A Classification of Reversible Bit and Stabilizer Operations

by

Daniel Grier

B.S., University of South Carolina (2013)

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of Masters of Science in Computer Science and Engineering at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2015

© Massachusetts Institute of Technology 2015. All rights reserved.
A Classification of Reversible Bit and Stabilizer Operations

by

Daniel Grier

Submitted to the Department of Electrical Engineering and Computer Science on May 20, 2015, in partial fulfillment of the requirements for the degree of Masters of Science in Computer Science and Engineering

Abstract

This thesis is an exposition of a classification of classical reversible gates acting on bits in terms of the reversible transformations they generate, which was recently completed by the author, Scott Aaronson, and Luke Schaeffer. In particular, we present those portions of the classification which were the main contributions of the author. Most importantly, this thesis contains the proof that every non-affine gate generates a Fredkin gate, which was one of the main technical hurdles in completing the classification. Our classification can be seen as the reversible-computing analogue of Post’s lattice, a central result in mathematical logic from the 1940s, where we allow arbitrary ancilla bits to be used in the computation provided they return to their initial configuration at the end of the computation. It is a step toward the ambitious goal of classifying all possible quantum gate sets acting on qubits.

This thesis also gives preliminary results for the classification of stabilizer gates, which have garnered much attention due to their role in unifying many of the known quantum error-correcting codes. In the stabilizer setting, we generalize the classical model to allow the use of arbitrary stabilizer ancillas and show that this leads to several nonintuitive results. In particular, we show that the CNOT and Hadamard gates suffice to generate all stabilizer operations (whereas the phase gate is required in a more group theoretic setting); present a complete classification of the “classical” stabilizer operations; and give exact generating sets for the one-qubit stabilizer operations.

Thesis Supervisor: Scott Aaronson
Title: Professor of Electrical Engineering and Computer Science
Acknowledgments

First and foremost, I would like to thank my advisor Scott Aaronson. He is a fountain of interesting ideas, and may in fact be the busiest man alive. Still, he’s always willing to meet with me to talk about research or whatever else. I feel pretty lucky to have found myself as one of Scott’s students, and I don’t think I’ll forget that.

I also obviously have to thank all the people in the theory community around MIT for making this place an awesome place to work. I can talk to those guys forever (which, yes, is often the reason I need to work from home), and I really can’t imagine what it’d be like to be here without them. I’d especially like to mention my main research collaborators Adam Bouland and Luke Schaeffer. Working with them has made me doubt how many single-author papers are in my future.

I’d also like to thank my undergraduate advisor Stephen Fenner for being one of the best undergraduate advisors one could hope for. Dr. Fenner gave me my first true glimpse into theoretical computer science, and I really don’t think I’d be here if it weren’t for him.

I’d also be remiss if I didn’t mention my good buddy Cole Franks, who’s been a great competitor and friend for so many years. Why’d you have to move away, man? I also want to mention and thank my parents for their ever-present support. If I’m ever mature enough to have children, I’ll be stealing a lot of plays from their book. It’s hard to find exactly the right way to give my brother the shoutout he deserves. We’ve been friends since the day I was born, and he’s always been there for me when I’ve needed him. I now formally forgive him for beating me at pretty much every game in existence when we were growing up. Finally, if you’re reading this and find that you weren’t thanked explicitly, then you probably deserve it too. Thank you all.
## Contents

### Contents

List of Figures

#### 1 Introduction

- 1.1 The Pervasiveness of Universality ........................................... 11
- 1.2 Classical Reversible Gates ................................................. 13
- 1.3 Stabilizer Gates ............................................................. 15
- 1.4 Ground Rules ................................................................. 18
- 1.5 Results ........................................................................... 19
  - 1.5.1 Classical Reversible Gates\(^1\) ........................................... 19
  - 1.5.2 Stabilizer Gates .......................................................... 21

#### 2 Precisely Describing the Model

- 2.1 Notation and Definitions ..................................................... 23
  - 2.1.1 Gates ...................................................................... 24
  - 2.1.2 Gate Classes ............................................................ 26
  - 2.1.3 Alternative Kinds of Generation ..................................... 28
- 2.2 Stating the Classical Classification Theorem ......................... 29

#### 3 Ubiquity of the Fredkin Gate

- 3.1 Number-theoretic Properties of Reversible Gates\(^2\) .................... 34
- 3.2 Computing with Garbage ...................................................... 38
- 3.3 Conservative Generates Fredkin ............................................. 41
- 3.4 Non-Conservative Generates Fredkin ...................................... 45

#### 4 Towards a Full Stabilizer Classification

- 4.1 Single-qubit Gate Sets ...................................................... 51

---

\(^1\)The results mentioned in this section were the result of joint work by the author, Scott Aaronson, and Luke Schaeffer. We mention these results here to put the main result of the thesis in context. We do not intend for this thesis to be full exposition of that work.

\(^2\)Many of these results were proven in joint work with Scott Aaronson and Luke Schaeffer and were used extensively throughout the entire classification of reversible bit operations.
4.2 The Power of Ancillary Qubits .................................................. 53
4.3 Separating Classical Power from Quantum Power ....................... 56

5 Discussion and Future Work .................................................. 61

Bibliography .............................................................................. 63
List of Figures

1-1 Generating the GHZ-state from |000⟩ using Hadamard and CNOT . . . . . . . . . . . 17
1-2 Generating the swap gate from three CNOT gates . . . . . . . . . . . . . . . . . . . . . . . . . 18
2-1 Generating a Controlled-Controlled-Swap gate from Fredkin . . . . . . . . . . . . . . . . . 28
2-2 The inclusion lattice of reversible gate classes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
4-1 Hasse diagram for single qubit stabilizer subgroups . . . . . . . . . . . . . . . . . . . . . . . . . 53
4-2 Leveraging the ancillary qubit |i⟩ to form the HSHS gate . . . . . . . . . . . . . . . . . . . . . 55
4-3 The inclusion lattice of gates below the CNOT class using quantum ancillas . 60
Chapter 1

Introduction

1.1 The Pervasiveness of Universality

The *pervasiveness of universality*—that is, the likelihood that a small number of simple operations already generate all operations in some relevant class—is one of the central phenomena in computer science. It appears, among other places, in the ability of simple logic gates to generate all Boolean functions (and of simple quantum gates to generate all unitary transformations); and in the simplicity of the rule sets that lead to Turing-universality, or to formal systems to which Gödel’s theorems apply. Yet precisely because universality is so pervasive, it is often more interesting to understand the ways in which systems can *fail* to be universal.

In 1941, the great logician Emil Post [21] published a complete classification of all the ways in which sets of Boolean logic gates can fail to be universal: for example, by being monotone (like the AND and OR gates) or by being affine over $\mathbb{F}_2$ (like NOT and XOR).

This thesis stems from the ambition to find the analogue of Post’s lattice for all possible sets of *quantum* gates acting on qubits. We view this as a large, important, and underappreciated goal: something that could be to quantum computing theory almost what
the Classification of Finite Simple Groups was to group theory. To provide some context, there are many finite sets of 1-, 2- and 3-qubit quantum gates that are known to be universal—either in the strong sense that they can be used to approximate any n-qubit unitary transformation to any desired precision, or in the weaker sense that they suffice to perform universal quantum computation (possibly in an encoded subspace). To take two examples, Barenco et al. [41 showed universality for the CNOT gate plus the set of all 1-qubit gates, while Shi [23] showed universality for the Toffoli and Hadamard gates.

There are also sets of quantum gates that are known not to be universal: for example, the basis-preserving gates, the 1-qubit gates, and most interestingly, the so-called stabilizer gates [10, 3] (that is, the CNOT, Hadamard, and π/4-Phase gates), as well as the stabilizer gates conjugated by 1-qubit unitary transformations. What is not known is whether the preceding list basically exhausts the ways in which quantum gates on qubits can fail to be universal. Are there other elegant discrete structures, analogous to the stabilizer gates, waiting to be discovered? Are there any gate sets, other than conjugated stabilizer gates, that might give rise to intermediate complexity classes, neither contained in P nor equal to BQP?¹ How can we claim to understand quantum circuits—the bread-and-butter of quantum computing textbooks and introductory quantum computing courses—if we do not know the answers to such questions?

Unfortunately, working out the full "quantum Post’s lattice" appears out of reach at present. This might surprise readers, given how much is known about particular quantum gate sets (e.g., those containing CNOT gates), but keep in mind that what is asked for is an accounting of all possibilities, no matter how exotic. Indeed, even classifying 1- and

---

¹To clarify, there are many restricted models of quantum computing known that are plausibly "intermediate" in that sense, including BosonSampling [1], the one-clean-qubit model [15], and log-depth quantum circuits [6]. However, with the exception of conjugated stabilizer gates, none of those models arises from simply considering which unitary transformations can be generated by some set of k-qubit gates. They all involve non-standard initial states, building blocks other than qubits, or restrictions on how the gates can be composed.
2-qubit quantum gate sets remains wide open (!), and seems, without a new idea, to require studying the irreducible representations of thousands of groups. Recently, Aaronson and Bouland [2] completed a much simpler task, the classification of 2-mode beamsplitters; that was already a complicated undertaking.

1.2 Classical Reversible Gates

So one might wonder: can we at least understand all the possible sets of classical reversible gates acting on bits, in terms of which reversible transformations they generate? This an obvious prerequisite to the quantum case, since every classical reversible gate is also a unitary quantum gate. But beyond that, the classical problem is extremely interesting in its own right, with (as it turns out) a rich algebraic and number-theoretic structure, and with many implications for reversible computing as a whole.

The notion of reversible computing [9, 24, 16, 5, 18, 22] arose from early work on the physics of computation, by such figures as Feynman, Bennett, Benioff, Landauer, Fredkin, Toffoli, and Lloyd. This community was interested in questions like: does universal computation inherently require the generation of entropy (say, in the form of waste heat)? Surprisingly, the theory of reversible computing showed that, in principle, the answer to this question is “no.” Deleting information unavoidably generates entropy, according to Landauer’s principle [16], but deleting information is not necessary for universal computation.

Formally, a reversible gate is just a permutation \( G : \{0, 1\}^k \to \{0, 1\}^k \) of the set of \( k \)-bit strings, for some positive integer \( k \). The most famous examples are:

- the 2-bit CNOT (Controlled-NOT) gate, which flips the second bit if and only if the first bit is 1;
- the 3-bit Toffoli gate, which flips the third bit if and only if the first two bits are both 1;
• the 3-bit Fredkin gate, which swaps the second and third bits if and only if the first bit is 1.

These three gates already illustrate some of the concepts that play important roles in the classification. The CNOT gate can be used to copy information in a reversible way, since it maps $x0$ to $xx$; and also to compute arbitrary affine functions over the finite field $\mathbb{F}_2$. However, because CNOT is limited to affine transformations, it is not computationally universal. Indeed, in contrast to the situation with irreversible logic gates, one can show that no 2-bit classical reversible gate is computationally universal. The Toffoli gate is computationally universal, because (for example) it maps $x, y, 1$ to $x, y, \overline{xy}$, thereby computing the NAND function. Moreover, Toffoli showed [24] that the Toffoli gate is universal in a stronger sense: it generates all possible reversible transformations $F : \{0, 1\}^n \rightarrow \{0, 1\}^n$ if one allows the use of ancilla bits, which must be returned to their initial states by the end.

But perhaps the most interesting case is that of the Fredkin gate. Like the Toffoli gate, the Fredkin gate is computationally universal: for example, it maps $x, y, 0$ to $x, \overline{xy}, xy$, thereby computing the AND function. But the Fredkin gate is not universal in the stronger sense. The reason is that it is conservative: that is, it never changes the total Hamming weight of the input. Far from being just a technical issue, conservativity was regarded by Fredkin and the other reversible computing pioneers as a sort of discrete analogue of the conservation of energy—and indeed, it plays a central role in certain physical realizations of reversible computing (for example, billiard-ball models, in which the total number of billiard balls must be conserved).

However, all we have seen so far are three specific examples of reversible gates, each leading to a different behavior. The question remains: what are all the possible behaviors? For example: is Hamming weight the only possible “conserved quantity” in reversible computation? Are there other ways, besides being affine, to fail to be computationally universal? Can one derive, from first principles, why the classes of reversible transformations generated
by CNOT, Fredkin, etc. are somehow special, rather than just pointing to the sociological fact that these are classes that people in the early 1980s happened to study?

1.3 Stabilizer Gates

One might also wonder how much we can classify of the truly quantum gates. Understanding the stabilizer gates, which we will precisely define shortly, seems like an ideal first step. For one, the stabilizer group\(^2\) (i.e. operations that one can perform using stabilizer gates on a fixed number of wires) is discrete and finite. Furthermore, even though the stabilizer states exhibit large amounts of entanglement, a stabilizer circuit can be efficiently simulated by a classical machine by the now-famous Gottesman-Knill theorem [12]. In fact, Aaronson and Gottesman [3] show that simulating a circuit of stabilizer gates is a \(\oplus L\)-complete problem, meaning that stabilizer circuits cannot even perform all of classical computation if widely-accepted complexity conjectures are true. The restricted, discrete nature of the stabilizer group suggests that we may already have the necessary mathematical tools to understand them.

Stabilizer circuits are somewhat remarkable in that they may in fact be integral to our eventual development of a full-purpose quantum computer. Since quantum error correction will likely play a large role in determining when a quantum computer will be viable, there has been considerable research in building and analyzing quantum error correcting codes. The stabilizer formalism arose as powerful way of unifying the analyses of many of these codes [11], and as a consequence, understanding the nature of the stabilizer states has been of particular interest [8, 19]. Of course, even though stabilizer circuits are classically simulable, this does not mean we necessarily have the technology available to freely apply stabilizer gates in the quantum setting. It may be that physical restrictions imply that we can actually only

\(^2\)Often referred to as the Clifford group in the literature.
apply some subset of the stabilizer gates. We seek to address this problem by classifying the families of functions that can be computed using subsets of stabilizer gates. To understand the stabilizer group, it helps to understand the meaning of the name “stabilizer”, which is tightly connected with the Pauli operators, defined below:

\[
\begin{align*}
I &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & X &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & Y &= \begin{bmatrix} 0 & -i \\ -i & 0 \end{bmatrix} & Z &= \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}
\end{align*}
\]

The Pauli group \( \mathcal{P} \) on \( n \) qubits consists of all elements of the form \( i^a P \otimes^n \) where \( a \in \mathbb{Z} \) and \( P \otimes^n \) is an \( n \)-fold tensor product of the Pauli operators. We say that a state \( |\psi\rangle \) is stabilized by an operation \( U \) iff \( U|\psi\rangle = |\psi\rangle \). The Pauli elements and their corresponding stabilized states are below:

\[
\begin{align*}
Z : |0\rangle & \quad - Z : |1\rangle \\
X : |+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} & \quad - X : |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \\
Y : |i\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} & \quad - Y : |-i\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}
\end{align*}
\]

A stabilizer state \( |\psi\rangle \) on \( n \) qubits can be equivalently specified by \( n \) linearly independent elements from the Pauli group such that each element stabilizes \( |\psi\rangle \). Because of this equivalence between states and the list of operations that stabilize them, one can track the state of a system by simply updating a list of stabilizing operations. More precisely, suppose we have a state \( |\psi\rangle \) stabilized by operation \( P \). Suppose now that we apply some unitary \( U \) to \( |\psi\rangle \). Let \( P' = UPU^\dagger \) be the conjugation of \( P \) by \( U \). We now have \( P'(U|\psi\rangle) = UP(U|\psi\rangle) = UP|\psi\rangle = U|\psi\rangle \). Therefore, \( P' \) stabilizes \( U|\psi\rangle \), so by maintaining the list of stabilizers under conjugation we can also recover the underlying state of our system.

Unfortunately, conjugation by unitaries is often just as hard to keep track of as the underlying state itself. However, stabilizer gates are special in that they preserve the fact
that the elements of the stabilizer group are elements from the Pauli group. That is, a stabilizer gate $G$ is any operator such that $G P G^\dagger \in \mathcal{P}$ for all $P \in \mathcal{P}$. It is for this exact reason, along with simple update rules for the Pauli elements of the stabilizer group, that stabilizer circuits can be simulated in classical polynomial time. In fact, it turns out that all stabilizer circuits can be composed from the following three gates acting on the computational basis, where we extend to larger domains in the natural way (i.e. by taking tensor products with the identity function over the qubits on which the gate is not applied):

\[
\begin{align*}
CNOT &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \\
H &= \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \\
S &= \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}
\end{align*}
\]

We will refer to these as the CNOT, Hadamard, and phase gates, respectively. An example of constructing the GHZ-state from the all-zeros input is given in Figure 1-1. For completeness, we also note here that the SWAP gate

\[
\text{SWAP} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\]

is a stabilizer gate, evidenced by the equivalence in Figure 1-2. We also extend SWAP gates to act on a larger number of qubits in the natural way. Let $\text{SWAP}_{a,b}$ be a SWAP gate from qubit $a$ to qubit $b$.

\[
\begin{align*}
|0\rangle & \quad \xrightarrow{\text{H}} \\
|0\rangle & \quad \text{⊕} \\
|0\rangle & \quad \xrightarrow{\sqrt{2} \left( |000\rangle + |111\rangle \right)} \\
\end{align*}
\]

Figure 1-1: Generating the GHZ-state from $|000\rangle$ using Hadamard and CNOT

Once again, we can observe that each gate has certain properties. For instance, the CNOT
Figure 1-2: Generating the swap gate from three CNOT gates

gate is the only gate of the three that can create entanglement between two unentangled qubits. Starting from a computational basis state, the Hadamard gate is the only gate that can create a superposition of computational basis states. The phase gate is the only one of the three whose entries involve complex numbers. Indeed, we are in need of a classification theorem which would enumerate all such possible qualities.

1.4 Ground Rules

In this thesis, we aim to classify both classical reversible gate sets and stabilizer gate sets in terms of the functions they generate. To make the stabilizer model physically realistic, we say that a set of stabilizer gates generates some function over stabilizer states if one can construct a circuit of gates from the gate set that are equivalent to the function modulo some global phase. Before describing our result further, let us carefully explain the ground rules.

First, we assume that swapping is free. This simply means that we do not care how the inputs are labeled—or, if we imagine the inputs carried by wires, then we can permute the wires in any way we like. In the classical setting, the second rule is that an unlimited number of ancilla bits may be used, provided the ancilla bits are returned to their initial states by the end of the computation. In the stabilizer setting, we allow an arbitrary ancilla stabilizer state to be used in the computation provided that it returns to its initial state at the end of the computation. This second rule might look unfamiliar, but in the context of reversible computing, it is the right choice.

We need to allow ancillary inputs because if we do not, then countless transformations
are disallowed for trivial reasons. (Restricting a reversible circuit to use no ancillas is like restricting a Turing machine to use no memory, besides the \( n \) bits that are used to write down the input.) We are forced to say that, although our gates might generate some reversible transformation \( F(x, 0) = (G(x), 0) \), they do not generate the smaller transformation \( G \).

The exact value of \( n \) then also takes on undeserved importance, as we need to worry about "small-\( n \) effects": e.g., that a 3-bit gate cannot be applied to a 2-bit input.

On the other hand, the ancillary inputs must be returned to their original states because if they are not, then the computation was not really reversible. One can then learn something about the computation by examining the ancilla bits—if nothing else, then the fact that the computation was done at all. The symmetry between input and output is broken; one cannot then run the computation backwards without setting the ancilla bits differently. This is not just a philosophical problem: if the ancilla bits carry away information about the input \( x \), then entropy, or waste heat, has been leaked into the computer’s environment. Worse yet, if the reversible computation is a subroutine of a quantum computation, then the leaked entropy will cause decoherence, preventing the branches of the quantum superposition with different \( x \) values from interfering with each other, as is needed for a quantum speedup.

1.5 Results

We separate the classical and quantum settings for clarity.

1.5.1 Classical Reversible Gates\(^3\)

Although this thesis will focus mainly on the contributions of the author to the classification of reversible bit operations, we list of important aspects of the classification completed in

\(^3\)The results mentioned in this section were the result of joint work by the author, Scott Aaronson, and Luke Schaeffer. We mention these results here to put the main result of the thesis in context. We do not intend for this thesis to be full exposition of that work.
collaboration with Scott Aaronson and Luke Schaeffer below. We hope that this will serve
as greater context for the results contained within the remainder of the thesis. The complete
lattice of reversible gate classes is shown in Figure 2.2.

(1) **Conserved Quantities.** The following is the complete list of the “global quantities”
that reversible gate sets can conserve (if we restrict attention to non-degenerate gate
sets, and ignore certain complications caused by linearity and affineness): Hamming
weight, Hamming weight mod $k$ for any $k \geq 2$, and inner product mod 2 between pairs
of inputs.

(2) **Anti-Conservation.** There are gates, such as the NOT gate, that “anti-conserve” the
Hamming weight mod 2 (i.e., always change it by a fixed nonzero amount). However,
there are no analogues of these for any of the other conserved quantities.

(3) **Encoded Universality.** In terms of their “computational power,” there are only
three kinds of reversible gate sets: degenerate (e.g., NOTs, bit-swaps), non-degenerate
but affine (e.g., CNOT), and non-affine (e.g., Toffoli, Fredkin). More interestingly,
every non-affine gate set can implement every reversible transformation, and every
non-degenerate affine gate set can implement every affine transformation, if the input
and output bits are encoded by longer strings in a suitable way.

(4) **Sporadic Gate Sets.** The conserved quantities interact with linearity and affineness
in complicated ways, producing “sporadic” affine gate sets that we have classified. For
example, non-degenerate affine gates can preserve Hamming weight mod $k$, but only
if $k = 2$ or $k = 4$. All gates that preserve inner product mod 2 are linear, and all
linear gates that preserve Hamming weight mod 4 also preserve inner product mod 2.
As a further complication, affine gates can be orthogonal or mod-2-preserving or
mod-4-preserving in their linear part, but not in their affine part.
(5) **Finite Generation.** For each closed class of reversible transformations, there is a single gate that generates the entire class. *(A priori, it is not even obvious that every class is finitely generated, or that there is “only” a countable infinity of classes!)*

(6) **Symmetry.** Every reversible gate set is symmetric under interchanging the roles of 0 and 1.

### 1.5.2 Stabilizer Gates

Our classification of the stabilizer gates is yet far from completion. Completing the classification is one of the main open questions left by this research. One of the aspects of the stabilizer classification that makes it more difficult than its classical counterpart is the fact that even the degenerate stabilizer operations are rather complicated. In the classical model, there are only two functions on one bit: doing nothing or negating the input. However, in the stabilizer model, every subgroup of Hadamard and phase gates is its own class of functions. The set of these transformations turns out to be isomorphic to the permutation group on 4 elements. We give an exact lattice of these subgroups in terms of their stabilizer generators in Figure 4-1.

Furthermore, we know several interesting things about the classification that arise from the quantum nature of the ancillary qubits. For instance, classes that were separate in the classical setting collapse in the quantum setting. More surprisingly, the phase gate can be generated from the CNOT and Hadamard gates using an appropriate ancillary state. Oddly, this implies that we, in fact, do not need any matrix representing the gates in our generating set to contain any complex values in order to generate the full stabilizer group.

We also give a complete classification of the classical affine subgroups in the quantum setting in Figure 4-3. Because of effects like those mentioned above, this is not simply a reiteration of the classical results. Nontrivial collapses *do* occur in the quantum setting, and
new techniques are required to prove separations.
Chapter 2

Precisely Describing the Model

2.1 Notation and Definitions

\( \mathbb{F}_2 \) means the field of 2 elements. We denote by \( e_1, \ldots, e_n \) the standard basis for the vector space \( \mathbb{F}_2^n \): that is, \( e_1 = (1,0,\ldots,0) \), etc. Let \( S_n \) be the set of \( n \)-qubit stabilizer states, in which states that differ by a global phase are considered equivalent. Let \( \{|x\} \mid x \in \{0,1\}^n \} \) be the set of computational basis states for \( S_n \).

Let \( x = x_1 \ldots x_n \) be an \( n \)-bit string. Then \( \overline{x} \) means \( x \) with all \( n \) of its bits inverted. Also, \( x \oplus y \) means bitwise XOR, \( x, y \) or \( xy \) means concatenation, \( x^k \) means the concatenation of \( k \) copies of \( x \), and \( |x| \) means the Hamming weight. The parity of \( x \) is \( |x| \mod 2 \). The inner product of \( x \) and \( y \) is the integer \( x \cdot y = x_1y_1 + \cdots + x_ny_n \). Note that

\[ x \cdot (y \oplus z) \equiv x \cdot y + x \cdot z \pmod{2}, \]

but the above need not hold if we are not working mod 2.

By \( \text{gar}(x) \), we mean garbage depending on \( x \): that is, “scratch work” that a reversible computation generates along the way to computing some desired function \( f(x) \). Typically,
the garbage later needs to be *uncomputed*. Uncomputing, a term introduced by Bennett [5], simply means running an entire computation in reverse, after the output $f(x)$ has been safely stored.

### 2.1.1 Gates

By a *(reversible)* gate, throughout this thesis we will mean a reversible transformation $G$ on the set of $k$-bit strings or $k$-qubit stabilizer states. Formally, the terms ‘gate’ and ‘reversible transformation’ will mean the same thing; ‘gate’ just connotes a reversible transformation that is particularly small or simple.

A gate is *nontrivial* if it does something other than permute the wires of the circuit, and *non-degenerate* if it does something other than permute the wires and/or apply single-(qu)bit operations to some subset of them.

Given two gates $G$ and $H$, their tensor product, $G \otimes H$, is a gate that applies $G$ and $H$ to disjoint sets of wires. We will often use the tensor product to produce a single gate that combines the properties of two previous gates. Also, we denote by $G^{\otimes t}$ the tensor product of $t$ copies of $G$. Note that a tensor product of degenerate operations is still degenerate.

The following definitions apply only in the classical model. A gate $G$ is *conservative* if it satisfies $|G(x)| = |x|$ for all $x$. A gate is *mod-$k$-respecting* if there exists a $j$ such that

$$|G(x)| \equiv |x| + j \pmod{k}$$

for all $x$. It’s *mod-$k$-preserving* if moreover $j = 0$. It’s *mod-preserving* if it’s mod-$k$-preserving for some $k \geq 2$, and *mod-respecting* if it’s mod-$k$-respecting for some $k \geq 2$.

As special cases, a mod-2-respecting gate is also called *parity-respecting*, a mod-2-preserving
gate is called \textit{parity-preserving}, and a gate $G$ such that

$$|G(x)| \neq |x| \pmod{2}$$

for all $x$ is called \textit{parity-flipping}. In Theorem 6, we will prove that parity-flipping gates are the \textit{only} examples of mod-respecting gates that are not mod-preserving.

The \textit{respecting number} of a gate $G$, denoted $k(G)$, is the largest $k$ such that $G$ is mod-$k$-respecting. By convention, if $G$ is conservative then $k(G) = \infty$, while if $G$ is non-mod-respecting then $k(G) = 1$. One easy consequence of these definitions is the following fact:

\textbf{Proposition 1} \textit{G is mod-$\ell$-respecting if and only if $\ell$ divides $k(G)$.}

\textbf{Proof.} If $\ell$ divides $k(G)$, then certainly $G$ is mod-$\ell$-respecting. Now, suppose $G$ is mod-$\ell$-respecting but $\ell$ does not divide $k(G)$. Then $G$ is both mod-$\ell$-respecting and mod-$k(G)$-respecting. So by the Chinese Remainder Theorem, $G$ is mod-$\text{lcm}(\ell, k(G))$-respecting. But this contradicts the definition of $k(G)$. \hfill \blacksquare

A gate $G$ is \textit{affine} if it implements an affine transformation over $\mathbb{F}_2$: that is, if there exists an invertible matrix $A \in \mathbb{F}_2^{k \times k}$, and a vector $b \in \mathbb{F}_2^k$, such that $G(x) = Ax \oplus b$ for all $x$. A gate is \textit{linear} if moreover $b = 0$. A gate is \textit{orthogonal} if it satisfies

$$G(x) \cdot G(y) \equiv x \cdot y \pmod{2}$$

for all $x, y$. (We will observe, in Lemma 8, that every orthogonal gate is linear.) Also, if $G(x) = Ax \oplus b$ is affine, then the \textit{linear part of $G$} is the linear transformation $G'(x) = Ax$. We call $G$ orthogonal in its linear part, mod-$k$-preserving in its linear part, etc. if $G'$ satisfies the corresponding invariant. A gate that is orthogonal in its linear part is also called an \textit{isometry}. 
2.1.2 Gate Classes

Let $S = \{G_1, G_2, \ldots\}$ be a set of gates, possibly on different numbers of inputs and possibly infinite. Let $\langle S \rangle = \langle G_1, G_2, \ldots \rangle$ be the class of transformations generated by $S$, where the only difference in the notion of generation in the models comes from the fact that we are working over fundamentally different fields. The classical transformations generated by $S$ can be defined as the smallest set of transformations $F : \{0, 1\}^n \rightarrow \{0, 1\}^n$ that satisfies the following closure properties:

1. **Base case.** $\langle S \rangle$ contains $S$, as well as the identity function $F(x_1 \ldots x_n) = x_1 \ldots x_n$ for all $n \geq 1$.

2. **Composition rule.** If $\langle S \rangle$ contains $F(x_1 \ldots x_n)$ and $G(x_1 \ldots x_n)$, then $\langle S \rangle$ also contains $F(G(x_1 \ldots x_n))$.

3. **Swapping rule.** If $\langle S \rangle$ contains $F(x_1 \ldots x_n)$, then $\langle S \rangle$ also contains all possible functions $\sigma(F(x_{\tau(1)} \ldots x_{\tau(n)}))$ obtained by permuting $F$’s input and output bits.

4. **Extension rule.** If $\langle S \rangle$ contains $F(x_1 \ldots x_n)$, then $\langle S \rangle$ also contains the function

   \[
   G(x_1 \ldots x_n, b) := (F(x_1 \ldots x_n), b),
   \]

   in which $b$ occurs as a “dummy” bit.

5. **Ancilla rule.** If $\langle S \rangle$ contains a function $F$ that satisfies

   \[
   F(x_1 \ldots x_n, a_1 \ldots a_k) = (G(x_1 \ldots x_n), a_1 \ldots a_k) \quad \forall x_1 \ldots x_n \in \{0, 1\}^n,
   \]

   for some smaller function $G$ and fixed “ancilla” string $a_1 \ldots a_k \in \{0, 1\}^k$ that do not depend on $x$, then $\langle S \rangle$ also contains $G$. (Note that, if the $a_i$’s are set to other values, then $F$ need not have the above form.)
The stabilizer transformations generated by $S$ can be defined as the smallest set of transformations $F : S_n \to S_n$ that satisfies the following closure properties:

1. **Base case.** $\langle S \rangle$ contains $S$, as well as the identity function $F(|\psi\rangle) = |\psi\rangle$ where $|\psi\rangle \in S_n$ for some $n \geq 1$.

2. **Composition rule.** If $\langle S \rangle$ contains $F : S_n \to S_n$ and $G : S_n \to S_n$, then $\langle S \rangle$ also contains $F \circ G$.

3. **Swapping rule.** If $\langle S \rangle$ contains $F : S_n \to S_n$, then $\langle S \rangle$ also contains $\text{SWAP}_{a,b}$ for all $a, b \leq n$.

4. **Extension rule.** If $\langle S \rangle$ contains $F : S_n \to S_n$, then $\langle S \rangle$ also contains the function $G : S_{n+1} \to S_{n+1}$ where $G = F \otimes I$.

5. **Ancilla rule.** If $\langle S \rangle$ contains a function $F : S_n \to S_n$ that satisfies

$$F(|\psi\rangle \otimes |\varphi\rangle) = G(|\psi\rangle) \otimes |\varphi\rangle \quad \forall |\psi\rangle \in S_m,$$

for some smaller function $G$ and fixed "ancilla" stabilizer state $|\varphi\rangle \in S_{n-m}$ that does not depend on $|\psi\rangle$, then $\langle S \rangle$ also contains $G$.

Note that because of reversibility and the discrete nature of the inputs, the set of $n$-wire transformations in $\langle S \rangle$ (for any $n$) always forms a group. Indeed, if $\langle S \rangle$ contains $F$, then clearly $\langle S \rangle$ contains all the iterates $F^2(x) = F(F(x))$, etc. But since there must be some positive integer $m$ such that $F^m(x) = x$, this means that $F^{m-1}(x) = F^{-1}(x)$. Thus, we do not need a separate rule stating that $\langle S \rangle$ is closed under inverses.

We say $S$ generates the transformation $F$ if $F \in \langle S \rangle$. We also say that $S$ generates $\langle S \rangle$. Given an arbitrary set $C$ of transformations, we call $C$ a class if $C$ is closed under rules (2)-(5) above: in other words, if there exists an $S$ such that $C = \langle S \rangle$. 

27
A circuit for the function $F$, over the gate set $S$, is an explicit procedure for generating $F$ by applying gates in $S$, and thereby showing that $F \in \langle S \rangle$. An example is shown in Figure 2-1. Circuit diagrams are read from left to right, with each wire that occurs in the circuit (which carry both the inputs and ancillas) represented by a horizontal line, and each gate represented by a vertical line.

![Figure 2-1: Generating a Controlled-Controlled-Swap gate from Fredkin](image)

2.1.3 Alternative Kinds of Generation

We now discuss two alternative notions of what it can mean for a classical reversible gate set to "generate" a transformation.

**Partial Gates.** A partial reversible gate is an injective function $H : D \rightarrow \{0, 1\}^n$, where $D$ is some subset of $\{0, 1\}^n$. Such an $H$ is consistent with a full reversible gate $G$ if $G(x) = H(x)$ whenever $x \in D$. Also, we say that a reversible gate set $S$ generates $H$ if $S$ generates any $G$ with which $H$ is consistent. As an example, COPY is the 2-bit partial reversible gate defined by the following relations:

$$\text{COPY}(00) = 00, \quad \text{COPY}(10) = 11.$$ 

If a gate set $S$ can implement the above behavior, using ancilla bits that are returned to their original states by the end, then we say $S$ "generates COPY"; the behavior on inputs 01 and 11 is irrelevant. Note that COPY is consistent with CNOT. One can think of COPY as a bargain-basement CNOT, but one that might be bootstrapped up to a full CNOT with further effort.
Generation With Garbage. Let $D \subseteq \{0, 1\}^m$, and $H : D \to \{0, 1\}^n$ be some function, which need not be injective or surjective, or even have the same number of input and output bits. Then we say that a reversible gate set $S$ generates $H$ with garbage if there exists a reversible transformation $G \in \langle S \rangle$, as well as an ancilla string $a$ and a function $\text{gar}$, such that $G(x, a) = (H(x), \text{gar}(x))$ for all $x \in D$. As an example, consider the ordinary 2-bit AND function, from $\{0, 1\}^2$ to $\{0, 1\}$. Since AND destroys information, clearly no reversible gate can generate it in the usual sense, but many reversible gates can generate AND with garbage: for instance, the Toffoli and Fredkin gates, as we saw in Section 1.2.

2.2 Stating the Classical Classification Theorem

In this section, we state the complete classical reversible bit classification to provide context for the main result of this thesis.

- NOT is the 1-bit gate that maps $x$ to $\overline{x}$.

- NOTNOT, or $\overline{\text{NOT}}^2$, is the 2-bit gate that maps $xy$ to $\overline{x}\overline{y}$. NOTNOT is a parity-preserving variant of NOT.

- CNOT (Controlled-NOT) is the 2-bit gate that maps $x, y$ to $x, y \oplus x$. CNOT is affine.

- CNOTNOT is the 3-bit gate that maps $x, y, z$ to $x, y \oplus x, z \oplus x$. CNOTNOT is affine and parity-preserving.

- Toffoli (also called Controlled-Controlled-NOT, or CCNOT) is the 3-bit gate that maps $x, y, z$ to $x, y, z \oplus xy$.

- Fredkin (also called Controlled-SWAP, or CSWAP) is the 3-bit gate that maps $x, y, z$ to $x, y \oplus x(y \oplus z), z \oplus x(y \oplus z)$. In other words, it swaps $y$ with $z$ if $x = 1$, and does nothing if $x = 0$. Fredkin is conservative: it never changes the Hamming weight.
- $C_k$ is a $k$-bit gate that maps $0^k$ to $1^k$ and $1^k$ to $0^k$, and all other $k$-bit strings to themselves. $C_k$ preserves the Hamming weight mod $k$. Note that $C_1 = \text{NOT}$, while $C_2$ is equivalent to $\text{NOT}\text{NOT}$, up to a bit-swap.

- $T_k$ is a $k$-bit gate (for even $k$) that maps $x$ to $\overline{x}$ if $|x|$ is odd, or to $x$ if $|x|$ is even. A different definition is

$$T_k(x_1 \ldots x_k) = (x_1 \oplus b_x, \ldots, x_k \oplus b_x),$$

where $b_x := x_1 \oplus \cdots \oplus x_k$. This shows that $T_k$ is linear. Indeed, we also have

$$T_k(x) \cdot T_k(y) \equiv x \cdot y + (k + 2) b_x b_y \equiv x \cdot y \pmod{2},$$

which shows that $T_k$ is orthogonal. Note also that, if $k \equiv 2 \pmod{4}$, then $T_k$ preserves Hamming weight mod 4: if $|x|$ is even then $|T_k(x)| = |x|$, while if $|x|$ is odd then

$$|T_k(x)| \equiv k - |x| \equiv 2 - |x| \equiv |x| \pmod{4}.$$

- $F_k$ is a $k$-bit gate (for even $k$) that maps $x$ to $\overline{x}$ if $|x|$ is even, or to $x$ if $|x|$ is odd. A different definition is

$$F_k(x_1 \ldots x_k) = \overline{T_k(x_1 \ldots x_k)} = (x_1 \oplus b_x \oplus 1, \ldots, x_k \oplus b_x \oplus 1)$$

where $b_x$ is as above. This shows that $F_k$ is affine. Indeed, if $k$ is a multiple of 4, then $F_k$ preserves Hamming weight mod 4: if $|x|$ is odd then $|F_k(x)| = |x|$, while if $|x|$ is even then

$$|F_k(x)| \equiv k - |x| \equiv |x| \pmod{4}.$$
Since $F_k$ is equal to $T_k$ in its linear part, $F_k$ is also an isometry.

We can now state the classification theorem.

**Theorem 2** Every set of reversible gates generates one of the following classes:

1. The trivial class (which contains only bit-swaps).
2. The class of all transformations (generated by Toffoli).
3. The class of all conservative transformations (generated by Fredkin).
4. For each $k \geq 3$, the class of all mod-$k$-preserving transformations (generated by $C_k$).
5. The class of all affine transformations (generated by CNOT).
6. The class of all parity-preserving affine transformations (generated by CNOTNOT).
7. The class of all mod-$4$-preserving affine transformations (generated by $F_4$).
8. The class of all orthogonal linear transformations (generated by $T_4$).
9. The class of all mod-$4$-preserving orthogonal linear transformations (generated by $T_6$).
10. Classes 1, 3, 7, 8, or 9 augmented by a NOTNOT gate (note: 7 and 8 become equivalent this way).
11. Classes 1, 3, 6, 7, 8, or 9 augmented by a NOT gate (note: 7 and 8 become equivalent this way).

Furthermore, all the above classes are distinct except when noted otherwise, and they fit together in the lattice diagram shown in Figure 2.2
Figure 2-2: The inclusion lattice of reversible gate classes
Chapter 3

Ubiquity of the Fredkin Gate

In this chapter, we focus on one critical aspect of the classical classification. Namely, we show that any non-affine gate set generates a Fredkin gate. Since it is clear that the number of affine gates is vanishingly small amongst all reversible transformations, this will show that with high probability any randomly chosen gate on a sufficiently large number of bits will generate a Fredkin gate. We prove the theorem in a sequence of smaller steps. In Section 3.1, we prove several general number-theoretic facts about the nature of reversible gates. In Section 3.2, we give proofs for results in [18, 7, 14] to show that any non-affine gate generates Fredkin provided we do not care about input dependent garbage. Leveraging results from the previous sections, we will then use slightly overly-sophisticated machinery in Section 3.3 to prove that any conservative non-affine gate set generates a Fredkin. Finally, in Section 3.4 we tweak that machinery to show that indeed every non-affine gate generates a Fredkin gate.
3.1 Number-theoretic Properties of Reversible Gates\footnote{Many of these results were proven in joint work with Scott Aaronson and Luke Schaeffer and were used extensively throughout the entire classification of reversible bit operations.}

We first need several easy propositions about the respecting number $k(G)$ of a gate.

**Proposition 3** $G$ is mod-$\ell$-respecting if and only if $\ell$ divides $k(G)$.

**Proof.** If $\ell$ divides $k(G)$, then certainly $G$ is mod-\(\ell\)-respecting. Now, suppose $G$ is mod-\(\ell\)-respecting but $\ell$ does not divide $k(G)$. Then $G$ is both mod-\(\ell\)-respecting and mod-\(k(G)\)-respecting. So by the Chinese Remainder Theorem, $G$ is mod-lcm ($\ell, k(G)$)-respecting. But this contradicts the definition of $k(G)$. \(\blacksquare\)

**Proposition 4** For all gates $G$ and $H$,

$$k(G \otimes H) = \gcd(k(G), k(H)).$$

**Proof.** Letting $\gamma = \gcd(k(G), k(H))$, clearly $G \otimes H$ is mod-$\gamma$-respecting. To see that $G \otimes H$ is not mod-\(\ell\)-respecting for any $\ell > \gamma$: by definition, $\ell$ must fail to divide either $k(G)$ or $k(H)$. Suppose it fails to divide $k(G)$ without loss of generality. Then $G$ cannot be mod-\(\ell\)-respecting, by Proposition 3. But if we consider pairs of inputs to $G \otimes H$ that differ only on $G$'s input, then this implies that $G \otimes H$ is not mod-\(\ell\)-respecting either. \(\blacksquare\)

**Proposition 5** Let $G$ be any non-conservative gate. Then for all integers $q$, there exists a $t$ such that $q \cdot k(G) \in W(G^{\otimes t})$.

**Proof.** Let $\gamma$ be the gcd of the elements in $W(G)$. Then clearly $G$ is mod-$\gamma$-respecting. By Proposition 3, this means that $\gamma$ must divide $k(G)$.\footnote{Indeed, by using Theorem 6, one can show that $\gamma = k(G)$, except in the special case that $G$ is parity-flipping, where we have $\gamma = 1$ and $k(G) = 2$.} \(\blacksquare\)
Call a reversible transformation a \textit{mod-shifter} if it always shifts the Hamming weight \text{mod} \ k of its input string by some fixed, nonzero amount. When \( k = 2 \), clearly mod-shifters exist: indeed, the humble NOT gate satisfies \( |\text{NOT} (x)| \equiv |x| + 1 \) \text{ (mod} 2 \text{)} for all \( x \in \{0, 1\} \), and likewise for any other parity-flipping gate. However, we now show that \( k = 2 \) is the only possibility: mod-shifters do not exist for any larger \( k \).

\textbf{Theorem 6} There are no mod-shifters for \( k \geq 3 \). In other words: let \( G \) be a reversible transformation on \( n \)-bit strings, and suppose

\[ |G (x)| \equiv |x| + j \text{ (mod} k \text{)} \]

for all \( x \in \{0, 1\}^n \). Then either \( j = 0 \) or \( k = 2 \).

\textbf{Proof.} Suppose the above equation holds for all \( x \). Then introducing a new complex variable \( z \), we have

\[ z^{|G(x)|} \equiv z^{|x| + j} \text{ (mod} (z^k - 1) \text{)} \]

(since working \text{mod} \( z^k - 1 \) is equivalent to setting \( z^k = 1 \)). Since the above is true for all \( x \),

\[ \sum_{x \in \{0,1\}^n} z^{|G(x)|} \equiv \sum_{x \in \{0,1\}^n} z^{|x|} z^j \text{ (mod} (z^k - 1) \text{)} . \]

(3.1)

By reversibility, we have

\[ \sum_{x \in \{0,1\}^n} z^{|G(x)|} = \sum_{x \in \{0,1\}^n} z^{|x|} = (z + 1)^n . \]

Therefore equation (3.1) simplifies to

\[ (z + 1)^n (z^j - 1) \equiv 0 \text{ (mod} (z^k - 1) \text{)} . \]
Now, since $z^k - 1$ has no repeated roots, it can divide $(z + 1)^n (z^j - 1)$ only if it divides $(z + 1) (z^j - 1)$. For this we need either $j = 0$, causing $z^j - 1 = 0$, or else $j = k - 1$ (from degree considerations). But it is easily checked that the equality

$$z^k - 1 = (z + 1) (z^{k-1} - 1)$$

holds only if $k = 2$. ■

We now show another general fact about the nature of reversible gates concerning their effect on inner products. We have seen that there exist orthogonal gates (such as the $T_k$ gates), which preserve inner products mod 2. We now show that no reversible gate that changes Hamming weights can preserve inner products mod $k$ for any $k \geq 3$. We then observe that, if a reversible gate is orthogonal, then it indeed must be linear.

**Theorem 7** Let $G$ be a non-conservative $n$-bit reversible gate, and suppose

$$G(x) \cdot G(y) \equiv x \cdot y \pmod{k}$$

for all $x, y \in \{0, 1\}^n$. Then $k = 2$.

**Proof.** As in the proof of Theorem 6, we promote the congruence to a congruence over complex polynomials:

$$z^{G(x) \cdot G(y)} \equiv z^{x \cdot y} \pmod{(z^k - 1)}$$

Fix a string $x \in \{0, 1\}^n$ such that $|G(x)| > |x|$, which must exist because $G$ is non-conservative. Then sum the congruence over all $y$:

$$\sum_{y \in \{0, 1\}^n} z^{G(x) \cdot G(y)} \equiv \sum_{y \in \{0, 1\}^n} z^{x \cdot y} \pmod{(z^k - 1)}.$$
The summation on the right simplifies as follows.

\[
\sum_{y \in \{0,1\}^n} z^y x = \prod_{i=1}^n \sum_{y_i \in \{0,1\}} z^{x_i y_i} = \prod_{i=1}^n (1 + z^{x_i}) = (1 + z)^{|x|} 2^{n-|x|}.
\]

Similarly,

\[
\sum_{y \in \{0,1\}^n} z^{G(x)-G(y)} = (1 + z)^{|G(x)|} 2^{n-|G(x)|},
\]

since summing over all \(y\) is the same as summing over all \(G(y)\). So we have

\[
(1 + z)^{|G(x)|} 2^{n-|G(x)|} \equiv (1 + z)^{|x|} 2^{n-|x|} \pmod{(z^k - 1)},
\]

\[
0 \equiv (1 + z)^{|x|} 2^{n-|G(x)|} \left(2^{|G(x)|-|x|} - (1 + z)^{|G(x)|-|x|}\right) \pmod{(z^k - 1)},
\]

or equivalently, letting

\[
p(x) := 2^{|G(x)|-|x|} - (1 + z)^{|G(x)|-|x|},
\]

we find that \(z^k - 1\) divides \((1 + z)^{|x|} p(x)\) as a polynomial. Now, the roots of \(z^k - 1\) lie on the unit circle centered at 0. Meanwhile, the roots of \(p(x)\) lie on the circle in the complex plane of radius 2, centered at -1. The only point of intersection of these two circles is \(z = 1\), so that is the only root of \(z^k - 1\) that can be covered by \(p(x)\). On the other hand, clearly \(z = -1\) is the only root of \((1 + z)^{|x|}\). Hence, the only roots of \(z^k - 1\) are 1 and -1, so we conclude that \(k = 2\).

We now study reversible transformations that preserve inner products mod 2.

**Lemma 8** Every orthogonal gate \(G\) is linear.

**Proof.** Suppose

\[
G(x) \cdot G(y) \equiv x \cdot y \pmod{2}.
\]
Then for all \( x, y, z \),

\[
G(x \oplus y) \cdot G(z) \equiv (x \oplus y) \cdot z
\equiv x \cdot z + y \cdot z
\equiv G(x) \cdot G(z) + G(y) \cdot G(z)
\equiv (G(x) \oplus G(y)) \cdot G(z) \pmod{2}.
\]

But if the above holds for all possible \( z \), then

\[
G(x \oplus y) \equiv G(x) \oplus G(y) \pmod{2}.
\]

Theorem 7 and Lemma 8 have the following corollary.

**Corollary 9** Let \( G \) be any non-conservative, nonlinear gate. Then for all \( k \geq 2 \), there exist inputs \( x, y \) such that

\[
G(x) \cdot G(y) \not\equiv x \cdot y \pmod{k}.
\]

We conclude with a simple observation.

**Lemma 10** No nontrivial affine gate \( G \) is conservative.

**Proof.** Let \( G(x) = Ax \oplus b \); then \( |G(0^n)| = |0^n| = 0 \) implies \( b = 0^n \). Likewise, \( |G(e_i)| = |e_i| = 1 \) for all \( i \) implies that \( A \) is a permutation matrix. But then \( G \) is trivial. ■

### 3.2 Computing with Garbage

In this section we reprove some lemmas first shown by Seth Lloyd [18] in an unpublished 1992 technical report, and later rediscovered by Kerntopf et al. [14] and De Vos and Storm [7]. We will use these lemmas to show the power of non-affine gates.
Recall the notion of generating with garbage from Section 2.1.3.

**Lemma 11 ([18, 7])** Every nontrivial reversible gate $G$ generates NOT with garbage.

**Proof.** Let $G(x_1 \ldots x_n) = y_1 \ldots y_n$ be nontrivial, and let $y_i = f_i(x_1 \ldots x_n)$. Then it suffices to show that at least one $f_i$ is a non-monotone Boolean function. For if $f_i$ is non-monotone, then by definition, there exist two inputs $x, x' \in \{0, 1\}^n$, which are identical except that $x_j = 1$ and $x'_j = 0$ at some bit $j$, such that $f_i(x) = 0$ and $f_i(x') = 1$. But then, if we set the other $n - 1$ bits consistent with $x$ and $x'$, we have $y_i = \text{NOT}(x_j)$.

Thus, suppose by contradiction that every $f_i$ is monotone. Then reversibility clearly implies that $G(0^n) = 0^n$, and that the set of strings of Hamming weight 1 is mapped to itself: that is, there exists a permutation $\sigma$ such that $G(e_j) = e_{\sigma(j)}$ for all $j$. Furthermore, by monotonicity, for all $j \neq k$ we have $G(e_j \oplus e_k) \geq e_{\sigma(j)} \oplus e_{\sigma(k)}$. But then reversibility implies that $G(e_j \oplus e_k)$ can only be $e_{\sigma(j)} \oplus e_{\sigma(k)}$ itself, and so on inductively, so that we obtain $G(x_1 \ldots x_n) = x_{\sigma^{-1}(1)} \ldots x_{\sigma^{-1}(n)}$ for all $x \in \{0, 1\}^n$. But this means that $G$ is trivial, contradiction. $\blacksquare$

**Proposition 12 (folklore)** For all $n \geq 3$, every non-affine Boolean function on $n$ bits has a non-affine subfunction on $n - 1$ bits.

**Proof.** Let $f : \{0, 1\}^n \to \{0, 1\}$ be non-affine, and let $f_0$ and $f_1$ be the $(n - 1)$-bit subfunctions obtained by restricting $f$'s first input bit to 0 or 1 respectively. If either $f_0$ or $f_1$ is itself non-affine, then we are done. Otherwise, we have $f_0(x) = (a_0 \cdot x) \oplus b_0$ and $f_1(x) = (a_1 \cdot x) \oplus b_1$, for some $a_0, a_1 \in \{0, 1\}^{n-1}$ and $b_0, b_1 \in \{0, 1\}$. Notice that $f$ is non-affine if and only if $a_0 \neq a_1$. So there is some bit where $a_0$ and $a_1$ are unequal. If we now remove any of the other rightmost $n - 1$ input bits (which must exist since $n - 1 \geq 2$) from $f$, then we are left with a non-affine function on $n - 1$ bits. $\blacksquare$

**Lemma 13 ([18, 7])** Every non-affine reversible gate $G$ generates the 2-bit AND gate with garbage.
Proof. Certainly every non-affine gate is nontrivial, so we know from Lemma 11 that $G$ generates NOT with garbage. For this reason, it suffices to show that $G$ can generate some non-affine 2-bit gate with garbage (since all such gates are equivalent to AND under negating inputs and outputs). Let $G(x_1 \ldots x_n) = y_1 \ldots y_n$, and let $y_i = f_i(x_1 \ldots x_n)$. Then some particular $f_i$ must be a non-affine Boolean function. So it suffices to show that, by restricting $n - 2$ of $f_i$’s input bits, we can get a non-affine function on 2 bits. But this follows by inductively applying Proposition 12.

By using Lemma 13, it is possible to prove directly that the only classes that contain a CNOT gate are $\langle \text{CNOT} \rangle$ (i.e., all affine transformations) and $\langle \text{Toffoli} \rangle$ (i.e., all transformations)—or in other words, that if $G$ is any non-affine gate, then $\langle \text{CNOT}, G \rangle = \langle \text{Toffoli} \rangle$. However, we will skip this result, since it is subsumed by our later results.

Recall that COPY is the 2-bit partial gate that maps 00 to 00 and 10 to 11.

Lemma 14 ([18, 14]) Every non-degenerate reversible gate $G$ generates COPY with garbage.

Proof. Certainly every non-degenerate gate is nontrivial, so we know from Lemma 11 that $G$ generates NOT with garbage. So it suffices to show that there is some pair of inputs $x, x' \in \{0, 1\}^n$, which differ only at a single coordinate $i$, such that $G(x)$ and $G(x')$ have Hamming distance at least 2. For then if we set $x_i := z$, and regard the remaining $n - 1$ coordinates of $x$ as ancillas, we will find at least two copies of $z$ or $\bar{z}$ in $G(x)$, which we can convert to at least two copies of $z$ using NOT gates. Also, if all of the ancilla bits that receive a copy of $z$ were initially 1, then we can use a NOT gate to reduce to the case where one of them was initially 0.

Thus, suppose by contradiction that $G(x)$ and $G(x')$ are neighbors on the Hamming cube whenever $x$ and $x'$ are neighbors. Then starting from $0^n$ and $G(0^n)$, we find that every $G(e_i)$ must be a neighbor of $G(0^n)$, every $G(e_i \oplus e_j)$ must be a neighbor of $G(e_i)$ and $G(e_j)$, and so on, so that $G$ is just a rotation and reflection of $\{0, 1\}^n$. But that means $G$
is degenerate, contradiction. ■

We arrive a very important consequence from the above.

**Corollary 15** Any non-affine gate generates all reversible transformations with garbage.

**Proof.** Notice that any non-affine gate satisfies the conditions of Lemmas 11, 13, and 14. Therefore, we have a NOT, AND, and COPY gate in our gate set provided that we allow input-dependent garbage. Notice that we can generate any individual output bit of a reversible function using NOT and AND gates, which are universal. The COPY gate allows us to repeat this procedure for each output bit of the reversible function. ■

### 3.3 Conservative Generates Fredkin

In this section, we prove the following theorem.

**Theorem 16** Let \( G \) be any nontrivial conservative gate. Then \( G \) generates Fredkin.

The proof will be slightly more complicated than necessary, but we will then reuse parts of it in Section 3.4, when we show that every non-affine, non-conservative gate generates Fredkin.

Given a gate \( Q \), let us call \( Q \) strong quasi-Fredkin if there exist control strings \( a, b, c, d \) such that

\[
\begin{align*}
Q(a, 01) &= (a, 01), \quad (3.2) \\
Q(b, 01) &= (b, 10), \quad (3.3) \\
Q(c, 00) &= (c, 00), \quad (3.4) \\
Q(d, 11) &= (d, 11). \quad (3.5)
\end{align*}
\]
Lemma 17 Let $G$ be any nontrivial $n$-bit conservative gate. Then $G$ generates a strong quasi-Fredkin gate.

Proof. By conservativity, $G$ maps unit vectors to unit vectors, say $G(e_i) = e_{\pi(i)}$ for some permutation $\pi$. But since $G$ is nontrivial, there is some input $x \in \{0,1\}^n$ such that $x_i = 1$, but the corresponding bit $\pi(i)$ in $G(x)$ is 0. By conservativity, there must also be some bit $j$ such that $x_j = 0$, but bit $\pi(j)$ of $G(x)$ is 1. Now permute the inputs to make bit $j$ and bit $i$ the last two bits, permute the outputs to make bits $\pi(j)$ and $\pi(i)$ the last two bits, and permute either inputs or outputs to make $x$ match $G(x)$ on the first $n - 2$ bits. After these permutations are performed, $x$ has the form $w01$ for some $w \in \{0,1\}^{n-2}$. So

$$G(0^{n-2}, 01) = (0^{n-2}, 01),$$
$$G(w, 01) = (w, 10),$$
$$G(0^{n-2}, 00) = (0^{n-2}, 00),$$
$$G(1^{n-2}, 11) = (11^{n-2}, 11),$$

where the last two lines again follow from conservativity. Hence $G$ (after these permutations) satisfies the definition of a strong quasi-Fredkin gate. ■

Next, call a gate $C$ a catalyzer if, for every $x \in \{0,1\}^{2n}$ with Hamming weight $n$, there exists a "program string" $p(x)$ such that

$$C(p(x), 0^n1^n) = (p(x), x).$$

In other words, a catalyzer can be used to transform $0^n1^n$ into any target string $x$ of Hamming weight $n$. Here $x$ can be encoded in any manner of our choice into the auxiliary program string $p(x)$, as long as $p(x)$ is left unchanged by the transformation. The catalyzer itself cannot depend on $x$. 42
Lemma 18 Let \( Q \) be a strong quasi-Fredkin gate. Then \( Q \) generates a catalyzer.

Proof. Let \( z := 0^n1^n \) be the string that we wish to transform. For all \( i \in \{1, \ldots, n\} \) and \( j \in \{n + 1, \ldots, 2n\} \), let \( s_{ij} \) denote the operation that swaps the \( i^{\text{th}} \) and \( j^{\text{th}} \) bit of \( z \). Then consider the following list of “candidate swaps”:

\[
s_1,n+1, \ldots, s_{1,2n}, \ s_2,n+1, \ldots, s_{2,2n}, \ \ldots, \ s_{n,n+1}, \ldots, s_{n,2n}.
\]

Suppose we go through the list in order from left to right, and for each swap in the list, get to choose whether to apply it or not. It is not hard to see that, by making these choices, we can map \( 0^n1^n \) to any \( x \) such that \( |x| = n \), by pairing off the first 0 bit that should be 1 with the first 1 bit that should be 0, the second 0 bit that should be 1 with the second 1 bit that should be 0, and so on, and choosing to swap those pairs of bits and not any other pairs.

Now, let the program string \( p(x) \) be divided into \( n^2 \) registers \( r_1, \ldots, r_{n^2} \), each of the same size. Suppose that, rather than applying (or not applying) the \( t^{\text{th}} \) swap \( s_{ij} \) in the list, we instead apply the gate \( F \), with \( r_t \) as the control string, and \( z_i \) and \( z_j \) as the target bits. Then we claim that we can map \( z \) to \( x \) as well. If the \( t^{\text{th}} \) candidate swap is supposed to occur, then we set \( r_t := b \). If the \( t^{\text{th}} \) candidate swap is not supposed to occur, then we set \( r_t \) to either \( a \), \( c \), or \( d \), depending on whether \( z_i z_j \) equals 01, 00, or 11 at step \( t \) of the swapping process. Note that, because we know \( x \) when designing \( p(x) \), we know exactly what \( z_i z_j \) is going to be at each time step. Also, \( z_i z_j \) will never equal 10, because of the order in which we perform the swaps: we swap each 0 bit \( z_i \) that needs to be swapped with the first 1 bit \( z_j \) that we can. After we have performed the swap, \( z_i = 1 \) will then only be compared against other 1 bits, never against 0 bits.

Finally:

Lemma 19 Let \( G \) be any non-affine gate, and let \( C \) be any catalyzer. Then \( G + C \) generates Fredkin.
Proof. We will actually show how to generate any conservative transformation \( F : \{0, 1\}^n \rightarrow \{0, 1\}^n \).

Since \( G \) is non-affine, Corollary 15 implies that we can use \( G \) to compute any Boolean function, albeit possibly with input-dependent garbage.

Let \( x \in \{0, 1\}^n \). Then by assumption, \( C \) maps \( 0^n1^n \) to \( F(x) \overline{F(x)} \) using the program string \( p(F(x) \overline{F(x)}) \). Now, starting with \( x \) and ancillas \( 0^n1^n \), we can clearly use \( G \) to produce

\[
x, \text{gar}(x), p(F(x) \overline{F(x)}), 0^n1^n,
\]

for some garbage \( \text{gar}(x) \). We can then apply \( C \) to get

\[
x, \text{gar}(x), p(F(x) \overline{F(x)}), F(x), \overline{F(x)}.
\]

Uncomputing \( p(F(x) \overline{F(x)}) \) yields

\[
x, F(x), \overline{F(x)}.
\]

Notice that since \( F \) is conservative, we have \( |x, \overline{F(x)}| = n \). Therefore, there exists some program string \( p(x, \overline{F(x)}) \) that can be used as input to \( C^{-1} \) to map \( x, \overline{F(x)} \) to \( 0^n1^n \). Again, we can generate this program string using the fact that \( G \) is non-affine:

\[
x, F(x), \overline{F(x)}, \text{gar}(F(x)), p(x, \overline{F(x)}).
\]

Applying \( C^{-1} \) and then uncomputing, we get

\[
F(x), 0^n1^n
\]

which completes the proof. \( \blacksquare \)
By Lemma 10, every nontrivial conservative gate is also non-affine. Therefore, combining Lemmas 17, 18, and 19 completes the proof of Theorem 16, that every nontrivial conservative gate generates Fredkin.

### 3.4 Non-Conservative Generates Fredkin

Building on our work in Section 3.3, in this section we handle the non-conservative case, proving the following theorem.

**Theorem 20** Every non-affine, non-conservative gate generates Fredkin.

Thus, let $G$ be a non-affine, non-conservative gate. Starting from $G$, we will perform a sequence of transformations to produce gates that are “gradually closer” to Fredkin. Some of these transformations might look a bit mysterious, but they will culminate in a strong quasi-Fredkin gate, which we already know from Lemmas 18 and 19 is enough to generate a Fredkin gate (since $G$ is also non-affine).

The first step is to create a non-affine gate with two particular inputs as fixed points.

**Lemma 21** Let $G$ be any non-affine gate on $n$ bits. Then $G$ generates a non-affine gate $H$ on $n^2$ bits that acts as the identity on the inputs $0^{n^2}$ and $1^{n^2}$.

**Proof.** We construct $H$ as follows:

1. Apply $G^\otimes n$ to $n^2$ input bits. Let $G_i$ be the $i^{th}$ gate in this tensor product.

2. For all $i \in [n - 1]$, swap the $i^{th}$ output bit of $G_i$ with the $i^{th}$ output bit of $G_n$.

3. Apply $(G^{-1})^\otimes n$.

It is easy to see that $H$ maps $0^{n^2}$ to $0^{n^2}$ and $1^{n^2}$ to $1^{n^2}$. (Indeed, $H$ maps every input that consists of an $n$-bit string repeated $n$ times to itself.) To see that $H$ is also non-affine, first
notice that $G^{-1}$ is non-affine. But we can cause any input $x = x_1 \ldots x_n$ that we like to be fed into the final copy of $G^{-1}$, by encoding that input “diagonally,” with each $G_i$ producing $x_i$ as its $i^{th}$ output bit. Therefore $H$ is non-affine. ■

As a remark, with all the later transformations we perform, we will want to maintain the property that the all-0 and all-1 inputs are fixed points. Fortunately, this will not be hard to arrange.

Let $H$ be the output of Lemma 21. If $H$ is conservative (i.e., $k(H) = \infty$), then $H$ already generates Fredkin by Theorem 16, so we are done. Thus, we will assume in what follows that $k(H)$ is finite. We will further assume that $H$ is mod-$k(H)$-preserving. By Theorem 6, the only gates $H$ that are not mod-$k(H)$-preserving are the parity-flipping gates—but if $H$ is parity-flipping, then $H \otimes H$ is parity-preserving, and we can simply repeat the whole construction with $H \otimes H$ in place of $H$.

Now we want to show that we can use $H$ to decrease the inner product between a pair of inputs by exactly 1 mod $m$, for any $m$ we like.

**Lemma 22** Let $H$ be any non-conservative, nonlinear gate. Then for all $m \geq 2$, there is a positive integer $t$, and inputs $x, y$, such that

$$H^\otimes t(x) \cdot H^\otimes t(y) - x \cdot y \equiv -1 \pmod{m}.$$ 

**Proof.** Let $m = p_1^{a_1} p_2^{a_2} \ldots p_s^{a_s}$, where each $p_i$ is a distinct prime. By Corollary 9, we know that for each $p_i$, there is some pair of inputs $x_i, y_i$ such that

$$H(x_i) \cdot H(y_i) \neq x_i \cdot y_i \pmod{p_i}.$$ 

In other words, letting

$$\gamma_i := H(x_i) \cdot H(y_i) - x_i \cdot y_i,$$
we have $\gamma_i \not\equiv 0 \pmod{p_i}$ for all $i \in \{1, \ldots, s\}$. Our goal is to find an $(x, y)$ such that

$$H^{\otimes t}(x) \cdot H^{\otimes t}(y) - x \cdot y \equiv -1 \pmod{m}.$$ 

To do so, it suffices to find nonnegative integers $d_1, \ldots, d_s$ that solve the equation

$$\sum_{i=1}^{s} d_i \gamma_i \equiv -1 \pmod{m}. \quad (3.6)$$

Here $d_i$ represents the number of times the pair $(x_i, y_i)$ occurs in $(x, y)$. By construction, no $p_i$ divides $\gamma_i$, and since the $p_i$'s are distinct primes, they have no common factor. This implies that $\gcd(\gamma_1, \ldots, \gamma_s, m) = 1$. So by the Chinese Remainder Theorem, a solution to (3.6) exists. 

Note also that, if $H$ maps the all-0 and all-1 strings to themselves, then $H^{\otimes t}$ does so as well.

To proceed further, it will be helpful to introduce some terminology. Suppose that we have two strings $x = x_1 \ldots x_n$ and $y = y_1 \ldots y_n$. For each $i$, the pair $x_i y_i$ has one of four possible values: 00, 01, 10, or 11. Let the type of $(x, y)$ be an ordered triple $(a, b, c) \in \mathbb{Z}^3$, which simply records the number of occurrences in $(x, y)$ of each of the three pairs 01, 10, and 11. (It will be convenient not to keep track of 00 pairs, since they don’t contribute to the Hamming weight of either $x$ or $y$.) Clearly, by applying swaps, we can convert between any pairs $(x, y)$ and $(x', y')$ of the same type, provided that $x, y, x', y'$ all have the same length $n$.

Now suppose that, by repeatedly applying a gate $H$, we can convert some input pair $(x, y)$ of type $(a, b, c)$ into some pair $(x', y')$ of type $(a', b', c')$. Then we say that $H$ generates the slope $(a' - a, b' - b, c' - c)$. 

47
Note that, if $H$ generates the slope $(p, q, r)$, then by inverting the transformation, we can also generate the slope $(-p, -q, -r)$. Also, if $H$ generates the slope $(p, q, r)$ by acting on the input pair $(x, y)$, and the slope $(p', q', r')$ by acting on $(x', y')$, then it generates the slope $(p + p', q + q', r + r')$ by acting on $(xx', yy')$. For these reasons, the achievable slopes form a 3-dimensional lattice—that is, a subset of $\mathbb{Z}^3$ closed under integer linear combinations—which we can denote $L(H)$.

What we really want is for the lattice $L(H)$ to contain a particular point: $(1, 1, -1)$. Once we have shown this, we will be well on our way to generating a strong quasi-Fredkin gate. We first need a general fact about slopes.

**Lemma 23** Let $H$ map the all-0 input to itself. Then $L(H)$ contains the points $(k(H), 0, 0)$, $(0, k(H), 0)$, and $(0, 0, k(H))$.

**Proof.** Recall from Proposition 5 that there exists a $t$, and an input $w$, such that $|H^{\otimes t}(w)| = |w| + k(H)$. Thus, to generate the slope $(k(H), 0, 0)$, we simply need to do the following:

- Choose an input pair $(x, y)$ with sufficiently many $x_iy_i$ pairs of the forms 10 and 00.
- Apply $H^{\otimes t}$ to a subset of bits on which $x$ equals $w$, and $y$ equals the all-0 string.

Doing this will increase the number of 10 pairs by $k(H)$, while not affecting the number of 01 or 11 pairs.

To generate the slope $(0, k(H), 0)$, we do exactly the same thing, except that we reverse the roles of $x$ and $y$.

Finally, to generate the slope $(0, 0, k(H))$, we choose an input pair $(x, y)$ with sufficiently many $x_iy_i$ pairs of the forms 11 and 00, and then use the same procedure to increase the number of 11 pairs by $k(H)$. ■

We can now prove that $(1, 1, -1)$ is indeed in our lattice.
Lemma 24 Let $H$ be a mod-$k$ ($H$)-preserving gate that maps the all-0 input to itself, and suppose there exist inputs $x, y$ such that

$$H(x) \cdot H(y) - x \cdot y \equiv -1 \pmod{k(H)}.$$ 

Then $(1, 1, -1) \in \mathcal{L}(H)$.

Proof. The assumption implies directly that $H$ generates a slope of the form $(p, q, -1 + rk(H))$, for some integers $p, q, r$. Thus, Lemma 23 implies that $H$ also generates a slope of the form $(p, q, -1)$, via some gate $G \in \langle H \rangle$ acting on inputs $(x, y)$. Now, since $H$ is mod-$k(H)$-preserving, we have $|G(x)| \equiv |x| \pmod{k(H)}$ and $|G(y)| \equiv |y| \pmod{k(H)}$. But this implies that $p \equiv 1 \pmod{k(H)}$ and $q \equiv 1 \pmod{k(H)}$. So, again using Lemma 23, we can generate the slope $(1, 1, -1)$.

Combining Lemmas 21, 22, and 24, we can summarize our progress so far as follows.

Corollary 25 Let $G$ be any non-affine, non-conservative gate. Then either $G$ generates Fredkin, or else it generates a gate $H$ that maps the all-0 and all-1 inputs to themselves, and that also satisfies $(1, 1, -1) \in \mathcal{L}(H)$.

We now explain the importance of the lattice point $(1, 1, -1)$. Given a gate $Q$, let us call $Q$ weak quasi-Fredkin if there exist strings $a$ and $b$ such that

$$Q(a, 01) = (a, 01),$$
$$Q(b, 01) = (b, 10).$$

Then:

Lemma 26 A gate $H$ generates a weak quasi-Fredkin gate if and only if $(1, 1, -1) \in \mathcal{L}(H)$. 

49
Proof. If $H$ generates a weak quasi-Fredkin gate $Q$, then applying $Q$ to the input pair $(a, 01)$ and $(b, 01)$ directly generates the slope $(1, 1, -1)$. For the converse direction, if $H$ generates the slope $(1, 1, -1)$, then by definition there exists a gate $Q \in \langle H \rangle$, and inputs $x, y$, such that $|Q(x)| = |x|$ and $|Q(y)| = |y|$, while

$$Q(x) \cdot Q(y) = x \cdot y - 1.$$ 

In other words, applying $Q$ decreases by one the number of 1 bits on which $x$ and $y$ agree, while leaving their Hamming weights the same. But in that case, by permuting input and output bits, we can easily put $Q$ into the form of a weak quasi-Fredkin gate. 

Next, recall the definition of a strong quasi-Fredkin gate from Section 3.3. Then combining Corollary 25 with Lemma 26, we have the following.

**Corollary 27** Let $G$ be any non-affine, non-conservative gate. Then either $G$ generates Fredkin, or else it generates a strong quasi-Fredkin gate.

**Proof.** Combining Corollary 25 with Lemma 26, we find that either $G$ generates Fredkin, or else it generates a weak quasi-Fredkin gate that maps the all-0 and all-1 strings to themselves. But such a gate is a strong quasi-Fredkin gate, since we can let $c$ be the all-0 string and $d$ be the all-1 string. 

Combining Corollary 27 with Lemmas 18 and 19 now completes the proof of Theorem 20: that every non-affine, non-conservative gate generates Fredkin. However, since every non-affine, conservative gate generates Fredkin by Theorem 16, we get the following even broader corollary.

**Corollary 28** Every non-affine gate generates Fredkin.
Chapter 4

Towards a Full Stabilizer Classification

We now shift our attention to the classification problem over stabilizer states. First, it is helpful to recognize that the stabilizer model is not a strictly more powerful generalization of the classical model. Although we now have the ability to create superpositions of basis states using the Hadamard gate, we are restricted to those entangling gates that can be constructed from CNOT gates. Even though the CNOT gate maps each computational basis state to another computational basis state via a simple affine operation over $\mathbb{F}_2$, one might suspect that with the aid of the Hadamard and phase gates one could achieve a non-affine transformation of computational basis states. Nevertheless, it is not too hard to see [8, 19] that the computational basis states of a stabilizer state must form an affine subspace over $\mathbb{F}_2$, eliminating this possibility.

But what happens when we restrict ourselves to only those gate sets that can be generated by CNOT, as in Figure 2.2? Is that a sub-lattice of the inclusion diagram for stabilizer gates? That answer turns out to be no due to somewhat nonintuitive effects introduced by the fact that we now have ancillary qubits (rather than just bits) at our disposal. We explore some of these consequences in Section 4.2, and give a complete description of the “classical” gate sets (i.e. those whose gates are compositions of CNOT gates) which have access to quantum
ancillary states in Section 4.3. We first start with the classification of just the single-qubit operations, and in doing so we hope to show that the stabilizer states admit a fairly rich structure.

4.1 Single-qubit Gate Sets

Recall that the single-qubit stabilizer gates are those gates that map Pauli operators to Pauli operations under conjugation. For stabilizer gate $G$ and Pauli operations $P$ and $P'$ we cannot have $GPG^\dagger = \pm i P'$. For suppose that we did. Then

$$I = GG^\dagger = GPPG^\dagger = GPG^\dagger GPG^\dagger = (\pm i P')(\pm i P') = -I$$

which is an obvious contradiction. Similarly, we cannot have $GPG^\dagger = I$ for $P \in \{X, Y, Z\}$ since conjugating both sides would imply that $P = I$. Finally we must have that $G$ preserves the product relation of the Pauli operators. If $P$ and $P'$ are two Pauli operators, then $GPP'G^\dagger = GPG^\dagger GP'G^\dagger$. That is, $G$'s action on any two Pauli operators determines its action on the third since $XY = i Z$. Combining the above pieces of information we see that any stabilizer operator can map $X$ to any of $\{X, Y, Z\}$ and then can map $Y$ to any of four other choices. This yields a group of order 24. The group corresponds to the number of rotational symmetries of the cuboctahedron\(^1\) or equivalently the number of ways to permute the diagonals of a cube [20]. Therefore, the group is isomorphic to the symmetry group $S_4$.

However in our model, we are concerned with the subgroups of the group rather than merely its order. There are, in fact, 30 subgroups of $S_4$ for which we give the corresponding lattice in Figure 4-1 in terms of the single qubit generators of the stabilizer group $H$ and $S$. For clarity, the generators are also expressed in terms of the Pauli matrices, which are

\(^1\)This can be more readily seen by appealing to the Bloch sphere representation of a qubit.
equivalent to the following sequence of gates up to global phases:

\[ X = HS^2H \quad Y = HS^2HS^2 \quad Z = S^2 \]

Figure 4-1: Hasse diagram for single qubit stabilizer subgroups

4.2 The Power of Ancillary Qubits

Upon first inspection, one might suspect that our model reduces to a purely group theoretic question concerning the transformations restricted to the input qubits such as the treatment in [13]. That is, it is not clear that the ability to compute on the ancillary qubits is useful.
A first observation is that at least classical (ancillas set to $|0\rangle$ and $|1\rangle$) are needed to perform some basic transformations. For instance, a gate set that included CNOT but not X would not be able to convert $|0\rangle$ to $|1\rangle$ without ancillary qubits. However, this is trivially possible given an ancilla set to $|1\rangle$; that is, $\text{CNOT}|1\rangle|x\rangle = |1\rangle|x \oplus 1\rangle$.

However, in the previous example, the ancilla qubits and the input qubits remain separable throughout the computation. One might still suspect that becoming entangled with the ancilla qubits cannot help. The following theorem shows that is surprisingly not true.

**Theorem 29** The CNOT and H gates suffice to generate any possible stabilizer transformation on the input qubits.

**Proof.** It suffices to show that CNOT and H can generate the S gate with the help of ancillas. First consider the circuit shown in Figure 4-2. It can be checked straightforwardly that the following holds:

\[
|\bar{i}\rangle|0\rangle \rightarrow |\bar{i}\rangle|\bar{-i}\rangle \\
|\bar{i}\rangle|1\rangle \rightarrow |\bar{i}\rangle|i\rangle
\]

Notice that our ancilla qubit really did return to its initial value. Secondly, it is clear that we were able to perform a transformation which we couldn’t have done without the ancilla since CNOT and H matrices only involve real numbers and $|\bar{i}\rangle$ is complex. The matrix of the transformation is

\[
\frac{1}{\sqrt{2}} \begin{bmatrix}
1 & 1 \\
-i & i
\end{bmatrix} = \frac{1-i}{\sqrt{2}} \text{HSHS}
\]

This corresponds to a rotation of $\frac{2\pi}{3}$ around the line $(1, -1, 1)$ in the Bloch sphere. Furthermore, HS is a rotation of $\frac{\pi}{3}$ around the line $(1, -1, 1)$, so $(\text{HS})^3 = I$ up to global phase. Since we also have H in our gate set, we can perform two HSHS operations followed
by and H operation:

\[ H(HSHS)(HSHS) = (HH)S(HSHSHS) = ISI = S. \]

\[ \]

\[ \]

Figure 4-2: Leveraging the ancillary qubit \(|i\rangle\) to form the HSHS gate

Perhaps then, our model is too strong in that the CNOT gate can always leverage the ancillary qubits to lift the gate set to the full power of the stabilizer group. We show now that this is indeed not the case. Call a gate \( G \) **magnitude-preserving** if it preserves the magnitude of the amplitudes of basis states on which it acts. To be more concrete, suppose we have the state \( \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle \), which is transformed to the state \( \sum_{x \in \{0,1\}^n} \beta_x |x\rangle \) under \( G \). \( G \) is magnitude preserving if the multisets \( \{\alpha_x : x \in \{0,1\}^n\} \) and \( \{\beta_x : x \in \{0,1\}^n\} \) are preserved for any starting state.

**Theorem 30** Any gate set that consists solely of magnitude-preserving gates cannot generate the \( H \) gate (or any other gate that can create a superposition of computational basis states from a state that is not a superposition of computational basis states).

**Proof.** Suppose the initial state of our system is \( \sum_{a \in \{0,1\}^m} \gamma_a |a\rangle |x\rangle \) where the \(|a\rangle\) qubits represent our ancillary state and \(|x\rangle\) our input as a computational basis state. Applying any circuit generated by our gate set, we get a final state \( \left( \sum_{a \in \{0,1\}^m} \gamma_a |a\rangle \right) \left( \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle \right) \) since our ancillary state must be returned to its initial value. Suppose that \( \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle \) is a superposition over basis states. We must have \( 0 < |\alpha_{x'}| < 1 \) for some \( x' \) since \( \sum_{x \in \{0,1\}^n} |\alpha_x| = 1 \). Let \(|a'\rangle\) be the ancillary basis state with the smallest non-zero magnitude\(^2\). The magnitude

\(^2\)Actually, from [8], we know that all stabilizer states have the same magnitudes on all their non-zero basis states.
on basis state $|a\rangle|x\rangle$ is $|\gamma_a\alpha_x\rangle$, but $|\gamma_a\alpha_x\rangle \leq |\gamma_a||\alpha_x\rangle < |\gamma_a\rangle$, which means our final state has a magnitude on a computational basis state that did not appear in our initial state, contradicting our notion of a magnitude-preserving gate set.

In particular, the theorem shows that no gate set consisting of built from the CNOT and phase gates can be universal.

4.3 Separating Classical Power from Quantum Power

Given the rather complex structure of the seemingly simple single-qubit operations (recall the diagram in Figure 4-1), a natural next step in the classification is separating the quantum single qubit operations, which introduce complex values and change of basis operations, from the “classical” operation CNOT. In fact, from the classical classification we know that the only distinct gate sets generated by the CNOT gate are the classes that are generated by CNOT, CNOTNOT, T₄, F₄, T₆, the identity, and combinations of the aforementioned gates with NOT and NOTNOT.

A first observation is that allowing stabilizer ancillas collapses the CNOT and CNOTNOT classes. Clearly, CNOT can generate CNOTNOT by simply applying the CNOT separately to each target qubit. To simulate a CNOT in the stabilizer setting, simply assign one of the targets of the CNOTNOT to an ancillary $|+\rangle$ state. Since $X$ stabilizes $|+\rangle$, the value of the ancillary qubit does not change based on the control qubit. Notice too that this implies that every class that contains a NOTNOT gate also contains a NOT gate.

Similarly, we have that the T₄ and T₆ collapse in the stabilizer model. We know that T₄ generates T₆, but by applying T₆ to four inputs plus the two qubit Bell state $\frac{|00\rangle+|11\rangle}{\sqrt{2}}$ we can simulate the T₄ gate with the T₆. Notice that $\frac{|00\rangle+|11\rangle}{\sqrt{2}}$ really is a stabilizer state since it is stabilized by the group generated by XX and ZZ.

Even more surprising, however, is the fact that the T₄ gate in the quantum setting
doesn't necessarily need to perform orthogonal linear transformations on the computational
basis states of the input as evidenced by the following theorem.

**Theorem 31** $T_4$ generates the NOT gate.

**Proof.** First show that $T$ can implement NOTNOT. Apply $T_4$ to the state $\frac{|01\rangle + |10\rangle}{\sqrt{2}}$ and then swap the two output bits. One can straightforwardly enumerate the possibilities for $x_1$ and $x_2$ to see that this does indeed produce the desired behavior. Once again, we can generate an NOT gate by simply letting one of the targets of the NOTNOT be the $|+\rangle$ state.

Similarly, applying $F_4$ to the state $\frac{|00\rangle + |11\rangle}{\sqrt{2}}$ and swapping the output bits, produces a NOTNOT gate. Therefore, the classes generated by $T_4, T_6$, and $F_4$ are all the same. The following theorem completes the classification by separating the class generated by $T_4$ and the class generated by CNOT.

**Theorem 32** CNOT generates a strictly larger class of functions than $T_4$.

**Proof.** We will show that no $T_4$ circuit can transform both $|01\rangle$ to $|01\rangle$ and $|10\rangle$ to $|11\rangle$. It is clear that CNOT can perform this transformation with the control on the first qubit and the target on the second. Suppose that $T_4$ could perform this transformation with some number of ancillary qubits. Let $T$ be the entire circuit built from $T_4$ gates and swap gates. Therefore $T$ is some orthogonal transformation. We have

$$
T\left(\sum_{a \in \{0,1\}^m} \gamma_a |a\rangle |01\rangle\right) = \sum_{a \in \{0,1\}^m} \gamma_a |a\rangle |01\rangle 
$$

$$
T\left(\sum_{a \in \{0,1\}^m} \gamma_a |a\rangle |10\rangle\right) = \sum_{a \in \{0,1\}^m} \gamma_a |a\rangle |11\rangle 
$$

Since $T$ is a magnitude preserving transformation, the number of non-zero $\gamma_a$ remains constant and no two basis vectors can ever interfere (i.e. cancel). For this reason, we can assume
that all the non-zero $\gamma_a$ are exactly the same value. Throughout this proof, whenever a specific $a \in \{0, 1\}^m$ is mentioned, it is assumed that $\gamma_a \neq 0$. Consider now equation (4.1). Although $T$ appears to be the identity, we do not necessarily have $T|a\rangle|01\rangle = |a\rangle|01\rangle$ since $T$ could permute the ancillary basis vectors. However, we do know that $T|a_1\rangle|01\rangle \neq |a_2\rangle|x\rangle$ for $x \in \{00, 10, 11\}$.

Therefore, we have $T|a_i\rangle|01\rangle = |a_j\rangle|01\rangle$ for $i, j$ such that $\gamma_{a_i} \neq 0$ and $\gamma_{a_j} \neq 0$. Since $T$ is reversible, we get $T^{-1}|a_j\rangle|01\rangle = |a_i\rangle|01\rangle$. Notice that $T^{-1}$ is also an orthogonal transformation so we can compute it using the $T_4$ gates. We want to map $|a_i\rangle|01\rangle$ to $|a_i\rangle|01\rangle$, but if we simply apply $T^{-1}$ we will reverse our entire circuit, which we still want to map $|10\rangle$ to $|11\rangle$. Therefore, create two new ancillary bits set to 0 and 1, and use those as the last two targets for the $T^{-1}$ transformation. This gives

$$T^{-1} \left( |01\rangle T \left( \sum_{a \in \{0,1\}^m} \gamma_a |a\rangle|01\rangle \right) \right) = T^{-1} \left( \sum_{a \in \{0,1\}^m} \gamma_a |01\rangle|a\rangle \right) |01\rangle = \sum_{a \in \{0,1\}^m} \gamma_a |01\rangle|a\rangle|01\rangle$$

and

$$T^{-1} \left( |01\rangle T \left( \sum_{a \in \{0,1\}^m} \gamma_a |a\rangle|10\rangle \right) \right) = T^{-1} \left( \sum_{a \in \{0,1\}^m} \gamma_a |01\rangle|a\rangle \right) |11\rangle = \sum_{a \in \{0,1\}^m} \gamma_a |01\rangle|a\rangle|11\rangle$$

so we can assume without loss of generality that in equation (4.1), we have that $T$ maps $|a_i\rangle|01\rangle$ to $|a_i\rangle|01\rangle$. This still does not mean that $T$ maps $|a_i\rangle|10\rangle$ to $|a_i\rangle|11\rangle$ However, suppose that it did for some $i$. Since $T$ is an orthogonal transformation we have that it preserves inner products of the basis strings. But,

$$T|a_i\rangle|01\rangle = |a_i\rangle|01\rangle$$

$$T|a_i\rangle|10\rangle = |a_i\rangle|11\rangle$$
which would imply that \( a_i \cdot a_i \equiv a_i \cdot a_i + 1 \pmod{2} \), a contradiction. We now generalize this argument for the case where no such \( i \) exists. There must exist some chain of the form

\[
T|a_1\rangle|10\rangle = |a_2\rangle|11\rangle \\
T|a_2\rangle|10\rangle = |a_3\rangle|11\rangle \\
\vdots \\
T|a_k\rangle|10\rangle = |a_1\rangle|11\rangle
\]

where the inner product relation of the first and last equation give that \( a_k \cdot a_1 + 1 \equiv a_1 \cdot a_2 \). Furthermore, the inner product of those two equations with the equation \( T|a_1\rangle|01\rangle = |a_1\rangle|01\rangle \) yields \( a_1 \cdot a_1 \equiv a_1 \cdot a_2 + 1 \) and \( a_1 \cdot a_k \equiv a_1 \cdot a_1 + 1 \). Combining these equations, we get \( a_1 \cdot a_2 \equiv a_1 \cdot a_k \), but we already have \( a_k \cdot a_1 + 1 \equiv a_1 \cdot a_2 \), a contradiction.  

Inspecting the proof, we see that we didn’t use anything in particular about the form of the ancillary qubits. This means, that we cannot even recover a CNOT operation if we use non-stabilizer ancillas such as \( |00\rangle + |01\rangle + |10\rangle \). This completes the classification of classical gates in the quantum setting. The simple lattice describing these gate sets is given in Figure 4-3.
Figure 4-3: The inclusion lattice of gates below the CNOT class using quantum ancillas
Chapter 5

Discussion and Future Work

This thesis attempts to give full classifications of quantum gates in one of their simplest settings—namely, when they are classically simulable. In the purely classical setting, which was settled completely in joint work with the author, Scott Aaronson, and Luke Schaeffer, we give a full exposition of the fact that any non-affine gate generates a Fredkin gate. In the stabilizer setting, we find that the use of stabilizer ancillary qubits gives rise to interesting behavior not seen or appreciated in the classical setting. In particular, we uncover a new notion of “magic” states that allow strict subsets of the stabilizer gates to generate all possible stabilizer circuits without damaging the magic state. Nevertheless, we do show certain settings in which even the resource of arbitrary stabilizer magic states does not suffice for universal stabilizer operations.

As discussed in Section 1.5.2, the central challenge left by this thesis is a complete classification of stabilizer gates based on the transformations they generate. In particular, we do not even have a complete classification of degenerate stabilizer operations. A simple-to-state and central question in this area asks whether or not the $H \otimes H$ can generate the Hadamard gate on a single qubit.

More ambitious yet, is a classification of all quantum gate sets acting on qubits, in terms
of which unitary transformations they can generate or approximate. Here, just like in this thesis, one should assume that qubit-swaps are free, and that arbitrary ancillas are allowed as long as they are returned to their initial states.

On the classical side, we have left completely open the problem of classifying reversible gate sets over non-binary alphabets. In the non-reversible setting, it was discovered in the 1950s (see [17]) that Post’s lattice becomes dramatically different and more complicated when we consider gates over a 3-element set rather than Boolean gates: for example, there is now an uncountable infinity of classes, rather than “merely” a countable infinity. Does anything similar happen in the reversible case? Even for reversible gates over (say) \( \{0, 1, 2\}^n \), we cannot currently give an algorithm to decide whether a given gate \( G \) generates another gate \( H \) any better than a triple-exponential-time algorithm that comes from clone theory, nor can we give reasonable upper bounds on the number of gates or ancillas needed in the generating circuit, nor can we answer basic questions like whether every class is finitely generated.
Bibliography


