

9-12-2014

Topological Code Architectures for Quantum Computation

Christopher Cesare

Follow this and additional works at: https://digitalrepository.unm.edu/phyc_etds

Recommended Citation

Cesare, Christopher. "Topological Code Architectures for Quantum Computation." (2014). https://digitalrepository.unm.edu/phyc_etds/9

This Dissertation is brought to you for free and open access by the Electronic Theses and Dissertations at UNM Digital Repository. It has been accepted for inclusion in Physics & Astronomy ETDs by an authorized administrator of UNM Digital Repository. For more information, please contact disc@unm.edu.

Christopher Anthony Cesare

Candidate

Physics and Astronomy

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Ivan Deutsch

, Chairperson

Andrew Landahl

Akimasa Miyake

Rouzbeh Allahverdi

Topological Code Architectures for Quantum Computation

by

Christopher Anthony Cesare

Bachelor of Science, Physics, University of California, Los Angeles, 2008

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Physics

The University of New Mexico

Albuquerque, New Mexico

July 2014

©2014, Christopher Anthony Cesare

Acknowledgments

It took the support of many people to complete graduate school and finish this dissertation. While I can't possibly provide an exhaustive list here, I'd like to take some space to mention the people that have been particularly meaningful and helpful to me.

First and foremost, I would like to thank my advisor Andrew Landahl for allowing me the freedom to pursue my own interests and for having the awareness to nudge me back on track when I was spinning my wheels. His passion for science and tireless devotion to details has been inspiring.

My time at the University of New Mexico has been filled with wonderful scientists and colleagues. In particular, I'd like to thank CQuIC and a local coterie of senior scientists: Ivan Deutsch, for transmitting his love of teaching and for serving as my dissertation committee chair; Carl Caves, for being a nearly unlimited source of wisdom and entertaining stories; Robin Blume-Kohout, for his insatiable curiosity and fearless questions; Rolando Somma, for his intuition about quantum information; and, finally, Akimasa Miyake and Rouzbeh Allahverdi, for serving as my committee members and taking the time to read this manuscript and provide constructive feedback.

I'd also like to thank the less senior members of the CQuIC group: Jonas Anderson, for serving as a second mentor while I was coming up to speed on research and for always taking the time to answer my questions; Josh Combes, for always providing considered perspectives on everything from career advice to philosophical arguments; Rob Cook, for humoring me on my rants; Chris Ferrie, for *not* humoring me on my rants and often providing thoughtful counter-rants slyly disguised as trolling (and occasionally providing entertaining trolling); Ben Baragiola, for all the tips on credit card mileage schemes and for being genuinely interested in all manner of interesting things; and Matthias Lang, for sharing my passion for beer and always forcing me to come up with better scientific explanations.

I feel lucky that I've continued to keep in touch with close high school and college friends, despite our dispersment around the country (and abroad). This list might get pretty long, so bear with me. From my group of high school friends, who knew me before I even imagined pursuing this degree, I'd like to thank: Chris Espinoza, for the shared interests, the discussions, the inside jokes, and the countless schemes we've hatched over the years; Nick Pierotti, for his selflessness and for always being excited; Nick Teramura, for his humor and for the willingness to always talk sports; Mark

Poyar, for his unfailing kindness and his proud declarations of belief; Liz Miller, for being one of my oldest friends and always making the effort to catch up; and, finally, Eliot Dewberry, Wes Dewberry, Carolyn Plou, Jason O’Grady, and Meg D’Amico, for all the good times in the past and the good times in the future.

From my group of college friends I’d like to thank: Bradley Matheson, for graciously hosting me on my many visits to Washington, D.C. and for always having something fun planned; Tommy Ursano, for having the best memory out of everyone I’ve met; Renata Skaletzky, for making me laugh more than anyone else; and Kyla Thomas, for her warmth, her kindness, and her inquisitiveness. Finally, even though we didn’t attend college together, we definitely *should* have: Bonnie Bontempo, for the trips, the adventures, the best New Year’s Eve party I’ve ever attended, and for being one of my best friends; and Katelyn Gallagher, for equal measures of level-headedness and an ability to unwind.

Most importantly, I’d like to thank my family: my brother-in-law, Tom Pfeiffer, for sparking my Belgian beer obsession and creating a new family party tradition; and my sister, Kelly Pfeiffer, for her love, support, and the way she makes being a parent look effortless. Finally, I quite literally could not have done this without my parents, Vince and Andi Cesare. Your constant and utter encouragement and love are worth more than I could ever repay.

I was supported during my time as a graduate student through NSF grant 0829944, through a Sandia National Laboratories Laboratory Directed Research and Development program, and as a Student Intern at Sandia. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-AC04-94AL85000.

Topological Code Architectures for Quantum Computation

by

Christopher Anthony Cesare

Bachelor of Science, Physics, University of California, Los Angeles, 2008

Doctor of Philosophy, Physics, University of New Mexico, 2014

Abstract

This dissertation is concerned with quantum computation using many-body quantum systems encoded in topological codes. The interest in these topological systems has increased in recent years as devices in the lab begin to reach the fidelities required for performing arbitrarily long quantum algorithms. The most well-studied system, Kitaev's toric code, provides both a physical substrate for performing universal fault-tolerant quantum computations and a useful pedagogical tool for explaining the way other topological codes work. In this dissertation, I first review the necessary formalism for quantum information and quantum stabilizer codes, and then I introduce two families of topological codes: Kitaev's toric code and Bombin's color codes. I then present three chapters of original work. First, I explore the distinctness of encoding schemes in the color codes. Second, I introduce a model of quantum computation based on the toric code that uses adiabatic interpolations between static Hamiltonians with gaps constant in the system size. Lastly, I describe novel state distillation protocols that are naturally suited for topological architectures and show that they provide resource savings in terms of the number of required ancilla states when compared to more traditional approaches to quantum gate approximation.

Contents

List of Figures	xiii
List of Tables	xxxii
1 Introduction	1
1.1 Setting the stage	2
1.2 Outline	6
1.3 List of papers and other projects	7
2 The Language of Quantum Information	9
2.1 Bits and qubits	10
2.2 Quantum measurements	15
2.3 Quantum channels	22
3 The Stabilizer Formalism and Quantum Error Correction	24
3.1 Overview	24

Contents

3.2	A classical interlude: the repetition code	26
3.3	Stabilizer generators and the codespace	29
3.4	Logical operators	32
3.5	Example: the Steane code	34
3.6	Decoding quantum codes	38
4	Topological Fault-Tolerance and Avenues to Universality	43
4.1	Overview	43
4.2	The toric code and planar surface codes	45
4.3	The color codes	57
4.4	Topological fault-tolerance	61
4.4.1	The laws of fault-tolerance	61
4.4.2	Topological code defects	64
4.4.3	Possibilities for self-correction	69
4.5	Topological code thresholds	71
4.6	Magic states and universality	75
4.7	Outline of the remaining chapters	80
5	Relationships Between Defect Encodings in Topological Color Codes	81
5.1	Triple-defect qubits	83
5.2	Failure of conversion via topology	90

Contents

5.3	Conclusion	96
6	Adiabatic Topological Quantum Computation	97
6.1	Introduction	98
6.2	Surface codes with defects	104
6.3	Adiabatic code deformations	111
6.4	Adiabatic code deformations of the surface code	113
6.4.1	Creation of a surface code without defects	114
6.4.2	Creation of a small $Z(X)$ defect in a $+1$ eigenstate of $Z(X)$	117
6.4.3	Adiabatic deformation of defects	121
6.4.4	Detaching and attaching surface code regions with defects	126
6.4.5	Creation of a $X(Z)$ defect in a ± 1 eigenstate of $Z(X)$	128
6.4.6	State injection into defects	130
6.5	Non-adiabatic procedures for surface code defects	134
6.5.1	Measurements of \overline{X} and \overline{Z} for defects	137
6.5.2	Heralded application of \overline{X} and \overline{Z} to defects	138
6.6	The completed model	139
6.7	Extension to 2D color codes	141
6.8	Conclusion	150
7	Direct Distillation of a New Family of Magic States	152

Contents

7.1	Introduction	153
7.2	Problem statement	157
7.3	Traditional quantum RISC architecture solution	159
7.3.1	Protocol	159
7.3.2	Resource analysis	164
7.4	Quantum CISC architecture solution	165
7.4.1	Protocol	166
7.4.2	Resource analysis	170
7.5	Conclusions	172
8	Summary and Outlook	176
8.1	Summary	176
8.2	Outlook	177
	Appendices	179
A	Basics of Algebraic Topology	180
A.1	Overview	180
A.2	Introduction	180
A.3	Homotopy and the fundamental group	181
A.4	Cellular complexes	184
A.4.1	Cellular construction of S^1 and S^2	184

Contents

A.5 Homology	186
A.6 Homology in topological codes	190
B Quantum Reed-Muller Codes	192
C Criteria for a Code to Admit Transversal $Z(\pi/2^k)$ Rotations	195
D Computational Tools for Code Deformation	199
D.1 Code deformation	199
D.2 Automated code deformation	201
D.3 Automated code deformation code listing	202
References	206

List of Figures

2.1	The truth table for the digital logic gate AND, demonstrating that it is not reversible. This is easily seen by noticing that given the output value 0, the input values for x and y are not uniquely determined.	11
2.2	A simple example of a quantum circuit that applies a sequence of unitary operators, called gates, to three qubits, represented by the three wires. Written as the action of unitary operators on the input state $ \psi\rangle$, the output state $ \phi\rangle = U_6U_5U_4U_3U_2U_1 \psi\rangle$	14
2.3	A typical set of gates discussed in relation to universal quantum computation.	15
2.4	A circuit with an input state $ \psi\rangle$ and two joint measurements: first qubits 1 and 2 are measured, followed by qubits 2 and 3.	20
2.5	A circuit with an input state $ \psi\rangle$ and two ancilla-coupled measurements.	20

List of Figures

3.1	A schematic of the process of encoding k logical qubits in n physical qubits. I include this mainly to reinforce the distinction between logical and physical and to remind the reader of the language. The 2^k -dimensional Hilbert space of the logical qubits is embedded in a precise way into the full 2^n -dimensional space, to be introduced and discussed below.	25
3.2	A schematic of the generic form assumed by quantum error correcting codes. A state $ \psi\rangle$ and ancilla qubits in the state $ 0\rangle \cdots 0\rangle$ are input. Next, the state is encoded using a specific quantum encoding circuit. After this, errors on the encoded data are diagnosed by performing measurements. The results of the measurements are processed on a classical device and a quantum correction is calculated. Lastly, the quantum correction is applied and the process repeats itself (except for the encoding step). The terminology used here is defined and discussed in this Chapter.	26
3.3	A schematic depicting the action of the bit flip channel on each bit x_i of a bit string x	27
3.4	A summary of the effects of single bit errors on the three-bit repetition parity checks.	28
3.5	An encoding circuit for the $[[7, 1, 3]]$ quantum Steane code.	35
3.6	The set of stabilizer generators for the Steane code.	35
3.7	A schematic depicting error correction in a concatenated code. First, error correction is performed on the level of concatenated blocks. Then, error correction is performed at the next highest level—the superblock for the physical concatenated blocks—and so forth. . . .	38

List of Figures

3.8	The S_X syndromes for the Steane code corresponding to all the single-qubit Z errors.	39
3.9	The S_Z syndromes for the Steane code corresponding to all the single-qubit X errors.	40
3.10	The S_X and S_Z syndromes for the Steane code corresponding to all the single-qubit Y errors.	41
4.1	The surface code defined on a square lattice. The edges correspond to qubits, the faces to Z -type stabilizer generators, and vertices to X -type stabilizer generators. Depicted is an $[[n, k, d]]$ quantum code with parameters $n = 60$ and $k = 0$. The trivial nature of this code is due to the fact that all the boundaries are “smooth,” as defined in the text.	46
4.2	An instance of the surface code encoding a single qubit. Note the difference in the boundary compared to Fig. 4.1.	48
4.3	A summary of the parameters for the surface code boundaries discussed in the text. The number of physical qubits is n , the number of stabilizer generators is $n - k$, and the number of logical qubits is k	49
4.4	The two logical operators for a surface code encoding one qubit. Logical Z is a string of Z operators connecting the two rough boundaries. Logical X is a ladder of X operators connecting the two smooth boundaries. It can also be thought of as a string of operators on the dual lattice (pictured here).	50

List of Figures

- 4.5 A sequence of deformations of a surface code logical operator. The dashed boxes represent Z type stabilizer generators defined on the faces. (a) shows a canonical choice for Z_L . In (b), this canonical choice has been multiplied by a stabilizer generator, deforming the string. (c) shows a further deformation along the same direction. Finally, (d) deforms Z_L one face in the other direction. 51
- 4.6 A single X error is detected by the two Z checks on adjacent faces. The violated Z check operators can be thought of as localized particles that live on the faces of the lattice. 53
- 4.7 For a chain of 3 X errors, still only two Z checks are violated. The particle description of violated checks still holds, and it is clear that the pair of particles is created at the endpoints of error chains. . . . 54
- 4.8 Two errors— E and E' —which lead to the same syndrome. Regardless of which of these the decoder decides is the “actual” error, the effect on the logical state is the same. If the real error is E , then the effect of the error and the correction E is $E^2|\psi_L\rangle = |\psi_L\rangle$. If the real error is E , then the effect of the error and the correction E' is $E'E|\psi_L\rangle = S_f|\psi_L\rangle = |\psi_L\rangle$ 55
- 4.9 A distance-7 instance of the 4.8.8 topological color code. The label 4.8.8 corresponds to number of edges of the three faces adjacent to vertices away from the boundary: one square and two octagons. Color codes are defined by graphs that are face 3-colorable with the qubits on vertices and the stabilizer generators on faces. Each face actually corresponds to two generators: one X type and one Z type. 58

List of Figures

- 4.10 X_L and Z_L for the distance 7 4.8.8 color code. Both the logical operators have the same shape and can even act on the same qubits. X_L is a tensor product of X operators on the bottom qubits and Z_L is a tensor product of Z operators on those same qubits. 59
- 4.11 The string-net representation of color code logical operators. This pattern of operators is equivalent—up to multiplication by stabilizer generators—to the operators in Fig. 4.10. This figure also introduces the notion of colored strings, which is a convenient tool for discussing the homology of the color code graphs. Blue strings connect to blue boundaries and travel through blue faces. The same holds for the other two colors. The color of the boundary is determined by the color that is absent along that boundary. Strings of a single color can split into two strings of the remaining two colors, as is seen in this example. 60
- 4.12 A naïve implementation of the $CNOT$ gate for the three qubit repetition code. This circuit correctly implements the $CNOT$, but it does so in a way that is not fault-tolerant. Note that a single X error on the top qubit can propagate to three more errors on the bottom code block. 62
- 4.13 The fault-tolerant application of $CNOT$ between two qubits encoded in the repetition code. In this case, a single X error can spread to at most one other qubit in the other block. Since each block can correct for a single X error, single errors remain correctable as they traverse the circuit. 63

List of Figures

4.14	An example of the two types of defects that can be created in the toric code. One, corresponding to a Z -type face check that has been removed, has a logical Z operator equivalent to the removed face check and a logical X operator that tethers the removed face to a smooth boundary. The other—which corresponds to a X -type vertex check that has been removed—has a logical X operator equivalent to the removed vertex check and a logical Z operator that tethers the vertex to a rough boundary. More elaborate defect encodings are possible: for example, two smooth defects can be used to encode a single qubit. The important thing is that new boundaries are introduced, and any defect introduction in the planar version will change the boundaries. Note here that there are two types of boundaries in the graph without defects, but that the unpunctured lattice encodes no qubits.	66
------	--	----

List of Figures

- 4.15 A measurement-based method for defect movement. (a) A single smooth defect in the toric code. (b) The movement procedure begins by measuring a single X on a qubit adjacent to the defect. This operator would normally anti-commute with two of the code's check operators, but due to the defect it anti-commutes with only a single operator: the remaining adjacent face check. (c) This check is thus removed from the set of stabilizer generators and replaced with $\pm X$ based on the result of the measurement. The defect now spans two faces. (d) The original face occupied by the defect is remeasured. It commutes with all the remaining stabilizer generators, and anti-commutes only with the newly introduced $\pm X$ single-qubit check. (e) The reintroduction of the four-body face check removes the operator $\pm X$ from the stabilizer generators and replaces it with $\pm ZZZZ$. The defect has now been moved over one face at the cost of potentially modifying one of the check operators by a -1 67
- 4.16 A sequence of code deformations showing that braiding a smooth defect around a rough defect acts on the logical space as $ZI \rightarrow ZI$. Frame (d) is equivalent to frame (a) by multiplication of stabilizer generators. 68
- 4.17 A sequence of code deformations showing that braiding a smooth defect around a rough defect acts on the logical space as $IZ \rightarrow ZZ$. Frame (d) is not equivalent to frame (a) by multiplication of stabilizer generators. 69
- 4.18 A sequence of code deformations showing that braiding a smooth defect around a rough defect acts on the logical space as $XI \rightarrow XX$. Frame (d) is not equivalent to frame (a) by multiplication of stabilizer generators. 70

List of Figures

- 4.19 A sequence of code deformations showing that braiding a smooth defect around a rough defect acts on the logical space as $IX \rightarrow IX$. Frame (d) is trivially equivalent to frame (a). 71
- 4.20 A green defect, either of X -type or Z -type in the 4.8.8 color code. The string-like operator connecting to a boundary shares the color of the removed face, and the encircling operator has one of the other two colors. 72
- 4.21 A typical error pattern for the toric code in a code-capacity setting. The decoder only gets to see the endpoints of error chains (the red dots). The decoder may identify the corrective action with an error that is consistent with the given syndrome. If the selected error and the actual error sum to something in the equivalence class of logical operators, then the algorithm fails; otherwise, it succeeds. For a given value of p , the value of p_L , the logical failure probability, is then the fraction of times the decoding fails. The same algorithm will be run with codes of different distance to study any finite size effects and to examine the sub-threshold error suppression achieved by moving to larger distance codes. 74

List of Figures

- 4.22 A flowchart for an on-demand quantum computer. First, a classical algorithm generates a description of the quantum circuit needed to solve some problem. Next, a quantum code is chosen to allow for non-ideal quantum gates and to provide easily fault-tolerant operations. The choice of code provides a natural universal gate basis to compile over, and the next step produces unitary approximations to all the gates in the circuit that can't be performed exactly. Finally, the gate basis will have some easy gates and some hard gates, and the hard gates are implemented via gate teleportation of distilled magic states. Each step introduces a resource overhead, explained more fully in the text. 76
- 4.23 A quantum circuit that uses the state $|M^G\rangle$ to apply the gate G or G^\dagger on $|\psi\rangle$. The gate applied depends on the outcome of measuring the first qubit, with a $+1$ heralding an application of G and a -1 heralding its inverse G^\dagger . This circuit can be made deterministic by allowing a corrective gate on the bottom qubit classically controlled on the -1 measurement outcome. However, for magic states yielding small Z rotations, this corrective gate (G^2) will most likely also have to be applied via a magic state. 78
- 5.1 A distance-7 instance of the 4.8.8 topological color code. The label 4.8.8 corresponds to number of edges of the three faces adjacent to vertices away from the boundary: one square and two octagons. Color codes are defined by graphs that are face 3-colorable with the qubits on vertices and the stabilizer generators on faces. Each face actually corresponds to two generators: one X type and one Z type. 82

List of Figures

5.2	The bulk of a 4.8.8 triangular color code abstracted away and the boundaries labeled by their color.	84
5.3	The string-net logical operator for a 4.8.8 triangular color code with the bulk structure abstracted away. These strings can be bent and pushed around, even split apart if the proper rules are followed. However, the colored strings must always end on the appropriate colored boundary.	84
5.4	The triple-defect encoding. Z -type stabilizer generators are removed from three regions, each with a different color. This creates three new logical qubits, but two are ignored as gauge degrees of freedom. The remaining qubit has logical operators chosen as shown, where there is additional freedom in the choice of Z_L . Due to the gauge fixing condition that the other two qubits are in the state $ 0\rangle$, Z_L could be chosen to be an enclosing loop around any of the three regions. Note the structure of X_L and its relationship to the structure of X_L for the string-net operators associated to the original qubit encoded in the surface (as shown in Fig. 5.3).	85
5.5	A logical Z operator for a triple-defect qubit.	86
5.6	An equivalent logical Z operator for a triple-defect qubit using the color code rules about the splitting of colored strings.	86
5.7	An equivalent logical Z operator for a triple-defect qubit using the color code rules about the splitting of colored strings and multiplication by stabilizer generators. There is no real distinction between these two rules, as the colored string splitting is just another instance of stabilizer generator multiplication.	87
5.8	An equivalent logical Z operator for a triple-defect qubit.	87

List of Figures

5.9	The three defects from Fig. 5.4 deformed so that they enclose a region of a color code. Note that the enclosed region now just looks like a smaller version of the full triangular code.	88
5.10	The three defects are deformed so that they touch and enclose a region of the code, isolating X_L in the process.	88
5.11	The three defects are deformed so that they touch and enclose a region of the code, partially isolating Z_L in the process but leaving a “byproduct operator” external to the isolated region. This byproduct operator must be measured before any logical operators are applied.	89
5.12	The three-qubit repetition code encoding circuit, showing that X_L for the single defect is mapped to XXX on the three single defects and Z_L for the single defect predictably is not modified. The propagation of Z'_L and Z''_L for the two other defect qubits and the gauge fixing condition—that $Z'_L = Z''_L = +1$ —allows for the triple defect Z_L to be a loop around any of the three defects.	90
5.13	A circuit allowing for a $CNOT$ to be performed between two defects of the same type. The labels c , c' , and c'' correspond to the color of the defects. Since a $CNOT$ cannot be performed between defects of the same color, c must be a different color than c'' and c' must be a different color than c'' . However, c and c' are allowed to be the same color.	91
5.14	After the circuit in Fig. 5.12, X_L is composed of the three string-like operators shown.	91
5.15	The red string-like part of X_L can be equivalently represented with a split into green and blue strings that fuse back together.	92

List of Figures

5.16	Through the multiplication of stabilizer generators, the blue and green splitting can be brought in contact with the boundaries of appropriate color.	92
5.17	Further multiplication by stabilizer generators allows for the endpoints of part of the split blue and green strings to merge with other blue and green endpoints coming from the other two defects.	93
5.18	A last round of stabilizer generator multiplication pulls the joined blue and green strings off the boundary and shows that the X_L is nearly the same as for the triple defect encoding.	93
5.19	The preparation of a color code surface encoding more than one logical qubit. This allows the different surface logical qubits to have different fixed gauges, which can be used to remove the leftover surface operators in Fig. 5.18.	94
5.20	A more realistic look at the deformations leading up to Fig. 5.18.	94
5.21	Stabilizer generator multiplication still allows the blue and green endpoints to be fused.	95
5.22	Unfortunately, when pulling the operators off the surface boundary with multiplication by stabilizer generators, the presence of other defects causes the strings to get “snagged.” Since these other qubits cannot all be gauged away, it is clear that this procedure will fail in general by introducing unintended logical errors.	95
6.1	Stabilizer generators (checks) for the surface code. An example of a plaquette check S_p and a vertex check S_v	105

List of Figures

- 6.2 A smooth (Z -type) defect. A logical Z operator is defined by a closed loop of Z s on the lattice that surrounds the defect and a logical X operator is defined by a connected path of X s on the dual lattice from the defect to a *smooth* (Z -type) boundary. Here we depict the removed region by removing that part of the lattice; this simply indicates that the code of the system factors into a code in the drawn region and a code inside of the defect. 106
- 6.3 A rough (X -type) defect. A logical X operator is defined by a closed loop of X s on the dual lattice that surrounds the defect and a logical Z operator is defined by a connected path of Z s on the lattice from the defect to a *rough* (X -type) boundary. 107
- 6.4 A large array of qubits in the state $|0\rangle$, each protected by a Hamiltonian $H = -\Delta Z$ 115
- 6.5 A large array of qubits, an 8-qubit region of which is now encoded in the surface code (shown in red). The boundaries of the code are chosen to be trivial so that the codespace is nondegenerate. 116
- 6.6 Growth of a small surface code region that involves only the qubits labeled 1, 2, and 3. 116
- 6.7 Operators involved in creating the defect which includes p_1 and p_2 . Note that the X operations span to a nearby smooth boundary. . . . 118
- 6.8 The four potential situations faced when growing a smooth defect. (a) Only one interior qubit. (b) Two interior qubits. (c) Three interior qubits. (d) Four interior qubits. 122

List of Figures

6.9	Growth of a smooth defect with only a single qubit on the interior after the procedure. (a) We wish to grow the defect to the indicated plaquette. (b) We adiabatically turn off the neighboring plaquette while (c) turning on a $-X$ Hamiltonian on the interior qubit. (d) This procedure causes modifications to the neighboring X checks which can be performed simultaneously with steps (b) and (c). . . .	123
6.10	Growth of a smooth defect with two qubits on the interior. (a) We wish to grow the defect to the indicated plaquette. (b) We adiabatically turn off the neighboring plaquette while (c) turning on two $-X$ Hamiltonians on the interior qubits. (d) This procedure causes modifications to the neighboring X checks.	124
6.11	Growth of a smooth defect with three qubits on the interior. The process is essentially the same as the one depicted in Fig. 6.10. . . .	125
6.12	Growth of a smooth defect with four qubits on the interior. The procedure is the same as the others, but there are no resulting X check modifications.	126
6.13	The setup for pinching off a smooth defect from a smooth wall. . . .	127
6.14	After the Hamiltonian deformation (or sequence of deformations), we are left with a surface code with trivial boundaries encoding a rough defect in the state $ +\rangle$	131
6.15	The interior qubit is adiabatically dragged to the state $ \psi\rangle$, the desired magic state.	132
6.16	The missing X checks are reintroduced to the code, causing neighboring Z checks to be removed. This new defect is now encoded in the state $ \psi\rangle$ with an encircling \bar{Z} and a single-qubit \bar{X}	133

List of Figures

6.17	Because \overline{X} was only a single-qubit operator, the two removed faces are moved apart and grown to combat decoherence.	134
6.18	One of the defects is merged with the boundary to make the standard single defect.	135
6.19	Distillation circuit for $T +\rangle$ states, constructed from the 15-qubit Reed-Muller code's encoding circuit.	136
6.20	A distillation protocol for $S +\rangle$ states based on the encoding circuit for the $[[7, 1, 3]]$ quantum Steane code.	137
6.21	An example of measuring \overline{Z} for a smooth qubit. It requires the preparation of a rough defect in a $+1$ eigenstate of \overline{Z} , as discussed in Sec. 6.4.5.	138
6.22	Circuit used to apply one of the Pauli operators to a smooth defect qubit. The outcome of the X measurement is $b \in \{0, 1\}$ and the outcome of the Z measurement is $a \in \{0, 1\}$. The outcomes of the measurement all occur with equal probability and the final state depends on these outcomes as shown. If an undesired operator is applied, the ancilla qubit is reinitialized and the circuit is implemented again. However, now the appropriate operator is the one which undoes the operator applied in the first iteration <i>and</i> applies the desired operator. (This, of course, will just be a different one of the four operators $\overline{X}^a \overline{Z}^b$.)	139
6.23	Gate teleportation circuit using the $T +\rangle$ state. The S correction needs to be performed half of the time and can be implemented in the same way using the state $S +\rangle = +i\rangle$ instead of $T +\rangle$ (and utilizing a Z correction half of the time).	140

List of Figures

- 6.24 Circuit for applying the Hadamard gate with an ancilla state. The correction A depends on the result of the measurement: if the measurement result is $+1$, then $A = X$, and if the measurement result is -1 , then $A = Z$. The S and S^\dagger gates can be performed using a circuit like the one in Fig. 6.23. 141
- 6.25 A lattice with colored plaquettes on which one can define the color codes. 141
- 6.26 Two adjacent green plaquettes are turned off while turning on the $-XX$ Hamiltonian shown, creating a Z -type defect in the $+1$ eigenstate of \bar{Z} (shown here as a light blue string encircling the defect). . 144
- 6.27 A Z -type red defect isolation procedure. The “drawbridge” in this case is the red plaquette adjacent to the yellow dots in the figure. The Z -type check on the red face is turned off while the two $-XX$ operators are turned on. The four-body X operator that is the product of the two $-XX$ Hamiltonians is in the stabilizer group before the evolution, and it is trivially in the stabilizer group of the code after the evolution. The blue and green plaquettes adjacent to the yellow dots are modified to be a four-body operators. (Also note that the X -type check on the red plaquette must also be turned off to fully isolate the region, and two $-ZZ$ Hamiltonians are turned on.) . . . 145
- 6.28 The creation of a defect region with both the X -type and Z -type green checks turned off. There are four interior qubits prepared in two Bell pairs by this procedure. 146
- 6.29 Four interior qubits are “exposed.” 147
- 6.30 The upper-right qubit is adiabatically dragged to the desired state. For instance, to inject $T|+\rangle$ states, $U = TH$ 147

List of Figures

6.31	The single-body terms in Fig. 6.30 are turned off while turning on the X -type and Z -type checks on the red plaquette.	148
6.32	\overline{X} and \overline{Z} after the reintroduction of the red plaquette in Fig. 6.31. .	149
6.33	The arrangement of the defect after reintroducing the X -type green plaquettes. \overline{X} is a string of Pauli X operators connecting two blue faces and \overline{Z} is a loop of Pauli Z operators around a blue face.	149
7.1	Circuit for teleporting the T gate from the $T +\rangle$ magic state.	161
7.2	Magic-state circuit for teleporting the Z_k gate.	166
7.3	Distillation circuit for $Z_3^\dagger +\rangle = \sqrt{T}^\dagger +\rangle$ states; it is the 31-qubit shortened quantum Reed-Muller code's encoding circuit applied to half of a Bell state followed by the logical Z_3 gate and M_X measurement of the qubits on this encoded half. The Z_3 gates are performed using the teleportation circuit depicted in Fig. 7.2. This circuit also distills $Z_3 +\rangle$ states on $Z_3^\dagger +\rangle$ inputs.	168
7.4	Log of the number of resource states required to synthesize the quantum $Z(\pi/2^k)$ gate as a function of the log of the inverse of the desired precision ϵ' for the RISC architecture described in the text and our CISC architecture.	173
A.1	The topologist's view of the world. See the link referenced in Footnote 1 to see an animation of the deformation.	181
A.2	Two slightly different spaces with different topologies. (a) A square X in \mathbb{R}^2 and a loop $\gamma : S^1 \rightarrow [0, 1] \times [0, 1]$. This loop can be continuously contracted to a point that is still in the space X . (b) A square X' in \mathbb{R}^2 and the same loop γ . The only difference between X and X' is that the latter has a single point removed. The loop γ can no longer be continuously deformed to a point that is also in X'	182

List of Figures

A.3	The construction of the circle S^1 as a cellular complex. The 0-skeleton is a single point, and the 1-skeleton is a single point and a line segment D^1 with its endpoints attached to the point.	185
A.4	The construction of the sphere S^2 as a cellular complex. The 0-skeleton contains only a single point, as does the 1-skeleton. The 2-skeleton consists of a single point and a single disk D^2 . The entire boundary of the disk gets glued to the single point, thus completing the construction.	186
A.5	A graph X composed of two points and four directed edges.	187
A.6	The space X' , nearly the same as the space X in Fig. A.5 but with a 2-cell attached to edges a and b . While not technically just a graph anymore, the attached 2-cell captures the notion of a graph face, and I'll use this intuition to relate this structure to the graphs embedded in surfaces that are used for topological codes.	189
A.7	Modified reproduction of Fig. 4.4 showing a single logical operator for a toric code encoding one qubit. Logical Z is a string of Z operators connecting the two rough boundaries. The points along the rough boundary are to be understood as either a "special set" of points that don't count as boundaries or, equivalently, as being identified with the trivial element of C_0 , the null point.	191

List of Figures

D.1	The surface codes corresponding to the example given in the code listing. The surface code on the left is going to be “merged” with the surface code on the right, effecting a joint logical measurement. This figure depicts the surface code in its medialized form, with the light purple face representing X -type stabilizer generators, the white faces representing Z -type stabilizer generators, and the qubits on the vertices.	201
-----	---	-----

List of Tables

7.1	Distillation polynomials (to most significant order) and distillation thresholds for distilling $Z_k^\dagger +\rangle$ states.	169
B.1	Parameters for (primal) Reed-Muller $R(r, m)$ codes, their duals $R(m-r-1, 1)$, and their CSS-combined shortened quantum versions $\overline{QRM}(r, m)$ for small values. Shortened $R(0, m)$ codes have no X generator, so the resulting quantum codes are just classical codes; they are referred to by \emptyset in the table.	194

Chapter 1

Introduction

*“Begin at the beginning,” the King
said gravely, “and go on till you
come to the end: then stop.”*

Lewis Carroll, *Alice in Wonderland*

This dissertation is a culmination of around six years of thinking about, speaking about, reading about, and arguing about ways to perform quantum computations, always while balancing the desire for the right output with the demands of overhead. By “perform quantum computations” I am not necessarily referring to any particular algorithm, but rather to the theoretical collection of tools and protocols that allow for the implementation of any quantum computation.

I’ve attempted to write this dissertation, where appropriate, in the same conversational manner that helped me learn about the wonderful variety of topics in the broader field of quantum information. In this vein, I’ve included many examples in lieu of rigorous proofs; when proofs are called for I mercilessly banish them to an appendix.

Chapter 1. Introduction

The setting for the story I’d like to tell is the realm of topological error correcting codes, and I will explore their properties, their uses, and their advantages. They also hold a fundamental interest for me because I’ve always found topology to be an exotic and exciting branch of mathematics. It is a branch I would know nothing about if I didn’t have the pedagogical tool of Kitaev’s toric code [Kit03] to provide a physical intuition for the abstract mathematics.

As an aspiring science writer, I’ve also endeavored to keep non-experts in quantum information in mind. My goal is to maintain technical accuracy while providing introductory material that is accessible to physicists in other fields. And so, as the King suggests, I begin at the beginning.

1.1 Setting the stage

While quantum computers promise to solve certain problems faster than the best known classical algorithms [Sho94, Kit95, Gro96, Mos08, Jor09], they are also of great fundamental interest. Barring major upheavals in the understanding of quantum mechanics and computational complexity, they likely¹ represent the limits of what physical devices are capable of computing. Though these fundamental aspects didn’t pique my original interest—I became excited about the field when I was an undergraduate via an article in *Scientific American* called “Computing with Quantum Knots” [Col06]—as a graduate student I have learned much about the foundational importance of quantum information as a whole. I also had no prior knowledge of topology entering graduate school, but the pictures in the magazine article of particles undergoing an elaborate dance captured my imagination, and as a student I’ve largely studied topological quantum error correcting codes. This dissertation collects

¹I may be channeling Scott Aaronson by overstating the relationship between physics and theoretical computer science. See Ref. [Aar13] for arguments in favor of promoting discoveries in computer science to physical laws.

Chapter 1. Introduction

some of the work I’ve done examining their uses as substrates for models of quantum computation.

A crucial step in early quantum computing research was Shor’s discovery of quantum error correction [Sho95]. This was arguably one of the most surprising results to come out of the nascent field of quantum information in the mid-1990s, following closely on the heels of Shor’s discovery of the quantum algorithm for factoring [Sho94]. Intuition suggested that it might be impossible to protect against an uncountable set of unitary errors, but the key insight—that performing quantum measurements could digitize the effect of errors—made the notion of large-scale quantum processing devices more feasible. The discovery of stabilizer codes by Gottesman [Got97] framed quantum error correction in the familiar language of classical error correction and allowed the structure of a large family of quantum codes to be understood in group-theoretic terms.

The original discovery of quantum error correcting codes was quickly followed by the proof of the fault-tolerance accuracy threshold theorem [ABO97, ABO99, Kit97a, Ste97, KLZ98, Pre98a, Pre98c] and the development of fault-tolerant methods [Sho96, Pre98a, Got98] that allow for arbitrarily long quantum computations with modest resource overheads. The insight, borrowed from classical fault-tolerance results, was to design circuits and protocols that did not allow errors to spread beyond one or two qubits. By preventing the catastrophic spread of errors, quantum error correcting codes are able to maintain information by correcting away errors faster than they can build up. Progress since then has refined the threshold theorem by proving it for concatenated codes [AGP06], calculating thresholds for various families of codes [CDT09, WFSH10], and developing fully fault-tolerant schemes for quantum computation with relatively high thresholds [FMMC12]. Topological codes have emerged as a promising path toward low-overhead fault-tolerant quantum computation in two dimensions, and one of the most promising topological codes for this

Chapter 1. Introduction

use is the toric code. (“Toric” here signifies that these codes were originally defined on the surface of a torus, although I will demonstrate in Chapter 4 that planar versions can also be defined.)

The introduction of toric codes by Kitaev in 1997 [Kit03] led to a litany of papers studying the various properties of these codes: the calculation of thresholds [DKLP02, Har04]; the development of different decoding algorithms [DKLP02, PC08, ES12, DCP10, Woo13]; the use of these codes in topological models of quantum computation [FSG09, FMMC12]; the study of topologically ordered Hamiltonians constructed from the stabilizer generators of these codes [BHM10]; and, more recently, the robustness of these codes to error channels with correlated errors [FM14, JNT⁺14].

Although topology has played a role in physical theories for decades² [LM77, Wit89], it is only recently that the language and concepts of topology have entered the field of quantum information. One recent example is the classification of topologically ordered states of matter by “string-net condensates,” which are quantum ground states of local Hamiltonians that obey certain rules [LW05]. These rules can be specified either by constraints on the algebra of quantum operators acting on the appropriate Hilbert space, or by a visual calculus of topologically inspired pictures. Kitaev’s toric code is an example of a system describable by these string-net condensates, and will serve as the main pedagogical tool for most of this dissertation.

The discovery of the toric code was followed by Bombin’s introduction of the topological color codes [BMD06]. This family of codes features several advantages over the toric codes, including a reduction in the required number of physical qubits to achieve a desired level of error protection and the availability of a larger set of naturally fault-tolerant gates. A disadvantage is their slightly lower threshold [LAR11] compared to the toric code, likely due to their more complicated syndrome

²Centuries, even. Kelvin argued [Tho69] that atoms were knots of the ether in the 1860s.

Chapter 1. Introduction

extraction circuits. Recent work by Duclos-Cianci *et al.* [BDCP12] demonstrates that these codes are equivalent to two copies of the toric code via a local unitary mapping. This implies that the problem of decoding the color codes can be mapped to the well-studied problem of decoding the toric code. Alternatively, as derived in Ref. [LAR11], the decoding problem can be mapped to the problem of solving a system of linear equations with integer coefficients, a problem expected to be computationally hard in general. Progress on finding a more computationally efficient algorithm was recently demonstrated by utilizing graph matching methods [Ste14].

Concurrent with the development of fault-tolerant methods using topological codes was the recognition that these systems needed to be augmented with additional protocols in order to allow for universal quantum computing. The Eastin-Knill Theorem proved that no quantum code capable of correcting any single qubit error can have a universal and transversal set of quantum gates [EK09]. By relaxing the condition of transversality, state distillation protocols introduced by Bravyi and Kitaev [BK05] allowed for a universal extension of the quantum operations that could be performed in a fault-tolerant manner. Many new protocols for distilling these “magic states” have been proposed since [MEK12, BH12, CAB12, DCS13], and the magic state distillation field remains an active area of research with the goal of finding ever cheaper protocols for distilling states with a desired precision. As I will describe later, the requirements of distillation schemes fit especially nicely with the naturally fault-tolerant operations available to the toric code and the color codes.

These two families of topological codes feature prominently in this dissertation, and many of the discussions involving the toric code translate directly to the color codes. I will make it clear when this is not the case. In the next section I provide a short description of each chapter to make the structure of this dissertation more clear.

1.2 Outline

In Chapter 2, I fix the notation I will use and additionally provide a short introduction to the basic building blocks of quantum information theory. Chapter 3 introduces the stabilizer formalism and its use in studying quantum error correcting codes. I give an example with the Steane code, and I describe the problem of decoding—that is, inferring a correction from a measured syndrome. I introduce fault-tolerance in Chapter 4 and focus in particular on the ways that it is achieved naturally in topological codes. Here I also describe how topological codes with a non-universal set of naturally fault-tolerant gates can be made into universal quantum computers, describing the idea of magic states in the process.

The next three chapters are comprised of original research projects I’ve completed as a graduate student. First, Chapter 5 studies the relationship between two different ways of encoding quantum information into topological color code defects. The “triple defects” introduced by Fowler [Fow11b] inherit some of the properties of the underlying code, and this is largely due to topology. I demonstrate that it is impossible to change Fowler’s triple defects into standard defects without the use of topology-changing operations and teleportation. The necessity of these operations demonstrates the fundamental difference between these two encoding schemes and suggests that the nice properties of the underlying code will never be as natural for the standard defects.

Second, Chapter 6 presents original research performed jointly with Dave Bacon, Steve Flammia, Andrew Landahl and Alice Neels on a Hamiltonian model of quantum computation that lives at the intersection of several other models: topological quantum computation, holonomic quantum computation, and adiabatic quantum computation. We demonstrate an approach to performing quantum computation that adiabatically interpolates between static Hamiltonians while never causing the

Chapter 1. Introduction

Hamiltonian gap to shrink below a constant. We analyze the procedures which enable universality in detail, taking special care to avoid potentially harmful excitations. Taken together with Chapter 7, these two chapters offer one vision of a functional quantum computer, albeit one that is not as robust as models utilizing active error correction.

Lastly, in Chapter 7 I describe joint work with my advisor Andrew Landahl that proposes a new family of magic state distillation protocols. These protocols are based on a new family of quantum Reed-Muller codes—which we also introduce—that admit certain gates in a transversal fashion. In Appendix C we provide sufficient conditions on code parity check matrices that guarantee that these gates are transversal. Our protocols outperform the standard methods of approximating quantum gates using a universal set for a wide regime of target accuracies.

1.3 List of papers and other projects

Chapters 6 and 7 collect results from two projects I’ve completed. Chapter 6, a collaboration with Dave Bacon, Steve Flammia, Andrew Landahl, and Alice Neels, is being polished and prepared for publication, and Chapter 7 has been posted as a preprint to the the arXiv [\[LC13\]](#). It will also likely be submitted for publication to a journal in the future.

I’ve worked on other projects as a graduate student, including a collaboration with Jonas Anderson on extending the work in Ref. [\[LC13\]](#), as well as a project on performing estimation of error channels during rounds of quantum error correction that is the brainchild of Joshua Combes. This latter project has a nearly-completed manuscript and its submission is planned for the near future. I provide a list below of the mostly finished projects, their titles, author list, and, if known, the submission journal.

Chapter 1. Introduction

1. C. Cesare, A.J. Landahl, D. Bacon, S. T. Flammia, and A. Neels, “Adiabatic topological quantum computing,” arXiv:1406.2690 [CLB⁺14]. To be submitted to *Physical Review A*. Appears in this dissertation as Chapter 6.
2. J. Combes, C. Ferrie, C. Cesare, M. Tiersch, G. J. Milburn, H. J. Briegel, and C. M. Caves, “In-situ characterization of quantum devices with error correction,” arXiv:1405.5656 [CFC⁺14]. To be submitted to *Physical Review X*.
3. A. J. Landahl and C. Cesare, “Complex instruction set computing architecture for performing accurate quantum Z rotations with less magic”, arXiv:1302.3240 [LC13]. To be submitted to *Physical Review A*. Appears in this dissertation as Chapter 7.

Chapter 2

The Language of Quantum Information

In this chapter I set my notation and present the basic building blocks of quantum information. My perspective is largely from the field of quantum error correction, and so I favor using notation that is used by that community. This chapter is organized into three sections. First, I introduce the building blocks of classical and quantum information, bits and qubits. Next, I introduce the mathematical representations of quantum operators—which represent physical observables—and give quantum circuit examples of how they might be measured. Lastly, I discuss quantum channels, focusing especially on non-unitary random error channels that are the typical models for studying the robustness of quantum codes. The standard reference for everything here is Ref. [\[NC00\]](#).

2.1 Bits and qubits

The currency of classical information are strings $x = x_1x_2 \dots x_n$ such that

$$x_i \in \{0, 1\}. \quad (2.1)$$

That is, they are sequences of length n that live in the n -fold Cartesian product of the two-element set $\{0, 1\}$. Each x_i is called a *bit*, a shortening of *binary digit*. Bit strings are communicated in classical communication protocols and computed on using classical digital logic. These computations are often represented as circuits of classical logic gates that serve as the building blocks of any classical computation. A classical circuit can be specified as a function from an input bit string x to an output bit string y . The length of y could be anything in general, but for simplicity I assume that it has the same size as x . (Note that n -to-1 functions, which have outputs in the set $\{0, 1\}$, could be thought of as special cases of n -to- n functions that have a canonical value chosen for the remaining $n - 1$ output bits.) In general, classical circuits are not reversible. This is easily seen by examining the truth table for the AND gate, listed in Fig. 2.1. However, it is possible to formulate universal reversible models of classical computing, and in this case reversible circuits can be represented as permutation matrices on the space of all n -bit strings [Ben73]. A crucial limitation in classical computing (not including probabilistic computing) is the inability to operate on superpositions of input bit strings. In other words, the state space of classical bit strings lacks a vector space structure.

Quantum bits, called *qubits*, generalize the notion of classical bits by having the state space structure of a complex Hilbert space. Colloquially, this is often expressed as the capability of quantum states to be a classical 0 and a classical 1 *at the same time*, but this is a statement laced with popular imprecision. An analogy certainly holds, but the geometry of the two state spaces are vastly different. The general idea is that each classical bit string gets promoted to a basis vector in a complex

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

Figure 2.1: The truth table for the digital logic gate AND, demonstrating that it is not reversible. This is easily seen by noticing that given the output value 0, the input values for x and y are not uniquely determined.

Hilbert space, which is a complete vector space with a bounded norm. In the case of a single qubit, it is isomorphic to the complex vector space \mathbb{C}^2 . Single-qubit pure states $|\psi\rangle$ are represented by vectors in this Hilbert space, and are expressed using Dirac’s “bra-ket” notation as

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad (2.2)$$

where α and β are complex numbers obeying $|\alpha|^2 + |\beta|^2 = 1$. The basis states $|0\rangle$ and $|1\rangle$ comprise the computational basis, and they are eigenstates of a particular matrix to be introduced below. Although pure qubit states seem to be specified by three independent parameters—the real and imaginary parts of α and β gives four parameters, and the normalization constraint takes one away—in fact only the relative phase between the two basis states is physically meaningful. The global phase is physically unimportant, and states $|\phi\rangle$ and $|\psi\rangle$ can be identified with each other if

$$|\phi\rangle = e^{i\theta}|\psi\rangle. \quad (2.3)$$

The physical geometry of qubit pure states is then isomorphic to the points on the surface of the sphere \mathcal{S}^2 , called the Bloch sphere, and each of the points represents an infinite number of choices for the parameter θ .

The Bloch sphere together with its interior is called the Bloch ball, and this ball faithfully represents qubit states which are not pure. These mixed states—

Chapter 2. The Language of Quantum Information

represented by Hermitian density matrices $\rho \in \mathbb{C}^{2 \times 2}$ with $\text{tr}(\rho) = 1$ —represent classical statistical mixtures over sets of quantum pure states, but their decompositions into these sets is not unique. The purity of a general quantum state is defined as $\text{tr}(\rho^2)$, where the state ρ can in general be written (again, non-uniquely) as $\sum_i p_i |\psi_i\rangle\langle\psi_i|$ for some collection of pure states $\{|\psi_i\rangle\}$. The purity is bounded above by 1 and bounded below by $1/d$, where d is the dimension of the Hilbert space. For qubits, $d = 2$, and the so-called maximally mixed state can be written as $\rho = \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1|$. Mixed states model, for example, the uncertainty inherent in imperfect state preparations.

Pure states of n qubits live in the tensor product of n copies of \mathbb{C}^2 , and mixed states are Hermitian operators in the complex vector space $\mathbb{C}^{2^n \times 2^n}$. With only two qubits, there already exist quantum states that have no classical analog. For example, entangled pure states, which cannot be written in a separable fashion as $|\psi\rangle = |\phi_1\rangle \otimes |\phi_2\rangle$, exhibit correlations that are strictly stronger than any local hidden variable theory can admit [Bel64]. Entanglement is a ubiquitous primitive in quantum information theory, and the canonical entangled state of two qubits, the Bell state

$$|\psi\rangle = \frac{1}{\sqrt{2}} (|00\rangle + |11\rangle), \quad (2.4)$$

can be used in a variety of quantum information protocols: from teleportation [BBC⁺93] to superdense coding [BW92] to quantum key distribution [BB85].

Hamiltonians generate the time evolution of quantum states as given by the Schrödinger equation,

$$-i\hbar \frac{\partial}{\partial t} |\psi\rangle = H |\psi\rangle. \quad (2.5)$$

Hamiltonians on systems of n qubits are represented by Hermitian 2^n by 2^n matrices. Any generic Hermitian operator has a decomposition into a basis of Hermitian operators, and one choice of such a basis for qubits is the Pauli operators and their n -fold tensor products. The qubit Pauli operators have the following representation

in terms of a particular qubit basis:

$$\sigma_X = X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\sigma_Y = Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

$$\sigma_Z = Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The canonical basis that these operators are expressed in is called the computational basis, labeled by the states $|0\rangle$ and $|1\rangle$. These two states are the $+1$ and -1 eigenstates of Z , respectively. The use of X, Y, Z instead of $\sigma_X, \sigma_Y, \sigma_Z$ is a common quantum information notational convenience. Additionally, I will often write ZZ to mean $Z \otimes Z$ rather than the matrix product. The case should be clear from context.

Each of the single-qubit Pauli operators has an eigenbasis. Since the canonical basis we use is that of the operator Z , it is helpful to know what the other eigenvectors look like in this basis. The eigenvectors of X and Y in the Z basis are given by

$$|\pm\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle) \quad (2.6)$$

and

$$|\pm i\rangle = \frac{1}{\sqrt{2}}(|0\rangle \pm i|1\rangle). \quad (2.7)$$

The \pm in each state name labels its eigenvalue—that is, $X|\pm\rangle = \pm|\pm\rangle$ and $Y|\pm i\rangle = \pm|\pm i\rangle$.

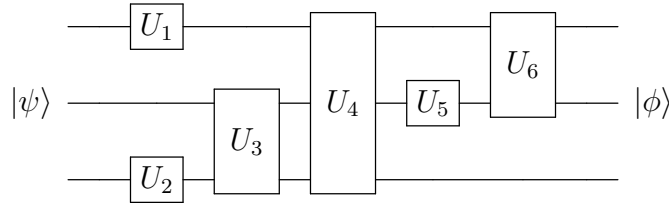


Figure 2.2: A simple example of a quantum circuit that applies a sequence of unitary operators, called gates, to three qubits, represented by the three wires. Written as the action of unitary operators on the input state $|\psi\rangle$, the output state $|\phi\rangle = U_6 U_5 U_4 U_3 U_2 U_1 |\psi\rangle$.

The Schrödinger equation can be formally solved, but a time-dependent Hamiltonian will in general not commute with itself at different times. The Dyson series, an ansatz for constructing the evolution operator induced by a Hamiltonian H , is evaluated in such cases. The resulting evolution operator is a unitary operator, represented by a 2^n by 2^n unitary matrix. Quantum computations can then be represented as sequences of these time evolution operators using the graphical tool of quantum circuit diagrams, shown in Fig. 2.2. These abstract away the underlying Hamiltonian dynamics and problems related to solving the Schrödinger equation, and they are a standard tool for representing coherent quantum dynamics.

There are several special unitary operators that appear frequently in error correction and quantum computation. These are listed in Fig. 2.3. Instead of referring to these as unitary operators, and in analogy with classical logic, I will henceforth call them gates. The gates S , H , and $CNOT$ generate a finite group, called the two-qubit Clifford group—endomorphisms on the set of Pauli operators under conjugation—that is intimately connected with quantum error correction and the classical simulatability of quantum circuits [Got99]. S is typically called the phase gate, since it adds a phase of i to the computational basis state $|1\rangle$. Geometrically it can also be thought of as a $\pi/2$ rotation about the Z axis of the Bloch sphere. The gate H , called the Hadamard gate, interchanges the eigenbases of X and Z . $CNOT$, a 2-

$$\begin{aligned}
 S &= \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \\
 H &= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \\
 T &= \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix} \\
 CNOT &= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}
 \end{aligned}$$

Figure 2.3: A typical set of gates discussed in relation to universal quantum computation.

qubit gate, is a reversible analog of the classical XOR gate, which performs an X on the second qubit register depending on the state of the first. It can entangle qubits given the right input state. The final gate, T , is referred to by many different names, but results in a rotation by an angle $\pi/4$ about the Z axis. This gate is important for magic state distillation protocols and is often used to extend the Clifford group to a set that can approximate any quantum gate to a desired precision.

So far I have introduced quantum states, quantum circuits, and quantum gates. However, these alone are not sufficient for universal quantum computation, since it is also necessary to make quantum measurements at the end of a computation to read out the results. I address these in the following section.

2.2 Quantum measurements

The notion of a generalized quantum measurement, given mathematically by a positive operator-valued measure over the space of quantum states, is important to tasks

Chapter 2. The Language of Quantum Information

like parameter estimation, state discrimination, and tomography [WM10, Par04]. However, for the results I present in this dissertation, these general measurements are not required to tell the story. The measurements that every quantum mechanics student learns about in introductory classes and texts will suffice.

A quantum measurement can be physically performed in a variety of ways, but here I'm not interested in the physical implementation. Mathematically, a measurement corresponds to an Hermitian operator, typically called an observable. The eigenvalues of this operator are the possible outcomes of the measurement, and the eigenvectors are the possible post-measurement states. The Pauli operators, mentioned earlier, play a particularly prominent role in quantum error correction, as the operators measured for stabilizer codes are all Pauli operators. Since many measurements of Pauli operators will be performed during quantum error correction, several properties of the Pauli operators, which form a group under matrix multiplication, are helpful to know.

The Pauli group of size n , \mathcal{P}_n , is generated by all operators P of the form

$$P = \omega \bigotimes_{i=1}^n O_i, \quad (2.8)$$

where O_i is one of X , Y , Z , or I defined above and $\omega \in \{\pm 1, \pm i\}$. Since each of these O_i obeys $O_i^2 = I$, each P also satisfies $P^2 = \omega^2 I = \pm I$. This means that each Pauli operator is, up to a sign, its own inverse, which means that each Pauli operator is invertible. Each P can also be diagonalized by some unitary operator, U , such that $P = U^\dagger D U$. Here D is a diagonal matrix, and its diagonal entries are the eigenvalues of P . Next, since

$$\begin{aligned} P^2 &= U^\dagger D U U^\dagger D U \\ \pm I &= U^\dagger D^2 U \\ \pm I &= D^2, \end{aligned}$$

D^2 is also $\pm I$. This implies that every Pauli operator either has eigenvalues ± 1 (when $D^2 = I$) or $\pm i$ (when $D^2 = -I$), since D is diagonal. For the rest of this

Chapter 2. The Language of Quantum Information

dissertation, I will only be concerned with Abelian subgroups of the full Pauli group (since these are precisely the subgroups that matter for stabilizer quantum error-correcting codes, introduced in Chapter 3). This restriction to Abelian subgroups, along with the added restriction that $-I$ is not an element of the subgroup, forces ω to be in the set $\{\pm 1\}$. Consequently, this restricts to operators P that only have eigenvalues ± 1 . In the rest of this section, where necessary, I will assume that this restriction is enforced.

For every P other than $\pm I^{\otimes n}$ and $\pm iI^{\otimes n}$, the eigenvalues appear an equal number of times on the diagonal of D . This is easy to see by noticing that the U which diagonalizes a given P has a very special structure. This U can be found simply by writing out the eigenstates of a Pauli operator P . As a simple example, if $P = Y \otimes X \otimes Z$, the eight eigenstates would be

$$|\pm i\rangle \otimes |\pm\rangle \otimes |0/1\rangle. \quad (2.9)$$

These are product states which can be transformed to computational basis states by local unitary rotations: the first qubit needs the Hadamard-like operator which exchanges Z and Y basis states, SHS^\dagger ; and the second qubit needs only the Hadamard H . In other words, the diagonalizing operator U is given by

$$U = SHS^\dagger \otimes H \otimes I, \quad (2.10)$$

which is just a product of local single-qubit unitary operations (and, notably, an element of the Clifford group). The operator P is transformed by U to $UPU^\dagger = Z \otimes Z \otimes Z$. This demonstrates that P also has an equal number of $+1$ and -1 eigenvalues since there is a choice of $|0\rangle$ or $|1\rangle$ for each local basis element (the operator $Z \otimes Z \otimes Z$ has manifestly balanced eigenvalues). This balanced nature of Pauli operator spectra aids in the geometric analysis of quantum error correcting code subspaces.

In particular, it is now simple to construct projectors onto the $+1$ or -1 eigenspaces

Chapter 2. The Language of Quantum Information

of a particular Pauli operator. For a Pauli operator $P \in \mathcal{P}_n$, they are given by

$$\Pi_{+1}^P = \frac{I_n + P}{2}, \quad (2.11)$$

and

$$\Pi_{-1}^P = \frac{I_n - P}{2}, \quad (2.12)$$

where I_n is the identity operator in \mathcal{P}_n . These will be especially useful in the next Chapter.

A further feature of the Pauli group is the fact that the elements all pairwise commute or anti-commute. For the single-qubit Pauli group, X , Y , and Z all mutually anti-commute, meaning, for example, that

$$\{X, Z\} = XZ + ZX = 0. \quad (2.13)$$

This useful feature of the single-qubit Pauli group extends to calculations involving larger Pauli group elements. For example, the two-qubit Pauli group elements $X \otimes X$ and $Z \otimes Z$ commute with other, which can be proven using the single-qubit Pauli anti-commutation relation above.

$$\begin{aligned} [X \otimes X, Z \otimes Z] &= XZ \otimes XZ - ZX \otimes ZX \\ &= XZ \otimes XZ - (-XZ + \{X, Z\}) \otimes (-XZ + \{X, Z\}) \\ &= 0. \end{aligned}$$

What decides whether two Pauli operators commute or anti-commute is simply whether an even or an odd number of single Pauli anti-commutators are used in calculations like the one above. Because each anti-commutator that is used introduces a single negative sign, if an even number appear, they cancel and the two operators commute because there is a remaining negative sign from the commutator. If an odd number of anti-commutators are used, an odd number of negative signs remain and will cause an anti-commutator to vanish. This second case can be seen

Chapter 2. The Language of Quantum Information

with a simple modification of the prior example.

$$\begin{aligned}
 \{X \otimes I, Z \otimes Z\} &= XZ \otimes IZ + ZX \otimes ZI \\
 &= XZ \otimes IZ + (-XZ + \{X, Z\}) \otimes ZI \\
 &= 0.
 \end{aligned}$$

Here only a single anti-commutator was used on the right hand side, causing the expression to vanish. This fact, that Pauli group elements either commute or anti-commute, will be used in the discussion of error correction in the next chapter.

Now that I've provided some of the basic properties of Pauli operators, I will show how to perform measurements of simple operators using quantum circuits. In particular, the way to perform a joint measurement of the operator $Z \otimes Z$. I will use a common technique that involves coupling the qubits of interest to an ancilla qubit and making a single-qubit measurement of the ancilla qubit. These circuits are the bread and butter of quantum error correction syndrome measurements.

The setting we will examine is pictured in Fig. 2.4. Here, two measurements are being performed on an input state $|\psi\rangle$: the first is a measurement of the operator $M_1 = Z \otimes Z \otimes I$, and the second is a measurement of the operator $M_2 = I \otimes Z \otimes Z$. Given outcomes of ± 1 for each measurement, the post-measurement state can be written as

$$|\psi'\rangle = \frac{\Pi_i^{M_2} \Pi_j^{M_1} |\psi\rangle}{\sqrt{\langle\psi| (\Pi_i^{M_2} \Pi_j^{M_1})^\dagger \Pi_i^{M_2} \Pi_j^{M_1} |\psi\rangle}}, \quad (2.14)$$

where i and j label the outcomes of the two measurements. In this particular case, $[M_1, M_2] = 0$ and Eq. 2.14 can be simplified, but this will not always be the case. In general, application of a composite projector Π , corresponding to the product of projectors for all measurements and outcomes, yields the post-measurement state.

Joint measurements, as represented by the operators M_1 and M_2 , are typically more challenging to perform. The generic scheme to combat this difficulty is to couple the qubits of interest to another qubit in a known state. Then, a single-qubit

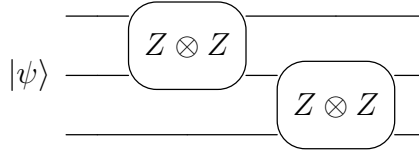


Figure 2.4: A circuit with an input state $|\psi\rangle$ and two joint measurements: first qubits 1 and 2 are measured, followed by qubits 2 and 3.

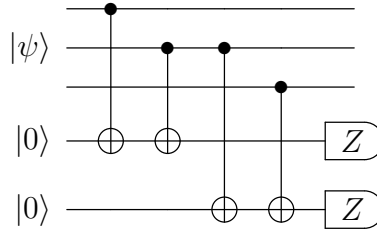


Figure 2.5: A circuit with an input state $|\psi\rangle$ and two ancilla-coupled measurements.

operator can be measured on the additional qubit, the result of which is the same as the result of the joint operators M . This is depicted in Fig. 2.5, which is the ancilla-coupled version of Fig. 2.4. It is easy to see that the two single-qubit measurements are equivalent to the joint measurements. To see this, it is sufficient to use the fact that $CNOT^2 = I$ several times along with the rules for how $CNOT$ conjugates Pauli operators. These rules are:

$$\begin{aligned} CNOT(X \otimes I)CNOT &= X \otimes X, \\ CNOT(I \otimes X)CNOT &= I \otimes X, \\ CNOT(I \otimes Z)CNOT &= Z \otimes Z, \text{ and} \\ CNOT(Z \otimes I)CNOT &= Z \otimes I. \end{aligned}$$

The rules for how $Y \otimes I$ and $I \otimes Y$ are conjugated are easy to derive from these four relations. It is now clear that the single-qubit operators measured in Fig. 2.5—namely $I \otimes I \otimes I \otimes Z \otimes I$ and $I \otimes I \otimes I \otimes I \otimes Z$ —become the operators $Z \otimes Z \otimes I \otimes Z \otimes I$ and $I \otimes Z \otimes Z \otimes I \otimes Z$ when they are conjugated in the proper way. The conjugation

Chapter 2. The Language of Quantum Information

rules are often used in the following form

$$\begin{aligned} CNOT(X \otimes I) &= (X \otimes X) CNOT, \\ CNOT(I \otimes X) &= (I \otimes X) CNOT, \\ CNOT(I \otimes Z) &= (Z \otimes Z) CNOT, \text{ and} \\ CNOT(Z \otimes I) &= (Z \otimes I) CNOT, \end{aligned}$$

to show what happens when a measurement is “pulled through” (*i.e.*, commuted with) a gate or vice versa. Additionally, since the ancilla qubits are prepared in known eigenstates of Z , measuring these three-qubit operators—already equivalent to measuring the single-qubit operators—are also equivalent to measuring the two-qubit operators. These manipulations anticipate the manipulations of the stabilizer group that I will introduce in the next Chapter, but for now I will abandon the discussion where it is.

Of course, the results of quantum measurements are probabilistic in nature. The likelihood p_i of getting measurement outcome i corresponding to the eigenstate $|i\rangle$ of an operator O is given by

$$p_i = \text{tr}(\Pi_i^O \rho) = |\langle i | \rho \rangle|^2. \quad (2.15)$$

This is known as the Born Rule and governs the way that quantum states are connected to experimental outcomes. Since for each i the Born Rule gives the probability of measurement result i , we can define the expectation value of the operator O in a given state as

$$\mathbb{E}[O] = \text{tr}(O\rho), \quad (2.16)$$

which gives the expected value of the average of many repeated measurements of the same state ρ .

The last piece of the puzzle, before describing stabilizer codes and quantum error correction, is a way of describing errors that can occur during quantum computations. As with the case of general quantum measurements, I will not need to describe the full power of general quantum operations. These are discussed in the following section.

2.3 Quantum channels

The notion of a quantum channel generalizes the notion of a classical channel, and provides a way of modeling and studying communication protocols. It is typically the case that a message—be it English text or a telephone signal—will be degraded while in transit to its destination. The field of classical error correction was developed to figure out how to robustly send messages that might be corrupted while traveling from sender to receiver. Quantum error correction solves essentially the same problem, although the “communication” in the context of quantum computation is usually just the output from one quantum gate being fed into the input of another quantum gate. In other words, the sender and receiver in the quantum error correction setting are at the same place but different times.

A standard classical error channel is the bit-flip channel. It acts by flipping each bit in a classical bit string $x = x_1x_2 \dots x_n$ independently with probability p . In other words, if the sender sends the bit string x , the receiver will receive the bit string y , which has values $y_i = x_i$ with probability $1 - p$ and $y_i = 1 \oplus x_i$ with probability p . Quantum channels resemble this classical channel, but of course act on a vector space of quantum states.

A typical quantum channel, the uniform depolarizing channel, acts on single-qubit quantum states as

$$\mathcal{D}(\rho) = (1 - p)\rho + \frac{p}{3}(X\rho X + Y\rho Y + Z\rho Z), \quad (2.17)$$

and corresponds to random X , Y , and Z errors that occur with probability $p/3$ each. This particular example demonstrates the trace-preserving property of general quantum channels since

$$\begin{aligned} \text{tr}[\mathcal{D}(\rho)] &= \text{tr}\left[(1 - p)\rho + \frac{p}{3}(X\rho X + Y\rho Y + Z\rho Z)\right] \\ &= (1 - p) + \frac{p}{3} + \frac{p}{3} + \frac{p}{3} \\ &= 1, \end{aligned}$$

Chapter 2. The Language of Quantum Information

where I used the cyclic property of the trace, $\text{tr}(X\rho X) = \text{tr}(\rho XX)$, and the fact that $\text{tr}(\rho) = 1$.

Another useful channel in the context of the codes discussed in the next two chapters is a composition of two channels. One is the analog of the classical bit-flip channel, given by

$$\mathcal{B}(\rho) = (1 - p)\rho + pX\rho X, \quad (2.18)$$

and the other is the phase-flip channel, given by

$$\mathcal{P}(\rho) = (1 - p)\rho + pZ\rho Z. \quad (2.19)$$

The composition of these two channels is like the depolarizing channel above, but the probabilities in front of X , Y , and Z errors are not the same. The composition is given by

$$\mathcal{P} \circ \mathcal{B}(\rho) = (1 - p)^2\rho + p(1 - p)X\rho X + p(1 - p)Z\rho Z + p^2(ZX)\rho(ZX), \quad (2.20)$$

and it is known as the bit-flip phase-flip channel. Note that in this channel X and Z errors occur with probability $O(p)$, while Y errors only occur with probability $O(p^2)$. I will use this fact in the next chapter when I discuss decoding.

For fault-tolerant analyses of quantum error correction, two-qubit channels are also used. The two-qubit depolarizing channel is defined as

$$\mathcal{T}(\rho) = (1 - p)\rho + \frac{p}{15} \sum O_i O_j \rho O_i O_j, \quad (2.21)$$

where the sum runs over the 15 non-trivial two-qubit Pauli operators.

It is usually assumed that the same channel \mathcal{E} acts on each of the qubits or pairs of qubits, and that the channels are uncorrelated. This situation is also called identical and independently distributed, or i.i.d., noise. This assumption aids in the decoding process by guaranteeing that the likelihood of m errors decreases as p^m .

Chapter 3

The Stabilizer Formalism and Quantum Error Correction

3.1 Overview

Physical errors on qubits—arising either from systematic errors in experimental control or random errors induced by coupling to an uncontrolled environment—can have deleterious effects on quantum information protocols. The theory of Quantum Error Correction (QEC) studies methods and protocols that enable quantum devices to function even in the presence of physical errors. The general strategy, depicted schematically in Fig. 3.1, is to embed the Hilbert space of one or several qubits into a larger Hilbert space, distributing the quantum information in such a way that physical errors can be detected, diagnosed, and corrected. There have been several approaches to this problem [Sho95, CRSS97, Ste96c], but one approach has dominated the field since its introduction.

This Chapter presents an overview of the dominant approach—the stabilizer formalism introduced by Gottesman in Ref. [Got97]. This formalism provides a unified

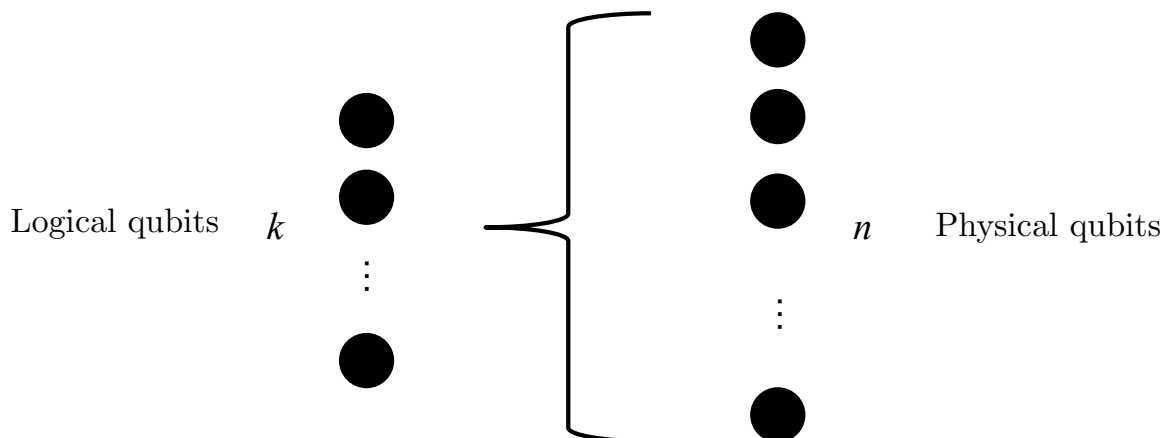


Figure 3.1: A schematic of the process of encoding k logical qubits in n physical qubits. I include this mainly to reinforce the distinction between logical and physical and to remind the reader of the language. The 2^k -dimensional Hilbert space of the logical qubits is embedded in a precise way into the full 2^n -dimensional space, to be introduced and discussed below.

method to study a large family of quantum codes, among which are the topological codes of Kitaev and Bombin discussed in Ch. 4. I begin by introducing some notation and language common to all quantum error correcting codes, proceed to a simple example called the Steane code, and then discuss the problem of decoding quantum codes—that is, classically processing the syndrome measurement results to infer a correction of the detected errors.

Quantum codes, much like classical codes, work by spreading the information of one qubit among many, achieving a redundancy that aids in diagnosing and fixing errors. Throughout this Chapter and the remainder of the dissertation, I will use the labels n , k , and d when discussing quantum codes to refer to the following properties: n is the number of physical qubits—the total number of qubits used by the code; k is the number of logical qubits—the number of qubits protected by the code; and d is the distance—a parameter relating to the code’s error correcting power that will be defined later in this Chapter.

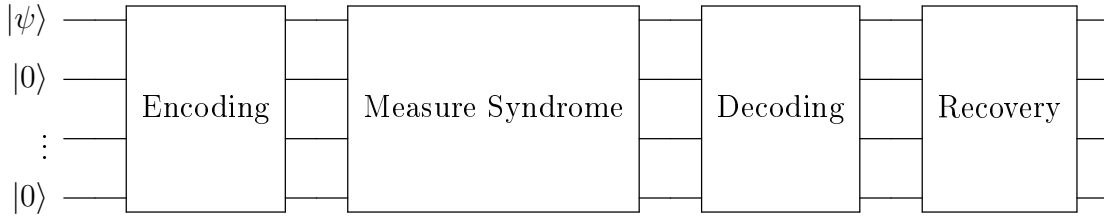


Figure 3.2: A schematic of the generic form assumed by quantum error correcting codes. A state $|\psi\rangle$ and ancilla qubits in the state $|0\rangle \cdots |0\rangle$ are input. Next, the state is encoded using a specific quantum encoding circuit. After this, errors on the encoded data are diagnosed by performing measurements. The results of the measurements are processed on a classical device and a quantum correction is calculated. Lastly, the quantum correction is applied and the process repeats itself (except for the encoding step). The terminology used here is defined and discussed in this Chapter.

Figure 3.2 provides a schematic of what quantum error correction looks like at the quantum circuit level.

3.2 A classical interlude: the repetition code

It is a useful pedagogical exercise—before delving into the details of quantum error correction—to understand a very simple classical code. The principles of quantum stabilizer codes and classical codes are very similar, and the classical repetition code is a very gentle introduction to the language and operation of coding theory.

As mentioned in the previous Chapter, encoding classical information is useful when it has to be transmitted over a noisy channel or protected for some amount of time. A typical way to model a noisy classical channel is by imagining that each bit traveling through the channel gets flipped with probability p —the bit-flip channel. The bit-flip channel is depicted schematically in Fig. 3.3. The strategy for classical codes is to send bit strings across the channel that can tolerate a few bit flips, still allowing the receiver to learn the intended message.

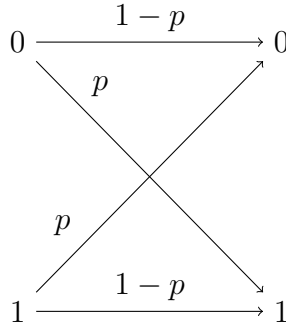


Figure 3.3: A schematic depicting the action of the bit flip channel on each bit x_i of a bit string x .

The repetition code is one of the simplest ways to solve this problem. To send the bit value 0, simply send a string of n 0s; to send the bit value 1, simply send a string of n 1s. We could identify the “logical” 0 and 1 values with these strings as

$$0_L = 0_1 0_2 \cdots 0_n \quad (3.1)$$

and

$$1_L = 1_1 1_2 \cdots 1_n. \quad (3.2)$$

Assuming the sender and the receiver agreed to the protocol beforehand, and assuming the probability of bit flips p is small enough—namely, $p < 0.5$ —the receiver can look at all the n bit chunks received and infer the intended bit value by doing a majority vote—counting the 1s and 0s in each n bit chunk and interpreting the intended bit value as whichever count is greater. Instead of performing this majority vote, another strategy is to look at the results of *parity checks*, which are often abbreviated to just *checks* when the context is clear. Here, the parity of all the neighboring bits are checked: an even parity for all neighboring bits corresponds to a valid codeword; any odd parity checks correspond to corrupted codewords. The parity checks can be written as a collection of conditions on the sums of neighboring bits modulo 2.

For example, in the three-bit repetition code, the bit value 0 is represented as

Chapter 3. The Stabilizer Formalism and Quantum Error Correction

Parity Check	Error on First Bit	Error on Second Bit	Error on Third Bit
$c_1 = x_1 \oplus x_2$	1	1	0
$c_2 = x_2 \oplus x_3$	0	1	1

Figure 3.4: A summary of the effects of single bit errors on the three-bit repetition parity checks.

000, and the bit value 1 is represented as 111. Imagine that during a communication round, the receiver reads the bit string 001 as the first three bits of a message. Majority vote decodes this message as a 0, since there are two 0s and only a single 1. The two parity checks— $c_1 = 0 \oplus 0 = 0$ and $c_2 = 0 \oplus 1 = 1$ —allow not only for the successful decoding of the message, but also directly suggest which of the bits to flip to return the corrupted codeword back to a true codeword. The effects of single-bit errors on these parity checks is summarized in Fig. 3.4. The three single-bit errors map to the three nontrivial parity check values. (The trivial case, when both parity check bits are 0, indicates either that no errors have occurred or three errors have occurred. The former occurs with probability $(1 - p)^3$ —corresponding to the probability of no errors on three independent bits—and the latter occurs with probability p^3 —corresponding to the probability of an error on each of three independent bits.) Two-bit errors lead to the same three non-trivial syndromes, but they only occur with probability p^2 . Using a decoding strategy that tries to determine the most likely error consistent with the syndrome will lead, in this case, to choosing the lowest-weight error. Thus, Fig. 3.4 functions as a decoding lookup table to restore corrupted bit strings back to proper codewords.

If the code is being used in the setting of a memory rather than as a communication tool, it might also be interesting to ask how bits encoding the value 0 can be changed into bits encoding the value 1. In the single-bit case this is just a single-bit flip, but in the case of the three-bit repetition code, we are looking for the equivalent

operator that will toggle between the encoded values 000 and 111. Performing an encoded bit flip in this case is not too much harder: all three bits must be flipped. The number of bits that need to be flipped to toggle between the two encoded states is also intimately related to the number of errors the code can correct. For example, suppose that a memory stores the value 0 as 000 and that the first two bits flip. The memory now reads 110, which, if we look at the table, has the parity check values $c_1 = 0$ and $c_2 = 1$. This corresponds, in our correction scheme, to flipping the third bit. If we proceed to correct the bits in memory using the lookup table, we will flip the third bit and end up with the memory reading 111. The correction has successfully returned the memory to a codeword but unfortunately to the wrong one. The reason for this is that the error that occurred—bit flips on the first two bits—in conjunction with the corrective action of flipping the third bit, is precisely the action that toggles between the codewords. This is an intuitive way to understand the connection between code distance, defined below, and the power of codes to correct errors up to a certain size.

3.3 Stabilizer generators and the codespace

I like to think of quantum stabilizer codes in the language of introductory quantum mechanics classes. The codes are defined by a set of Pauli operators called the stabilizer generators, and this set of operators is very nearly a complete set of commuting observables: they do mutually commute with each other, but they are not quite complete. (The stabilizer generators are generalizations of the classical parity checks, and indeed are also referred to as parity checks, or just checks, themselves.) I label the set of stabilizer generators \mathcal{S} , and the elements of this set are labeled S_i . Together, the S_i generate a group, called the stabilizer group, under multiplication. As already mentioned, they satisfy $[S_i, S_j] = 0$ for all i and j , and for an $[[n, k, d]]$

Chapter 3. The Stabilizer Formalism and Quantum Error Correction

quantum code there are $n - k$ stabilizer generators. Since there are fewer than n generators, this set of operators is not complete. However, the generators can be simultaneously diagonalized because of the commutativity, and this simultaneous eigenbasis is what provides a basis for the encoded qubits.

The codespace \mathcal{C} of a stabilizer code is the 2^k -dimensional Hilbert space defined as

$$|\psi\rangle \in \mathcal{C} \iff S_i|\psi\rangle = |\psi\rangle \quad \forall S_i \in \mathcal{S}. \quad (3.3)$$

In words, \mathcal{C} is the simultaneous $+1$ eigenspace of the set \mathcal{S} of stabilizer generators, and it is simple to demonstrate that \mathcal{C} has dimension 2^k .

Imagine enforcing the constraints $S_i|\psi\rangle = |\psi\rangle$ one by one, for each S_i . Initially, the Hilbert space of n qubits has dimension 2^n . Since the S_i are Pauli operators, they have an equal number of $+1$ and -1 eigenvalues. This means that each new constraint introduced by the S_i divides this Hilbert space in half, since, in the diagonal basis, half the states have eigenvalue $+1$ and half have eigenvalue -1 . Dividing this 2^n dimensional space in half $n - k$ times—once for each generator—leaves a Hilbert space of dimension $2^{n-(n-k)} = 2^k$.

Another important fact is that the operator $-I^{\otimes n}$ cannot be an element of \mathcal{S} , since it only has -1 eigenvalues¹. This requirement also disallows operators like $iZ \otimes Z$ from being in \mathcal{S} , since $(iZ \otimes Z)^2 = -I \otimes I$. All other operators S_i need only be mutually commuting Pauli operators. As discussed in Ch. 2, the Pauli group is allowed to additionally have multiplicative phases of $\{+1, -1, +i, -i\}$, but as mentioned there, the restriction to an Abelian subgroup that does not contain the element $-I^{\otimes n}$ limits the coefficients to values in $\{\pm 1\}$.

The strategy of quantum error correction is to perform measurements of the operators in the set \mathcal{S} . Typically, these measurements are performed in the ancilla-

¹Note that the condition $-I^{\otimes n}|\psi\rangle = |\psi\rangle$ forces $|\psi\rangle = 0$. Thus, adding $-I^{\otimes n}$ to \mathcal{S} guarantees that the codespace is trivial.

Chapter 3. The Stabilizer Formalism and Quantum Error Correction

coupled manner described in Ch. 2. For states in the codespace, the measurement should return a result of $+1$, but errors can move the state out of the codespace and lead to measurement results of -1 . Since errors are usually modeled as random Pauli channels, I will leverage the properties of the Pauli operators introduced in the previous Chapter.

Beginning with a state $|\psi\rangle \in \mathcal{C}$ —meaning that $S_i|\psi\rangle = |\psi\rangle$ —we can examine the action of an operator E on the codespace. Suppose first that $[E, S_i] = 0$. Then,

$$\begin{aligned} E|\psi\rangle &= ES_i|\psi\rangle \\ &= S_iE|\psi\rangle \\ &= S_i(E|\psi\rangle). \end{aligned}$$

In other words, the state $E|\psi\rangle$ is still a $+1$ eigenstate of the stabilizer generator S_i . There are several possibilities for how to classify the operator E in the case when it commutes with a particular S_i . First, it could be an element of the stabilizer group itself, in which case $E|\psi\rangle = |\psi\rangle$ and the operator E has a trivial action. Second, it could be an error that is not detected by measurement of the operator S_i —no single S_i can detect all errors—in which case it would presumably be detected by a subsequent measurement of a different stabilizer generator S_j . Lastly, it could be a logical operator for the code. These will be discussed in the next section.

The other case to examine is when E does not commute with S_i . Since E and S_i are both Pauli operators, this means that they anti-commute, and in that case

$$\begin{aligned} E|\psi\rangle &= ES_i|\psi\rangle \\ &= -S_iE|\psi\rangle \\ S_i(E|\psi\rangle) &= -E|\psi\rangle. \end{aligned}$$

In this case, we call E a detectable error, since if E is applied to $|\psi\rangle$ a measurement of S_i signals that $E|\psi\rangle$ is no longer in the codespace.

The preceding discussion indicates that any element in the Pauli group can be

decomposed into pieces which belong to the stabilizer group, the set of detectable errors, and the set of logical operators. The logical operators will be discussed next.

3.4 Logical operators

A quantum code allows qubits to be protected to a certain degree, but so far I have only discussed quantum codes in the context of a quantum memory—simply storing a quantum state in a redundant and robust fashion. The ability to perform computations—particularly computations on encoded quantum information—is also desirable. For this reason I now introduce the notion of logical operators for quantum stabilizer codes.

As mentioned in the last section, logical operators are Pauli group elements which commute with all of the stabilizer generators. The additional restriction is that these operators themselves cannot be elements of the stabilizer group—that is, the logical operators are not products of stabilizer generators. This allows for the manipulation of encoded information without introducing any detectable errors into the system. Indeed, logical operators can also be called undetectable errors, since their application does not lead to any -1 results while performing the measurements of the stabilizer generators. The idea here is that if the environment couples strongly to a logical operator, it can quickly corrupt the encoded information without disturbing the operation of the code.

To discuss the action of logical operators—an action which follows naturally from the unencoded single-qubit case—I first have to introduce the codewords. The codewords are the states of the codespace \mathcal{C} that are identified with the encoded $|0\rangle$ and $|1\rangle$ states—they represent the encoded computational basis. I will label these states

Chapter 3. The Stabilizer Formalism and Quantum Error Correction

with a subscript L , and a general encoded qubit state can be written as

$$|\psi_L\rangle = \alpha|0_L\rangle + \beta|1_L\rangle. \quad (3.4)$$

Here I've specialized to an $[[n, 1, d]]$ quantum code—a quantum code encoding only a single-qubit—but what I introduce next easily generalizes to the k -qubit case.

Each stabilizer code has an encoding circuit, consisting entirely of Clifford group unitary gates, that takes as input a state to encode and a set of ancilla qubits. I will give an explicit example of such an encoding circuit in the next section when I describe the Steane code, but for now the precise form of the circuit is not important. The essential function of this circuit is to map an input $|0\rangle$ to $|0_L\rangle$ and an input $|1\rangle$ to $|1_L\rangle$. Since the unitary gates are linear operators, their action extends to arbitrary linear combinations of $|0\rangle$ and $|1\rangle$ as input, and hence will encode arbitrary quantum states. The state $|0_L\rangle$ can be canonically represented as the state $|0\rangle^{\otimes n}$ projected onto the codespace—that is, up to normalization,

$$|0_L\rangle = \frac{1 + S_1}{2} \frac{1 + S_2}{2} \dots \frac{1 + S_{n-k}}{2} |0\rangle^{\otimes n}. \quad (3.5)$$

Written using the projector notation of Ch. 2, the state $|0_L\rangle$ is given as

$$|0_L\rangle = \prod_i^{n-k} \Pi_{+1}^i |0\rangle^{\otimes n}. \quad (3.6)$$

The logical X operator, denoted by X_L , can be used to define the state $|1_L\rangle$ as

$$|1_L\rangle = X_L |0_L\rangle = X_L \prod_i^{n-k} \Pi_{+1}^i |0\rangle^{\otimes n}. \quad (3.7)$$

The actions of Z_L and Y_L are defined in the obvious way.

The distance of a quantum error correcting code, labeled d , is the minimum weight of all the logical operators. The weight of an operator is the number of qubits on which the operator acts nontrivially. This can be a tricky parameter to calculate, since the logical operators are not defined uniquely. For example, due to

the definition of states in the codespace, there are many equivalent X_L operators. This equivalence class can be defined as

$$\{O \mid O \in \mathcal{P}_n, O = SX_L \ \forall \ S \in \mathcal{S}\}, \quad (3.8)$$

that is, by first selecting a canonical X_L , a set of equivalent X_L operators O can be generated simply by multiplying the canonical operator by any element of the stabilizer group. This is due to the fact that $X_LS|\psi_L\rangle = X_L|\psi_L\rangle$ by the definition of the codespace. This freedom to deform the logical operators plays a prominent role in the topological codes of Ch. 4 and is discussed mathematically in Appendix A, and I will provide an example of this in the following section on the Steane code.

As a final note, in Ch. 5 I will use the notion of a “gauge qubit.” Borrowing language from quantum field theory, where a gauge corresponds to a kind of coordinate freedom, gauge qubits in quantum error correcting codes represent parts of the codespace that do not store any important quantum information. As in quantum field theory, a gauge can be fixed, and this is the setting in which I discuss gauge qubits. Essentially, fixing the gauge corresponds to promoting one of the logical operators for an encoded qubit to a stabilizer generator. The gauge qubit is thus in a definite eigenstate, typically of X_L or Z_L .

3.5 Example: the Steane code

The Steane code, introduced in Ref. [Ste96c], is a $[[7, 1, 3]]$ quantum code. It can be constructed from a classical Hamming code, or a punctured classical Reed-Muller code, but I will simply give its encoding circuit and define its stabilizer generators, codewords, and logical operators. The Steane code is particularly simple to understand because of the high degree of symmetry in its stabilizer generators and, consequently, in its codewords.

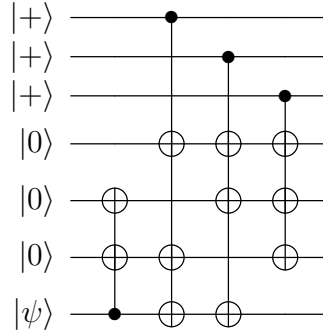


Figure 3.5: An encoding circuit for the $[[7, 1, 3]]$ quantum Steane code.

$$\begin{aligned}
 S_1 &= X \otimes X \otimes X \otimes X \otimes I \otimes I \otimes I \\
 S_2 &= X \otimes X \otimes I \otimes I \otimes X \otimes X \otimes I \\
 S_3 &= X \otimes I \otimes X \otimes I \otimes X \otimes I \otimes X \\
 S_4 &= Z \otimes Z \otimes Z \otimes Z \otimes I \otimes I \otimes I \\
 S_5 &= Z \otimes Z \otimes I \otimes I \otimes Z \otimes Z \otimes I \\
 S_6 &= Z \otimes I \otimes Z \otimes I \otimes Z \otimes I \otimes Z
 \end{aligned}$$

Figure 3.6: The set of stabilizer generators for the Steane code.

I’ll begin by giving an encoding circuit for the Steane code, pictured in Fig. 3.5. This circuit takes an input state $|\psi\rangle$ and six ancilla states, and it outputs an encoded version of $|\psi\rangle$. This circuit corresponds to the “Encoding” box in Fig. 3.2. The next step in Fig. 3.2, “Measure syndrome,” corresponds to measurements of the stabilizer generators introduced next.

The stabilizer generators of the Steane code are given in Fig. 3.6. The first thing to note is that each of the generators is comprised either entirely of X -type Pauli group elements or Z -type Pauli group elements. This is not always the case with stabilizer codes, but it can be a useful feature when decoding in the presence of certain noise models, like the bit-flip composed with phase-flip channel introduced in Sec. 2.3. Codes with this property are called CSS codes, the name referring to the authors who first studied them in depth [CS96, Ste96b]. Additionally, in the case of the Steane code, there is a great deal of symmetry between the first three stabilizer

Chapter 3. The Stabilizer Formalism and Quantum Error Correction

generators and the last three: they are the same under the exchange of Z and X . CSS codes with this additional symmetry are called “strong” CSS codes. Kitaev’s toric codes, mentioned in Chapter 1, are a family of non-strong CSS codes, while Bombin’s color codes, also mentioned in Chapter 4, are a family of strong CSS codes.

The logical states can be calculated as above, yielding for $|0_L\rangle$

$$\begin{aligned} |0_L\rangle = \frac{1}{\sqrt{8}} & \left(|000000\rangle + |1111000\rangle + |1100110\rangle + |1010101\rangle \right. \\ & \left. + |0011110\rangle + |0101101\rangle + |0110011\rangle + |1001011\rangle \right). \end{aligned}$$

Similarly, the state $|1_L\rangle$ can be written as

$$\begin{aligned} |1_L\rangle = \frac{1}{\sqrt{8}} & \left(|1111111\rangle + |0000111\rangle + |0011001\rangle + |0101010\rangle \right. \\ & \left. + |1100001\rangle + |1010010\rangle + |1001100\rangle + |0110100\rangle \right). \end{aligned}$$

From these expressions, it is obvious that one choice for X_L is simply $X_L = X^{\otimes 7}$. This operator simply exchanges all the components of $|0_L\rangle$ and $|1_L\rangle$. However, due to the freedom of the equivalence class of logical operators, we could also choose the operator $X'_L = S_1 X_L = I \otimes I \otimes I \otimes I \otimes X \otimes X \otimes X$.

The operator Z_L can be chosen in a similar way, but by using the intuition gained by knowing the form of X_L , another method is to simply make an educated guess and check if it works. Again, the operator must commute with all the stabilizer generators and not be an element of the stabilizer group. Additionally, since it is a logical Pauli operator, it also must anti-commute with the operator X_L . Consider the operator $Z_L = Z^{\otimes 7}$. This commutes with all of the stabilizer generators, but cannot be constructed by multiplying any of them together. Additionally, this operator anti-commutes with X_L —and indeed also with X'_L as it must. The only thing left to check is that it has the proper action on the codewords. Indeed it does, and this can be seen by noticing that every component of $|0_L\rangle$ has an even number of 1s and every component of $|1_L\rangle$ has an odd number of 1s. Thus the state $|0_L\rangle$ will accumulate no phase from this operator, while the state $|1_L\rangle$ will pick up a -1 .

Chapter 3. The Stabilizer Formalism and Quantum Error Correction

As in the case of the operator X_L , Z_L can also be multiplied by a stabilizer group element to construct an equivalent logical operator. Due to the symmetry of the generators and the logical operators, Z'_L can be chosen to be $Z'_L = I \otimes I \otimes I \otimes I \otimes Z \otimes Z \otimes Z$. It is the case that X'_L and Z'_L are both the smallest weight logical operators that can be constructed. Thus, the Steane code has a distance $d = 3$.

As mentioned above, the distance of the code is related to its error correcting power. Power in this case refers to the number of errors that the code can correct. A code of distance d can correct all Pauli errors up to weight t where $d = 2t + 1$. This implies that the Steane code can correct any single error on a qubit. The three-bit repetition code introduced earlier also had $d = 3$, and as we saw it could correct an error on any single bit. In the next section I will list the lookup table for correcting errors in the Steane code.

I'd like to briefly discuss a strategy that allows the Steane code to correct more errors. This is clearly not possible with just the 7-qubit Steane code; the number of qubits will need to be increased. The question is whether there is an appropriate strategy for increasing the number of qubits in a particular way that maintains the structure of the code. It turns out that one way of doing this is to *concatenate* the Steane code with itself.

Concatenation simply replaces each of the 7 qubits in the Steane code with 7 more qubits, each also encoded in the Steane code. At the physical level, then, there are now 49 qubits, blocked up into 7 groups of 7. Error correction is first performed on each of the 7 blocks and then on the superblock of 7 qubits as shown in Fig. 3.7. The concatenation allows the code to tolerate more errors at the physical level, as now it takes at least 5 errors to cause a failure since the concatenated code has distance 9. A comprehensive study of different concatenated encoding schemes for use in fault-tolerant quantum computing was performed in Ref. [CDT09].

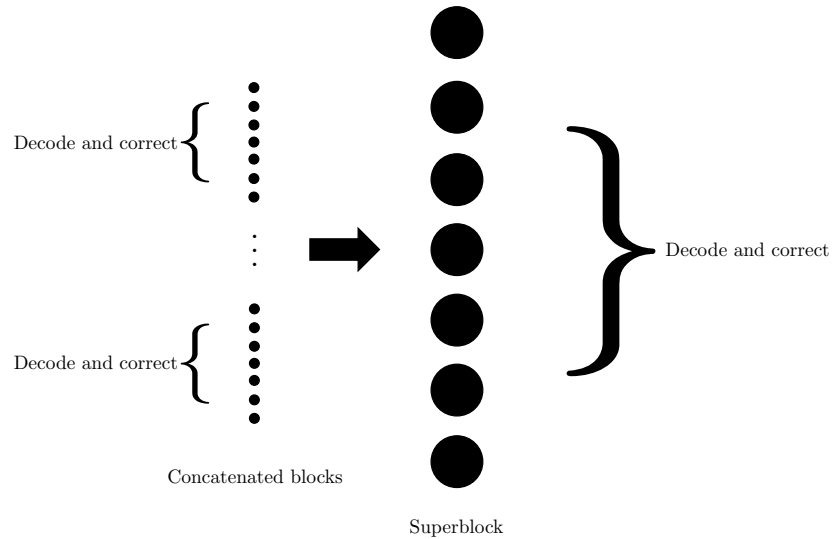


Figure 3.7: A schematic depicting error correction in a concatenated code. First, error correction is performed on the level of concatenated blocks. Then, error correction is performed at the next highest level—the superblock for the physical concatenated blocks—and so forth.

3.6 Decoding quantum codes

Decoding, in the context of quantum error correction, refers to the process of classically analyzing the results of syndrome measurements and inferring a correction that is compatible with the given syndrome. For instance, one strategy for decoding is to choose the most probable error consistent with the syndrome, but this is not the only such strategy. Assumptions about the noise model are an integral part of this inference, and they can inform the choice of decoding algorithm.

For the case of the Steane code, decoding is largely an exercise in constructing a lookup table that shows the syndrome for all the correctable errors. Then, given a syndrome, the most probable error can be identified and corrected. Decoding the topological codes, introduced in Ch. 4, is a more complicated procedure due to the fact that different correctable errors can lead to the same syndrome.

Chapter 3. The Stabilizer Formalism and Quantum Error Correction

Stabilizer Generator	Z_1	Z_2	Z_3	Z_4	Z_5	Z_6	Z_7	$I^{\otimes 7}$
$(-1)^{\langle S_1 \rangle}$	1	1	1	1	0	0	0	0
$(-1)^{\langle S_2 \rangle}$	1	1	0	0	1	1	0	0
$(-1)^{\langle S_3 \rangle}$	1	0	1	0	1	0	1	0

Figure 3.8: The S_X syndromes for the Steane code corresponding to all the single-qubit Z errors.

I will now provide the lookup table for the Steane code and discuss some of the reasons that the code is unable to generically correct two-qubit errors. Recall the stabilizer generators for the Steane code, provided in Fig. 3.6. As mentioned before, there is a high degree of symmetry among the six generators. Depending on the noise model, this may mean that the syndromes from only the X -type generators need be discussed, since the same will be true of the Z -type generators as well. However, I will discuss all the generators here for completeness and to aid a discussion on some subtleties of decoding due to the error model that is assumed.

By X_i , Y_i , and Z_i below, I will mean an operator acting on qubit i with a Pauli operator and identity on the other six qubits. Additionally, I will label by S_X the collection of stabilizer generators $\{S_1, S_2, S_3\}$ and by S_Z the collection $\{S_4, S_5, S_6\}$. I will first examine the syndromes generated by all single-qubit Z errors, and I will convert the measurement outcomes ± 1 into binary strings to aid the discussion.

The binary strings corresponding the syndromes for all single-qubit Z errors are listed in Fig. 3.8. Each stabilizer generator in S_X corresponds to a row. A 0 along a row means the corresponding stabilizer generator commutes with the error labeling the column. Likewise, a 1 means that the stabilizer generator anti-commutes with the error, leading to a -1 measurement result when the generator is measured. I've included the last column—the identity operator leading to no errors—to complete the collection of binary expansions of numbers from 0 to 7. If read from right to left, the columns correspond exactly to these expansions, and the importance of this fact

Chapter 3. The Stabilizer Formalism and Quantum Error Correction

Stabilizer Generator	X_1	X_2	X_3	X_4	X_5	X_6	X_7	$I^{\otimes 7}$
$(-1)^{\langle S_4 \rangle}$	1	1	1	1	0	0	0	0
$(-1)^{\langle S_5 \rangle}$	1	1	0	0	1	1	0	0
$(-1)^{\langle S_6 \rangle}$	1	0	1	0	1	0	1	0

Figure 3.9: The S_Z syndromes for the Steane code corresponding to all the single-qubit X errors.

is that bit strings of length three can provide unique labels for 8 different items. In the case of the S_X generators in the Steane code, these 8 items are the 7 different single-qubit Z errors and the trivial case of no errors.

Of course, there are two-qubit (and above) Z errors that will also lead to these same 8 syndromes, since the S_X generators are the only measurements which will detect Z errors of any size. The decoding strategy is to select the most likely error that leads to the observed syndrome, and for most error models that are studied this means selecting the single-qubit errors. The multi-qubit errors occur with probability at most $O(p^2)$ in these cases, and will usually lead to logical errors causing the code to fail. As an example, consider the error $E = Z_6 Z_7$. This will lead to the syndrome 011, the same syndrome caused by the error Z_5 . The decoding procedure would decide to apply the operator Z_5 to fix the error, but the total effect of applying that correction is now that the operator $Z_5 Z_6 Z_7$ has been applied. As mentioned above, this is a representative of the class of Z_L operators for the Steane code. Unbeknownst to the Steane code user, a logical Z has been applied.

I will also provide the collection of syndromes measured by the stabilizer generators S_Z in the presence of single-qubit X errors. Due to the total symmetry between S_X and S_Z , the form of these syndromes are not surprising. They are given in Fig. 3.9. A entirely analogous discussion regarding multi-qubit X errors also applies here, so I will refrain from restating myself.

Chapter 3. The Stabilizer Formalism and Quantum Error Correction

Stabilizer Generator	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	$I^{\otimes 7}$
$(-1)^{\langle S_1 \rangle}$	1	1	1	1	0	0	0	0
$(-1)^{\langle S_2 \rangle}$	1	1	0	0	1	1	0	0
$(-1)^{\langle S_3 \rangle}$	1	0	1	0	1	0	1	0
$(-1)^{\langle S_4 \rangle}$	1	1	1	1	0	0	0	0
$(-1)^{\langle S_5 \rangle}$	1	1	0	0	1	1	0	0
$(-1)^{\langle S_6 \rangle}$	1	0	1	0	1	0	1	0

Figure 3.10: The S_X and S_Z syndromes for the Steane code corresponding to all the single-qubit Y errors.

The syndromes caused single-qubit Y errors—which, for the purposes of the syndromes should just be thought of as single-qubit XZ errors—are then the same three-bit strings for both S_X and S_Z . For completeness, I include these in Fig. 3.10. Again, to stress the point, the S_X and S_Z syndromes are perfectly correlated. The only syndromes we observe in the presence of single-qubit errors are of the form

$$x_1x_2x_3000 \tag{3.9}$$

for single-qubit Z errors;

$$000x_1x_2x_3 \tag{3.10}$$

for single-qubit X errors; or

$$x_1x_2x_3x_1x_2x_3 \tag{3.11}$$

for single-qubit Y errors. There are 22 unique six-bit strings that have one of these three forms. The following question arises: what happened to the other 42 six-bit strings?

It is natural to think that the Steane code might be able to correct *some* two-qubit errors. As demonstrated above, the code cannot correct two-qubit X or Z errors, as the inferred corrections in these cases can lead to undesired logical errors. How about errors consisting of an X and a Z on different qubits? Consider the error X_1Z_2 . We can easily look up the syndrome in the tables above—it will have

Chapter 3. The Stabilizer Formalism and Quantum Error Correction

an S_X syndrome of 110 and an S_Z syndrome of 111. This syndrome does not fall into any of the formats described above, and so it clearly does not correspond to a single-qubit error. However, there are two other two-qubit errors that yield the same syndrome. Both Y_1Z_7 and Y_2X_7 lead to the same syndrome. If X , Y , and Z errors are equally likely, as for the uniform depolarizing channel, there is not a unique error that is most likely to have caused the syndrome, and the wrong choice quickly leads to undetectable logical errors. For channels in which Y errors are less likely than X and Z errors, such as the bit-flip phase-flip channel, it is possible to come up with a successful decoding strategy and correct two-qubit errors of the form X_iZ_j for $i \neq j$. For the standard case of a depolarizing channel, these additional syndromes cannot be decoded successfully.

The fact that the two-qubit error syndromes are distinct from any of the single-qubit error syndromes means that the Steane code is able to *detect* two qubit errors in general. Error detection can be useful in some circumstances, such as in decoding the distance 9 concatenated Steane code, but I will not be discussing it any further.

Chapter 4

Topological Fault-Tolerance and Avenues to Universality

4.1 Overview

The previous chapter introduced the stabilizer family of quantum error correcting codes, which provides a simple framework for defining and understanding a large class of quantum codes. In this chapter, I will focus on stabilizer quantum codes that leverage the non-local correlations present in certain many-body quantum systems. These codes utilize the topology of such systems to encode quantum information, and they are broadly referred to as topological codes.

The mathematical field of topology, like geometry, is concerned with the shapes of objects. The difference is that in topology, the fine-grained structure of a manifold is not of interest. It helps to think of topology as a coarse-graining of geometry in which only global properties are retained. In particular, metric distance is not a topological property, and, indeed, a metric need not even be defined for topological spaces. However, it is not necessary to understand topology with a mathematician's

Chapter 4. Topological Fault-Tolerance and Avenues to Universality

rigor in order to appreciate its application to topological codes. It is really a subfield of algebraic topology called homology that is the operative mathematics. I describe some of the basics of homology theory applied to surfaces in Appendix A.

This chapter will be concerned almost entirely with two-dimensional arrangements of qubits. More specifically, it is concerned with systems of qubits, and their interactions, that can be embedded into surfaces. Such constraints are motivated by experimental technologies that are limited to nearest-neighbor interactions between qubits in two dimensions, although technologies that allow for more complicated interactions are not incompatible with topological codes. In addition to this geometric locality, topological codes also satisfy another kind of locality: the cardinality of the interactions—that is, the number of qubits involved in each interaction—is a constant that does not change with system size. This notion of locality need not indicate that qubits are also close together, and so I mention it here as an additional restriction.

I begin this discussion of topological codes in the next section by introducing Kitaev’s toric code. In addition to being the primary pedagogical tool in the field, it also serves as the best example of how to perform universal fault-tolerant quantum computing in encoded form. The toric code is most easily understood with lots of figures, and many will be provided. I will introduce the topological nature of the logical operators in terms of loops and boundaries on a graph, and describe the set of easily applied logical operators.

Next, I will describe a family of topological codes discovered by Bombin called the color codes. These codes have a constant improvement in rate— k/n —over the toric code, and offer a richer set of natural logical operations. The logical operators for the color codes have some interesting symmetry properties, and I will discuss these properties along with work that has been done on the fundamental differences between the toric code and the color codes.

After introducing these two well-studied families of topological codes, I introduce the notion of fault-tolerance and show the ways in which the topological codes are natural candidates for fault-tolerant architectures. In particular, the topological codes allow for both transversal implementation of logical gates as well as topologically protected braiding operations. Here I briefly mention progress on self-correcting quantum codes and discuss the relation of self-correction to the topological nature of logical operators.

Fault-tolerant computation is only possible if errors can be dealt with faster than they appear, and in the next section I introduce the notion of a code threshold to study how robust the topological codes are to physical errors of a certain likelihood p . The threshold of a code is a number p_{th} such that for physical errors of probability $p < p_{th}$, arbitrarily long computations can be performed with only a modest increase in the required resources.

Lastly, a universal set of encoded quantum gates does not exist for these topological codes, so a method of approximating arbitrary gates to a desired precision is necessary. This typically involves compiling approximations over a finite universal set, and then distilling states capable of teleporting gates that aren't already available. The penultimate section of this chapter describes these procedures and provides the final preparation and motivation for the remainder of the dissertation.

4.2 The toric code and planar surface codes

The toric code is a stabilizer quantum code that was first introduced by Kitaev in Ref. [Kit03]. It can be defined on any graph but is most often defined using a square lattice like that in Fig. 4.1. Graphs are simple mathematical objects defined by two sets: a set $V = \{v_1, v_2, \dots, v_n\}$ of vertices and a set E of edges. The set E itself has elements in the Cartesian product $V \times V$ such that $e_{ij} \in E$ can be written as

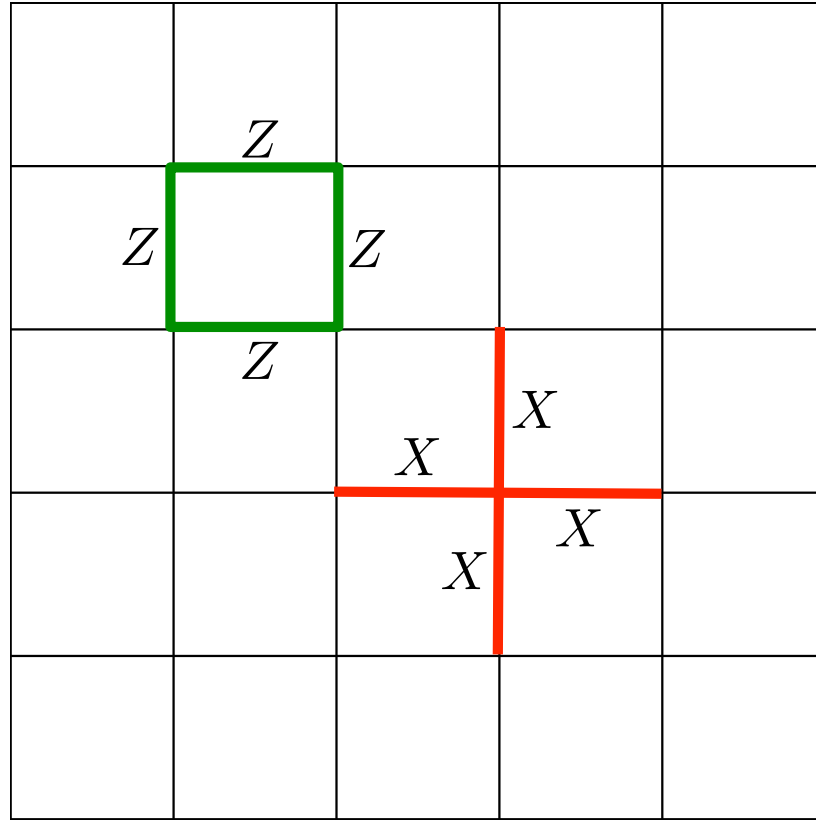


Figure 4.1: The surface code defined on a square lattice. The edges correspond to qubits, the faces to Z -type stabilizer generators, and vertices to X -type stabilizer generators. Depicted is an $[[n, k, d]]$ quantum code with parameters $n = 60$ and $k = 0$. The trivial nature of this code is due to the fact that all the boundaries are “smooth,” as defined in the text.

$e_{ij} = (v_i, v_j)$. Graphs can be represented visually by drawing a point for each vertex and a line connecting vertices v_i and v_j only if there is an edge e_{ij} . There are a variety of descriptors that can be used in front of “graph” that signify restrictions on these sets, but I’ll just mention that in what follows the graphs will be simple, meaning the edges have no orientation or weighting, and there are no self-loops nor multiple edges between vertices.

The definition of a toric code is actually slightly more general than what follows,

but it is unnecessary to define them in full generality. Indeed, most of the work on performing quantum computations with the toric code makes precisely the same restrictions. The original introduction of the code had the graph embedded on a torus—or even surfaces of higher genus—but I will only consider embeddings in the plane with boundaries. These “planar surface codes,” referred to henceforth as “surface codes” for brevity, can arise, for instance, from a code defined on a sphere with a puncture. The boundaries introduced by this puncture are important, but before discussing the boundaries I will define the general construction from a graph.

For the surface code, each edge of the graph corresponds to a physical qubit. The stabilizer generators are also defined via graph elements. For each vertex of the graph v , there is an X -type stabilizer generator defined by

$$S_v = \bigotimes_{e|v \in e} X_e. \quad (4.1)$$

Likewise, for each face of the graph f , there is a Z -type stabilizer generator defined by

$$S_f = \bigotimes_{e|e \in f} Z_e. \quad (4.2)$$

In words, the X -type generators are defined as a product of Pauli X operators on each edge incident on a given vertex and I on all other qubits. The Z -type generators are a product of Pauli Z operators on each edge that is adjacent to a given face and I on all other qubits. For the surface code instance pictured in Fig. 4.1, most of these operators have weight four, although some of the X -type checks on the boundary have weight two or three.

Each face and vertex define one of these operators, and the set of all such operators comprise the set of stabilizer generators. By construction, the X -type and Z -type operators are guaranteed to commute since they will either not overlap at all or will overlap on two edges. The two species of stabilizer generator are shown in Fig. 4.1.

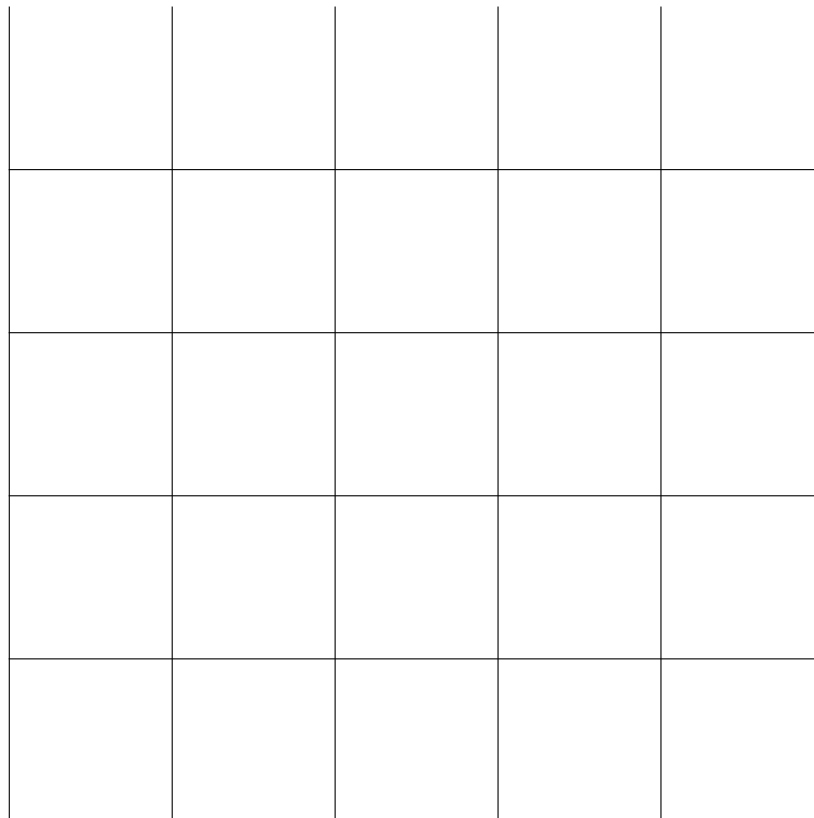


Figure 4.2: An instance of the surface code encoding a single qubit. Note the difference in the boundary compared to Fig. 4.1.

The boundaries of a code instance are important, as they completely determine how many logical qubits are encoded (see Appendix A). In Fig. 4.1 there is only a single type of boundary—it is called a *smooth* boundary and has weight-four Z -type generators and weight-three (and weight-two at the corners) X -type generators. This is in contrast to the boundaries pictured in Fig. 4.2, which are both smooth and rough. A rough boundary is so named because the four-body X checks stick out like spokes at the boundary. By a counting argument, I will demonstrate how many qubits are encoded in each instance. Fig. 4.1—the case of only smooth boundaries—has 60 edges, 25 faces, and 36 vertices. Naïvely, it seems there are 60 qubits and 61 stabilizer generators. However, not all of the generators defined by the vertices are

Code Boundaries	n	$n - k$	k
All smooth boundaries	60	60	0
Alternating boundaries	50	49	1

Figure 4.3: A summary of the parameters for the surface code boundaries discussed in the text. The number of physical qubits is n , the number of stabilizer generators is $n - k$, and the number of logical qubits is k .

independent. In fact, multiplying together any 35 of the vertex checks will result in an operator equal to the remaining vertex check. The final count is then 60 qubits and 60 independent stabilizer generators: the codespace does not encode a qubit, but merely fixes a single qubit state. In contrast, the code pictured in Fig. 4.2 has more interesting boundaries. The graph has 50 edges, 24 vertices, 15 faces, and 10 partial faces that only have three sides but that still define generators. In this case, there are 50 qubits and only 49 stabilizer generators, so the code contains one logical qubit. This counting is summarized in Fig. 4.3

This logical qubit has corresponding logical operators. These correspond to string-like operators that connect boundaries of the same type. One choice for a complete set of logical operator representatives in this case is shown in Fig. 4.4. These operators commute with all of the stabilizer generators, since X_L is incident twice on each face it touches and Z_L is incident on two adjacent edges for each vertex it traverses. They also intersect on a single edge, ensuring the proper commutation relations for logical Pauli operators.

The structure of these operators demonstrates the precise fashion in which the quantum information is associated with non-local degrees of freedom of the many-body system. They must stretch from boundary to boundary, but the actual path that they take is not important. Recall from the previous chapter that logical oper-

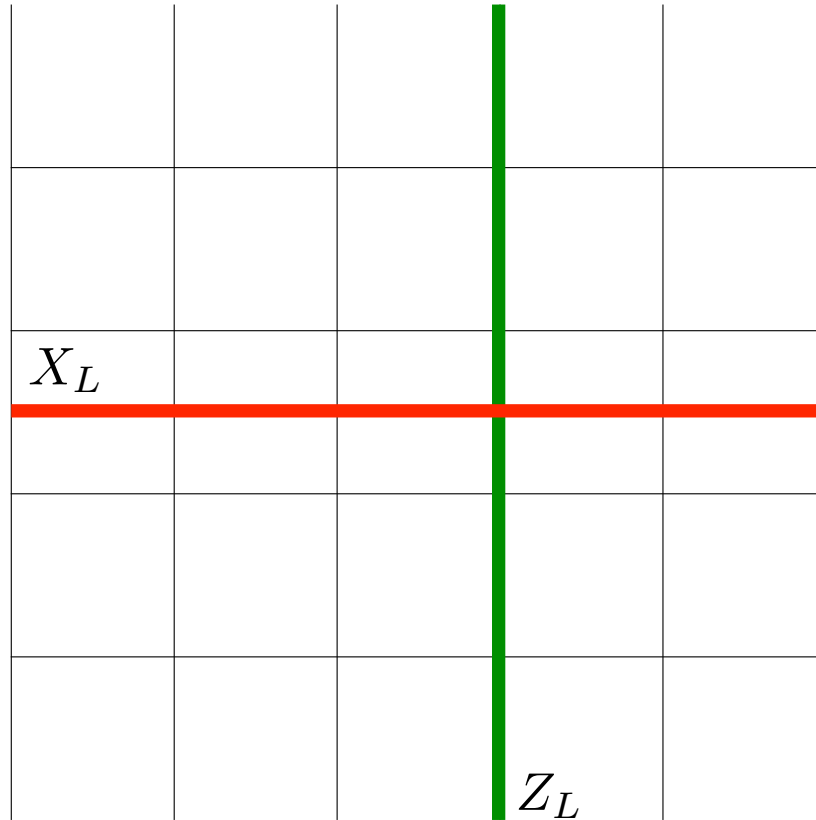


Figure 4.4: The two logical operators for a surface code encoding one qubit. Logical Z is a string of Z operators connecting the two rough boundaries. Logical X is a ladder of X operators connecting the two smooth boundaries. It can also be thought of as a string of operators on the dual lattice (pictured here).

ators O_L live in an equivalence class, with an equivalence relation

$$O_L \sim S_i O_L \quad \forall S_i \in \mathcal{S}. \quad (4.3)$$

The string-like logical operators of the surface code can be deformed by the multiplication of any element of the stabilizer group. A representative deformation is shown in Fig. 4.5. The insensitivity of the logical operator to the local twists and turns are indicative of its topological nature. It only matters that the operator start on one boundary and end on a boundary of the same type, without being a product solely of stabilizer generators.

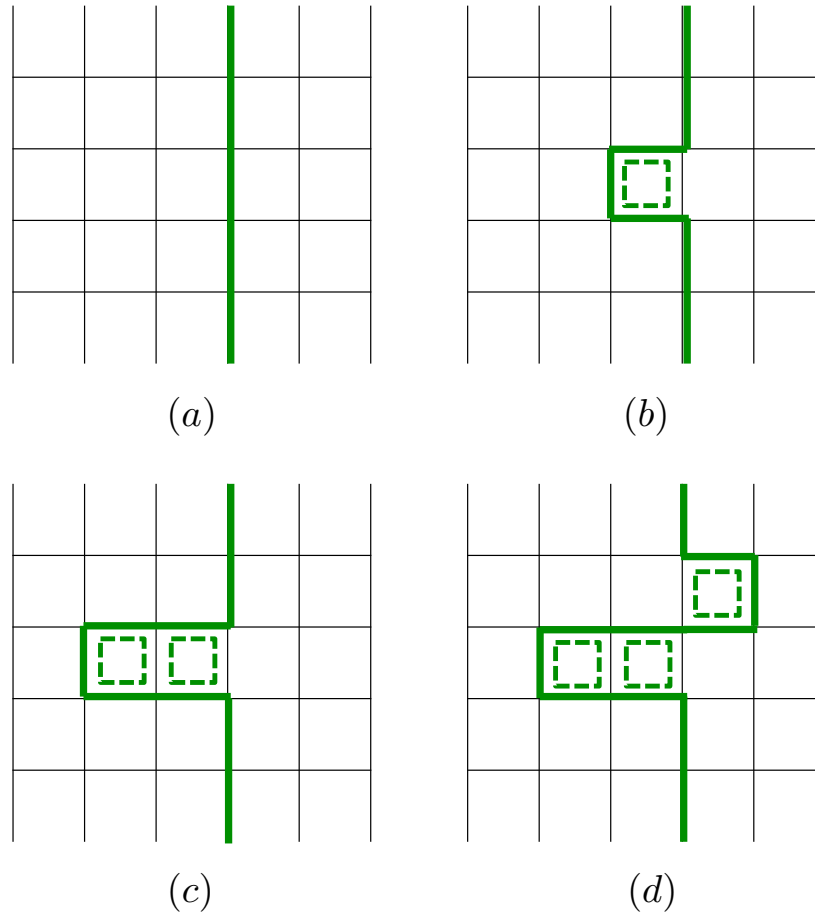


Figure 4.5: A sequence of deformations of a surface code logical operator. The dashed boxes represent Z type stabilizer generators defined on the faces. (a) shows a canonical choice for Z_L . In (b), this canonical choice has been multiplied by a stabilizer generator, deforming the string. (c) shows a further deformation along the same direction. Finally, (d) deforms Z_L one face in the other direction.

The distance of the code pictured in Fig. 4.4 is $d = 5$, the minimum of the weights of X_L and Z_L . This implies that the code can correct all errors up to weight $t = 2$. However, the distance is not a good indicator of the error correcting power of topological codes as there are many errors with weight above two that can also be corrected. Before discussing the most pernicious errors of weight 3 and greater, as well as higher-weight errors which can be corrected, I will describe the nature of

toric code errors.

Random errors are typically represented by a Pauli channel acting independently on each qubit. Errors that are not adjacent to the boundaries (or other errors) will anti-commute with an even number of stabilizer generators. Since the generators are localized to the vertices and faces of the graph, it is often convenient to think of the violated checks as quasiparticles that inherit this localization. These quasiparticles can correspond, for example, to the endpoints of *error chains*. By error chains, or simply *chains*, I mean connected paths of operators on the graph or its dual. Z -type error chains will be paths of Z operators on the graph while X -type error chains will be paths of X operators on the dual graph¹. The location of violated stabilizer generators is identified with a quasiparticle that can move around the lattice in particular ways. In the case of a single-qubit X error, two neighboring Z -type stabilizer generators will detect the error, as depicted in Fig. 4.6. Chains of errors that do not reach the boundary will always result in only two violated checks—namely, the checks at the endpoints of the chain. This situation, for a three-qubit error chain, is depicted in Fig. 4.7. This interpretation of violated checks—as particles at the ends of error chains—becomes a useful language when examining certain decoders for the toric and surface codes.

One type of decoding, known as most likely error decoding, corresponds to finding the most likely error that could have caused a given pattern of violated checks. Due to the existence of multiple correctable errors that lead to the same syndrome (the defining property of *degenerate* codes), there exist more refined ways of determining a correction. For the topological codes, another approach is to try and determine the most likely class of errors which produced a given syndrome. The class of an

¹The dual graph is constructed by sending each face in the original graph to a vertex in the dual graph, and each vertex in the original graph to a face in the dual graph. Adjacent faces in the original graph become adjacent vertices in the dual graph, and this defines the dual graph edges.

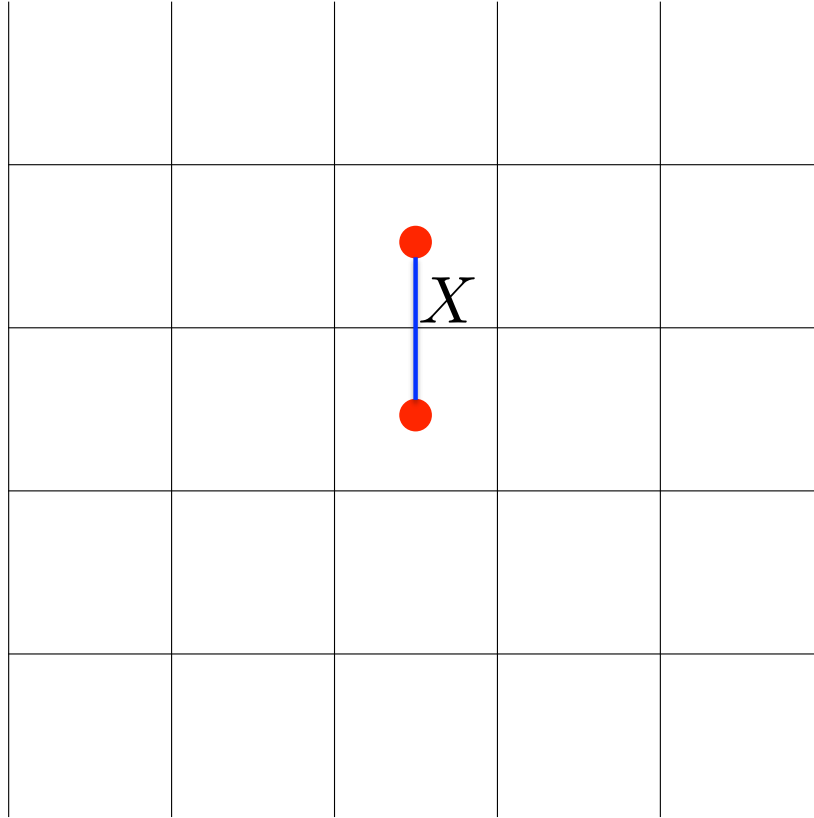


Figure 4.6: A single X error is detected by the two Z checks on adjacent faces. The violated Z check operators can be thought of as localized particles that live on the faces of the lattice.

error, which is defined via homology theory in Appendix A, boils down to whether or not the error corresponds to a logical operator or to an element of the stabilizer group. Once the class is decided, a representative member of the class is applied and, due to the topological properties of the code, the code is returned to the codespace. An example of the degeneracy of quantum codes and of the importance of the error class is shown in Fig. 4.8. Imagine that the weight-2 X -type error E occurs, causing the two indicated face checks to be violated. For an error model with identical probabilities of X errors on each qubit, the likelihoods of the error chain E and the error chain E' are the same. The correction that the decoder decides to apply might

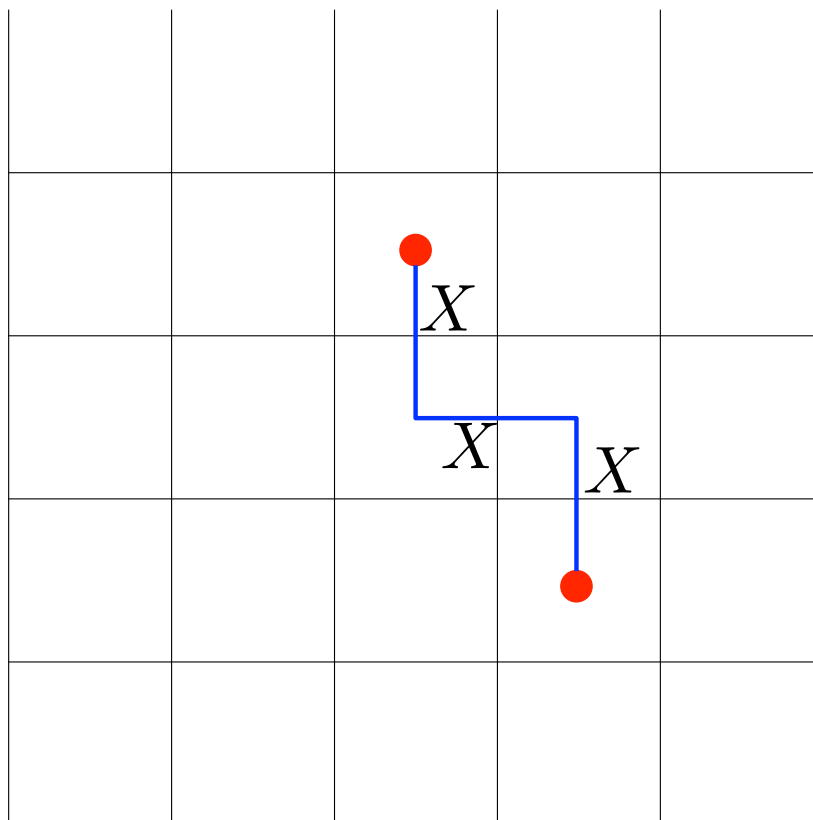


Figure 4.7: For a chain of 3 X errors, still only two Z checks are violated. The particle description of violated checks still holds, and it is clear that the pair of particles is created at the endpoints of error chains.

be either E or E' , but in the end it doesn't matter which correction is selected: correcting with E will trivially lead to an identity; correcting with E' won't lead to an identity, but the product of E and E' is an element of the stabilizer generators. Acting on a logical state with a stabilizer generator is also a trivial action on the logical space, and so in both cases the state is returned to the logical space and preserved in the process.

A complementary way to study the surface code, or any stabilizer code, is with a Hamiltonian constructed from the set of stabilizer generators. Indeed, the ground

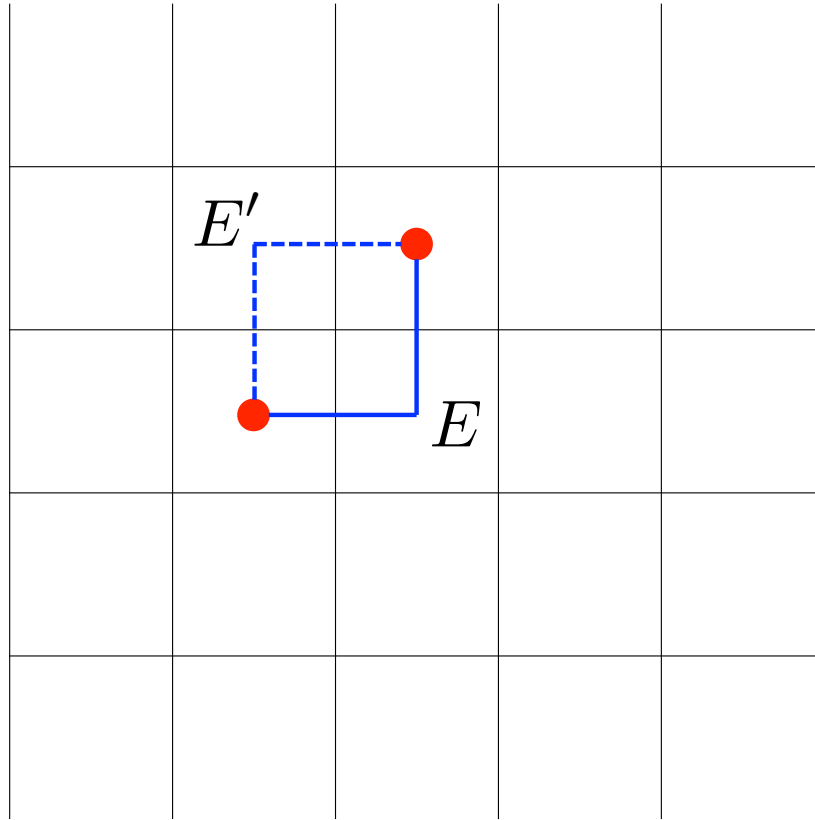


Figure 4.8: Two errors— E and E' —which lead to the same syndrome. Regardless of which of these the decoder decides is the “actual” error, the effect on the logical state is the same. If the real error is E , then the effect of the error and the correction E is $E^2|\psi_L\rangle = |\psi_L\rangle$. If the real error is E , then the effect of the error and the correction E' is $E'E|\psi_L\rangle = S_f|\psi_L\rangle = |\psi_L\rangle$.

space of the Hamiltonian

$$H = - \sum_v S_v - \sum_f S_f \quad (4.4)$$

is precisely the same as the codespace—codewords will have the lowest possible energy, namely $-(|V|+|E|)$. Errors act to raise the energy and corrections return the system to the ground space. The Hamiltonian description of stabilizer codes is useful when studying self-correction or in the context of the adiabatic code deformation techniques described in Ch. 6.

The dream of a self-correcting quantum memory, discussed in more detail in Sec. 4.4.3, is that the system dynamics, governed by a Hamiltonian like Eq. 4.4 and a coupling to a thermal bath, will keep the system near the ground space. The string-like nature of the surface code logical operators has consequences for the self-correcting capabilities of the code. Because only the endpoints of error chains violate check operators, only the endpoints of error chains raise the energy of the system. The amount the energy is raised is independent of the length of the chain; each error chain contributes only a constant energy. Left to its own devices, a surface code that is not actively corrected will quickly develop error chains that spread and lead to uncorrectable logical errors [AFH09]. If the mechanism of random error creation is via coupling to a bath at some temperature T , then it is possible to maintain the surface code provided T is small relative to the energy gap in the system. This is the setting imagined in Chapter 6, where the gap is also constant as a function of system size. It puts an upper limit on the number of qubits that can be used, since each added qubit also adds a spot for the bath to couple but does not increase the system gap. Once an excitation is created, it can wander freely and corrupt the data. Computations in such settings also have a short lifetime with respect to the system size, which is precisely the wrong hallmark to have for self-correction.

As mentioned above, the violated check operators have an interpretation in terms of particles. For the surface code, there are two species of particles. One species corresponds to violated X checks and lives on the vertices; the other corresponds to violated Z checks and lives on the faces. Consider the Z checks. On any given face, a check will be either satisfied or violated. These correspond to the absence and presence of a particle respectively. Given the checks and the fact that the quantum systems are qubits, the quantum double [dB94] construction will produce the spectrum of particles that can arise as well as the way particles interact via braiding. For the surface code, this construction is almost as trivial as possible: there is one type of particle that can live on the faces and one that can live on

vertices. Both are their own antiparticle and have no additional internal structure. The presence of a particle can simply be labeled “1,” with the label “0” corresponding to the trivial case of no particle. Thus, the particles can be described as elements of the group Z_2 , with the group operation of addition corresponding to the fusion of two particles. The two different species of particle—which go under the various names of face and vertex, flux and charge, or magnetic and electric—interact when one is braided around the other. However, the mutual statistics generated by this interaction is abelian and introduces only a global -1 to the quantum state of the system. By deforming the surface code graph, introduced as a “twist” in Ref. [Bom10], these mutual statistics can be modified to allow for nonabelian interactions. The new interactions correspond to a theory of “Ising anyons,” but the details are beyond the scope of this dissertation. For more on the topological origin of anyon mutual statistics, see Ref. [LM77], and for more on a study of the allowed graph deformations, see Ref. [KK12].

4.3 The color codes

The second family of topological codes I will introduce are Bombin’s topological color codes [BMD06]. As studied in Ref. [And11], under only mild assumptions about stabilizer codes, the color codes and the surface codes are the only topological codes that exist in two dimensions. Other work [BDGP12] has shown that the color codes are locally equivalent to two copies of the surface code, but they are still interesting in their own right in terms of studying resource requirements for universal quantum computation.

As for the surface code, the color codes are defined on graphs but with the additional restriction of face 3-colorability. Graph colorability is typically a statement about vertices, but with the dual transformation it becomes a statement about faces.

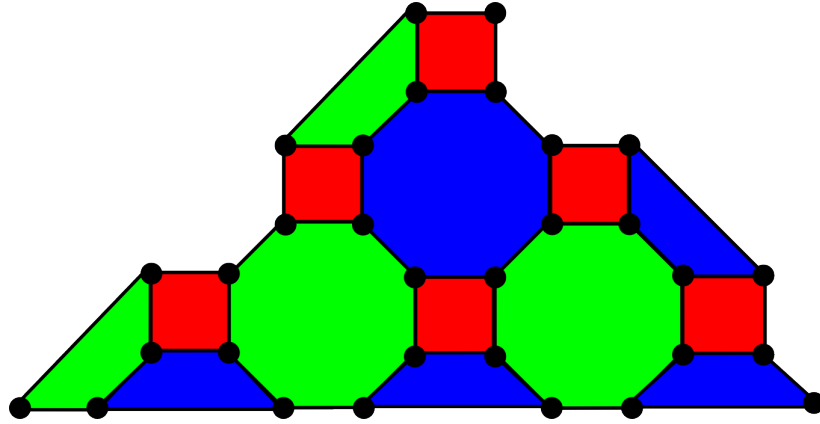


Figure 4.9: A distance-7 instance of the 4.8.8 topological color code. The label 4.8.8 corresponds to number of edges of the three faces adjacent to vertices away from the boundary: one square and two octagons. Color codes are defined by graphs that are face 3-colorable with the qubits on vertices and the stabilizer generators on faces. Each face actually corresponds to two generators: one X type and one Z type.

In terms of the faces, 3-colorability means that each face can be colored with one of three colors—in this case red, green, or blue—in such a way that adjacent faces do not have the same color. This limits the types of graphs that furnish color codes, and, in particular, requires that such graphs have vertices with degree three away from boundaries.

The other difference with respect to the definition of the surface codes provided in Sec. 4.2 is that the roles of the graph structures are slightly different. For color codes, qubits live on the vertices of the graph as opposed to the edges. (I should note that this is not really a fundamental difference. A suitable graph transformation—the medial transformation—will map a surface code to a model with the qubits on the vertices. The original surface code definition, with the qubits on edges, was inspired by the methods and language of lattice gauge theory.) Each face is then associated with *two* stabilizer generators: one is a product of X operators on the qubits adjacent to the face, and the other is a product of Z operators on those same adjacent qubits. A distance-7 example is shown in Fig. 4.9. The graph in Fig. 4.9

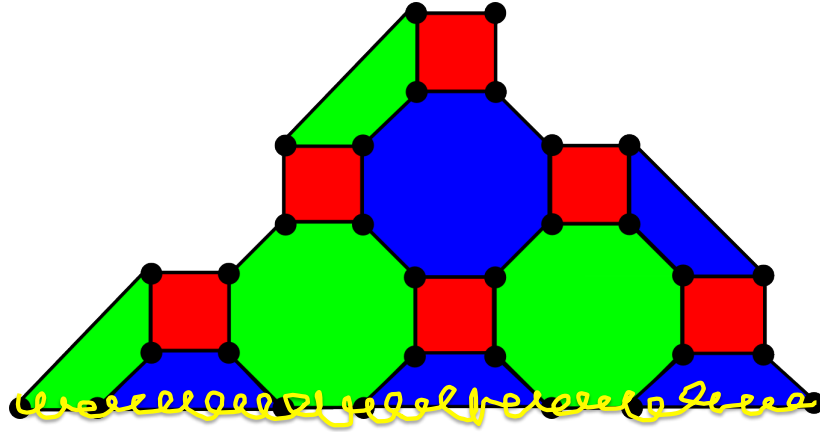


Figure 4.10: X_L and Z_L for the distance 7 4.8.8 color code. Both the logical operators have the same shape and can even act on the same qubits. X_L is a tensor product of X operators on the bottom qubits and Z_L is a tensor product of Z operators on those same qubits.

is only one of several satisfying the rules about face colorability that can tile the plane. It is called the 4.8.8 color code due to the number of edges of each face surrounding vertices away from the boundary. Although there are other tilings that can be used for color codes, the 4.8.8 code admits the entire Clifford group in a transversal manner. This allows for the easy application of a large portion of the gates required to perform a quantum computation. Other color codes, such as those defined on the 6.6.6 (hexagonal) lattice, do not admit the full Clifford group in a transversal manner. They are less useful as quantum computational substrates for this reason.

Logical operators in the color code have a similar string-like structure as those in the toric code, as pictured in Fig. 4.10. However, these logical operators can also be represented graphically in a different way: as what might be called a string-net operator (though this form should not be confused with the string nets of Levin and Wen [LW05]). This form of the logical operators is presented in Fig. 4.11. Presenting the logical operators in this way demonstrates several interesting features of the color

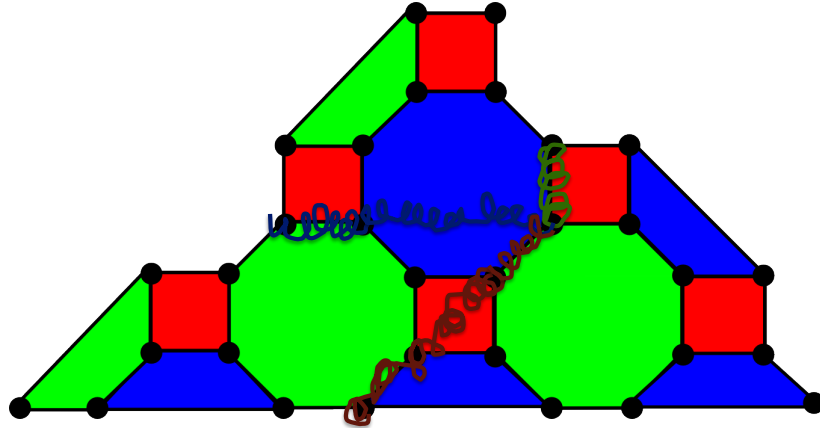


Figure 4.11: The string-net representation of color code logical operators. This pattern of operators is equivalent—up to multiplication by stabilizer generators—to the operators in Fig. 4.10. This figure also introduces the notion of colored strings, which is a convenient tool for discussing the homology of the color code graphs. Blue strings connect to blue boundaries and travel through blue faces. The same holds for the other two colors. The color of the boundary is determined by the color that is absent along that boundary. Strings of a single color can split into two strings of the remaining two colors, as is seen in this example.

code. First, string operators can be assigned a color, as can the boundaries. Blue strings must end on blue boundaries and travel through blue faces, and so forth. The color of the boundary is determined by the color that is missing from said boundary. The blue string terminates on a blue boundary in Fig. 4.11 since on that side of the “triangle” there are only red and green faces. There is also a color symmetry in that blue strings can split into red and green strings.

In the Hamiltonian description of the color codes, the excitations corresponding to the violations of one type of check can be labeled by elements of the quantum double of $Z_2 \times Z_2$, which is just a Cartesian product of the group that leads to the surface code excitations. The explicit local mapping between the two models can be found in Ref. [BDCP12].

4.4 Topological fault-tolerance

4.4.1 The laws of fault-tolerance

Even with quantum information encoded in the non-local degrees of freedom in topologically ordered quantum many-body systems, there is still no guarantee that useful computations can be performed robustly. Naïve designs for encoded gates can allow errors to spread in harmful ways, and the goal of (non-topological) fault-tolerant protocols is to prevent the catastrophic spread of errors through a quantum circuit.

To aid in the design of fault-tolerant circuits, Preskill [[Pre98c](#)] provided five laws of fault-tolerant computation, loosely paraphrased in the following list:

1. Don't reuse ancilla qubits too many times.
2. Syndrome extraction should copy the errors, not the data.
3. Verify preparations of known states.
4. Repeat syndrome extraction.
5. Choose the right code.

These laws—really guidelines—allow the designer of quantum circuits to minimize the ways that errors can spread. The first law and second laws are closely related; both have to do with the fact that ancilla qubits are the shuttles that carry entropy away from the data qubits. Thus, the ancilla qubits should be refreshed often enough so that errors in the ancillae do not corrupt many rounds of syndrome extraction. The syndrome extraction should also avoid coupling any logical operators to the ancilla qubits—unless at infrequent times during the computation when non-destructive

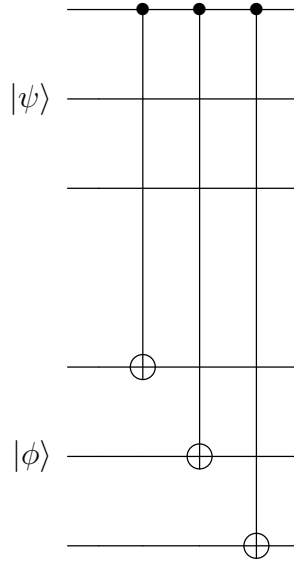


Figure 4.12: A naïve implementation of the $CNOT$ gate for the three qubit repetition code. This circuit correctly implements the $CNOT$, but it does so in a way that is not fault-tolerant. Note that a single X error on the top qubit can propagate to three more errors on the bottom code block.

logical measurements are required—so as to limit the environment’s access to the encoded information. The third and fourth laws also help to mitigate the ways that entropy can creep into the computation: the third law by ensuring that prepared states are pure through verification and the fourth law by ensuring that the result of syndrome measurements is as errant as the encoded data. The fifth law favors the use of a code that has as many *easy* fault-tolerant operations as possible—it is an expression of the natural fault-tolerance of transversal quantum gates.

An example is the simplest way to understand these laws in action. Consider performing an encoded $CNOT$ in the three qubit repetition code. A naïve method is to use the circuit depicted in Fig. 4.12. The goal of fault-tolerant protocols is to prevent errors from spreading too badly, and the circuit in Fig. 4.12 allows a single X error to spread in a maximally bad way. An X error on the top wire of the upper block of qubits can propagate to three errors on the lower block of qubits. In

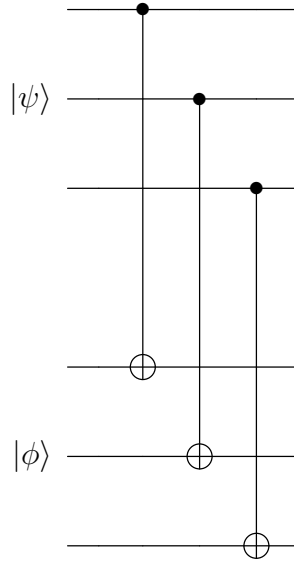


Figure 4.13: The fault-tolerant application of $CNOT$ between two qubits encoded in the repetition code. In this case, a single X error can spread to at most one other qubit in the other block. Since each block can correct for a single X error, single errors remain correctable as they traverse the circuit.

the case of the repetition code, the three errors in the lower block correspond to a logical fault. If X errors have probability p , then a probability p event can cause immediate corruption of one of the qubits. So, while the encoding was chosen so that only X errors that happen with probability p^2 are harmful, when a logical gate is implemented in a naïve way, this protection can be lost.

The fault-tolerant way of implementing a logical $CNOT$ between two qubits encoded in the repetition code is shown in Fig. 4.13. Here, a single X error can propagate only as far as one more qubit, and that qubit will be in the other code block. Since each block can protect against a single X error, this circuit design retains the error correcting power of the original code.

4.4.2 Topological code defects

Up to now, I’ve been discussing $[[n, 1, d]]$ quantum codes. To do anything besides just store a quantum state and apply single-qubit rotations, I need to introduce a way to have more than one qubit. One option has already been hinted at by the example of a fault-tolerant *CNOT* in Fig. 4.13: simply store each qubit in a separate surface code (or color code, etc.). This “pancake” architecture—used in Ref. [DKLP02]—requires a three-dimensional arrangement of qubit lattices. Much better is to use *defects* (also occasionally referred to as *punctures* in this setting) to increase the codespace dimension of a single toric code lattice, and this is by far the most popular way to introduce more qubits to topological codes.

One way to understand the effect of creating a defect is by observing what happens to the set of stabilizer generators. A defect in the toric code is just a “missing” face or vertex check. Nothing is done to the lattice of qubits; one merely stops measuring some of the check operators and possibly modifies neighboring check operators accordingly. This means that the set of stabilizer generators no longer has $n - k$ elements, but rather $n - k - 1$. Due to the decrease in the size of the set of generators, the Hilbert space dimension of the codespace is doubled, from 2 to 4. A 4-dimensional Hilbert space now has room for two qubits, so another logical qubit has successfully been added.

An alternate way to reason about why a defect introduces a logical qubit is with homology theory. Adding a defect to a surface code lattice introduces a new boundary on which operator chains can end. Since logical operators correspond to strings that end on boundaries and that commute with all stabilizer generators, having more boundaries means there will be more such operators. Loops around the defect, trivial before a check was removed, are now nontrivial and also correspond to new logical operators.

Since there are two types of check operators, there are also two types of defects. Confusingly, for the surface codes, each type has two names. “Smooth” defects are also called Z -type defects, since they correspond to removed Z checks and have a logical X operator that ends on a smooth boundary. “Rough” defects are also called X -type defects, and they are created by removing X checks and have a logical Z operator that ends on a rough boundary. Both types are shown in Fig. 4.14, along with their associated logical operators.

These defects can interact with each other by *braiding*: one defect can be moved around another defect, using the code deformation techniques described in Appendix D, and returned to its original location, enacting a quantum gate. For the remainder of this dissertation, only the topological code defects introduced here are discussed.

Braiding defects around one another requires only the ability to move defects around the lattice, a procedure shown in Fig. 4.15. This sequence of measurements modifies the code without disturbing the logical information associated with the defect qubit. However, more elaborate deformations of the code allow for defects to execute nontrivial loops around other defects, leading to nontrivial action in the codespace. In particular, braiding a smooth defect around a rough defect performs a $CNOT$ gate, as I will now demonstrate. Code deformation is described in more detail in Appendix D.

In addition to moving the location of defects, code deformation techniques also modify the surface-code logical operators. By observing the way that a complete cycle modifies the logical operators of a smooth defect and a rough defect, it is simple to show that such a deformation enacts a logical $CNOT$ between the two qubits. The sequence of Figs. 4.16 to 4.19 shows this. It is demonstrated by showing that the braid has the correct action on the logical operators if the smooth defect is the control and the rough defect is the target.

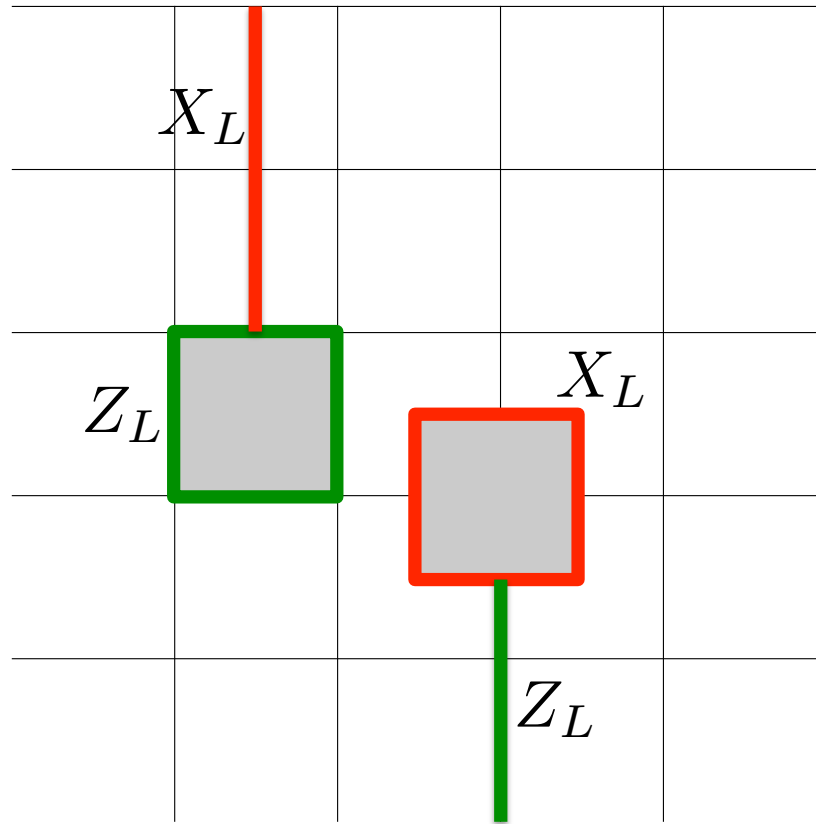


Figure 4.14: An example of the two types of defects that can be created in the toric code. One, corresponding to a Z -type face check that has been removed, has a logical Z operator equivalent to the removed face check and a logical X operator that tethers the removed face to a smooth boundary. The other—which corresponds to a X -type vertex check that has been removed—has a logical X operator equivalent to the removed vertex check and a logical Z operator that tethers the vertex to a rough boundary. More elaborate defect encodings are possible: for example, two smooth defects can be used to encode a single qubit. The important thing is that new boundaries are introduced, and any defect introduction in the planar version will change the boundaries. Note here that there are two types of boundaries in the graph without defects, but that the unpunctured lattice encodes no qubits.

Defects can also be created in the color codes in an analogous fashion. Here the defects are referred to only as X -type or Z -type based on the checks removed to create them. Additionally, they are assigned a color, and this is the same color as

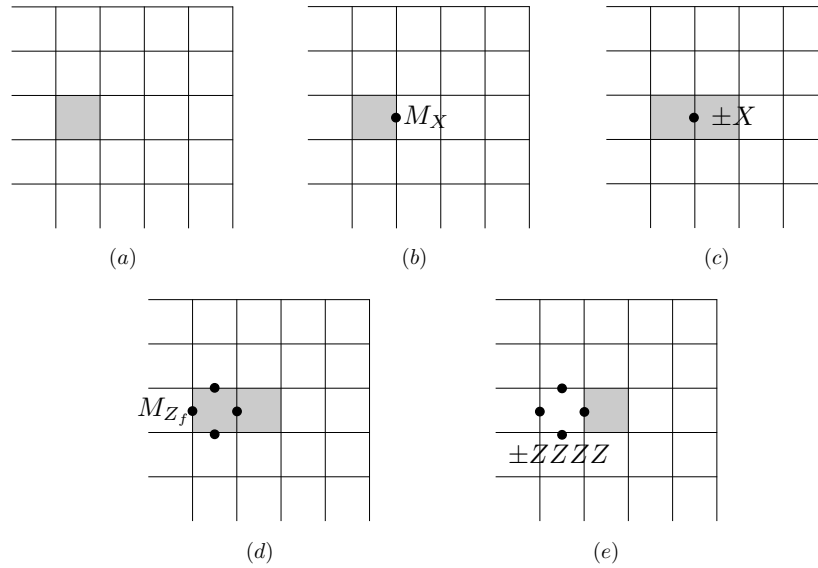


Figure 4.15: A measurement-based method for defect movement. (a) A single smooth defect in the toric code. (b) The movement procedure begins by measuring a single X on a qubit adjacent to the defect. This operator would normally anti-commute with two of the code’s check operators, but due to the defect it anti-commutes with only a single operator: the remaining adjacent face check. (c) This check is thus removed from the set of stabilizer generators and replaced with $\pm X$ based on the result of the measurement. The defect now spans two faces. (d) The original face occupied by the defect is remeasured. It commutes with all the remaining stabilizer generators, and anti-commutes only with the newly introduced $\pm X$ single-qubit check. (e) The reintroduction of the four-body face check removes the operator $\pm X$ from the stabilizer generators and replaces it with $\pm ZZZZ$. The defect has now been moved over one face at the cost of potentially modifying one of the check operators by a -1 .

the string that connects the defect to the appropriate boundary (X_L for a Z -type defect, for instance). The only subtlety arising from the color is that the color of the encircling operator (Z_L for a Z -type defect) is one of the other two colors. Color code defect encodings are discussed in Chapter 5, but their movement in terms of code deformation and the logical action of braiding—a $CNOT$ —are essentially the same as defects in the toric code. An example is shown in Fig. 4.20 for clarity.

The fault-tolerance of braiding [BMD09] is due to the fact that the defects are

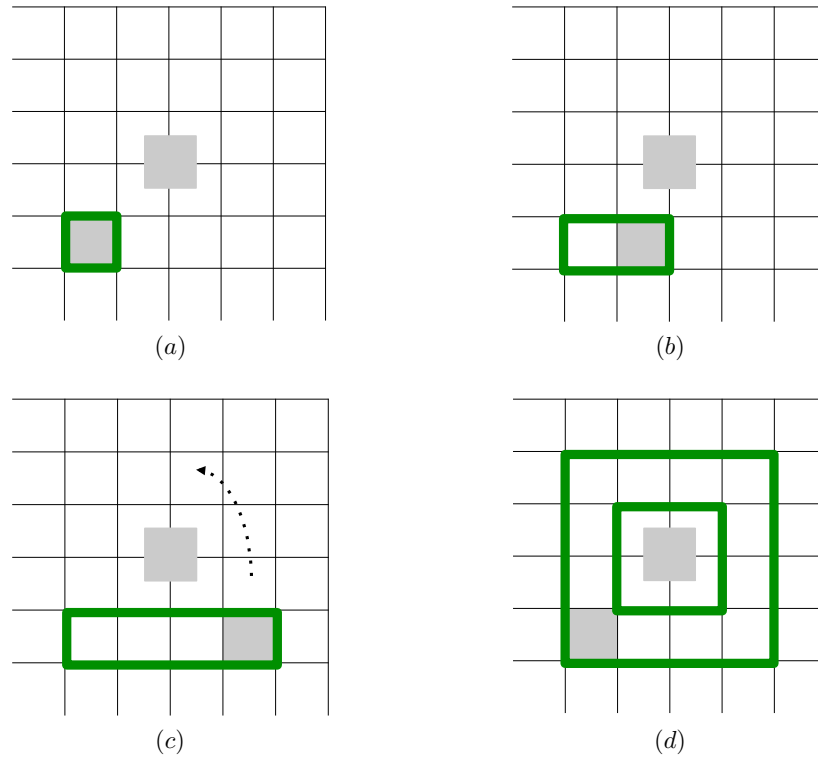


Figure 4.16: A sequence of code deformations showing that braiding a smooth defect around a rough defect acts on the logical space as $ZI \rightarrow ZI$. Frame (d) is equivalent to frame (a) by multiplication of stabilizer generators.

well-separated, and if movement operations are interleaved with error correction, errors cannot spread too badly. These code deformation techniques also describe the creation of defects and measurements, although in both cases a change in the surface topology is required—these deformations are not “smooth,” where this use of “smooth” has nothing to do with the boundary type or defect type, but rather with homeomorphisms. It is natural to ask whether or not the active error correction rounds can be abandoned for such code deformation techniques. Can the system be engineered to correct itself?

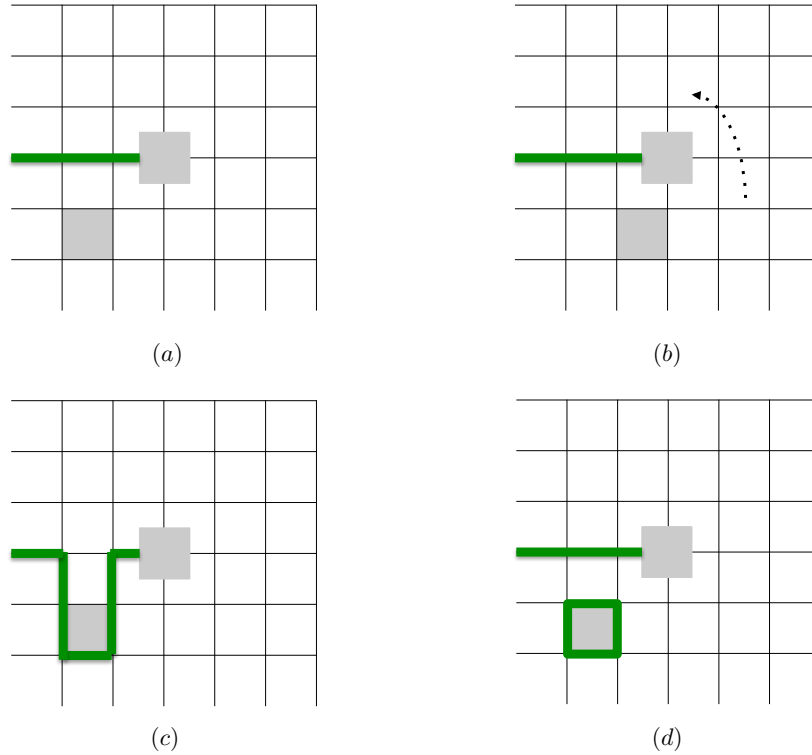


Figure 4.17: A sequence of code deformations showing that braiding a smooth defect around a rough defect acts on the logical space as $IZ \rightarrow ZZ$. Frame (d) is not equivalent to frame (a) by multiplication of stabilizer generators.

4.4.3 Possibilities for self-correction

Self-correcting codes utilize the physics of interacting qubits to passively prevent the catastrophic spread of errors. It is known that the 4-dimensional version of the toric code exhibits self-correction [DKLP02, AHHH10]. There are, unfortunately, many negative results for 2- and 3-dimensional systems [Yos11, BT09, KC08, CLBT10]. Ref. [Yos11] in particular draws a strong analogy between the topological nature of logical operators in a system and its stability to thermal fluctuations, providing the intuition already presented above that string-like logical operators tend to wander because of a constant energy barrier. The search for self-correcting quantum memories has recently been active in 3-dimensional systems that are not translationally

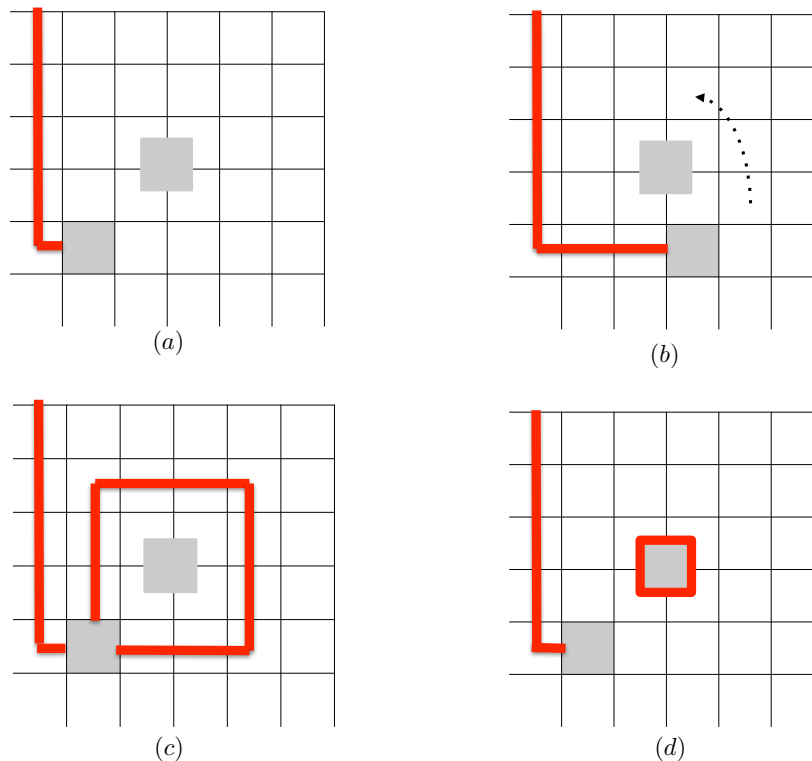


Figure 4.18: A sequence of code deformations showing that braiding a smooth defect around a rough defect acts on the logical space as $XI \rightarrow XX$. Frame (d) is not equivalent to frame (a) by multiplication of stabilizer generators.

invariant, breaking one of the key assumptions of the no-go results in Ref. [Yos11].

In particular, the cubic code, introduced by Haah in Ref. [Haa11], explicitly addresses the energy barrier problem by finding a local code in three dimensions that has logical operators with an energy barrier that grows logarithmically with system size. The self-correction properties of this family of codes was studied in Refs. [BH11] and [BH13], which found that for system sizes smaller than a critical value, the memory lifetime was a polynomial in the linear size of the system. Ideally, a self-correcting memory has a lifetime that is exponential in the system size. The 4-dimensional toric code has this feature [AHHH10], but the search is ongoing for a code with more physical spatial locality demands.

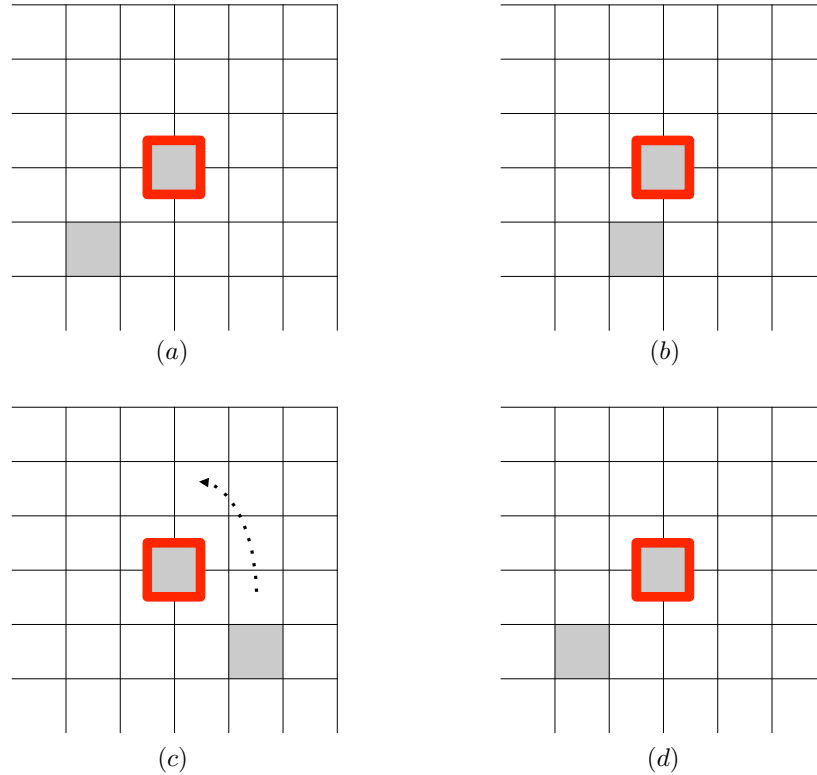


Figure 4.19: A sequence of code deformations showing that braiding a smooth defect around a rough defect acts on the logical space as $IX \rightarrow IX$. Frame (d) is trivially equivalent to frame (a).

4.5 Topological code thresholds

One feature of particular interest for any code—quantum or otherwise—is the code threshold. This is a value, p_{th} , of the physical error probability below which the code suppresses the *logical* error probability $p_L < p_{th}$. Above the threshold, the physical qubits are so errant that there is no advantage to encoding—it would be better off to simply leave the information in unencoded form. The true threshold is a thermodynamic quantity, and characterizes a phase transition from a “correctable” phase ($p_L = 0$) to an “uncorrectable” phase ($p_L = 1$). There is a step-function transition as a function of p between these two regions in the limit of an infinite

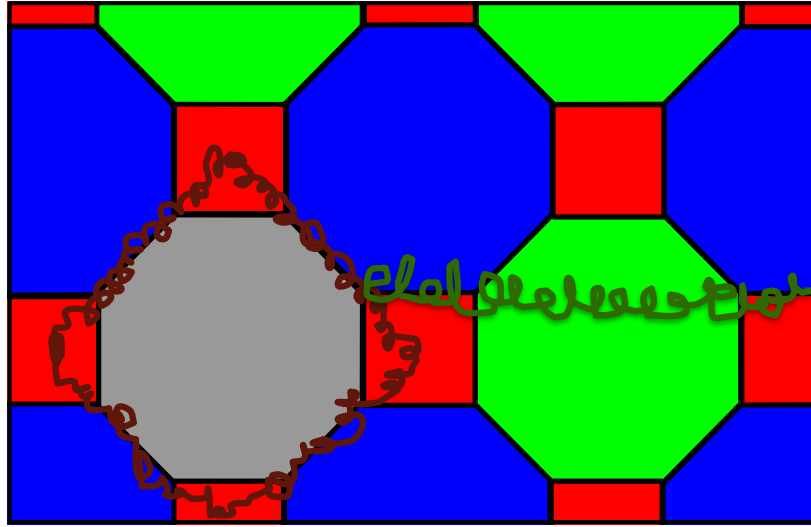


Figure 4.20: A green defect, either of X -type or Z -type in the 4.8.8 color code. The string-like operator connecting to a boundary shares the color of the removed face, and the encircling operator has one of the other two colors.

system size. However, the Monte Carlo methods typically used to calculate the threshold can only simulate finite-size systems, and the threshold can wiggle around due to finite size effects. In these cases, it is more proper to refer to the threshold as a pseudthreshold that approaches the true threshold as the sizes of the codes are increased.

Threshold calculations for the topological codes can be performed in a variety of ways and in a variety of settings. One might assume that stabilizer group measurements are error free and that ancilla qubits can be prepared perfectly. In the literature this is usually referred to as the code capacity threshold, as it closely resembles the absolute error rate the codes can handle in the presence of only data errors. By more carefully modeling measurement and ancilla errors, more realistic thresholds can be calculated, and the fully fault-tolerant treatment that models each step in the error correction process as faulty provides the best estimate for the purposes of quantum computation. (It is a bit of a leap to claim that the fault-tolerant threshold

for error correction can be identified with the fault-tolerant threshold for universal quantum computation. However, the protocols needed to allow for universality can be designed in a fault-tolerant way and interleaved with rounds of error correction. The threshold for computation will be slightly reduced from the threshold for error correction due to there being more spots in a circuit for errors to occur, but that reduction will not be dramatic. The extended rectangle analysis in Ref. [AGP06] studies this problem.)

Monte Carlo threshold estimation proceeds by choosing an error model, a code, and a classical decoding algorithm. Depending on the setting, random errors are applied to the data qubits, the data qubits and the syndrome qubits, or each gate in a full circuit treatment, and the decoding algorithm examines syndrome measurements and makes a guess for an action that will correct the errors. The correction either leads to a logical fault or it doesn't. The procedure is repeated and the fraction of logical failures out of the number of rounds of simulation gives the logical failure rate for a given p . Many different values for p are used to trace out a curve of p_L vs p , and the point at which this curve crosses the line $p_L = p$ is the pseudothreshold for the chosen code. Many decoders have been used in threshold calculations for the surface codes and color codes. Some of these are listed here:

1. Minimum-weight perfect matching [DKLP02, RHG07, RHG06, WFSH10, Ste14]
2. Renormalization group methods [DCP10]
3. Greedy expanding diamond algorithms [Den03, Woo13]
4. Integer programming methods [LAR11]
5. Optimal (free-energy minimizing) decoders [KBMD09]

A typical error pattern for a surface code in the non-fault-tolerant setting is shown in Fig. 4.21. A decoder starts with the error pattern—just the red dots in Fig. 4.21.

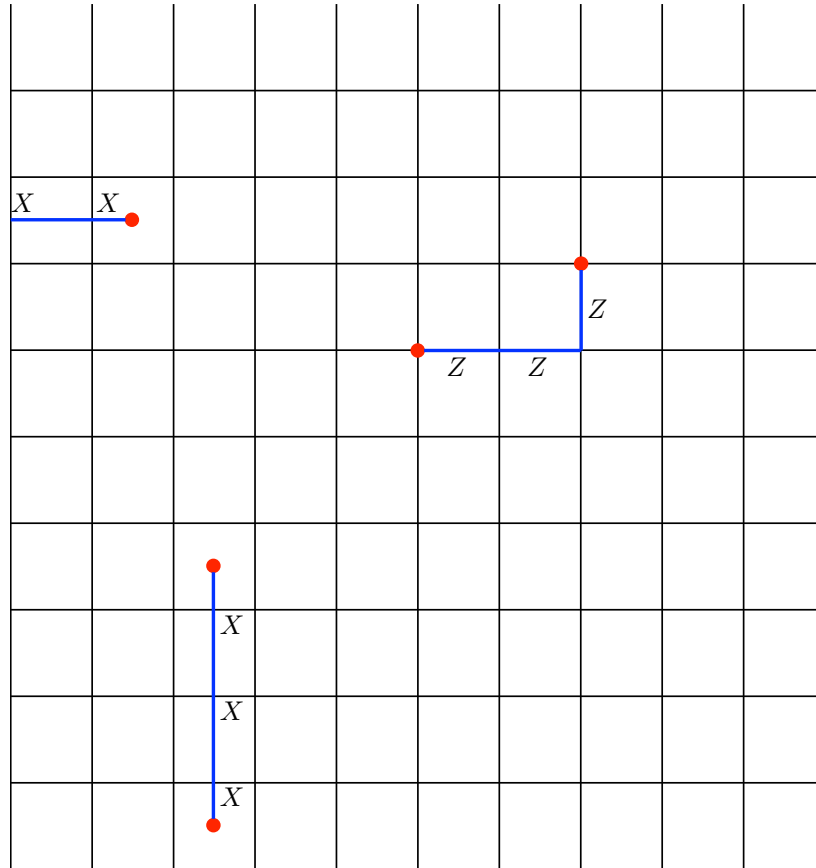


Figure 4.21: A typical error pattern for the toric code in a code-capacity setting. The decoder only gets to see the endpoints of error chains (the red dots). The decoder may identify the corrective action with an error that is consistent with the given syndrome. If the selected error and the actual error sum to something in the equivalence class of logical operators, then the algorithm fails; otherwise, it succeeds. For a given value of p , the value of p_L , the logical failure probability, is then the fraction of times the decoding fails. The same algorithm will be run with codes of different distance to study any finite size effects and to examine the sub-threshold error suppression achieved by moving to larger distance codes.

The first step of a decoder is often to identify an error that is consistent with the positions of the violated checks. The decoder typically favors the lowest weight error, as that is also usually the most likely of all the errors that could have caused the given syndrome, but this is not necessarily the optimal approach to take. The degeneracy

of the errors could also be taken into account. For instance, perhaps there is only one error of weight w consistent with the syndrome, but there are 100 errors of weight $w + 1$ that are consistent with it. Depending on the physical error probability p , the latter set—with 100 members—might be more probable *on the whole* than the singleton set with a weight w error. The example is contrived, but the idea in optimal decoding is the same: take the degeneracy of errors into account and select the most likely class instead of settling on the lowest weight error. This allows for minimizing the error in decoding instead of maximizing the probability of identifying the error that actually occurred.

4.6 Magic states and universality

The Eastin-Knill Theorem [EK09] states, with a modest set of assumptions, that no quantum code can admit a universal and transversal set of gates. Research on magic state distillation—initiated by Bravyi and Kitaev [BK05] but foreseen by Shor [Sho96]—allowed for the fault-tolerant extension of non-universal gate sets by adding one additional operation to the Clifford group: the ability to prepare mixed states whose purity could be increased by special protocols. To perform a quantum computation, one then typically proceeds as in Fig. 4.22. The situation described in the figure involves classical steps and quantum steps and is a particular on-demand way of imagining the operation of a quantum computer. It does not require a single machine that can run any algorithm, but neither is it incompatible with such a notion. It may even inform a decision on what a generic all-purpose machine might look like, and I will comment on this later. However, it is not the only way to imagine a fault-tolerant architecture. Recent work by Gottesman [Got13] shows that it is possible, in principle, to perform a constant-overhead fault-tolerant simulation of a given circuit, but this work leverages a family of quantum codes, the hypergraph

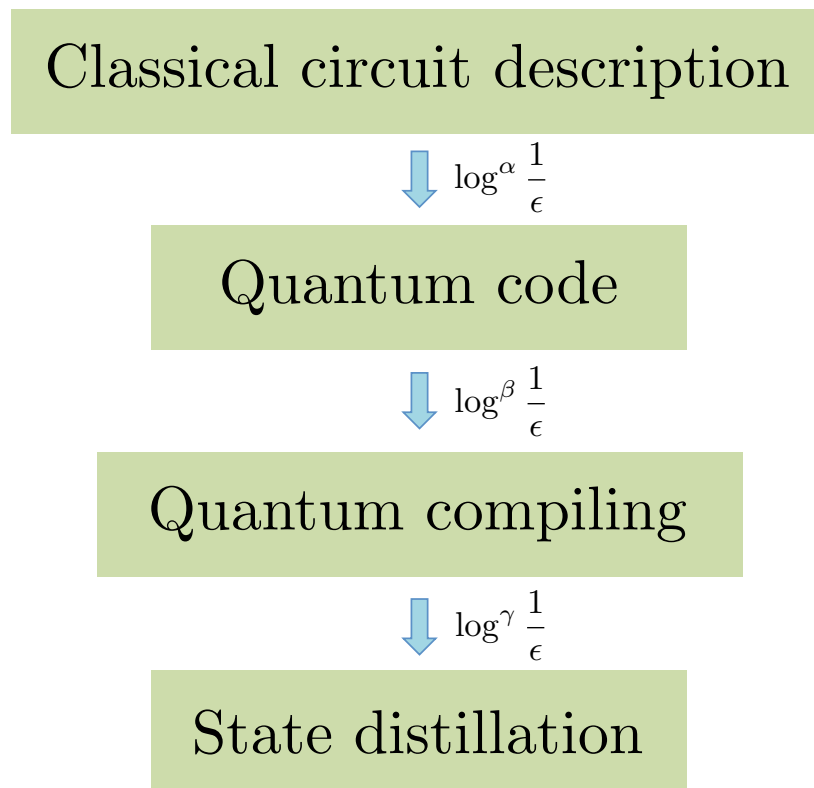


Figure 4.22: A flowchart for an on-demand quantum computer. First, a classical algorithm generates a description of the quantum circuit needed to solve some problem. Next, a quantum code is chosen to allow for non-ideal quantum gates and to provide easily fault-tolerant operations. The choice of code provides a natural universal gate basis to compile over, and the next step produces unitary approximations to all the gates in the circuit that can't be performed exactly. Finally, the gate basis will have some easy gates and some hard gates, and the hard gates are implemented via gate teleportation of distilled magic states. Each step introduces a resource overhead, explained more fully in the text.

product codes [TZ09], for which there is no known efficient decoder. For the purpose of concreteness, I will consider the series of steps natural to the topological codes I've been discussing and ignore other proposals.

I will briefly describe the steps listed in Fig. 4.22. First, given the inputs to the quantum algorithm—for Shor's algorithm this would be the number to factor—

Chapter 4. Topological Fault-Tolerance and Avenues to Universality

a classical computer generates a description of the quantum circuit that needs to be run. This is the ideal circuit that would provide the correct answer (perhaps probabilistically) in the absence of all errors. Next, a topological code is chosen to combat errors and allow for the implementation of as many easy operations as possible (ideally the entire Clifford group). This causes a blowup in the number of qubits needed for each of the original ideal qubits that is polylogarithmic in the desired logical error rate ϵ . (In this flowchart I abuse the ϵ notation to avoid a morass of ϵ 's and ϵ ''s. I will clearly state their meanings here in the text.) Quantum compiling, which is a name for a family of classical approximation algorithms, is then performed to get ϵ -approximations to all the gates in the circuit that are not already available. This compiling is done over a universal gate set \mathcal{G} , and a typical choice is $\mathcal{G} = \{H, S, CNOT, T, S^\dagger, T^\dagger\}$. Thus, for each U in the circuit that needs to be approximated, the quantum compiling algorithm produces a sequence of $G_i \in \mathcal{G}$ that is ϵ away from the target U in a chosen distance measure (usually trace distance). The overhead introduced here is in the number of gates required to approximate U , and there exist protocols compiling over the \mathcal{G} basis mentioned above which have $\beta = 1$. Lastly, some of the gates in \mathcal{G} are typically “easy,” meaning they have a natural fault-tolerant implementation with constant overhead, and some of the gates are “hard.” The T gate is usually one of the hard gates to perform, and magic state distillation protocols are required which take faulty copies of the state $T|+\rangle$ and return copies of higher fidelity. Distilling a $T|+\rangle$ state with a target fidelity of ϵ incurs an additional polylogarithmic overhead, and the current best value is $\gamma = 1.6$ [BH12]. It is unknown if $\gamma = 1$ can be achieved.

In recent years, the quantum compiling and magic state distillation steps above have received a lot of attention, as researchers have tried to beat down resource costs to the minimum achievable [BH12, Sel12, KMM13a, MEK12]. Other lines of research have attempted to circumvent the assumptions of the Eastin-Knill Theorem by demonstrating ways of designing fault-tolerant circuits without the use of

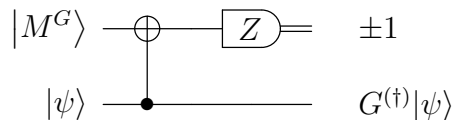


Figure 4.23: A quantum circuit that uses the state $|M^G\rangle$ to apply the gate G or G^\dagger on $|\psi\rangle$. The gate applied depends on the outcome of measuring the first qubit, with a $+1$ heralding an application of G and a -1 heralding its inverse G^\dagger . This circuit can be made deterministic by allowing a corrective gate on the bottom qubit classically controlled on the -1 measurement outcome. However, for magic states yielding small Z rotations, this corrective gate (G^2) will most likely also have to be applied via a magic state.

transversal gates [JOL13] or by leveraging unneeded degrees of freedom in gauge codes [PR13]. It is safe to say that no one has yet discovered the definitive optimal approach in terms of the added resources required to achieve universal encoded quantum computation. In Chapter 7 I present work that essentially hops over the quantum compiling step in the flowchart above and directly distills gates capable of implementing Z rotations by angles of $\pi/2^k$. As that chapter comprises a significant portion of the original work in this dissertation, I want to briefly describe the idea of magic states.

A magic state for performing a gate G that is diagonal in the computational basis is given by

$$|M^G\rangle = GH|0\rangle = G|+\rangle = \frac{1}{\sqrt{2}} (|0\rangle + e^{i\theta}|1\rangle), \quad (4.5)$$

where G can be read off as

$$G = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}. \quad (4.6)$$

The circuit in Fig. 4.23 then implements the gate G or G^\dagger in a random fashion. The ability to perform Clifford group operations, naturally fault-tolerant for many quantum codes, is augmented by the ability to prepare ancilla qubits in $|M^G\rangle$ states above some threshold fidelity, completing the set of gates to one that is universal. Preparing high-fidelity copies involves a protocol for distilling them from lower fidelity

copies using only Clifford gates and measurements in the computational basis. (This condition on allowing only Clifford gates can be relaxed as long as a full accounting of the required resources is made.) Success of these protocols relies on having input states below a certain error threshold (examples of which are calculated below) and is heralded by the results of single qubit measurements furnishing a protocol's final step.

The calculation of the threshold for a given protocol proceeds by examining the action of projecting a product state of noisy inputs $\rho^{\otimes n}$ onto the codespace of an $[[n, k, d]]$ quantum code. To determine the logical state ρ_L one ends up with after such a projection, it suffices to find matrix elements in the $\{|0_L\rangle, |1_L\rangle\}$ basis to construct

$$\rho_L = \begin{pmatrix} \langle 0_L | \rho^{\otimes n} | 0_L \rangle & \langle 0_L | \rho^{\otimes n} | 1_L \rangle \\ \langle 1_L | \rho^{\otimes n} | 0_L \rangle & \langle 1_L | \rho^{\otimes n} | 1_L \rangle \end{pmatrix}. \quad (4.7)$$

Assuming that the noisy input state is

$$\rho = (1 - \varepsilon) |M^G\rangle\langle M^G| + \varepsilon |-M^G\rangle\langle -M^G|, \quad (4.8)$$

where $|-M^G\rangle$ is the unique pure state orthogonal to $|M^G\rangle$ (an assumption justified by a preparatory twirling operation, or results due to Jochym-O'Connor *et al.* [JOYHL13]), the threshold can be backed out by finding ε_{out} from

$$\rho_L = (1 - \varepsilon_{out}) |M_L^G\rangle\langle M_L^G| + \varepsilon_{out} |-M_L^G\rangle\langle -M_L^G|, \quad (4.9)$$

which will be in terms of ε . An alternate way of viewing this procedure is as a measurement of the logical operator that has the state $|M^G\rangle$ as an eigenstate, namely GXG^\dagger . Measurements in a Pauli basis can be converted into such a logical measurement by simply rotating qubits prior to the measurement. For the case of transversal logical operators, these rotations are simple to enact. It is also possible to derive the distillation threshold by alternative methods. For instance, Ref. [MEK12] uses a method of counting all the possible locations of $O(p)$ errors in the distillation circuit.

This is a useful tool for analyzing more complicated distillation protocols that don't rely as heavily on symmetries of the code.

4.7 Outline of the remaining chapters

The remaining chapters of this dissertation comprise some of the original research I've performed over the last six years. Loosely, they explore topological code architectures for quantum computation. In Chapter 5, I give a largely pictorial description of a result about different defect encodings for the 4.8.8 color code. It demonstrates the intuition that homology provides about what can be done by continuous deformation in topological codes. Chapter 6 describes a model of quantum computation that uses local adiabatic evolutions and the toric code to simulate a measurement-based model, while maintaining a Hamiltonian gap that is constant in the problem size. Lastly, Chapter 7 introduces a magic-state distillation protocol that attempts to circumvent the quantum compiling step shown in Fig. 4.22. By directly distilling magic states capable of performing Z rotations by angles of $\pi/2^k$, it is possible to save on the required number of resource states compared to a near-optimal protocol based on the standard method of approximation. Along the way, I introduce a new family of quantum codes that allows for transversal application of the $\pi/2^k$ gates. While not topological codes themselves, the encoding circuits for these codes have natural fault-tolerant realizations in topological codes, and so distillation protocols based on their encoding circuits are easily implemented in topological code substrates.

Chapter 5

Relationships Between Defect Encodings in Topological Color Codes

The role that topology plays in topological codes is important, and in Appendix A I demonstrate that stabilizer generators and logical operators have descriptions in the language of homology. In this chapter I present a related argument about the difference between two strategies for using defects in the color codes. As was mentioned in Chapter 4, using defects in topological codes is a simple way to introduce more logical qubits to a code. This can be understood in two ways. From the viewpoint of stabilizer codes, introducing defects decreases the number of stabilizer generators and, hence, increases the number of logical qubits. From the viewpoint of homology theory, introducing defects yields a different first homology group for the surface. The new nontrivial cycles are precisely the logical operators for the newly introduced qubits.

The importance of allowing topology-changing operations has been recognized in proposals for fault-tolerant quantum computing with defects in topological codes [RHG07]. However, here I want to study the similarities between two different ways

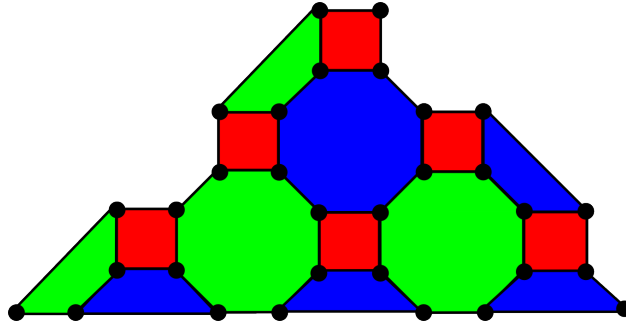


Figure 5.1: A distance-7 instance of the 4.8.8 topological color code. The label 4.8.8 corresponds to number of edges of the three faces adjacent to vertices away from the boundary: one square and two octagons. Color codes are defined by graphs that are face 3-colorable with the qubits on vertices and the stabilizer generators on faces. Each face actually corresponds to two generators: one X type and one Z type.

of using defects, and I probe these similarities in a way motivated by topology: I seek to turn a qubit encoded in a single defect into a qubit encoded in a *triple defect*, first described in Ref. [Fow11b]. In the process, I do not want to allow any topology-changing operations, as such actions will confuse the issue by changing the number of logical qubits. In some sense this is an artificial restriction, but my goal is to explore the nature of different defect encodings instead of proposing a useful quantum computational protocol.

Throughout this chapter I will use members of the 4.8.8 family of triangular color codes—pictured in Fig. 5.1—due to the fully transversal availability of the Clifford group. Defects will be either X -type or Z -type and will have a color given by the color of the removed face. Recall that in the color codes only the faces correspond to stabilizer generators, and that the boundaries can also be given colors. For example, the bottom boundary in the figure has the color red because it is the color that is missing. Red strings, which connect red defects to the boundary, can end at the lower boundary. I will abstract away most of the detailed colors in this chapter and draw instead only the boundary colors and the colors of the defects. This will make

the numerous figures less busy and simpler to parse.

The goal will be to convert three single defects—one in a state $|\psi\rangle$ and two in a known state—into a triple defect in the same state $|\psi\rangle$. It is possible to simply teleport a single defect into a triple defect prepared in a fiducial state. While this captures the appropriate flow of quantum information, it does not address the topological question of whether or not there is an operation that can take three defects and turn them into a triple defect through local deformations. The demonstration will proceed along the lines of a sequence of obstacles and solutions until the solutions have led to an insurmountable problem. The takeaway is that the triple defects are a different beast, and this is a reflection of the structure they share with the overall code.

The original motivation for this work was to optimize color code computations for space savings. The idea was to store qubits that weren't participating in any gates in single defects. Then, when a gate was called for, a single-defect qubit would be turned into a triple defect for ease of logical gate application. After the gate, the qubit would be returned to the single-defect encoding. This can be done using teleportation, but the setting of this chapter is an exploration of the ways in which the single-defect and triple-defect encodings are distinct. This is the nature of the following discussion.

5.1 Triple-defect qubits

Before introducing triple defects, I'd like to abstract away the details of the color code bulk. Instead of showing each individual qubit and face, I will just depict the boundaries. The color code in Fig. 5.1 becomes the arrangement of colored boundaries in Fig. 5.2. With this simplification made, the string-net logical operators appear like Fig. 5.3. Again, the geometry of the logical operators doesn't matter.

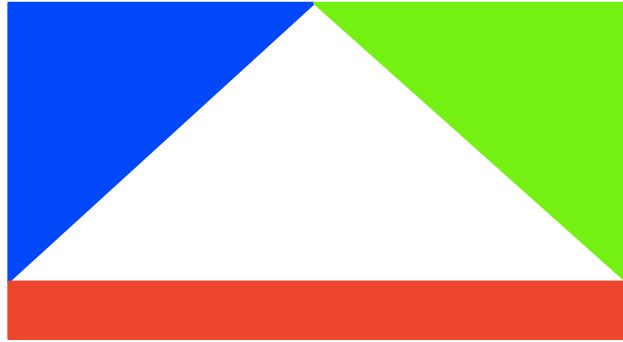


Figure 5.2: The bulk of a 4.8.8 triangular color code abstracted away and the boundaries labeled by their color.

All that matters is that the endpoints of the strings are fixed to boundaries and that any color splitting is consistent.

I now introduce the triple-defect encoding, shown in Fig. 5.4. It shows three regions of Z -type stabilizer generators removed, one of each color. X_L in this case has a string-net structure, while Z_L is depicted as a green string around the blue defect region. In fact, removing three regions like this creates *three* new logical



Figure 5.3: The string-net logical operator for a 4.8.8 triangular color code with the bulk structure abstracted away. These strings can be bent and pushed around, even split apart if the proper rules are followed. However, the colored strings must always end on the appropriate colored boundary.

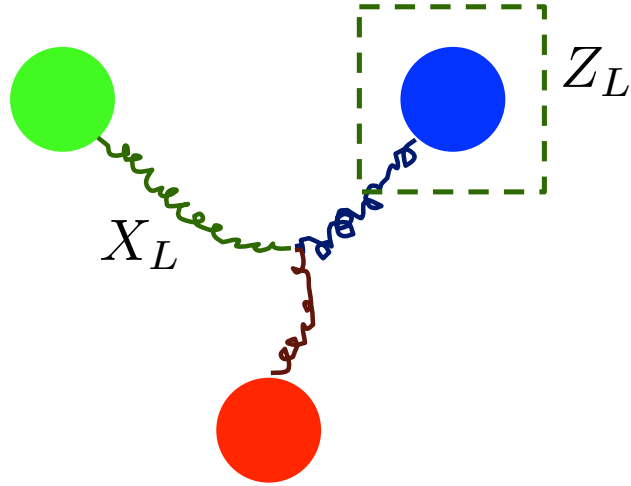


Figure 5.4: The triple-defect encoding. Z -type stabilizer generators are removed from three regions, each with a different color. This creates three new logical qubits, but two are ignored as gauge degrees of freedom. The remaining qubit has logical operators chosen as shown, where there is additional freedom in the choice of Z_L . Due to the gauge fixing condition that the other two qubits are in the state $|0\rangle$, Z_L could be chosen to be an enclosing loop around any of the three regions. Note the structure of X_L and its relationship to the structure of X_L for the string-net operators associated to the original qubit encoded in the surface (as shown in Fig. 5.3).

qubits, but two of the qubits are essentially being ignored. In fact, since the system was in the $+1$ eigenstate of all the operators removed to create the defects, they all initially encode the state $|0\rangle$. If the two extra qubits are ignored and if they do not succumb to any logical errors, then their own logical operators can be treated as a physically meaningless gauge degree of freedom, as mentioned in Sec. 3.4. In this case, the gauge is fixed: both ignored qubits are in the state $|0\rangle$. This allows for the manipulation of the logical operators of the remaining qubit, and they can be chosen as pictured in Fig. 5.4. The operator Z_L is not unique—it can be deformed by gauge logical operators to be an enclosing loop around any of the defect regions, provided it is of a different color than the region itself.

The advantage of this encoding is that the logical Pauli operators, the logical

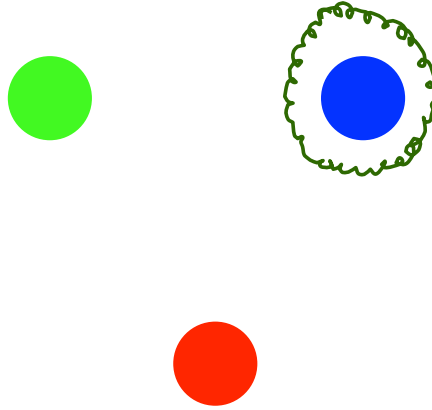


Figure 5.5: A logical Z operator for a triple-defect qubit.

Hadamard, and the logical S can all be applied in a simple transversal manner. The application requires the defects to be deformed in a particular way, but first I want to show that Z_L has a form that is similar to X_L . The loop of Pauli Z operators shown in Fig. 5.5 is one choice for Z_L . However, as for all stabilizer codes, this is not its unique form. Fig. 5.6 shows that the singly-colored loop can be split at two points into the two other colors. Multiplication by stabilizer generators allows the three different colors to be brought into contact with the three colored boundaries

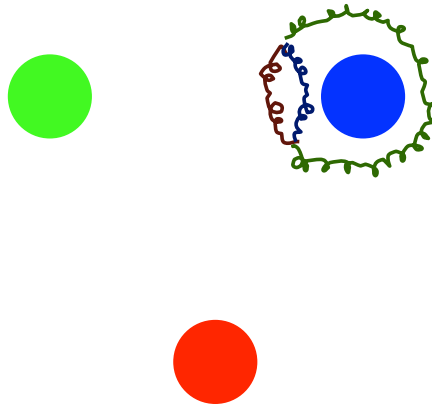


Figure 5.6: An equivalent logical Z operator for a triple-defect qubit using the color code rules about the splitting of colored strings.

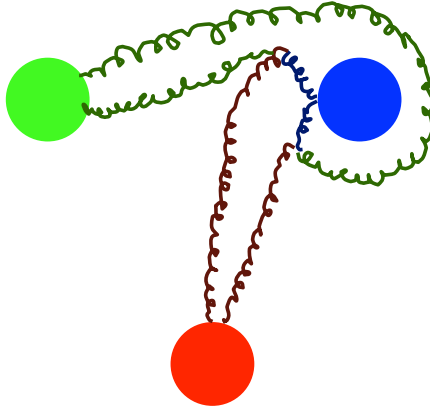


Figure 5.7: An equivalent logical Z operator for a triple-defect qubit using the color code rules about the splitting of colored strings and multiplication by stabilizer generators. There is no real distinction between these two rules, as the colored string splitting is just another instance of stabilizer generator multiplication.

provided by the defects, as shown in Fig. 5.7. Finally, further stabilizer generator multiplication deforms Z_L into the pattern shown in Fig. 5.8.

Now that I've introduced an alternate form for Z_L , I will show how to prepare the defects for the application of the transversal (single-qubit) Clifford group. First, the three defects are deformed such that they touch and enclose a region, as shown in Fig. 5.9. This isolates X_L from the rest of the code, as shown in Fig. 5.10, and

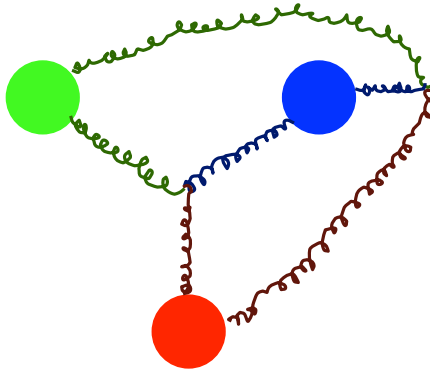


Figure 5.8: An equivalent logical Z operator for a triple-defect qubit.



Figure 5.9: The three defects from Fig. 5.4 deformed so that they enclose a region of a color code. Note that the enclosed region now just looks like a smaller version of the full triangular code.

partially isolates Z_L , as shown in Fig. 5.11. (For the case of X -type triple defects, the same story holds with the roles of X_L and Z_L reversed.) In order to properly apply a transversal logical operator, the exterior remnant of Z_L , called a “byproduct operator” must be measured. This “pruning” amounts to measuring a small exterior

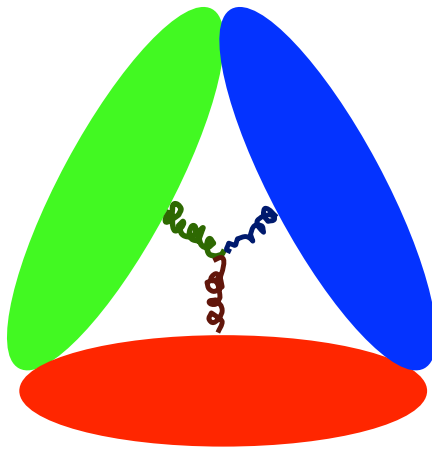


Figure 5.10: The three defects are deformed so that they touch and enclose a region of the code, isolating X_L in the process.

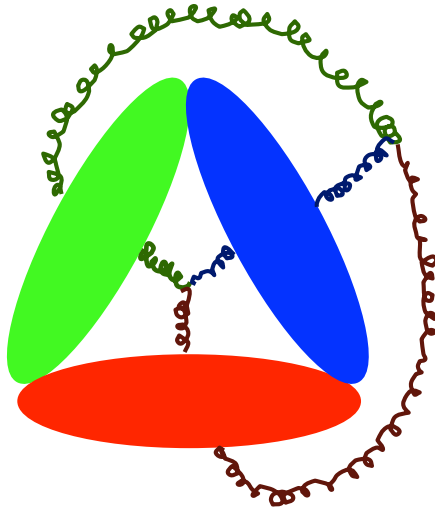


Figure 5.11: The three defects are deformed so that they touch and enclose a region of the code, partially isolating Z_L in the process but leaving a “byproduct operator” external to the isolated region. This byproduct operator must be measured before any logical operators are applied.

region and learning the eigenvalue of the exterior operator. The result of the measurement is stored and used to modify the outcome of any future Z_L measurement. Then, with the exterior part of Z_L removed, the desired logical operator is performed transversally on the interior. Next, the check operators removed by the exterior measurement are reintroduced, and the code in the exterior region is fixed. Finally, the defects are shrunk back to their original size and the computation proceeds.

There are subtleties related to whether or not S_L is transversal on the interior region, but the criteria for S_L transversality are given in Appendix C and can be checked. The interior region can be sized appropriately for either transversal S_L or transversal S_L^\dagger , both implemented with transversal physical S .

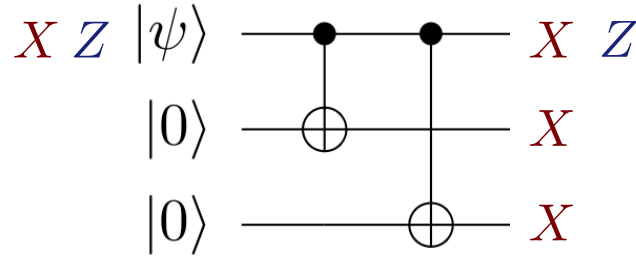


Figure 5.12: The three-qubit repetition code encoding circuit, showing that X_L for the single defect is mapped to XXX on the three single defects and Z_L for the single defect predictably is not modified. The propagation of Z'_L and Z''_L for the two other defect qubits and the gauge fixing condition—that $Z'_L = Z''_L = +1$ —allows for the triple defect Z_L to be a loop around any of the three defects.

5.2 Failure of conversion via topology

I will now present the sequence of obstacles and solutions that lead to the inability to convert three single defects into a triple defect. As before, I will consider Z -type defects, and the goal will be to take three single defects, encoding $|\psi\rangle$ and two $|0\rangle$ states, and turn them into a triple defect.

I begin by noting that using a repetition encoding almost seems to do the trick. The effect on the logical operators is nearly the desired effect, modulo a leftover surface logical operator. Fig. 5.12 shows that the X and Z operators are mapped to the appropriate things through the repetition code encoding circuit. The form of Z_L after the circuit is precisely what is needed for the triple defect, including the gauge freedom of multiplying by the gauge-fixed logical Z operators for the two single defects in the state $|0\rangle$. However, it is not clear that the product of the three X operators is the same as the string-net version shown in Fig. 5.4. Additionally, there is no direct way to perform a $CNOT$ between two Z -type single-qubit defects. This latter objection is easily overcome by the circuit shown in Fig. 5.13, which requires two additional single defect ancilla per $CNOT$. It turns out that the three

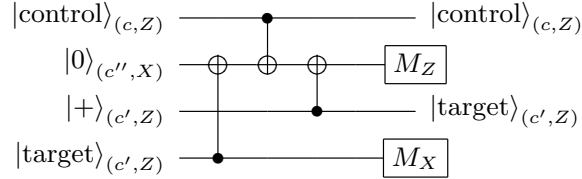


Figure 5.13: A circuit allowing for a $CNOT$ to be performed between two defects of the same type. The labels c , c' , and c'' correspond to the color of the defects. Since a $CNOT$ cannot be performed between defects of the same color, c must be a different color than c'' and c' must be a different color than c'' . However, c and c' are allowed to be the same color.

X operators are almost equivalent to the proper logical string-net operator, but I will have to enforce another gauge fixing condition. First, notice the sequence of deformations shown in Figs. 5.14 to 5.18. This collection of figures shows how the three X operators—which tether each of the Z -type defects to the appropriate boundary—can be deformed into the desired string-net X_L operator for a triple defect and a leftover piece that corresponds to the logical X operator of the qubit encoded in the surface. Rather than repeating them in the main body of the text, I'll let the figure captions tell the details of the story.

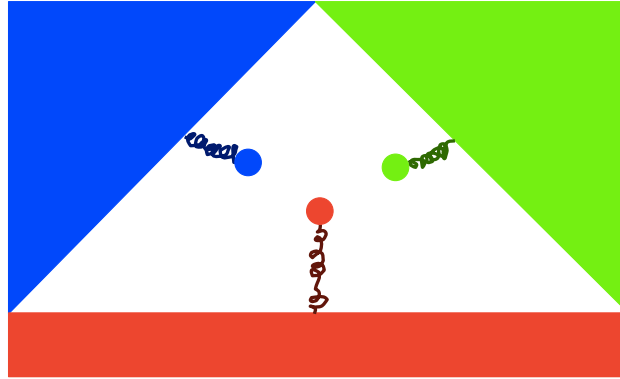


Figure 5.14: After the circuit in Fig. 5.12, X_L is composed of the three string-like operators shown.

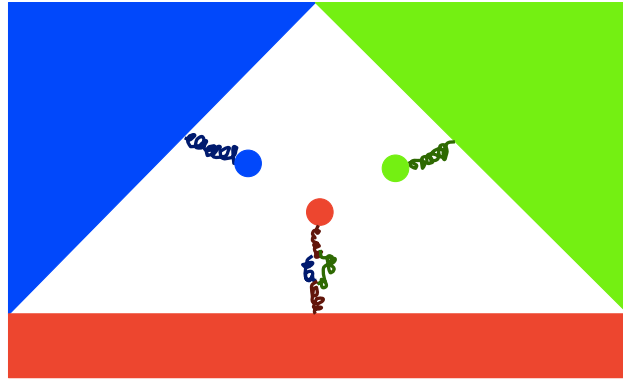


Figure 5.15: The red string-like part of X_L can be equivalently represented with a split into green and blue strings that fuse back together.

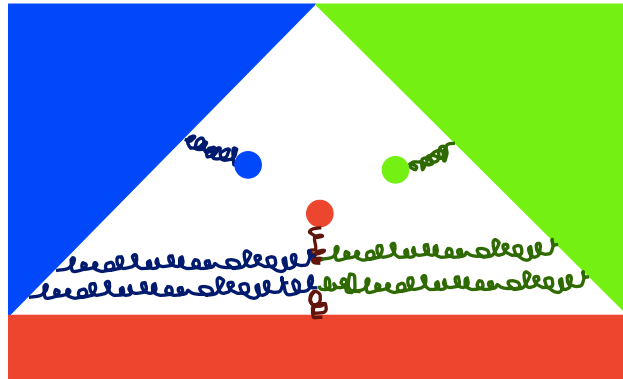


Figure 5.16: Through the multiplication of stabilizer generators, the blue and green splitting can be brought in contact with the boundaries of appropriate color.

The situation seems good if it is possible to make the qubit encoded in the surface another gauge degree of freedom. If it's possible to prepare this qubit in the state $|+\rangle$, for instance, then the leftover part in Fig. 5.18 would simply correspond to a multiplication by $+1$ of the true triple defect X_L —in other words, it would be a trivial modification and could simply be ignored. However, this would only benefit the use of Z -type triple defects. The X -type triple defects would not yield to the same kind of modification of Z_L . The trick around this is to not use a triangular code, but rather to use a code with boundaries that encode more than one qubit.

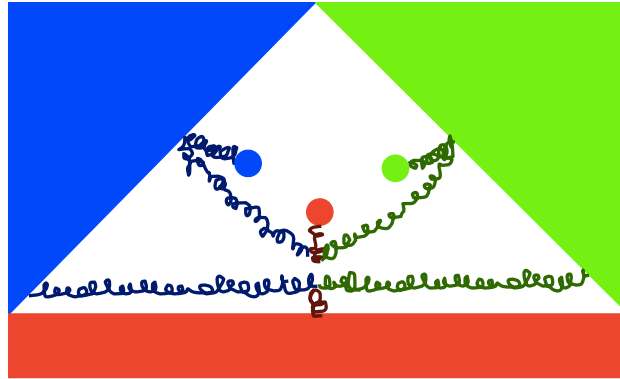


Figure 5.17: Further multiplication by stabilizer generators allows for the endpoints of part of the split blue and green strings to merge with other blue and green endpoints coming from the other two defects.

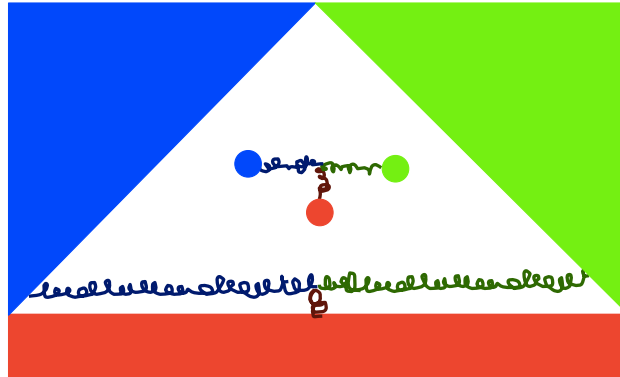


Figure 5.18: A last round of stabilizer generator multiplication pulls the joined blue and green strings off the boundary and shows that the X_L is nearly the same as for the triple defect encoding.

Pictured schematically in Fig. 5.19, one of the logical qubits could be prepared in $|+\rangle$ and one could be prepared in $|0\rangle$. Then, triple defects of the appropriate type could be created in the appropriate region, allowing for the necessary gauge fixing requirements. The code distance for the qubits encoded in the original, unpunctured surface will be much larger than for the qubits encoded in defects—be they single or triple. In fact, it will grow linearly with both the distance of the defect qubits and their number—in big- O notation, the surface qubit distance will be $O(dn)$. This

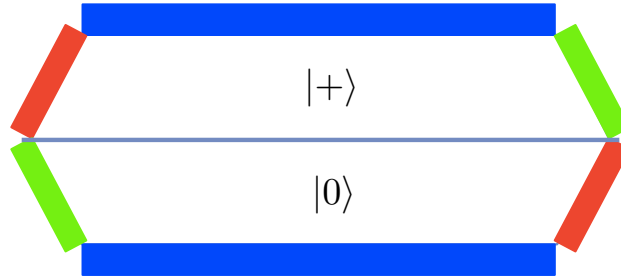


Figure 5.19: The preparation of a color code surface encoding more than one logical qubit. This allows the different surface logical qubits to have different fixed gauges, which can be used to remove the leftover surface operators in Fig. 5.18.

means that it will be very unlikely for the gauge fixing condition to be broken by the action of random errors.

At this point, the problem seems solved. Three single defects have been converted into a triple defect, so what's the catch? The catch is that I have neglected the presence of other defect qubits that are floating around in the surface. After all, the goal is to perform a computation, and many qubits will be needed. Figs. 5.20 to 5.22 show how the presence of other defect qubits can interfere with the deformations presented in Figs. 5.14 to 5.18. At this point, the gauge fixing trick runs out of

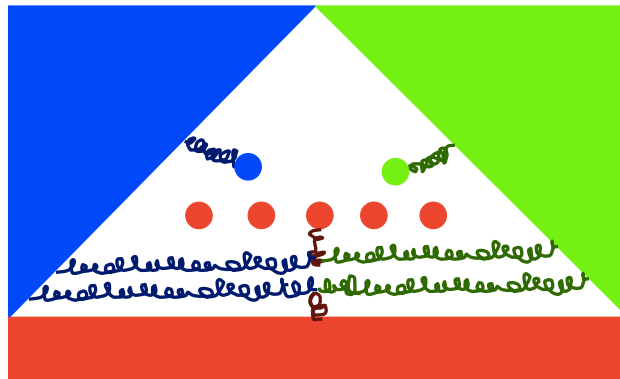


Figure 5.20: A more realistic look at the deformations leading up to Fig. 5.18.

steam. It is not reasonable to demand that each of the qubits in the computation be

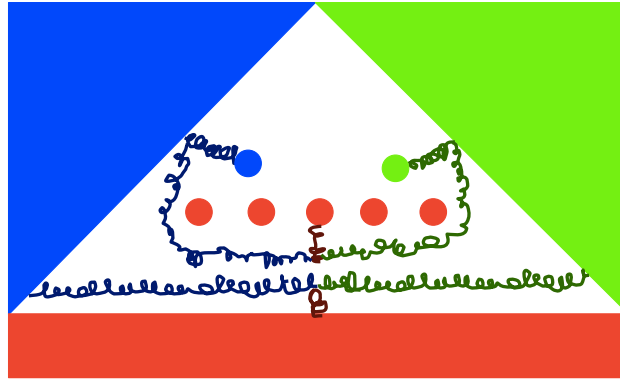


Figure 5.21: Stabilizer generator multiplication still allows the blue and green endpoints to be fused.

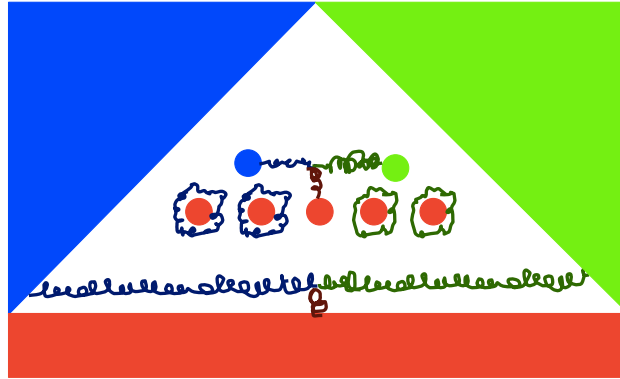


Figure 5.22: Unfortunately, when pulling the operators off the surface boundary with multiplication by stabilizer generators, the presence of other defects causes the strings to get “snagged.” Since these other qubits cannot all be gauged away, it is clear that this procedure will fail in general by introducing unintended logical errors.

in a fixed, known state. This would amount to an entirely trivial computation. This demonstrates that there is a fundamental topological obstruction—the existence of a larger homology group than was considered at the start—that disallows the on-demand conversion from single to triple defects without teleportation.

5.3 Conclusion

The work presented in this chapter is arguably more mathematical than physical, but it does provide good intuition for physical processes related to single-defect encodings. For example, the inability to convert a single defect to a triple defect without using teleportation means that it is likely a waste of time to try and find protocols for applying S_L and H_L to a single defect qubit without first creating a triple defect and teleporting the quantum state into it. However, it is not a proof of impossibility, and more clever techniques may still exist.

Chapter 6

Adiabatic Topological Quantum Computation

This chapter describes joint work performed with Dave Bacon, Steve Flammia, Andrew Landahl, and Alice Neels, the point of which is to present a model of quantum computation that utilizes adiabatic interpolations between static Hamiltonians for as many procedures as possible. Information encoded into a Hamiltonian with a toric code codespace as its groundspace will have a lifetime that is exponential in the inverse temperature of a thermal environment, but its lifetime will not grow with system size. We studied this model to bring together ideas from several different models of quantum computing, including the adiabatic model, the holonomic model, and the topological model. This work can be seen as an explicit analysis of general schemes presented in Refs. [OBL09] and [ZB14], as well as an extension of the work by two of my coauthors on adiabatic code deformation [BF09].

The contents of this chapter have been around in various forms since 2009, and the writing was contributed to by all my coauthors. However, this version is my own final edit. Additionally, my technical contributions to this project include the careful

analysis of state injection and logical measurement, as well as the extension to the color codes in Sec. 6.7.

6.1 Introduction

There are many approaches to constructing a quantum computer. In addition to the numerous different physical substrates available, there are a plethora of different underlying computational architectures from which to choose. Two major classes of architectures can be distinguished: those requiring a substantial external active classical control system to suppress errors [Sho95, Ste96a, Pre98a], and those whose underlying physics eliminates much, if not all, of the need for such a control system [Kit03, DKLP02, FGGS00]. Here we focus on the latter class of architectures and address the question: “How does one quantum compute on a system protected from decoherence by a static (*i.e.*, time-independent) Hamiltonian?” We present a solution that adiabatically interpolates between static Hamiltonians, each of which protects the quantum information stored in its ground space. Since each of these ground spaces can be described as a quantum error-correcting codespace, we call this process *adiabatic code deformation* [OBL09, BF09]. This procedure amounts to a simulation of the measurement-based process of code deformation employed in the first class of architectures [DKLP02, RHG06, BMD09, Bon13]. We further show that this procedure preserves the energy gap of the system throughout the evolution.

While previous work has made reference to adiabatic evolutions as a method for performing topological quantum computation [NSS⁺08], our work can be seen as making the assumptions of adiabatic evolution explicit for certain models of topological quantum computers. In contrast, for example, to topological quantum computing in fractional quantum Hall systems where even the ground state of the system is subject to debate, our models are exactly solvable and simple. Similar work has

been performed for Kitaev’s honeycomb model by Lahtinen and Pachos [LP09], who examined the adiabatic transport of vortices in Kitaev’s honeycomb lattice model numerically. Here, we are able to investigate these issues analytically.

Our results marry three different lines of research, which we now describe. The first is the idea originated by Kitaev [Kit03] that quantum information can be protected from decoherence by encoding into the degenerate ground space of a many-body quantum system. In particular, Kitaev suggested a family of systems such that each system has a ground space equivalent to a quantum error-correcting codespace. Moreover each of these ground spaces is separated from their first-excited space by an energy gap—a gap which does not shrink with the system size (*i.e.*, the gap is “constant”). In Kitaev’s original construction, the quantum error-correcting code also possesses a topological property that makes the distance of the code grow with the number of qubits in the system. This implies that any local perturbing interaction will only split the energy of a degenerate ground state by an exponentially small amount in the size of the system [BHM10]. Information encoded into the ground space should therefore remain well-protected from the detrimental effects of decoherence. Further, if one immerses the system in a bath with a temperature lower than that of the energy gap in the system, then one should expect a suppression of thermal excitations out of the ground space. The decay rate of the quantum information encoded into the ground space is *not* set by a length scale in the system, but instead the lifetime scales as $\exp(c\beta\Delta)$ where β is the inverse temperature, Δ is the energy gap of the Hamiltonian, and c is a constant [AFH09]. Crucially, this implies that the lifetime of the information is exponentially lengthened as a function of the inverse temperature. While one does not obtain, using Kitaev’s original idea, a method for protecting quantum information with a lifetime that grows with the size of the system—a hallmark of “self-correcting” quantum memories [DKLP02, Bac06]—for a suitably low temperature, the information lifetime will be long enough for all practical purposes. Thus, via the use of a static many-body Hamiltonian, Kitaev proposed that quantum infor-

Chapter 6. Adiabatic Topological Quantum Computation

mation could be protected without resorting to active quantum error-correcting algorithms. Following Kitaev’s introduction of this idea, numerous authors put forward similar approaches. Many of these ideas stayed within the realm of topological protection [Fre03, FNS05, BW03, Kit06, FNW06, NSS⁺08], but others explored energetic protection without reference to topological ideas [BW00, BBW01, WH05, Bac08]. Here we will focus on the topological models, but many of our results apply in the more general setting.

Kitaev noted in his original proposal that the excited states of his Hamiltonian act as particles with exotic statistics. In particular, he showed that the excitations were quasiparticles called *anyons* [Wil82]—particles that exist in two spatial dimensions that exhibit statistics different from fermions and bosons and which interact by braiding around one another in spacetime—an interaction that only depends on the topology of the anyon worldlines. These excitations not only describe errors in the codespace but can also be thought of as quantum information carriers in their own right. Indeed for some many-body Hamiltonians, it is possible to have *nonabelian* anyons (anyons whose braidings do not commute) that perform universal quantum computation in the label space of the anyons. This is known as *topological quantum computing* [Kit03, FKW02, FLW02, KKR10], the principal model of quantum computing we will consider here.

In a topological quantum computation, one creates anyons from the vacuum, braids them around one another in spacetime, fuses them together, then records their label types. Although the topological nature of the anyonic interaction provides a degree of control robustness, it is not immediately clear why the processes of anyon creation and fusion could not create new unwanted anyons. Such anyons could in turn wander and disrupt the desired braid. The initialization process in particular is quite subtle [Kön10]. Moreover, there will likely be a background of thermal anyons and anyons arising from material defects which could also disor-

der the quantum computation. On top of all of this, even if a spacetime braid is topologically correct, the mere act of moving anyons around—even adiabatically—has the potential to generate new excitations because the adiabatic approximation is not exact. Measurement-based topological quantum computation [BFN08, BFN09] has the potential to overcome this last problem, but the other problems remain. In summary, the great merit of topological quantum computation is that the “only” thing that can corrupt it is uncontrolled anyons—the problem is that there are many ways that uncontrolled anyons can arise. Even something as seemingly innocuous as a lack of complete knowledge of the system’s Hamiltonian could do this because it could lead to anyons being trapped or leaking out of the system unbeknownst to the computer operator [NSS⁺08]. We do not claim to address every possible adversarial scenario for topological quantum computation here; our focus is on constructing an architecture which limits the chances for uncontrolled anyons to appear.

The second line of research relevant to our proposal is the recent use of code deformations to perform quantum computation on topological quantum error-correcting codes [DKLP02, RHG06, BMD09, KKR10]. In this approach, one works directly with the quantum-error correcting code used in topological quantum computing without introducing a Hamiltonian to provide energetic protection of the quantum information. Instead, one focuses on active error correction, but performed with the topological quantum codes. Consideration of such codes for quantum error correction was first examined in detail by Dennis *et al.* [DKLP02]. In this approach, qubits are arranged on a two-dimensional surface with a boundary, resulting in a single encoded qubit for each such surface. In order to build a quantum computer with more than one qubit, such surfaces are stacked on top of each other so that transversal gates can be achieved between the neighboring surfaces. Since the original analysis, modifications [RHG07, BMD09] of this architecture have been introduced which have considerable advantages over the three-dimensional stacking of Dennis *et al.* In these models, one takes a surface code and “punctures” it by removing the quantum

check operators (stabilizer generators) from a region, creating a *defect* [BK98]. For each defect one obtains an encoded qubit with a code distance that is the minimum of the perimeter of the defect and the distance from the defect to the nearest appropriate boundary (which may lie on another defect). One can show that, via a sequence of adaptive measurements, one can *deform* the boundary of the defect, and, by using suitable deformations, braid defects in such a way that logical operations are performed between the logical qubits associated with the defects.

The third line of research relevant to our proposal is the recent discovery of methods to perform holonomic [ZR99] and open-loop holonomic [KÅS06] universal quantum computation in a stabilizer code setting [OBL09, BF09, Ore09]. In holonomic quantum computing, adiabatic changes of a Hamiltonian with degenerate energy levels around a loop in parameter space induce unitary gates on each energy eigenspace. The enacted gate depends on geometric properties of the Hamiltonian path and not on the exact timing used to traverse it (to within the limits of the adiabatic approximation), thus offering a method to avoid some timing errors. Universal quantum computation using holonomic methods was originally studied in Ref. [ZR99]. Recently, Oreshkov *et al.* demonstrated a novel manner for achieving universality within the context of fault-tolerant quantum computing [OBL09]. In particular, this result showed how to perform gates on information encoded into a quantum stabilizer code. Building along these lines, two of the present authors (DB and STF) have shown how to achieve similar constructions within the context of open-loop holonomic quantum computation [BF09, BF10]. In this setting, instead of using cyclic evolutions, one can quantum compute using non-cyclic evolutions. A consequence of this is a scheme known as *adiabatic gate teleportation* where one mimics gate teleportation via a very simple interpolation between two-qubit interactions [BF09]. Another consequence is that it is possible to perform measurement-based quantum computing [RB01] using only adiabatic deformations of a Hamiltonian [BF10]. Holonomic quantum computation, whether performed

Chapter 6. Adiabatic Topological Quantum Computation

cyclically or non-cyclically, should be distinguished from (universal) adiabatic quantum computation, in which the ground state is always nondegenerate throughout the non-cyclic adiabatic evolution [FGG⁺01, AvK⁺04, OT08, MLM07, KKR06].

Here we combine many of the above insights into a new method for computing on information encoded into the energy levels of a Hamiltonian. We consider a situation where, as in the first line of research, quantum information is encoded into the ground state of a topologically ordered many-body system. Rather than storing information in the label space of anyons themselves, we consider information stored in defects, which act somewhat like anyons, as in the second line of research. Finally, we examine explicit adiabatic interpolations between Hamiltonians that simulate code deformation, as in the third line of research. This is all done while keeping the energy gap in the system constant, a necessary requirement to use these techniques to maintain the topological protection offered by these systems. Further, we demonstrate how to prepare quantum information into fiducial states using adiabatic evolutions. Some of these state-preparation procedures are robust to error, but some (*e.g.*, the preparation of certain “magic states” [BK05]) are not robust and thus require distillation protocols. Finally we discuss how one can use code deformations to facilitate measurements of certain logical operators. We discuss all of these procedures first within the context of Kitaev’s surface codes with defects, and then we discuss how these results can be extended to the topological color codes [BMD06].

The systems and protocols we use are not strictly fault-tolerant. Without active error correction, the lifetime of the codes studied are a constant independent of the system size [AFH09]. As mentioned above, here we rely on a coupling to a cold (with respect to the gap) thermal bath, which suppresses the creation of errors exponentially in the size of the gap. We retain robustness to things like control errors by virtue of the holonomic nature of the logical operations we implement, and robustness to correlated fluctuations induced by the environment by keeping defects

well-separated during braiding. Once the environment creates an excitation, it is free to wander and corrupt the computation. We prevent the environment from doing this by ensuring that it is cold, and we prevent ourselves from introducing excitations accidentally by carefully designing our procedures. Mizel has taken a closer look at the problem of fault-tolerant adiabatic quantum computation in Refs. [Miz10, Miz14], but it remains unclear if these approaches work.

6.2 Surface codes with defects

We begin by working with a simple class of surface codes with defects to establish the main ideas behind our procedures. In Section 6.7 we extend these ideas to the topological color codes. We assume that the reader is familiar with the theory of stabilizer codes [Got97], toric codes [Kit03], and with surface codes [BK98], the specialization of toric codes to bounded planar surfaces. However, we review these results to set our notation.

Let \mathcal{L} be a two-dimensional square lattice that is l edges wide and l edges tall, with the leftmost l vertical edges and bottommost l horizontal edges removed. (Other lattices are possible; we make this restriction only to be concrete.) We call the sides of the lattice with the edges removed the *rough* or *X-type* boundaries and the other sides the *smooth* or *Z-type* boundaries; see Fig. 6.2. (The reason for having more than one name for each kind of boundary is that the pictographic mnemonics of “rough” and “smooth” do not persist for other topological codes, like the color codes, but the distinctions of X-type and Z-type do.) A qubit is associated with each edge of the lattice so that there are $2l^2$ qubits in total. For each plaquette (or face), p , of the lattice, define the *plaquette* operator $S_p = \bigotimes_{e \in \partial p} Z_e$ where ∂p denotes the edges bounding the plaquette and Z_e is the Pauli Z operator acting on the qubit at edge e . In other words S_p acts as the tensor product of Z operators on the qubits touching the

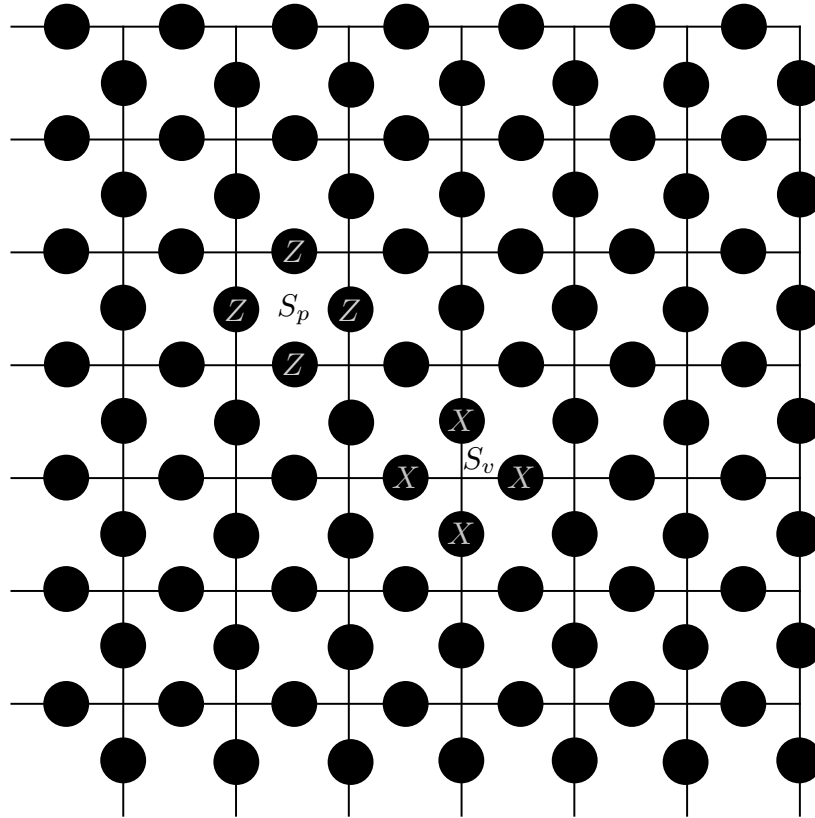


Figure 6.1: Stabilizer generators (checks) for the surface code. An example of a plaquette check S_p and a vertex check S_v .

plaquette p and acts trivially everywhere else in the lattice (see Fig. 6.1). Similarly, for each vertex in the lattice, define a *vertex* operator $S_v = \bigotimes_{e \in \delta v} X_e$, where δv denotes the edges incident at vertex v and X_e is the Pauli X operator acting on the qubit at edge e . In other words, S_v acts as a tensor product of Pauli X operators on all the edges surrounding a vertex and acts trivially on all the other qubits in the lattice, as shown in Fig. 6.1.

It is important to note that the rough and smooth boundaries still have plaquette and vertex operators defined; these operators simply act nontrivially on fewer qubits than the operators in the bulk of the lattice. Since the lattice \mathcal{L} has l^2 plaquettes

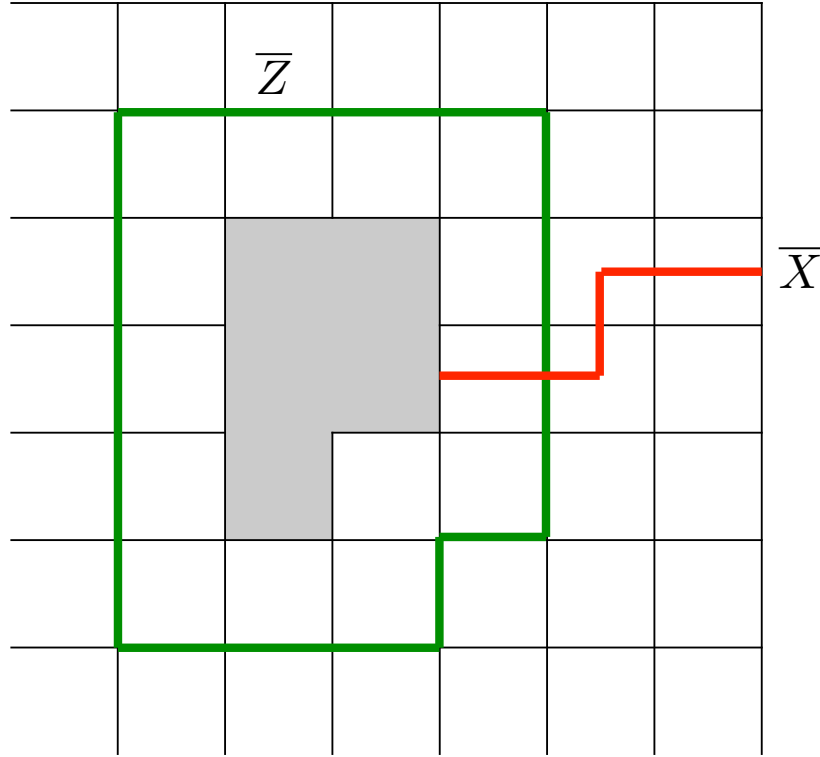


Figure 6.2: A smooth (Z -type) defect. A logical Z operator is defined by a closed loop of Z s on the lattice that surrounds the defect and a logical X operator is defined by a connected path of X s on the dual lattice from the defect to a *smooth* (Z -type) boundary. Here we depict the removed region by removing that part of the lattice; this simply indicates that the code of the system factors into a code in the drawn region and a code inside of the defect.

and l^2 vertices, there are also l^2 plaquette operators and l^2 vertex operators. These operators are all independent in the sense that no strict subset can generate the rest, and, moreover, they all commute since they are incident on each other an even number of times.

The collection of all the S_p and S_v operators comprises the set of stabilizer generators for a quantum surface code, the codespace being defined by the simultaneous $+1$ eigenspace of all the stabilizer generators. This set generates the stabilizer group

not touch the boundary of \mathcal{L} . Call the interior of this loop, excluding c itself, I_c . Consider “removing” all of the qubits in I_c . Here by “removing” we do not mean physically removing the qubits, but rather that we consider a new code in which the stabilizer generators exterior to the region I_c are consistent with the description above, while the region I_c has a different set of stabilizer generators (not necessarily of the plaquette and vertex type). We call this process *puncturing* (not to be confused with the notion of puncturing associated with classical coding theory [MS77]), and the resulting region of removed qubits is called a *defect*. Given such a defect, we can study the properties of the new code induced on the exterior of I_c . Careful counting of the stabilizer generators and qubits in this new code reveals that the puncturing procedure has created a logical qubit [BK98]. The logical operators for the new logical qubit can be chosen as follows: an encoded Z is a closed loop of Z operators on the lattice \mathcal{L} that encircles the defect and an encoded X is a connected path of X operators on the dual lattice \mathcal{L}^* that starts on the smooth (Z -type) boundary of the defect and ends on a smooth (Z -type) boundary of the lattice \mathcal{L} which is not the loop c (see Fig. 6.2). The distance of this code is the minimum of the length of curves on \mathcal{L} bounding the defect and the length of paths connecting the defect to a smooth (Z -type) boundary of \mathcal{L} . We note that the curve c itself is the minimum weight choice for the encircling logical Z operator. Similarly, instead of starting with a simple closed curve on the lattice, we can consider a simple closed curve on the dual lattice and remove the interior of this curve. To be consistent with the definition given for the former kind of defect, we must define the encoded X to be a closed loop c^* of X operators on the dual lattice \mathcal{L}^* that encircles the defect and the encoded Z to be a connected path of Z operators on the lattice \mathcal{L} that starts on the rough (X -type) boundary of the defect and ends on a rough (X -type) boundary of the lattice \mathcal{L} which is not in the loop c^* (see Fig. 6.3).

Puncturing the surface code creates a single encoded qubit. By puncturing multiple times we can create a code with more than one encoded qubit, one for each

additional puncture. The boundary curves of these defects can be on the lattice, in which case we call the defect *smooth* (*Z-type*), or on the dual lattice, in which case we call the defect *rough* (*X-type*). The distance of such a code is the minimum of the distance between defects, the distance of a defect and the boundary of the lattice, and the circumference of a defect.

Surface codes with defects were first explored within the framework of active quantum error correction. Here we consider an alternative situation in which we construct a Hamiltonian with a ground space that is degenerate and identical to the codespace of a quantum error correcting code. The construction of such a Hamiltonian is easy from a theoretical point of view; it is simply the negative sum of the stabilizer generators, \mathcal{G} ,

$$H = -\frac{\Delta}{2} \sum_{S \in \mathcal{G}} S. \quad (6.1)$$

The constant in front is chosen so that all errors will have an energy penalty of at least Δ (errors adjacent to a boundary will have this penalty, while errors away from boundaries will have a penalty of 2Δ). Since the set of generators is commutative, the eigenspaces of H can be labeled by their eigenvalues with respect to the operators S , and since the eigenvalues of all the S are ± 1 , the ground state of this Hamiltonian is equivalent to the codespace of the quantum code generated by \mathcal{G} : $S|\psi\rangle = |\psi\rangle$ for all $S \in \mathcal{G}$.

Hamiltonians like that in Eq. (6.1), which we call *stabilizer Hamiltonians*, have interesting properties for protecting quantum information. The first property is that operators which act nontrivially on the codespace (the degenerate ground space) must be nonlocal, having a Pauli-weight at least as large as the code's distance. This allows for the system to retain its information even when perturbed by a local Hamiltonian [HL08, BHM10]. For toric codes, surface codes, color codes, and, more generally, codes formed from quantum double models [dB94], this is a partial indication of a topological order in the system. (A more robust indicator would be a

nontrivial topological entanglement entropy [[HIZ05](#), [KP06](#), [LW06](#), [FHHW09](#)].)

While a stabilizer Hamiltonian is robust to local perturbations, if the system is immersed in a thermal bath, the lifetime of information encoded into the ground state does not necessarily scale as the size of the system (or the size of the defect for a surface code with defects). For example, for the toric code, the lifetime of this information is proportional to $\exp(2\beta\Delta)$ [[AFH09](#)], where $\beta = (k_B T)^{-1}$ is the inverse temperature of the bath. It is widely believed that all stabilizer Hamiltonians with local terms embedded in two spatial dimensions have a similar lifetime [[BT09](#)]. The more challenging issue is how to compute with them without increasing the rate at which information is destroyed. As mentioned in Sec. 6.1, if a stabilizer Hamiltonian describes a topologically ordered system possessing anyons that have a sufficiently rich nonabelian structure, then quantum computation can be carried out by creating, braiding, and fusing the anyons. However, it is not entirely clear that one can controllably create single excitations without also creating other uncontrolled excitations that could then disorder the system, nor how one can move the anyons without causing other anyons to be produced. This has led to the search for self-correcting quantum systems where the excitations are not point-like particles like anyons but have boundaries with dimension [[DKLP02](#), [Bac06](#), [BT09](#)]. The energetic cost of an excitation in such a system is proportional to the size of its boundary and thus would be robust to errors during creation and movement processes—such a system would energetically favor shrinking the boundaries of the errors to zero, causing them to vanish. In particular, it has been argued that such systems would have a lifetime proportional to their size, indicating that the system and the environment to which it is coupled participate in a form of “self-correction” in which the environment that creates the errors can also fix the errors; at a low enough temperature, the rate of the latter process dominates the rate of the former. In this paper, we do not directly address the question of self-correction; instead we attempt to better understand how computation can be done adiabatically within existing models.

6.3 Adiabatic code deformations

Before showing how to perform the adiabatic deformations and creation of fiducial states, we briefly review a scheme for performing adiabatic gate teleportation [BF09] (AGT), as this gives an idea of how the protocols we introduce below operate. AGT is a procedure for transferring information in one qubit to information in another qubit (with a possible gate applied to this information) via the use of an adiabatic evolution and an ancillary qubit. This example is on a system composed of three qubits and here we consider the case where no gate is applied during the swapping of the qubit between the first and third qubits. Initially the system evolves under a Hamiltonian given by

$$H_i = -\Delta(I_1 X_2 X_3 + I_1 Z_2 Z_3), \quad (6.2)$$

where P_i represents the operator P acting on the i th qubit and where we soon omit the identity operators I . A final Hamiltonian is defined as

$$H_f = -\Delta(X_1 X_2 I_3 + Z_1 Z_2 I_3). \quad (6.3)$$

The AGT protocol begins with the information encoded in the first qubit and H_i turned on. Then, H_i is adiabatically turned off while simultaneously turning on H_f . In other words, the evolution is described by

$$H(t) = f(t)H_i + g(t)H_f, \quad (6.4)$$

where $f(0) = 1$, $f(T) = 0$, $g(0) = 0$ and $g(T) = 1$ and T is the time taken to perform the evolution. If $f(t)$ and $g(t)$ are chosen to be slowly varying and the time T is long enough such that the evolution is adiabatic (meaning here that the probability of exciting the system out of its ground state is made small), then the above evolution will take information in the first qubit and send it to information in the third qubit. For example, one may choose $f(t) = 1 - g(t)$ and $g(t) = \frac{t}{T}$ so that the evolution is made adiabatic for sufficiently large T . A constant error can be achieved for a fixed constant T .

Chapter 6. Adiabatic Topological Quantum Computation

To see that a constant energy gap is maintained during the above evolution and that the information is transported from the first to third qubit, it is convenient to use the formalism of stabilizer codes to describe this evolution. Indeed, it is actually useful to define three codes. The first code, call it S_1 , is defined by the stabilizer generators X_2X_3 and Z_2Z_3 and the logical Pauli operators $\bar{Z} = Z_1Z_2Z_3$ and $\bar{X} = X_1X_2X_3$. A second code, call it S_2 , is defined by the stabilizer generators X_1X_2 and Z_1Z_2 and the logical Pauli operators $\bar{Z} = Z_1Z_2Z_3$ and $\bar{X} = X_1X_2X_3$. Suppose information is encoded into the stabilizer code S_1 so that it is in the $+1$ eigenstate of both X_2X_3 and Z_2Z_3 . Notice then that because $X_1 = \bar{X}(X_2X_3)$ and $Z_1 = \bar{Z}(Z_2Z_3)$, information encoded into this code can be accessed by making a measurement on the first qubit. Similarly, information encoded into the second code, S_2 , is localized in the third qubit. The adiabatic evolution in Eq. (6.4) can now be seen as adiabatically dragging a Hamiltonian which is a sum over stabilizer generators in S_1 to a sum over stabilizer generators in S_2 such that the information in the encoded qubit described by \bar{X} and \bar{Z} is not touched.

To analyze how the dragging between S_1 and S_2 occurs, it is useful to introduce a new code, S_3 . This code has no non-identity stabilizer operators, but has three encoded qubits. These are defined by

$$\begin{aligned} \bar{X}_1 &= X_1X_2 & \bar{Z}_1 &= Z_2Z_3 \\ \bar{X}_2 &= X_2X_3 & \bar{Z}_2 &= Z_1Z_2 \\ \bar{X}_3 &= X_1X_2X_3 & \bar{Z}_3 &= Z_1Z_2Z_3 \end{aligned} \tag{6.5}$$

Notice that \bar{Z}_1 and \bar{X}_2 are the stabilizer generators of S_1 and \bar{X}_1 and \bar{Z}_2 are the stabilizer generators for S_2 . From this perspective, then, the adiabatic evolution is from the initial Hamiltonian $-\Delta(\bar{Z}_1 + \bar{X}_2)$ to the final Hamiltonian $-\Delta(\bar{X}_1 + \bar{Z}_2)$. These are then simple interpolations between single operators on encoded qubits, and will have a constant energy gap. Indeed both S_1 and S_2 can be turned into S_3 by promoting stabilizer generators in these codes to logical Pauli operators. When

it is possible to perform such a change between codes via an adiabatic evolution we say that we can adiabatically *deform* one code into the other. This technique is at the heart of the constructions in this paper.

To see that the information encoded in the first qubit ends up at the third qubit, first note that, during the above evolution, the third encoded qubit is not involved. This implies that information encoded into this qubit will not be affected by the evolution. Next note that $X_1 = \overline{X}_3 \overline{X}_2$, $Z_1 = \overline{Z}_3 \overline{Z}_1$ and $X_3 = \overline{X}_3 \overline{X}_1$, $Z_3 = \overline{Z}_3 \overline{Z}_2$. Recall that we are dragging between the +1 eigenstate of \overline{X}_2 and \overline{Z}_1 to the +1 eigenstate of \overline{Z}_2 and \overline{X}_1 . Thus, since information encoded into the third qubit is not changed during the above evolution, we see that the protocol transports the information in the first qubit to the third qubit.

More generally, the AGT protocol can be extended to enable universal quantum computation [BF09]. We omit the details of this construction except for noting that even when generalized, the energy gap used to guarantee adiabatic evolution is a *constant* with respect to the number of qubits in the system. We will often refer to this by saying that the energy gap of an adiabatic evolution is *constant* when considered by itself—we use this language merely to imply that stringing together similar parallel evolutions will not shrink the gap as a function of the number of qubits involved in the evolution.

6.4 Adiabatic code deformations of the surface code

With the punctured surface code defined, we now present a series of adiabatic code deformations that allow for a nearly universal set of operations. First, we show how to prepare a surface code without any defects. Next, we show how to prepare smooth defects in the +1 eigenstate of \overline{Z} and rough defects in the +1 eigenstate of \overline{X} . We then show how to prepare smooth defects in ± 1 eigenstates of \overline{X} and rough defects

in ± 1 eigenstates of \bar{Z} . (These procedures prepare the defects in eigenstates of the string-like logical operators that tether the defects to a boundary.) Following this, we introduce a procedure to allow code regions containing defects to be separated from and attached to the rest of the code. We next show how defects can be deformed, allowing them to be moved around the lattice. This additionally allows for the *CNOT* to be enacted between a smooth and a rough defect. Finally, we show how arbitrary ancilla states can be injected into defects and utilized in a computation.

The procedures above can be performed in an entirely adiabatic fashion and thus benefit from the protection of a Hamiltonian gap. Additionally, procedures like defect braiding also benefit from the topological nature of the surface code Hamiltonian, requiring high-weight correlated errors corresponding to nontrivial cycles on the lattice or dual lattice. We mention this now to highlight the difference between the entirely adiabatic operations presented in this section and operations we present in Sec. 6.5—such as measurement or heralded gate application—that do not inherit any protection from the gap or the topology.

6.4.1 Creation of a surface code without defects

We begin by assuming that we have a large array of qubits, shown in Fig. 6.4, stabilized by a Hamiltonian H_i given by

$$H_i = -\Delta \sum_j Z_j, \quad (6.6)$$

where the sum runs over all the qubits. The ground state of this Hamiltonian is unique and has all the qubits in the state $|0\rangle$. To prepare the surface, standard active error correction techniques call for the stabilizer generators to be measured. Here, we simulate these measurements in the vein of the “forced measurements” introduced in Ref. [Bon13] by slowly turning off H_i and turning on the Hamiltonian introduced in Eq. 6.1 for the specific instance of a “small” surface code. Turning on a

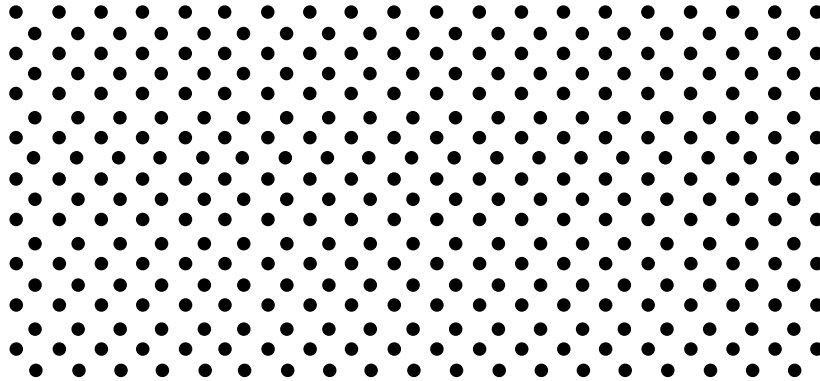


Figure 6.4: A large array of qubits in the state $|0\rangle$, each protected by a Hamiltonian $H = -\Delta Z$.

Hamiltonian with a “large” surface code as the ground state would cause the system gap to shrink proportional to the size of the code, so to be concrete we choose to evolve initially to a Hamiltonian with a small surface code ground state. (We will subsequently show how its size can be sequentially increased.) In other words, we adiabatically follow the Hamiltonian

$$H(t) = \left(1 - \frac{t}{T}\right) \sum_{j \in \mathcal{Q}} (-\Delta Z_j) + \frac{t}{T} \sum_{S \in \mathcal{G}} \left(-\frac{\Delta}{2} S\right) + \frac{t}{T} \sum_{j \notin \mathcal{Q}} (-\Delta Z_j), \quad (6.7)$$

where \mathcal{Q} is the set of qubits participating in the surface code terms. In this case, \mathcal{G} has 8 elements, the four plaquette operators and the four vertex operators shown in Fig. 6.5. Provided T is large, the system will remain in the ground state. As we showed before, the ground state of the Hamiltonian in Eq. 6.1 is the codespace if a surface code. We choose it to be nondegenerate by our choice of boundaries, although this is not a necessity. After the evolution, the array of qubits looks like Fig. 6.5.

Having created a small surface code that encodes no qubits, we can increase its size by modifying the boundaries adiabatically. For example, we can grow out part

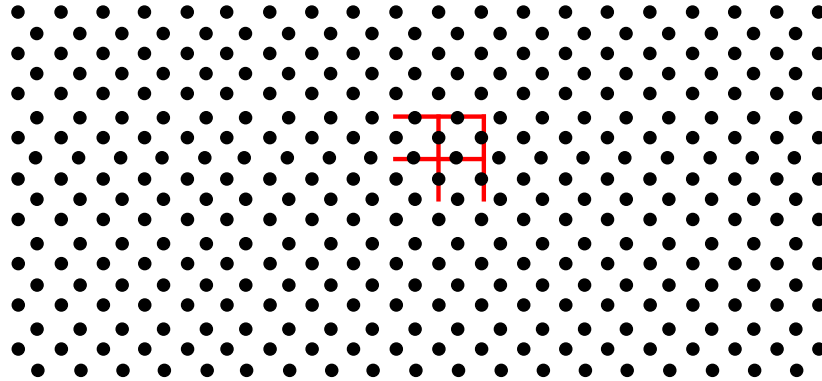


Figure 6.5: A large array of qubits, an 8-qubit region of which is now encoded in the surface code (shown in red). The boundaries of the code are chosen to be trivial so that the codespace is nondegenerate.

of the smooth boundary by performing an evolution of the form

$$H(t) = \left(1 - \frac{t}{T}\right) (-\Delta Z_1 - \Delta Z_2 - \Delta Z_3) + \frac{t}{T} \left(-\frac{\Delta}{2} Z_1 Z_2 Z_3 - \frac{\Delta}{2} X_1 X_2 - \frac{\Delta}{2} X_2 X_3\right),$$

where the numbering corresponds to Fig. 6.6. This also requires that the modification

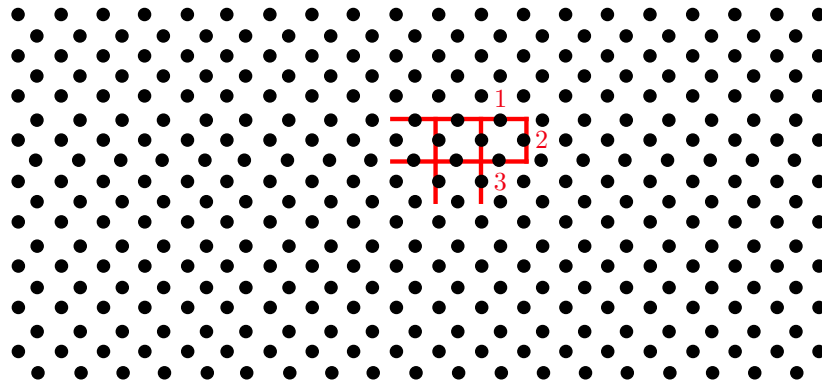


Figure 6.6: Growth of a small surface code region that involves only the qubits labeled 1, 2, and 3.

of vertex checks on the smooth boundary being extended, which can be performed

at the same time. Additionally, a similar procedure will allow the extension of rough boundaries. By piecing these additional evolutions together, a larger surface code region can be constructed while maintaining a Hamiltonian gap that is lower bounded by a constant proportional to Δ .

For the remainder of this section, we will specialize our figures so that they do not include the black dots that represent qubits, instead keeping only the underlying square lattice structure of the code. However, the full plane of qubits is still assumed to exist.

6.4.2 Creation of a small $Z(X)$ defect in a $+1$ eigenstate of $Z(X)$

Here we describe how to create a two-plquette smooth defect in an unpunctured surface code. (The creation of a rough defect will proceed in an exactly analogous way with the roles of Z and X interchanged.) We create defects using two neighboring plaquettes for pedagogical clarity, although creating single defects is also possible. With two-plaquette defects, it is obvious that the creation process inherits protection from a Hamiltonian gap and the topological nature of logical operators; for single-plaquette defects, the Hamiltonian gap protection is not present.

To begin the creation procedure, the Hamiltonian is initially given by Eq. 6.1, the negative sum of all the plaquette and vertex stabilizer generators for the code. The defect will consist of two adjacent plaquettes, bounded by a curve c that encloses these plaquettes. If the stabilizer generators associated to these two plaquettes are S_{p_1} and S_{p_2} , we can promote them to \overline{Z} operators for two encoded qubits— \overline{Z}_{p_1} and \overline{Z}_{p_2} respectively—of a new code where the stabilizer generators S_{p_1} and S_{p_2} have been removed. If we do this, then \overline{X} for each qubit can be chosen as a string of Pauli X operators beginning on the appropriate plaquette, traversing the dual lattice, and

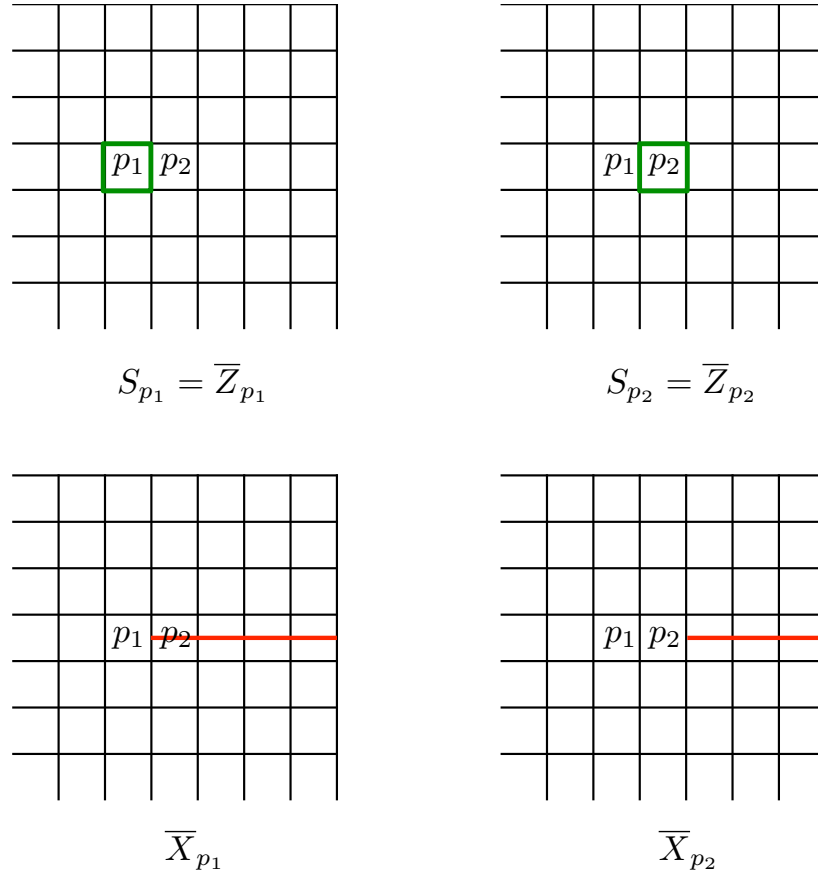


Figure 6.7: Operators involved in creating the defect which includes p_1 and p_2 . Note that the X operations span to a nearby smooth boundary.

ending on a smooth boundary (see Figure 6.7). In fact, we can always choose these operators so that they overlap on all but the qubit separating the two plaquettes. We call these two encoded logical X operators \bar{X}_{p_1} and \bar{X}_{p_2} . The operator $\bar{X}_{p_1}\bar{X}_{p_2}$ is then the single Pauli X operator acting on the qubit between the plaquettes.

Suppose that we now perform the following adiabatic evolution: while turning off the two plaquette operators, S_{p_1} and S_{p_2} in the Hamiltonian, we simultaneously turn on the Pauli X operator between these two plaquettes. In terms of the encoded logical

Chapter 6. Adiabatic Topological Quantum Computation

operators we have defined above, this is equivalent to starting with the Hamiltonian

$$H_i = -\frac{\Delta}{2}(S_{p_1} + S_{p_2}) = -\frac{\Delta}{2}(\bar{Z}_{p_1} + \bar{Z}_{p_2}) \quad (6.8)$$

and ending with the Hamiltonian

$$H_f = -\frac{\Delta}{2}\bar{X}_{p_1}\bar{X}_{p_2} \quad (6.9)$$

All the other terms in the Hamiltonian commute with the relevant operators and therefore do not contribute to any spectral shifts which might cause crossings.

In order to understand what happens in interpolating between H_i and H_f , it is convenient to note that $\bar{Z}_{p_1}\bar{Z}_{p_2}$ (which is a closed loop of Pauli Z operators surrounding the smooth defect we are creating) commutes with these Hamiltonians. Also note that initially the system is in the $+1$ eigenstate of both \bar{Z}_{p_1} and \bar{Z}_{p_2} , and hence also in the $+1$ eigenstate of $\bar{Z}_{p_1}\bar{Z}_{p_2}$. Because $\bar{Z}_{p_1}\bar{Z}_{p_2}$ commutes with both H_i and H_f , we may work in a basis in which $\bar{Z}_{p_1}\bar{Z}_{p_2}$ and the full Hamiltonian are simultaneously diagonal. This commutativity ensures that the eigenvalue of $\bar{Z}_{p_1}\bar{Z}_{p_2}$ is conserved throughout the evolution. If we perform this evolution via a simple adiabatic dragging between these Hamiltonians (as described in Section 6.3) then the energy gap in the system during this evolution remains constant. At the end of the evolution, the system is in the $+1$ eigenstate of both $\bar{Z}_{p_1}\bar{Z}_{p_2}$ and $\bar{X}_{p_1}\bar{X}_{p_2}$, which is simply a single Pauli X on the qubit between the plaquettes.

The above can be interpreted in terms of codes. By turning off two stabilizer generators and turning on only a single Pauli X , we have introduced an encoded qubit by decreasing the number generators. The product of the two missing plaquette checks is \bar{Z} , and either \bar{X}_{p_1} or \bar{X}_{p_2} can be chosen as \bar{X} . Additionally, because the operator \bar{Z} commuted with the Hamiltonian throughout the adiabatic evolution, the encoded qubit is prepared in the $+1$ eigenstate of \bar{Z} .

After this adiabatic evolution, the Hamiltonian does not quite factor into two separate codes on the interior and exterior of the defect. The vertex operators adja-

Chapter 6. *Adiabatic Topological Quantum Computation*

cent to the defect region still check the single qubit on the interior. As a generating set, the four-body checks adjacent to the defect and the single-body “check” on the interior qubit can equally well be thought of as a generating set with two three-body operators that do not act on the interior qubit, and the single-body operator that does. However, in the Hamiltonian framework we must explicitly remove support of these four-body checks on the interior qubits. We do this either by including the modification of the adjacent vertex checks in the evolution discussed above, or by using another evolution afterward that performs the modification. We will assume that the former modification is used.

We note at this point that, while the defect we’ve created is small and thus susceptible to relatively low-weight loops of Z errors, these errors actually have no effect. Since \bar{Z} acts trivially on the state we’ve prepared—namely, $|\bar{0}\rangle$ —the fact that the defect has a small perimeter is not detrimental. Once we start performing gates that change the state, we will have to make sure that the perimeter is large, and that the defect is far from the boundaries and other defects.

As mentioned above, the same arguments can be made for preparing rough defects in the $+1$ eigenstate of \bar{X} . In that case, two adjoining vertex checks are turned off while a single-body Z on the qubit in the middle is turned on. Two adjacent four-body Z checks have to be modified in this case, but the arguments are exactly the same as above.

It might be useful to address a question that may have entered the reader’s head. The procedures above adiabatically interpolate between a Hamiltonian with a nondegenerate groundspace to a Hamiltonian with a degenerate groundspace. Isn’t there a level crossing between the groundspace and an excited space that can cause transitions away from the state we want to prepare? Protection from this coupling is provided by the topological nature of the logical operators. The only operator that can couple $|\bar{0}\rangle$ and $|\bar{1}\rangle$ for a smooth defect is the string-like operator \bar{X} that connects

the defect to a boundary. This amounts to another way of saying that the eigenvalue of the operator \bar{Z} is a conserved quantity throughout the evolution, and so such a crossing is not meaningful.

Now that we've introduced a method for creating smooth defects in the $+1$ eigenstate of \bar{Z} and rough defects in the $+1$ eigenstate of \bar{X} , we'd like to show how these defects can be grown and moved around the lattice. This will allow us to introduce other procedures, such as the isolation a defect from the bulk of the code and an adiabatic code deformation that performs the *CNOT*.

6.4.3 Adiabatic deformation of defects

We now show how to deform a defect. This involves modifying the Hamiltonian by adding or removing stabilizer generators, the combination of which allows defects to be moved.

Consider a smooth defect that we wish to grow by turning off a single adjacent plaquette check in the bulk of the system. The number of edges bordering the interior of the defect is either 1, 2, 3, or 4, as shown in Fig. 6.8. The procedure in each case is basically the same, with the clean-up or potential removal of the adjacent vertex checks being the only difference. The growth is achieved by turning off the plaquette check in the Hamiltonian and turning on a single-qubit $-\frac{\Delta}{2}X$ Hamiltonian for each qubit in the interior after the evolution. We also modify any adjacent vertex checks at the same time to make the code properly factor into an interior and an exterior. We will briefly analyze the different interior edge cases.

For a single interior edge, as shown in Fig. 6.9, there is not much different with respect to the case of defect creation. As the plaquette check to grow into is turned off, a single-body X on the qubit adjacent to the defect and the plaquette is turned on. To fully sever the interior and exterior regions, the only thing left to do is modify

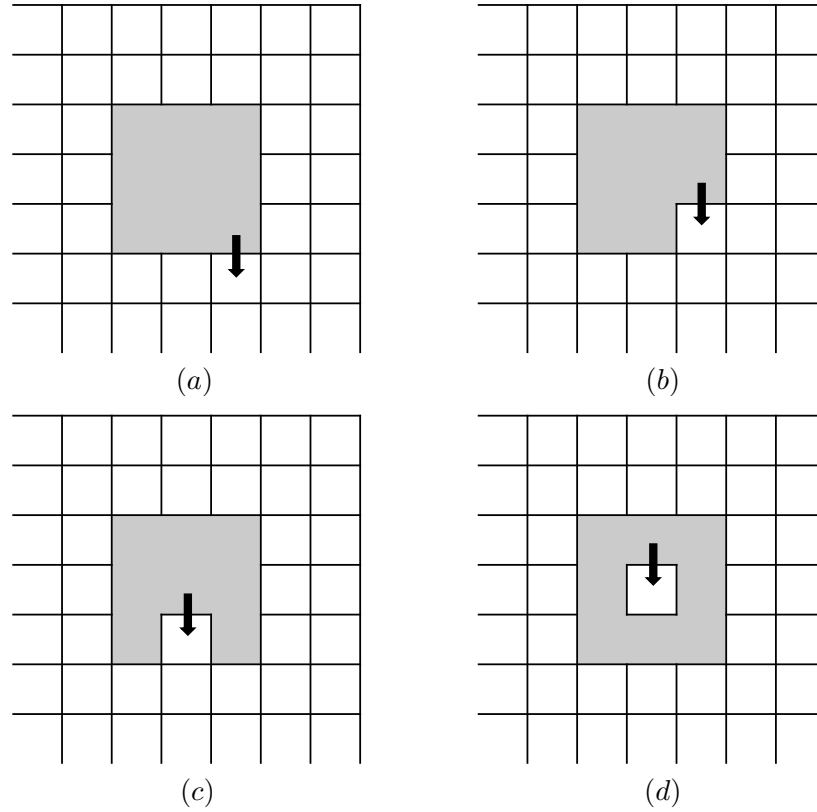


Figure 6.8: The four potential situations faced when growing a smooth defect. (a) Only one interior qubit. (b) Two interior qubits. (c) Three interior qubits. (d) Four interior qubits.

the two adjacent vertex checks from three-body operators to two-body operators.

The cases of 2, 3, and 4 interior edges are different in that some vertex checks are not only modified but turned off completely. For the case of 2 interior edges, as shown in Fig. 6.10, the appropriate evolution turns off the plaquette check while turning on two single-body X Hamiltonians on the interior edges. Note that the two-body vertex check that operated on both the interior qubits is now redundant in terms of stabilizer generators: it is simply the product of the two single-body X terms that were turned on. As such, it can simply be turned off without having to worry

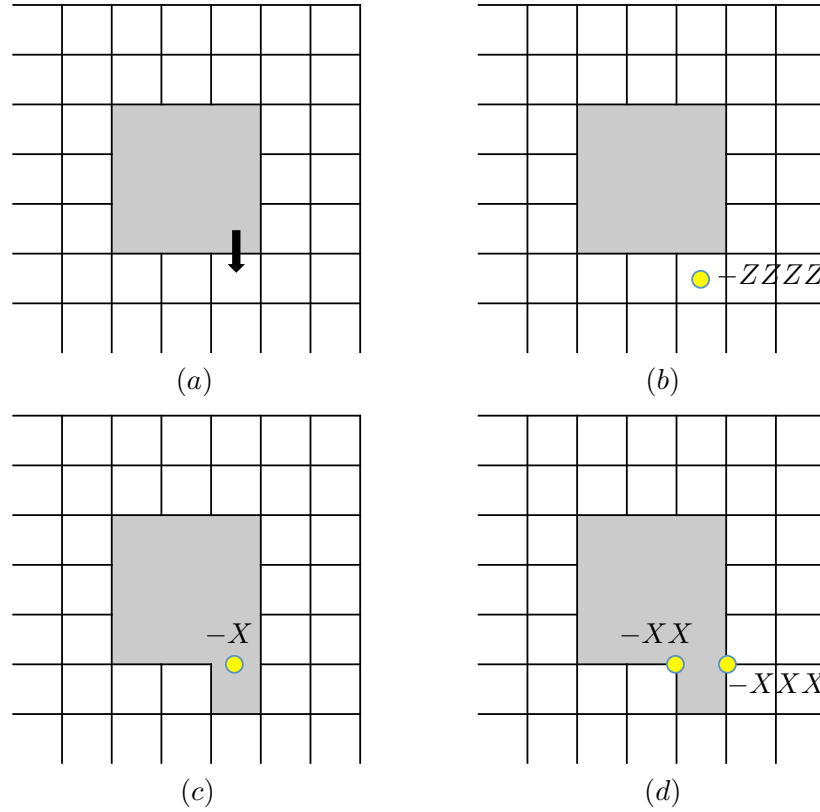


Figure 6.9: Growth of a smooth defect with only a single qubit on the interior after the procedure. (a) We wish to grow the defect to the indicated plaquette. (b) We adiabatically turn off the neighboring plaquette while (c) turning on a $-X$ Hamiltonian on the interior qubit. (d) This procedure causes modifications to the neighboring X checks which can be performed simultaneously with steps (b) and (c).

about the codespace being affected; it merely provides an additional energy penalty for errors on the two interior qubits. The result is that we've removed two stabilizer generators—the plaquette check and the two-body vertex check—and added two stabilizer generators—the two single-body X operators. Thus, we haven't added any additional logical qubits, merely grown the perimeter of an existing one. As a final note, the two adjacent four-body vertex checks also must be modified to three-body checks, and again, this can happen simultaneously with the other adiabatic

evolutions. The case of 3 and 4 interior qubits, shown in Fig. 6.11 and Fig. 6.12

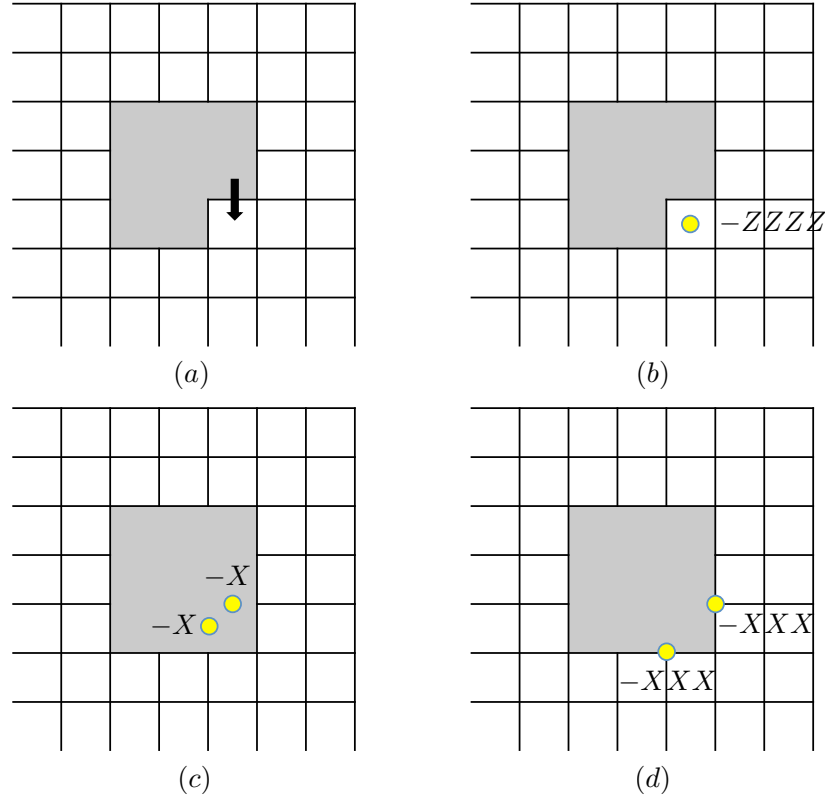


Figure 6.10: Growth of a smooth defect with two qubits on the interior. (a) We wish to grow the defect to the indicated plaquette. (b) We adiabatically turn off the neighboring plaquette while (c) turning on two $-X$ Hamiltonians on the interior qubits. (d) This procedure causes modifications to the neighboring X checks.

respectively, is almost identical. For the case of 3, the plaquette check is turned off while three single-body X Hamiltonians are turned on. In this case, two weight-two vertex checks are now redundant, and as before they can simply be turned off without worrying about level crossings. The counting works in a similar way, in that we've removed three stabilizer generators and added three, preserving the number of logical qubits. The two adjacent weight-four vertex checks also get modified to weight-three operators. Finally, in the case of 4 interior qubits, the same adiabatic deformation is

performed: the plaquette check is turned off and four single-body X Hamiltonians are turned on. Only three of the two-body vertex checks are independent, and so only those three appeared in the original Hamiltonian. They are the three checks made redundant by the single-body X Hamiltonians in this case. Unlike the other cases, in this case there are no other vertex checks that need to be modified.

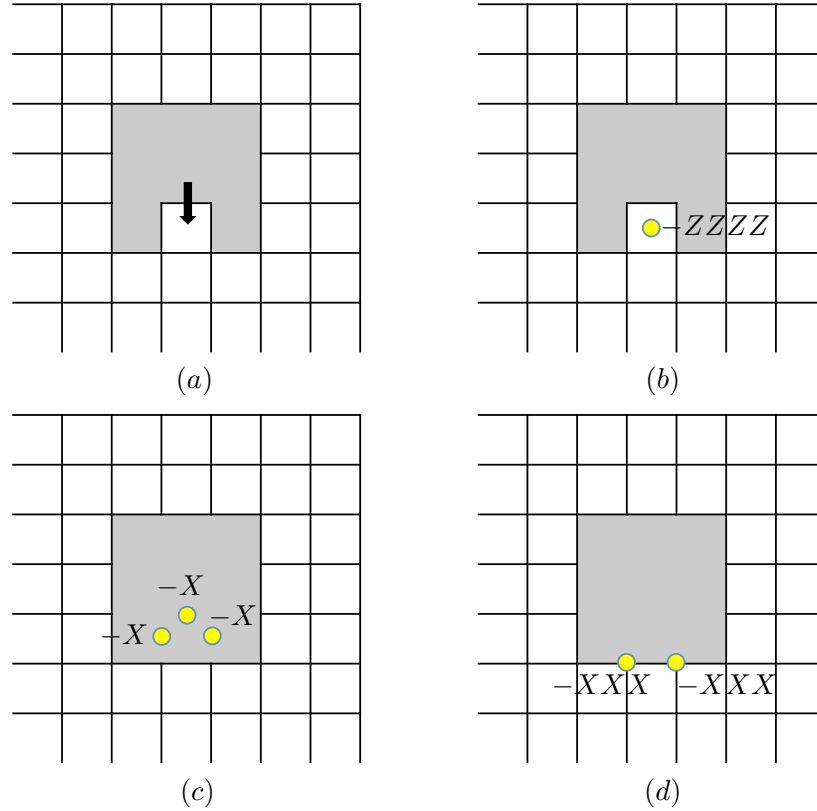


Figure 6.11: Growth of a smooth defect with three qubits on the interior. The process is essentially the same as the one depicted in Fig. 6.10.

The procedure for shrinking defects is simply the inverse of the procedures introduced above. By combining the “grow” and “shrink” operations, we can move defects. As demonstrated in Ref. [RH07], an encoded $CNOT$ gate can be performed by moving a smooth defect in a full loop around a rough defect. The smooth defect is the

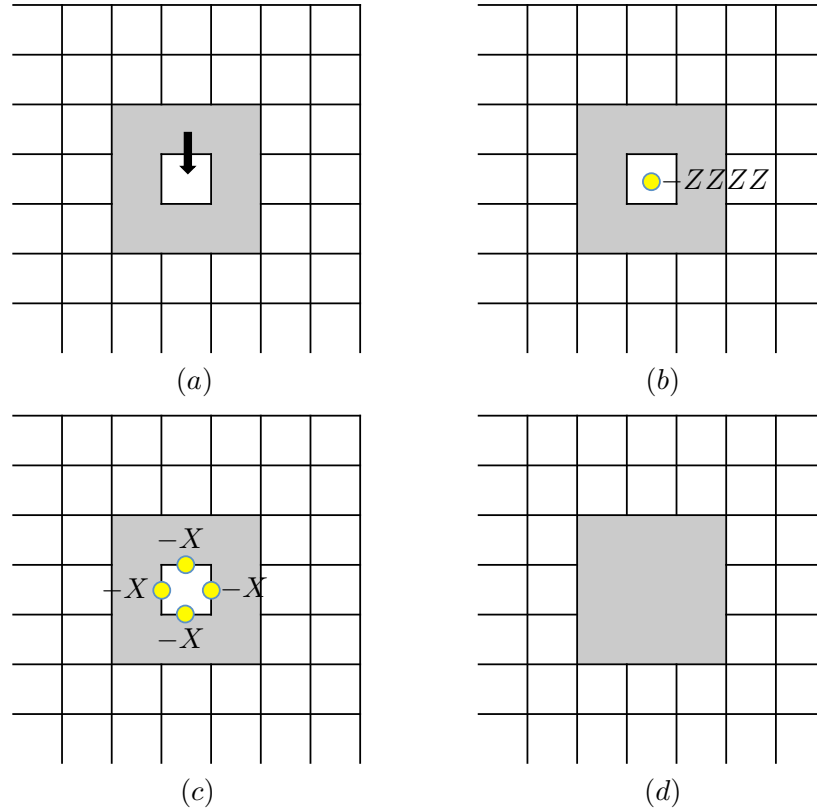


Figure 6.12: Growth of a smooth defect with four qubits on the interior. The procedure is the same as the others, but there are no resulting X check modifications.

control and the rough defect is the target, and the direction of movement—clockwise or counterclockwise—is unimportant.

6.4.4 Detaching and attaching surface code regions with defects

For some subsequent procedures we will consider, it is helpful to have an operation that isolates a defect from the surface code or reintroduces a defect to the surface code that was previously isolated. By using defect creation and growth operations

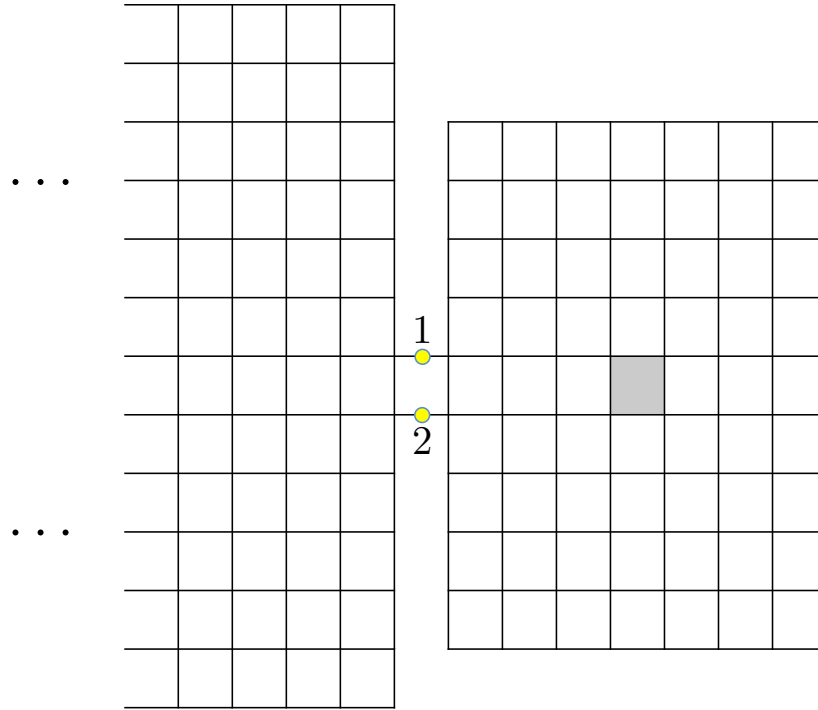


Figure 6.13: The setup for pinching off a smooth defect from a smooth wall.

described in Secs. 6.4.2 and 6.4.3, we can grow a defect “moat” around a defect of interest so that the “castle” surrounding the defect has just a single “drawbridge” connecting it to the rest of the surface, as depicted in Fig. 6.13. The only additional operation we must consider to complete the isolation procedure is how to “lift the drawbridge” by modifying the remaining check operators adjacent to it. As before, we will only consider the case of manipulating smooth defects—the case for rough defects is similar.

To isolate smooth defect, we must use smooth boundaries on the “castle” to ensure that \overline{X} for the defect will have a place to terminate once the “drawbridge” is lifted. For concreteness, we assume that this smooth boundary corresponds to the large boundary of the surface, but the same procedure could be performed using a defect to create the isolated region.

To remove the “drawbridge,” we simply turn off the single plaquette check that connects the two regions while turning on a single-body X on each of the two qubits that need to be removed. (These qubits are labeled 1 and 2 in Fig. 6.13.) The operator X_1X_2 , which was an element of the stabilizer group before the evolution, is now redundant, just as in the case of the interior checks that appear during defect growth in Sec. 6.4.3, and it is also removed. Thus, we remove two checks—the check associated with the “drawbridge” and the two-body check X_1X_2 —and replace them with two single-body X checks in the Hamiltonian. As before, the vertex checks adjacent to the “drawbridge” must be modified, and in this case they become three-body operators. (As a closing aside, if we had tried to detach a smooth defect through a rough boundary, the operator X_1X_2 would no longer have been an element of the stabilizer group.)

Reversing the detachment procedure allows regions with defects to be attached to the surface, introducing (or reintroducing) isolated defects back into the code. This attachment procedure is an important step in our protocols for making measurements of \overline{X} and \overline{Z} and injecting ancilla states into the system, as discussed in Sec. 6.5.1 and Sec. 6.4.6. Additionally, we mention here that it is also possible to isolate and reintroduce a rough defect through a rough boundary in an analogous fashion.

6.4.5 Creation of a $X(Z)$ defect in a ± 1 eigenstate of $Z(X)$

Another capability that will be useful for later procedures is the ability to prepare rough defects in an eigenstate of \overline{Z} and smooth defects in eigenstate of \overline{X} . The preparation of these defects is performed in a region that is disconnected from the main surface, and it is then attached to the surface using the procedure described in Sec. 6.4.4 to introduce it to the bulk surface.

To prepare a rough qubit in the $+1$ eigenstate of \overline{Z} , we utilize a procedure very

Chapter 6. Adiabatic Topological Quantum Computation

similar to the original creation of the surface, described in Sec. 6.4.1. Recall that the stabilizer Hamiltonian on a region disconnected from the surface is simply a sum of single-body $-Z$ operators on each qubit. Once the location and size of the disconnected region is chosen, we prepare it in a surface with solely rough boundaries. Rather than following this up with the creation of a rough defect, we simply prepare the surface by leaving a region of adjacent X checks turned off and the single-body Z terms on the interior of the region unchanged. Since the system began in an eigenstate of any product of Z operators, and since \bar{Z} for the rough qubit commutes with all of the check operators we turn on, the system remains in the $+1$ eigenstate of \bar{Z} after the evolution.

We also could have prepared the rough defect in the -1 eigenstate of \bar{Z} by first performing an adiabatic evolution on each qubit of the form $-Z \rightarrow X \rightarrow Z$. This has the effect of dragging each of the qubits into the -1 eigenstate of the local Z operators, and now, given a region of appropriate size, \bar{Z} will have an eigenvalue of -1 both before and after the defect creation process. (The size constraints amount to ensuring that the weight of the logical operator is odd.)

Smooth defects can be prepared in ± 1 eigenstates of \bar{X} in much the same way, requiring only simple modifications. To prepare a smooth defect in the $+1$ eigenstate of \bar{X} , each qubit first undergoes the evolution induced by the adiabatic sequence $-Z \rightarrow -X$. Likewise, to prepare a smooth defect in the -1 eigenstate of \bar{X} , each qubit first undergoes the adiabatic evolution $-Z \rightarrow X$. Now \bar{X} will have the correct value before and after the evolution that creates the defect, subject to the same size constraints mentioned above.

6.4.6 State injection into defects

Creating defects in known ancilla states is another important building block for our model. In typical architectures based on the surface code, completing a universal set of encoded quantum gates requires the ability to “distill” high fidelity states—called “magic states”—using protocols like the one discovered by Bravyi and Kitaev [BK05]. In this section, we describe how to implement these preparations in an adiabatic simulation of the process of state injection.

In measurement-based injection of a magic state [RHG07], one first exposes a qubit by preparing a single (unencoded) qubit in the state $|\psi\rangle$. Then, the state is quickly encoded in a surface code defect, and the procedure is finished by growing the defect to a sufficiently large size so that it is well-protected from noise. This process need not be perfect, but any error introduced by the injection procedure must keep the total error in the encoded state $|\bar{\psi}\rangle$ below the threshold of the distillation protocol.

We describe our adiabatic simulation of this process for an injection into a smooth defect, but the rough-defect case is similar. We begin by preparing an all-smooth-boundary surface near the edge of the bulk surface using the method described in Sec. 6.4.1. We then create a *rough* defect in a $+1$ eigenstate of \bar{X} in this region using the procedure described in Sec. 6.4.2. The situation is depicted in Fig. 6.14. Because this region has only smooth boundaries, there is nowhere for a string of X operators from the defect to connect. Indeed, if we ignore the one qubit on the interior of the defect, then what we would normally call \bar{X} , a string of X operators enclosing the defect, is already an element of the stabilizer group. It can be formed by taking the product of all the vertex checks. (As an aside, we note that this is a consequence of the topology of the sphere, for which all loops remain homotopic when a single point is removed.) As discussed in Sec. 6.4.2, this leaves the single qubit on the interior of the defect in the $+1$ eigenstate of Z .

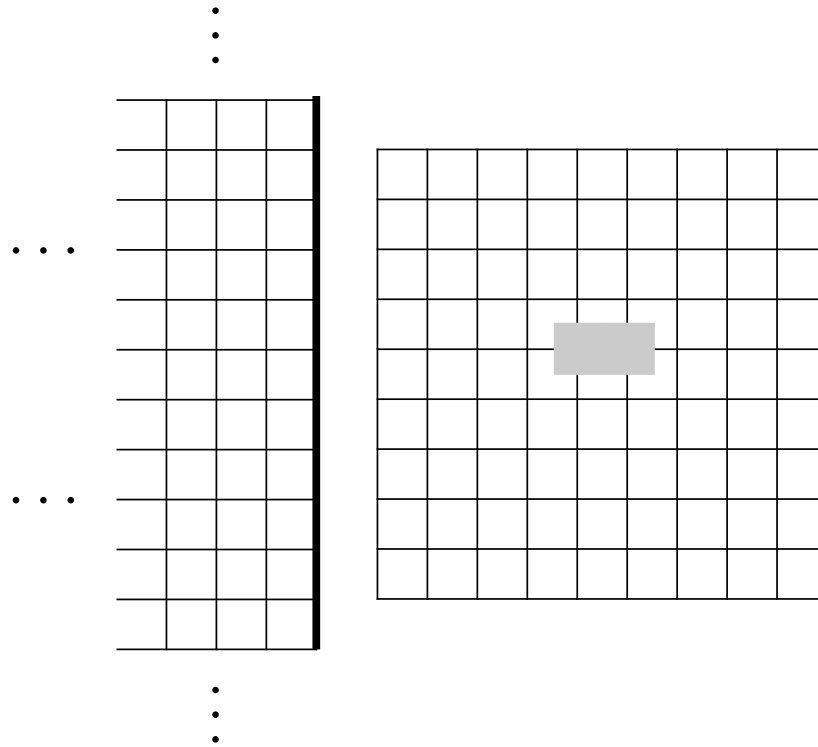


Figure 6.14: After the Hamiltonian deformation (or sequence of deformations), we are left with a surface code with trivial boundaries encoding a rough defect in the state $|+\rangle$.

We then transform this interior qubit to the desired state by an adiabatic evolution. For example, if we want to prepare the state $T|+\rangle$, we evolve using the Hamiltonian $H(s) = (1-s)(-Z) + sUZU^\dagger$, where in this case $U = TH$. If we think of this as a logical qubit, then \overline{X} is a single X on the qubit and \overline{Z} is a single Z on the qubit. Recall that the face checks originally incident on the interior qubit have been modified and are no longer incident. The situation is now described by Fig. 6.15. Next, we adiabatically turn on the two vertex checks that were originally turned off to create the defect. We simultaneously (and adiabatically) also turn off the three-body plaquette checks, as they would otherwise anti-commute with the final Hamiltonian. This evolution transforms the logical operators, since the initial single-body \overline{Z} does not commute with the final X checks. The transformation \overline{Z}

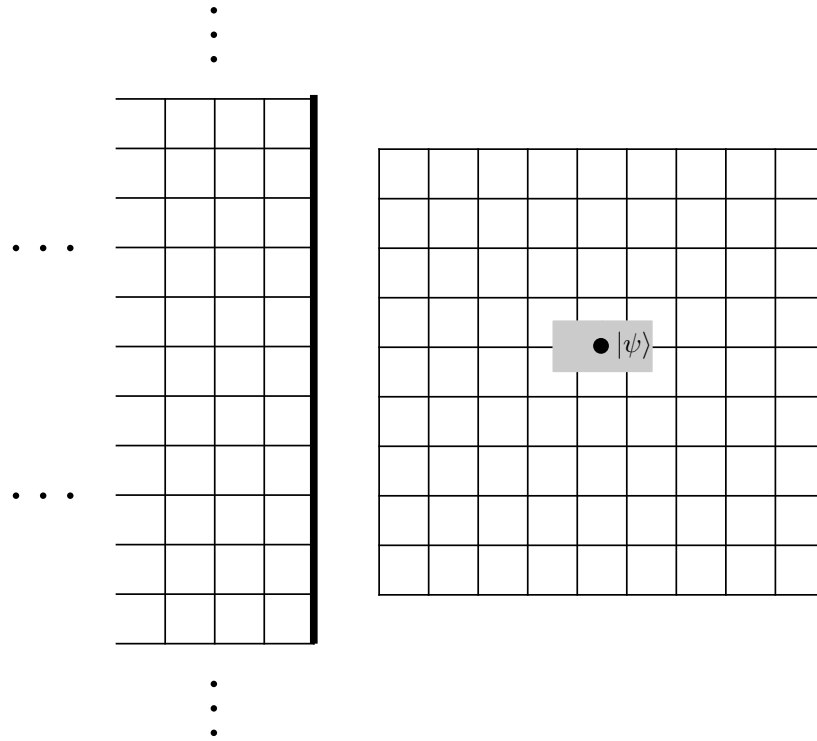


Figure 6.15: The interior qubit is adiabatically dragged to the state $|\psi\rangle$, the desired magic state.

undergoes is determined by the Pauli algebra and the demands of a stabilizer code. Since \overline{Z} must still commute with the code after the vertex checks are turned back on (note that the formerly interior qubit has now been reintroduced to the code because the vertex checks are incident on it once again), and it must not be in the stabilizer group itself, a suitable choice of the new \overline{Z} is the product of the old \overline{Z} and one of the three-body plaquette checks that also did not commute with the vertex checks. What's left is what appears to be a normal two-plaquette defect as shown in Fig. 6.16, but the crucial difference is that there is now no sense of an isolated interior, since the neighboring vertex checks are still incident on the qubit inside. In fact, since \overline{X} has never been disturbed by any of the evolutions we performed, it is still a single-body operator localized to the qubit inside the defect. This leaves the encoded qubit prone to decohering environmental interactions, and so we make it

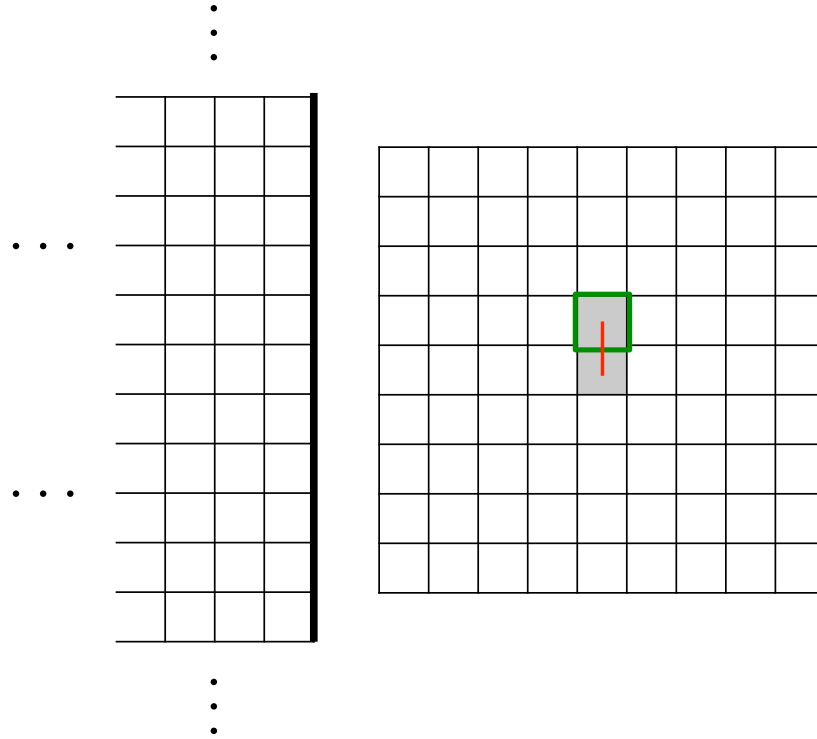


Figure 6.16: The missing X checks are reintroduced to the code, causing neighboring Z checks to be removed. This new defect is now encoded in the state $|\psi\rangle$ with an encircling \overline{Z} and a single-qubit \overline{X} .

larger by “splitting” the defect apart into a pair of defects, as depicted in Fig. 6.17. As we move the parts away from each other, we also grow their perimeters using the methods described above to protect against \overline{Z} errors.

These defects qubits could be used as-is, but to make them more like the defects we’ve worked with so far, we simply take one of the halves and merge it with the global smooth boundary of our preparation region, as depicted in Fig. 6.18. Finally, we attach the surface containing this defect to the main surface using the procedure described in Sec. 6.4.4. This defect can be shuttled in and the boundary can be modified to the original shape.

Encoded distillation circuits, such as the ones depicted in Figs. 6.19 and 6.20,

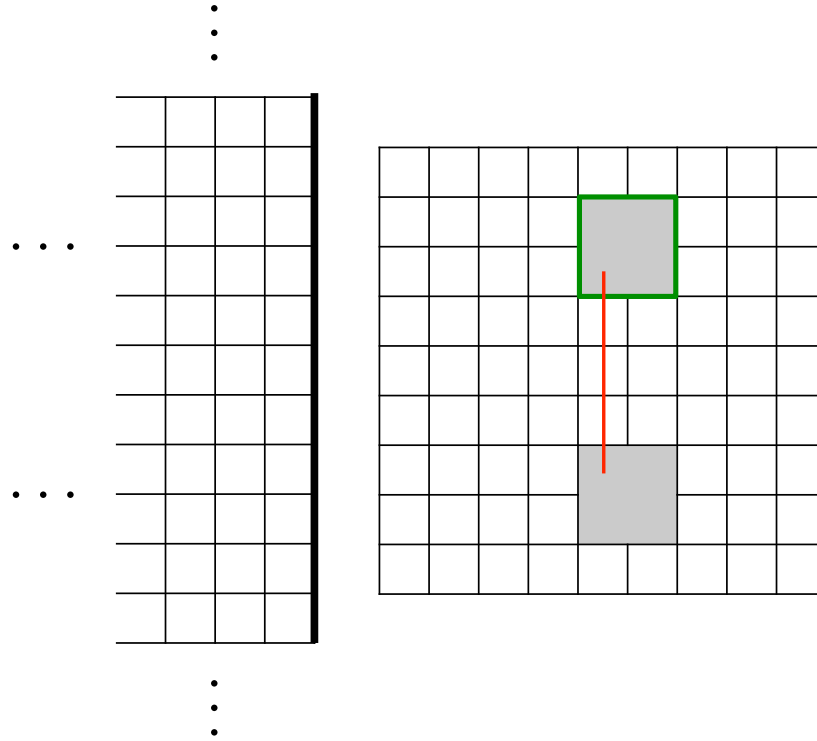


Figure 6.17: Because \overline{X} was only a single-qubit operator, the two removed faces are moved apart and grown to combat decoherence.

require only one more procedure in addition to Pauli X and Z eigenstate preparations, encoded $CNOT$ operations, and the preparation of defects in encoded $T|+\rangle$ and $S|+\rangle$ states, both of which have now been introduced. The last procedure is the measurement of encoded Pauli \overline{X} and \overline{Z} operators, to be described in Sec. 6.5.1.

6.5 Non-adiabatic procedures for surface code defects

The procedures presented in Sec. 6.4 are fully implemented using only adiabatic evolutions of stabilizer Hamiltonians. However, these operations do not allow for

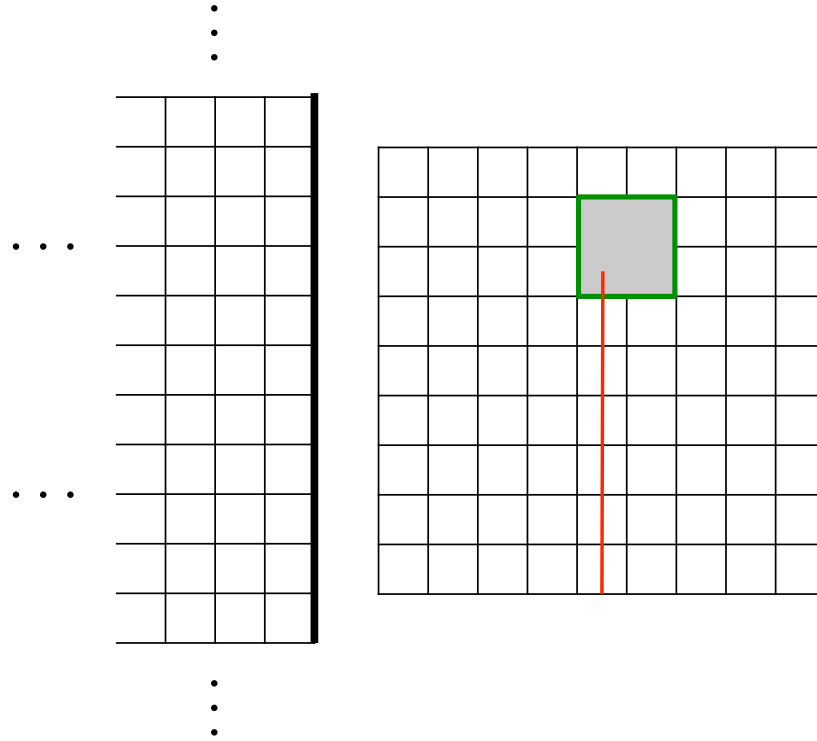


Figure 6.18: One of the defects is merged with the boundary to make the standard single defect.

universal quantum computation. The key missing ingredient is the capability to perform logical measurements—namely, the ability to measure \overline{X} and \overline{Z} for smooth and rough defects. These measurements are the only non-adiabatic ingredients appearing in our model. In this section we describe how to perform them as well as use them in additional procedures, such as heralded application of \overline{X} and \overline{Z} gates. Although the measurements are not protected by adiabaticity or a Hamiltonian gap, their topological nature provides robustness to local errors.

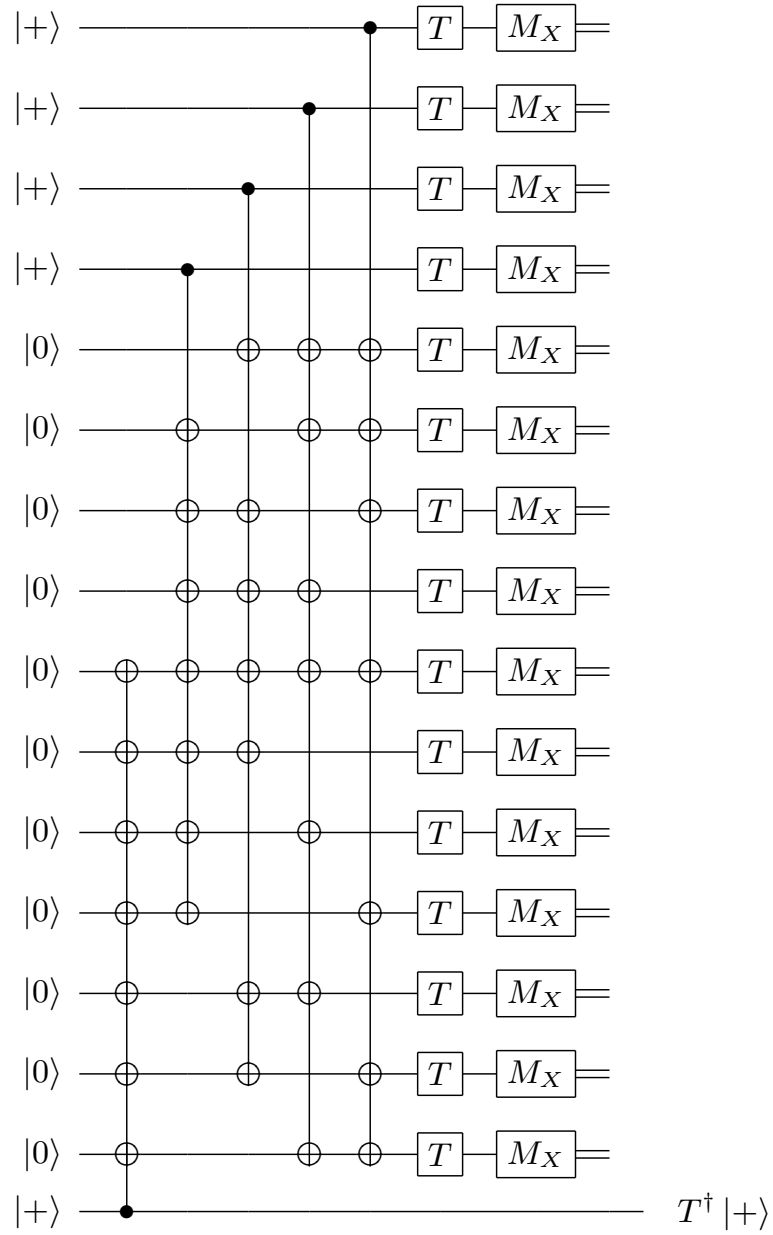


Figure 6.19: Distillation circuit for $T|+\rangle$ states, constructed from the 15-qubit Reed-Muller code's encoding circuit.

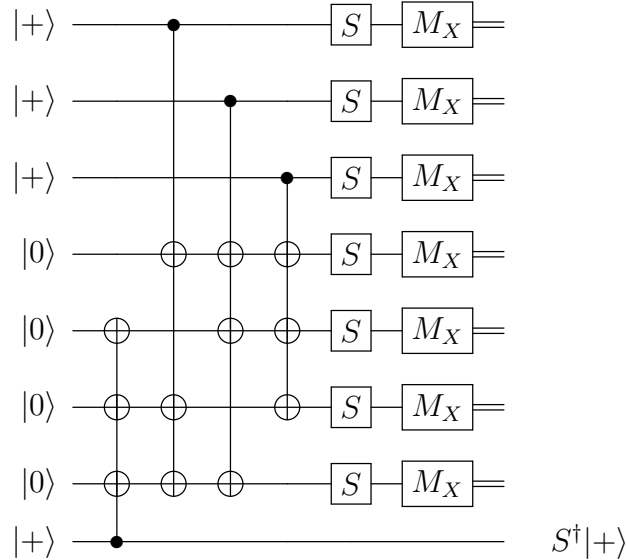


Figure 6.20: A distillation protocol for $S|+\rangle$ states based on the encoding circuit for the $[[7, 1, 3]]$ quantum Steane code.

6.5.1 Measurements of \overline{X} and \overline{Z} for defects

In measurement-based surface code models, defect logical operators are measured in-situ by simply measuring a region of individual qubits in the surface. The parities of the measurements are then used to infer the eigenvalue of \overline{X} or \overline{Z} with probability $1 - O(p^d)$, where d is the distance of the code and p is the probability that an individual qubit measurement is faulty. In our Hamiltonian model, this in-situ measurement is an issue because single-qubit measurements in the surface will necessarily anti-commute with the code Hamiltonian, leading to excitations out of the ground space. If it is the end of the computation, and we want to know the state of all the defect qubits, we can just turn the Hamiltonian off and measure everything. However, the use of magic states via gate teleportation (described later) requires conditioning future actions on the classical outcome of logical qubit measurements. In this section we present an ancilla-coupled method to perform these logical measurements.

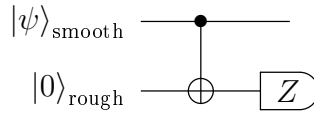


Figure 6.21: An example of measuring \overline{Z} for a smooth qubit. It requires the preparation of a rough defect in a $+1$ eigenstate of \overline{Z} , as discussed in Sec. 6.4.5.

To measure \overline{X} or \overline{Z} for a defect in a “non-destructive” way (meaning that the post-measured state stays in the codespace), we use the method of ancilla-coupled measurement introduced by Steane in Ref. [Ste96c]. Fig. 6.21 depicts this process for measuring \overline{Z} for a smooth defect qubit in the state $|\psi\rangle$. First, we prepare a rough defect in the $+1$ eigenstate of \overline{Z} as described in Sec. 6.4.5. Next, we perform a sequence of adiabatic deformations, described in Sec. 6.4.3, to enact a $CNOT$ gate between the smooth and rough defects. Then, the rough-defect ancilla is detached from the code using the method demonstrated in Sec. 6.4.4. Finally, we turn off the Hamiltonian and destructively measure the isolated region in the Z basis. A similar procedure performs a measurement of \overline{X} for a smooth qubit (simply measure the isolated region in the X basis), and a similar circuit can be used to measure logical operators for a rough defect.

6.5.2 Heralded application of \overline{X} and \overline{Z} to defects

With the ability to perform ancilla-coupled measurements, introduced in Sec. 6.5.1, and the Hamiltonian evolutions described in Sec. 6.4, we can apply \overline{X} and \overline{Z} to defects using the circuit shown in Fig. 6.22, where the measurements are assumed to be of the type described in the previous section. All of the pieces in this circuit have been described previously. The preparation of a rough defect in the $+1$ eigenstate of \overline{Z} is described in Sec. 6.4.5; performing a $CNOT$ between a smooth defect and

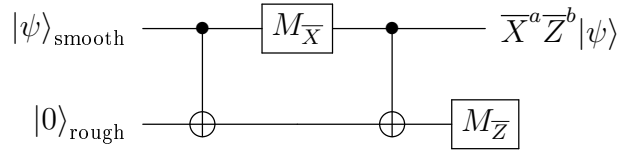


Figure 6.22: Circuit used to apply one of the Pauli operators to a smooth defect qubit. The outcome of the X measurement is $b \in \{0, 1\}$ and the outcome of the Z measurement is $a \in \{0, 1\}$. The outcomes of the measurement all occur with equal probability and the final state depends on these outcomes as shown. If an undesired operator is applied, the ancilla qubit is reinitialized and the circuit is implemented again. However, now the appropriate operator is the one which undoes the operator applied in the first iteration *and* applies the desired operator. (This, of course, will just be a different one of the four operators $\overline{X}^a \overline{Z}^b$.)

a rough defect is described in Sec. 6.4.3; and making measurements of \overline{X} and \overline{Z} for smooth and rough defects was just described in Sec. 6.5.1.

6.6 The completed model

To summarize our surface code model, we list the procedures we have defined in Sec. 6.4 and Sec. 6.5:

1. Sec. 6.4.1: Adiabatic preparation of a surface code encoding no qubits
2. Sec. 6.4.2: Adiabatic preparation of smooth defects in the $+1$ eigenstate of \overline{Z} and rough defects in the $+1$ eigenstate of \overline{X}
3. Sec. 6.4.3: Adiabatic deformation of smooth and rough defects, allowing for defect movement
4. Sec. 6.4.4: Adiabatic detaching and attaching procedures, allowing for the isolation of regions containing defects

Chapter 6. Adiabatic Topological Quantum Computation

5. Sec. 6.4.5: Adiabatic preparation of smooth defects in the ± 1 eigenstate of \overline{X} and rough defects in the ± 1 eigenstate of \overline{Z}
6. Sec. 6.4.6: Adiabatic injection of ancilla states into defects
7. Sec. 6.5.1: Non-adiabatic procedures for “non-destructive” ancilla-coupled measurement of \overline{X} and \overline{Z} for defects
8. Sec. 6.5.2: Non-adiabatic, measurement-based procedure for the heralded application of \overline{X} and \overline{Z}

Magic state gate teleportation of the T gate is performed using the circuit in Fig. 6.23, and the Hadamard gate can be performed with an ancilla state using the circuit in Fig. 6.24. In both cases, the only operations required involve the procedures defined

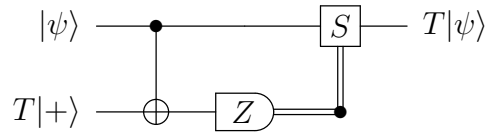


Figure 6.23: Gate teleportation circuit using the $T|+\rangle$ state. The S correction needs to be performed half of the time and can be implemented in the same way using the state $S|+\rangle = |+i\rangle$ instead of $T|+\rangle$ (and utilizing a Z correction half of the time).

in the list above. Other procedures, such as performing a $CNOT$ between two smooth qubits, have been studied previously [RHG07] and also only require operations from the list above. Thus, in encoded form, we can prepare Pauli \overline{X} and \overline{Z} eigenstates, perform a universal gate set, and measure any qubit in either the X or Z basis. Taken together, these procedures allow for universal quantum computation.

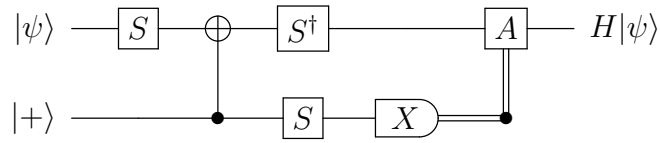


Figure 6.24: Circuit for applying the Hadamard gate with an ancilla state. The correction A depends on the result of the measurement: if the measurement result is $+1$, then $A = X$, and if the measurement result is -1 , then $A = Z$. The S and S^\dagger gates can be performed using a circuit like the one in Fig. 6.23.

6.7 Extension to 2D color codes

We briefly discuss how we can adapt our surface code procedures to the two-dimensional color codes, in particular to the 4.8.8 2D color code.

Color codes in two dimensions are defined on a two-dimensional lattice which is trivalent (each vertex is of degree three) and three-colorable (we can color the plaquettes by three colors such that no two adjacent plaquettes are the same color). Fig. 6.25 is an example of such a lattice. As opposed to the surface code, the color

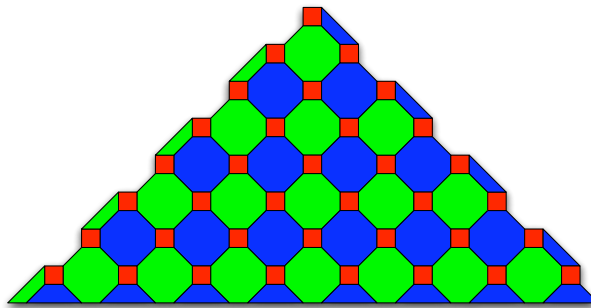


Figure 6.25: A lattice with colored plaquettes on which one can define the color codes.

codes have qubits on the vertices of the lattice. Let $V(p)$ denote the vertices that

Chapter 6. Adiabatic Topological Quantum Computation

are on the boundary of a plaquette, and define a stabilizer group structure of the color codes as follows. To every plaquette p , associate two stabilizer generators, the tensor product of Pauli X on the adjacent qubits, given by

$$S_p^X = \bigotimes_{v \in V(p)} X_v, \quad (6.10)$$

as well as the tensor product of Pauli Z on the adjacent qubits, given by

$$S_p^Z = \bigotimes_{v \in V(p)} Z_v. \quad (6.11)$$

The representative code in Fig. 6.25 has four-body (red) and eight-body (blue and green) stabilizer generators. (These are the weights away from the boundaries of the code, where four-body blue and green faces also exist.) Boundaries in the color code also have a slightly richer structure. They are no longer smooth and rough, but rather, they have a color associated to them. This color is determined by the boundary's *missing color*. For example, in Fig 6.25, the bottom boundary is red, since there are no red plaquettes adjacent to the bottom edge. A careful accounting of qubits and checks in Fig. 6.25 indicates that there is a single logical qubit associated with the surface. For our purposes, we will treat it as a “gauge” degree of freedom using the subsystem stabilizer code formalism [Bac06]. The operators \bar{X} and \bar{Z} associated with this qubit can be chosen as strings of Pauli X and Z operators, respectively, along the bottom boundary.

Just as with the surface codes, we can create defects in the color code to store more logical qubits. In addition to having a type (X or Z), the defects now also have a color. To create the analog of a smooth defect, we remove a Z -type stabilizer generator, and to create the analog of a rough defect, we remove an X -type generator. For a Z -type defect, one choice for \bar{Z} is the removed generator (equivalent to a string of a *different color* around the defect), and one choice for \bar{X} is a string of X s connecting to the appropriately-colored boundary, corresponding to the color of the removed plaquette.

Chapter 6. Adiabatic Topological Quantum Computation

As for any stabilizer code, we can define the Hamiltonian in Eq. 6.1, and it has a ground space equivalent to the codespace of the code. In the case of the color codes it can be written as

$$H = - \sum_p (S_p^X + S_p^Z). \quad (6.12)$$

The color-code Hamiltonian, like the surface-code Hamiltonian, does not lead to a self-correcting quantum memory, but we can use adiabatic interpolations between static Hamiltonians of the type in Eq. 6.12.

As in Sec. 6.4.1, we can perform an adiabatic interpolation to initially create the color code without any defects. We imagine the same setting—a large number of qubits in the ground state of local Hamiltonians $H = -Z$ —and prepare the code by using an interpolation of the form

$$H(t) = \left(1 - \frac{t}{T}\right) \sum_{j \in \mathcal{Q}} (-Z_j) + \frac{t}{T} \sum_p (-S_p^X - S_p^Z) + \frac{t}{T} \sum_{j \notin \mathcal{Q}} (-Z_j). \quad (6.13)$$

(Since \bar{Z} for the newly created code commutes with this Hamiltonian at all times, and since it initially has eigenvalue +1, the qubit associated with the surface is prepared in the +1 eigenstate of \bar{Z} . This is the gauge degree of freedom mentioned above.) As for the surface code, we choose to create a small color code first and then grow it to avoid a shrinking gap. The small color code is then grown in a manner similar to Sec. 6.4.1. To create a green Z -type defect in the +1 eigenstate of \bar{Z} , described for the surface code in Sec. 6.4.2, two Z -type green plaquettes separated by one red plaquette are turned off while simultaneously turning on $-XX$ on a pair of qubits in between, as shown in Fig. 6.26. Note that during the defect creation a neighboring blue plaquette gets modified to a six-body operator and a neighboring red plaquette gets modified to a two-body operator. Red and green defects can also be created in a similar way, and X -type defects can be created in the state $|+\rangle$ by reversing the roles of X and Z above.

The surface code procedures for growing and moving defects, presented in Sec. 6.4.3,

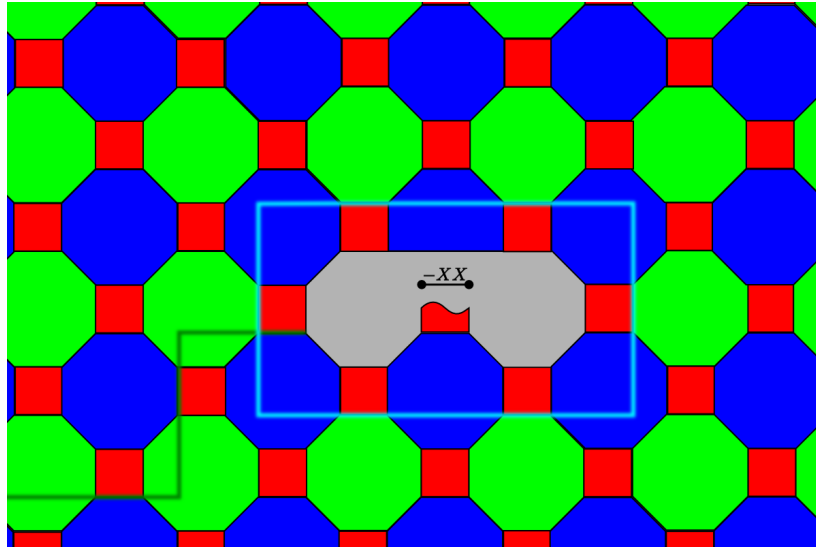


Figure 6.26: Two adjacent green plaquettes are turned off while turning on the $-XX$ Hamiltonian shown, creating a Z -type defect in the $+1$ eigenstate of \bar{Z} (shown here as a light blue string encircling the defect).

can also be adapted to the color codes. We won't present the the cases for different numbers of interior qubits separately here. Rather, we examine the simplest case when there are only two neighboring qubits. The other cases, as in the surface code, simply require more modifications of adjoining checks. To grow a Z -type green defect like the one in Fig. 6.26, first pick another green face. It will be separated from the defect region by a red plaquette. Along one of the two lines connecting the defect region to the green check, turn on $-XX$ while turning off the green plaquette. This will incur a modification a neighboring blue plaquette as well as the red plaquette itself.

Next, we show that the color code also supports detachment and attachment procedures, described in Sec. 6.4.4 for the surface code. Imagine a two-plaquette red defect, depicted in Fig. 6.27, that we would like to isolate from the bulk code. To complete the detachment procedure for a Z -type red defect, two $-XX$ Hamiltonians—on the qubits indicated by yellow dots—are turned on while turning off the Z -type red

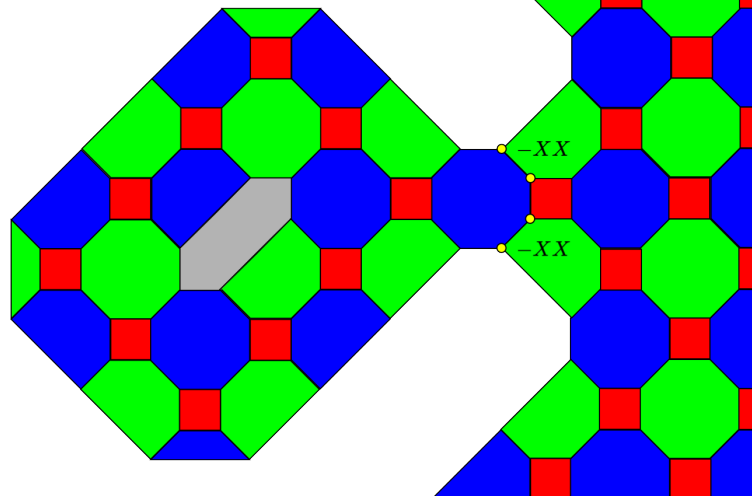


Figure 6.27: A Z -type red defect isolation procedure. The “drawbridge” in this case is the red plaquette adjacent to the yellow dots in the figure. The Z -type check on the red face is turned off while the two $-XX$ operators are turned on. The four-body X operator that is the product of the two $-XX$ Hamiltonians is in the stabilizer group before the evolution, and it is trivially in the stabilizer group of the code after the evolution. The blue and green plaquettes adjacent to the yellow dots are modified to be a four-body operators. (Also note that the X -type check on the red plaquette must also turned off to fully isolate the region, and two $-ZZ$ Hamiltonians are turned on.)

plaquette operator adjacent to the dots. In the process, the adjacent blue and green plaquettes get modified to four-body operators. Since the four-body X operator that is the product of the two $-XX$ Hamiltonians is in the stabilizer group at the beginning *and* at the end of the evolution, we have successfully severed the two code regions.

As discussed in Sec. 6.4.5, it is important that we are able to prepare Z -type defects in eigenstate of \overline{X} and vice versa. In this case, the procedure is essentially identical, and proceeds by preparing single qubits in particular states (± 1 eigenstates of X for Z -type defects and ± 1 eigenstates of Z for X -type defects). Just as before, a defect location is anticipated and the preparation of the surface proceeds normally

everywhere except for the defect.

Ancilla state injection for the color codes is slightly different compared to the procedures for the surface code introduced in Sec. 6.4.6. After isolating a region with green boundaries, or creating such a region adjacent to a green boundary, we use the procedures described above to introduce an X -type *and* a Z -type defect at the same location, as pictured in Fig. 6.28. Notice that the interior red checks

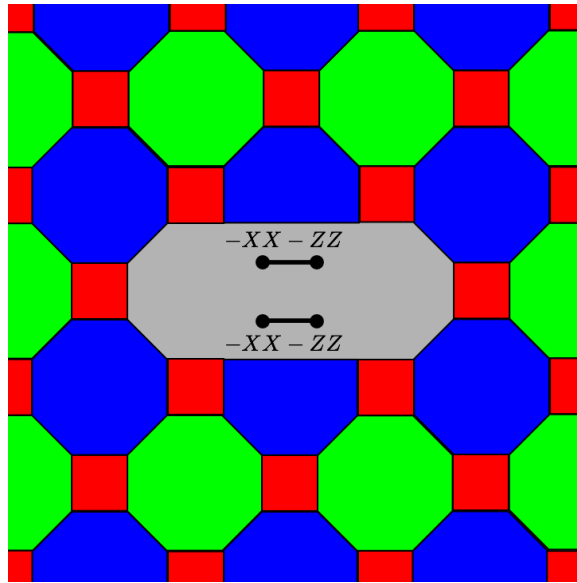


Figure 6.28: The creation of a defect region with both the X -type and Z -type green checks turned off. There are four interior qubits prepared in two Bell pairs by this procedure.

have also been modified during this procedure, putting the four interior qubits into two Bell pairs. Additionally, the neighboring blue plaquettes have been modified to six-body operators. An evolution is then performed that only touches these four interior qubits, turning on the Hamiltonians pictured in Fig. 6.29 while turning off the two $-XX - ZZ$ Hamiltonians. Next, just as we did for the surface code, we adiabatically drag a qubit to the desired state, as pictured in Fig. 6.30. The “logical qubit” is localized to the upper-right qubit, with single-body \bar{X} and \bar{Z} operators. The

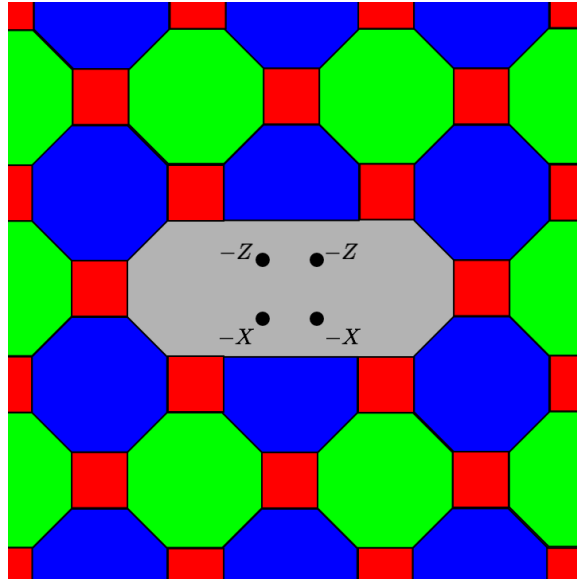


Figure 6.29: Four interior qubits are “exposed.”

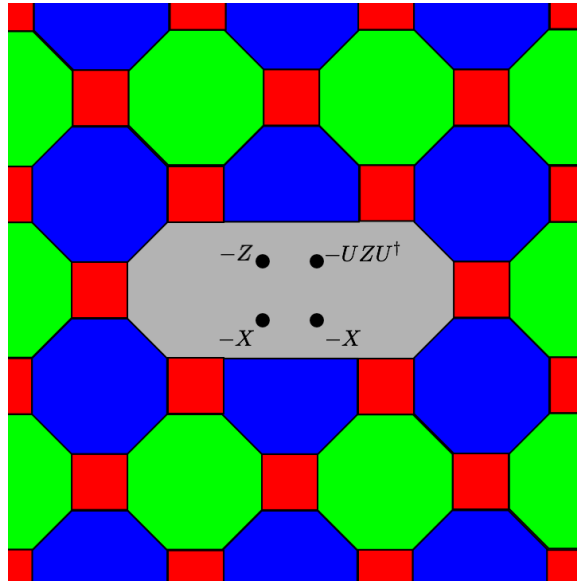


Figure 6.30: The upper-right qubit is adiabatically dragged to the desired state. For instance, to inject $T|+\rangle$ states, $U = TH$.

next step is to “grow” these logical operators in a particular way. This is achieved by performing another adiabatic evolution on the four qubits to the Hamiltonian

represented in Fig. 6.31, which is just the reintroduction of the red face checks that we turned off at the beginning. This evolution modifies \overline{X} and \overline{Z} from single-body

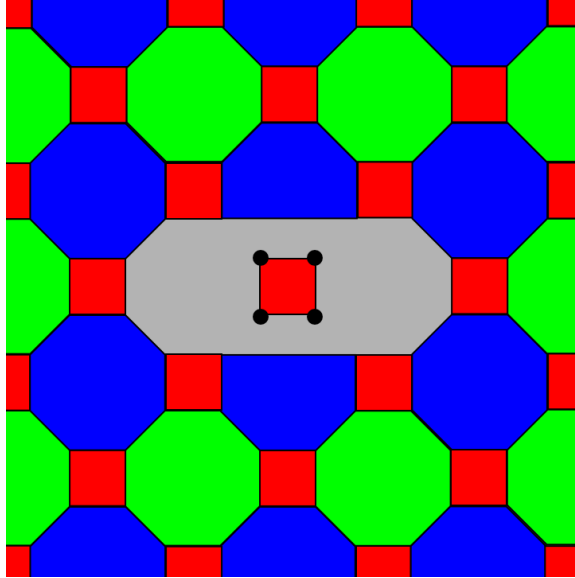


Figure 6.31: The single-body terms in Fig. 6.30 are turned off while turning on the X -type and Z -type checks on the red plaquette.

operators to the operators shown in Fig. 6.32. Finally, the X -type checks on the green faces currently housing the defect are turned on while the adjacent Z -type blue faces are turned off, leading to the situation depicted in Fig. 6.33. As in the case of the surface code, one of these faces is moved away and absorbed into the green boundary of the region. Then the region is attached and the green defect encoding the state is moved into the bulk computational region.

None of the other procedures introduced in Sec. 6.4 and Sec. 6.5 are appreciably different for the color codes. Measurements are still performed in an ancilla-coupled manner, and \overline{X} and \overline{Z} can still be applied in a heralded fashion. Logical $CNOT$ gates are still performed by braiding, with the control being a Z -type defect and the target being an X -type defect. Ref. [LAR11] discusses how to perform a $CNOT$ between defects of the same type (or color). Thus, all the ingredients are precisely the

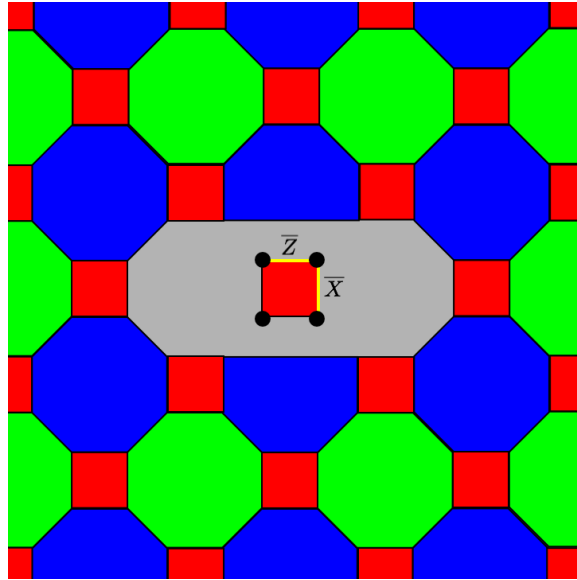


Figure 6.32: \bar{X} and \bar{Z} after the reintroduction of the red plaquette in Fig. 6.31.

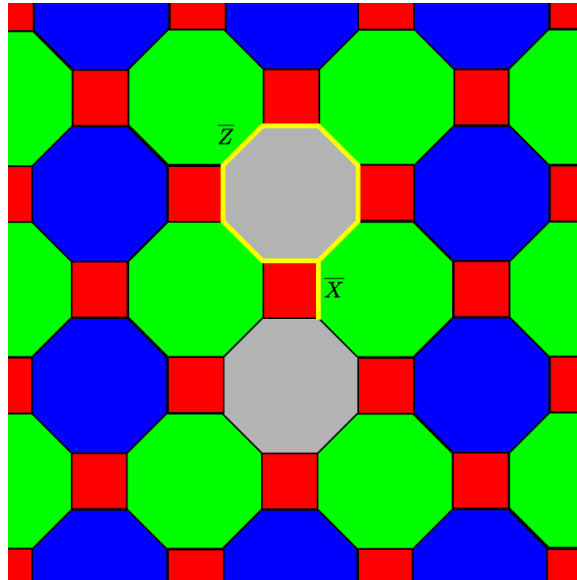


Figure 6.33: The arrangement of the defect after reintroducing the X -type green plaquettes. \bar{X} is a string of Pauli X operators connecting two blue faces and \bar{Z} is a loop of Pauli Z operators around a blue face.

same, and encoded universal quantum computation can be performed via adiabatic interpolations between static Hamiltonians and ancilla-coupled measurements.

6.8 Conclusion

We have presented a model of quantum computation that utilizes adiabatic interpolations between static Hamiltonians that encode quantum information in their degenerate ground spaces. By utilizing the process of adiabatic code deformation, we create and grow small code regions, introduce and braid defects, and inject arbitrary states into defects. Surprisingly, these procedures never cause the Hamiltonian gap to shrink below a constant proportional to Δ , and they can all be performed with the protection of a gap and topology. However, to perform logical measurements we use an ancilla-coupled scheme, braiding and isolating an ancilla defect and then turning pieces of the Hamiltonian off and destructively measuring a code region. Taken together, these procedures allow for universal quantum computation.

Our model lives at the intersection of three other models of quantum computation. It provides explicit examples of adiabatic evolutions in the setting of a topological code, and we make an effort to supply procedures that do not increase the rate at which errors (anyons) are introduced to the system. Since we store information in the ground space of a changing Hamiltonian, our model also borrows intuition and robustness from holonomic quantum computing. Indeed, the braiding operations we perform rely precisely on the non-trivial structure of ground space holonomies. Lastly, our adiabatic interpolations are like miniature adiabatic quantum computations, and their implementations are made less noisy by traversing an adiabatic path more slowly.

Unfortunately, the model we present is not fault-tolerant. While the lifetime of the ground space, and thus the encoded quantum information, is exponential in Δ/T

in the presence of coupling to a thermal bath, no protection is gained by increasing the size of the code. It would be interesting to study a model that can actively remove entropy from the system, utilizing active error correction in a way that is compatible with the Hamiltonian nature of the model, but we do not address these problems in this work.

We hope that the model we have analyzed here can be useful for a further understanding of the properties of quantum computation based on stabilizer Hamiltonians. In particular, it would be interesting to extend this work to models such as Kitaev's quantum double model [Kit03] or the Turaev-Viro codes [KKR10], where universality can be achieved without the creation and distillation of magic states.

Chapter 7

Direct Distillation of a New Family of Magic States

This chapter describes research with Andrew Landahl on a new family of magic state distillation protocols and is available in Ref. [LC13]. Portions of this chapter were originally written by Andrew Landahl, but this final edit it my own. The technical results here are almost entirely my own, from the calculation of distillation thresholds to the counting of resource states.

Since being posted as an e-print to arxiv.org, much other work has been performed in the general area of addressing resource requirements in fault-tolerant quantum computation. A non-exhaustive list includes approaches that don't use distillation [PR13, JOL13], repeat-until-success methods for distilling states [PS13], bottom-up approaches to distillation [DCP14], and asymptotically optimal compiling methods with better constants [RS14, Kli13]. It remains unclear if any of these methods supersede our work; a useful future project would be to compare and contrast the variety of approaches that have been developed to address the ways that arbitrary gates can be synthesized fault-tolerantly.

7.1 Introduction

One of the biggest challenges in quantum information science is that quantum information is incredibly fragile. Even with great experimental care, decoherence can quickly corrupt key features such as superposition and entanglement. To circumvent the ravages of decoherence, one can consider alternative models of quantum computation, such as adiabatic quantum computation [FG98, AvK⁺04, MLM07], which may offer direct physical immunity to certain classes of noise [CFP01, ÅKS05b, ÅKS05a, SL05, RC05, AJN06, Gai06, TS07, AAN09, ATA09, dBP10]. Another approach is to encode quantum information redundantly in an error-correcting code and process it fault-tolerantly to suppress the catastrophic propagation of errors [Sho95, Sho96]. Somewhat miraculously, this latter approach works, and works arbitrarily well, when quantum computations are expressed as quantum circuits in which each elementary operation has a failure probability below a value known as the *accuracy threshold* [ABO97, ABO99, Kit97a, Ste97, KLZ98, Pre98a, Pre98c]. Estimates for the accuracy threshold vary, and depend in part on the specifics of the fault-tolerant quantum computing protocol used. One of the more favorable estimates is $\approx 1\%$ for a protocol based on Kitaev’s surface codes [Kit97b, DKLP02, RHG07, FSG09]. An outstanding grand challenge in quantum information science is finding a way to marry fault-tolerance methods with intrinsically robust computational models to achieve fault tolerance with more achievable resource requirements [JFS06, Lid08, PSRDL12, YS13].

One of the factors driving up the resource requirements in fault-tolerant quantum computing is the need to restrict the set of elementary operations in the “primitive” or “physical” instruction set to be finite. This is necessary because these instructions are presumed to be implementable only up to some maximal accuracy. One of the main jobs of a fault-tolerant quantum computing protocol is to define how one should sequence these primitive instructions together to synthesize arbitrarily accurate versions of each element of a universal “encoded” or “logical” instruction

Chapter 7. Direct Distillation of a New Family of Magic States

set, even when the primitive instructions themselves are faulty. Then, using these logical instructions, one can realize any quantum algorithm arbitrarily reliably, even in the face of decoherence and other sources of noise.

In a typical fault-tolerant quantum computing protocol, some logical instructions are “easy” to synthesize in that they are either transversal or otherwise naturally fault-tolerant. The accuracy of these logical instructions can be improved arbitrarily well by using quantum codes with larger distance. More quantitatively, the number of gates and qubits required to achieve approximation error ϵ for the “easy” instructions scales as $\mathcal{O}(\log^\alpha(1/\epsilon))$, where α depends on the protocol, predominantly on the quantum code and classical decoding algorithm it uses. Standard techniques for realizing such gates include transversal action [Pre98a, Pre98c] and code deformation [DKLP02, RHG07]. 2D topological codes using most-likely-error decoding can achieve $\alpha = 3$ [DKLP02, RHG07]; Pippenger has conjectured that it should be possible to lower α all the way to 1 [Ahn04].

Most protocols also have a set of logical instructions that are “hard” to synthesize, requiring additional methods and resources. The Eastin-Knill theorem, for example, guarantees that no protocol based on quantum codes that can detect arbitrary single-qubit errors can realize a universal logical instruction set by transversal action alone [EK09]. A typical approach to synthesizing these hard logical instructions is to use the “magic state” approach, in which the “hard” instructions are state preparations that are distilled to high fidelity using the “easy” instructions [BK05]. The number of ideal gates and qubits required to achieve approximation error ϵ in this approach scales as $\mathcal{O}(\log^\beta(1/\epsilon))$, where β depends on the magic-state distillation protocol. When the resource costs for the “easy” gates are also considered, the combined overhead scales as $\mathcal{O}(\log^{\alpha+\beta}(1/\epsilon))$. In the well-studied Bravyi-Kitaev 15-to-1 distillation protocol [BK05], $\beta = \log_3 15 \approx 2.47$. More recent constructions by Bravyi and Haah [BH12] and by Jones [JWM⁺12] achieve $\beta = \log_2 3 \approx 1.58$. Bravyi and Haah

conjecture that it should be possible to lower β all the way to 1 [BH12].

As an aside, it is worth mentioning that fault-tolerant quantum computing protocols based on some quantum codes have no “hard” logical instructions at all. For example, the 3D (and higher-dimensional) topological color codes have this feature [BMD07, LAR11]. They cleverly circumvent the Eastin-Knill theorem by making (non-transversal!) quantum error correction be the process by which magic-states are prepared. A challenge to using these codes in practice is that implementing them without relying on long-distance quantum communication requires 3D spatial geometry, but many quantum technologies are naturally restricted to 1D or 2D. Even more challenging is that the only explicit 3D color code of which we are aware is the 15-qubit shortened quantum Reed-Muller code [BMD07]. Concatenated schemes using the 15-qubit code would lead to a fault-tolerant scheme with only “easy” instructions, but concatenated schemes typically suffer significant performance losses when realized in a fixed spatial dimension. For example, the largest accuracy threshold of which we are aware for a concatenated-coding protocol in a semiregular 2D geometry is 1.3×10^{-5} [SR09].

Because of the additional overhead incurred in synthesizing “hard” logical instructions, research to date has focused on what one might term *reduced instruction set computing*, or RISC, architectures in which only a single “hard” logical instruction is added to an otherwise “easy” logical instruction set. However, while a RISC architecture minimizes the number of hard instructions in an instruction set, it does not necessarily minimize the number of hard instructions used in specific algorithms. For example, in order to compile the logical instructions into a sequence that approximates a quantum computation with error at most ϵ , one must use $\mathcal{O}(\log^\gamma(1/\epsilon))$ gates, where γ depends on the quantum compiling algorithm used. The overall cost of fault-tolerantly implementing a quantum computation is then $\mathcal{O}(\log^{\alpha+\beta+\gamma}(1/\epsilon))$. By increasing the size of the instruction set so that one has a *complex instruction set*

computing, or CISC architecture, one can optimize both β and γ together rather than separately. When quantum compiling is optimized independently, γ can be no lower than 1 [HRC02], a value recently achieved by an explicit Diophantine-equation-based algorithm by Selinger [Sel12] and Kliuchnikov *et al.* [KMM13a]. For comparison's sake, the more well-studied Dawson-Nielsen variant of the Solovay-Kitaev algorithm achieves $\gamma = \log 5 / \log(3/2) \approx 3.97$ [DN06].

To compare and contrast the RISC and CISC approaches more concretely without being encumbered by details of quantum error correcting codes and fault tolerance (which only contribute to α and a delineation of which logical instructions are “easy” or “hard”—properties shared by both approaches), we abstract these details away and simply consider the straightforward problem of how to approximate $\pi/2^k$ rotations of a qubit about its Z axis with a desired error at most ϵ' when we are given the ability to perform a proscribed set of “easy” instructions that are error-free and a proscribed set of “hard” instructions that have error at most $\epsilon > \epsilon'$. In this setting, it is clear that some kind of distillation of the hard instructions will be necessary to synthesize the Z rotations with lower error. $Z(\pi/2^k)$ rotations are a natural candidate transformation to use to compare RISC and CISC approaches, because they arise in many quantum algorithms, for example those that make use of the quantum Fourier transform [NC00].

In Sec. 7.2, we formulate the statement of the problem we are considering more precisely. In Sec. 7.3, we review the standard RISC solution to this problem. In Sec. 7.4, we describe our CISC solution, and compare it to the RISC solution, demonstrating that for a regime of target ϵ' our solution offers a reduction in the number of resource states used to achieve this task. Sec. 7.5 concludes. Appendix B elaborates the shortened quantum Reed-Muller codes we use to effect our protocol, and Appendix C formulates a testable set of criteria one can use to check if a code admits $Z(\pi/2^k)$ transversally.

7.2 Problem statement

Consider quantum Z rotations of the form

$$Z_k := \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/2^k} \end{pmatrix} = e^{i\pi/2^{k+1}} R_z\left(\frac{\pi}{2^k}\right), \quad (7.1)$$

for integers $k \geq 0$. As a shorthand, we use Z to denote the Pauli operator Z_0 and S and T to denote the rotations Z_1 and Z_2 respectively. We are interested in the scenario in which the Z_k gates are not available directly, but rather their action on $|+\rangle$ states is, where $|+\rangle := H|0\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ and $H := (X + Z)/\sqrt{2}$. For concreteness, let $\mathcal{Z}_{k_{\max}}$ denote the set of states of the form

$$Z_k|+\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle + e^{i\pi/2^k} |1\rangle \right) \quad (7.2)$$

for $2 \leq k \leq k_{\max}$.

In conjunction with the set \mathcal{S} of *stabilizer operations* [Got99], the set $\mathcal{Z}_{k_{\max}}$ can effect universal quantum computation, even when restricted to $k_{\max} \leq 2$ [NC00]. Here we restrict our attention to a certain (overcomplete) generating set for \mathcal{S} , namely the set consisting of the operations

$$\{I, X, Y, Z, S, S^\dagger, H\} \cup \{|0\rangle, |+\rangle, M_Z, M_X\} \quad (7.3)$$

and

$$\{\Lambda(X^{q_1} \otimes \cdots \otimes X^{q_m}) \mid q_i \in \{0, 1\}\}, \quad (7.4)$$

where I, X, Y , and Z denote the Pauli operators, M_X and M_Z denote projective measurements in the X and Z bases (but which may be “destructive” in that they do not necessarily prepare X or Z eigenstates after the measurement), and $\Lambda(X^q)$ denotes the one-control, many-target controlled-NOT gate, where the number of targets m is some efficiently computable number. The unitary gates in this generating set

generate a subgroup of the stabilizer operations known as the *Clifford* group [Got99], which is the set of operations that conjugate (tensor products of) Pauli operators to (tensor products of) Pauli operators.

These generators of \mathcal{S} are “easy” to perform at the logical level for the 4.8.8 2D color codes, motivating our choice [LAR11]. The set is also almost “easy” for Kitaev’s 2D surface codes [Kit97b], except generating S and S^\dagger requires some constant startup costs that can be amortized [And12]. Amazingly, as noted in the introduction, all elements from the set $\mathcal{S} \cup \mathcal{Z}_2$ —a universal set—are “easy” to perform at the logical level for 3D color codes, but 3D geometries are required to realize error correction with these codes in a spatially local manner [LAR11].

While errors in the “easy” operations can be suppressed arbitrarily close to zero by using arbitrarily large 2D topological codes, errors in the operations in $\mathcal{Z}_{k_{\max}}$ cannot, making these operations “hard” for these codes. The states in $\mathcal{Z}_{k_{\max}}$ can be “injected” into such codes at the logical level [RHG07], but doing so also injects the errors in the state. In other words, if the states in $\mathcal{Z}_{k_{\max}}$ have errors that are at most ϵ (as measured by the trace distance [NC00]) as primitive instructions, then the injected states will have errors that are essentially the same when they become logical instructions, assuming the injection process itself adds errors at a low enough probability¹.

Motivated by these properties of 2D topological codes, we will fix the control model for our study to be the aforementioned generators of \mathcal{S} and $\mathcal{Z}_{k_{\max}}$, and the error model to be one in which the operations in \mathcal{S} are error-free but in which the $\mathcal{Z}_k|+\rangle$ states in $\mathcal{Z}_{k_{\max}}$ each err by at most ϵ , as measured by the trace distance. Notice that this control model makes no reference to codes or fault-tolerant quantum computing protocols. We have abstracted these away to focus on how to combine

¹How errors propagate in the injection process is an understudied problem in our opinion. However, we will not consider this issue here because we are abstracting away the details of quantum error correcting codes in our analysis.

elementary operations in \mathcal{S} and $\mathcal{Z}_{k_{\max}}$ to achieve high-fidelity Z rotations.

The question we address here is,

How many resource states drawn from $\mathcal{Z}_{k_{\max}}$ does it take to approximate Z_k with error at most $\epsilon' < \epsilon$ as a function of k_{\max} , k , ϵ , and ϵ' ?

The values of k we are interested in could be smaller than, equal to, or larger than k_{\max} . However, since Z_0 and Z_1 are both in the error-free set \mathcal{S} , we restrict our attention to $k \geq 2$.

7.3 Traditional quantum RISC architecture solution

The standard method for refining the accuracy of a Z_k rotation is to synthesize it with what one might term a quantum *reduced instruction set computing*, or quantum RISC, architecture. The main idea is to only synthesize $T := Z_2$ gates to high accuracy and then rely on a quantum compiling algorithm to approximate Z_k arbitrarily well with a quantum circuit over T gates and adaptive stabilizer operations. The overall process can be broken into the three steps of *quantum compiling*, *quantum gate teleportation*, and *magic-state distillation*.

7.3.1 Protocol

Quantum compiling

The first step, *quantum compiling*, generates a classical description of an ideal quantum circuit that approximates Z_k to accuracy ϵ_{qc} using $\mathcal{O}(\log^\gamma(1/\epsilon_{\text{qc}}))$ quantum operations drawn from some instruction set, for some small constant γ . While the

Chapter 7. Direct Distillation of a New Family of Magic States

error ϵ_{qc} can be measured in multiple ways, a wise choice is to measure ϵ_{qc} using the completely-bounded (“diamond”) trace distance [Kit97a, Sac05, Wat09] for reasons that we will explain later. Examples of quantum compiling algorithms include the Solovay-Kitaev algorithm [Sol00, Kit97a, NC00, HRC02, DN06, Fow05, TVH12], the Kitaev phase kickback algorithm [Kit95, CEMM98, KSV02], programmed ancilla algorithms [IWPK08, JWM⁺12, DCS13], genetic algorithms [MK13], and even Diophantine-equation algorithms [KMM13a, Sel12]. When the accuracy demand is not great, it is sometimes even plausible to use algorithms which take exponential time to find very short approximation sequences [Fow11a, AMMR13, KMM13b, BS12]. As noted in the introduction, values for γ range from 3.97 to 1.

Quantum compiling algorithms typically assume that the elements of the instruction set are error-free. If one implements the compiled circuit $Z_k^{(\text{qc})}$ for Z_k with operations that may be in error, the resulting approximation error will increase. To calculate the total error ϵ_k in this flawed circuit $\tilde{Z}_k^{(\text{qc})}$, we use the fact that the diamond norm has many useful mathematical properties, including obeying the triangle inequality, the chaining inequality, and unitary invariance [GLN05]. Using these, we can bound ϵ_k as

$$\epsilon_k = d_{\diamond}(Z_k, \tilde{Z}_k^{(\text{qc})}) \quad (7.5)$$

$$\leq d_{\diamond}(Z_k, Z_k^{(\text{qc})}) + d_{\diamond}(Z_k^{(\text{qc})}, \tilde{Z}_k^{(\text{qc})}) \quad (7.6)$$

$$\leq \epsilon_{\text{qc}} + n_T \epsilon_T, \quad (7.7)$$

where the compiled circuit uses n_T T gates, each with error at most ϵ_T . To achieve the desired approximation error of ϵ' , it follows that sufficient conditions are

$$\epsilon_{\text{qc}} \leq C_{\text{qc}} \epsilon' \quad (7.8)$$

$$\epsilon_T \leq C_T \epsilon' / n_T, \quad (7.9)$$

for positive constants constrained to obey

$$C_T + C_{\text{qc}} \leq 1. \quad (7.10)$$

For comparison with our protocol introduced later, we chose the Diophantine equation-based compiling protocol presented by Selinger in Ref. [Sel12]. This protocol saturates the asymptotic lower bound (up to constants) on the number of T gates required to approximate a single-qubit gate, and for Z rotations has a T count of

$$n_T(\epsilon_{\text{qc}}) \approx 11 + 4 \log_2 \left(\frac{1}{\epsilon_{\text{qc}}} \right). \quad (7.11)$$

Quantum gate teleportation

The second step, *quantum gate teleportation*, replaces each T gate in the quantum-compiled circuit by an adaptive stabilizer circuit that teleports the T gate from the state $T|+\rangle$ or $T^\dagger|+\rangle$ to the desired qubit. An example of a teleportation circuit using $T|+\rangle$ is depicted in Fig. 7.1. The circuit is also correct if both T operators are changed to T^\dagger ; it is even correct if only one of the T operators is changed to a T^\dagger if the classical control is also changed to act on a 0 instead of a 1.

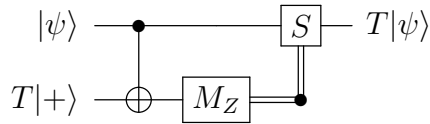


Figure 7.1: Circuit for teleporting the T gate from the $T|+\rangle$ magic state.

Each teleportation circuit requires the use of just a single $T|+\rangle$ resource state. The accuracy requirement set by ϵ_T will determine whether these are ‘bare’ $T|+\rangle$ states of accuracy ϵ or whether these states are the result of one or more rounds of distillation, described in the next section.

Magic-state distillation

The third step, *magic-state distillation*, generates $T|+\rangle$ or $T^\dagger|+\rangle$ states with accuracy ϵ_T from a much larger collection of states whose accuracy is only ϵ . Reichardt showed that this is possible using an *ideal* (error-free) stabilizer circuit if and only if ϵ is less than the distillation threshold $(2 - \sqrt{2})/4 \approx 0.146$ [Rei05]. When operations in the stabilizer circuit can err, the evaluation of the threshold is more complex, as studied by Jochym-O'Connor *et al.* [JOYHL13].

There are multiple variations on how to implement magic-state distillation discussed in the literature [Kni04, Rei04, Rei09, BK05, MEK12, BH12]; a popular one is the 15-to-1 Bravyi-Kitaev protocol [BK05] based on the 15-qubit shortened quantum Reed-Muller code $\overline{QRM}(1, 4)$. (See Appendix B for an explanation of this notation.)

To date, the best distillation scheme in terms of resource costs is a hybrid of the 15-to-1 Bravyi-Kitaev protocol [BK05], the 10-to-2 Meier-Eastin-Knill protocol [MEK12], and the $(3k+8)$ -to- k family of protocols discovered by Bravyi and Haah [BH12]. Bravyi and Haah optimized combinations of these protocols to find the most efficient way of producing a state $T|+\rangle$ of target accuracy ϵ_T [BH12]. The optimization yields about a factor of two improvement over a scheme which utilizes only a combination of the 15-to-1 and the 10-to-2 protocols. We perform no such optimization over protocols when we compare to our own distillation protocols, because we already see a savings of more than an order of magnitude over these.

We chose to compare our protocol to resource costs incurred by the Selinger approximation protocol in conjunction with the Meier-Eastin-Knill (MEK) 10-to-2 protocol. For completeness we now provide a brief description of how the MEK protocol functions [MEK12]. The goal is to prepare a target resource state, in our case $T|+\rangle$, with some desired accuracy ϵ_T given only faulty copies of the same state with error $\epsilon > \epsilon_T$. The simplest way to prepare such a state would be to measure

an operator whose eigenstate is $T|+\rangle$, but given access to only Clifford operations this cannot be done. To circumvent this problem, more resource states of accuracy ϵ are consumed to perform the desired measurement. For the MEK protocol, the total number of resource states consumed per round is 10. Additionally, the measurement performed is a *logical* measurement on an encoded qubit (or qubits). This allows for the detection of errors during the measurement procedure and is responsible for the increased accuracy of the output resource states. If the desired logical measurement outcome has been observed, the syndrome for the code is measured and, conditioned on the syndrome being error-free, running the decoding circuit leaves two resource states with error $\mathcal{O}(\epsilon^2)$.

The code utilized by the MEK protocol is the $\llbracket 4, 2, 2 \rrbracket$ quantum error-detecting code. The distilled states are the eigenstates of H , denoted $|H\rangle$, which are related to $T|+\rangle$ by a Clifford rotation as follows:

$$|H\rangle = SH(T|+\rangle). \quad (7.12)$$

The protocol proceeds as follows:

1. Encode two (“twirled”) copies of $|H\rangle$ in the $\llbracket 4, 2, 2 \rrbracket$ code. “Twirling” is performed by the probabilistic process that applies either I or H to the state, each with probability $1/2$.
2. Perform a measurement of *logical* H_1H_2 , which for this code is the same as transversal H up to a SWAP. This measurement uses eight additional $|H\rangle$ states, which can be inferred from the identities in Fig. 1 of Ref. [MEK12].
3. If a -1 outcome is obtained for the measurement of H_1H_2 , start over. If a $+1$ outcome is obtained, measure the syndrome for the code.
4. If an error-free syndrome is reported, decode. Otherwise, start over.

5. After decoding there will be two higher fidelity $|H\rangle$ states with error $\mathcal{O}(\epsilon^2)$.

Note that the syndrome measurements can be pushed through the decoding circuit, becoming single-qubit measurements after decoding is performed.

Counting the number of resource states required to produce n_T states of accuracy ϵ_T is accomplished by numerically evaluating the recursive relationship

$$n_T(\ell) = 5n_T(\ell - 1)/a(\epsilon_\ell), \quad (7.13)$$

where $a(\epsilon_\ell)$ is the probability of the protocol accepting, given above Eq. (3) in Ref. [MEK12], ϵ_ℓ is the accuracy after ℓ rounds of distillation, and the base of the recursion is simply $n_T(0) = 5/a(\epsilon)$. Intuitively, this just says that to produce one resource state of accuracy $\mathcal{O}(\epsilon^2)$ requires on average $5/a(\epsilon)$ states of fidelity ϵ . We use this, in conjunction with Eq. (3) in [MEK12] to calculate how many resource states are required to achieve a target ϵ_T .

7.3.2 Resource analysis

As mentioned in the introduction, asymptotically the total number of operations required to approximate a Z_k gate with error ϵ' is $\mathcal{O}(\log^{\alpha+\beta+\gamma}(1/\epsilon'))$, where the exponents describe various overheads of the steps involved: fault-tolerant stabilizer operations (α), magic-state distillation (β), and quantum compiling (γ). While a good starting point, asymptotic analysis like this fails to convey the great number of elementary operations needed to implement Z_k gates, as it sweeps the (large!) constants under the rug. The explicit expression for the expected number of states used by the RISC approach to approximate Z_k to error ϵ' using $T|+\rangle$ states whose

error is ϵ is

$$n_{\text{states}}^{\text{RISC}}(Z_k, \epsilon', \epsilon) = \left[11 + 4 \log_2 \left(\frac{1}{C_{\text{qc}} \epsilon'} \right) \right] \times n_{\text{states}}^T \left(\frac{C_T \epsilon'}{n_T}, \epsilon \right), \quad (7.14)$$

where $n_{\text{states}}^T(C_T \epsilon' / n_T, \epsilon)$ is the number of $T|+\rangle$ states of error ϵ required to produce a $T|+\rangle$ state of error $C_T \epsilon' / n_T$. The idea here is to first use the results of Ref. [Sel12] to approximate Z_k to accuracy $C_{\text{qc}} \epsilon'$, and then replace each T gate in the compiled sequence with a teleportation circuit using a $T|+\rangle$ state of accuracy $C_T \epsilon' / n_T$.

To better appreciate the compiling resources needed, we consider the case when $C_{\text{qc}} = C_T = 1/2$, which balances the quality demands of quantum compiling and magic-state distillation. We give the $T|+\rangle$ state a generous error rate of $\epsilon = 10^{-4}$, which is well below the estimated threshold of $\approx 1\%$ for fault-tolerant quantum computation with surface codes [RHG07, FSG09]. The number of states $n_{\text{states}}^{\text{RISC}}$ required to synthesize Z_k with these parameters to various approximation levels are plotted in the dashed curve in Fig. 7.4. One appealing feature, especially for large values of k , is that the curve does not depend on k —the number of states needed is solely a function of the desired output precision.

7.4 Quantum CISC architecture solution

Now that we've described how to implement Z_k rotations using a quantum RISC architecture, it's natural to ask if extending the instruction set to a quantum *complex instruction set computing* architecture, or quantum CISC architecture, could provide any advantage in terms of a reduction in the required number of resource states. The point is that in any given quantum algorithm instance, one isn't interested in applying *arbitrary* gates but rather a specific set of gates, say Z_k gates up to some maximum value of k in a quantum Fourier transform. Because of this, it may make

more sense to just include those gates in the instruction set to begin with rather than compiling them from a more limited instruction set. Even if it is only feasible to include gates up to some value of $Z_{k_{\max}}$, it is reasonable to expect that the length of the resulting compiled sequences will be shorter if an arbitrary gate is required.

7.4.1 Protocol

In our protocol we consider a programmed-ancilla CISC architecture in which we pre-compile $Z_k|+\rangle$ states offline that can be used later to teleport the gate Z_k on demand via the circuit in Fig. 7.2. While the teleportation may require a Z_{k-1} gate for correction, iterating this process recursively is a negative binomial process that converges exponentially quickly—the expected number of Z rotations for any k is two: Z_k on $|+\rangle$ and Z_{k-1} after the measurement. To achieve error at most ϵ' on the teleported Z_k gate, the $Z_k|+\rangle$ state and the Z_{k-1} gate need to be performed with errors at most $C_1\epsilon'$ and $C_2\epsilon'$ respectively, where $C_1 + C_2 \leq 1$.

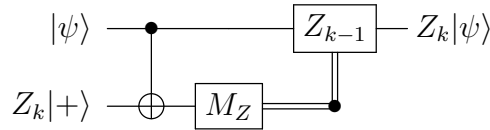


Figure 7.2: Magic-state circuit for teleporting the Z_k gate.

Our CISC approach is distinguished from previous programmed-ancilla approaches [IWPK08, JWM⁺12, DCS13] in that we distill ancilla $Z_k|+\rangle$ states directly as instructions unto themselves. This is a “top-down” approach in which some of the time auxiliary $Z_{k-1}|+\rangle$ states are needed, and even less of the time $Z_{k-2}|+\rangle$ states are needed, and so on, until we get to the point that very rarely do we need $T|+\rangle$ states. The previous approaches are “bottom-up” in that they always compile from $T|+\rangle$ states upwards until the Z_k gate is performed; some of these schemes (notably the recent one by Duclos-Cianci and Svore [DCS13]) reduce resources by including

intermediate targets, but ultimately they all start from $T|+\rangle$ preparations at the lowest level. By starting from the top, we avoid the need to probe all the way to the bottom most of the time. As we will see, this results in savings in the number of operations needed to synthesize Z_k gates.

The key to our construction is a family of shortened quantum Reed-Muller codes that are defined in Appendix B. The property of these codes that we harness here is that the $\overline{QRM}(1, k+2)$ codes admit the logical Z_k gate *transversally*, namely by applying Z_k^\dagger to each qubit independently. We know this because these codes satisfy the conditions we derived in Appendix C. Because of this transversality property, we can use the $\overline{QRM}(1, k+2)$ code to distill $Z_k^\dagger|+\rangle$ states using circuits that are essentially the same as the one used in Refs. [RHG07, FMMC12] to distill $Z_2|+\rangle$ states using the 15-qubit code, a circuit that is more compact than the one originally described by Bravyi and Kitaev [BK05]. Specifically, if we replace the encoding circuit for $\overline{QRM}(1, 4)$ with the encoding circuit for $\overline{QRM}(1, k+2)$ and replace each T with Z_k , the circuit becomes a distillation circuit for $Z_k^\dagger|+\rangle$ states. Due to the numerical results in Ref. [JOYHL13] that showed that magic states which are left untwirled can still be distilled, we also omit twirling our bare input states. As an example, we depict the distillation circuit for Z_3^\dagger in Fig. 7.3; we derived the encoding circuit for $\overline{QRM}(1, 5)$ in the figure using the methods outlined in Refs. [Got97, NC00]. We defer a proof of why these codes have the transversality property to Appendix C and instead focus on how the protocol works here. We will note here, though, that our proof generalizes the “tri-orthogonality” condition that Bravyi and Haah used to establish the transversality of T gates for their codes to a lemma in coding theory proved by Ward that we call *Ward’s Divisibility Test* [War90, Liu06].

Using the $\overline{QRM}(1, k+2)$ code to distill $Z_k|+\rangle$ states yields the following distil-

Chapter 7. Direct Distillation of a New Family of Magic States

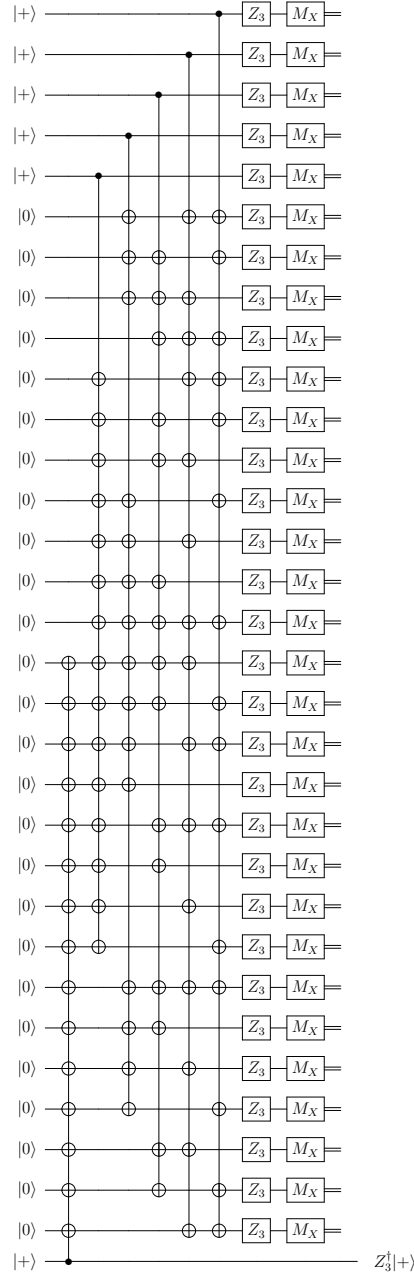


Figure 7.3: Distillation circuit for $Z_3^\dagger|+\rangle = \sqrt{T}^\dagger|+\rangle$ states; it is the 31-qubit shortened quantum Reed-Muller code's encoding circuit applied to half of a Bell state followed by the logical Z_3 gate and M_X measurement of the qubits on this encoded half. The Z_3 gates are performed using the teleportation circuit depicted in Fig. 7.2. This circuit also distills $Z_3|+\rangle$ states on $Z_3^\dagger|+\rangle$ inputs.

lation polynomial:

$$\epsilon_{\text{out}}(\epsilon) = \frac{1 - (1 - 2\epsilon)^{2^{k+1}-1} \left[2\epsilon(2^{k+2} - 1) + (1 - 2\epsilon)^{2^{k+1}} \right]}{2 \left[1 + (2^{k+2} - 1)(1 - 2\epsilon)^{2^{k+1}} \right]} \quad (7.15)$$

$$\approx (1 - 3 \cdot 2^{k+1} + 2^{2k+3}) (\epsilon^3/3 + \epsilon^4 + \mathcal{O}(\epsilon^5)). \quad (7.16)$$

Approximate values for the distillation threshold for various k are listed in Table 7.1; these are the same threshold values one would have obtained if one had used the code for distilling $Z_k|+\rangle$ to distill $Z_{k+1}|+\rangle$ instead, but the improvement in accuracy in such a case would only be to $\mathcal{O}(\epsilon)$ instead of $\mathcal{O}(\epsilon^3)$ by generalizing the method of Reichardt [Rei05].

k	$\epsilon_{\text{out}}/\epsilon^3$	ϵ_k^{th}
2	35	14.15%
3	155	6.94%
4	651	3.44%
5	2 667	1.71%
6	10 795	0.85%
7	43 435	0.43%
8	174 251	0.21%
9	698 027	0.11%
10	2 794 155	0.05%

Table 7.1: Distillation polynomials (to most significant order) and distillation thresholds for distilling $Z_k^\dagger|+\rangle$ states.

Although the distillation threshold drops as k increases, it is still larger than or comparable to the threshold of $\approx 1\%$ for fault-tolerant quantum computation with surface codes [DKLP02, RHG07, FSG09] for values of k less than or equal to 6, where it takes the value $\epsilon_6^{\text{th}} \approx 0.85\%$. This then sets a reasonable upper limit on the size of the complex instruction set one should consider for performing Z_k gates in this way; going further would place greater fidelity demands on the elementary operations than fault-tolerance does.

To achieve $\epsilon_{\text{out}} \leq \epsilon'$, one must iterate the distillation circuit

$$\ell(\epsilon', \epsilon) = \left\lceil \frac{\log \epsilon'}{\log \epsilon_{\text{out}}(\epsilon)} \right\rceil \quad (7.17)$$

times. The expected number of repetitions per iteration needed to achieve distillation success is

$$\mathbb{E}[t(\epsilon)] = \frac{2^{k+2}}{1 + (2^{k+2} - 1)(1 - 2\epsilon)^{2^{k+1}}}. \quad (7.18)$$

Unlike in the RISC protocol, in which the corrective step in the teleportation circuit added no error, in our protocol each teleportation circuit may add error in its adaptive Z_{k-1} gate. Therefore, we must implement the Z_{k-1} gate with low error using our protocol recursively. We require that the error in the corrective Z_{k-1} gate be *at most* the error in the Z_k gates in Fig. 7.3. Due to the differences in the distillation polynomials for different values of k , it turns out that the error in the Z_{k-1} gates for the corrective step is always less than the error in the Z_k gates as long as both are being implemented by magic states that have been subjected to the same number of levels of distillation using our protocols.

7.4.2 Resource analysis

Asymptotically, our CISC protocol achieves a value of $\beta = \beta_k := \log_3(2^{k+2} - 1)$ and $\gamma = 0$. The sum $\beta + \gamma$ is less than the sum of the 15-to-1 Bravyi-Kitaev magic-state distillation β and the Dawson-Nielsen compiling γ for $k \leq 9$. However, since the distillation threshold drops below 0.85% after $k = 6$, as argued earlier, it is probably wisest to stop at $k = 6$. Compared to the best values we know for β (≈ 1.58 by Refs. [BH12, JWM⁺12]) and γ (1 by Ref. [KMM13a]), our CISC protocol would appear to be only superior for $k \leq 2$. However it is important to remember, as mentioned earlier, that arguing about asymptotics in this way can be very misleading as the constants involved can be huge. Indeed, asymptotically our

Chapter 7. Direct Distillation of a New Family of Magic States

protocol is inferior in that it requires many more resource states than the Selinger + MEK scheme. However, we find that for a fairly large range of values of ϵ' and k , our protocol performs better, not becoming worse until $\epsilon' \approx 10^{-10}$ for $k = 5$ and $k = 6$ and staying comparable or better for $k = 3$ and $k = 4$ to accuracies of $\epsilon' < 10^{-70}$. Due to the discrete jumps taken in the resource requirements of our protocol, the precise analysis becomes a bit subtle. The plot in Fig. 7.4 gives a better feel for when it is favorable to use our CISC protocol.

An important difference in accounting for the resource demands of our protocol as compared to the RISC solution is that, while we incur no overhead from quantum compiling, we do have a potentially more resource intensive teleportation step. While in the RISC protocol the eventual use of a distilled magic state required only a possible Clifford correction in the teleportation procedure, in the CISC protocol we have to also account for the fact that when teleporting a $Z_k|+\rangle$ state it may be necessary to perform a Z_{k-1} correction that is accurate to *at least* the same ϵ' .

For the CISC architecture, we only allow ourselves access to $Z_k|+\rangle$ states of precision ϵ and the use of QRM-based distillation routines, even for $k = 2$. Because of this, we slightly overcount the resources required by not optimizing over the best routine to produce a $Z_2|+\rangle$ state of a desired ϵ' .

We produce our counts via the following recursive formula:

$$\begin{aligned} n_{\text{states}}^{\text{CISC}}(k, \ell) = & (2^{k+2} - 1) \left[n_{\text{states}}^{\text{CISC}}(k, \ell - 1) \right. \\ & \left. + \frac{1}{2} n_{\text{states}}^{\text{CISC}}(k - 1, \ell - 1) \right] \cdot \mathbb{E}[t(\epsilon)] \\ & + \frac{1}{2} n_{\text{states}}^{\text{CISC}}(k - 1, \ell), \end{aligned} \tag{7.19}$$

where the base of the recursion is given by

$$n_{\text{states}}^{\text{CISC}}(2, \ell) = \mathbb{E}[t(\epsilon)] 15^\ell. \tag{7.20}$$

The factor $\mathbb{E}[t(\epsilon)]$, which accounts for the need to repeat the protocol if an improper measurement outcome is obtained, is very nearly 1 for the first level of distillation given bare states of accuracy $\epsilon = 10^{-4}$, and is even closer to 1 at higher levels when the input states are accurate to even higher precision. The leading $2^{k+2} - 1$ is due to the number of $Z_k|+\rangle$ states needed at each level ℓ of distillation. The first term in the square brackets accounts for the fact that distilling a new state at level ℓ requires states already distilled to level $\ell - 1$, while the second term accounts for the fact that each of these $Z_k|+\rangle$ states from level $\ell - 1$ are injected to our circuit via teleportation and on average half will require a Z_{k-1} correction, also from distillation level $\ell - 1$. The final term counts the resources needed for the final teleportation step that consumes the distilled magic state. Here, half of the time we will need to perform a Z_{k-1} correction which must be distilled to the same level as the Z_k gate being applied.

7.5 Conclusions

Fig. 7.4 shows the results of counting resource states for the various protocols we've described. Interpreting the results is subtle, with our protocols performing better when using only one or two rounds of distillation and losing out later as the asymptotics take over. As mentioned earlier, our protocols are asymptotically much worse than the current state of the art, but for accuracies of $\epsilon' > 10^{-10}$, or indeed as low as 10^{-70} for $k = 3$ or $k = 4$, the CISC solution outperforms the RISC solution. Some of the CISC protocols show an interesting reentrant behavior, becoming better than the RISC protocol as accuracy demands increase even though they started out using more states at lower accuracies. This is due to the large jumps in accuracy when another level of distillation is used in our scheme.

The difference between the architectures at low precision demand reflects the fact

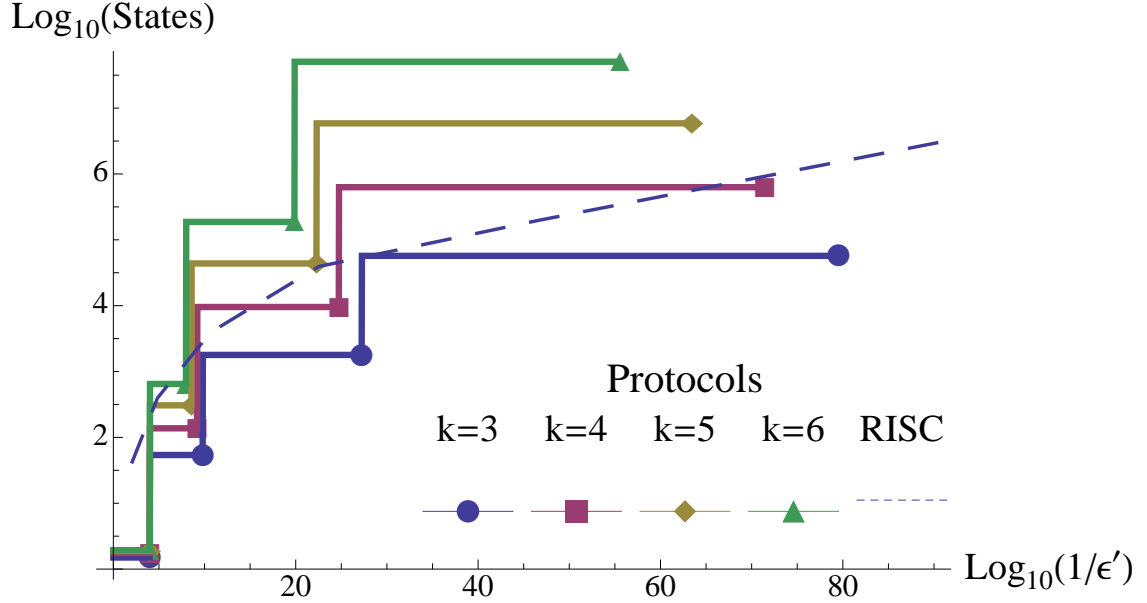


Figure 7.4: Log of the number of resource states required to synthesize the quantum $Z(\pi/2^k)$ gate as a function of the log of the inverse of the desired precision ϵ' for the RISC architecture described in the text and our CISC architecture.

that when the hardware error rate is already below this demand (*i.e.*, when $\epsilon < \epsilon'$), the only gates required by our quantum CISC architecture are those used to teleport the gate Z_k from the state $Z_k|+\rangle$ to the target state $|\psi\rangle$. The RISC architecture doesn't include the Z_k gate for $k > 2$, so it must instead use a quantum compiling strategy to synthesize Z_k from $T|+\rangle$ states.

Our CISC architecture does have some limitations. To begin, as can be seen in Fig. 7.4, as k increases, even at fixed precision demand ϵ' , the number of gates our CISC architecture uses increases. At any fixed ϵ' , even those corresponding to very low accuracies, there will be some k for which the RISC architecture uses fewer gates. However, a feature not apparent in this plot but apparent from Table 7.1 is that,

even before this happens, the distillation threshold for our CISC architecture drops to a point below the accuracy threshold for fault-tolerant quantum computation. Using our CISC architecture beyond $k = 6$ would be foolhardy, as suddenly the distillation of encoded instructions and not the capacity of the underlying code would set the experimental hardware demands at the physical level. For this reason, we advocate using our CISC architecture up to $k = 6$, and then relying on an external quantum compiling algorithm (but with a larger base instruction set than a quantum RISC architecture would have) to synthesize Z_k rotations for larger k values.

We focused on synthesizing Z_k rotations for two reasons. First, numerous quantum algorithms rely on the quantum Fourier transform, which in turn is naturally decomposed into Clifford operations and Z_k rotations. We thought it was important to focus on synthesizing transformations that arise in actual algorithms rather than operations that occur only in the abstract. Second, and more significantly, we were able to find a code family, the shortened quantum Reed-Muller codes, that we could leverage to create distillation protocols for Z_k rotations. The key enabling property these codes possess is *code divisibility*. With this insight, we generalized the “tri-orthogonality” condition of Bravyi and Haah [BH12] to a condition we call Ward’s Divisibility Test, which recognizes its analogous role in classical coding theory [War90]. We haven’t sought codes beyond the shortened quantum Reed-Muller codes that pass Ward’s Divisibility Test for admitting a Z_k -distillation protocol. However, we present and prove the correctness of this test in Appendix C in the hopes that others will find it helpful in the quest to improve quantum CISC architectures.

One of the overall messages of our work is that it is not optimal to first optimize the number of gates used to synthesize a universal instruction set and then optimize the number of universal instructions needed to synthesize a gate of interest, in this case, a Z_k gate. Instead, one can reap significant advantages by approaching this as a single optimization problem. The best conjectured asymptotic scaling

when approached as two separate problems requires a number of gates that scales as $\mathcal{O}(\log^2(1/\epsilon'))$. By approaching this as a single optimization problem, one may be able to achieve $\mathcal{O}(\log(1/\epsilon'))$ for the combined process.

The resource tradeoff space for implementing quantum operations with finite discrete instruction sets is an area ripe for investigation. Beyond just minimizing the number of resource states required to approximate transformations of interest (our focus here), one might be interested in minimizing other metrics, such as the number of gates, the number of qubits used, the depth of the approximating quantum circuit, or the size of the approximating quantum circuit (which is its depth times the number of qubits). Depending on the task at hand, one instruction set may be more suitable than another. Investigations along these lines help us better understand the limits and capabilities of finite-instruction-set quantum information processing.

Chapter 8

Summary and Outlook

8.1 Summary

In this dissertation, I've provided an introduction to fault-tolerant quantum computation using topological quantum error-correcting codes (Chapters 3 and 4), explored the relationship between two ways of using defects in the topological color codes (Chapter 5), presented a non-fault-tolerant model of quantum computation that synthesizes three prior models (Chapter 6), and described a new family of magic state distillation protocols that can implement a certain family of quantum gates with fewer resources than previous methods (Chapter 7). All of these topics fall under the general umbrella of using topologically ordered quantum systems to perform universal quantum computation, but they have varying degrees of robustness and resilience to noise.

For the models discussed in Chapter 4, fully fault-tolerant quantum computations can be implemented using only nearest neighbor interactions on a two-dimensional square lattice. A threshold of nearly 1% exists for a surface code architecture in the presence of independent depolarizing noise on each qubit, and superconducting

technologies are approaching gate fidelities that are at or below this threshold value [BKM⁺14]. In contrast, the model presented in Chapter 6, based on adiabatic interpolations between static Hamiltonians, is not fault-tolerant. This is closely related to the fact that the surface code Hamiltonian does not provide a self-correcting quantum memory due to a ground state lifetime that is constant in the size of the system [AFH09].

In both the Hamiltonian model for the surface code as well as its usual operation as an error correcting code, the injection of special ancilla states and their subsequent distillation to higher fidelities is required to enable a universal set of logical gates. Chapter 7 addresses the resource costs for such state distillation protocols by introducing a new family of protocols capable of directly distilling states that can implement single-qubit Z rotations by angles $\pi/2^k$. This direct distillation obviates the need for a quantum compiling procedure for rotations of this type and provides a savings in terms of the number of resource states needed to achieve a target gate fidelity (for a wide range of target fidelities).

8.2 Outlook

The work in this dissertation naturally leads to several interesting questions. For instance, it is natural to ask how the Hamiltonian model presented in Chapter 6 can be made fault-tolerant. In some sense, this question is equivalent to the problem of finding a self-correcting quantum memory, where entropy introduced to the system by environmental noise is naturally dissipated by the coupled environment-system dynamics. As such, it might be fruitful to apply the techniques developed in Chapter 6 to the (mildly unrealistic) four-dimensional toric code [DKLP02]—which is fully self-correcting—or to the family of three-dimensional cubic codes introduced by Haah [Haa11]—which are psuedo-self-correcting for systems up to a certain size

[BH11, BH13]. Unfortunately, there are strong no-go results prohibiting the existence of self-correcting quantum memories in two dimensions [Yos11], so searches for models using local Hamiltonians restricted to planar geometries will likely fail. Another approach is to search for clever ways to interleave the gap-protected Hamiltonian model with standard quantum error correction, although this approach begs the question as to why the standard approach isn't just fully adopted. The Hamiltonian techniques still need to prove their worth.

Follow-on questions to the work in Chapter 7 include extending the protocols to many-to-few schemes—examined for $T|+\rangle$ state distillation in Ref. [BH12]—performing a numerical optimization over the new collection of protocols to find combinations of protocols with smaller resource costs, and examining the problem of compiling over the enlarged quantum gate set provided by the ability to distill this richer set of states.

Appendices

Appendix A

Basics of Algebraic Topology

A.1 Overview

This appendix discusses one of the fundamental tools of algebraic topology, the first homology group. Before defining what a homology group is, I need to introduce a bit of machinery, but the point is that the elements of the homology group of particular spaces are in one-to-one correspondence with the logical operators of topological codes.

A.2 Introduction

Topology is the mathematical field that studies the invariant properties of spaces under continuous deformations. For the purposes of this appendix, by “spaces” I basically just mean surfaces—that is, two dimensional manifolds. However, I will not be concerned with their geometric properties.

The classic joke is that, to a topologist, a donut and a coffee mug are the same

Appendix A. Basics of Algebraic Topology

thing (Fig. A.1). This is because there exists¹ a continuous mapping of the torus

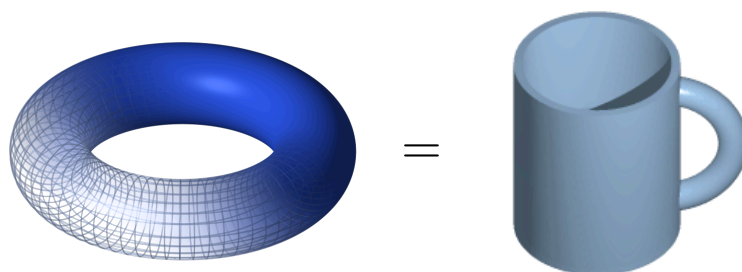


Figure A.1: The topologist’s view of the world. See the link referenced in Footnote 1 to see an animation of the deformation.

(the mathematical name for the donut) to the coffee cup. This mapping doesn’t need to tear or puncture the surface of the torus; it only needs to stretch and pull and push it around in order to complete the transformation. I find putty to be a helpful mental picture.

A.3 Homotopy and the fundamental group

The approach I will take here is to introduce techniques of surface classification that fall under the purview of a field called algebraic topology, which associates algebraic objects—like groups or chain complexes—to topological spaces. The idea is to guarantee that distinct topological spaces correspond to distinct algebraic structures. For example, the torus and the sphere should “map” (in a technical sense) to different groups.² Making these maps precise is then the goal. One way is to study the prop-

¹If you’ve never seen the animation, I suggest taking a look at the Topology Wikipedia page. Or you can visit the page for the animation directly: http://en.wikipedia.org/wiki/File:Mug_and_Torus_morph.gif.

²For a daily dose of metamathematics, think of algebraic topology as a collection of functors from the category of topological spaces to the category of groups. Objects in

Appendix A. Basics of Algebraic Topology

erties of loops in the space. Loops are just embeddings of a circle into a space X , represented by maps as

$$\gamma : S^1 \rightarrow X. \quad (\text{A.1})$$

For example, Fig. A.2 shows an embedding of a loop in two spaces that seem very similar, but which have different descriptions in terms of algebraic topology. Suppose

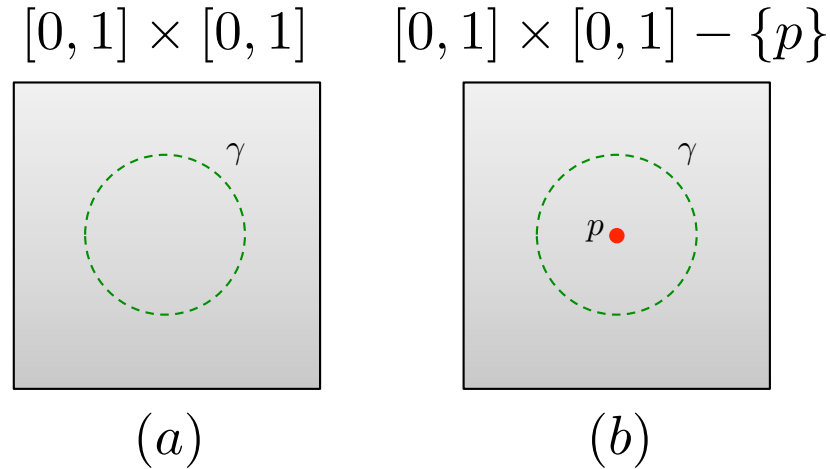


Figure A.2: Two slightly different spaces with different topologies. (a) A square X in \mathbb{R}^2 and a loop $\gamma : S^1 \rightarrow [0, 1] \times [0, 1]$. This loop can be continuously contracted to a point that is still in the space X . (b) A square X' in \mathbb{R}^2 and the same loop γ . The only difference between X and X' is that the latter has a single point removed. The loop γ can no longer be continuously deformed to a point that is also in X' .

that the loop γ in each case is given by the embedding

$$\gamma(\theta) = (r \cos \theta + 1/2, r \sin \theta + 1/2). \quad (\text{A.2})$$

where $\theta \in [0, 2\pi)$ and $r < 1/2$. Shown are two square regions taken from \mathbb{R}^2 , but one of them has a single point removed. The loop γ in (a) can be parameterized by a parameter $t \in [0, 1]$. Now define a family of such maps with decreasing radius,

the category of topological spaces are mapped to groups, and continuous homeomorphisms acting in the category of topological spaces are mapped to trivial endomorphisms in the category of groups.

Appendix A. Basics of Algebraic Topology

parameterized by t , as

$$\Phi : S^1 \times [0, 1] \rightarrow X. \quad (\text{A.3})$$

Explicitly, as coordinates in the space X , Φ_t is given by

$$\Phi_t = (r(1 - t) \cos \theta + 1/2, r(1 - t) \sin \theta + 1/2). \quad (\text{A.4})$$

It is easy to see that the original embedding is retained at $t = 0$ and that at $t = 1$ only a single point satisfies the equation. These two embeddings—and all the embeddings for intermediate values of t —are called *homotopic* to one another. The difference in the space X' is that the loop γ is no longer homotopic to a single point since the point with coordinates $(1/2, 1/2)$ is not contained in X' . It is possible to construct a group out of the different equivalence classes of homotopic loops, where two loops are equivalent if and only if they are homotopic. The group elements are then loops, and the group operation is simply concatenating one loop with another. The orientation of the loops provides a notion of a group inverse, and for this reason orientable surfaces are required. The group of loops for a space Y is called the *fundamental group*, and is labeled $\pi_1(Y)$. For the two examples above, $\pi_1(X)$ is the trivial group with only one element and $\pi_1(X')$ is the group \mathbb{Z} , the group of integers. For the torus, the fundamental group is $\mathbb{Z} \times \mathbb{Z}$.

The fundamental group thus helps to classify topological spaces: two spaces are homeomorphic—continuously deformable into one another—if and only if they have the same fundamental group. Unfortunately, the fundamental group, due to its reliance on embeddings of S^1 , does not do well distinguishing between spaces of higher dimension. There are generalizations to embeddings of higher dimensional objects, but calculations become very difficult. Due to this difficulty, mathematicians have developed tools with easier calculability at the expense of some classification refinement. I will discuss one of these tools—homology—shortly, but first I'll have to introduce the cellular complex decomposition of spaces.

A.4 Cellular complexes

A cellular complex provides a simple way of constructing topological spaces. It builds up a space X from collections of n -skeletons, consisting of objects of dimension n . The 0-skeleton is just a collection of points, each of which can also be thought of as 0-dimensional disks D^0 . The 1-skeleton is a collection of lines—1-dimensional disks D^1 —that are attached to the points in the 0-skeleton by chosen maps. The 2-skeleton is a collection of disks—the common 2-dimensional disks D^2 —attached to the 1-skeleton. This procedure can be terminated at some n -skeleton, in which case the resulting space has dimension n . For my purposes here, I will only be concerned with at most a 2-skeleton. I'll provide some examples to shine more light on the construction.

A.4.1 Cellular construction of S^1 and S^2

The cellular decomposition of the circle is very simple, and it is shown in Fig. A.3. I will label the elements of each n -skeleton as e_i^n , where each e^n is just a copy of D^n , the n -disk. The circle has a 0-skeleton with only one element: the point e_1^0 . Additionally, the 1-skeleton has the point e_1^0 as well as the line segment e_1^1 . The only missing ingredient is a map that generically operates as

$$\phi_i : \partial e_i^n \rightarrow X^{n-1} \tag{A.5}$$

for each e_i^n in the n -skeleton X^n and where the operator ∂ means “the boundary of the thing to the right.” In the particular case of the circle, there is only a single map ϕ_1 for the single element of the 1-skeleton e_1^1 . Additionally, the image of the map is only a single point in the 0-skeleton— e_1^0 —and so the map is pretty trivial. It maps the two endpoints of the line e_1^1 to the point e_1^0 . This construction is reminiscent of the construction of a circle by identifying opposite sides of a line segment. Such an

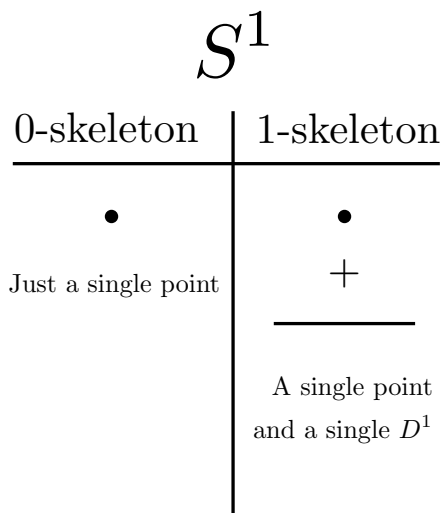


Figure A.3: The construction of the circle S^1 as a cellular complex. The 0-skeleton is a single point, and the 1-skeleton is a single point and a line segment D^1 with its endpoints attached to the point.

identification becomes manifest if we write the n -skeleton as a disjoint union of the $(n - 1)$ -skeleton and the collection of e_i^n :

$$Y = X^{n-1} \amalg_i D_i^n, \quad (\text{A.6})$$

where the symbol \amalg corresponds to the disjoint union of two sets. However, the resulting set Y does not contain any information about the gluing maps ϕ_i . The quotient space $Y/\{x \sim \phi_i(x) \ \forall \ x \in \partial e_i^n\}$ —which identifies points in Y based on the maps ϕ_i —is the real thing of interest. This quotient space is X^n .

The sphere S^2 can be constructed in a very similar fashion, depicted in Fig. A.4. The only difference between the sphere and the circle is the dimension of the disk whose boundary gets glued to the point. In fact, this is the generic way of decomposing S^n in terms of a cellular complex: a single point and a disk D^n whose boundary is glued to the point.

Graphs are typically constructed as 1-skeletons only, but here we'll be interested in the faces of graphs as well, and these are 2-dimensional objects. Thus, we will

$$S^2$$



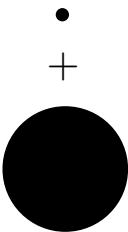
0-skeleton	1-skeleton	2-skeleton
 Just a single point	 Just a single point	 A single point and a single D^2

Figure A.4: The construction of the sphere S^2 as a cellular complex. The 0-skeleton contains only a single point, as does the 1-skeleton. The 2-skeleton consists of a single point and a single disk D^2 . The entire boundary of the disk gets glued to the single point, thus completing the construction.

imagine graphs as having the structure of a 2-skeleton, and we'll discuss the homology of such objects in the next section.

A.5 Homology

Hatcher [[Hat01](#)] has a nice example that introduces the ideas I'll need in this section, and so I'll mostly follow him here.

Homology, as I mentioned before, is an attempt to study the algebraic properties of topological spaces with tools that are easier to calculate than homotopy groups. The algebraic structures will still be groups, but they will all end up being abelian groups. This contrasts homotopy theory, which can result in nonabelian groups. (This is partly the reason for calculational difficulties.)

Appendix A. Basics of Algebraic Topology

Homology can be thought of as a way to capture the role that “holes” play in topological spaces. Alternately, since holes have boundaries, it can also be thought of as a formal study of boundaries. I will focus on the boundary aspect of things, but the example from Hatcher will be relevant to both interpretations.

Consider the graph X shown in Fig. A.5. It consists of two 0-cells e_1^0 and e_2^0

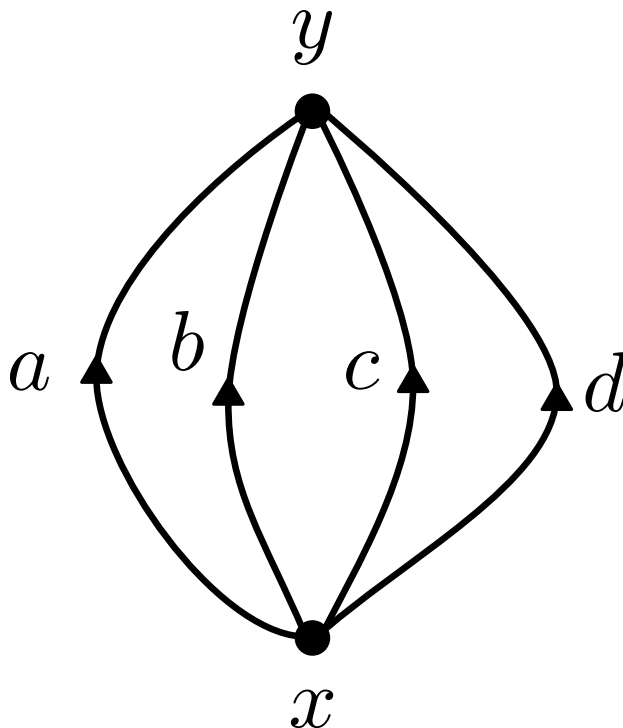


Figure A.5: A graph X composed of two points and four directed edges.

(labeled as points x and y) and four 1-cells e_1^1 , e_2^1 , e_3^1 , and e_4^1 (labeled as the directed edges a , b , c , and d). Briefly, the fundamental group for X is generated by the equivalence classes of homotopic loops—for example, the loops ab^{-1} , bc^{-1} , cd^{-1} , ac^{-1} , ad^{-1} , and bd^{-1} . The group inverses correspond to crossing an edge opposite to its orientation. It is helpful to enforce that certain loops are equivalent. For instance, the loops ab^{-1} and $b^{-1}a$ are really the same circle, but they start at different points. Enforcing an equivalence relation on these loops—equivalent loops have the same

Appendix A. Basics of Algebraic Topology

orientation corresponding to the same circle but start at different points—*abelianizes* the group of loops and allows a switch to a notation more familiar for abelian groups. The loop ab^{-1} becomes $a - b$ and the loop $ab^{-1}dc^{-1}$ becomes $a - b + d - c$.

Without a starting point, loops are now referred to as cycles, and it is these cycles that are the objects of interest in homology theory. One way to define a cycle is as a combination of edges that has no endpoints. To get at this structure of cycles, define the *boundary operator*, ∂ . This operator acts on formal linear combinations of cells in a cellular complex. Let C_0 define a free abelian group with generators e_i^0 , C_1 define a free abelian group with generators e_i^1 , and so forth. Then the boundary operator ∂_n is a linear map,

$$\partial_n : C_n \rightarrow C_{n-1} \quad (\text{A.7})$$

that returns the boundary of elements in C_n (which will be elements of C_{n-1}). It also respects the orientation of the cells, so that for our space X in Fig. A.5, $\partial_1 a = y - x$. A general linear combination of the edges is simply

$$\alpha a + \beta b + \gamma c + \delta d, \quad (\text{A.8})$$

which, when acted on by ∂_1 , yields

$$\partial_1 (\alpha a + \beta b + \gamma c + \delta d) = (\alpha + \beta + \gamma + \delta) y - (\alpha + \beta + \gamma + \delta) x. \quad (\text{A.9})$$

In order for an arbitrary linear combination of edges to be a cycle, they must have no boundary. This requires that $(\alpha + \beta + \gamma + \delta) = 0$, and for the space X in Fig. A.5 this is the whole story. The combinations $a - b$, $b - c$ and $c - d$ form a basis for all the cycles—in other words, they span $\ker \partial_1$. The fact that there are three basis elements corresponds to the fact that there are three “holes” in the space: between a and b , between b and c , and between c and d . This is almost everything needed to understand the role that homology plays in topological codes. The last piece will require a slight modification of the space X .

Appendix A. Basics of Algebraic Topology

Imagine modifying X to X' by attaching a 2-cell to the edges a and b , as shown in Fig. A.6. The intuition from homotopy theory is that the cycle $a - b$ can now be

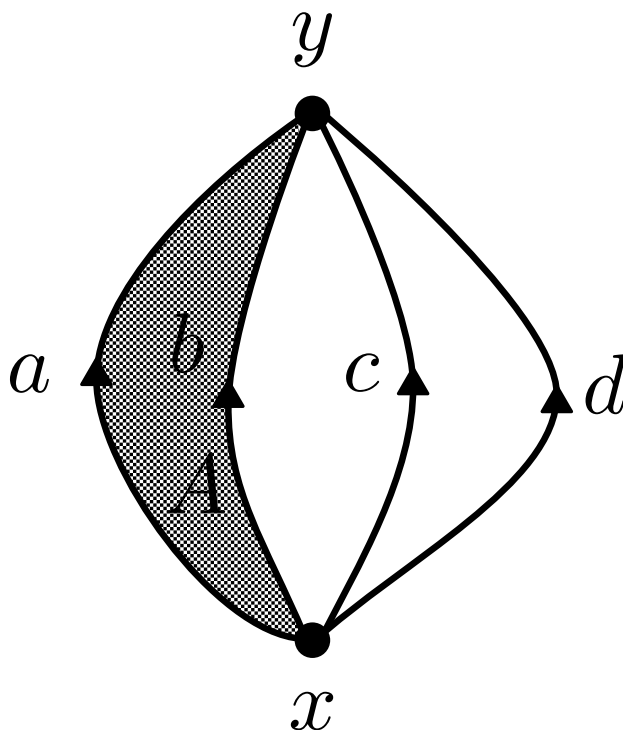


Figure A.6: The space X' , nearly the same as the space X in Fig. A.5 but with a 2-cell attached to edges a and b . While not technically just a graph anymore, the attached 2-cell captures the notion of a graph face, and I'll use this intuition to relate this structure to the graphs embedded in surfaces that are used for topological codes.

contracted to a point through the filled in space between edges a and b . It would be nice if the homology could also pick this up and only count the two remaining holes in the space. It turns out that the way to do this is to look at the boundary operator for the 2-skeleton as well. If the added 2-cell is called A and given a proper orientation, then $\partial_2 A = a - b$. The space of cycles that is really of interest is not just $\ker \partial_1$, but rather $\ker \partial_1 / \text{Im } \partial_2$ —the space of cycles that are boundaryless and that are not themselves boundaries! The identification induced by the quotient then means that, for instance, the cycle $a - b + c - d$ is the same as the cycle $c - d$. This

quotient space is referred to as the *first homology group* of the space, $H_1(X')$. The flavor of this construction should be familiar from the logical operators and stabilizer generators of the topological codes, and I will explain the precise connection in the following section.

A.6 Homology in topological codes

The connection between homology and topological codes can now be made explicit. The logical operators will end up being the elements of $H_1(X)$, equivalent up to multiplication by stabilizer generators. This freedom to multiply by generators is precisely the act of identifying cycles in $\ker \partial_1$ by equivalence up to elements of $\text{Im } \partial_2$. Consider the example of the small toric code shown in Fig. 4.4, reproduced in a modified form here in Fig. A.7 for convenience. As a cellular complex, the surface is constructed from many 0-cells (vertices), 1-cells (edges), and 2-cells (faces). The boundaries of the space (rough and smooth or, alternatively, X and Z) introduce a subtlety that has not yet been addressed. The problem is that string operators like Z_L will not be boundaryless; they very clearly have a nontrivial boundary and won't be in $\ker \partial_1$. One way to sweep this problem away is to identify the two rough boundaries with each other. Now $Z_L \in \ker \partial_1$. Another way to achieve the same goal is to simply identify all the points on the boundary with the trivial 0-cell: the identity element of C_0 , 0. This amounts to letting each of the black points in Fig. A.7 belong to a “special set” that is not counted as a boundary.

It can now be seen that the class of Z_L operators corresponds exactly to the nontrivial elements of $H_1(X)$, which in this case is just the group \mathbb{Z}_2 (for qubits—it is \mathbb{Z}_d for qudits). Strings connecting the two rough boundaries have no boundary by the definitions introduced above (or by identifying the boundaries), and the space of equivalent logical operators is precisely these strings up to multiplication by stabilizer

Appendix A. Basics of Algebraic Topology

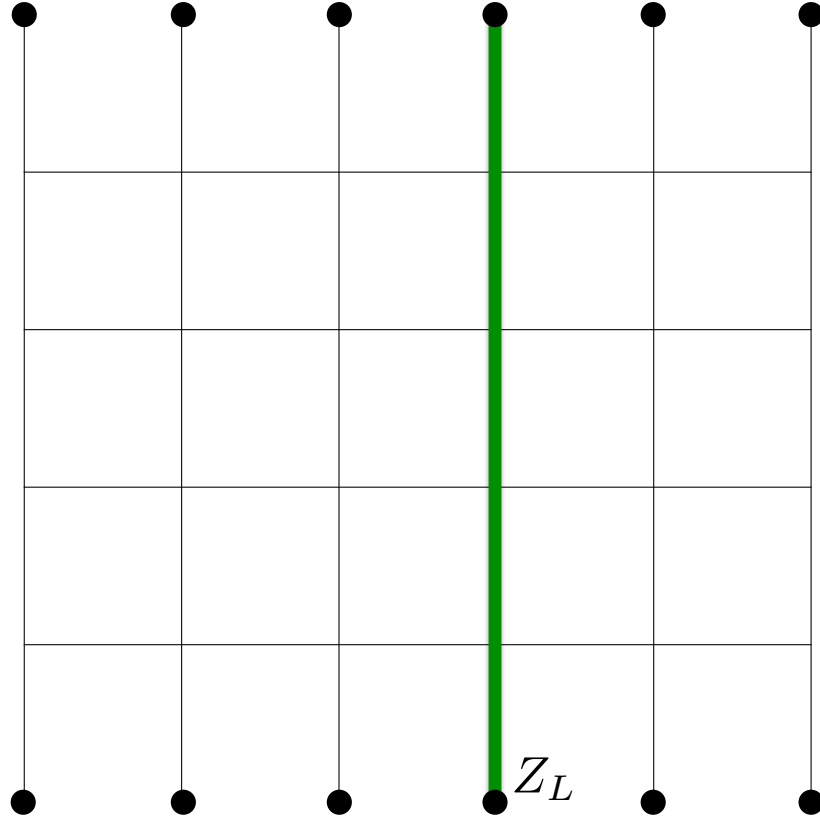


Figure A.7: Modified reproduction of Fig. 4.4 showing a single logical operator for a toric code encoding one qubit. Logical Z is a string of Z operators connecting the two rough boundaries. The points along the rough boundary are to be understood as either a “special set” of points that don’t count as boundaries or, equivalently, as being identified with the trivial element of C_0 , the null point.

generators. That is, Z_L is the only non-trivial element in $\ker \partial_1 / \text{Im } \partial_2$. Using the dual lattice, the same calculations yield the class of X_L operators.

Appendix B

Quantum Reed-Muller Codes

One of the challenges in discussing quantum Reed-Muller codes is that there is not a unique definition of what a quantum Reed-Muller code is in the literature [Ste96d, ZF97, Pre98b, BK05, SK05, CAB12]. Fortunately, there is at least a well-established definition for what a classical Reed-Muller code is. We state the definition for classical Reed-Muller codes below, confining our attention to binary codes. We refer the reader to standard texts for the definitions of supporting concepts such as Boolean monomials and $GF(2)$ [MS77].

Definition 1. The r th-order binary Reed-Muller code of length 2^m , denoted $RM(r, m)$, is the linear code over $GF(2)$ whose generator matrix is composed of row vectors corresponding to the Boolean monomials over $GF(2)^{2^m}$ of degree at most r .

Appendix B. Quantum Reed-Muller Codes

As an example, the generator matrix for the $RM(1, 4)$ code is

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}. \quad (\text{B.1})$$

From this definition, the codespace of binary Reed-Muller codes is just the space of Boolean polynomials over $GF(2)^{2^m}$ of degree at most r . It is a minor combinatoric exercise to work out that the code $RM(r, m)$ has rank $k = \sum_{i=0}^r \binom{m}{i}$ and code distance $d = 2^{m-r}$. In standard coding theory notation, we say that the code $RM(r, m)$ is an

$$[n, k, d] = \left[2^m, \sum_{i=0}^r \binom{m}{i}, 2^{m-r} \right] \quad (\text{B.2})$$

code.

It is straightforward to work out that the dual code to $RM(r, m)$ is $RM(m - r - 1, m)$. We use this to define a quantum Reed-Muller code as a CSS code composed of $RM(r, m)$ and its dual:

Definition 2. The r th-order quantum binary Reed-Muller code of length 2^m , denoted $QRM(r, m)$, is the CSS code [CS96, Ste96b] whose defining X and Z parity check matrices are the generator matrices for $RM(r, m)$ and its dual $RM(m - r - 1, m)$ respectively.

Notice that in this definition, somewhat confusingly, the quantum parity-check matrices are formed from classical *generator* matrices, not classical parity-check matrices.

We are most interested in the *shortened* quantum binary Reed-Muller codes, which we denote by $\overline{QRM}(r, m)$. These codes are formed by shortening each of the

Appendix B. Quantum Reed-Muller Codes

binary Reed-Muller codes from which it is formed. The process of shortening first punctures a code by removing a bit on which only row of the generator matrix has support and then expurgates it by removing the row in the generator matrix that had support on that bit. For the Reed-Muller codes, this corresponds to removing the first row and last column of the generator matrix when presented in standard form, as in Eq. (B.1). In essence, shortening a Reed-Muller code restricts the space of Boolean polynomials defining the code to those which have no constant term and which also satisfy $p(0) = 0$. An equivalent way of characterizing the shortened Reed-Muller code is as the even subcode of the punctured Reed-Muller code. The parameters of the resulting quantum code are $\llbracket 2^m - 1, 1 \rrbracket$. Code parameters for small Reed-Muller codes, their duals, and their shortened quantum construct are listed in Table B.1. Notice that the length of the code n does not uniquely specify which shortened quantum Reed-Muller code one is referring to for $n > 15$.

(r, m)	$(m - r - 1, m)$	$[n, k, d]$ primal	$[n, k, d]$ dual	$\llbracket n, k \rrbracket$
(0,1)	(0,1)	[2,1,2]	[2,1,2]	\emptyset
(0,2)	(1,2)	[4,1,4]	[4,3,2]	\emptyset
(0,3)	(2,3)	[8,1,8]	[8,7,2]	\emptyset
(1,3)	(1,3)	[8,4,4]	[8,4,4]	$\llbracket 7, 1 \rrbracket$
(0,4)	(3,4)	[16,1,16]	[16,15,2]	\emptyset
(1,4)	(2,4)	[16,5,8]	[16,11,4]	$\llbracket 15, 1 \rrbracket$
(0,5)	(4,5)	[32,1,32]	[32,31,2]	\emptyset
(1,5)	(3,5)	[32,6,16]	[32,26,4]	$\llbracket 31, 1 \rrbracket$
(2,5)	(2,5)	[32,16,8]	[32,16,8]	$\llbracket 31, 1 \rrbracket$
(0,6)	(5,6)	[64,1,64]	[64,63,2]	\emptyset
(1,6)	(4,6)	[64,7,32]	[64,57,4]	$\llbracket 63, 1 \rrbracket$
(2,6)	(3,6)	[64,22,32]	[64,42,8]	$\llbracket 63, 1 \rrbracket$

Table B.1: Parameters for (primal) Reed-Muller $R(r, m)$ codes, their duals $R(m - r - 1, 1)$, and their CSS-combined shortened quantum versions $\overline{QRM}(r, m)$ for small values. Shortened $R(0, m)$ codes have no X generator, so the resulting quantum codes are just classical codes; they are referred to by \emptyset in the table.

Appendix C

Criteria for a Code to Admit Transversal $Z(\pi/2^k)$ Rotations

The shortened quantum Reed-Muller codes $\overline{QRM}(1, k+2)$ admit a transversal implementation of Z_k by applying Z_k^\dagger to each qubit in the code independently. This result follows from arguments made by Campbell *et al.* in Ref. [CAB12]. Another way to see this is to note that these codes obey Theorem 1 below. We offer this alternative approach because it may be generalizable in a way that others could use to find more efficient codes that admit Z_k transversally. It also relies on a lemma (Lemma 1) that naturally generalizes an otherwise unusual criterion of “tri-orthogonality” noted by Bravyi and Haah [BH12] for the $\overline{QRM}(1, 4)$ code. We believe that this Lemma, which we call *Ward’s Divisibility Test*, makes better contact with the classical coding theory literature.

Theorem 1. A quantum $[[n, 1]]$ CSS code [CS96, Ste96b] with stabilizer generators defined by the parity check matrix $H = \text{diag}(H^X, H^Z)$ via

$$S_i^X := \bigotimes_{j=1}^n X^{H_{ij}^X} \qquad S_i^Z := \bigotimes_{j=1}^n Z^{H_{ij}^Z}, \qquad (\text{C.1})$$

Appendix C. Criteria for a Code to Admit Transversal $Z(\pi/2^k)$ Rotations

where H^X has rows v_1, \dots, v_{k+2} , implements $(Z_k)^a$ transversally if

$$\text{wt}(v_{\sigma(1)} \cdots v_{\sigma(j)}) \equiv 0 \pmod{2^{k+2-j}} \quad (\text{C.2})$$

for all $1 \leq j \leq k+2$ and all $\sigma \in \Sigma_j$, and

$$n \equiv a \pmod{2^{k+1}}, \quad (\text{C.3})$$

where ‘ \otimes ’ denotes the tensor product, ‘wt’ denotes the Hamming weight of a binary vector, ‘ Σ_j ’ denotes the permutation group on j items, and ‘ $v_1 \cdots v_j$ ’ denotes the componentwise product of v_1, \dots, v_j .

When a in this Theorem is odd, $\gcd(a, 2^{k+1}) = 1$, which means we can use an algorithm like the extended Euclidean algorithm [CLRS01] to efficiently find numbers x and y such that $ax + 2^{k+1}y = 1$. Iterating $(Z_k)^a$ x times results in a conditional phase of $\pi(1 - 2^{k+1}y)/2^k \cong \pi/2^k$; in other words, $(Z_k)^{ax} \cong Z_k$ when a is odd.

Condition (C.2) generalizes the tri-orthogonality condition of Bravyi and Haah [BH12] into a kind of $(k+1)$ -orthogonality condition. More fundamentally, we want the classical linear code generated by H^X to be a code in which every codeword has a Hamming weight divisible by 2^{k+1} . Ward studied such *divisible codes* in depth and one of his results is that 2^{k+1} -divisibility is testable by the condition of Eq. (C.2) [War90]. More explicitly, Ward’s Divisibility Test is captured by Lemma 1 below. (Ward’s result is actually more general; we use a version specialized to the binary case, as noted by Proposition 4.2 in Ref. [Liu06].)

Lemma 1 (Ward’s Divisibility Test [War90]). The binary linear code with generator matrix H^X whose row vectors are v_1, \dots, v_{k+2} is divisible by 2^{k+1} if and only if

$$2^{k+2-j} \mid \text{wt}(v_{\sigma(1)} \cdots v_{\sigma(j)}) \quad (\text{C.4})$$

for all $1 \leq j \leq k+1$ and all permutations $\sigma \in \Sigma_j$.

Appendix C. Criteria for a Code to Admit Transversal $Z(\pi/2^k)$ Rotations

While Ward's Divisibility Test has the advantage of being an explicit algorithm for testing divisibility, it is not particularly efficient, as it takes a time that is exponential in k to execute. For codes with a high degree of structure, such as the shortened $\overline{RM}(1, k+2)$ Reed-Muller codes, demonstrating 2^{k+1} divisibility is much simpler, as noted in Ref. [Liu06].

Proof of Theorem 1. By Ward's Divisibility Test, every vector v in the rowspan \mathcal{L} of H^X has a Hamming weight divisible by 2^{k+1} . Since the logical $|0\rangle$ for the code is $|\bar{0}\rangle := \sum_{v \in \mathcal{L}} |v\rangle$ (ignoring normalization), the action of transversal Z_k on $|\bar{0}\rangle$ is

$$Z_k^{\otimes n} |\bar{0}\rangle = \sum_{v \in \mathcal{L}} Z_k^{\otimes n} |v\rangle \quad (\text{C.5})$$

$$= \sum_{v \in \mathcal{L}} \left(e^{i\pi/2^k} \right)^{|v|} |v\rangle \quad (\text{C.6})$$

$$= \sum_{v \in \mathcal{L}} |v\rangle \quad (\text{C.7})$$

$$= |\bar{0}\rangle. \quad (\text{C.8})$$

Similarly, using Eq. (C.3), the action of transversal Z_k on (unnormalized) $|\bar{1}\rangle = \overline{X}|\bar{0}\rangle$ is

$$Z_k^{\otimes n} |\bar{1}\rangle = Z_k^{\otimes n} \overline{X} |\bar{0}\rangle \quad (\text{C.9})$$

$$= \sum_{v \in \mathcal{L}} Z_k^{\otimes n} \overline{X} |v\rangle \quad (\text{C.10})$$

$$= \sum_{v \in \mathcal{L}} Z_k^{\otimes n} |v \oplus \mathbf{1}\rangle \quad (\text{C.11})$$

$$= \sum_{v \in \mathcal{L}} \left(e^{i\pi/2^k} \right)^{n-|v|} |v \oplus \mathbf{1}\rangle \quad (\text{C.12})$$

$$= \sum_{v \in \mathcal{L}} \left(e^{i\pi a/2^k} \right) |v \oplus \mathbf{1}\rangle \quad (\text{C.13})$$

$$= e^{i\pi a/2^k} |\bar{1}\rangle, \quad (\text{C.14})$$

where $\mathbf{1} := (1, \dots, 1)$ denotes the all-ones vector, whose appearance comes from the fact that up to local qubit basis changes, $\overline{X} = X^{\otimes n}$ for all CSS codes. These

Appendix C. Criteria for a Code to Admit Transversal $Z(\pi/2^k)$ Rotations

actions of $Z^{\otimes n}$ replicate $(Z_k)^a$ on the logical basis, and therefore Z_k implements $(Z_k)^a$ transversally. \square

Appendix D

Computational Tools for Code Deformation

D.1 Code deformation

Code deformation—whether performed by measurements or via adiabatic Hamiltonian deformations—amounts to a sequential update to the set of stabilizer generators of a quantum code. Using only group theory and a minimum input from quantum mechanics, it is possible to determine the effect on the generating set and the logical operators when a measurement is made.

This discussion is framed by the setting of stabilizer codes. Consider an $[[n, k, d]]$ stabilizer code with a generating set \mathcal{S} and a set of logical operators \mathcal{L} . Recall that the operators in these sets are all Pauli operators and that for every $S_i, S_j \in \mathcal{S}$

$$[S_i, S_j] = 0, \tag{D.1}$$

and

$$[S_i, L_j] = 0 \tag{D.2}$$

Appendix D. Computational Tools for Code Deformation

for all $L_j \in \mathcal{L}$. The only operators that don't commute are the representatives of $X_i, Z_i \in \mathcal{L}$ for a single qubit.

Consider making a measurement of a Pauli operator M . After performing the measurement, the resulting n -qubit state $|\psi\rangle$, may or may not be in the original codespace, but it is in a definite eigenstate of M . Since M is a Pauli operator, its eigenvalue is either $+1$ or -1 . If M commutes with all the elements of \mathcal{S} and \mathcal{L} , there is only one possibility: an element of the stabilizer group has been measured, in which case the action on the codespace is trivial. The commuting case is uninteresting and does not lead to a modification of either \mathcal{S} or \mathcal{L} .

If M does not commute with elements in either or both of \mathcal{S} and \mathcal{L} , then something has to give. The state $|\psi\rangle$ cannot be a simultaneous eigenstate of M and the anti-commuting operators. If M only anti-commutes only with elements of \mathcal{L} , then M is equivalent to a logical measurement of one—or several—of the logical qubits. This also requires no modification of either sets; it simply projects the codespace into a definite state of the measured logical qubits.

However, if M anti-commutes with elements of \mathcal{S} , then the generating set needs to be modified. Recall that the generating set provides the conditions that define the codespace. Thus, if M does not commute with all the generators, a new generating set—one that incorporates the fact that M has essentially been promoted to a generator—must be defined. The generating set is thus *deformed* by the measurement, and the procedure is called code deformation.

How is the new generating set chosen? Call \mathcal{A} the set of operators in \mathcal{S} that anti-commute with M . The new generating set \mathcal{S}' —initially empty—can then be constructed in the following way:

1. Add M to \mathcal{S}' .

Appendix D. Computational Tools for Code Deformation

2. Choose one of the elements of \mathcal{A} and call it A .
3. For all the other elements of \mathcal{A} , multiply them by A and add the result to \mathcal{S}' .
4. Add all of the elements in $\mathcal{S} - \mathcal{A}$ to \mathcal{S}' .

Operators in \mathcal{L} might also not commute with M . Replace all of these with the product of the A chosen above and themselves. This trick—creating commuting operators by multiplying together two anti-commuting operators—leverages the properties of the Pauli group introduced in Sec. 2.2. The new generating set \mathcal{S}' defines a commutative Pauli subgroup and also commutes with the modified logical operators. It defines a new $[[n, k, d]]$ quantum stabilizer code.

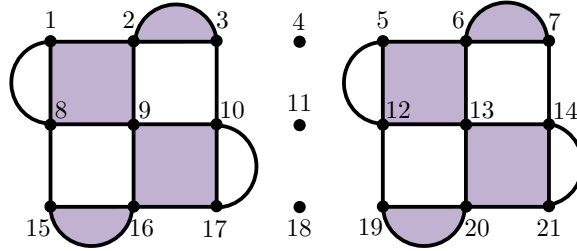


Figure D.1: The surface codes corresponding to the example given in the code listing. The surface code on the left is going to be “merged” with the surface code on the right, effecting a joint logical measurement. This figure depicts the surface code in its medialized form, with the light purple face representing X -type stabilizer generators, the white faces representing Z -type stabilizer generators, and the qubits on the vertices.

D.2 Automated code deformation

The code provided in Sec. D.3 below simulates a step in the code deformation techniques introduced in Ref. [HFDv12] for performing a $CNOT$ gate between two qubits encoded in surface codes (rather than by defect braiding). However, it has more

Appendix D. Computational Tools for Code Deformation

general application and can simulate arbitrary deformations of CSS codes. The numbering scheme used in the code example is shown in Fig. D.1.

D.3 Automated code deformation code listing

```
1 # The idea here is to automate the process of code deformation performed
# by making measurements and updating the set of stabilizer generators
3 # and logical operators.
#
5 #
# My idea is to store the generators as key:value pairs in a dictionary. E.g.,
7 #
# 'S1':[ 'Z', '2', '3', '5', '6']
9 #
# Logical operators are stored similarly .
11
13 stab_gens = {}
logical_ops = {}
15 measurements = []
17 mode = 'toric'
19 if mode == 'toric':
# I'll write this with the built in case of merging two toric codes (in
21 # the Bombin picture).
23 # Below are the 8 stabilizers generators for the code "on the left " in
# the merging picture.
25 stab_gens['S1'] = ['Z', '1', '8']
stab_gens['S2'] = ['X', '1', '2', '8', '9']
27 stab_gens['S3'] = ['X', '2', '3']
stab_gens['S4'] = ['Z', '2', '3', '9', '10']
29 stab_gens['S5'] = ['Z', '8', '9', '15', '16']
stab_gens['S6'] = ['X', '15', '16']
31 stab_gens['S7'] = ['X', '9', '10', '16', '17']
stab_gens['S8'] = ['Z', '10', '17']
33
# Below are the 8 stabilizer generators for the code "on the right ."
35 stab_gens['S9'] = ['Z', '5', '12']
```

Appendix D. Computational Tools for Code Deformation

```

39 stab_gens['S10'] = ['X', '5', '6', '12', '13']
stab_gens['S11'] = ['X', '6', '7']
stab_gens['S12'] = ['Z', '6', '7', '13', '14']
41 stab_gens['S13'] = ['Z', '12', '13', '19', '20']
stab_gens['S14'] = ['X', '19', '20']
43 stab_gens['S15'] = ['X', '13', '14', '20', '21']
stab_gens['S16'] = ['Z', '14', '21']

45 # Below are the stabilizer generators for the *added* qubits along the merge
47 # boundary.
stab_gens['S17'] = ['Z', '4']
49 stab_gens['S18'] = ['Z', '11']
stab_gens['S19'] = ['Z', '18']

51 # The input of the above could obviously be aided by a helper script which
53 # reads a text file in a specified format and generates the stab_gens
# dictionary.

55 # Below are the two sets of logical operators for the two codes to be merged.
57 logical_ops['X1'] = ['X', '5', '12', '19']
logical_ops['Z1'] = ['Z', '5', '6', '7']
59 logical_ops['X2'] = ['X', '3', '10', '17']
logical_ops['Z2'] = ['Z', '1', '2', '3']

61 # Below is the set of measurements we will be performing to deform the code.
63 measurements.append(['Z', '4', '5', '11', '12'])
measurements.append(['Z', '10', '11', '17', '18'])
65 measurements.append(['Z', '5', '6', '7'])
measurements.append(['X', '3', '4', '10', '11'])
67 measurements.append(['X', '4', '5'])
measurements.append(['X', '17', '18'])
69 measurements.append(['X', '11', '12', '18', '19'])

71 # Some functions to help in doing the updates.
73
75 def common_elements(list1, list2):
# Returns the set theoretic intersection of two lists.
return list(set(list1) & set(list2))

77 # Now for the actual code. The idea is the loop through the dictionary of
79 # measurements, updating the stab_gens and logical_ops dictionaries as
# needed.

```

Appendix D. Computational Tools for Code Deformation

```
81 for meas in measurements:
    print "\n\n=====Performing measurement of " + str(meas) + "====="
85     # First, determine whether we are measuring an X-type or Z-type
    # operator and which qubits it has support on.
87     meas_basis = meas[0]
    meas_support = meas[1:]
89
    # Next, loop through the stab_gens dictionary to figure out which
91     # elements anticommute with the measurement we are making.
    replaced = {}
93     print "\n-----Stabilizer modifications-----\n"
    for stab in stab_gens:
95
        stab_basis = stab_gens[stab][0]
97
        if stab_basis != meas_basis:
99
            stab_support = stab_gens[stab][1:]
            overlap = common_elements(stab_support, meas_support)
101
            if overlap and (len(overlap) % 2) != 0:
103
                # A pythonic way of ensuring that the returned list is
                # not empty.
105
                if not replaced:
107
109
                    print "Replacing " + str(stab_gens[stab]) + " with " + str(meas)
111                    replaced[stab] = stab_gens[stab]
                    replaced_support = replaced[stab][1:]
113                    stab_gens[stab] = meas
115
                else :
117
                    # Here's where the fancy replacement rules happen.
                    # The Python operator "^" performs the symmetric difference between two sets.
                    # Here it is essentially doing the XOR (or addition modulo 2) for us.
                    new_gen_support = list(set(stab_support) ^ set(replaced_support))
119
                    print "Modifying " + str(stab_gens[stab]) + " to " + str([stab_basis] + new_gen_support)
                    stab_gens[stab][1:] = new_gen_support
123
            if not replaced:
125                print str(meas) + " commuted with all stabilizer generators!"
```

Appendix D. Computational Tools for Code Deformation

```
129     print "\n-----Logical operator modifications-----\n"

131     for log in logical_ops:

133         log_basis = logical_ops[log][0]

135         if log_basis != meas_basis:

137             log_support = logical_ops[log][1:]
            log_overlap = common_elements(log_support, meas_support)

139             if replaced and log_overlap and (len(log_overlap) % 2) != 0:
                new_log_op_support = list(set(log_support) ^ set(replaced_support))
                print "Modifying " + str(logical_ops[log]) + " to " + str([log_basis] + new_log_op_support)
141                 logical_ops[log][1:] = new_log_op_support

143             if not replaced and log_overlap and (len(log_overlap) % 2) != 0:
                print "WARNING: NUMBER OF LOGICAL QUBITS IS BEING REDUCED!"
145                 print "Replacing " + str(logical_ops[log]) + " with " + str(meas)
                replaced[log] = logical_ops[log]
147                 replaced_support = replaced[log][1:]
                logical_ops[log] = meas

149             if not replaced:

151                 print str(meas) + " commuted with all logical operators!"

153     print "\n\n\nStabilizer generators"
    for gens in stab_gens:
155         print stab_gens[gens]

157     print "Logical operators"
    for log in logical_ops:
159         print log, logical_ops[log]

161
```

References

- [AAN09] M. H. S. Amin, Dmitri V. Averin, and James A. Nesteroff. Decoherence in adiabatic quantum computation. *Phys. Rev. A*, 79:022107, 2009, arXiv:0708.0384.
- [Aar13] Scott Aaronson. The ghost in the quantum Turing machines, 2013, arXiv:1306.0159.
- [ABO97] D. Aharonov and M. Ben-Or. Fault tolerant quantum computation with constant error. In F. Tom Leighton and Peter Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing*, pages 176–188, El Paso, TX, USA, 1997. ACM Press, New York, quant-ph/9611025. See also extended version [ABO99].
- [ABO99] Dorit Aharonov and Michael Ben-Or. Fault tolerant quantum computation with constant error rate, 1999, quant-ph/9906129. See also condensed version [ABO97].
- [AFH09] R. Alicki, M. Fannes, and M. Horodecki. On thermalization in Kitaev’s 2D model. *J. Phys. A: Math. Theo.*, 42(6):065303, 2009, arXiv:0810.4584.
- [AGP06] Panos Aliferis, Daniel Gottesman, and John Preskill. Quantum accuracy threshold for concatenated distance-3 codes. *Quant. Inf. Comp.*, 6(2):97–165, 2006, quant-ph/0504218.
- [AHHH10] R. Alicki, M. Horodecki, P. Horodecki, and R. Horodecki. On thermal stability of topological qubit in Kitaev’s 4D models. *Open Syst. Inf. Dyn.*, 17:1, 2010.
- [Ahn04] Charlene Sonja Ahn. *Extending quantum error correction: new continuous measurement protocols and improved fault-tolerant overhead*. PhD thesis, Caltech, 2004.

References

- [AJN06] S. Ashhab, J. R. Johansson, and Franco Nori. Decoherence in a scalable adiabatic quantum computers. *Phys. Rev. A*, 74:052330, 2006, quant-ph/0608212.
- [ÅKS05a] Johan Åberg, David Kult, and Erik Sjöqvist. The quantum adiabatic search with decoherence in the instantaneous energy eigenbasis. *Phys. Rev. A*, 72:042317, 2005, quant-ph/0507010.
- [ÅKS05b] Johan Åberg, David Kult, and Erik Sjöqvist. Robustness of the adiabatic quantum search. *Phys. Rev. A*, 71:060312(R), 2005, quant-ph/0412124.
- [AMMR13] Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Trans. Comp.-A. Des. Int. Cir. Sys.*, 32:818–830, 2013, arXiv:1206.0758.
- [And11] Jonas T. Anderson. Homological stabilizer codes, 2011, arXiv:1107.3502.
- [And12] Jonas T. Anderson. On the power of reusable magic states, 2012, arXiv:1205.0289.
- [ATA09] M. H. S. Amin, C. J. S. Truncik, and D. V. Averin. Role of single qubit decoherence time in adiabatic quantum computation. *Phys. Rev. A*, 80:022303, 2009, arXiv:0803.1196.
- [AvK⁺04] Dorit Aharonov, Wim van Dam, Julia Kempe, Zeph Landau, Seth Lloyd, and Oded Regev. Adiabatic quantum computation is equivalent to standard quantum computation. In Deeber Azada, editor, *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science*, pages 42–51, Oct. 17–19, Rome, IT, 2004. IEEE, IEEE Press, Los Alamitos, CA, quant-ph/0405098.
- [Bac06] Dave Bacon. Operator quantum error-correcting subsystems for self-correcting quantum memories. *Phys. Rev. A*, 73(1):012340, Jan. 2006, quant-ph/0506023v4.
- [Bac08] D. Bacon. Stability of quantum concatenated code Hamiltonians. *Phys. Rev. A*, 78:042324, 2008, arXiv:0806.2160.
- [BB85] Charles H. Bennett and Gilles Brassard. Quantum public key distribution. *IBM Technical Disclosure Bulletin*, 28(7):3153–3163, Dec. 1985.
- [BBC⁺93] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. Wootters. Teleporting an unknown quantum state via dual classical and EPR channels. *Phys. Rev. Lett.*, 70:1895–1899, 1993.

References

- [BBW01] D. Bacon, K. R. Brown, and K. B. Whaley. Coherence-preserving quantum bits. *Phys. Rev. Lett.*, 87:247902, 2001, quant-ph/0012018.
- [BDGP12] H. Bombin, Guillaume Duclos-Cianci, and David Poulin. Universal topological phase of 2D stabilizer codes. *New J. Phys.*, 14:073048, 2012.
- [Bel64] J. S. Bell. On the Einstein-Podolsky-Rosen paradox. *Physics*, 1:195–200, 1964. Reprinted in [Bel87].
- [Bel87] J. S. Bell. *Speakable and Unspeakable in Quantum Mechanics*. Cambridge University Press, Cambridge / New York, 1 edition, 1987.
- [Ben73] Charles H. Bennett. Logical reversibility of computation. *IBM J. Res. Develop.*, 17(6):525–532, Nov. 1973.
- [BF09] D. Bacon and S. T. Flammia. Adiabatic gate teleportation. *Phys. Rev. Lett.*, 103:120504, 2009, arXiv:0905.0901.
- [BF10] Dave Bacon and Steven T. Flammia. Adiabatic cluster-state quantum computing. *Phys. Rev. A*, 82(3):030303(R), 2010, arXiv:0912.2098.
- [BFN08] Parsa Bonderson, Michael Freedman, and Chetan Nayak. Measurement-only topological quantum computation. *Phys. Rev. Lett.*, 101:010501, Jun 2008, arXiv:0802.0279.
- [BFN09] Parsa Bonderson, Michael Freedman, and Chetan Nayak. Measurement-only topological quantum computation via anyonic interferometry. *Ann. Phys.*, 324(4):787–826, April 2009, arXiv:0808.1933.
- [BH11] Sergey Bravyi and Jeongwan Haah. On the energy landscape of 3D spin Hamiltonians. *Phys. Rev. Lett.*, 107:150504, 2011.
- [BH12] Sergey Bravyi and Jeongwan Haah. Magic state distillation with low overhead. *Phys. Rev. A*, 86:052329, 2012, 1209.2426.
- [BH13] Sergey Bravyi and Jeongwan Haah. Analytic and numerical demonstration of quantum self-correction in the 3D Cubic Code. *Phys. Rev. Lett.*, 111:200501, 2013.
- [BHM10] S. Bravyi, M. Hastings, and S. Michalakis. Topological quantum order: stability under local perturbations. *J. Math. Phys.*, 51(9):093512, Sep 2010, arXiv:1001.0344.
- [BK98] S. B. Bravyi and A. Yu. Kitaev. Quantum codes on a lattice with boundary, 1998, quant-ph/9810052.

References

- [BK05] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal Clifford gates and noisy ancillas. *Phys. Rev. A*, 71:022316, Feb. 2005, quant-ph/0403025.
- [BKM⁺14] R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, T. C. White, J. Mutus, A. G. Fowler, B. Campbell, Y. Chen, Z. Chen, B. Chiaro, A. Dunsworth, C. Neill, P. O'Malley, P. Roushan, A. Vainsencher, J. Wenner, A. N. Korotkov, A. N. Cleland, and John M. Martinis. Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature*, 508:500–503, 2014.
- [BMD06] H. Bombin and M. A. Martin-Delgado. Topological quantum distillation. *Phys. Rev. Lett.*, 97:180501, Nov. 2006, quant-ph/0605138.
- [BMD07] H. Bombin and M. A. Martin-Delgado. Exact topological quantum order in $d = 3$ and beyond: Branyons and brane-net condensates. *Phys. Rev. B*, 75:075103, Feb 2007, cond-mat/0607736.
- [BMD09] H. Bombin and M.A. Martin-Delgado. Quantum measurements and gates by code deformation. *J. Phys. A: Math. Theor.*, 42(9):095302, Mar. 2009, arXiv:0704.2540.
- [Bom10] H. Bombin. Topological order with a twist: Ising anyons from an Abelian model. *Phys. Rev. Lett.*, 105:030403, 2010.
- [Bon13] Parsa Bonderson. Measurement-only topological quantum computation via tunable interactions. *Phys. Rev. B*, 87:035113, 2013.
- [BS12] Alex Bocharov and Krysta Svore. A depth-optimal canonical form for single-qubit quantum circuits. *Phys. Rev. Lett.*, 109:190501, 2012, arXiv:1206.3223.
- [BT09] Sergey Bravyi and Barbara Terhal. A no-go theorem for a two-dimensional self-correcting quantum memory based on stabilizer codes. *New J. Phys.*, 11:043029, 2009, arXiv:0810.1983.
- [BW92] Charles H. Bennett and Stephen J. Wiesner. Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states. *Phys. Rev. Lett.*, 69(20):2881–2884, Nov. 1992.
- [BW00] Jeff P. Barnes and Warren S. Warren. Automatic quantum error correction. *Phys. Rev. Lett.*, 85(4):856–859, Jul. 2000, quant-ph/9912104.

References

- [BW03] James Brink and Zhenghan Wang. On Freedman’s lattice models for topological phases. *Quant. Inf. Proc.*, 2(1–2):81–96, 2003, math-ph/0303018.
- [CAB12] Earl T. Campbell, Hussain Anwar, and Dan E. Browne. Magic state distillation in all prime dimensions using quantum Reed-Muller codes. *Phys. Rev. X*, 2:041021, 2012, arXiv:1205.3104.
- [CDT09] Andrew W. Cross, David P. DiVincenzo, and Barbara M. Terhal. Comparative code study for quantum fault tolerance. *Quant. Inf. Comp.*, 9(7–8):541–572, 2009, arXiv:0711.1556.
- [CEMM98] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca. Quantum algorithms revisited. *Proc. Roy. Soc. London A*, 454(1969):339–354, Jan. 1998, quant-ph/9708016.
- [CFC⁺14] Joshua Combes, Christopher Ferrie, Chris Cesare, Markus Tiersch, G. J. Milburn, Hans J. Briegel, and Carlton M. Caves. In-situ characterization of quantum devices with error correction, 2014, arXiv:1405.5656.
- [CFP01] Andrew M. Childs, Edward Farhi, and John Preskill. Robustness of adiabatic quantum computation. *Phys. Rev. A*, 65(1):012322, Dec. 2001, quant-ph/0108048.
- [CLB⁺14] Chris Cesare, Andrew J. Landahl, Dave Bacon, Steven T. Flammia, and Alice Neels. Adiabatic topological quantum computing, 2014, arXiv:1406.2690.
- [CLBT10] Stefano Chesi, Daniel Loss, Sergey Bravyi, and Barbara M. Terhal. Thermodynamic stability criteria for a quantum memory based on stabilizer and subsystem codes. *New J. Phys.*, 12:025013, 2010.
- [CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2001.
- [Col06] Graham P. Collins. Computing with Quantum Knots. *Sci. Am.*, April:56–63, 2006.
- [CRSS97] A. R. Calderbank, E. M. Rains, P. W. Shor, and N. J. A. Sloane. Quantum error correction and orthogonal geometry. *Phys. Rev. Lett.*, 78(3):405–408, Jan. 1997, quant-ph/9605005.
- [CS96] A. R. Calderbank and Peter W. Shor. Good quantum error-correcting codes exist. *Phys. Rev. A*, 54(2):1098–1105, Aug. 1996, quant-ph/9512032.

References

- [dB94] Mark de Wild Propitius and F. Alexander Bais. Discrete gauge theories. In *Particles and Fields '94 CRM-CAP Summer School*, 1994.
- [dBP10] Inez de Vega, Marie Carmen Bañuls, and A. Pérez. Effects of dissipation on an adiabatic quantum search algorithm. *New J. Phys.*, 12:123010, Dec. 2010, arXiv:1006.0461.
- [DCP10] Guillaume Duclos-Cianci and David Poulin. Fast decoders for topological quantum codes. *Phys. Rev. Lett.*, 104:050504, 2010, arXiv:0911.0581.
- [DCP14] Guillaume Duclos-Cianci and David Poulin. Reducing the quantum computing overhead with complex gate distillation, 2014, arXiv:1403.5280.
- [DCS13] Guillaume Duclos-Cianci and Krysta M. Svore. A state distillation protocol to implement arbitrary single-qubit rotations. *Phys. Rev. A*, 88:042325, 2013, arXiv:1210.1980.
- [Den03] Eric Dennis. *Purifying Quantum States: Quantum and Classical Algorithms*. PhD thesis, University of California, Santa Barbara, September 2003.
- [DKLP02] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *J. Math. Phys.*, 43(9):4452–4505, Sep. 2002, quant-ph/0110143.
- [DN06] Christopher M. Dawson and Michael A. Nielsen. The Solovay-Kitaev algorithm. *Quant. Inf. Comp.*, 6(1):81–95, 2006, quant-ph/0505030.
- [EK09] Bryan Eastin and Emmanuel Knill. Restrictions on transversal encoded quantum gate sets. *Phys. Rev. Lett.*, 102(11):110502, 2009, arXiv:0811.4262.
- [ES12] Zachary W. E. Evans and Ashley Stephens. Optimal decoding in fault-tolerant concatenated quantum error correction. *Quant. Inf. Proc.*, 11:1511, 2012, arXiv:0902.4506.
- [FG98] Edward Farhi and Sam Gutmann. Analog analogue of a digital quantum computation. *Phys. Rev. A*, 57(4):2403–2406, Apr. 1998, quant-ph/9612026.
- [FGG⁺01] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, Joshua Lapan, Andrew Lundgren, and Daniel Preda. A quantum adiabatic evolution applied to random instances of an NP-complete problem. *Science*, 292(5516):472–475, Apr. 2001, quant-ph/0104129.

References

- [FGGS00] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. Quantum computation by adiabatic evolution, 2000, quant-ph/0001106.
- [FHHW09] Steven T. Flammia, Alioscia Hamma, Taylor L. Hughes, and Xiao-Gang Wen. Topological entanglement Rényi entropy and reduced density matrix structure. *Phys. Rev. Lett.*, 103(26):261601, 2009, arXiv:0909.3305.
- [FKW02] Michael H. Freedman, Alexei Kitaev, and Zhenghan Wang. Simulation of topological field theories by quantum computers. *Comm. Math. Phys.*, 227(3):587–603, Jun. 2002, quant-ph/0001071.
- [FLW02] Michael H. Freedman, Michael Larsen, and Zhenghan Wang. A modular functor which is universal for quantum computation. *Comm. Math. Phys.*, 227(3):605–622, Jun. 2002, quant-ph/0001108.
- [FM14] Austin G. Fowler and John M. Martinis. Quantifying the effects of local many-qubit errors and non-local two-qubit errors on the surface code. *Phys. Rev. A*, 89:032316, 2014.
- [FMMC12] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A*, 86(3):032324, 2012, arXiv:1208.0928.
- [FNS05] M.H. Freedman, C. Nayak, and K. Shtengel. Extended Hubbard model with ring exchange: A route to non-Abelian topological phase. *Phys. Rev. Lett.*, 94:066401, 2005, cond-mat/0309120. E-print differs significantly.
- [FNW06] M. Freedman, C. Nayak, and K. Walker. Towards universal topological quantum computation in the $\nu = (5/2)$ fractional quantum Hall state. *Phys. Rev. B*, 73(24):245307, 2006, cond-mat/0512066.
- [Fow05] Austin Greig Fowler. *Towards large-scale quantum computation*. PhD thesis, University of Melbourne, 2005, quant-ph/0506126.
- [Fow11a] Austin Fowler. Constructing arbitrary Steane code single logical qubit fault-tolerant gates. *Quant. Inf. Comp.*, 11(9&10):867–873, Sep. 2011, quant-ph/0411206.
- [Fow11b] Austin G. Fowler. Two-dimensional color-code quantum computation. *Phys. Rev. A*, 83:042310, 2011, arXiv:0806.4827.
- [Fre03] Michael H. Freedman. A magnetic model with a possible Chern-Simons phase (with an appendix by F. Goodman and H. Wenzl). *Comm. Math. Phys.*, 234(1):129–183, Mar. 2003, quant-ph/0110060.

References

- [FSG09] Austin G. Fowler, Ashley M. Stephens, and Peter Groszkowski. High-threshold universal quantum computation on the surface code. *Phys. Rev. A*, 80:052312, 2009, arXiv:0803.0272.
- [Gai06] Frank Gaitan. Simulation of quantum adiabatic search in the presence of noise. *Int. J. Quant. Inf.*, 4(5):843–870, Oct. 2006, quant-ph/0601116.
- [GLN05] Alexei Gilchrist, Nathan K. Langford, and Michael A. Nielsen. Distance measures to compare real and ideal quantum processes. *Phys. Rev. A*, 71:062310, 2005, quant-ph/0408063.
- [Got97] Daniel Gottesman. *Stabilizer codes and quantum error correction*. PhD thesis, Caltech, 1997, quant-ph/9705052.
- [Got98] Daniel Gottesman. Theory of fault-tolerant quantum computation. *Phys. Rev. A*, 57(1):127–137, Jan. 1998, quant-ph/9702029.
- [Got99] Daniel Gottesman. The Heisenberg representation of quantum computers. In S. P. Corney, R. Delbourgo, and P. D. Jarvis, editors, *Group22: Proceedings of the XXII International Colloquium on Group Theoretical Methods in Physics*, pages 32–43, 13–17 Jul. 1998, Hobart, Australia, 1999. International Press, Cambridge, MA, quant-ph/9807006.
- [Got13] Daniel Gottesman. Fault-tolerant quantum computation with constant overhead, 2013, arXiv:1310.2984.
- [Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In Gary L. Miller, editor, *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 212–219, 22–24 May 1996, Philadelphia, PA, USA, 1996. ACM Press, New York, quant-ph/9605043. See also [Gro97].
- [Gro97] Lov K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.*, 79(2):325–328, Jul. 1997, quant-ph/9706033. See also [Gro96].
- [Haa11] Jeongwan Haah. Local stabilizer codes in three dimensions without string logical operators. *Phys. Rev. A*, 83:042330, 2011.
- [Har04] James William Harrington. *Analysis of quantum error-correcting codes: symplectic lattice codes and toric codes*. PhD thesis, Caltech, 2004.
- [Hat01] Allen Hatcher. *Algebraic Topology*. Cambridge University Press, 2001.

References

- [HFDv12] Clare Horsman, Austin G. Fowler, Simon Devitt, and Rodney van Meter. Surface code quantum computing by lattice surgery. *New J. Phys.*, 14:123011, Dec 2012, 1111.4022.
- [HIZ05] Alioscia Hamma, Radu Ionicioiu, and Paolo Zanardi. Bipartite entanglement and entropic boundary law in lattice spin systems. *Phys. Rev. A*, 71(2):022315, Feb 2005, quant-ph/0409073.
- [HL08] Alioscia Hamma and Daniel A. Lidar. Adiabatic preparation of topological order. *Phys. Rev. Lett.*, 100(3):030502, 2008, quant-ph/0607145.
- [HRC02] Aram W. Harrow, Benjamin Recht, and Isaac L. Chuang. Efficient discrete approximations of quantum gates. *J. Math. Phys.*, 43(9):4445, Sep. 2002, quant-ph/0111031.
- [IWPk08] N. Isailovic, M. Whitney, Y. Patel, and J. Kubiawicz. Running a quantum circuit at the speed of data. page 177, 21–25 Jun., 2008, Beijing, China, 2008. IEEE, IEEE Press, Los Alamitos, arXiv:0804.4725.
- [JFS06] Stephen P. Jordan, Edward Farhi, and Peter W. Shor. Error correcting codes for adiabatic quantum computation. *Phys. Rev. A*, 74:052322, 2006, quant-ph/0512170.
- [JNT⁺14] Pejman Jouzdani, E. Novais, I. S. Tupitsyn, Robert Raussendorf, and Eduardo R. Mucciolo. Fidelity threshold of the surface code beyond single-qubit error models, 2014, arXiv:1401.6540.
- [JOL13] Tomas Jochym-O’Connor and Raymond Laflamme. Using concatenated quantum codes for universal fault-tolerant quantum gates, 2013, arXiv:1309.3310.
- [Jor09] Stephen Jordan. The quantum algorithm zoo, 2009.
- [JOYHL13] Tomas Jochym-O’Connor, Yafei Yu, Bassam Helou, and Raymond Laflamme. The robustness of magic state distillation against errors in Clifford gates. *Quant. Inf. Comp.*, 13:361–378, 2013, arXiv:1205.6715.
- [JWM⁺12] N. Cody Jones, James D. Whitfield, Peter L. McMahon, Man-Hong Yung, Rodney Van Meter, Alán Aspuru-Guzik, and Yoshihisa Yamamoto. Simulating chemistry efficiently on fault-tolerant quantum computers. *New J. Phys.*, 14:115023, 2012, arXiv:1204.0567.
- [KÅS06] David Kult, Johan Åberg, and Erik Sjöqvist. Noncyclic geometric changes of quantum states. *Phys. Rev. A*, 74(2):022106, 2006, quant-ph/0512045.

References

- [KBMD09] Helmut G. Katzgraber, H. Bombin, and M. A. Martin-Delgado. Error threshold for color codes and random three-body Ising models. *Phys. Rev. Lett.*, 103:090501, Aug. 2009, 0902.4845.
- [KC08] Alistair Kay and Roger Colbeck. Quantum self-correcting stabilizer codes, 2008, arXiv:0810.3557.
- [Kit95] A. Yu. Kitaev. Quantum measurements and the abelian stabilizer problem, 1995, quant-ph/9511026.
- [Kit97a] A. Yu. Kitaev. Quantum computations: algorithms and error correction. *Russian Math. Surveys*, 52(6):1191–1249, 1997.
- [Kit97b] A. Yu. Kitaev. Quantum error correction with imperfect gates. In O. Hirota, A. S. Holevo, and C. M. Caves, editors, *Proceedings of the Third International Conference on Quantum Communication, Computing and Measurement*, New York, 1997. Plenum Press.
- [Kit03] A. Yu. Kitaev. Fault-tolerant quantum computation by anyons. *Ann. Phys.*, 303(1):2–30, 2003, quant-ph/9707021.
- [Kit06] A. Kitaev. Anyons in an exactly solved model and beyond. *Ann. Phys.*, 321(1):2–111, 2006, cond-mat/0506438.
- [KK12] Alexei Kitaev and Liang Kong. Models for gapped boundaries and domain walls. *Commun. Math. Phys.*, 313:351–373, 2012.
- [KKR06] Julia Kempe, Alexei Kitaev, and Oded Regev. The complexity of the local Hamiltonian problem. *SIAM J. Comp.*, 35(5):1070–1097, 2006, quant-ph/0406180.
- [KKR10] Robert Koenig, Greg Kuperberg, and Ben W. Reichardt. Quantum computation with Turaev-Viro codes. *Ann. Phys.*, 325:2707–2749, 2010.
- [Kli13] Vadym Kliuchnikov. Synthesis of unitaries with Clifford+T circuits, 2013, arXiv:1306.3200.
- [KLZ98] Emmanuel Knill, Raymond Laflamme, and Wojciech H. Zurek. Resilient quantum computation: Error models and thresholds. *Proc. Roy. Soc. London A*, 454(1969):365–384, Jan. 1998, quant-ph/9702058.
- [KMM13a] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Asymptotically optimal approximation of single qubit unitaries by Clifford and T circuits using a constant number of ancillary qubits. *Phys. Rev. Lett.*, 110:190502, 2013, arXiv:1212.0822.

References

- [KMM13b] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Fast and efficient exact synthesis of single qubit unitaries generated by Clifford and T gates. *Quant. Inf. Comp.*, 13:607–630, 2013, arXiv:1206.5236.
- [Kni04] Emmanuel Knill. Fault-tolerant postselected quantum computation: Schemes, 2004, quant-ph/0402171.
- [Kön10] Robert König. Composite anyon coding and the initialization of a topological quantum computer. *Phys. Rev. A*, 81(5):052309, May 2010, arXiv:0910.2427.
- [KP06] Alexei Kitaev and John Preskill. Topological entanglement entropy. *Phys. Rev. Lett.*, 96(11):110404, 2006, hep-th/0510092.
- [KSV02] Alexei Yuri Kitaev, Alexander Shen, and Mihkail N. Vyalyi. *Classical and Quantum Computation*, volume 47 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2002. Translated from the Russian edition by Lester J. Senechal.
- [LAR11] Andrew J. Landahl, Jonas T. Anderson, and Patrick R. Rice. Fault-tolerant quantum computing with color codes, 2011, arXiv:1108.5738.
- [LC13] Andrew J. Landahl and Chris Cesare. Complex instruction set computing architecture for performing accurate quantum Z rotations with less magic, 2013, arXiv:1302.3240.
- [Lid08] Daniel A. Lidar. Towards fault tolerant adiabatic quantum computation. *Phys. Rev. Lett.*, 100:160506, 2008, arXiv:0707.0021.
- [Liu06] Xiaoyu Liu. *On divisible codes over finite fields*. PhD thesis, Caltech, 2006.
- [LM77] J. M. Leinaas and J. Myrheim. On the theory of identical particles. *Nuovo Cimento B*, 37:1, 1977.
- [LP09] V. Lahtinen and J. K. Pachos. Non-Abelian statistics as a Berry phase in exactly solvable models. *New J. Phys.*, 11:093027, 2009, arXiv:0901.3674.
- [LW05] Michael A. Levin and Xiao-Gang Wen. String-net condensation: A physical mechanism for topological phases. *Phys. Rev. B*, 71:045110, 2005.

References

- [LW06] Michael Levin and Xiao-Gang Wen. Detecting topological order in a ground state wave function. *Phys. Rev. Lett.*, 96(11):110405, 2006, cond-mat/0510613.
- [MEK12] Adam M. Meier, Bryan Eastin, and Emmanuel Knill. Magic-state distillation with the four-qubit code, 2012, arXiv:1204.4221.
- [Miz10] Ari Mizel. Fixed-gap adiabatic quantum computation, 2010, arXiv:1002.0846.
- [Miz14] Ari Mizel. Fault-tolerant, universal adiabatic quantum computation, 2014, arXiv:1403.7694.
- [MK13] Ross B. McDonald and Helmut G. Katzgraber. Genetic braid optimization: A heuristic approach to compute quasiparticle braids. *Phys. Rev. B*, 87:054414, 2013, arXiv:1211.7359.
- [MLM07] Ari Mizel, Daniel A. Lidar, and Morgan Mitchell. Simple proof of equivalence between adiabatic quantum computation and the circuit model. *Phys. Rev. Lett.*, 99:070502, 2007, quant-ph/0609067.
- [Mos08] Michele Mosca. Quantum algorithms, 2008, arXiv:0808.0369.
- [MS77] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The Theory of Error-Correcting Codes*, volume 16 of *North-Holland mathematical library*. North-Holland, New York, 1977.
- [NC00] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, 2000.
- [NSS⁺08] Chetan Nayak, Steven H. Simon, Ady Stern, Michael Freedman, and Sankar Das Sarma. Non-Abelian anyons and topological quantum computation. *Rev. Mod. Phys.*, 80(3):1083, 2008, arXiv:0707.1889.
- [OBL09] O. Oreshkov, T. A. Brun, and D. A. Lidar. Fault-tolerant holonomic quantum computation. *Phys. Rev. Lett.*, 102(7):070502, 2009, 0806.0875.
- [Ore09] Ognian Oreshkov. Holonomic quantum computation in subsystems. *Phys. Rev. Lett.*, 103(9):090502, Aug 2009, arXiv:0905.1249.
- [OT08] Roberto Oliveira and Barbara M. Terhal. The complexity of quantum spin systems on a two-dimensional square lattice. *Quant. Inf. Comp.*, 8(10):0900, Nov 2008, quant-ph/0504050.

References

- [Par04] Lecture notes in physics. In M. Paris and J. Rehacek, editors, *Quantum State Estimation*, volume 649. Springer, 2004.
- [PC08] David Poulin and Yeojin Chung. On the iterative decoding of sparse quantum codes. *QIC*, 8:987, 2008, arXiv:0801.1241.
- [PR13] Adam Paetznick and Ben W. Reichardt. Universal fault-tolerant quantum computation with only transversal gates and error correction. *Phys. Rev. Lett.*, 111:090505, 2013.
- [Pre98a] John Preskill. Fault-tolerant quantum computation. In Hoi-Kwong Lo, Tim Spiller, and Sandu Popescu, editors, *Introduction to Quantum Computation and Information*, chapter 8, pages 213–269. World Scientific, Singapore / River Edge, NJ, 1998, quant-ph/9712048.
- [Pre98b] John Preskill. Lecture notes for Caltech Ph 219: Quantum Information and Computation, 1998.
- [Pre98c] John Preskill. Reliable quantum computers. *Proc. Roy. Soc. London A*, 454(1969):385–410, Jan. 1998, quant-ph/9705031.
- [PS13] Adam Paetznick and Krysta M. Svore. Repeat-until-success: Non-deterministic decomposition of single-qubit unitaries, 2013, arXiv:1311.1074.
- [PSRDL12] Gerardo A. Paz-Silva, A. T. Rezakhani, Jason M. Dominy, and D. A. Lidar. Zeno effect for quantum computation and control. *Phys. Rev. Lett.*, 108:080501, 2012, arXiv:1104.5507.
- [RB01] Robert Raussendorf and Hans J. Briegel. A one-way quantum computer. *Phys. Rev. Lett.*, 86(22):5188–5191, May 2001.
- [RC05] Jeremie Roland and Nicolas J. Cerf. Noise resistance of adiabatic quantum computation using random matrix theory. *Phys. Rev. A*, 71:032330, 2005, quant-ph/0409127.
- [Rei04] Ben W. Reichardt. The quantum adiabatic optimization algorithm and local minima. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the Thirty-Sixth Annual ACM Symposium on the Theory of Computing*, pages 502–510, 13–15 Jun. 2004, Chicago, IL, USA, 2004. ACM Press, New York.
- [Rei05] Ben W. Reichardt. Quantum universality from magic states distillation applied to CSS codes. *Quant. Inf. Proc.*, 4(3):251–264, Aug. 2005, quant-ph/0411036.

References

- [Rei09] Ben W. Reichardt. Quantum universality by state distillation. *Quant. Inf. Comp.*, 9:1030–1052, 2009, quant-ph/0608085.
- [RH07] Robert Raussendorf and Jim Harrington. Fault-tolerant quantum computation with high threshold in two dimensions. *Phys. Rev. Lett.*, 98:190504, May 2007, quant-ph/0610082.
- [RHG06] R. Raussendorf, J. Harrington, and K. Goyal. A fault-tolerant one-way quantum computer. *Ann. Phys.*, 321:2242–2270, 2006, quant-ph/0510135.
- [RHG07] Robert Raussendorf, Jim Harrington, and Kovid Goyal. Topological fault-tolerance in cluster state quantum computation. *New J. Phys.*, 9:199, 2007, quant-ph/0703143.
- [RS14] Neil J. Ross and Peter Selinger. Optimal ancilla-free Clifford+T approximation of z-rotations, 2014, arXiv:1403.2975.
- [Sac05] Massimiliano F. Sacchi. Optimal discrimination of quantum operations. *Phys. Rev. A*, 71:062340, 2005, quant-ph/0505183.
- [Sel12] Peter Selinger. Efficient Clifford+T approximation of single-qubit operators, 2012, arXiv:1212.6253.
- [Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In S. Goldwasser, editor, *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 20–22 Nov. 1994, Santa Fe, NM, USA, 1994. IEEE, IEEE Press, Los Alamitos, CA, quant-ph/9508027. See also extended version in [Sho97].
- [Sho95] Peter W. Shor. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A*, 52(4):R2493, Oct. 1995.
- [Sho96] Peter W. Shor. Fault-tolerant quantum computation. In Regina Spencer Sipple, editor, *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 56–65, 14–16 Oct. 1996, Burlington, VT, USA, 1996. IEEE, IEEE Press, Los Alamitos, CA, quant-ph/9605011.
- [Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comp.*, 26(5):1484–1509, 1997, quant-ph/9508027. See also earlier version in [Sho94].

References

- [SK05] Pradeep Sarvepalli and Andreas Klappenecker. Nonbinary quantum Reed-Muller codes. pages 1023–1027, 4–9 Sep. 2005, Adelaide, Australia, 2005. IEEE, IEEE Press, quant-ph/0502001.
- [SL05] M. S. Sarandy and D. A. Lidar. Adiabatic quantum computation in open systems. *Phys. Rev. Lett.*, 95:250503, 2005, quant-ph/0502014.
- [Sol00] R. Solovay. Lie groups and quantum circuits, 2000. MSRI presentation at <http://www.msri.org/publications/ln/msri/2000/qcomputing/solovay/1/>. According to Ref. [DN06], the proof was announced by Solovay in 1995 on a virtual reading group e-mail discussion list.
- [SR09] Federico M. Spedalieri and Vwani P. Roychowdhury. Latency in local, two-dimensional, fault-tolerant quantum computing. *Quant. Inf. Comp.*, 9(7–8):666–682, Jul. 2009, arXiv:0805.4213.
- [Ste96a] A. M. Steane. Error correcting codes in quantum theory. *Phys. Rev. Lett.*, 77(5):793–797, Jul. 1996.
- [Ste96b] A. M. Steane. Simple quantum error-correcting codes. *Phys. Rev. A*, 54(6):4741, 1996, quant-ph/9605021.
- [Ste96c] Andrew Steane. Multiple particle interference and quantum error correction. *Proc. Roy. Soc. London A*, 452(1954):2551–2577, 1996, quant-ph/9601029.
- [Ste96d] Andrew Steane. Quantum Reed-Muller codes, 1996, quant-ph/9608026.
- [Ste97] A. M. Steane. Active stabilization, quantum computation, and quantum state synthesis. *Phys. Rev. Lett.*, 78(11):2252–2255, Mar. 1997, quant-ph/9611027.
- [Ste14] Ashley M. Stephens. Efficient fault-tolerant decoding of topological color codes, 2014, arXiv:1402.3037.
- [Tho69] W. H. Thomson. On vortex motion. *Trans. Roy. Soc. Edinburgh*, 25:217–260, 1869.
- [TS07] Markus Tiersch and Ralf Schützhold. Non-Markovian decoherence in the adiabatic quantum search algorithm. *Phys. Rev. A*, 75:062313, 2007, quant-ph/0608123.
- [TVH12] Pham Tien Trung, Rodney Van Meter, and Clare Horsman. Optimising the Solovay-Kitaev algorithm, 2012, arXiv:1209.4139.

References

- [TZ09] Jean-Pierre Tillich and Gilles Zemor. Quantum LDPC codes with positive rate and minimum distance proportional to $n^{1/2}$, 2009, arXiv:0903.0566.
- [War90] Harold N. Ward. Weight polarization and divisibility. *Discrete Math.*, 83(2–3):315–326, Aug. 1990.
- [Wat09] John Watrous. Semidefinite programs for completely bounded norms, 2009, arXiv:0901.4709.
- [WFSH10] D. S. Wang, A. G. Fowler, A. M. Stephens, and L. C. L. Hollenberg. Threshold error rates for the toric and surface codes. *Quant. Inf. Comp.*, 10:546, 2010.
- [WH05] Y. S. Weinstein and C. S. Hellberg. Energetic suppression of decoherence in exchange-only quantum computation. *Phys. Rev. A*, 72:022319, 2005, quant-ph/0408037.
- [Wil82] Frank Wilczek. Quantum mechanics of fractional-spin particles. *Phys. Rev. Lett.*, 49:957–959, 1982.
- [Wit89] Edward Witten. Quantum field theory and the Jones polynomial. *Commun. Math. Phys.*, 121:351–399, 1989.
- [WM10] Howard M. Wiseman and Gerard J. Milburn. *Quantum Measurement and Control*. Cambridge University Press, 2010.
- [Woo13] James R. Wootton. A simple decoder for topological codes. *Phys. Rev. A*, 88:062312, 2013.
- [Yos11] Beni Yoshida. Feasibility of self-correcting quantum memory and thermal stability of topological order. *Ann. Phys.*, 326:2566–2633, 2011.
- [YS13] Kevin C. Young and Mohan Sarovar. Unification and limitations of error suppression techniques for adiabatic quantum computing. *Phys. Rev. X*, 3:041013, 2013, arXiv:1208.6371.
- [ZB14] Yi-Cong Zheng and Todd A. Brun. A fault-tolerant scheme of holonomic quantum computation on stabilizer codes with robustness to low-weight thermal noise. *Phys. Rev. A*, 89:032317, 2014.
- [ZF97] Lin Zhang and Ian Fuss. Quantum Reed-Muller codes, 1997, quant-ph/9703045.
- [ZR99] Paolo Zanardi and Mario Rasetti. Holonomic quantum computation. *Phys. Lett. A*, 264:94–99, 1999, quant-ph/9904011.