Code Space Dimension of Translation-Invariant Qubit Stabilizer Codes

von
Andrés Wilhelm Goens Jokisch

Bachelorarbeit in Physik

vorgelegt der
Fakultät für Mathematik, Informatik und
Naturwissenschaften der RWTH Aachen

im November 2011

angefertigt im
Institut für Quanteninformation bei
Prof. Dr. Barbara M. Terhal

RWTH AACHEN UNIVERSITY

Ich versichere, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe.

Aachen, den 4. November 2011

Contents

0.1 A Note on Notation

This thesis uses many concepts from abstract algebra. Appendix A has a supplement on many terms, and can be consulted when in doubt about a specific definition. The following list is a small reference on a few notational conventions used on the thesis:

- $\underline{n} := \{1, 2, \ldots, n\}$
- $< \vec{e}_1, \vec{e}_2 >_{\mathbb{Z}} := \{x\vec{e}_1 + y\vec{e}_2 \mid x, y \in \mathbb{Z}\}$
- $A \triangleleft B$ means that A is an ideal or normal subgroup of B, depending on the context (if B is a ring or a just a group)
- $f \mid g$ means that f divides g, i.e. $\exists h : g = fh$.
- $f \nmid g$ means that f does not divide g.
- $< g_1, \ldots, g_n > := \{h_1 \cdots h_k \mid h_i \in \{g_1, g_1^{-1}, g_2, g_2^{-1}, \ldots, g_n, g_n^{-1}\} \, \forall \, i \in \underline{k}\}$ the group generated by $g_1, \ldots, g_n$.
- $\mathbb{F}_2 = \mathrm{GF}(2)$ The finite field, or galois field, with two elements.

1 Introduction

Quantum information and quantum computation are fields of much interest, not only from a theoretical standpoint, but also quite practically for building a quantum computer. Classical computers are not perfect: during the storage, transmission and processing they are bound to make errors and corrupt the data. There is a mature theory in mathematics which helps deal with that: coding theory. Through coding theory we can reliably store, transmit and process data in classical computers, but what about quantum computers? Through quantum codes, an analogous concept to what is used in classical computers, we try to do the same for quantum computers. In this thesis we will briefly introduce both, classical and quantum codes and we will develop a very important class of quantum codes called stabilizer or additive codes.

With the idea of application in mind, we will investigate a subclass stabilizer codes: geometrical stabilizer codes. It has the advantage of being a physically plausible model. We will introduce the concept of a self-correcting quantum memory, and look for codes which could realize one in this subclass. We will study STS codes, a subclass of geometrical stabilizer codes, which amongst others, has a code space dimension that is independent of the lattice size. We will see that it was proven by Yoshida in [9] that STS codes are not to be feasible for self-correcting quantum memory. After this, we will study a code proposed by Haah in [3] which escapes this conclusion, and motivated by this code, we will develop a formalism for a larger class of codes without the assumption on the code space dimension. Using this formalism we will develop an algorithm for calculating this dimension and apply it to Haah's code.

Acknowledgements:

First and foremost, I would like to thank Barbara Terhal for her feedback, for her help with the subject and especially for her patience with me while writing this thesis. I would also like to thank Jeongwan Haah for sharing his unpublished notes which were a central to the main part of this thesis. Last, I would like to thank my parents; in particular my mother, for helping me with the proof-reading for the language used in here.

2 Qubits and Stabilizer Codes

2.1 Classical Codes

Before we study quantum codes, it might be a good idea to get at least a very small feel for classical error-correcting codes. Classical error-correcting codes are a mathematical structure that we experience every day. They are mainly used to make sure a message arrives, even through a channel that can produce errors. Speech itself is a rather complex code: we can usually understand someone, even when there is some background noise, or when the person speaking makes a small mistake. Much staighter forward are for instance the codes used on CD-roms, which are called linear codes (Reed-Solomon codes are a specific example used on CD-ROMs). Linear codes will only be briefly introduced to give an idea of coding theory.

2.1.1 Linear Codes

Before we get into a mathematical definition of linear codes, it might be prudent to refresh a few simple concepts.

$\mathbb{F}_2$ (sometimes also called GF(2)) is a field, just like $\mathbb{R}$ or $\mathbb{C}$, in which there is an addition $+$ and a multiplication $\cdot$, but has only two elements. We shall call these 0 and 1 respectively. To calculate in $\mathbb{F}_2$, we can calculate over $\mathbb{R}$ and take the result modulo 2. By the way, this would work for any prime number instead of 2, yielding other finite fields, but we are not interested in them here. Concepts like vector spaces, subspaces and dimension work for $\mathbb{F}_2$, just as they do for $\mathbb{R}$ or $\mathbb{C}$.

Definition 2.1. A linear code C coding k bits on n bits, also called [n,k]-Code is a k-dimensional $\mathbb{F}_2$ subspace of $\mathbb{F}_2^n$.

The problem of saving the code in some way (knowing what the code-words are) and of encoding some message, is easily solved by using so-called generator matrices. If you take a basis $b_1, \ldots, b_k$ of the code C as a subspace of $\mathbb{F}_2^n$, and let G be the matrix with $b_1, \ldots, b_k$ as columns, then you get the code as the span of this matrix G. Even further, if $v \in \mathbb{F}_2^k$ is some k-bit string, coded as a column vector, then you can encode v just by calculating $Gv \in \mathbb{F}_2^n$, which is read as an n-bit string. Note that the generator matrix is not unique, and so the encoding might differ. This must be kept in mind for decoding.

Example 2.2. One of the simplest linear codes there are, is the repetition code. You just repeat the bit three times: you code 0 as 000, and 1 as 111. If later an error occurs and you get the string 101, it is more likely that it was

a 111 than a 000, that is assuming only one error can occur, we can retrieve the original bit. We say that this code corrects one error. A basis for this code is $b_1 = (1, 1, 1)^T$, where v^T denotes the transpose.

$$\text{then G} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \text{ is a Generator matrix for the repetition code.}$$

Another important concept is that of the parity-check matrix, which tends to be more useful for error correction. Instead of a generating set, we let H be a matrix with kernel C, that is with $Hv = 0$ iff $v \in C$. It is straightforward to get the parity-check matrix from the code, but we will not dwell into that.

What this means is that we can easily detect if a received message is part of the code, in which case we assume it was sent correctly. While strictly speaking it might not be the case, we assume some maximal number of mistakes which would rule it out. If while sending the coded message $m \in \mathbb{F}_2^n$ some minor error $e \in \mathbb{F}_2^n$ happened, where e has a 1 in each place where an error has occurred, then by applying the parity check matrix H we get the so-called syndrome: $s = H(m + e)$. Here, the linearity of the code (for it being represented by matrices) comes into play:

$$s = H(m + e) = Hm + He = 0 + He$$

So the syndrome is directly related to the error. From here we can proceed, using the syndrome s, to find e and correct the code. We will not get into the specifics of this.

Example 2.3. For the repetition code above, a parity-check matrix is given by

$$H = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

2.1.2 Dual Codes

The main advantage of linear codes is that they have an algebraic structure which can be used in many ways. One of these is by defining a bilinear form on the code:

Definition 2.4. Let $b = (b_1, \ldots, b_n), c = (c_1, \ldots, c_n) \in \mathbb{F}_2^n$ be two codewords. Define $< b, c >:= \sum_{i=1}^n b_i c_i \in \mathbb{F}_2$ to be the product of b and c. $< \cdot, \cdot >$ thus defines what is called a symmetric bilinear form, which is a generalization of the scalar product in a sense.

Bilinear forms are an object that has been studied intensively in mathematics. One thing we immediately get from introducing this product is the concept of duality, and in particular, of basis independent dual space to the code space.

Definition 2.5. Let $C \leq \mathbb{F}_2^n$ be a linear code. Set $C^\perp := \{v \in \mathbb{F}_2^n \mid < v, c >= 0 \ \forall \ c \in C\}$. $C^\perp$ is called the dual code to C. From the theory of bilinear forms we immediately get the result: Dim C + Dim $C^\perp = n$, so we know directly what the number of encoded bits is. A code which satisfies $C \subseteq C^\perp$ is called weakly self dual, and self dual if $C = C^\perp$. Note that this means that for a self dual code, n has to be even and Dim $C = \frac{n}{2}$.

2.1.3 Code Distance

One last important concept from classical coding theory, which has an important analogous concept in quantum codes, is the distance of a code. First, we have to introduce some sort of measure of distance in the code.

Definition 2.6. Let $C \leq \mathbb{F}_2^n$ be a linear code, and $c^{(1)}, c^{(2)} \in C$. The weight of $c^{(1)}$ is defined as $wt(c^{(1)}) := |\{i \in \underline{n} \mid c_i^{(1)} \neq 0\}|$, i.e. the number of bits of $c^{(1)}$ that are not 0. The Hamming distance between $c^{(1)}$ and $c^{(2)}$ is defined similarly, as the number of different bits between them: $d(c^{(1)}, c^{(2)}) = |\{i \in \underline{n} \mid c_i^{(1)} \neq c_i^{(2)}\}| = wt(c^{(1)} - c^{(2)})$. For the whole code, the minimal distance and minimal weight, which for linear codes are the same, are defined as: $\min_{c,c' \in C} d(c, c') = \min_{0 \neq c \in C} wt(c)$.

Probably by far the most important property of the minimal distance is that a code with distance d can always correct up to $t < \frac{d}{2}$ errors. The reason for this is simple to understand, since if $t < \frac{d}{2}$ errors occur, then the distance between the received bit string and the original codeword will be strictly smaller than the distance between the received string and any other codeword, by definition of the minimal distance.

2.2 Qubits and quantum codes

The main subjects of this thesis are quantum codes. For these, we no longer work with bits, but instead do so with so-called 'qubits', or 'quantum bits'.

Definition 2.7. A qubit is a two dimensional $\mathbb{C}$ vector space, $\mathbb{C}^2$. It is an abstraction of a two state quantum system, whose physical realization we disregard, and is described by the Hilbert space $\mathcal{H} = \mathbb{C}^2$. For convention we will call two orthonormal states of $\mathcal{H}$ $|0\rangle$ and $|1\rangle$ in analogy to the two states of a bit, 0 and 1. Hence, span$(|0\rangle, |1\rangle) = \mathcal{H}$.

Remark 2.8. A many qubit system is, as by the postulates of quantum mechanics, described by the tensor product Hilbert space of its single qubits. This is the reason why an n-qubit system is described by $\mathcal{H} = \underbrace{\mathbb{C}^2 \otimes \ldots \otimes \mathbb{C}^2}_{\text{n times}} \cong \mathbb{C}^{2^n}$. The basis of this space given by the $|0\rangle, |1\rangle$ basis for each qubit can be abbreviated by binary strings: $|b_1 b_2 \ldots b_n\rangle := |b_1\rangle \otimes |b_2\rangle \ldots \otimes |b_n\rangle$. For instance $|010\rangle = |0\rangle \otimes |1\rangle \otimes |0\rangle$. This notation also makes the analogy to bits further clear, hence the name, qubits.

Just as we did with linear classical codes, we can now define quantum codes:

Definition 2.9. A quantum code $\mathcal{C}$ is a (linear) subspace of an n-qubit space $\mathbb{C}^{2^n}$. If the code is a 2^k dimensional subspace of $\mathbb{C}^{2^n}$ for some integer $k \leq n$, then we call the code $\mathcal{C}$ an $[[n, k]]$-Code. In this case there is a bijective linear mapping from a k-qubit system to the code $\mathcal{C}$. We say the code encodes k qubits in n qubits.

We shall now see how these codes help correct errors in qubits.

2.3 Error correction

What good is a code if it does not correct errors? We will see with the same example from classical coding theory, the repetition code, how error correction can work in principle. Before we do so, however, we must first consider what kinds of error can happen. From the postulates of quantum mechanics we know that the time evolution of a closed system is given by a unitary operator. This means that in principle, any unitary operator can act on the code. There is, however, no reason for the system to be closed: it will interact with the environment, meaning that measurements can also be made, amongst others. There is a formalism called quantum operation formalism that describes all these errors; a development of this formalism can be found in [2]. Here we will restrict ourselves to unitary operators. We will concentrate only on a small subset of these, the so-called Pauli group, which will turn out to be enough thanks to theorem 2.13:

Definition 2.10. The one-qubit unitary operators defined by

$$I, X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, Y = \begin{bmatrix} 0 & i \\ -i & 0 \end{bmatrix}, Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{1}$$

are called Pauli operators.

On n qubits, the operators of the form $\sigma_1 \otimes \ldots \otimes \sigma_n$, where $\sigma_i \in \{I, X, Y, Z\}$ for all i, are called n-qubit Pauli operators. If you take all products of n-qubit Pauli operators you get a finite group $\mathcal{P}_n$, called the Pauli group. It is easy to see that $\mathcal{P}_n = \{i^j \sigma_1 \otimes \ldots \otimes \sigma_n \mid \sigma_k \in \{I, X, Y, Z\} \text{ for all } k \in \underline{n}, j \in \underline{4}\}$.

Example 2.11. A first, naive approach for building a quantum code is just the repetition code in qubits:

$$|0\rangle \mapsto |000\rangle$$

$$|1\rangle \mapsto |111\rangle$$

It is not hard to see that this code will protect from one Pauli X error, for example, X_2:

$$X_2|000\rangle = |010\rangle$$

And just as in the classical analog, we can see this (assuming only one X_i error) can only have been $|000\rangle$. It is not difficult to see that this code will protect from a single X_i error on an arbitrary qubit. Simple, right? The problem is, we have to measure in some way the state of the code in order to find out what the state is and what error happened. Fortunately, there is a way of covering all 'bit flip' (X_i) errors with a measurement: Define the following measurement operators:

$$P_0 := |000\rangle\langle 000| + |111\rangle\langle 111|,\ P_1 := |100\rangle\langle 100| + |011\rangle\langle 011|$$

$$P_2 := |010\rangle\langle 010| + |101\rangle\langle 101|,\ P_3 := |001\rangle\langle 001| + |110\rangle\langle 110|$$

From the operators themselves it is easy to see that a measurement in this basis will immediately identify if exactly one 'bit flip' error (but nothing else) has happened, and leave everything as it is.

This example also shows the problem mentioned at the beginning of this section; that unlike in classical coding theory, there is a continuous, infinite set of possible errors ($\mathrm{SU}(2, \mathbb{C})$ for a single qubit). Consider the state $|+\rangle := 1/\sqrt{2}(|0\rangle + |1\rangle)$. Coding this would yield $1/\sqrt{2}(|000\rangle + |111\rangle)$. Since all error operators are linear, a bit flip would still be corrected with the measurement above. But how about a phase flip? Consider the error Z_1, for example. $Z_1(\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)) = \frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$, which would go undetected, but would be decoded to be $|-\rangle := 1/\sqrt{2}(|0\rangle - |1\rangle)$.

We can modify the code to detect phase flip (Z_i) errors: We have seen that the operator Z takes $|+\rangle$ to $|-\rangle$ and vice-versa. That is, it flips both states. If we were to rename our states $|0\rangle$ and $|1\rangle$, Z would act as an X. With this in mind, it is not difficult to see why coding $|0\rangle \mapsto |++\rangle$, $|1\rangle \mapsto |--\rangle$

would be a good idea. If we make the same measurement as in example 2.11, but in the $|+\rangle, |-\rangle$ basis instead, a phase flip would be the same as a bit flip in that basis, and would be detected. It can be immediately checked that the operator

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

also called a Haddamard gate, takes $(|0\rangle, |1\rangle) \mapsto (|+\rangle, |-\rangle)$. So the measurement operators would be $HP_iH^\dagger = HP_iH, i = 0, 1, 2, 3$.

But this code, again, would not be able to detect a bit flip. We can try combining them, on a 9 qubit code:

Example 2.12. This 9 qubit code, also known as Shor code [2] can correct an X or Z error on **a single qubit**. This is achieved by combining the two codes previously mentioned. First the phase flip code, $|0\rangle \mapsto |++\rangle$, $|1\rangle \mapsto |--\rangle$, then the bit flip code to each of the three qubits of the first encoding: $|++\rangle = |+\rangle|+\rangle|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \mapsto \frac{1}{2\sqrt{2}}(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)$ and in an analogous fashion $|--\rangle \mapsto \frac{1}{2\sqrt{2}} = (|000\rangle - |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle)$

This, we have seen, can correct a single one of two different errors on a single qubit. It might seem like a lot of work for very little gain. Fortunately, as we will see, the correcting of these two errors is enough to correct an arbitrary single-qubit error! In fact, the following theorem (2.13) is one of the results that make quantum error correction possible in the first place, since it helps to bypass the problem of a continuous, infinite set of possible errors.

Theorem 2.13. Let $\{E_i \mid i \in I\}$ for some index set I be a set of unitary operators, and C be a quantum code that can correct errors on all these $E_i, i \in I$. Further let $\{F_j \mid j \in J\}$ for another index set J be another set of unitary operators, which is a linear combination of the E_i, i.e. $F_j = \sum_{i \in I} c_{i,j} E_i$ for all $j \in J$ and for some $c_{i,j} \in \mathbb{C}$. Then, C can also correct all errors $F_j, j \in J$. This theorem actually holds for a wider class of errors, where E_i, F_j are quantum operations; but this goes beyond the scope of this thesis. The more general statement, along with a proof, can be found in [2].

Now, with a feel for quantum codes, we can start studying the stabilizer codes, a very important class of quantum codes with a core role for this thesis:

2.4 The stabilizer formalism

Before we get into the formalism itself, we need to cover some background.

Remark 2.14. The Pauli Group $\mathcal{P}_n$ has the following interesting properties:

- Two operators in the Pauli group either commute or anti-commute, i.e. $\forall\ A, B \in \mathcal{P}_n : [A, B] = AB - BA = 0$ or $\{A, B\} = AB + BA = 0$. This follows immediately from its being a property of the Pauli matrices.
- The commutator subgroup of the Pauli group, defined as $\mathcal{P}_n' := \{ABA^{-1}B^{-1} \mid A, B \in \mathcal{P}_n\}$ is given by $\mathcal{P}_n = \{\pm I\}$, which is another way of saying that they only commute or anticommute.
- The centre of the Pauli group, defined as $Z(\mathcal{P}_n) = \{A \in \mathcal{P}_n \mid AB = BA \text{ for all } B \in \mathcal{P}_n\}$ is given by $Z(\mathcal{P}_n) = \{\pm I, \pm iI\}$.
- The factor group $\mathcal{P}_n / Z(\mathcal{P}_n)$ is an n-dimensional $\mathbb{F}_2$ vector space. This property is a special case of the Burnside basis theorem, see [13].
- Every element in the Pauli group can be written as follows:

$$i^j X_1^{a_1} Z_1^{b_1} X_2^{a_2} Z_2^{b_2} \dots X_n^{a_n} Z_n^{b_n} \tag{2}$$

  where $j \in \{0, 1, 2, 3\}$, $a_i, b_i \in \{0, 1\}\, \forall\, i \in \underline{n}$. That this is true can be seen immediately from $Y = iXZ$ and the fact that elements commute or anti-commute.

A more detailed analysis of the Pauli group can be found in [5]. There is a special type of subgroup of the Pauli group in which we are interested:

Definition 2.15. A subgroup $S \leq \mathcal{P}_n$ is called a stabilizer group iff $-I \notin S$. Note that this implies that S is abelian, since elements in the Pauli group either commute or anticommute, so if there were two non-commuting elements, they would have to anticommute, and $-I$ would be in S.

We are now ready to define a stabilizer code.

Definition 2.16. Let $S \leq \mathcal{P}_n$ be a stabilizer group. The subspace of the n-qubit space $\mathbb{C}^{2^n}$ defined by $V_S := \{|v\rangle \in \mathbb{C}^{2^n} \mid A|v\rangle = |v\rangle \text{ for all } A \in S\}$ is called the stabilizer code induced by S.

From the definition it is clear why it receives that name: it is the set of all elements *stabilized* by all of S. It is also immediately clear why we define stabilizer groups as we do: If $-I \in S$, then for all $|v\rangle \in V_S : |v\rangle = -I|v\rangle = -|v\rangle$, hence $|v\rangle = 0$, so $V_S = \{0\}$ is trivial, and we are not interested in it.

Remark 2.17. Physically, there is a canonical quantum system that implements a stabilizer quantum code. Let $S \leq \mathcal{P}_n$ be a stabilizer group with its corresponding code $\mathcal{C} = V_S$ and $\{s_1, \dots s_k\} \subset S$ be a set of stabilizers which generate S (not necessarily minimal). Then the code $\mathcal{C}$ is the same as the ground space of the Hamiltonian $H = -\sum_{i=1}^{k} s_i$. This is also a good motivation for studying stabilizer codes. Note that we only sum over some stabilizer group elements. For this to be a physically plausible model, we usually require the s_i to be local (what this means will be better understood in section 3) and have a weight that is $\mathcal{O}(1)$ in the system size (see def. 2.24). We will not dwell further into this.

Further than eliminating many trivial codes, the definition of a stabilizer group as above gives us a way of knowing the dimension of the generated code space, that is, how many qubits it encodes. We first need a few definitions to understand how this works:

Definition 2.18. Let $S \leq \mathcal{P}_n$ be a stabilizer group. Any $A \in S$ as in remark 2.14 can be written as $A = i^j X_1^{a_1} Z_1^{b_1} X_2^{a_2} Z_2^{b_2} \dots X_n^{a_n} Z_n^{b_n}$, where $j \in \{0, 1, 2, 3\}$, $a_i, b_i \in \{0, 1\} \, \forall \, i \in \underline{n}$. Then the mapping

$$A = i^j X_1^{a_1} Z_1^{b_1} X_2^{a_2} Z_2^{b_2} \dots X_n^{a_n} Z_n^{b_n} \mapsto (a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_n) \in \mathbb{F}_2^{2n} \quad (3)$$

is an homomorphism of vector spaces (a linear mapping). We call the resulting vector in $\mathbb{F}_2^{2n}$ the symplectic notation for A.

Note that this is well defined: since S is abelian, multiplication with

$$A' = i^{j'} X_1^{a'_1} Z_1^{b'_1} X_2^{a'_2} Z_2^{b'_2} \dots X_n^{a'_n} Z_n^{b'_n} \in S$$

$$\text{yields } AA' = i^{j+j'} X_1^{a_1+a'_1} Z_1^{b_1+b'_1} X_2^{a_2+a'_2} Z_2^{b_2+b'_2} \dots X_n^{a_n+a'_n} Z_n^{b_n+b'_n}$$

and since $X^2 = Z^2 = I$, the addition in the exponents is the same as over $\mathbb{F}_2$. In other words, multiplication in the stabilizer group is the same as addition in symplectic notation.

Similarly, any element in the Pauli group can be written that way. If we define an equivalency class in the Pauli group by saying that two operators are equivalent iff they differ by an overall scalar factor, we get the factor group $\mathcal{P}_n / Z(\mathcal{P}_n)$, which is an n-dimensional $\mathbb{F}_2$ vector space. We can write this in symplectic notation the same way we did with the stabilizer element by just caring about the exponents of the X_i and Z_i and not about the overall factor. Then, the commutator of two elements

$$[\cdot, \cdot] : \mathcal{P}_n / Z(\mathcal{P}_n) \times \mathcal{P}_n / Z(\mathcal{P}_n) \to \mathbb{F}_2, (A, B) \mapsto [A, B] = \begin{cases} 0 & \text{, AB = BA} \\ 1 & \text{, AB = -BA} \end{cases} \quad (4)$$

defines a symplectic bilinear form on $\mathcal{P}_n/Z(\mathcal{P}_n)$; hence the name, sympletic notation.

Definition 2.19. Let S be a stabilizer group generated by the elements $A_1, \ldots, A_k$, and let $v_1, \ldots, v_k$ be the corresponding elements in symplectic notation. A subset of these stabilizer operators, without loss of generality $A_1, \ldots, A_l$ is called independent iff the corresponding vectors in symplectic notation $v_1, \ldots, v_l$ are linearly independent over $\mathbb{F}_2^{2n}$.

Theorem 2.20. Let $S \leq \mathcal{P}_n$ be a stabilizer group and $\mathcal{C} = V_S$ the corresponding stabilizer code. Further let $s_1, \ldots, s_k \in S$ be a set of independent generators of S. The dimension of $\mathcal{C}$ as a $\mathbb{C}$ vector space is given by $\text{Dim }_{\mathbb{C}}\mathcal{C} = 2^{n-k}$, where n is the number of qubits of the encoding space and k the number of independent generators of S. For a proof see [1] or [2].

Example 2.21. With stabilizer codes now properly defined, we can return to the previous example (2.12), the 9-qubit Shor code. The Shor code is generated by the 8 independent generators in table 1 (see [2]). This is, in general, a much more compact way to write down the code. We see that, of course, the code is stabilized by all operators and there is 8 of them (what is in agreement with theorem 2.20). From their symplectic notation it is immediately clear that they are independent generators.

Name	Operator
g_1	$ZZIIIIIII$
g_2	$IZZIIIIII$
g_3	$IIIZZIIII$
g_4	$IIIIZZIII$
g_5	$IIIIIIZZI$
g_6	$IIIIIIIZZ$
g_7	$XXXXXXIII$
g_8	$IIIXXXXXX$

Table 1: Stabilizer Generators for the 9-Qubit Shor Code

2.5 Logical Operators

There is another class of operators which is very important for stabilizer codes: logical operators.

Definition 2.22. Let $S \leq \mathcal{P}_n$ be a stabilizer group. The centralizer of S in $\mathcal{P}_n$, $C_{\mathcal{P}_n}(S)$ is defined as:

$$C_{\mathcal{P}_n}(S) = \{A \in \mathcal{P}_n \mid [A, B] = 0 \text{ for all } B \in S\}$$

Note that since S is abelian, $S \leq C_{\mathcal{P}_n}(S)$. An element $L \in C_{\mathcal{P}_n}(S) \backslash S$ is called a logical operator.

This definition, while formally correct and unambiguous, does not help us understand what a logical operator really is. The first way to see this, and probably what gave logical operators their name too, is the following: An [[n,k]] qubit code encodes k qubits in $n > k$. Once encoded, we are confronted with n physical qubits, but know that logically, only k are represented. If we were to do an operator A on the unencoded k qubits, we would obtain a different state for these k qubits. But, how would this new state look like when encoded in the n qubits? There has to be an operator $A^{(n)}$ on n qubits that applies A on the encoded qubits: $A^{(n)}$ is a logical operator, as it acts as A on the logical qubits.

But why are these exactly the elements in $C_{\mathcal{P}_n}(S) \backslash S$? Well, for a logical operator to act on the logical, encoded qubits, it has to map codewords to codewords; after all, the new encoded state is part of the code as well. It is precisely the elements of $C_{\mathcal{P}_n}(S)$ that do that. To see why this is the case, consider the coded state $|\psi\rangle \in C$, and an arbitrary Pauli operator $\mathcal{O}$. Recall from remark 2.17 the Hamiltonian $H = -\sum_i s_i$. It has the property that $|\psi\rangle \in C \Leftrightarrow |\psi\rangle$ is a ground state of H. It is

$$H\mathcal{O}|\psi\rangle = -\sum_i s_i \mathcal{O}|\psi\rangle = -\sum_{i,[s_i,\mathcal{O}]=0} \mathcal{O} \underbrace{s_i|\psi\rangle}_{=|\psi\rangle} + \sum_{i,\{s_i,\mathcal{O}\}=0} \mathcal{O} \underbrace{s_i|\psi\rangle}_{=|\psi\rangle}$$

Thus, $\mathcal{O}|\psi\rangle$ is a ground state of $H \Leftrightarrow [s_i, \mathcal{O}] = 0 \, \forall \, i$. Since the s_i generate S, this is in turn equivalent to $[A, \mathcal{O}] = 0 \; \forall \; \in S \Leftrightarrow \mathcal{O} \in C_{\mathcal{P}_n}(S)$. On the other hand, the elements of S, per definition, act as the identity on the code, leaving it unchanged. We want to define only non-trivial logical operators as such, and leave therefore S out. With a little more mathematics (the development of which would be too long for this thesis), logical operators can be further understood:

For each operator A on k qubits, there is - of course - an operator $A^{(n)}$ on the encoding n qubits. This operator is not unique however: an equivalency class can be defined for logical operators, making two operators equivalent, when they act as the same operator on the k logical qubits. Two such equivalent operators differ actually always by an element of the stabilizer group, i.e. : $A^{(n)}$ and $B^{(n)}$ are equivalent iff $\exists T \in S$ with $A^{(n)} = B^{(n)}T$. That these is true can be made at least plausible by reminding us of the definition of the stabilizer group: it does not alter the code!

Example 2.23. Again, the 9-qubit Shor code (ex. 2.12): It encodes only one qubit, so we will only search for the two logical operators $\overline{X}$ and $\overline{Z}$. These

are given by $\overline{X} = ZZZZZZZZZ$ and $\overline{Z} = XXXXXXXXX$, as can be checked with a simple calculation. Note that it is not as one might naively expect at first, but exactly the opposite instead.

2.6 Code distance

There is an easy way of quantifying how many errors a code can correct: the code distance. It is analogous to what is used in classical coding theory. It is also a good reason for studying logical operators. Before introducing it however, we need a further definition:

Definition 2.24. Let $E \in \mathcal{P}_n$ be an arbitrary Pauli operator. The weight of E, $|E|$, is defined as the number of qubits on which E is supported, i.e. the number of qubits on which E acts non-trivially, while disregarding an overall phase (factor).

For example, the operator $Z_1X_2I_3X_4$ has weight 3, whereas $-I_1I_2X_3Z_3I_4$ has only weight 1.

Definition 2.25. Let $S \leq \mathcal{P}_n$ be a stabilizer group with corresponding code C. Then, the minimal distance (or code distance) ρ of C is defined as:

$$\rho = \min\{|A| \in \mathcal{P}_n | A \text{ is a logical operator}\} = \min\{|A| \mid A \in C_{\mathcal{P}_n}(S) \backslash S\}$$

An [[n,k]] code with distance ρ is also called an [[n,k,ρ]] code.

The basic idea for the definition of ρ is that it marks the number of qubits an operator has to minimally affect for it to change one codeword to another. An argument similar to that of the classical coding theory (see [8]) can be made to prove that a code with distance ρ can correct all errors E which satisfy $|E| < \frac{\rho}{2}$. A direct proof for quantum coding theory can be found in [1].

Example 2.26. We can calculate the code distance of ex. 2.12. We know two logical operators, $\overline{X} = ZZZZZZZZZ$ and $\overline{Z} = XXXXXXXXX$, and know that they represent the only two equivalency classes of logical operators, since the Shor code encodes a single qubit. This means we can get all logical operators this way, by going through the equivalency classes:

$$C_{\mathcal{P}_n}(S) = \{A\overline{X} \mid A \in S\} \cup \{A\overline{Z} \mid A \in S\} \tag{5}$$

From table 1 we get the generators of S, so we know in principle all logical operators. We can look at, for example $\overline{Z}g_8 = ZZZIIIIII$, or $g_2g_3g_6\overline{X} = ZIIIIZZII$. It is not hard to see that these operators are of minimal weight in the two classes from eqn. 5. This means that $\rho = 3$, and hence, the Shor code corrects $1 < \frac{3}{2}$ errors.

2.7 CSS Codes

A class of codes worth mentioning, a subclass of stabilizer codes, are CSS codes. CSS codes are named after their inventors: A. R. Calderbank, Peter Shor and Andrew Steane, and some codes we will be investigating in this thesis can be classified as CSS codes. The original construction actually uses classical codes, and is one of the advantages of CSS codes, since you can derive much information about the code from the classical codes used on its construction.

Definition 2.27. Let $D_1, D_2 \leq \mathbb{F}_2^n$ be two (classical) linear codes, which encode k_1 and k_2 bits respectively, i.e. one is an $[n, k_1]$ code and the other one $[n, k_2]$, such that $D_2 \subseteq D_1$. Further let D_1 and $D_2^\perp$ have both minimal distance $d \geq 2t + 1$ for some positive integer t. For $x \in \mathbb{F}_2^n$ let $|x\rangle \in \mathbb{C}^{2^n}$ be the n-qubit state indexed by the bit string x, and define $|x + D_2\rangle := \frac{1}{\sqrt{|D_2|}} \sum_{y \in D_2} |x+y\rangle$ for all $x \in D_1$. Now define $C := C(D_1, D_2) := \{|x+D_2\rangle \mid x \in D_1\}$ as the CSS code from D_1 over D_2.

Remark 2.28. The above defined $C = C(D_1, D_2)$ is an $[[n, k_1 - k_2, d]]$ code, with $d \geq 2t + 1$. A proof can be found in [2].

3 Geometrical Quantum Codes

We will now devote our attention to a subclass of stabilizer codes, namely geometrical stabilizer codes. Geometrical quantum codes are such, that they have an interpretation as embedded on a geometrical surface. This also makes it clear why they are of special interest for applications, as a quantum code on a computer would have to physically be somewhere, and thus have some geometric structure. This is why it is a good idea to study them. The codes on this thesis also make use of the geometric structure to define a concept of location, in particular for stabilizers to be confined to a specific location or area. Models for quantum coding which are localized are more physically plausible; in fact, many of the interactions described by the types of codes we will study have been already seen and studied in condensed matter physics, which makes it natural to study models with these local restrictions.

3.1 The Surface Code

A first example of a geometrical code, to get the idea of embedding qubits on a surface, will be the surface code on two dimensions with open boundary conditions (see [1]. note: it is called 'toric code with open boundary conditions' there). We call $\mathbb{Z}^2 =< \vec{e}_1, \vec{e}_2 >_{\mathbb{Z}} \subset \mathbb{R}^2$ the square lattice on two dimensions, where $\{\vec{e}_1, \vec{e}_2\}$ is an orthonormal basis of $\mathbb{R}^2$, and set $\Omega \subset \mathbb{Z}^2$ as follows:

$$\Omega := \{x\vec{e}_1 + y\vec{e}_2 \mid 0 \leq x \leq k_1, 0 \leq y \leq k_2\} \hat{=} \{0, 1, \ldots, k_1\} \times \{0, 1, \ldots, k_2\} \subset \mathbb{Z}^2$$

for some fixed $k_1, k_2 \in \mathbb{Z}$. We will further subdivide Ω in a partition of two disjoint sets, $\Omega_{odd} := \{x\vec{e}_1 + y\vec{e}_2 \in \Omega \mid x + y \equiv 1(\text{mod}2)\}$ and similarly $\Omega_{even} := \{x\vec{e}_1 + y\vec{e}_2 \in \Omega \mid x + y \equiv 0(\text{mod}2)\}$ This partition is just for clarity: we will not think of every point having a qubit, but rather, only the points in Ω_{even}, and on the sites of Ω_{odd} we will place stabilizer operators. Note that this yields $n := \frac{1}{2}((k_1 + 1)(k_2 + 1) - 1) = 2k_1k_2 + k_1 + k_2 + 1$ sites on Ω_{even} and thus n qubits. We will denote a single-qubit operator A acting on the qubit at site u as A_u, and define two types of operators which we will call plaquette and star operators.

For placing these two operators, we will further partition Ω_{odd}. Let $\Omega_P := \{x\vec{e}_1 + y\vec{e}_2 \in \Omega_{odd} \mid x \equiv 1(\text{mod } 2)\}$ and $\Omega_T := \{x\vec{e}_1 + y\vec{e}_2 \in \Omega_{odd} \mid x \equiv 0(\text{mod } 2)\}$.

Definition 3.1. Using the notation introduced above, let

$$\forall u \in \Omega_P : P_u := \begin{cases} Z_{u-\vec{e}_1} Z_{u+\vec{e}_1} Z_{u+\vec{e}_2} & \text{for } x\vec{e}_1 + 0\vec{e}_2 \\ Z_{u-\vec{e}_1} Z_{u+\vec{e}_1} Z_{u-\vec{e}_2} & \text{for } x\vec{e}_1 + k_2\vec{e}_2 \\ Z_{u-\vec{e}_1} Z_{u+\vec{e}_1} Z_{u+\vec{e}_2} Z_{u-\vec{e}_2} & \text{otherwise} \end{cases} \tag{6}$$

$$\forall u \in \Omega_T : T_u := \begin{cases} X_{u-\vec{e}_2} X_{u+\vec{e}_2} X_{u+\vec{e}_1} & \text{for } 0\vec{e}_1 + y\vec{e}_2 \\ X_{u-\vec{e}_2} X_{u+\vec{e}_2} X_{u-\vec{e}_1} & \text{for } k_1\vec{e}_1 + y\vec{e}_2 \quad \forall \\ X_{u-\vec{e}_1} X_{u+\vec{e}_1} X_{u+\vec{e}_2} X_{u-\vec{e}_2} & \text{otherwise} \end{cases} \tag{7}$$

We call P_u a plaquette operator, and T_u a star operator. As already suggested by their definition, we will place P operators in Ω_P and T operators in Ω_T. See figure 1

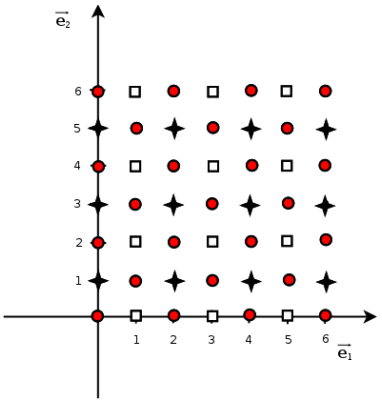

Figure 1: The generators of the surface code. The red circles represent the qubits: Ω_{even}, the white squares represent the plaquette operators: Ω_P, and the black stars represent the star operators: Ω_T.

Remark 3.2. Set $S := \langle P_u, T_v \mid u \in \Omega_P, v \in \Omega_T \rangle \leq \mathcal{P}_n$. Then S is a stabilizer group, and thus defines a stabilizer code on the n qubits on Ω_{even}.

Proof: From the way we defined the operators, we know that there are no overall factors involved, and we thus only have to prove that S is abelian. To do that, observe first that all operators of the same type commute with each other, since the P operators only have Z Pauli operators, and the T ones similarly only have X Pauli operators. We then only have to focus on the commutator of P_u and T_v for all $u \in \Omega_P, v \in \Omega_T$, and only when there is overlapping in their support (otherwise they trivially commute). It is easy to see, that when these operators overlap, they always do so in an even number of sites, and thus commute (see figure 1). □

Now that we know that S is a stabilizer group and defines a code, we can try to analyze it a bit. We will restrict here to the dimension of the code. Further analysis -like the minimal distance - can be found in [1].

Proposition 3.3. For any k_1, k_2 the 2D surface code with open boundary conditions encodes a single qubit.

We will prove this using theorem 2.20. For this, we first need to find a set of independent generators for the stabilizer group. We will start with the set we have now: $G := \{P_u \mid u \in \Omega_P\} \cup \{T_v \mid v \in \Omega_T\}$. By the very definition of the code, this set G generates the stabilizer group $S = \langle G \rangle$. We would like to find an independent subset of G, and to do this, we simply look for linear dependencies in symplectic notation: Let $\varphi : S \to \mathbb{F}_2^{2n}$ be the group homomorphism that maps S to its symplectic notation. By definition, the set of vectors $\varphi(G)$ is linearly independent if the following equation has no solution other than $a_u = 0 \, \forall \, u \in \Omega_{odd}$:

$$0 = \sum_{u \in \Omega_P} a_u \varphi(P_u) + \sum_{v \in \Omega_T} a_v \varphi(T_v) \overset{\varphi \text{ is an homomorphism}}{=} \varphi\Big(\prod_{u \in \Omega_P} P_u^{a_u} \prod_{v \in \Omega_T} T_v^{a_v}\Big) \quad (8)$$

This means nothing more than that $\prod_{u \in \Omega_P} P_u^{a_u} \prod_{v \in \Omega_T} T_v^{a_v} \in \mathrm{Ker}(\varphi)$. It is easy to see that for the sympletic notation for the whole Pauli group $\tilde{\varphi} : \mathcal{P}_n \to \mathbb{F}_2^{2n}$ the kernel is $\mathrm{Ker}(\tilde{\varphi}) = \{\pm iI, \pm I\}$, since $\tilde{\varphi}(A) = 0$ means that the exponent in all X_i and Z_i is 0. Since $S \cap \mathrm{Ker}(\tilde{\varphi}) = \{I\}$, it must be that

$$\prod_{u \in \Omega_P} P_u^{a_u} \prod_{v \in \Omega_T} T_v^{a_v} = Id \quad (9)$$

We know that the P_u are only Z-type operators, and the T_u are only X-type. Therefore, products of P_u's cannot be the inverse of products of T_v's and vice-versa, which means that equation 9 has to hold separately for both T and P operators:

$$\mathcal{I}_P := \prod_{u \in \Omega_P} P_u^{a_u} = I \quad (10)$$

$$\mathcal{I}_T := \prod_{v \in \Omega_T} T_v^{a_v} = I \quad (11)$$

If $\mathcal{I}_P = I$, it means that it acts on every qubit as the identity, we can hence look at the qubits directly: Consider the site at the origin ($\vec{0}$): The only P operator acting on it is $P_{\vec{e}_1}$ which acts as a Z on $\vec{0}$: If $\mathcal{I}_P$ is to act as the identity on $\vec{0}$, the exponent of $P_{\vec{e}_1}$, $a_{\vec{e}_1}$ must be 0. If we go on with the next qubit in that direction, the one at $2\vec{e}_1$, is acted upon by two P operators: $P_{\vec{e}_1}$ and $P_{\vec{e}_3}$, which act both with a Z on the qubit at $2\vec{e}_1$. Since, however, $a_{\vec{e}_1} = 0$, it means that $a_{\vec{e}_3}$ must be it too, otherwise $\mathcal{I}_P$ would act as a Z on the qubit at $2\vec{e}_1$. We can go on like this and see that all exponents for this row must be 0, i.e. $a_{(2k+1)\vec{e}_1} = 0, k = 0, 1, 2, \ldots$. We go on to the next column, starting with the qubit at $u = \vec{e}_1 + \vec{e}_2$: The P operators acting on this are $P_{\vec{e}_1}$ and $P_{\vec{e}_1 + 2\vec{e}_2}$. A similar reasoning, since we know that $a_{\vec{e}_1} = 0$, yields $a_{\vec{e}_1 + 2\vec{e}_2} = 0$.

In a similar fashion we can go on for all $P_u, u \in \Omega_P$ to show that $a_u = 0$. From an analogous argument we can see that the same holds for all $T_v, v \in \Omega_T$, and hence, the generator set G is already independent! It is a matter of simple combinatorics to see that $|\Omega_{odd}| = 2k_1k_2 + k_1 + k_2 = |\Omega_{even}| - 1 = n - 1$. Theorem 2.20 immediately yields the statement: the surface code encodes $k = n - (n - 1) = 1$ qubits.

This code is called the surface code with open boundary conditions since we assume that the end points of the code are open, and do not interact with anything else besides their inner neighbours in the lattice. If we identify the points in the end with the beginning, i.e. look at the points modulo the length: $\tilde{\Omega} = \{(x \bmod k_1)\vec{e}_1 + (y \bmod k_2\vec{e}_2 \mid x, y \in \mathbb{Z}\}$ then we get what is called the toric code with periodic boundary conditions. From this identification it is also clear why the code is called toric code. With an argument similar to proposition 3.3 it can be shown that the toric code with periodic boundary conditions always encodes 2 qubits.

3.2 Self-correcting Quantum Memory

The main goal of quantum error correction is, of course, to produce codes that can be used on a quantum computer. One very important direct application for codes is storage. We want to be able to reliably store and retrieve data, and on a quantum computer, qubits. If we have any kind of error correcting codes, even codes which immediately require an active process of error correction (usually designed for transmitting data over a faulty medium), we can find a way to store data. We simply actively do the error correction with some regularity which would depend on the code and the medium, and get a reliable way to store. This, however, has a great disadvantage: active error correction requires active use of diverse technologies: gates to do the error correction which wear down with time, measurements of syndromes, etc, all of which make it more complex and limit the useful life of the storage medium.

There is another way of reliably storing data, which can be more long-term and is in principle more energy-efficient: a self-correcting memory. The main idea behind a self-correcting quantum memory would be for it to be a quantum code, which after being perturbed, naturally tends to return to the original codeword, thus correcting the error 'by itself'. The feasibility of such a self-correcting quantum memory in a 3-dimensional geometrical quantum code is still an open problem [9], and is the main motivation for the codes studied in this thesis.

3.3 The STS model

Our first approach to finding a self-correcting quantum memory will be the restriction to a certain class of geometrical quantum codes. We will study codes embedded in a 3-dimensional lattice, for which we will assume periodic boundary conditions. We will make further assumptions about the codes we study, assumptions which, however, have a physical motivation.

- Locality:

  As discussed at the beginning of this section, one of the main assumptions we make on most geometrical quantum codes is locality. If geometrical quantum codes should result from interactions between the qubits, it is very reasonable to assume they will do so only locally. We will formally define this as a restriction on the size of the support of the stabilizer group elements, which we will require to be constant; in particular, independent of the size of the lattice ($\mathcal{O}(1)$).

- Translation-Symmetry:

  The next assumption we will make is translation symmetry. This basically means that the code 'looks' the same from every point in the lattice: a physical symmetry of the code, which is a plausible assumption. A more formal definition will be given in section 4.

- Scale-Symmetry:

  The last assumption is that of scale-symmetry of the code space dimension; this means that the code space dimension should not depend on the size of the lattice. A good motivation for this comes from the intuition that there is a tradeoff between the number of encoded qubits and the minimal distance of the code [11]. For this reason "it may be legitimate to limit our considerations to the cases where the number of logical qubits k remains small when the system size increases" [9].

  This assumption is probably the least well-founded of the three, and while reasonable, I would argue is not an assumption that has to be made. In a search for any plausible self-correcting quantum memory, finding a bad code - with a small distance, and/or not coding many qubits - is still better than no code at all.

We call stabilizer codes which satisfy these three conditions STS codes (for Stabilizers with Translation and Scale symmetries), and note that it is in fact a very large class of codes.

Example 3.4. The toric code with periodic boundary conditions is an STS code: From the stabilizers we immediately see both locality and translation-invariance of the stabilizers. The analysis of the code space dimension (which is always 4, for 2 qubits) immediately satisfies the third condition.

3.4 A Classic Self-Correcting Memory And The Energy Barrier

Now that we have a first model to search for self-correcting memory, it would be good to also get a feel for what a self-correcting quantum memory is, and how it can work. To do this we will start with an example for a classically self-correcting memory.

In an $L \times L$ lattice consider the following 2D Ising model:

$$H = -\sum_{x,y} Z_{x\vec{e}_1+y\vec{e}_2} Z_{(x+1)\vec{e}_1+y\vec{e}_2} - Z_{x\vec{e}_1+y\vec{e}_2} Z_{x\vec{e}_1+(y+1)\vec{e}_2}$$

The ground space of this Hamiltonian is spanned by $|0\ldots0\rangle$ and $|1\ldots1\rangle$ and can thus be used for storing a classical bit: $1 \mapsto |1\ldots\rangle, 0 \mapsto |0\ldots\rangle$. If we use this system as a memory, it will interact with the environment with time and produce errors. For these errors to make an error in the encoded classical bit, they have to take $|0\ldots\rangle$ to $|1\ldots\rangle$ or vice-versa. This means errors will have to flip all the spins (which are the qubits in this particular example) from $|0\rangle$ to $|1\rangle$ or vice-versa, which requires an energy that is $\mathcal{O}(L)$. For large L this is a large *energy barrier* that has to be be crossed by the error-producing interactions (assumed to be mostly of thermic nature) in order to make an error on the encoded bit. In most cases, before the system has gathered enough energy from the environment to corrupt this bit, natural thermal dissipation has brought the system back to its ground state, correcting the errors that occurred. In this way, the system corrects the errors by itself.

We are interested in the same principle for quantum codes: where the code is the ground state and there is an energy barrier separating two ground states in such a manner, that the code will mostly correct itself before making an error (logical operator) on the encoded data. This energy barrier is central to the concept of a self-correcting memory, as it has to be 'large' for self-correction to be feasible. We will take here 'large' to mean not $\mathcal{O}(1)$, but at least $\mathcal{O}(L^q)$ for some $q \in \mathbb{R}$, and while in principle a small q would be good as well, we do rather look for a code with energy barrier that is $\mathcal{O}(L)$.

Theorem 3.5 (Yoshida)**.** There exists no 3-dimensional STS code that can work as a self-correcting quantum memory, i.e. all 3-dimensional STS codes have an energy barrier that is $\mathcal{O}(1)$. A proof can be found in [9].

3.5 The Haah Code

Noteworthy of mentioning is that in the examples specifically studied by Yoshida in [9], as he himself remarks, the codes have "string-like" logical operators. This means operators with support that looks like a string on the lattice: for a more formal definition see [4]. These string-like operators make the energy barrier $\mathcal{O}(1)$ since only the boundaries of the string affect the energy. This might lead to an idea of how to search for codes which might be a self-correcting quantum memory: codes without such operators.

An example of a code without string-like logical operators was proposed by Haah in [3].

In the next section we will analyze this code, in particular it's code space dimension as a function of the lattice size L. We will see that it depends very strongly on L and for that reason it is not scale-invariant, hence, not an STS code. In fact, in [3] Haah shows that this code has no string-like logical operators. While this does not mean that the code is a self-correcting quantum memory, Yoshida's theorem (3.5) does not apply, and it certainly is a plausible candidate for a 3-dimensional self-correcting quantum memory!

4 Translation-Invariant Stabilizer Codes

In section 3 we got an introduction to geometric codes, and the problem of finding a self-correcting quantum memory. From theorem 3.5 we know that STS models are not the way to go in 3 dimensions, and mentioned the Haah code, which ignores the third condition on STS codes, namely that of scale invariance. In this section we will further pursue this model, translation invariant, local geometrical stabilizer codes, without any restriction on the dependence of the number of encoded qubits on the size of the lattice.

4.1 Stabilizer codes on a lattice with periodic boundary conditions

We begin by formalizing a few of the concepts seen on section 3.

Definition 4.1. Let $B = (\vec{e}_1, \cdots, e_n) \subset \mathbb{R}^n$ be a Basis of $\mathbb{R}^n$. Here, we will assume B is an orthonormal basis, although it is not necessary. We call $\tilde{L} = \{\sum_{i=1}^n a_i e_i \mid a_i \in \mathbb{Z}\} =: \langle \vec{e}_1, \cdots e_n \rangle_{\mathbb{Z}}$ a lattice with basis B. We can think of this as a discrete arrangement of points which differ by multiple integers of the basis vectors (see fig. 2).

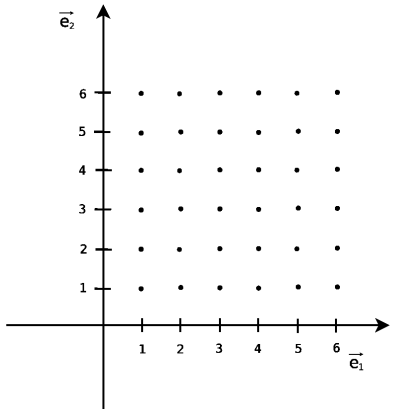

Figure 2: Part of a lattice in $\mathbb{R}^2$

We will slightly change the 'embedding' we saw on section 3.1: we will picture a qubit on every point of the lattice. There is of course no reason to restrict the number of qubits per point(also called site) to one; we could include m qubits per site and just label them from 1 to m. Since the lattice defined on 4.1 has infinitely many points, and we would want to consider finite codes, we have to include boundary conditions as well. We will consider

periodic boundary conditions: we consider points to be the same if they differ by a multiple of some number l_i in the direction e_i, i.e. for some $l = (l_1, \cdots, l_n)$ let $L := \tilde{L}/l\mathbb{Z} := \{\sum_{i=1}^{n}(a_i \bmod l_i)e_i\}$ be the factor lattice of $\tilde{L}$ modulo $l\mathbb{Z}$ or the Lattice $\tilde{L}$ with periodic boundary conditions l. Once we have labeled the qubits, it is easier to specify some classes of stabilizer generators. Note that this definition is not exactly the same as is usual in mathematics, but it is easier to understand and the resulting structures are equivalent.

Definition 4.2.

- Let C be a stabilizer code in the lattice with periodic boundary conditions L with stabilizer group S. We call C a translation-invariant code iff for each $s \in S$ and each $\vec{t} \in L$ there exists an $s' \in S$ which differs by $\vec{t}$, i.e. : for each $\vec{r} \in L$ s acts on the qubit labeled by $\vec{r}$ as s' acts on that labeled by $\vec{r} + \vec{t}$. If there is more than one qubit per site this has to hold for each of the qubits independently.

- For an operator $A_{\vec{0}} \in \mathcal{P}_n$ we define the translation-invariant generated group as the group including all translations of $A : \{A_{\vec{0}+\vec{t}} \mid \vec{t} \in L\}$.

As this whole section is motivated by it, we will dive in directly into the Haah code from section 3.5. We will develop the formalism and the method for calculating the dimension alongside applying it to the Haah code. Consider the 3D-lattice L with periodic boundary conditions $l_1 = l_2 = l_3 =: l \in \mathbb{Z}$ and two qubits per site ($m = 2$). We will label the qubits by $\vec{v}, j$; $j = 1, 2$, $v = \sum_{i=1}^{3}(a_i \bmod l_i)\vec{e}_i$. Consider then the two stabilizers

$$\tilde{X}_{\vec{0}} = X_{\vec{0},1}X_{\vec{0},2}X_{\vec{e}_1,2}X_{\vec{e}_2,2}X_{\vec{e}_3,2}X_{(\vec{e}_1+\vec{e}_2),1}X_{(\vec{e}_2+\vec{e}_3),1}X_{\vec{(e}_1+\vec{e}_3),1} \tag{12}$$

$$\text{and } \tilde{Z}_{\vec{0}} = Z_{(\vec{e}_1+\vec{e}_2+\vec{e}_3),1}Z_{(\vec{e}_1+\vec{e}_2+\vec{e}_3),2}Z_{\vec{e}_1,2}Z_{\vec{e}_2,2}Z_{\vec{e}_3,2}Z_{(\vec{e}_1+\vec{e}_2),1}Z_{(\vec{e}_2+\vec{e}_3),1}Z_{\vec{(e}_1+\vec{e}_3),1} \tag{13}$$

See figure 3.

We want this to be a translation-invariant code, so for each translation $\vec{t} \in L$ we take

$$\tilde{X}_{\vec{0}+\vec{t}} := X_{(\vec{0}+\vec{t}),2}X_{(\vec{e}_1+\vec{t}),1}X_{(\vec{e}_3+\vec{t}),1}X_{(\vec{e}_3+\vec{t}),2}X_{(\vec{e}_2+\vec{t}),1}X_{(\vec{e}_1+\vec{e}_3+\vec{t}),2}X_{(\vec{e}_2+\vec{e}_3+\vec{t}),2}X_{(\vec{e}_1+\vec{e}_2+\vec{e}_3+\vec{t}),1}$$

and $\tilde{Z}_{\vec{0}+\vec{t}}$ accordingly into S. One easily calculates from figure 3 that all generators commute, and since there are no overall factors on the generators, it is easy to see that this is indeed a stabilizer code ($-I \notin S$).

Questions about the properties of this code immediately arise: Do they encode qubits? If so, how many? And what is the minimal distance? In

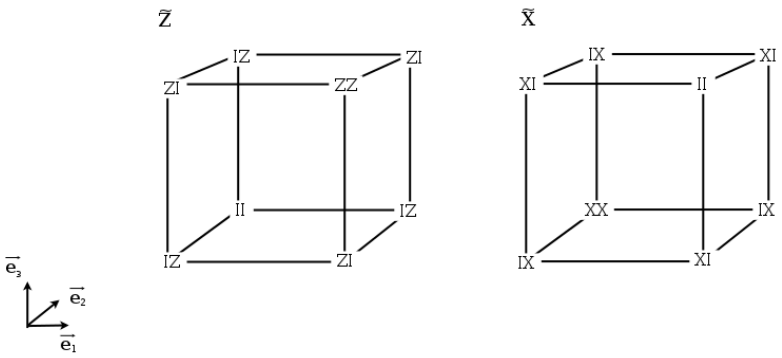

Figure 3: Haah code generators

this thesis we will address the first two questions, which are in some sense more fundamental. The number of encoded qubits is of course also highly interesting for this code as it will turn out to depend very much on L, breaking the scale-invariance and making the Haah code a non-STS code. The minimal distance, however, is still of extreme importance as it is a measure of the error-correcting properties of the code. It is a good point for further study.

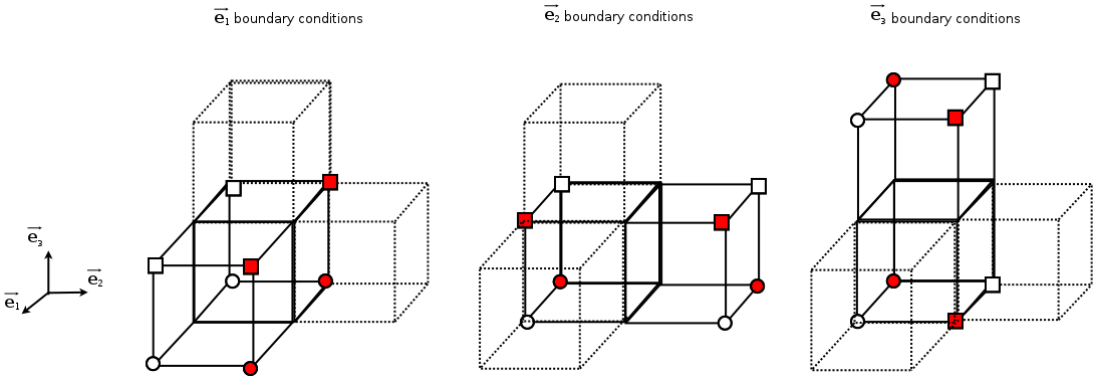

Figure 4: Boundary conditions for $l = 2$: the points marked with the same figures (including colour) are identified to be the same

Example 4.3. We begin examining the Haah code in the smallest non-trivial lattice: $l = 2$. This code can be thought of as an arrangement of cubes like the one on figure 3, but with the boundary conditions that identify the cubes as marked on figure 4. So, for instance, going from $\vec{e}_1$ to the right once $2\vec{e}_1 \equiv 0\vec{e}_1(\text{mod}2)$ takes you back to $\vec{0}$. Hence, the lattice L has 8 distinct points, and two qubits per point, for a total of 16 qubits. An important

observation, and one which applies to the Haah code in general is that it is a CSS code. This also means that the stabilizer group is a direct product of two smaller subgroups, $\mathcal{X}$ and $\mathcal{Z}$, which are generated by the X and Z type operators respectively. We see that $\tilde{Z}_{\vec{0}}$ is the same as $\tilde{X}_{\vec{0}}$ with Z's instead of X's, except for a reflection along the $\vec{e}_1 + \vec{e}_2 + \vec{e}_3$-axis. This means however, that the subgroups will be isomorphic (the reflection is just a renaming of the points) and will have the same number of independent generators. For this reason it is enough to determine the number of independent generators k for $\mathcal{X}$ and forget about $\mathcal{Z}$, the number of qubits encoded by the code altogether will be given by $n - 2k$.
If we label the points as in figure 5, and use the short notation (i, j) for $X_{i,j}$, we get the following table for all generators:

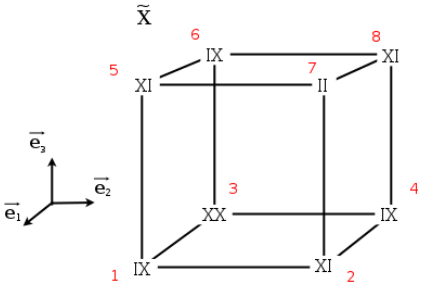

Figure 5: Labeling of the qubit sites for the Haah code generator $\tilde{X}$

translation $\vec{t}$	Stabilizers in short form (i,j)
0	(1,2)(2,1)(3,1)(3,2)(4,2)(5,1)(7,2)(8,1)
$\vec{e}_1$	(3,2)(4,1)(1,1)(1,2)(2,2)(7,1)(5,2)(6,1)
$\vec{e}_2$	(2,2)(1,1)(4,1)(4,2)(3,2)(6,1)(8,2)(7,1)
$\vec{e}_3$	(5,2)(6,1)(7,1)(7,2)(8,2)(1,1)(3,2)(4,1)
$\vec{e}_1 + \vec{e}_2$	(4,2)(3,1)(2,1)(2,2)(1,2)(8,1)(6,2)(5,1)
$\vec{e}_1 + \vec{e}_3$	(7,2)(8,1)(5,1)(5,2)(6,2)(3,1)(1,2)(2,1)
$\vec{e}_2 + \vec{e}_3$	(6,2)(5,1)(8,1)(8,2)(7,2)(2,1)(4,2)(3,1)
$\vec{e}_1 + \vec{e}_2 + \vec{e}_3$	(8,2)(7,1)(6,1)(6,2)(5,2)(4,1)(2,2)(1,1)

Table 2: Stabilizer generators for l=2

If we then write this in symplectic notation, we get the matrix:

$$\left(\begin{array}{cccccccccccccccc}
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1
\end{array}\right)$$

This is a rank 5 matrix over $\mathbb{F}_2$, which means that there is only 5 independent generators, hence altogether 10 with $\mathcal{Z}$, for a total of $16 - 10 = 6$ encoded qubits.

This was a pretty tedious calculation and this is only the smallest example! Luckily, while doing the calculation we can already see hints that there is an easier way of doing it. We first note that the translations just move the single qubit Pauli operators from the stabilizers around. By labeling the points in the lattice from 1 to 8 we saw that each translation can be understood as a simple permutation of those 8 points. In the language of group theory: the abelian group (with respect to addition) $G := L$ acts on the Pauli group, and the generators are the orbits of $\tilde{X}_{\vec{0}}$ and $\tilde{Z}_{\vec{0}}$. We can use this realization to generalize this method for an arbitrary L:

We noted above that the lattice L acts as a permutation group on the Pauli group elements. If we number the lattice points in a clever way, the action will be very easy to describe. It is a simple matter to implement said action, and input the first generator, to generalize the method we used on example 4.3 and for it to work on the code for a general size. In principle we are done: this whole approach reduces the problem to linear algebra, and calculating the rank of a matrix over $\mathbb{F}_2$. Unfortunately, this is a matrix with size $\mathcal{O}(l^6)$, and it soon becomes incalculable: Only for $l = 15$ the matrix has around 22 million entries and it took about 45 minutes to calculate it and its rank on an Intel i3 Pc with an implementation using a computer algebra system (which can definitely be improved upon, but gives an idea of the scaling with the size of the problem).

4.2 Translation-invariant codes on lattices as factor rings of polynomial rings

Fortunately, we can recognize more structure on the codes. We will try to develop the analysis of the structure directly in a general fashion, so it can

be used for any translation-invariant stabilizer codes on a lattice. We know that the stabilizer group is a vector space over $R := \mathbb{F}_2$, making it with the action of G what is called an RG-Module, where RG is the so-called group ring.

Definition 4.4. Let G be a finite group and R be a commutative ring with unity. Set the ring $RG := \{\sum_{g\in G} r_g g \mid r_g \in R\}$ with the product in the group extended such, that the distributive law holds. Then RG is called the group ring of G over R. A field K is also a ring, so that the group ring KG is also an algebra over the field K.

The following is partially based on unpublished notes by Jeongwan Haah:

Let G be a finite abelian group, and RG its group ring over $R = \mathbb{F}_2$. G will be the lattice, and although the group needs not be abelian for this analysis, a lattice should always be. At each $g \in G$ we place $m \in \mathbb{N}$ qubits. Up to an overall phase, we can then express every Pauli operator as $X_{f_1,1} \cdots X_{f_m,m} Z_{g_1,1} \cdots X_{g_m,m}$, where $f_i, g_i \in RG$ for all $i \in \underline{m}$ express where on the lattice the operator is supported. Note that this notation differs from the one introduced above, but there is no danger of confusion, as it shall be immediately clear from the context what it means (we will use this notation only when talking about group ring elements). For example, the operator $P_{\vec{e}_1} = Z_{\vec{0}} Z_{\vec{e}_1+\vec{e}_1} Z_{\vec{e}_1+\vec{e}_2}$ from the surface code (section 3.1) would be written as $X_f Z_g$ (we leave the qubit numbering out, since it is one qubit per site), with $f = 0, g = \vec{0} + \vec{e}_1\vec{e}_1 + \vec{e}_1\vec{e}_2$. Note that we write addition in the lattice multiplicatively, as it is the operation in the group $G = L$, to differentiate it from addition in the group ring RG.

Remark 4.5. Since the lattice G is a finite abelian group, by the structure theorem of finitely generated abelian groups (see, for example [7]) it is isomorphic to $\tilde{L}/(l_1, \ldots, l_n)\mathbb{Z}$ for some $l_1, \ldots, l_n \in \mathbb{N}$. Hence, it is with loss of generality that we can assume that structure for the lattice. In particular, this means that, for example, changing the way the boundary conditions are set will not change the structure of the code: The lattice will always be an abelian group, and structurally it would only mean a renaming of the points.

Remark 4.6. As a $\mathbb{F}_2$ vector space, an element of RG is the same as the Pauli group element in symplectic notation. In particular, stabilizer groups in RG have an $\mathbb{F}_2$ Dimension equal to the uniquely defined number of independent generators.

At this point, it might be useful to do a summary of the identifications with the group ring RG and Pauli group elements: As in remark 4.6, we can

think of an element of RG simply as the Pauli group element in symplectic notation. RG has two operations, product and sum:

- The sum in RG is the sum over $\mathbb{F}_2$ in symplectic notation. For the Pauli operators it is the same as the product in the Pauli group.
- The product with an element of $g \in G \subset RG$ (as opposed to a sum of such) is a translation of the Pauli element with the 'lattice vector' g.
- From those identifications above, we can understand the product in RG: $RG \ni r = \sum_{g\in G} a_g g; a_g \in R = \mathbb{F}_2$. If $r \neq 0$, multiplying the Pauli group element $d \in RG$ with r means that for each $a_g \neq 0$ we take the translation gd and multiply all those in the Pauli group: $rd = \sum_{g,a_g\neq 0} gd$.
- There is a special case which has been left out: $r = 0$. $0 \notin G \subset RG$. Product with 0 can be understood as a 'translation to nowhere', since it takes the Pauli group element to the identity in the Pauli group. Alternatively, one can understand $0d = d0, d \in RG$ as a sum of the translations of d for the identity, as above.

A further thing to notice is then the following: Let $d \in RG$ be a Pauli group element. Then, the translation-invariant generated group is the ideal $(d) = \{rd \mid r \in RG\}$: We first take all translations of d: $\{gd \mid g \in G \subset RG\}$, and then take the group generated by them, i.e. all products in the Pauli group of such elements: $\{\sum_{g\in H} gd \mid H \subset G \subset RG\} = (d)$.

Remark 4.7. Let $\bar{}\ : RG \to RG, g \mapsto \overline{g} := g^{-1}$ be the homomorphism which maps group elements to their inverses(since G is abelian, it is indeed a homomorphism). Set further for $R \ni r = \sum_{g\in G} r_g g$, $tr(r) := r_1 \in \mathbb{F}_2$, where 1 is the unit element in G. Then, the symplectic product of two Pauli group elements $P = X_{f_1,1} \cdots X_{f_m,m} Z_{g_1,1} \cdots X_{g_m,m}$,
$P' = X_{f'_1,1} \cdots X_{f'_m,m} Z_{g'_1,1} \cdots X_{g'_m,m}$ is given by

$$[P, P'] = \sum_{i=1}^{m} [X_{f_i,i} Z_{g_i,i}, X_{f'_i,i} Z_{g'_i,i}] = \sum_{i=1}^{m} \sum_{h\in G} (f_i)_h (g'_i)_h + (f'_i)_h (g_i)_h \qquad (14)$$

$$= \sum_{i=1}^{m} tr(\overline{f_i} g'_i + \overline{f'_i} g_i) = tr(\sum_{i=1}^{m} \overline{f_i} g'_i + \overline{f'_i} g_i) \qquad (15)$$

where the 2. equality follows from [1] and $(f_i)_h$ denotes the coefficient of h of f_i.

Definition 4.8 (CSS Code)**.** Set for $d_1, d_2 \in RG$, $\mathcal{X} = \langle X_{\overline{rd_1},1} X_{\overline{rd_2},2} \mid r \in RG \rangle$ and $\mathcal{Z} = \langle Z_{rd_1,1} Z_{rd_2,2} \mid r \in RG \rangle$ ($m = 2$ qubits per site). Then $S = \mathcal{X} \times \mathcal{Z}$ is a stabilizer group, since from eqn. all generators commute and by construction there is no phase factors, hence $-I \notin S$.

It is worth noting that this way we define a general class of translation-invariant CSS codes, including the Haah code, but the results from later do not need this CSS structure. To show the main result of this thesis, we will need a few mathematical definitions and technical results:

Remark 4.9.

- The function tr defines a bilinear form $(\cdot, \cdot) : R \times R \to \mathbb{F}_2$ through $(a, b) := tr(ab)$. It can be easily seen that the Gram-Matrix of this bilinear form in the basis G is the identity, thus, $(\cdot, \cdot)$ is non-degenerate and symmetric.

- For a subspace V of RG let $V^\perp := \{r \in RG \mid (r, v) = 0 \text{ for all } v \in V\}$ be the orthogonal complement of V with respect to $(\cdot, \cdot)$. Then $(V^\perp)^\perp = V$.

- Let $RG^* := \{\varphi : RG \to R \mid \varphi \text{ is linear}\}$ be the dual space of RG. Then we have a canonical (base independent) isomorphism between RG and RG^* given by $RG \ni a \mapsto (a, \cdot) \in RG^*$.

Definition 4.10. Let R,S be two commutative rings with unity, and $\varphi : R \to S$ be a ring homomorphism. Set $\mathrm{Im}(\varphi) := \{\varphi(r) \mid r \in R\}$. Then the Cokernel of φ is defined as:

$$\mathrm{Coker}(\varphi) := S/\mathrm{Im}(\varphi) = \{s + \mathrm{Im}(\varphi) \mid s \in S\} \tag{16}$$

Lemma 4.11. Let G be a finite abelian group, $R := \mathbb{F}_2 G$ be its group ring and $d_1, \ldots, d_m \in R$. Further let $I, J \lhd R$ be the two ideals in R defined by $I := \bigcap_{i=1}^m Ann(d_i) := \bigcap_{i=1}^m \{r \in R \mid rd_i = 0\} = \{r \in R \mid rd_i = 0 \text{ for all } i \in \underline{m}\}$, and $J := (d_1, \ldots, d_m) := \{\sum_{i=1}^m r_i d_i \mid r_i \in R \text{ for all } i \in \underline{m}\}$.[1] Then $R/J \cong I$ as vector spaces, which is equivalent to $\dim_{\mathbb{F}_2}(R/J) = \dim_{\mathbb{F}_2} I$.

Proof: Consider the following mapping:

$$\varphi : R \to I^*, R \ni r \mapsto r^*_{|I} = (i \mapsto (r, i) \forall i \in I) \in I^*$$

[1] If you have never seen the notation $Ann(d_i)$ just don't worry about it, take it as just a name

We want to show the assertion using the fundamental theorem on homomorphisms. If we can show that $\varphi(R) = I^*$ and $\mathrm{Ker}(\varphi) = J$ (see fig. 6), then it immediately follows from it that $R/J \cong I^*$ as $\mathbb{F}_2$-algebras, and this implies the assertion for vector spaces as well. Since $(\cdot,\cdot)$ is non-degenerate, we know that $^* : I \to I^*, i \mapsto i^* = (i,\cdot)$ is a surjective mapping, and since $I \subseteq R$, it is immediately clear that φ is surjective as well, and hence: $\varphi(R) = I^*$. Now we turn to $\mathrm{Ker}(\varphi) = \{r \in R \mid (r,\cdot)_{|I} \equiv 0\} = \{r \in R \mid (r,i) = 0 \,\forall\, i \in I\} = I^\perp$. This means we only need to show that $I^\perp = J$. Note that since $(\cdot,\cdot)$ is non-degenerate, $rd_i = 0 \Leftrightarrow (rd_i, g) = 0 \forall g \in G$, as G is a basis of RG. It follows then that

$$Ann(d_i) = \{r \in R \mid rd_i = 0\} = \{r \in R \mid (rd_i, g) = \mathrm{tr}(\overline{r}\overline{d_i}g) \tag{17}$$

$$= \mathrm{tr}(g\overline{r}\overline{d_i} = (\overline{g}r, d_i) = 0 \text{ for all } g \in G\} = (d_i)^\perp \tag{18}$$

Since $Ann(d_i)$ is an ideal, we can forget about the $\overline{g}$, as $rd_i = 0 \Rightarrow grd_i = 0$, which gives the last equality from equation 18. We get that:

$$I = \bigcap_{i=1}^{m} Ann(d_i) = \bigcap_{i=1}^{m} \overline{(d_i)^\perp} = \overline{(\sum_{i=1}^{m}(d_i))^\perp} = \overline{J^\perp}$$

It follows that $\mathrm{Ker}(\varphi) = I^\perp = \overline{J}$ and since $\overline{\ }$ is an isomorphism, $\dim_{\mathbb{F}_2} R/J = \dim_{\mathbb{F}_2} R/\overline{J} = \dim_{\mathbb{F}_2} I$. □

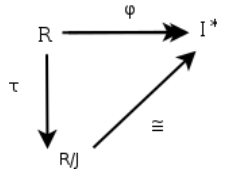

Figure 6: Fundamental theorem on homomorhisms, here φ is surjective as indicated by the double arrow, and τ is the canonical homomorphism: $\tau R \to R/J, r \mapsto r + J$

Definition 4.12. The free group on $\{x, y\}$ is defined as follows:

$$\mathrm{Fr}(\{x,y\}) := \{w \mid w \text{ is a finite product of } \{x, y, x^{-1}, y^{-1}\}\}$$

whilst we consider two such finite products to be the same if they differ by pairs of inverse elements at some point. For instance $xxyy \neq xyyx = xyy^{-1}yyx^{-1}xx (= xy(y^{-1}y)y(x^{-1}x)x)$. Any group G that can be generated by two elements $G = \langle g_1, g_2\rangle$ can be 'presented' as a factor group of $\mathrm{Fr}(x,y)$.

The kernel R of the mapping $x \mapsto g_1, y \mapsto g_2$ is called the relations of G and has exactly the products of g_1, g_2, expressed in x, y which multiply to 1. If R is generated by $r_1, \ldots, r_k$ we write $G \cong \langle x, y \mid r_1, \ldots, r_k \rangle$, this is called a presentation of G. From this definition it should be clear how to generalize this for n generators $x_1, \ldots, x_n$ instead of x, y.

Example 4.13. For example, a presentation of the 2-dimensional lattice L as an abelian group: $\tilde{L}/(l_1, l_2)\mathbb{Z}$, can be found as follows: We know that L is generated by the two elements $\vec{e}_1, \vec{e}_2$, so we take those for the mapping: $x \mapsto e_1, y \mapsto e_2$. We know that e_1, e_2 commute, so that $xyx^{-1}y^{-1}$ should be a relation (additively in L this means $(\vec{e}_1 + \vec{e}_2) - (\vec{e}_2 + \vec{e}_1) = \vec{0}$). We further know the boundary conditions: $l_1\vec{e}_1 = l_2\vec{e}_2 = \vec{0}$, which translate to $x^{l_1} = 1 = y^{l_2}$, so we should take those two relations as well. It is not hard to see that we are already done, as we can immediately tell by counting the number of elements in the group with this presentation. So, altogether we get $L \cong \langle x, y \mid xyx^{-1}y^{-1}, x^{l_1}, y^{l_2} \rangle$.

Remark 4.14. Let K be a field. Then the set of all polynomials (of finite degree) in the variables $x_1, \ldots, x_n$ and coefficients in K, denoted by $K[x_1, \ldots, x_n]$, is a commutative ring with unity. This can be read in most introductory algebra textbooks, for example [7].

Definition 4.15. Let K be a field and R be a K-Algebra, $d_1, \ldots, d_n \in R$ are called algebraically independent iff every polynomial $f \in K[x_1, \ldots, x_n]$ which satisfies $f(d_1, \ldots, d_n) = 0$ is the 0 polynomial, i.e. there exists no non-zero polynomial which satisfies $f(d_1, \ldots, d_n) = 0$. Elements $d_1, \ldots, d_n$ which are not algebraically independent are also called algebraically dependent.

Example 4.16. The polynomials $x + 1, y^2 + y \in K[x, y]$ are algebraically independent, which can be seen immediately from degree arguments. On the other hand, $x^2 + 1$ and x are not algebraically independent over $\mathbb{F}_2$, since $x^2 + 1 + (x + 1)^2 = 0$, which means that the non-trivial polynomial $f = x + (y + 1)^2 \in \mathbb{F}_2[x, y]$ satisfies $f(x^2 + 1, x) = 0$.

Proposition 4.17. Let G be a finite abelian Group, and

$$\langle x_1, \ldots x_n \mid r_1, \ldots r_m, x_1x_2x_1^{-1}x_2^{-1}, x_1x_3x_1^{-1}x_3^{-1}, \ldots, x_{n-1}x_nx_{n-1}^{-1}x_n^{-1} \rangle$$

be a presentation of G. Note that we always get the relations $x_ix_jx_i^{-1}x_j^{-1}$ for abelian groups but we are not interested in them here. Let further K be a field. Then the group ring $R := KG$ is isomorphic as a ring to $K[x_1, \ldots, x_n]/(r_1+1, \ldots, r_m+1)$ where $K[x_1, \ldots, x_n]$ where $(r_1, +1 \ldots, r_m + 1) := \{\sum_{i=1}^{m} a_i(r_i+1) \mid a_i \in K[x_1, \ldots, x_n]\}$ denotes the ideal of $K[x_1, \ldots, x_n]$ generated by $\{r_1 + 1, \ldots, r_m + 1\}$. Is $I = \bigcap_{i=1}^{k} Ann(d_i) \lhd KG$ for some $d_1, \ldots, d_k \in KG$, then $I \cong K[x_1, \ldots, x_n]/(r_1 + 1, \ldots, r_m + 1, d_1, \ldots, d_k)$

Proof: For the first part an isomorphism is given by $x_i \mapsto x_i$, which is well defined since G is abelian. For the second part we will first take the case $k = 1$. Look at the mapping $\varphi : R \to R, r \mapsto rd$. We immediately get that $\mathrm{Ker}(\varphi) = Ann(d), \mathrm{Im}(\varphi) = (d), \mathrm{Coker}(\varphi) = R/(d)$. From the fundamental theorem on homomorphisms we know $\mathrm{Im}(\varphi) \cong R/\mathrm{Ker}(\varphi) = R/Ann(d)$, which we can embed in R since $\mathrm{Im}(\varphi) \subset R$. We will call this embedding $R/Ann(d) \cong J \triangleleft R$. With this embedding we can look at $\mathrm{Coker}(\varphi) = R/\mathrm{Im}(\varphi) \cong R/J$(" $\cong R/(R/Ann(d))$"), and $R/J \cong Ann(d)$, hence $Ann(d) \cong \mathrm{Coker}(\varphi) = R/(d) \cong K[x_1, \ldots, x_n]/(r_1+1, \ldots, r_m+1, d)$, where the last isomorphy is immediately clear from the mapping $x_i \mapsto x_i$. Now for a general k, we know from prop. 4.11 that $R/J \cong I$, which means in particular from the fundamental theorem on homomorphisms, that there is a mapping $\varphi : R \mapsto R$ with $\mathrm{Im}(\varphi) = I$ and $\mathrm{Ker}(\varphi) = J$. The rest of the argument never explicitly used more of I and J, so it holds for this case as well. □

Remark 4.18. Let C be a translation-invariant stabilizer code 'embedded' on the lattice $L = G$. Let KG be the group ring over $K = \mathbb{F}_2$. Let $A_1, \ldots, A_k$ be a set of stabilizers such, that the stabilizer group is the translation-invariant group generated by the A_i, as in def. 4.2. Then (up to a scalar factor, for which we don't care here) we can write each A_i as $X_{f_1,1} \ldots X_{f_m,m} Z_{g_1,m} \ldots Z_{g_m,m}$. For simplicity of the argument we will assume that $k = m = 1$; a generalization of the result is straightforward. Then we can write the generator A_1 as $X_f Z_g$. But there is no need to separate f and g: If we define the ring R to be $KG^2 = KG \oplus KG$, we can write $d \in KG^2$ for the whole support of A_1 ($d = f \oplus g$). We can thus write the stabilizer code S as an ideal $I \triangleleft R$:

$$I := (d) := \{rd \mid r \in R\} \triangleleft R \tag{19}$$

The general case with k translation-invariant generators and m qubits per site simply yields an ideal generated by many elements, on a larger ring:

$$I = (d_1, \ldots, d_k) := \{r_1 d_1 + \ldots + r_k d_k \mid r_1, \ldots, r_k \in R = KG^{2m}\} \triangleleft R = KG^{2m} \tag{20}$$

Note that this is not necessarily a stabilizer code for any arbitrary $d_1, \ldots, d_k$; but (up to a scalar factor) every translation-invariant stabilizer code on a lattice can be written this way. We can always choose a set of generators $\tilde{d}_1, \ldots, \tilde{d}_l \in KG^2, l \leq k$ so, that it is algebraically independent.

Proposition 4.17 gives us, using remark 4.18, a good tool for calculating the dimension of a general translation-invariant code:

Theorem 4.19. Let C be a translation-invariant 3D stabilizer code with one qubit per site, generated translation-invariant from $A_1, \ldots, A_l$ as in def. 4.2 and $d_1, \ldots, d_k$ be the corresponding algebraically independent ideal generators as in remark 4.18. Further let $K = \mathbb{F}_2$ and $E_i := K[x_1, \ldots, x_6]/(x_1^{l_1} + 1, x_2^{l_2} + 1, x_3^{l_3} + 1, x_4^{l_1} + 1, x_5^{l_2} + 1, x_6^{l_3} + 1, d_i)$. Then the $\mathbb{F}_2$ dimension of the stabilizer group S (which is the same as the number of independent generators) is given by $\text{Dim } S = 2k|G| - \sum_{i=1}^{k} \text{Dim } E_i$.

Proof: We will first prove it for $k = 1$: Set the mapping $\varphi : KG^2 \to KG^2, r \mapsto rd$. The image of φ is given by $\text{Im}(\varphi) = \{rd \mid r \in KG^2\} = (d) \triangleleft KG^2$ and the kernel $\text{Ker}(\varphi) = \{r \in KG^2 \mid rd = 0\} = Ann(d) \triangleleft KG^2$. From the fundamental theorem on homomorphisms we know that $KG^2/Ann(d) \cong (d)$. If we look at KG^2 as a $\mathbb{F}_2$ vector space only, an operator in KG^2 is the same as the operator in symplectic notation. That is why, $(d) \cong S$ as $\mathbb{F}_2$ vector spaces (see remark 4.18). On the other hand, we know from prop. 4.17 that $E \cong \text{Ker}(\varphi)$, which gives the result. For a general k, we change the mapping to $\varphi : KG^{2k} \to KG^2, (r_1, \ldots, r_k) \mapsto \sum_{i=1}^{k} r_i d_i$. The image is $\text{Im}(\varphi) = (d_1, \ldots, d_k)$. The kernel is given by $\text{Ker}(\varphi) = \oplus_{i=1}^{n} Ann(d_i)$. That this is in the kernel is obvious, that it is the whole kernel is a consequence of the algebraic independence of the d_i's, since an element $\in Ker(\varphi) \backslash \oplus_{i=1}^{n} Ann(d_i)$ would immediately yield a non-trivial way of combining $d_1, \ldots, d_k$ to 0. The rest is analogous to the case for $k = 1$. □

Using proposition 4.17 it is straightforward to extend theorem 4.19 for m qubits per site.

Corollary 4.20. Set $E := \mathbb{F}_2[x_1, x_2, x_3]/(x_1^{l_1} + 1, x_2^{l_2} + 1, x_3^{l_3} + 1, d_1, d_2)$. Then, for the CSS code defined in 4.8 we have in symplectic notation $\dim \mathcal{X} = dim \mathcal{Z}$, and so we get the number of independent generators k as: $k = 2|G| - 2\text{Dim } E \Rightarrow 2\text{Dim } E = 2|G| - k$, and so, the number of encoded qubits is given by 2Dim E.

Proof. The proof is almost the same as in theorem 4.19, but we set the mapping $\varphi : KG \to KG^2, r \mapsto rd_1 + rd_2$, and make use of the fact that $\text{Dim } \mathcal{X} = \text{Dim } \mathcal{Z}$, which we can immediately see from their definition. □

Theorem 4.19 is a great result. Through it, we have managed to reduce the problem of finding the code space dimension to a well-studied one in commutative algebra: that of finding out the dimension of factor rings of polynomial rings over a field. To solve this we will need a little more foundations in this subject.

4.3 The dimension of factor rings of polynomial rings

For this section let $K = \mathbb{F}_2$ and $R := K[x_1, \ldots, x_n]$ be a polynomial ring over K on n variables. The next results actually depend on the monomial ordering, but this is not really important for us now. For this reason we will fix a monomial order:
We let $x_1 > x_2 > \ldots x_n$ and we compare first the grade of x_1, then of x_2 and so on. For example, $x_1^2x_2^2 > x_1x_2^7$ and $x_1x_2^2x_3^2 > x_1x_2^2$. This is usually called lexicographical ordering. In the general case we should also be careful with the coefficients, but since over $\mathbb{F}_2$ there is only 1 or 0 we will not make these distinctions for simplicity.

Definition 4.21. A monomial of R is a polynomial from the form $x_1^{i_1} \cdot x_n^{i_n}$ (in contrast to a sum of such). Let $f \in R$. We denote with $LM(f)$ the leading monomial of f with respect to the ordering we fixed, that is, the monomial which is the greatest with respect to $>$.

Theorem 4.22 (Multivariate division)**.** Let $F = [f_1, \ldots, f_m] \in R^m$ be a tuple of polynomials $\neq 0$ and let $f \in R$. Then there exist $q_1, \ldots, q_m, r \in R$ such, that $f = q_1f_1 + \ldots + q_mf_m + r$ and $r = 0$ or each monomial of r can't be divided by any of the $LM(f_i)$. We call r the remainder of the division and denote it by $r = \overline{f}^F$. A proof of this can be found at [10].

Definition 4.23.

- For two monomials $f = x_1^{i_1} \cdot x_n^{i_n}$, $g = x_1^{j_1} \cdot x_n^{j_n}$ we denote with $lcd(f, g) := x_1^{\max(i_1,j_1)} \ldots x_n^{max(i_n,j_n)}$ the lowest common multiple of f and g.
- Let $f, g \in R\backslash\{0\}$. The S-polynomial of f and g is defined by $S(f, g) := \frac{lcd(f,g)}{LM(f)}f - \frac{lcd(f,g)}{LM(g)}g$.
- Let $(f_1, \ldots, f_m) = I \triangleleft R$ be an ideal in R generated by $G = (f_1, \ldots, f_m)$. Then G is called a Gröbner basis of I, iff $S(f_i, f_j) = 0$ for all $i, j \in \underline{m}$.

The really interesting property of Gröbner basis for us is given by following theorem:

Theorem 4.24. Let $I \triangleleft R$ be an ideal and G be a Gröbner basis for I. Then a $\mathbb{F}_2$-Basis of R/I is given by $B := \{f + I \mid f$ is a monomial in R and $LM(g) \nmid f$ for all $g \in G\}$. A proof of this can be found in [10].

Example 4.25. Let $I = (b_1 := x+y+z+1, b_2 := y^2+yz+z^2+y+z+1, b_3 := z8 + 1, b_4 := yz^7 + yz^6 + z^7 + yz^5 + z^6 + yz^4 + z^5 + yz^3 + z^4 + yz^2 + z^3 + yz + z^2 + y + z + 1) \triangleleft K[x, y, z]$. $G := \{b_1, b_2, b_3, b_4\}$ is a Gröbner basis

for I, which is by the way the ideal corresponding to the E from corollary 4.20 to the Haah code for $l = 8$ (see ex. 4.27 to see how we come to this). Theorem 4.24 says we are only interested in the leading monomials of G. So, $\{LM(g) \mid g \in G\} = \{x, y^2, z^8, yz^7\}$. We need to find all monomials $f \in R$ which satisfy $LM(g) \nmid f$ for all $g \in G$: There is one monomial that always satisfies it: 1. We will go on looking for other monomials. As soon as we have one x in f, $x \mid f$, so we know that the exponent of x has to be 0 for all such f. We do the same for y: the exponent of y has to be 0 or 1, after that $y^2 \mid f$. Now we go to z: If the exponent of y is 0, we can go to z^7, after that $z^8 \mid f$; otherwise, if the exponent of y is 1, we can only go to z^6, since afterwards $yz^7 \mid f$. We have no more variables, so we are done. This means B from theorem 4.24 is given by:

$$B = \{1 + I, z + I, z^2 + I, \ldots, z^7 + I, y + I, yz + I, yz^2 + I, \ldots, yz^6 + I\} \quad (21)$$

This means in particular that $|B| = 15$, and so, Dim $E = 15$.

This theorem gives us a very easy way to determine the dimension of the factor ring, and this in turn the number of encoded qubits, once we have a Gröbner basis. Luckily there are many algorithms for computing this, of which probably the best known is Buchberger's:

Algorithm 4.26 (Buchberger)**.** Given some generators $F = (f_1, \ldots, f_m)$ of an ideal I, we can calculate a Gröbner basis G of I the following way:
Initialize G' := F.
Step 1: for each pair $f \neq g \in G'$ calculate $S(f, g)$
Step 2: Add each $\overline{S(f,g)}^{G'} \neq$ to G'.
Step 3: If no polynomials were added to G', terminate returning G'. Otherwise go back to step 1.

A proof that Buchberger's algorithm terminates can be found in [10]. There are of course some improvements to this, which make the calculation go faster. For instance, it is usually a good idea to reduce G' before step 1, that is, to replace each $g \in G'$ with $\overline{g}^{G'}$.

Example 4.27. With all this new machinery, we can go back to Haah's code. As a first example, we will do the same calculation as in 4.3 for $l = 2$ but with this method. We have to first find E from corollary 4.20. This means we have to look at the d_i. Examining figure 3, if we set the three directions as $\vec{e}_1 = x, \vec{e}_2 = y, \vec{e}_3 = z$ we get: $d_1 = 1+x+y+z$ and $d_2 = 1+xy+yz+xz$. So we have to find a Gröbner basis for $F = (d_1, d_2, p_3 := x^2+1, p_4 := y^2+1, p_5 := z^2 + 1)$.

We start by reducing F. The leading monomials of F are (x, xy, x^2, y^2, z^2), in the same order. This means that no leading monomial of F divides $LM(d_1)$. We now turn our attention to d_2: $x = LM(d_1) \mid xy = LM(d_2)$, so we replace d_2 with $d_2 + yd_1 = xz + y^2$, which we note by $d_2 \to +yd_1 = xz + y^2$. We see that now $x = LM(d_1) \mid xz = LM(d_2)$, so we replace again: $d_2 \to +zd_1 = y^2+yz+y+z^2+z+1$. We have eliminated x now! But d_2 is still not done, since $y^2 = LM(d_2) = LM(p_4)$ and z^2 too, so $d_2 \to +p_4+p_5 = yz+y+z+1$. Now we are done with d_2. We go on with $p_3 \to (y+z+1)d_1+p_4+p_5 = 0$, and p_4 and p_5 are reduced. So we ended up with $(1+x+y+z, yz+y+z+1, y^2+1, z^2+1)$. But if we do step 1 we see that we already have a Gröbner basis! For theorem 4.24 we only care about the leading monomials: (x, yz, y^2, z^2) and we can immediately write down a basis B for A: $B = \{1 + I, y + I, z + I\}$. We find that $\dim A = 3$, and by corollary 4.20 we have $2 \cdot 3 = 6$ total encoded qubits.

There is a lot of many well implemented algorithms for calculating Gröbner bases, and we have gained a method for calculating the number of qubits of a translation-invariant stabilizer code. Although a complexity analysis is very difficult, the main part of the calculation is indeed obtaining the Gröbner basis. While formally the complexity of Buchberger's algorithm seems to be bad, the algorithms tend to do much better than it, see [12]). In praxis it turns out to be much more efficient than the other method. On the same Intel i3 Pc, it took an implementation of this method for Haah's code roughly the same time to calculate the dimension for the first 2000 values of l than it took for $l = 15$ directly with linear algebra. The first few can be seen in table 3.

From table 3 we see that the code space dimension is highly dependent on the lattice size l, which makes it not be an STS code. A logical next step would be trying to apply Buchberger's algorithm to the E from corollary 4.20 for the Haah code for a general l to get a closed formula for the code space dimension as a function of l. While quite a few steps are indeed possible, one soon has to recursively define coefficients to keep the general case. Partial results on this direction already became inefficient for growing l, such, that it was faster to apply the algorithm for a particular l than to calculate the coefficients for this partial result. This led me to strongly suspect it would not be a fruitful pursuit, and it was better to abandon it.

l	Dim (E)	Encoded Qubits
2	3	6
3	1	2
4	7	14
5	1	2
6	3	6
7	1	2
8	15	30
9	1	2
10	3	6
11	1	2
12	7	14
13	1	2
14	3	6
15	25	50
16	31	62
17	1	2
18	3	6
19	1	2
20	7	14
21	1	2
22	3	6
23	1	2
24	15	30
25	1	2
26	3	6
27	1	2
28	7	14

Table 3: The code space dimension for the first few dimensions of the Haah code

5 Conclusion

We have studied two special models of geometrical stabilizer codes. First we reviewed STS codes, for which we assume translation symmetries and scale invariance. We saw that by Yoshida's theorem (3.5) STS codes are not the way to go for making a self-correcting quantum memory. We then dropped one of the assumptions, namely that of scale invariance. We saw with Haah's code that there exist codes which satisfy the new assumptions but are not subjected to Yoshida's theorem, and developed a formalism for these new types of codes: translation-invariant stabilizer codes on a lattice. With this new formalism using tools from commutative algebra, we managed to reduce the problem of finding out the code space dimension to a known problem from commutative algebra (theorem 4.19), namely that of finding out the dimension of a factor module of a polynomial ring. We introduced Buchberger's algorithm that solves it (algorithm 4.26), and applied it to Haah's code.

This results are staisfying: while they do not say there is a 3D self-correcting quantum memory, they at least don't say there is none. Further, they give us the tools to keep on investigating general translation-invariant codes, as the Haah code, where on such 3D self-correcting quantum memory might be. There is more to the formalism than the code space dimension. Looking at the code as an ideal in a commutative $\mathbb{F}_2$-algebra lets us use much more of the structure the code has. Since the mathematics describe such objects are very mature, it should let us get a more thorough understanding of the code. It would be interesting to further investigate this and try to obtain other properties from the code, such as the distance, from this formalism.

A Mathematical Supplement

A few mathematical concepts used in this thesis might not be familiar to all readers. I will provide here just the definitions, the bare minimum to know what it is. In most cases however, this will be by far not enough to really get a feel for the concepts. I would recommend looking them up; they should be in any standard algebra textbook. A good one, which covers pretty much all concepts used here is [7].

Definition A.1 (Group)**.** A set G (can be infinite, etc.) and a mapping $\cdot : G \times G \to G$ is called a group, iff it satisfies the following:

- $\cdot$ is associative: $a \cdot (b \cdot c) = (a \cdot b) \cdot c$ for all $a, b, c \in G$.
- There exists a neutral element $1 \in G$ for which the following is true: $1 \cdot a = a \cdot 1 = a$ for all $a \in G$.
- Every element has an inverse: $\forall\, a \in G \exists a^{-1} \in G : aa^{-1} = a^{-1}a = 1$.

Often the notation $\cdot$ is omitted, and the operation is written simply as a product: $a \cdot b =: ab$. A group which also satisfies the commutative law: $ab = ba \,\forall\, a, b \in G$ is called an *abelian group*.

Definition A.2 (Ring, Field)**.** A ring is a more complex structure than a group. There is a more general definition of ring, but we will limit ourselves here to a special class which encompasses all rings in this thesis: A set R, along with two operations: $\cdot, +$ is called a commutative ring with unity, iff it satisfies the following:

- R with $+$ is an abelian group. The unit element of this group structure is called 0.
- R with $\cdot$ satisfies all group axioms except the existence of inverses, and is also commutative. The unit element here is called 1.
- The distributive law holds: $a(b + c) = ab + ac$ for all $a, b, c \in R$.
- $1 \neq 0$.

This has a few implications, for instance, that $0 \cdot a = 0 \,\forall\, a \in R$, etc. If $R \backslash \{0\}$ with $\cdot$ is an abelian group (the inverse law holds), R is called a field.

Definition A.3 (Homomorphism, Isomorphism, Kernel)**.** Let G, H be two groups. A mapping $\varphi : G \to H$ is called a group homomorphism iff for all $g, g' \in G : \varphi(gh) = \varphi(g)\varphi(h)$, i.e. if it respects the group multiplication. If φ

is bijective, it is called an isomorphism. Note that this implies that $\varphi(1) = 1$. Similar definitions can be made for other structures, like homomorphism of rings, where the mapping has to be an homomorphism for + and respect the product for $\cdot$ as well. The kernel of a homomorphism is all what maps to 1 (or 0 in the case of rings): If $\varphi : G \to H$ is a group homomorphism, then $\mathrm{Ker}(\varphi) := \{g \in G \mid \varphi(g) = 1\}$. If $\phi : R \to T$ is a ring homomorphism, then $\mathrm{Ker}(\phi) := \{r \in R \mid \phi(r) = 0\}$. The kernel of a homomorphism is usually an important structure. It is easy to show that an homomorphism φ is injective $\Leftrightarrow \mathrm{Ker}(\varphi) = \{0\}$ (or $\{1\}$ for group homomorphisms).

Definition A.4. Let G be a group and M be a set. We say that the group G acts on M iff there is a mapping $G \times M \to M, (g, m) \mapsto gm$, which satisfies: $g(hm) = (gh)m$. This means nothing more that that there is a homomorphism $G \to S_M$, where S_M is the symmetric group of M, i.e. the group of all bijective mappings $M \to M$. For an element $m \in M$ the set $Gm := \{gm \mid g \in G\}$ is called the orbit of m.

Definition A.5 (Subgroup, Subring, Normal Subgroup, Ideal)**.** Let G,H be groups, R,T commutative rings with unity. A subgroup F of G (notation: $F \leq G$) is a subset of G, $F \subset G$, which is a group with the restriction of $\cdot$ to F: $\cdot_{|F}$. Similarly, a subring J of R (notation: $I \leq R$) is a subset of R which is a ring on its own, with the same operations from R restricted to it. Let further $\varphi : G \to H$ be a group homomorphism, then $N := \mathrm{Ker}(\varphi)$ is a subgroup of G which has even more special properties. It is called a normal subgroup (notation: $N \triangleleft G$). Similarly the kernel $I =: \mathrm{Ker}(\phi)$ of a ringhomomorphism, $\phi : R \to T$ is called an ideal, is a subring, and has more special properties.

Theorem A.6 (Fundamental Theorem of Finite Abelian Groups)**.** Let G be a finite abelian group. Then there exist $l_1, \ldots, l_n, \in \mathbb{N}$ such, that $G \cong \times_{i=1}^{n} \mathbb{Z}/l_i\mathbb{Z}$, where $\times$ denotes the direct product. A proof can be found, for example, in [7].

Definition A.7 (Module, Submodule, Vector Space, Subspace, Algebra)**.** Let R be a commutative ring with unity. An abelian group M with a product $\cdot : R \times M \to M$ is called an R-module if it satisfies the following (we write $\cdot$ multiplicatively and define right multiplication ,$M \times R \to M$, to be the same as left multiplication) :

- $a(v + w) = av + aw$, for all $a \in R, v, w \in M$.
- $(a + b)v = av + bv$, for all $a, b \in R, v \in M$.
- $a(bv) = (ab)v$, for all $a, b \in R, v \in M$.

- $1v = v$, for all $v \in M$.

If R is a field, M is also called a vector space. A subset of a module which is a module by itself with the restriction of the operations is called a submodule; a submodule of a vector space is also called a subspace. A ring, which is a K vector space at the same time is called a K algebra.

There is a few ways to construct further groups from those we have:

Definition A.8 (Direct Prodct, Coset, Factor Group, Factor Ring)**.** Let G, H be two groups and $N \triangleleft G$. The group defined by $G \times H := \{(g,h) \mid g \in G, h \in H\}$ with product $(g,h)(g',h') := (gg', hh')$ is called the direct product of G and H. An analogous definition can be made for direct sum of rings. The set $gN := \{gn \mid n \in N\}$ is called a coset. The set defined by $G/N := \{gN \mid g \in G\}$ is a group with the product $gNg'N := gg'N$, which is well defined since N is a normal subgroup. G/N is called the factor ideal of G modulo N. Let R be a ring and $I \triangleleft R$ an ideal. Set for $r \in R : r + I := \{r + i \mid i \in I\}$, then the factor ring is defined as $R/I := \{r + I \mid r \in R\}$, which is well defined as a ring since I is an ideal.

Theorem A.9 (Fundamental Theorem on Homomorphisms)**.** Let $\varphi : G \to H$ be a group-homomorphism. Then $\mathrm{Im}(\varphi) = \{\varphi(g) \mid g \in G\} \cong G/\mathrm{Ker}(\varphi)$.

Definition A.10. Let V be an n-dimensional K-vector space (K is a field) and $(\cdot,\cdot)$ be a bilinear form, i.e. a mapping $(\cdot,\cdot) : V \times V \to K$ which satisfies $(av+w, v') = a(v,v')+(w,v')$ and $(v', av+w) = a(v',v)+(v',w)$ for all $v, v', w \in V, a \in K$. Let $B := (b_1, \ldots, b_n) \in V^n$ be a basis of V. Then the $n \times n$ matrix $\mathcal{G}_{i,j} := (b_i, b_j)$ is called the Gram-matrix of $(\cdot,\cdot)$. In particular $(\cdot,\cdot)$ is symmetric, i.e. $(v,w) = (w,v)$ for all $v, w \in V$, iff the Gramm matrix is symmetric, and non-degenerate ($(v,w) = 0$ for all $w \in V \Rightarrow v = 0$) iff the gram matrix has full rank.

References

[1] Bernhard Leemhuis, *Geometrical Quantum Codes*(Master thesis), University of Amsterdam, 2010, available at `http://www.science.uva.nl/onderwijs/thesis/centraal/files/f2140130669.pdf`

[2] Michael Nielsen, Isaac Chuang, *Quantum Computation and Quantum Information*, 10^{th} Annniversary Edition (2010), Cambridge University Press

[3] Jeongwan Haah, *Local stabilizer codes in three dimensions without string logical operators*, arXiv:1101.1962v2 [quant-ph], 2011

[4] Sergey Bravyi, Jeongwan Haah, *On the energy ladscape of 3D spin Hamiltonians with topological order*, arXiv:1105.4159v1 [quant-ph], 2011

[5] Gabriele Nebe, Eric M. Rains, Neil James Alexander Sloane, *Self-dual codes and invariant theory*, Springer, 2006

[6] A. R. Calderbank, E. M. Rains, P. W. Shor, N. J. A. Sloane, *Quantum Error Correction Via Codes Over GF(4)*, arXiv:quant-ph/9608006v5, 1997

[7] Serge Lang, *Algebra*, Revised Third Edition, Springer, 2005

[8] Wolfgang Ebeling, *Lattices and Codes*, Vieweg Wiesbaden Braunschweig, 2002

[9] Beni Yoshida, *Feasibility of self-correcting quantum memory and thermal stability of toplogical order*, arXiv:1103.1885v3 [quant-ph], 2011

[10] William W. Adams, Philippe Loustaunau, *An Introduction to Gröbner Bases*, AMS, 1996 (reprint)

[11] S. Bravyi, D. Poulin, B. Terhal, *Tradeoffs for reliable quantum information storage in 2d systems*, arXiv:0909.5200v1 [quant-ph], 2009

[12] Mohamed Saeed Taha, *The Computational Complexity of Groebner Bases*, Univ. of Stellenbosch, 2006, available at `http://resources.aims.ac.za/archive/2005/saeed.ps`

[13] Marshall Hall, *The Theory of Groups*, AMS Chealsea Publishing, 1999