so in conjunction with the linear programming bounds, we can conclude that there are no distance three degenerate stabilizer codes that exceed the quantum Hamming bound.

We can make a similar argument for codes to correct two errors. Now let D be generated by the operators of weight four or less in S. There must be at least $n - 4l$ qubits that are unaffected by operators in D. All the possible weight one and two errors on those qubits give orthogonal states, so

$$\left[1 + 3(n-4l) + \frac{9}{2}(n-4l)(n-4l-1) \right] 2^k \leq 2^{n-l} \tag{7.44}$$

$$\left[1 - \frac{3}{2}n + \frac{9}{2}n^2 + 6l(1+12l-6n) \right] 2^l \leq 2^{n-k}. \tag{7.45}$$

The quantum Hamming bound will still hold if

$$\left[1 - \frac{3}{2}n + \frac{9}{2}n^2 + 6l(1+12l-6n) \right] 2^l \geq 1 - \frac{3}{2}n + \frac{9}{2}n^2 \tag{7.46}$$

$$\left[1 - \frac{6l(6n-12l-1)}{1-3n/2+9n^2/2} \right] 2^l \geq 1. \tag{7.47}$$

Now, $l(6n-12l-1) = -12[l^2 - (6n-1)l/12]$ is maximized for $l = (6n-1)/24$. That means (7.47) will be satisfied when

$$\left[1 - \frac{(6n-1)^2}{8-12n+36n^2} \right] 2^l \geq 1 \tag{7.48}$$

$$\frac{7}{8-12n+36n^2}\, 2^l \geq 1 \tag{7.49}$$

$$7 \cdot 2^{l-2} \geq 9n^2 - 3n + 2. \tag{7.50}$$

If this is true, the code will satisfy the quantum Hamming bound. If it is *not* true, then

$$l \leq 2 - \log_2 7 + \log_2(9n^2 - 3n + 2) \tag{7.51}$$

$$\leq 3 + 2\log_2 n. \tag{7.52}$$

Then $l(6n - 12l - 1) \leq 6nl \leq 6n(3 + 2\log_2 n)$, so equation (7.47) will again be satisfied when

$$\left[1 - \frac{6n(3 + 2\log_2 n)}{1 - 3n/2 + 9n^2/2}\right] 2^l \geq 1. \tag{7.53}$$

However, for $n \geq 30$,

$$\frac{6n(3 + 2\log_2 n)}{1 - 3n/2 + 9n^2/2} \leq 0.58, \tag{7.54}$$

so (7.47) will be satisfied for any l with $1 < l \leq n/4$ in the regime of interest. When $l = 1$, (7.47) becomes

$$1 - \frac{6(6n - 13)}{1 - 3n/2 + 9n^2/2} \geq 1/2. \tag{7.55}$$

However, for $n \geq 30$,

$$\frac{6(6n - 13)}{1 - 3n/2 + 9n^2/2} \leq 0.26, \tag{7.56}$$

so (7.47) is satisfied for $l = 1$ as well.

Therefore, we are left with $l > n/4$. Again, this implies that $k \leq n - l < 3n/4$. This is at least as restrictive than the quantum Hamming bound for $n \geq 52$. For $n = 31$, the quantum Hamming bound says $k \leq n - 13$. Therefore, for $31 \leq n \leq 51$, the only remaining region of interest, the code must have $l \leq n/4 + 5$ to violate the quantum Hamming bound. The only possibility for $l > n/4 + 4$ is $l = 12$, $n = 31$. Assume for the moment that $l \leq n/4 + 4$. Then there are at least $n - 16$ qubits in the code that are affected by at most one of the generators of D. This is more than $l + 3$, so either at least two of the generators of D must each affect two qubits that are fixed by all of the other generators, or one generator fixes four qubits that are unaffected by all of the other generators. The second case will be more restrictive to the code than the first one, so I will assume the first case holds. Assume without loss of generality that the two generators are M_{l-1} and M_l. Then errors on the four qubits affected only by these generators leave the codewords within the subspace fixed by D', the group generated by $M_1, \ldots, M_{l-2}$. There are 67 errors of weight zero, one and two on the four qubits, so

$$67 \cdot 2^k \leq 2^{n-(l-2)} \tag{7.57}$$

$$k \leq n - l - 5. \tag{7.58}$$

This is at least as restrictive as the quantum Hamming bound for any n between 31 and 51.

That leaves the case $l = 12$, $n = 31$. Even in this case, there must be at least fourteen qubits that are affected by at most one of the generators of D. As before, this is enough to ensure that we can pick two generators of D that will together act on four qubits unaffected by any of the other generators. Again, $k \leq n - l - 5$, which is more restrictive than the quantum Hamming bound. Therefore, there are no two-error-correcting degenerate stabilizer codes exceeding the quantum Hamming bound.

The methods of this section could be adapted and perhaps applied to codes correcting three or more errors, but it gets more difficult for each additional error, since the cases with $l > n/(2t)$ must be treated on a special basis, and the range of n for which this could violate the quantum Hamming bound grows rapidly with t. Eventually, it might well be true that some code with enough degeneracies does violate the quantum Hamming bound.

Even though we cannot rule out the possibility of a sufficiently large degenerate code violating the quantum Hamming bound, we can still set a less restrictive bound on degenerate stabilizer codes by constructing a classical code from the quantum code [58]. Since bounds on the efficiencies of classical codes are known, we can therefore get bounds on the possible parameters of quantum codes.

To produce a classical code from a quantum code, first put the code in standard form, as per (4.3). In particular, note the $r \times k$ matrix A_2. $r \leq n-k$, but by performing single qubit rotations from $N(\mathcal{G})$, we can always convert one generator to the product of σ_z's, so we can ensure that $r \leq n-k-1$. If we look at the classical code C with $k \times (r+k)$ generator matrix $(A_2^T|I)$, then C encodes k bits in at most $n-1$ bits. If the original quantum code could correct t quantum errors, it turns out that the classical code C can correct t classical bit flip errors, whether the quantum code was degenerate or nondegenerate. Therefore, the existence of an $[n,k,d]$ quantum code implies that an $[n-1,k,d]$ classical code exists.

7.4 Error-Correcting Codes and Entanglement Purification Protocols

Before discussing bounds on the channel capacity, I will discuss another way of looking at quantum codes that is sometimes helpful for thinking about the channel capacity. Consider the situation where Alice prepares a number of EPR pairs and sends one member of the pair to Bob. In general, both the qubits that Alice keeps and the qubits she sends to Bob may be subject to errors and decoherence. This means that Alice and Bob will share a number of imperfect pairs. If Alice attempts to teleport a state using these imperfect EPR pairs, for instance, the state that Bob receives will be incorrect. Alice and Bob wish to perform some local operations on their halves of the imperfect pairs so that they are left with a smaller number of perfect pairs (or at least better ones). A protocol to do this is called an *entanglement purification protocol* (or EPP) [17, 42].

Depending on the situation, Bob and Alice may or may not be allowed to communicate with each other and perform operations conditioned on the results of measurements by the other one. If both Bob and Alice can communicate with each other via classical communication channels, the possible protocols they can implement are called two-way error purification protocols (or 2-EPPs). If Bob can only receive classical information (as well as qubits) from Alice, but not

transmit, then Bob and Alice are restricted to using one-way error purification protocols (or 1-EPPs). In principle, there is another possibility. Bob and Alice might not be able to communicate classically at all. However, it turns out that the protocols available for them in this case are equivalent to the 1-EPPs. On the other hand, it is known that in some circumstances, 2-EPPs allow more good pairs to be purified than 1-EPPs do [17].

One remarkable fact about 1-EPPs is that they are equivalent to quantum error-correcting codes. Suppose we have a quantum code. We can make a 1-EPP out of it as follows: Alice encodes the qubits she is going to send to Bob using the code, then Bob corrects and decodes. The encoded qubits that are thus preserved in the channel retain their entanglement with the qubits Alice kept, and thus form part of a good EPR pair. The number of good pairs is just equal to the number of encoded qubits.

Conversely, suppose we have a 1-EPP that distills k good pairs from n noisy pairs and we wish to make a quantum code. In this case Alice is the encoder and Bob is the decoder for the code. Alice creates n EPR pairs and sends them to Bob, then performs her half of the 1-EPP. Since she cannot receive transmissions from Bob, she does not need to wait until Bob receives the qubits to do this. This is why a quantum code is equivalent to a 1-EPP and not a 2-EPP. After she has performed her half of the purification protocol, sending any necessary classical information, she takes the k qubits she wishes to protect and performs her half of the teleportation protocol using her half of what will be the k good pairs. Again, she sends the classical information about the measurement results to Bob. Bob now receives the qubits, plus all the classical information. He completes the purification protocol, purifying k good pairs. Since they are good EPR pairs, when he then completes the teleportation protocol, the resulting state is the correct one, and the whole process acts like a code encoding k qubits in n qubits.

7.5 Capacity of the Erasure Channel

Most quantum channels are very difficult to analyze. However, the channel capacity is known for at least one simple channel of interest. The *erasure channel* is the channel for which every qubit sent through the channel has some chance p of being totally randomized. However, when this happens, we always know on which qubit it occurred. The capacity of the erasure channel for both quantum codes and 2-EPPs is straightforward to calculate [59].

The capacity for 2-EPPs is particularly straightforward. If Alice sends n EPR pairs through the channel, pn of them will be destroyed, but $(1-p)n$ will remain intact. Furthermore, Bob will know which pairs remain intact, so he tells Alice and they discard the useless pairs. This achieves a rate of $1-p$. Clearly, it is impossible to do better than this. This means that the capacity for a 2-EPP is just $1-p$.

With a 1-EPP or quantum code, we cannot do as well, because Bob cannot tell Alice which pairs she should keep and which she should throw away. In fact,

we can set an upper bound on the capacity of $1 - 2p$. Suppose the erasure rate of p in the channel is actually caused by Charlie, who steals any given qubit with probability p, replaces any stolen qubits with random ones, and then tells Bob which qubits he stole. When $p = 1/2$, Bob has exactly the same number of valid pairs as Charlie. If there were any operations Alice could make without consulting Bob that enabled him to purify even a single valid pair, Charlie could do the same thing as Bob, also giving a valid pair. Now when Alice attempts to teleport something to Bob, she is also teleporting it to Charlie. This would allow the cloning of a quantum state. Therefore, the rate for $p > 1/2$ is zero. For $p < 1/2$, we can imagine Alice somehow knows $n(1 - 2p)$ of the pairs that will not be stolen by Charlie. The remaining $2pn$ pairs she is uncertain about. Of them, pn will be stolen by Charlie, again leaving him with the same number of good pairs from this set as Bob has. If Alice attempts to purify more than $n(1-2p)$ pairs with Bob, she will therefore also be purifying pairs with Charlie, again leading to state cloning. Therefore, the capacity is bounded above by $1 - 2p$.

This is, in fact, the actual achievable capacity for this channel. Suppose we take a random Abelian subgroup of $\mathcal{G}_n$ with $n - k$ generators. This subgroup will act as the stabilizer S of a code. If we encode k qubits using this code, and then send them through the erasure channel, for large n, with high probability, pn known qubits will have been randomized. We need to distinguish between the 4^{pn} possible errors on these qubits. Since the error operators are all on the same pn qubits, there are again 4^{pn} products of these operators. If measure one of these products anticommute with some element of S, then we will be able to correct the errors and decode the k qubits, with fidelity approaching one for large n. Since the generators are chosen randomly, each one will commute with half of the possible operators of weight pn and anticommute with half of the possible operators. The different generators commute and anticommute with operators independently, so the number of operators that commute with all $n - k$ generators is

$$4^{pn}/2^{n-k} = 2^{k-(1-2p)n} = 2^{(r-1+2p)n}, \tag{7.59}$$

where r is the rate: $k = rn$. As long as $r < 1 - 2p$, the chance of not being able to distinguish all the likely errors goes to zero as $n \to \infty$. Therefore, a random stabilizer code can give us rate $1 - 2p$. Since this coincides with the upper bound on the capacity, it is the actual capacity of the erasure channel.

7.6 Capacity of the Depolarizing Channel

The *depolarizing channel* is a very natural channel to consider. In this channel, with probability $1 - p$, each qubit is left alone. In addition, there are equal probabilities $p/3$ that σ_x, σ_y, or σ_z affects the qubit. We can apply similar methods to the depolarizing channel as with the erasure channel to place upper and lower bounds on its capacity. However, currently these bounds do not meet, so the actual capacity of the depolarizing channel is unknown.

The depolarizing channel can also simulated by imagining Charlie is randomly stealing some qubits from the channel. If Charlie steals a qubit with probability q and replaces it with a random qubit (not telling Bob which one was stolen), there is still a $1/4$ chance that Charlie happens to replace the stolen qubit with one in the same state. There is only a chance $q/4$ of Charlie applying each of σ_x, σ_y, and σ_z. Therefore, this situation corresponds to the depolarizing channel with $p = 3q/4$. We can make a cloning argument just as with the erasure channel to set an upper bound on the capacity. Again we find that the capacity is limited by $1-2q = 1-8p/3$. When $p > 3/8$, the rate of transmission is necessarily zero.

Actually, we can set a tighter upper bound than this. Randomly stealing qubits is not the best eavesdropping method available to Charlie that will look like the depolarizing channel. The best eavesdropping method actually allows him to produce the same state as Bob whenever $p > 1/4$ [60]. This means that the rate is limited to $1 - 4p$. This is the asymptotic form of the Knill-Laflamme bound, which was derived for codes with a fixed minimum distance in section 7.1.

We can set a lower bound for the achievable rate by again considering the rate for a random stabilizer code. If we encode k qubits in n qubits using a random stabilizer S, the expected number of errors is pn. We need measure one of the errors to be distinguishable from each other. The errors E and F are distinguishable if $E^\dagger F$ anticommutes with some elements of S, and are not if they do not. The typical product $E^\dagger F$ actually does not have weight $2pn$. There is a chance p^2 that E and F will both have nontrivial action on a given qubit. If they act as different Pauli matrices, the product will still act on that qubit. If they act as the same Pauli matrix, the product will not act on that qubit at all. The probability of having both act as the same Pauli matrix is $p^2/3$. Therefore, the expected length of the product $E^\dagger F$ is $(2p - 4p^2/3)n$. Let $x = 2p - 4p^2/3$.

Let the number of errors of weight w be $N(w)$. Then the number of different products of weight xn is $N(xn)$, and therefore the number of typical products that commute with everything in S is $N(xn)/2^{n-k}$. Now, there are $N(pn)$ likely errors, so the number of ways we can pair them into products is $N(pn)[N(pn) - 1]/2$. This means that the number of ways of getting any given operator O of weight xn is

$$\binom{N(pn)}{2} \Big/ N(xn). \tag{7.60}$$

For each of the pairs that gives one of the $N(xn)/2^{n-k}$ products that commute with S, we must remove one of the errors in the pair from the group of likely errors. Therefore, we must remove

$$\binom{N(pn)}{2} \Big/ 2^{n-k} \tag{7.61}$$

errors. We want to remove only measure zero of the errors, so we wish this

number to be small compared to $N(pn)$ for large n. Thus,

$$N(pn)/2^{n-k+1} \ll 1 \tag{7.62}$$

$$N(pn) \ll 2^{n-k+1} \tag{7.63}$$

$$k/n < 1 - \frac{1}{n}\log_2 N(pn) = 1 - p\log_2 3 - H(p). \tag{7.64}$$

This is just the quantum Hamming bound (7.3). In other words, a random code saturates the quantum Hamming bound.

However, the quantum Hamming bound only limits the efficiency of non-degenerate codes. The typical element of a random stabilizer will have weight $3n/4$, which is much larger than pn for any p where the rate could possibly be nonzero. Therefore, a random code will have a negligable number of degenerate errors, and the quantum Hamming bound will still apply. However, if we choose the stabilizer to be of a restricted form rather than totally random, we can choose it to have very many degeneracies, and the quantum Hamming bound may be exceeded [61], although existing codes only allow us to exceed the rate of a random code by a very small amount. Shor and Smolin showed that by concatenating a random code with a simple repetition code ($|0\rangle$ becomes the tensor product of $|0\rangle$'s and $|1\rangle$ becomes the tensor product of $|1\rangle$'s), the rate of the code is improved slightly near the zero-rate limit. The optimum block size for repetition turns out to be five.

We can still set an upper bound on the efficiency of a degenerate stabilizer code using similar arguments to those that gave us the capacity of a random stabilizer code. Note that this upper bound does not necessarily apply to all codes, so it may not be a strict upper bound on the capacity. However, non-stabilizer codes are very difficult to work with, so it does provide a practical upper bound on the capacity.

To give this bound, assume that every element of S actually has weight xn. This bound is unlikely to be achievable, since the product of two operators of weight xn will only rarely have weight xn again. There are at least $N(xn)/2^{n-k}$ operators of weight n that commute with S, but 2^{n-k} of them are in S. Therefore, in the best case, there are only $N(xn)/2^{n-k} - 2^{n-k}$ operators that can potentially cause a problem. In the limit where n and $k = rn$ are both large, either $N(xn)/2^{n-k}$ will dominate the number of troublesome operators, or $N(xn)/2^{n-k} \ll 2^{n-k}$. In the first case, the calculation goes through as for a completely random stabilizer, giving us a capacity only at the quantum Hamming bound. In the second case,

$$N(xn) \ll 2^{2(n-k)} \tag{7.65}$$

$$r = k/n < 1 - \frac{1}{2n}\log_2 N(xn) = 1 - \frac{x}{2}\log_2 3 - \frac{1}{2}H(x). \tag{7.66}$$

Since $x = 2p - 4p^2/3$, this is higher than the quantum Hamming bound. Equation (7.66) gives an upper bound on the capacity of the depolarizing channel achievable using stabilizer codes. It is shown in figure 7.1 along with the Knill-Laflamme bound and the quantum Hamming bound. Cleve has also proved a

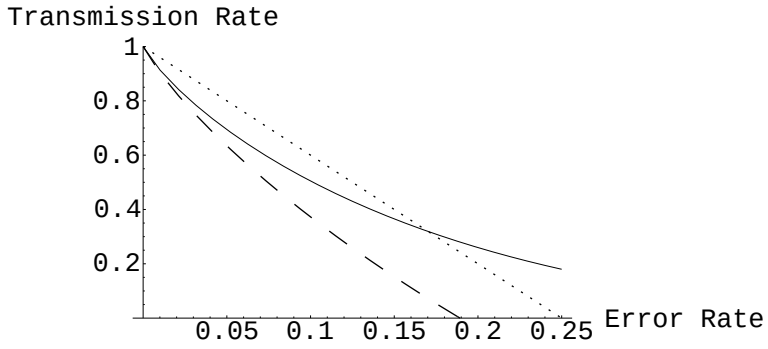

Figure 7.1: The quantum Hamming bound (dashed), the Knill-Laflamme bound (dotted), and the bound from equation (7.66) (solid).

bound on the capacity achievable using degenerate stabilizer codes [58], but it is slightly worse than (7.66) everywhere in the region of interest, so it is not shown in the figure.

Chapter 8

Examples of Stabilizer Codes

There are many known stabilizer codes [10, 11, 17, 18, 19, 20, 21, 22, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 42]. I will not attempt to list them all here, but will instead concentrate on a few interesting individual codes and classes of codes. In a number of cases, I will not just describe the stabilizers of the codes, but will also discuss the normalizers and automorphism groups of the stabilizers, since these are important to realizing fault-tolerant computation in the most efficient possible way.

8.1 Distance Two Codes

For even n, there is always an $[n, n-2, 2]$ code. The stabilizer S has two generators, one the product of all n σ_x's and one the product of all the σ_z's. For even n, these commute. $N(S)$ consists of tensor products in $\mathcal{G}$ that contain an even number of σ_x's, an even number of σ_y's, and an even number of σ_z's. We can write

$$\overline{X}_i = \sigma_{x1}\sigma_{x(i+1)} \tag{8.1}$$

$$\overline{Z}_i = \sigma_{z(i+1)}\sigma_{zn}, \tag{8.2}$$

for $i = 1, \ldots, n-2$.

The automorphism group $\mathcal{A}(S)$ contains all possible permutations of the qubits and the Hadamard rotation R applied to all n qubits at once. If n is a multiple of four, any single-qubit operation in $N(\mathcal{G})$ applied to all the qubits gives an element of $\mathcal{A}(S)$. The order of $\mathcal{A}(S)$ is thus either $2n!$ or $6n!$. Swapping qubit i with qubit j switches the $(i-1)$th encoded qubit with the $(j-1)$th encoded qubit (for $1 < i, j < n$). Swapping qubit 1 with qubit $i+1$ $(i = 1, \ldots, n-2)$ transforms

$$\overline{X}_i \rightarrow \overline{X}_i$$

$$\begin{aligned} \overline{X}_j &\rightarrow \overline{X}_i\overline{X}_j \ (i \neq j) \\ \overline{Z}_i &\rightarrow \overline{Z}_1\overline{Z}_2 \cdots \overline{Z}_{n-2} \\ \overline{Z}_j &\rightarrow \overline{Z}_j \ (i \neq j). \end{aligned} \tag{8.3}$$

Similarly, swapping qubit n with qubit $i+1$ $(i = 1, \ldots, n-2)$ transforms

$$\begin{aligned} \overline{X}_i &\rightarrow \overline{X}_1\overline{X}_2 \cdots \overline{X}_{n-2} \\ \overline{X}_j &\rightarrow \overline{X}_j \ (i \neq j) \\ \overline{Z}_i &\rightarrow \overline{Z}_i \\ \overline{Z}_j &\rightarrow \overline{Z}_i\overline{Z}_j \ (i \neq j). \end{aligned} \tag{8.4}$$

Swapping the first qubit with the nth qubit performs the transformation

$$\begin{aligned} \overline{X}_i &\rightarrow \overline{X}_1 \cdots \overline{X}_{i-1}\overline{X}_{i+1} \cdots \overline{X}_{n-2} \\ \overline{Z}_i &\rightarrow \overline{Z}_1 \cdots \overline{Z}_{i-1}\overline{Z}_{i+1} \cdots \overline{Z}_{n-2}. \end{aligned} \tag{8.5}$$

Performing R on every qubit performs the same transformation as swapping the first and nth qubits, but also performs R on every encoded qubit. For n a multiple of four, performing P on every qubit performs the following operation:

$$\begin{aligned} \overline{X}_i &\rightarrow -\overline{X}_i\overline{Z}_1 \cdots \overline{Z}_{i-1}\overline{Z}_{i+1} \cdots \overline{Z}_{n-2} \\ \overline{Z}_i &\rightarrow \overline{Z}_i. \end{aligned} \tag{8.6}$$

Because these codes are of the CSS form, a CNOT applied to every qubit transversally between two blocks is also a valid fault-tolerant operation, and performs CNOTs between the corresponding encoded qubits.

The case of $n = 4$, the smallest distance two code, is of particular interest. The code from figure 3.5 can be converted into the form of the codes currently under consideration using single-qubit rotations, although the $\overline{X}$ and $\overline{Z}$ operators will need to redefined. It can be used to detect a single error [18] or to correct a single erasure [19]. In this case,

$$\begin{aligned} \overline{X}_1 &= \sigma_{x1}\sigma_{x2} \\ \overline{X}_2 &= \sigma_{x1}\sigma_{x3} \\ \overline{Z}_1 &= \sigma_{z2}\sigma_{z4} \\ \overline{Z}_2 &= \sigma_{z3}\sigma_{z4}. \end{aligned} \tag{8.7}$$

Switching the second and third qubits or switching the first and fourth qubits both swap the two encoded qubits. Swapping the first and second qubits or the third and fourth qubits produces the transformation

$$\begin{aligned} \overline{X}_1 &\rightarrow \overline{X}_1 \\ \overline{X}_2 &\rightarrow \overline{X}_1\overline{X}_2 \\ \overline{Z}_1 &\rightarrow \overline{Z}_1\overline{Z}_2 \\ \overline{Z}_2 &\rightarrow \overline{Z}_2. \end{aligned} \tag{8.8}$$

This is just a CNOT from the second encoded qubit to the first encoded qubit. Similarly, swapping the first and third qubits or the second and fourth qubits performs a CNOT from the first encoded qubit to the second encoded qubit. The transversal Hadamard rotation in this case performs the Hadamard rotations on both qubits and switches them. Applying P to all four qubits performs the gate

$$\begin{array}{rcl} \overline{X}_1 & \rightarrow & -\overline{X}_1\overline{Z}_2 \\ \overline{X}_2 & \rightarrow & -\overline{Z}_1\overline{X}_2 \\ \overline{Z}_1 & \rightarrow & \overline{Z}_1 \\ \overline{Z}_2 & \rightarrow & \overline{Z}_2. \end{array} \tag{8.9}$$

We can recognize this as the encoded conditional sign gate followed by an encoded $\sigma_{z1}\sigma_{z2}$.

A more extensive discussion of the properties of distance two codes (and a few codes of greater distances) appears in [62].

8.2 The Five-Qubit Code

The five-qubit code is the shortest possible quantum code to correct one error, and is therefore of immense interest [17, 24]. Its stabilizer is given in table 3.2. Recall that the stabilizer is simply generated by cyclic permutations of $\sigma_x \otimes \sigma_z \otimes \sigma_z \otimes \sigma_x \otimes I$. There are five cyclic permutations of this, but only four produce independent generators. The stabilizer has sixteen elements: the identity, and the 3×5 cyclic permutations of $\sigma_x \otimes \sigma_z \otimes \sigma_z \otimes \sigma_x \otimes I$, $\sigma_y \otimes \sigma_x \otimes \sigma_x \otimes \sigma_y \otimes I$, and $\sigma_z \otimes \sigma_y \otimes \sigma_y \otimes \sigma_z \otimes I$. $\overline{X}$ is just the tensor product of five σ_x's and $\overline{Z}$ is the tensor product of the five σ_z's.

As I noted in section 3.4, the five-qubit code is a linear GF(4) code. Therefore, the operation

$$T : \sigma_x \rightarrow \sigma_y, \ \sigma_z \rightarrow \sigma_x \tag{8.10}$$

applied transversally is a valid fault-tolerant operation and performs an encoded version of itself. We can use this operation to derive a valid three-qubit operation for the five-qubit code:

$$\begin{array}{rcl} \sigma_x \otimes I \otimes I & \rightarrow & \sigma_x \otimes \sigma_y \otimes \sigma_z \\ I \otimes \sigma_x \otimes I & \rightarrow & \sigma_y \otimes \sigma_x \otimes \sigma_z \\ I \otimes I \otimes \sigma_x & \rightarrow & \sigma_x \otimes \sigma_x \otimes \sigma_x \\ \sigma_z \otimes I \otimes I & \rightarrow & \sigma_z \otimes \sigma_x \otimes \sigma_y \\ I \otimes \sigma_z \otimes I & \rightarrow & \sigma_x \otimes \sigma_z \otimes \sigma_y \\ I \otimes I \otimes \sigma_z & \rightarrow & \sigma_z \otimes \sigma_z \otimes \sigma_z. \end{array} \tag{8.11}$$

We can, of course, permute the qubits on the right and apply T or T^2 to any or all of them and still get a valid three-qubit operation.

Using measurements and this three-qubit operation, we can generate directly a number of additional one- and two-qubit operations. We can always get such gates using the protocol described in section 5.5, but it may be more efficient to get some gates using this three-qubit operation. Suppose we place the data qubit in the third place and prepare the first two qubits in encoded $|0\rangle$ states. Then apply the three-qubit operation and measure σ_y on the first two qubits. The effect is to perform a Hadamard rotation R on the data qubit. Alternatively, prepare the first two qubits in $+1$ eigenstates of σ_x, apply the three-qubit gate, and measure σ_z on the first two qubits. This performs P on the data qubit. By preparing a single ancilla qubit, applying the three-qubit operation, and making a single measurement, we can also get a variety of two-qubit operations.

8.3 A Class of Distance Three Codes

The eight-qubit code of table 3.3 is just one of a class of codes with parameters $[2^j, 2^j - j - 2, 3]$ [21]. Note that according the quantum Hamming bound, this is the maximal number of encoded qubits for $n = 2^j$, $d = 3$. These codes are related to the classical Reed-Muller codes [28], but are more efficient than CSS codes formed from the classical Reed-Muller codes. Like the classical Reed-Muller codes, the codes described in this section allow us to efficiently compute the actual error occuring from the measured error syndrome.

The first two generators of these codes are always the same. One is the product of 2^j σ_x's and the second is the product of 2^j σ_z's. We will call these generators M_X and M_Z, and the remaining j generators will be M_1 through M_j. The stabilizers of these codes always include the distance two codes discussed in section 8.1. This is convenient when correcting errors — we can measure the first two generators and use them to detect whether any error has occurred. If not, we do not need to go any further.

It will be convenient to construct the codes by describing the error syndromes of the $3n$ possible one-qubit errors. I will show that they are all distinct and then that the generators that give those error syndromes all commute. For these codes, the error syndrome $f(E)$ for error E is a $(j+2)$-bit number. Recall that each bit corresponds to a generator of S, and the ith bit is 0 iff E commutes with generator M_i. $f(E)$ is a group homomorphism from $\mathcal{G}$ to $(\mathbf{Z}_2)^{j+2}$.

Because of the form of the first two generators, the first two bits of $f(\sigma_{xi})$ are always 01, the first two bits of $f(\sigma_{zi})$ are always 10, and the first two bits of $f(\sigma_{yi})$ are always 11, as they must be to preserve the group structure of $\mathcal{G}$. For the remaining bits of the error syndrome, we will number the qubits from 0 to $n-1$ and write the number in base two. Then

$$f(\sigma_{xi}) = 01 \oplus i \tag{8.12}$$

$$f(\sigma_{zi}) = 10 \oplus \sigma(i) \tag{8.13}$$

$$f(\sigma_{yi}) = 11 \oplus (i + \sigma(i)). \tag{8.14}$$

The function $\sigma(i)$ is some as yet undefined additive group automorphism on

M_X	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x
M_Z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z
M_1	I	σ_x	I	σ_x	I	σ_x	I	σ_x	σ_z	σ_y	σ_z	σ_y	σ_z	σ_y	σ_z	σ_y
M_2	I	σ_x	I	σ_x	σ_z	σ_y	σ_z	σ_y	σ_x	I	σ_x	I	σ_y	σ_z	σ_y	σ_z
M_3	I	σ_x	σ_z	σ_y	σ_x	I	σ_y	σ_z	I	σ_x	σ_z	σ_y	σ_x	I	σ_y	σ_z
M_4	I	σ_y	σ_x	σ_z	I	σ_y	σ_x	σ_z	I	σ_y	σ_x	σ_z	I	σ_y	σ_x	σ_z

Table 8.1: The stabilizer for a $[16, 10, 3]$ code.

$(\mathbf{Z}_2)^j$. We will be able to completely describe it by defining its action on $0 \ldots 01$, $0 \ldots 010, \ldots, 10 \ldots 0$.

For this to give a distance three code, the error syndrome must have the property that $f(E) \neq 0$ for any weight two operator $E \in \mathcal{G}$. By including the stabilizer of a distance two code, we have already insured that any weight one operator has non-zero error syndrome. We can immediately see that $f(E) \neq 0$ unless E is the product of two Pauli matrices of the same type. Therefore, we need to consider

$$\begin{aligned} f(\sigma_{xl}\sigma_{xm}) &= 00 \oplus (l+m) && (8.15)\\ f(\sigma_{zl}\sigma_{zm}) &= 00 \oplus \sigma(l+m) && (8.16)\\ f(\sigma_{yl}\sigma_{ym}) &= 00 \oplus (l+m) + \sigma(l+m), && (8.17) \end{aligned}$$

for $l \neq m$. The second and third equations follow because σ is a group homomorphism. Since $i = l + m$ can be anything but 0, $\sigma(l+m)$ will not be 0 either, and we need only choose σ so that $\sigma(i) \neq i$ for any $i \neq 0$.

The actual function σ we want to use will depend on whether j is even or odd. For even j, consider the following function σ:

$$\begin{aligned} \sigma(0 \ldots 0001) &= 11 \ldots 11 \\ \sigma(0 \ldots 0010) &= 0 \ldots 001 \\ \sigma(0 \ldots 0100) &= 0 \ldots 010 \qquad (8.18)\\ &\vdots \\ \sigma(1000 \ldots 0) &= 010 \ldots 0. \end{aligned}$$

Then clearly $\sigma(i) = i/2$ for any nonzero i ending in 0. If i does end in 1, for $\sigma(i)$ to end in 1 also, the previous bit must have been 0, which means that the bit before that must have been 1, and so on. Therefore, the only possible number for which $i = \sigma(i)$ is $i = 010 \ldots 101$. Because j is even, the first bit must be 0. But $\sigma(l)$ always begins in 1 for any l ending in 1, so even for this particular i, $\sigma(i) \neq i$. Therefore, the error syndrome produces a distance three code. The smallest case is a $[16, 10, 3]$ code, which is given in table 8.1.

We do still need to verify that it is an actual code by verifying that there are commuting generators that give these error syndromes. The first two generators M_X and M_Z will always commute with the other j generators, since $f(\sigma_{xi})$ and

$f(\sigma_{zi})$ each have a 0 in the rth position for $n/2$ i's and a 1 in the rth position for $n/2$ i's. When the rth bit of $f(\sigma_{xi})$ is 0 and the rth bit of $f(\sigma_{zi})$ is 1, then the rth generator is the tensor product of σ_{xi} with something else (thus, this generator commutes with σ_{xi} and anticommutes with σ_{zi}). Other combinations will produce I, σ_{yi}, or σ_{zi}, and we can determine the complete form of M_r in this way.

We need only check that M_r and M_s commute. Let $f_r(E)$ be the $(r+2)$th bit of $f(E)$, that is, the bit corresponding to M_r. I assume without loss of generality that $s > r$. The binary matrix representation of S is closely related to the error syndrome, and M_r and M_s commute iff

$$\sum_{i=0}^{n} (f_r(\sigma_{xi})f_s(\sigma_{zi}) + f_r(\sigma_{zi})f_s(\sigma_{xi})) = 0. \tag{8.19}$$

There are a few possible cases to consider:

- $j > s > r+1 > 2$: In this case, $f_s(\sigma_{zi})$ is equal to the sum of the jth bit of i and the $(s-1)$th bit and $f_r(\sigma_{zi})$ is the sum of the jth bit of i and the $(r-1)$th bit. On the other hand, $f_r(\sigma_{xi})$ is just equal to the rth bit of i and $f_s(\sigma_{zi})$ is equal to the sth bit of i. The jth, $(r-1)$th, and $(s-1)$th bits are distinct from bits r and s. Therefore, the $f_r(\sigma_{xi})f_s(\sigma_{zi})$ term contributes to the sum when the rth bit of i is 1 and the jth and $(s-1)$th bits of i are different. This is true for $n/4$ values of i. The $f_r(\sigma_{zi})f_s(\sigma_{xi})$ term similarly contributes to the sum for $n/4$ i's. Since $n/4+n/4$ is even, M_r and M_s commute.

- $j > s > r+1 = 2$: In this case, $f_s(\sigma_{zi})$ is still equal to the sum of the jth bit of i and the $(s-1)$th bit, but $f_r(\sigma_{zi})$ is just equal to the jth bit of i. However, both the $f_r(\sigma_{xi})f_s(\sigma_{zi})$ and the $f_r(\sigma_{zi})f_s(\sigma_{xi})$ terms still contribute to the sum for $n/4$ i's, so M_r and M_s still commute.

- $j = s > r+1 > 2$: Both $f_s(\sigma_{zi})$ and $f_r(\sigma_{zi})$ are given as in the first case. $f_r(\sigma_{xi})f_s(\sigma_{zi})$ still contributes to $n/4$ terms in the sum. Now, however, $f_r(\sigma_{zi})f_s(\sigma_{xi})$ can only contribute when the jth bit of i is 1. Since we also need $f_r(\sigma_{zi}) = 1$, this term only contributes when the jth bit of i is 1 and the $(r-1)$th bit is 0. This still contributes to $n/4$ terms in the sum, so M_r and M_s again commute.

- $j > s = r+1 > 2$: Now, the $(s-1)$th bit is equal to the rth bit. That means $f_r(\sigma_{xi})f_s(\sigma_{zi})$ only contributes when the rth bit of i is 1 and the jth bit of i is 0. This contributes to $n/4$ terms in the sum, as does $f_r(\sigma_{zi})f_s(\sigma_{xi})$, so M_r and M_s commute in this case as well.

- $j = s = r+1 > 2$: This is a combination of the previous two cases. $f_r(\sigma_{xi})f_s(\sigma_{zi})$ only contributes when the rth bit of i is 1 and the jth bit of i is 0 and $f_r(\sigma_{zi})f_s(\sigma_{xi})$ contributes when the jth bit of i is 1 and the $(r-1)$th bit is 0. Again, this is an even number of contributing terms, so M_r and M_s commute.

- $j > s = r + 1 = 2$: $f_r(\sigma_{zi})$ is again equal to the jth bit of i. However, this does not affect $f_r(\sigma_{xi})f_s(\sigma_{zi})$, which contributes to $n/4$ terms in the sum, as in the previous two cases. It does affect $f_r(\sigma_{zi})f_s(\sigma_{xi})$, but this term still contributes to $n/4$ terms, so M_r and M_s commute.

- $j = s > r + 1 = 2$: As before, $f_r(\sigma_{xi})f_s(\sigma_{zi})$ contributes to $n/4$ terms in the sum. Now, however, $f_r(\sigma_{zi})f_s(\sigma_{xi})$ contributes whenever the jth bit of i is 1. This means it contributes to $n/2$ terms instead of $n/4$. Therefore, there are a total of $3n/4$ contributing terms. However, since $j \geq 3$, $n/4$ is still even, and M_1 and M_j commute too.

- $j = s = r + 1 = 2$: Since $j \geq 3$, this case is impossible.

For the case of odd j, we do something very similar. Now let

$$\begin{aligned} \sigma(0\ldots0001) &= 11\ldots11 \\ \sigma(0\ldots0010) &= 0\ldots001 \\ \sigma(0\ldots0100) &= 0\ldots010 \\ &\vdots \\ \sigma(0100\ldots0) &= 001\ldots0 \\ \sigma(1000\ldots0) &= 101\ldots1. \end{aligned} \tag{8.20}$$

An example of a code using this σ is the $[8, 3, 3]$ code given in table 3.3. In this case, if the first bit is 0, the last bit must also be 0 for the first bits of i and $\sigma(i)$ to match. However, $\sigma(i)$ is certainly not equal to i for any i with both first and last bits 0. If the first bit is 1, the last bit must be 0 in order for the first bits of i and $\sigma(i)$ to match. Thus, the second bit must be 0, which means the third bit must be 1, and so on. However, since j is odd, this progression would mean that the jth bit would have be 1, while we already know it must be 0. Therefore, there is no i for which $\sigma(i) = i$. Again, we have a distance three code.

We again need to check that the generators commute. As for even j, everything immediately commutes with M_X and M_Z. We consider similar cases to see if M_r and M_s commute:

- $j > s > r + 1 > 3$: Here, $f_r(\sigma_{zi})$ is the sum of the first, jth, and $(r-1)$th bits of i, and $f_s(\sigma_{zi})$ is the sum of the first, jth, and $(s-1)$th bits of i. This still leads to both $f_r(\sigma_{xi})f_s(\sigma_{zi})$ and $f_r(\sigma_{zi})f_s(\sigma_{xi})$ contributing to $n/4$ terms each in the sum, so M_r and M_s commute.

- $j > s > r + 1 = 3$: Now $f_r(\sigma_{zi})$ is just equal to the jth bit of i, as in the case $j > s > r + 1 = 2$ for even j. As then, M_r and M_s commute.

- $j > s > r + 1 = 2$: Now $f_r(\sigma_{zi})$ is the sum of the first and jth bits of i, and $f_r(\sigma_{xi})f_s(\sigma_{zi})$ contributes only when the first bit of i is 1 and the $(s-1)$th and jth bits of i agree, but this still contributes to $n/4$ terms in the sum, so M_r and M_s still commute.

- $j = s > r + 1 > 3$: In this case, $f_r(\sigma_{zi})f_s(\sigma_{xi})$ only contributes when the jth bit of i is 1 and the first and $(r-1)$th bits are the same. This still occurs for $n/4$ i's, so M_r and M_s commute.

- $j > s = r + 1 > 3$: Now, $f_r(\sigma_{xi})f_s(\sigma_{zi})$ contributes when the rth bit of i is 1 and the first and jth bits are the same. This occurs for $n/4$ i's, so M_r and M_s commute.

- $j = s = r + 1 > 3$: $f_r(\sigma_{xi})f_s(\sigma_{zi})$ contributes to $n/4$ terms in the sum, as in the previous case, and $f_r(\sigma_{zi})f_s(\sigma_{xi})$ does too, as in the case before that. Therefore, M_r and M_s still commute.

- $j > s = r + 1 = 3$: As with the previous two cases, $f_r(\sigma_{xi})f_s(\sigma_{zi})$ contributes to $n/4$ terms in the sum. $f_r(\sigma_{zi})$ is equal to the jth bit of i, so $f_r(\sigma_{zi})f_s(\sigma_{xi})$ contributes only when the sth and jth bits of i are both 1. This is still $n/4$ values of i, so M_r and M_s again commute.

- $j > s = r + 1 = 2$: In this case, $f_s(\sigma_{zi})$ is the jth bit of i and $f_r(\sigma_{zi})$ is the sum of the first and jth bits. That means $f_r(\sigma_{xi})f_s(\sigma_{zi})$ contributes when the first and jth bits of i are 1, and $f_r(\sigma_{zi})f_s(\sigma_{xi})$ contributes when the second bit of i is 1 and the first and jth bits are different. Both of these terms therefore contribute to $n/4$ terms in the sum, so M_r and M_s commute.

- $j = s > r + 1 = 3$: As usual, $f_r(\sigma_{xi})f_s(\sigma_{zi})$ contributes to $n/4$ terms in the sum. $f_r(\sigma_{zi})f_s(\sigma_{xi})$ contributes whenever the jth bit of i is 1. This means it contributes to $n/2$ terms in the sum, for a total of $3n/4$ nonzero terms. Again, since $j \geq 3$, $3n/4$ is even, so M_r and M_s commute.

- $j = s > r + 1 = 2$: Now, $f_r(\sigma_{xi})f_s(\sigma_{zi})$ contributes whenever the first bit of i is 1 and the jth and $(j-1)$th bits agree. This is true for $n/4$ i's. $f_r(\sigma_{zi})f_s(\sigma_{xi})$ contributes when the first bit of i is 0 and the jth bit of i is 1, which is again true for $n/4$ i's. Therefore, M_r and M_s commute.

- $j = s = r + 1 = 3$: This case only arises for the $[8, 3, 3]$ code, so we can just check it by looking at table 3.3. Again, the case $j = s = r + 1 = 2$ does not arise at all.

Now I will describe the $\overline{X}$ and $\overline{Z}$ operators for these codes. I will choose all of the $\overline{X}$ operators to be of the form $\sigma_{xa}\sigma_{xi}$ (for some $i \neq a$) times the product of σ_z's. In order to do this, we just need to find a set K of $j+1$ σ_z's (not including σ_{za}) for which $f(\sigma_{zl})$ over the $\sigma_{zl} \in K$ form a spanning set of binary vectors in $(\mathbf{Z}_2)^{j+1}$ (skipping M_Z, which σ_z will never anticommute with). Then we will be able to pick some operator E that is a product of these σ_z's so that $\overline{X}_i = \sigma_{xa}\sigma_{xi'}E$ commutes with all the generators of S, and another operator E' so that $\overline{Z}_i = \sigma_{zi'}E'$ also is in $N(S)$. If we choose the possible values of i' so that they do not overlap with the qubits l for which $\sigma_{zl} \in K$, then $\{\overline{X}_i, \overline{Z}_i\} = 0$ and $[\overline{X}_i, \overline{Z}_m] = 0$ for $i \neq m$.

For even j, K will consist of σ_{z2^l} for $l = 1, \ldots, j-1$, plus σ_{z0} and $\sigma_{z(n-1)}$ (recall the qubits are numbered 0 to $n-1$). $f(\sigma_{z0}) = 10 \oplus 0 \ldots 0$, $f(\sigma_{z(n-1)}) = 10 \oplus 10 \ldots 0$, and $f(\sigma_{z2^l})$ is 10 followed by the binary representation of 2^{l-1}. This set K has the desired properties. We pick $a = 1$.

For odd j, K will again include σ_{z2^l}, but only for $l = 1, \ldots j-2$. The remaining elements of K will be σ_{z0}, $\sigma_{z(2^{(j-1)}+1)}$, and $\sigma_{z(n-2)}$. Now, $f(\sigma_{z(2^{(j-1)}+1)}) = 10 \oplus 010 \ldots 0$, and $f(\sigma_{z(n-2)}) = 10 \oplus 10 \ldots 0$, so again K will have the desired property. We again pick $a = 1$. Note that for the eight-qubit code, this will actually give us a different definition of $\overline{X}_i$ and $\overline{Z}_i$ than in table 3.3.

I will conclude this section with a brief discussion of the automorphism groups of these codes. There will not generally be a simple transversal operation in $\mathcal{A}(S)$ for one of these codes, but they have a number of symmetries when we allow permutations of the qubits. One simple but large class of symmetries switches qubit i with qubit $i + l$, where the addition is bitwise binary. For instance, we might swap the first $n/2$ qubits with the last $n/2$ qubits, or the first $n/4$ qubits with the second $n/4$ and the third $n/4$ with the last $n/4$. The effect of this swap is to add 1 to any bit r of $f(\sigma_{xi})$ (for all i) where l is 1 in the rth bit. This much is equivalent to multiplying M_r by M_Z. We also add 1 to any bit r of $f(\sigma_{zi})$ (for all i) where $\sigma(l)$ is 1 in the rth bit. This is equivalent to multiplying M_r by M_X. Whether we multiply by M_X, M_Z, or both, the product is still in S, so the operation preserves S and is a valid fault-tolerant operation. There may be other symmetries of these codes, as well.

8.4 Perfect One-Error-Correcting Codes

A *perfect* quantum code is a nondegenerate code for which the inequality of the quantum Hamming bound becomes an equality. For one-error-correcting codes, that means $(1 + 3n)2^k = 2^n$. The possibility of a perfect code therefore exists whenever $1 + 3n$ is a power of two (up to 2^n). For instance, the five-qubit code is a perfect code. $1 + 3n$ will be a power of two iff $n = (2^{2j} - 1)/3$ for some j. Therefore there could be perfect codes for $n = 5$, $n = 21$, $n = 85$, and so on, with parameters $[(2^{2j} - 1)/3, (2^{2j} - 1)/3 - 2j, 3]$. In fact, perfect codes do exist for all these parameters.

One construction of these codes uses the Hamming codes over GF(4) [26]. Another construction is to paste together one of the codes from the previous section with an earlier perfect code. The stabilizer S_1 of any code from section 8.3 contains the stabilizer $R_1 = \{I, M_X, M_Z, M_X M_Z\}$ for a distance two code. To make the perfect code for $j \geq 3$, let S_1 be the stabilizer for the $[2^{2j-2}, 2^{2j-2} - 2j, 3]$ code, and S_2 be the stabilizer for the perfect code for $j - 1$, with parameters $[(2^{2j-2} - 1)/3, (2^{2j-2} - 1)/3 - 2j + 2, 3]$. For $j = 2$, S_2 is the stabilizer for the five-qubit code. Then using trivial R_2 (which still has distance one), the pasting construction of section 3.5 gives us a new code of distance three. The total number of qubits used by the code is

$$2^{2j-2} + (2^{2j-2} - 1)/3 = (4\ 2^{2j-2} - 1)/3 = (2^{2j} - 1)/3. \tag{8.21}$$

It encodes $(2^{2j}-1)/3-2j$ qubits, and therefore is the perfect code for j.

8.5 A Class of Distance Four Codes

We can extend the stabilizers of the codes from section 8.3 to get distance four codes. The parameters of these distance four codes will be $[2^j, 2^j-2j-2, 4]$. The first two generators of S will again be M_X and M_Z. The next j generators of S are the generators M_1 through M_j from section 8.3, so S includes the stabilizer for a distance three code. The last j generators of S are $N_i = RM_iR$ for $i = 1, \ldots, j$, where R is applied to all 2^j qubits. As with the codes of section 8.3, the error occuring for these codes can be efficiently determined from the error syndrome.

We can summarize this by writing the error syndromes for σ_{xi} and σ_{zi}:

$$\begin{aligned} f(\sigma_{xi}) &= 01 \oplus i \oplus \sigma(i) && (8.22)\\ f(\sigma_{zi}) &= 10 \oplus \sigma(i) \oplus i. && (8.23) \end{aligned}$$

Since S includes the stabilizer of a distance three code, it automatically has distance at least three. We need to check that $f(E) \neq 0$ for any weight three operator E. The only form of an operator E for which the first two bits of $f(E)$ could be 00 is $E = \sigma_{xa}\sigma_{yb}\sigma_{zc}$. Then

$$\begin{aligned} f(E) &= 00 \oplus (a + \sigma(b) + b + \sigma(c)) \oplus (\sigma(a) + b + \sigma(b) + c) && (8.24)\\ &= 00 \oplus (a + b + \sigma(b + c)) \oplus (b + c + \sigma(a + b)). && (8.25) \end{aligned}$$

If $r = a + b$ and $s = b + c$, then $f(E)$ is nonzero as long as $r \neq \sigma(s)$ or $s \neq \sigma(r)$. This means that we need

$$s \neq \sigma(\sigma(s)) = \sigma^2(s) \tag{8.26}$$

for all nonzero s (when $r = s = 0$, $E = I$). To see that this is true, note that for even j,

$$\begin{aligned} \sigma^2(0\ldots0001) &= 10\ldots00 \\ \sigma^2(0\ldots0010) &= 11\ldots11 \\ \sigma^2(0\ldots0100) &= 0\ldots001 \\ &\vdots \\ \sigma^2(1000\ldots0) &= 001\ldots0. \end{aligned} \tag{8.27}$$

If s has a 0 in the next-to-last bit, it cannot have $\sigma^2(s) = s$ unless $s = 0$. If s has a 1 in the next-to-last bit, it must have a 0 for the fourth-from-the-last bit, and so on. If j is a multiple of four, we find that the first bit must be a 0, which means that the last bit of s must be a 1. This in turn implies that the third-from-the-last bit is 0, and so on until we reach the second bit of s, which must be 0, so $s = 001100\ldots11$. However, the second bit of $\sigma^2(s)$ is 1 because

M_X	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x
M_Z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z
M_1	I	σ_x	I	σ_x	I	σ_x	I	σ_x	σ_z	σ_y	σ_z	σ_y	σ_z	σ_y	σ_z	σ_y
M_2	I	σ_x	I	σ_x	σ_z	σ_y	σ_z	σ_y	σ_x	I	σ_x	I	σ_y	σ_z	σ_y	σ_z
M_3	I	σ_x	σ_z	σ_y	σ_x	I	σ_y	σ_z	I	σ_x	σ_z	σ_y	σ_x	I	σ_y	σ_z
M_4	I	σ_y	σ_x	σ_z	I	σ_y	σ_x	σ_z	I	σ_y	σ_x	σ_z	I	σ_y	σ_x	σ_z
N_1	I	σ_z	I	σ_z	I	σ_z	I	σ_z	σ_x	σ_y	σ_x	σ_y	σ_x	σ_y	σ_x	σ_y
N_2	I	σ_z	I	σ_z	σ_x	σ_y	σ_x	σ_y	σ_z	I	σ_z	I	σ_y	σ_x	σ_y	σ_x
N_3	I	σ_z	σ_x	σ_y	σ_z	I	σ_y	σ_x	I	σ_z	σ_x	σ_y	σ_z	I	σ_y	σ_x
N_4	I	σ_y	σ_z	σ_x	I	σ_y	σ_z	σ_x	I	σ_y	σ_z	σ_x	I	σ_y	σ_z	σ_x

Table 8.2: The stabilizer for a $[16, 6, 4]$ code.

the next-to-last bit is. Therefore, $\sigma(s) \neq s$ in this case. If j is even, but not a multiple of four, the first bit of s must be 1, which means that the last bit is 0. Again we follow the chain of logic back to the second bit of s and again find that it must be 0, again giving a contradiction. Therefore $\sigma^2(s) \neq s$ for any nonzero s for any even j. An example for even j is the $[16, 6, 4]$ code given in table 8.2.

If j is odd,

$$\begin{aligned}
\sigma^2(0\ldots0001) &= 0111\ldots11 \\
\sigma^2(0\ldots0010) &= 1111\ldots11 \\
\sigma^2(0\ldots0100) &= 000\ldots001 \\
\sigma^2(0\ldots1000) &= 000\ldots010 \\
&\vdots \\
\sigma^2(010\ldots00) &= 0001\ldots00 \\
\sigma^2(1000\ldots0) &= 0101\ldots11.
\end{aligned} \tag{8.28}$$

In order to have $\sigma^2(s) = s$, we cannot have the first bit and last two bits of s all 0. If the first bit of s is 1, then the next-to-last bit of s must also be 1. Then if the last bit is 0, the third-from-the-last bit must be 0 and the fourth-from-the-last bit must be 1. Also, the second bit is 0 and the third bit is 1. After the third bit, they must continue to alternate 0 and 1 until the next-to-last bit. This means odd numbered bits are 1 and even numbered bits are 0. However, the fourth-from-the-last bit is an even numbered bit, giving a contradiction. Therefore, if the first bit of s is 1, the last two bits must both be 1 also. That means the third-from-the-last and fourth-from-the-last bits must both be 0. However, it also means that the second bit of s is 1 and the third bit of s is 0. The fourth bit is 0 again, but the fifth bit is 1, and after that they alternate until the last two bits. This contradicts the fact that the third- and fourth-from-the-last bits must both be 0.

That leaves the possibility that the first bit of s is 0. Then the next-to-last bit is 0 too, so the last bit must be 1. That means the third-from-the-last bit

M_X	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x
M_Z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z
M_1	I	σ_x	I	σ_x	σ_y	σ_z	σ_y	σ_z
M_2	I	σ_x	σ_z	σ_y	I	σ_x	σ_z	σ_y
M_3	I	σ_y	σ_x	σ_z	σ_x	σ_z	I	σ_y
N_1	I	σ_z	I	σ_z	σ_y	σ_x	σ_y	σ_x
N_2	I	σ_z	σ_x	σ_y	I	σ_z	σ_x	σ_y
N_3	I	σ_y	σ_z	σ_x	σ_z	σ_x	I	σ_y

Table 8.3: The stabilizer for the $[8, 0, 4]$ code.

is 0 and the fourth-from-the-last bit is 1. Also, the second and third bits of s are both 1. The next two bits are both 0, and the two after that are both 1. The bits pair up to be the same, with the pairs alternating between 0 and 1. However, the fourth- and third-from-the-last bits form one of these pairs, and they are different, giving another contradiction. Therefore, $\sigma^2(s) \neq s$ for any nonzero s for odd j as well as for even j. An example for odd j is the $[8, 0, 4]$ code shown in table 8.3.

To show that this set of generators forms the stabilizer for a code, we still have to show that they all commute. From the fact that M_r and M_s commute with each other and M_X and M_Z, we can immediately conclude that N_r and N_s commute with each other and M_X and M_Z. Also, M_r and N_r commute, since they get one sign of -1 for each σ_x or σ_z in M_r, and there are an even number of σ_x's and σ_z's. We must show that M_r commutes with N_s for $r \neq s$. Now,

$$f_{M_r}(N_s) = \sum_{i=0}^{n-1} \left[i^{(r)} i^{(s)} + \sigma(i)^{(r)} \sigma(i)^{(s)} \right]. \tag{8.29}$$

Here, $x^{(r)}$ is the rth bit of x. Now, σ is a permutation of 0 through $n-1$, so the second term in the sum is equal to the first term in the sum. Therefore, the sum is automatically zero, and these generators do form a stabilizer.

8.6 CSS Codes

As discussed in section 3.3, a CSS code [29, 30] is one where some of the generators are tensor products of σ_x's and the rest are tensor products of σ_z's. The σ_x generators and the σ_z generators correspond to the parity check matrices of two classical codes C_1 and C_2, with $C_1^\perp \subseteq C_2$. For instance, the classical Reed-Muller codes can be used to create a number of good quantum codes. CSS codes cannot be as efficient as the most general quantum code, but they can still be quite good. We can set upper and lower bounds using adaptations of the classical Hamming bound and Gilbert-Varshamov bound. This argument shows that the rate k/n of a CSS code to correct t arbitrary errors is asymptotically limited by

$$1 - 2H(2t/n) \leq k/n \leq 1 - 2H(t/n). \tag{8.30}$$

The CSS codes are a particularly interesting class of codes for two reasons: First, they are built using classical codes, which have been more heavily studied than quantum codes, so it is fairly easy to construct useful quantum codes simply by looking at lists of classical codes. Second, because of the form of the generators, the CSS codes are precisely those for which a CNOT applied between every pair of corresponding qubits in two blocks performs a valid fault-tolerant operation (see section 5.3). This makes them particularly good candidates for fault-tolerant computation.

In order to get universal fault-tolerant computation for a code, the first step is to produce the encoded CNOT for the code. For the most general stabilizer code, this requires performing a four-qubit operation using two ancilla qubits and making two measurements. In a CSS code, this process is reduced to a single transversal operation. Next, in order to produce one-qubit operations, we need to use one ancilla qubit, perform a CNOT, and make a measurement. For the most general CSS code, we will still have to do this. However, if the code has the property that $C_1 = C_2$ (so $C_1^\perp \subseteq C_1$), then the σ_x generators have the same form as the σ_z generators, so a transversal Hadamard rotation is also a valid fault-tolerant operation. If we further have the property that the parity check matrix of C_1 has a multiple of four 1s in each row, then the transversal phase P is a valid fault-tolerant operation too. For a general CSS code satisfying these conditions, these operations will perform some multiple-qubit gate on the qubits encoded in a single block. However, if each block only encodes a single qubit, we can choose the $\overline{X}$ and $\overline{Z}$ operators so that transversal Hadamard performs an encoded Hadamard rotation, and so that the transversal P performs an encoded P or $P^\dagger$. In particular, when C_1 is a punctured doubly-even self-dual classical code, all these conditions are satisfied, and we can perform any operation in $N(\mathcal{G})$ by performing a single transversal operation [38]. In order to get universal computation, we will also need the Toffoli gate or some other gate outside $N(\mathcal{G})$, and this will almost always require a more complicated construction.

8.7 Amplitude Damping Codes

Suppose we restrict attention to the amplitude damping channel. In this channel, each qubit behaves independently according to one of the following matrices:

$$\begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-\epsilon^2} \end{pmatrix} \text{ or } \begin{pmatrix} 0 & \epsilon \\ 0 & 0 \end{pmatrix}. \tag{8.31}$$

It is difficult to create efficient codes that will deal with the exact evolution produced by this channel. However, when ϵ is fairly small, it is sufficient to merely satisfy equation (2.10) approximately [20]. If we wish to correct the equivalent of one error, corrections of $O(\epsilon^3)$ will not matter, since that would be equivalent to distinguishing one error from two errors. Let us expand

$$\begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-\epsilon^2} \end{pmatrix} = I - \frac{1}{4}\epsilon^2(I - \sigma_z) + O(\epsilon^4). \tag{8.32}$$

M_1	σ_x	σ_x	σ_x	σ_x
M_2	σ_z	σ_z	I	I
M_3	I	I	σ_z	σ_z
$\overline{X}$	σ_x	σ_x	I	I
$\overline{Z}$	σ_z	I	σ_z	I

Table 8.4: A four-qubit code for the amplitude damping channel.

All of the higher order corrections to this equation will be powers of $I - \sigma_z$. Therefore, if we let

$$A = \sigma_x(I - \sigma_z) = \frac{2}{\epsilon}\begin{pmatrix} 0 & \epsilon \\ 0 & 0 \end{pmatrix}, \tag{8.33}$$

and

$$B = I - \sigma_z, \tag{8.34}$$

we need to consider all terms of the form

$$\langle\psi_i|E^\dagger F|\psi_j\rangle, \tag{8.35}$$

where E and F are products of A and B. We get one factor of ϵ for each A and one factor of ϵ^2 for each B. We only need to consider those terms that have total order less than ϵ^d to have an effectively distance d code. This corrects t errors where $d = 2t + 1$.

One possible way to achieve this is to have a CSS code for which the σ_z generators can *correct* t σ_x errors and the σ_x generators can *detect* t σ_z errors. For instance, the code given in table 8.6 will work if we first map $\sigma_z \to \sigma_x$ and $\sigma_y \to \sigma_z$. For such a code, we are correcting I and σ_z rather than B. Since B is in the linear span of σ_z and the identity, it is handled by these codes as well.

We can expand the range of possible codes by taking the actual linear combination of I and σ_z that appears in A and B into account. For instance, consider the code from table 8.4 [20]. This code can correct one amplitude damping error (i.e., it satisfies (2.10) to $O(\epsilon^3)$). We can instantly see that (2.10) is satisfied for $E^\dagger F = A_i$ (the subscript indicates the affected qubit) or $E^\dagger F = A_i^\dagger A_j$, where $(i, j) \neq (1, 2), (3, 4)$. When $(i, j) = (1, 2)$ (or $(3, 4)$), something interesting and unusual happens:

$$\begin{aligned} \langle\psi_i|A_1^\dagger A_2|\psi_j\rangle &= \langle\psi_i|(I - \sigma_{z1})\sigma_{x1}\sigma_{x2}(I - \sigma_{z2})|\psi_j\rangle & (8.36)\\ &= \langle\psi_i|\sigma_{x1}\sigma_{x2}(I + \sigma_{z1})\,(I - \sigma_{z2})|\psi_j\rangle. & (8.37)\end{aligned}$$

Now, $\sigma_{z1}\sigma_{z2}|\psi_j\rangle = |\psi_j\rangle$, so

$$\begin{aligned} \langle\psi_i|\sigma_{x1}\sigma_{x2}(I + \sigma_{z1})\,(I - \sigma_{z2})|\psi_j\rangle &= \langle\psi_i|\sigma_{x1}\sigma_{x2}(I + \sigma_{z1})\,(I - \sigma_{z1})|\psi_j\rangle & (8.38)\\ &= 0, & (8.39)\end{aligned}$$

since $(I + \sigma_{z1})\,(I - \sigma_{z1}) = 0$. We also need to consider the terms $E^\dagger F = B$ and $E^\dagger F = A_i^\dagger A_i = I - \sigma_{zi} = B$. In this case, we can again separate B into I and σ_z, and the latter is handled by the generator M_1.

By applying similar principles, we can see that Shor's nine-qubit code (table 3.1) can be used to correct two amplitude damping errors. We need to consider products of one through four A's and products of one or two B's, as well as the product of a B with one or two A's. Shor's code breaks down into three blocks of three. If for any block of three, we have one or two A's acting on that block, $E^\dagger F$ will anticommute with one of the σ_z generators for that block, and $\langle\psi_i|E^\dagger F|\psi_j\rangle = 0$. This takes care of all possible operators $E^\dagger F$ involving one, two, or four A's. We still need to consider $A_1^\dagger A_2 A_3$ (and similar terms) and products of one or two B's. The products of B's we again expand into I and σ_z, producing products of zero, one, and two σ_z's. Operators with one σ_z or with two σ_z's in different blocks of three will anticommute with one of the σ_x operators. Operators such as $\sigma_{z1}\sigma_{z2}$ that act on two qubits in the same block of three are in the stabilizer and are thus equivalent to the identity. Finally, operators such as $A_1^\dagger A_2 A_3$ are dealt with similarly to $A_1^\dagger A_2$ for the four qubit code above:

$$\begin{aligned}
\langle\psi_i|A_1^\dagger A_2 A_3|\psi_j\rangle &= \langle\psi_i|(I-\sigma_{z1})\sigma_{x1}\sigma_{x2}(I-\sigma_{z2})\sigma_{x3}(I-\sigma_{z3})|\psi_j\rangle && (8.40)\\
&= \langle\psi_i|\sigma_{x1}\sigma_{x2}\sigma_{x3}(I+\sigma_{z1})\,(I-\sigma_{z2})\,(I-\sigma_{z3})|\psi_j\rangle && (8.41)\\
&= \langle\psi_i|\sigma_{x1}\sigma_{x2}\sigma_{x3}(I+\sigma_{z1})\,(I-\sigma_{z1})\,(I-\sigma_{z3})|\psi_j\rangle && (8.42)\\
&= 0. && (8.43)
\end{aligned}$$

Thus, the nine qubit code can correct two amplitude damping errors.

Fault tolerance for these codes must be handled carefully. Transversal operations of any sort will not respect the form of the error operators, so we need to be sure the code will be able to correct the new error operators. For instance, the CNOT applied to $I \otimes A$ produces $(I \otimes \sigma_x)\,(I \otimes I - \sigma_z \otimes \sigma_z)$. This cannot be written as the tensor product of A's and B's. However, $I \otimes A_i$ is still distinguishable from the images of $I \otimes A_j$ (since $(I \otimes I + \sigma_z \otimes \sigma_z)\,(I \otimes I - \sigma_z \otimes \sigma_z) = 0$) and $A_j \otimes I$. Therefore, transversal CNOT is a valid fault-tolerant operation for the four-qubit code as long as we correct errors taking its effects into account.

8.8 Some Miscellaneous Codes

In this section I present a few more codes that do not fit easily into any of the classes I have already discussed. Figure 8.5 shows an $[11, 1, 5]$ code, the smallest code to correct two errors [26]. Figure 8.6 gives a code that can correct one σ_x error or one σ_z error, but not a σ_y error. This code is better than any possible distance three code, and is another example illustrating the utility of stabilizer codes for more general channels than the depolarizing channel. It is based on the classical Hamming code with an additional generator to distinguish between σ_x and σ_z errors. In fact, this code also detects if a σ_y error has occurred, although it cannot tell us where the error occurred.

The set of all possible codes includes many codes that are not equivalent to stabilizer codes. Currently, however, only one is known that is better than any stabilizer code [63]. This code has distance two and encodes six states using five

M_1	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	I	I	I	I	I
M_2	σ_x	σ_x	σ_x	σ_x	σ_x	σ_x	I	I	I	I	I
M_3	I	I	I	σ_z	σ_x	σ_y	σ_y	σ_y	σ_y	σ_x	σ_z
M_4	I	I	I	σ_x	σ_y	σ_z	σ_z	σ_z	σ_z	σ_y	σ_x
M_5	σ_z	σ_y	σ_x	I	I	I	σ_z	σ_y	σ_x	I	I
M_6	σ_x	σ_z	σ_y	I	I	I	σ_x	σ_z	σ_y	I	I
M_7	I	I	I	σ_z	σ_y	σ_x	σ_x	σ_y	σ_z	I	I
M_8	I	I	I	σ_x	σ_z	σ_y	σ_z	σ_x	σ_y	I	I
M_9	σ_z	σ_x	σ_y	I	I	I	σ_z	σ_z	σ_z	σ_x	σ_y
M_{10}	σ_y	σ_z	σ_x	I	I	I	σ_y	σ_y	σ_y	σ_z	σ_x
$\overline{X}$	I	I	I	I	I	I	σ_x	σ_x	σ_x	σ_x	σ_x
$\overline{Z}$	I	I	I	I	I	I	σ_z	σ_z	σ_z	σ_z	σ_z

Table 8.5: The stabilizer for an $[11, 1, 5]$ code.

M_1	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z	σ_z
M_2	σ_y	σ_y	σ_y	σ_y	I	I	I
M_3	σ_y	σ_y	I	I	σ_y	σ_y	I
M_4	σ_y	I	σ_y	I	σ_y	I	σ_y
$\overline{X}_1$	σ_x	σ_x	I	I	I	I	σ_z
$\overline{X}_2$	σ_x	I	σ_x	I	I	σ_z	I
$\overline{X}_3$	σ_x	I	I	σ_z	σ_x	I	I
$\overline{Z}_1$	I	σ_z	I	σ_z	I	σ_z	I
$\overline{Z}_2$	I	I	σ_z	σ_z	I	I	σ_z
$\overline{Z}_3$	I	I	I	I	σ_z	σ_z	σ_z

Table 8.6: The stabilizer for a code to correct one σ_x or σ_z error.

qubits, whereas any distance two stabilizer code could only encode two qubits (four states) with five qubits. It can be given in terms of the projection P onto the subspace of valid codewords:

$$\begin{aligned} P = 1/16\ [3\ I \otimes I \otimes I \otimes I \otimes I + (I \otimes \sigma_z \otimes \sigma_y \otimes \sigma_y \otimes \sigma_z)_{\rm cyc} \\ + (I \otimes \sigma_x \otimes \sigma_z \otimes \sigma_z \otimes \sigma_x)_{\rm cyc} - (I \otimes \sigma_y \otimes \sigma_x \otimes \sigma_x \otimes \sigma_y)_{\rm cyc} \\ + 2\ (\sigma_z \otimes \sigma_x \otimes \sigma_y \otimes \sigma_y \otimes \sigma_x)_{\rm cyc} - 2\ \sigma_z \otimes \sigma_z \otimes \sigma_z \otimes \sigma_z \otimes \sigma_z]. \end{aligned} \tag{8.44}$$

The subscript "cyc" means that we actually add the five cyclic permutations of the indicated term. Note that this means the projection operator, and therefore the code, is itself cyclic. The trace of P is six, so P projects onto a six-dimensional space and the code can therefore be used to encode six basis states. Conjugation of P by σ_x, σ_y, or σ_z on any single qubit will produce P' with $PP' = 0$, so the code for this projection operator satisfies (2.10) for a distance two code, with $C_{ab} = \delta_{ab}$.

Appendix A

Quantum Gates

It is usually helpful to think of a quantum computer as performing a series of *gates*, drawn from some fairly small basic set of physically implementable unitary transformations. The net transformation applied to the quantum computer is the product of the unitary transformations associated with the gates performed. In order to have a universal quantum computer, it should be possible to get arbitrarily close to any unitary transformation. This property makes no guarantees about how many gates are required to get within ϵ of the desired unitary operation, and figuring out how to get a given operator with the minimum number of basic gates is the goal of quantum algorithm design.

There are a number of known sets of universal quantum gates [64, 65]. For instance, all single-qubit unitary operators and the controlled-NOT together comprise a universal set. The controlled-NOT gate (or CNOT) is a two-qubit operator that flips the second qubit iff the first qubit is $|1\rangle$. It has the matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}. \tag{A.1}$$

In fact, the controlled-NOT and one single-qubit operator are sufficient, as long as the the single-qubit rotation acts by an angle incommensurate with 2π. Another finite universal set of quantum gates consists of the Hadamard rotation R,

$$R = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \tag{A.2}$$

the phase gate P,

$$P = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}, \tag{A.3}$$

the controlled-NOT, and the Toffoli gate, which is a three-qubit gate which flips the third qubit iff the first two qubits are in the state $|11\rangle$.

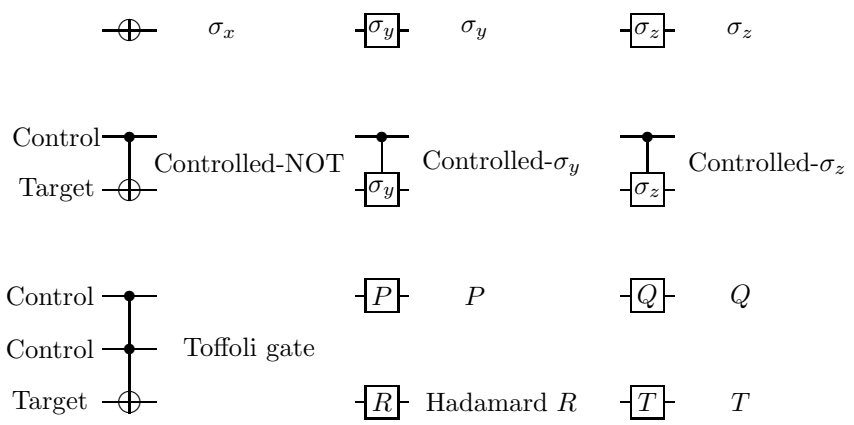

Figure A.1: Various quantum gates.

In addition to the gates mentioned above, I refer to a number of other simple gates in this thesis. For instance, the simple NOT gate, the sign gate, and the combined bit and sign flip gate (which are equal to σ_x, σ_z, and σ_y, respectively) play a crucial role in the stabilizer formalism. I also refer to two other single-qubit gates related to P and R. They are

$$Q = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & i \\ -i & -1 \end{pmatrix}, \tag{A.4}$$

and

$$T = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -i \\ 1 & i \end{pmatrix}. \tag{A.5}$$

I also occasionally refer to the "conditional sign" gate, which is a two-qubit gate that gives the basis state $|11\rangle$ a sign of -1 and leaves the other three basis states alone. The conditional sign gate is equivalent to the controlled-NOT via conjugation of one qubit by R. The conditional sign gate is effectively a controlled-σ_z gate, where σ_z gets applied to one qubit iff the other qubit is $|1\rangle$. I also use an analogous controlled-σ_y operator. The CNOT is the controlled-σ_x.

To describe a series of gates, it is usually helpful to draw a diagram of the gate array. Horizontal lines represent the qubits of the quantum computer, which enter at the left and leave from the right. A summary of the symbols I use for the various gates is given in figure A.1.

Appendix B

Glossary

additive code Another name for a stabilizer code. Often contrasted with linear quantum codes, which are a subclass of additive codes.

amplitude damping channel A channel for which the $|1\rangle$ state may relax to the $|0\rangle$ state with some probability. An example is a two-level atom relaxing via spontaneous emission.

cat state The n-qubit entangled state $|0\ldots0\rangle + |1\ldots1\rangle$. Cat states act as ancillas in many fault-tolerant operations.

coding space The subset of the Hilbert space corresponding to correctly encoded data. The coding space forms a Hilbert space in its own right.

concatenation The process of encoding the physical qubits making up one code as the logical qubits of a second code. Concatenated codes are particularly simple to correct, and can be used to perform arbitrarily long fault-tolerant computations as long as the physical error rate is below some threshhold.

CSS code Short for Calderbank-Shor-Steane code. A CSS code is formed from two classical error-correcting codes. CSS codes can easily take advantage of results from the theory of classical error-correcting codes and are also well-suited for fault-tolerant computation. See sections 3.3 and 8.6.

cyclic code A code that is invariant under cyclic permutations of the qubits.

decoherence The process whereby a quantum system interacts with its environment, which acts to effectively measure the system. The world looks classical at large scales because of decoherence. Decoherence is likely to be a major cause of errors in quantum computers.

degenerate code A code for which linearly independent correctable errors acting on the coding space sometimes produce linearly dependent states. Degenerate codes bypass many of the known bounds on efficiency of quantum

codes and have the potential to be much more efficient than any nondegenerate code.

depolarizing channel A channel that produces a random error on each qubit with some fixed probability.

distance The minimum weight of any operator $E_a^\dagger E_b$ such that equation (2.10) is *not* satisfied for an orthonormal basis of the coding space. A quantum code with distance d can detect up to $d-1$ errors, or it can correct $\lfloor (d-1)/2 \rfloor$ general errors or $d-1$ located errors.

entanglement Nonlocal, nonclassical correlations between two quantum systems. The presence of entangled states gives quantum computers their additional computational power relative to classical computers.

entanglement purification protocol Often abbreviated EPP. An EPP is a protocol for producing high-quality EPR pairs from a larger number of low-quality EPR pairs. EPPs are classified depending on whether they use one-way or two-way classical communication. A 1-way EPP (or 1-EPP) is equivalent to a quantum error-correcting code.

EPR pair Short for Einstein-Podalsky-Rosen pair. An EPR pair is the entangled state $(1/\sqrt{2})\left(|00\rangle + |11\rangle\right)$, and acts as a basic unit of entanglement.

erasure channel A channel that produces one or more located errors.

error syndrome A number classifying the error that has occurred. For a stabilizer code, the error syndrome is a binary number with a 1 for each generator of the stabilizer the error anticommutes with and a 0 for each generator of the stabilizer the error commutes with.

fault-tolerance The property (possessed by a network of gates) that an error on a single physical qubit or gate can only produce one error in any given block of an error-correcting code. A fault-tolerant network can be used to perform computations that are more resistant to errors than the physical qubits and gates composing the computer, provided the error rate is low enough to begin with. A valid fault-tolerant operation should also map the coding space into itself to avoid producing errors when none existed before.

leakage error An error in which a qubit leaves the allowed computational space. By measuring each qubit to see if it is in the computational space, a leakage error can be converted into a located error.

linear code A stabilizer code that, when described in the GF(4) formalism (section 3.4), has a stabilizer that is invariant under multiplication by ω. Often contrasted with an additive code.

located error Sometimes called an erasure. A located error is an error which acts on a known qubit in an unknown way. A located error is easier to correct than a general error acting on an unknown qubit.

nice error basis A basis which shares certain essential properties with the Pauli matrices and can be used to define a generalized stabilizer code. See section 3.6.

nondegenerate code A code for which linearly independent correctable errors acting on the coding space always produce linearly independent states. Nondegenerate codes are much easier to set bounds on than degenerate codes.

pasting A construction for combining two quantum codes to make a single larger code. See section 3.5.

perfect code A code for which every error syndrome corresponds to a correctable error. See section 8.4 for a construction of the distance three perfect codes.

quantum error-correcting code Sometimes abbreviated QECC. A QECC is a set of states that can be restored to their original state after some number of errors occur. A QECC must satisfy equation (2.10).

qubit A single two-state quantum system that serves as the fundamental unit of a quantum computer. The word "qubit" comes from "quantum bit."

qudit A d-dimensional generalization of a qubit.

shadow The set of operators in $\mathcal{G}$ which commute with the even-weight elements of the stabilizer and anticommute with the odd-weight elements of the stabilizer.

shadow enumerator The weight enumerator of the shadow. It is useful for setting bounds on the existence of quantum codes.

stabilizer The set of tensor products of Pauli matrices that fix every state in the coding space. The stabilizer is an Abelian subgroup of the group $\mathcal{G}$ defined in section 2.3. The stabilizer contains all of the vital information about a code. In particular, operators in $\mathcal{G}$ that anticommute with some element of the stabilizer can be detected by the code.

stabilizer code A quantum code that can be described by giving its stabilizer. Also called an additive code or a GF(4) code.

teleportation A process whereby a quantum state is destroyed and exactly reconstructed elsewhere. Quantum teleportation of a single qubit requires one EPR pair shared between the source and destination, and involves two measurements on the source qubit. The two bits from the measurements must be classically transmitted to the destination in order to reconstruct the original quantum state.

threshhold The error rate below which a suitably configured quantum computer can be used to perform arbitrarily long computations. Current methods for proving the existence of a threshhold use concatenated codes. Most estimates of the threshhold lie in the range 10^{-6} – 10^{-4}.

transversal operation An operation applied in parallel to the various qubits in a block of a quantum error-correcting code. Qubits from one block can only interact with corresponding qubits from another block or from an ancilla. Any transversal operation is automatically fault-tolerant.

weight A property of operators only defined on operators which can be written as the tensor product of single-qubit operators. For such an operator, the weight is the number of single-qubit operators in the product that are not equal to the identity.

weight enumerator A polynomial whose coefficients c_n are the number of elements of weight n in some set, such as the stabilizer or the normalizer of the stabilizer. Weight enumerators are very helpful in setting bounds on the possible existence of quantum error-correcting codes through identities such as the quantum MacWilliams identities (equation (7.14)).

Bibliography

[1] A. Church, "An unsolvable problem of elementary number theory," Amer. J. Math **58**, 345 (1936); A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," Proc. Lond. Math. Soc. (2) **42**, 230 (1936) and Proc. Lond. Math. Soc. (2) **43**, 544 (1937).

[2] R. P. Feynman, "Simulating physics with computers," Int. J. Theor. Phys. **21**, 467 (1982).

[3] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," Proceedings, 35th Annual Symposium on Fundamentals of Computer Science, (1994).

[4] L. K. Grover, "A fast quantum mechanical algorithm for database search," Proceedings, 28th ACM Symposium on Theory of Computation, 212 (1996).

[5] C. B. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, "Strengths and weaknesses of quantum computing," quant-ph/9701001 (1997).

[6] J. I. Cirac and P. Zoller, "Quantum computations with cold trapped ions," Phys. Rev. Lett. **74**, 4091 (1995).

[7] C. Monroe, D. M. Meekhof, B. E. King, W. M. Itano, and D. J. Wineland, "Demonstration of a fundamental quantum logic gate," Phys. Rev. Lett. **75**, 4714 (1995).

[8] Q. A. Turchette, C. J. Hood, W. Lange, H. Mabuchi, and H. J. Kimble, "Measurement of conditional phase shifts for quantum logic," Phys. Rev. Lett. **75**, 4710 (1995).

[9] N. Gershenfeld and I. Chuang, "Bulk spin resonance quantum computation," Science **275**, 350 (1997).

[10] P. Shor, "Scheme for reducing decoherence in quantum memory," Phys. Rev. A **52**, 2493 (1995).

[11] A. M. Steane, "Error correcting codes in quantum theory," Phys. Rev. Lett. **77**, 793 (1996).

[12] C. Cohen-Tannoudji, *Quantum Mechanics*, Wiley, New York (1977).

[13] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland Publishing Company, New York (1977).

[14] W. K. Wootters and W. H. Zurek, "A single quantum cannot be cloned," Nature **299**, 802 (1982).

[15] C. E. Shannon, "A mathematical theory of communication," Bell Sys. Tech. J. **27**, 379, 623 (1948).

[16] E. Knill and R. Laflamme, "A theory of quantum error-correcting codes," Phys. Rev. A **55**, 900 (1997).

[17] C. Bennett, D. DiVincenzo, J. Smolin, and W. Wootters, "Mixed state entanglement and quantum error correction," Phys. Rev. A **54**, 3824 (1996).

[18] L. Vaidman, L. Goldenberg, and S. Wiesner, "Error prevention scheme with four particles," Phys. Rev. A **54**, 1745R (1996).

[19] M. Grassl, Th. Beth, and T. Pellizzari, "Codes for the quantum erasure channel," quant-ph/9610042 (1996).

[20] D. W. Leung, M. A. Nielsen, I. L. Chuang, Y. Yamamoto, "Approximate quantum error correction can lead to better codes," quant-ph/9704002 (1997).

[21] D. Gottesman, "Class of quantum error-correcting codes saturating the quantum Hamming bound," Phys. Rev. A **54**, 1862 (1996).

[22] A. R. Calderbank, E. M. Rains, P. W. Shor, and N. J. A. Sloane, "Quantum error correction and orthogonal geometry," Phys. Rev. Lett. **78**, 405 (1997).

[23] E. Rains, "Quantum shadow enumerators," quant-ph/9611001 (1996).

[24] R. Laflamme, C. Miquel, J. P. Paz, and W. Zurek, "Pefect quantum error correction code," Phys. Rev. Lett. **77**, 198 (1996).

[25] D. Gottesman, "Pasting quantum codes," quant-ph/9607027 (1996).

[26] A. R. Calderbank, E. M. Rains, P. W. Shor, and N. J. A. Sloane, "Quantum error correction via codes over GF(4)," quant-ph/9608006 (1996).

[27] A. Steane, "Simple quantum error correcting codes," Phys. Rev. A **54**, 4741 (1996).

[28] A. Steane, "Quantum Reed-Muller codes, " quant-ph/9608026 (1996).

[29] A. R. Calderbank and P. W. Shor, "Good quantum error-correcting codes exist," Phys. Rev. A **54**, 1098 (1996).

[30] A. Steane, "Multiple particle interference and quantum error correction," Proc. Roy. Soc. Lond. A **452**, 2551 (1996).

[31] E. Knill, "Non-binary error bases and quantum codes," quant-ph/9608048 (1996); E. Knill, "Group representations, error bases and quantum codes," quant-ph/9608049 (1996).

[32] H. F. Chau, "Correcting quantum errors in higher spin systems," quant-ph/9610023 (1996)

[33] H. F. Chau, "Five quantum register error correction code for higher spin systems," quant-ph/9702033 (1997).

[34] D. Aharonov and M. Ben-Or, "Fault-tolerant quantum computation with constant error," quant-ph/9611025 (1996).

[35] E. Rains, "Nonbinary quantum codes," quant-ph/9703048 (1997).

[36] R. Cleve and D. Gottesman, "Efficient computations of encodings for quantum error correction," quant-ph/9607030 (1996).

[37] D. P. DiVincenzo, "Quantum gates and circuits," quant-ph/9705009 (1997).

[38] P. Shor, "Fault-tolerant quantum computation," quant-ph/9605011 (1996).

[39] D. DiVincenzo and P. Shor, "Fault-tolerant error correction with efficient quantum codes," Phys. Rev. Lett. **77**, 3260 (1996).

[40] D. Gottesman, "A theory of fault-tolerant quantum computation," quant-ph/9702029 (1997).

[41] C. H. Bennett, G. Brassard, C. Crepeau, R. Josza, A. Peres, and W. K. Wootters, "Teleporting an unknown quantum state via dual classical and Einstein-Podalsky-Rosen channels," Phys. Rev. Lett. **70**, 1895 (1993).

[42] C. H. Bennett, G. Brassard, S. Popescu, B. Schumacher, J. A. Smolin, and W. K. Wootters, "Purification of noisy entanglement and faithful teleportation via noisy channels," Phys. Rev. Lett. **76**, 722 (1996).

[43] E. Knill, R. Laflamme, and D. Gottesman, in preparation.

[44] E. Knill, personal communication.

[45] E. Knill, R. Laflamme, and W. Zurek, "Accuracy threshold for quantum computation," quant-ph/9610011 (1996); E. Knill, R. Laflamme, and W. Zurek, "Resilient quantum computation: error models and thresholds," quant-ph/9702058 (1997).

[46] J. Evslin, S. Kakade, and J. P. Preskill, unpublished.

[47] A. M. Steane, "Active stabilization, quantum computation and quantum state synthesis," Phys. Rev. Lett. **78**, 2252 (1997).

[48] E. Knill and R. Laflamme, "Concatenated quantum codes," quant-ph/9608012 (1996).

[49] C. Zalka, "Threshold estimate for fault tolerant quantum computing," quant-ph/9612028 (1996).

[50] S. Lloyd, "The capacity of a noisy quantum channel," Phys. Rev. A **55**, 1613 (1997).

[51] B. Schumacher and M. A. Nielsen, "Quantum data processing and error correction," Phys. Rev. A **54**, 2629 (1996).

[52] H. Barnum, M. A. Nielsen, and B. Schumacher, "Information transmission through a noisy quantum channel," quant-ph/9702049 (1997).

[53] A. Ekert and C. Macchiavello, "Error correction in quantum communication," Phys. Rev. Lett. **77**, 2585 (1996).

[54] N. J. Cerf and R. Cleve, "Information-theoretic interpretation of quantum error-correcting codes," quant-ph/9702031 (1997).

[55] P. Shor and R. Laflamme, "Quantum analog of the MacWilliams identities for classical coding theory," Phys. Rev. Lett. **78**, 1600 (1997).

[56] E. M. Rains, "Quantum weight enumerators," quant-ph/9612015 (1996).

[57] E. M. Rains, "Polynomial invariants of quantum codes," quant-ph/9704042 (1997).

[58] R. Cleve, "Quantum stabilizer codes and classical linear vodes," quant-ph/9612048 (1996).

[59] C. H. Bennett, D. P. DiVincenzo, and J. A. Smolin, "Capacities of quantum erasure channels," quant-ph/9701015 (1997).

[60] C. Fuchs and J. Smolin, unpublished.

[61] P. Shor and J. Smolin, "Quantum error-correcting codes need not completely reveal the error syndrome," quant-ph/9604006 (1996).

[62] E. M. Rains, "Quantum codes of minimum distance two," quant-ph/9704043 (1997).

[63] E. M. Rains, R. H. Hardin, P. W. Shor, and N. J. A. Sloane, "A nonadditive quantum code," quant-ph/9703002 (1997).

[64] S. Lloyd, "Almost any quantum logic gate is universal," Phys. Rev. Lett. **75**, 346 (1995).

[65] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," Phys. Rev. A **52**, 3457 (1995).