Quantum codes and quantum error correction

November 18, 2018

1 Introduction

In one of the standard models of a universal quantum computer, quantum states are initially prepared to reflect some input, than manipulated by quantum gates, i.e. unitary transformations, to execute a quantum algorithm and eventually measured to obtain the results. In this model, the evolution of the quantum states over time is subject only to the quantum gates and measurements that make up this procedure, and a quantum computer is usually described as a closed quantum system.

In reality, the situation is more challenging. The state of a quantum computer is constantly interacting with the environment, an interaction which we can try to minimize, but not fully suppress. This will imply changes to the state of the system that are beyond our control. In addition, as every measurement, also the measurements in a quantum computer will be subject to errors, and our realization of the quantum gates will not be perfect, but will be approximations to the ideal gate.

All these factors introduce **errors** into a quantum computation. For an individual gate, this error might still be small, but if we want to build an universal quantum computer that executes algorithms consisting of thousands or millions of gates, the errors are going to propagate and will soon render the result unusable. So we will have to detect and control these errors.

This situation is not entirely new. Also in classical computers, bits in memory or an a hard drive can flip due to some unwanted interaction with the environment, and computer scientists have developed error correction methods to suppress these errors. Typically, these mechanisms create one or more copies of the state and, after some time, compare the copies. If a bit in one of the copies has flipped, chances are that the other bits remain intact, and we can detect the error by comparing the bits.

However, in the world of quantum mechanics, the situation is more complicated. One obstacle is that it is not possible to create a copy of a quantum state, a fact that is known as the **No-cloning theorem** (see [8] and [9]).

Let us see - following the argument given in chapter 12 of [1] - why this is true. Suppose we had a device that is able to create a copy of a quantum state $|\psi\rangle$. More formally, this device would act on a system consisting of two subsystems. We would initialize the first subsystem in the state $|\psi\rangle$ and the the second subsystem in some initial state $|e\rangle$. Then the device would operate and leave us with a system in which the second subsystem is in the state $|\psi\rangle$ as well.

As any time evolution of a quantum system allowed by the laws of quantum mechanics, this device would be described by a unitary operator, acting on the

combined system. This operator, let us call it U, would therefore act on the states according to the rule

$$U(|\psi\rangle|e\rangle) = |\psi\rangle|\psi\rangle$$

where we use the usual notation for a tensor product. This should be true for any state $|\psi\rangle$. Therefore, given any two states $|\psi\rangle$ and $|\Phi\rangle$, we would have the relations

$$U(|\psi\rangle|e\rangle) = |\psi\rangle|\psi\rangle$$
$$U(|\Phi\rangle|e\rangle) = |\Phi\rangle|\Phi\rangle$$

Let us now compute the scalar product of the two states on the right hand side. First, we can do this directly using the definition of the scalar product in a tensor product space and obtain

$$\langle|\psi\rangle|\psi\rangle, |\Phi\rangle|\Phi\rangle\rangle = \langle\psi, \Phi\rangle\langle\psi, \Phi\rangle$$

On the other hand, we could use the fact that the operator U is unitary to equate the tensor product with the tensor product of the states on the left hand side of our equation, and obtain

$$\langle|\psi\rangle|\psi\rangle, |\Phi\rangle|\Phi\rangle\rangle = \langle|\psi\rangle|e\rangle, |\Phi\rangle|e\rangle\rangle = \langle\psi, \Phi\rangle\langle e, e\rangle$$

Without loss of generality, we can assume that the state $|e\rangle$ is normalized. Then comparing these two expression yields the equality

$$\langle\psi, \Phi\rangle = \langle\psi, \Phi\rangle\langle\psi, \Phi\rangle$$

This is only possible if either the two states $|\psi\rangle$ and $|\Phi\rangle$ are orthogonal or if $\langle\psi, \Phi\rangle = 1$. But of course we can easily find examples of states for which neither of these relations hold and the equality is violated. Thus our device will never be able to clone an arbitrary state.

There is a second reason why simple error correction mechanisms as they are known from the world of classical computing will in general fail in the quantum world - these mechanisms perform measurements, However, in a quantum computer, a measurement will typically destroy our superposition and therefore affect the outcome of a quantum algorithm.

So we need a smarter approach. Fortunately, over time, several potentials solutions have been developed. Since the nineties of the last centuries, we know a few **quantum codes** that despite these obstacles are able to encode quantum information into a larger number of qubits. This allows us to detect and correct errors. A large class of these codes can be described using a formalism known as the **stabilizer formalism** which describes these codes in terms of groups and their normalizers. Further, codes can be **concatenated**, i.e. we apply another coding procedure to the encoded state to further increase the error protection. This might introduce additional errors, but the **threshold theorem** states that we will correct more errors than we introduce and can therefore reduce the error probability to any desired level provided that the physical error rate is below some threshold.

In addition to codes that are the equivalent of classical codes, interesting codes arise as **topological codes**. These codes can be described in terms of

strings and cycles on a surface like a torus or a planar surface, and are related to the homology groups of the surface. These codes do not only have an interesting interpretation in terms of solid state physics, but acting on the encoded states can, in some cases, be seen as purely topological operations, leading us into the field of **topological quantum computation**. And there are other approaches to quantum error correction like measurement based approaches that we will not even be able to touch in these notes.

After this overview, let us now dive straight into the first and most straightforward - though incomplete - example of a quantum code.

2 The three bit code

The three bit code provides a simple example that demonstrates that it is still possible to account for errors in a quantum setup. Suppose that we are looking at a communication channel over which we are somehow able to transmit individual qubits. Suppose further that the only error that can occur during this transmission is a bit flip, i.e. that with a probability p, the states $|0\rangle$ and $|1\rangle$ are switched during the transmission (this is of course not realistic, and we will investigate other sources of errors below). Using the standard notation

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

for the bit-flip operator, a state $|\psi\rangle$ will therefore be converted into the state $X|\psi\rangle$ with probability p.

Now suppose that we encode every single qubit state in a three-qubit state before transmitting it. Thus we use the following encoding

$$|0\rangle \mapsto |000\rangle$$
$$|1\rangle \mapsto |111\rangle$$

It is not difficult to see that this encoding can in fact be realized by a unitary circuit - the circuit in figure 1 will do the trick.

Let us see why this works. If $|\psi\rangle = |0\rangle$, the two CNOT gates do nothing, so the outcome is $|000\rangle$. If $|\psi\rangle = |1\rangle$, both CNOT gates will flip the respective qubit, so the outcome is $|111\rangle$. Thus a general state $|\psi\rangle = a|0\rangle + b|1\rangle$ will be encoded as

$$a|000\rangle + b|111\rangle$$

Note that this not equal to the tensor product $|\psi\rangle|\psi\rangle|\psi\rangle$ and therefore this circuit does not violate the no-cloning theorem.

We now send each of the three qubits through our channel independently. We then perform an operation called a **syndrome measurement**. Specifically, we apply the circuit displayed in figure 2.

This circuit uses two ancilla qubits to effectively compare the parity of the individual qubits transmitted by the channel. To see this, note that the combination of the first two CNOT gates act on the same ancilla qubit. If the first

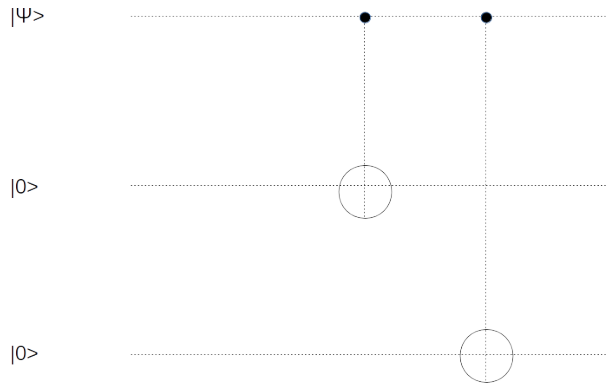

Figure 1: Three qubit encoding

two transmitted qubits are both $|0\rangle$, the ancilla is unchanged. But this is also true if both qubits are $|1\rangle$, as then the ancilla is inverted twice. So the first ancilla qubit will compares the first and second transmitted qubit. Similarly, the second ancilla qubit compares the first and third qubit.

Now suppose that at most one qubit has been flipped during the transmission, i.e. during the transmission, one of the following four unitary operators has been applied to our initial state $|\psi\rangle = a|000\rangle + b|111\rangle$.

$$\begin{aligned} 1 \otimes 1 \otimes 1 \\ X \otimes 1 \otimes 1 \\ 1 \otimes X \otimes 1 \\ 1 \otimes 1 \otimes X \end{aligned}$$

Let us now write down the state of the overall system after performing the syndrome measurement, i.e. after adding the two ancilla qubits and applying the syndrome measurement circuit. We obtain

$$\begin{aligned} (a|000\rangle + b|111\rangle)|00\rangle \\ (a|100\rangle + b|011\rangle)|11\rangle \\ (a|010\rangle + b|101\rangle)|10\rangle \\ (a|001\rangle + b|110\rangle)|01\rangle \end{aligned}$$

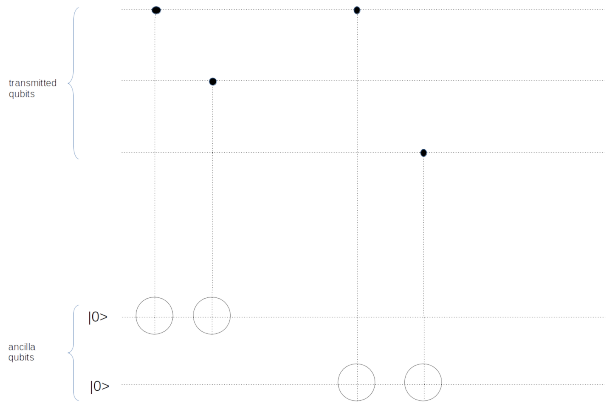

Figure 2: Syndrome measurement

We can now measure the two ancilla bits. This measurement will not change the state of the three transmitted qubits, but tell us which of the four cases we are in! Also note that if we measure the ancilla qubits, we will obtain the result with probability one in each of the four cases. Thus after performing the measurement, we know with certainty which of the four operators has been applied during the transmission and can perform a correction by flipping the respective bit once more. Of course this breaks down if more than one qubit is flipped, but as the probability for this to happen is smaller than p, we have been able to reduce the error rate.

So what we have learned so far is that it is possible, also in the quantum world, to detect and protect against pure bit flip errors without destroying the superposition. But there is more we can learn from that example. In fact, let us revisit our original assumption that the only thing that can go wrong is that the operator X is applied to some of the qubits and allow a more general operator. Suppose, for instance, that our error is represented by applying to at most one of our qubits the operator

$$E = (1 - \epsilon)1 + \epsilon X$$

for a small ϵ. This operator is in general not unitary, and thus the operator represents the impact of decoherence, i.e. an interaction with the environment (see section 5.1 of [10] for a discussion of the most general type of errors that we have to deal with). Suppose we apply this operator to the first qubit during the transmission. Then our state after the transmission will be

$$(1 - \epsilon)(\alpha|000\rangle + \beta|111\rangle) + \epsilon(\alpha|100\rangle + \beta|011\rangle)$$

After applying the error syndrome measurement circuit, this state will again be in a superposition, namely

$$(1 - \epsilon)(\alpha|000\rangle + \beta|111\rangle)|00\rangle + \epsilon(\alpha|100\rangle + \beta|011\rangle)|11\rangle$$

Assuming that our original state was normalized, the square of the norm of this state will be $(1-\epsilon)^2 + \epsilon^2 = 1 - 2\epsilon$. Now we measure the syndrome. As always in quantum mechanics, measurement corresponds to projection onto an eigenspace. Thus, after the measurement, the state will - up to normalization - be one of the states

$$(\alpha|000\rangle + \beta|111\rangle)|00\rangle$$
$$(\alpha|100\rangle + \beta|011\rangle)|11\rangle$$

Thus the measurement will collapse the state into one of two states, where one state corresponds to no error at all, and one state corresponds to a bit flip error. In other words, whereas the actual error that was applied was a linear combination of the identity and a bit flip error, the process of detecting and correcting the error somehow forces the system to make a decision: either there is a full bit-flip error, or there is no error at all. This fact is called the **digitization of errors** (see also [10] for a more complete treatment). Thus instead having to prepare for a continuum of error operators, it suffices to fix a discrete set of errors - which obviously is of utmost importance for practical implementations.

3 Complete error codes

So far we have worked with a code which is able to protect against a bit flip error. But of course this is not the only type of error that can occur. At the first glance, it looks like there is a vast universe of potential errors that we have to account for, as in theory, the error could be any unitary operator. However, using arguments similar to the discussion in the last section, one can show that it suffices to protect against two types of errors: the bit flip error discussed above and the **phase flip error**, represented by the matrix

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

Note that the phase flip error has no classical equivalent, other than the bit flip error which can classically be interpreted as a bit flipping from one to zero or vice versa randomly. The first error correction code that was able to handle both, bit flip errors and phase flip errors (and combinations thereof), and therefore any possible type of error was described in 1995 by P. Shor in [12].

To understand Shor's code, it is useful to first design a code that protects against phase flip errors alone. The idea for this code is based on the relations

$$Z|+\rangle = |-\rangle$$
$$Z|-\rangle = |+\rangle$$

where we denote by $|+\rangle$ and $|-\rangle$ the alternative basis of a one-qubit Hilbert space given by

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = H|0\rangle$$
$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = H|1\rangle$$

Here H is the Hadamard operator which maps the standard basis to the basis given by these two vectors and is therefore described by the unitary matrix

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

In other words, applying a phase flip error is the same as applying the Hadamard operator, followed by a bit flip followed again by the Hadamard operator, i.e.

$$HXH = Z$$

This implies that a phase flip and a bit flip are two different views of the same thing and therefore that a code that is able to detect and correct bit flip errors is also able to correct phase flip errors if we apply the Hadamard matrix to convert back between these two views. Specifically, we can encode a state by applying the three bit code followed by the Hadamard operator on each qubit. Thus our code would be

$$|0\rangle \mapsto |+++\rangle$$
$$|1\rangle \mapsto |---\rangle$$

For the error syndrome measurement, we apply the Hadamard operator on each of the three qubits, followed by the circuit shown in figure 2, followed again by the Hadamard operator applied to each qubit. The values of the two ancilla qubits then again tell us whether a phase flip has occurred and for which qubit.

If, for instance, a phase flip is applied to the first qubit, a state $a|0\rangle + b|1\rangle$ becomes

$$a|-++\rangle + b|+--\rangle$$

Applying the Hadamard operator $H^{\otimes 3}$ then turns this state into the state that would be the result of a bit flip error, i.e. into

$$a|100\rangle + b|011\rangle$$

When we now apply our parity measurement circuit in figure 2, we obtain the state

$$(a|100\rangle + b|011\rangle)|11\rangle$$

Finally, we apply the Hadamard operator again to each of the three primary qubits, i.e. we apply $H^{\otimes 3}$, and obtain

$$(a|-++\rangle + b|+--\rangle)|11\rangle$$

We can now again measure the two ancilla qubits and use the result to distinguish between the four different error situations that can arise if we assume that at most one phase flip can take place, and use this information to apply the necessary corrections.

With this understanding, we are now ready to describe Shor's code that corrects phase flip and bit flip errors (and their combination ZX). The idea of that algorithm is to concatenate the encoding that protects against phase flip errors with the encoding that protects against bit flip errors.

Specifically, we encode our states as follows. We first encode once using the phase flip code. Thus, we map $|0\rangle$ to $|+++\rangle$ and $|1\rangle$ to $|---\rangle$. We then apply the bit flop code to each of the resulting qubits. The bit flip code maps $|0\rangle$ to $|000\rangle$ and $|1\rangle$ to $|111\rangle$, and therefore maps the linear combination $|+\rangle$ to $\frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$ and similarly $|-\rangle$ to $\frac{1}{\sqrt{2}}(|000\rangle - |111\rangle)$. Thus the full encoding uses nine bits and is given by

$$|0\rangle \mapsto \frac{1}{2\sqrt{2}}(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)$$
$$|1\rangle \mapsto \frac{1}{2\sqrt{2}}(|000\rangle - |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle)$$

Now suppose a bit flip error occurs on this encoded state, say on the first qubit. The second and third block of three qubits each are then unaffected, and the error to the first block corresponds to an error that occurs after applying a bit flip error correction code to the three bits in this block. It can therefore be detected and corrected as before.

Now let us see what happens if a phase flip occurs. First, let us introduce a shorthand notation. We abbreviate the three-qubit states that appear in the nine bit encoding as

$$|A\rangle = |000\rangle + |111\rangle$$
$$|B\rangle = |000\rangle - |111\rangle$$

With this notation, we can now simplify the expressions for the nine qubit encoding as follows.

$$|0\rangle \mapsto \frac{1}{2\sqrt{2}}|AAA\rangle$$
$$|1\rangle \mapsto \frac{1}{2\sqrt{2}}|BBB\rangle$$

Now let us calculate the result of applying the Hadamard operator to each block of the encoded state. A short calculation shows that

$$H|A\rangle = H(|000\rangle + |111\rangle) = \frac{1}{\sqrt{2}}(|000\rangle + |011\rangle + |110\rangle + |101\rangle)$$
$$H|B\rangle = H(|000\rangle - |111\rangle) = \frac{1}{\sqrt{2}}(|111\rangle + |001\rangle + |100\rangle + |010\rangle)$$

In the first line, each state contains an even numbers of ones, in the second line each, state contains an odd number of ones. Thus if we take an ancillary qubit with initial value $|0\rangle$ and apply a sequence of three CNOT gates to it, controlled by the three qubits, the result will be $|0\rangle$ for the first line and $|1\rangle$ for the second line. Thus this circuit will allow us to tell the state $|A\rangle$ and $|B\rangle$ apart.

Now let us take a look at the circuit in figure 3 that combines six of these detection circuits.

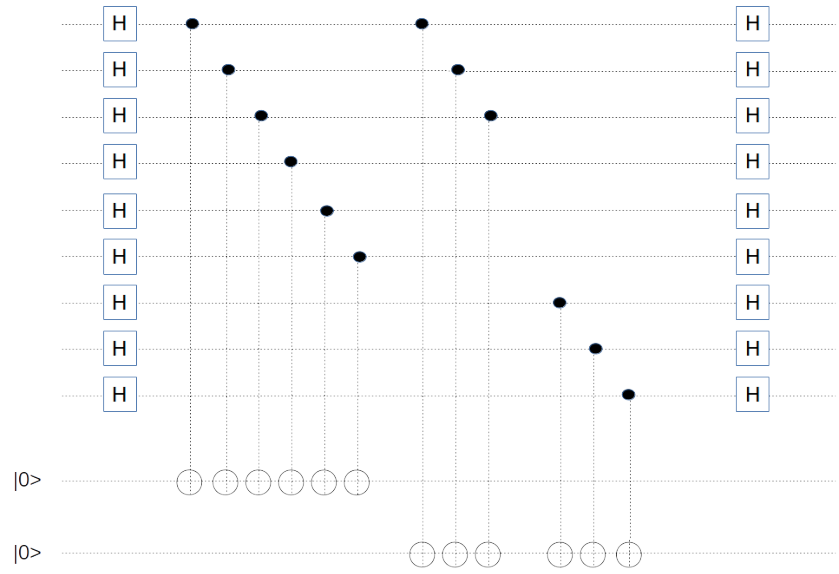

Figure 3: Nine qubit code - syndrome measurement

Repeating the arguments above for each of the blocks, we see that if we apply this circuit to the states $|AAA\rangle$ and $|BBB\rangle$, both ancillary qubits will be zero. Now suppose that a phase flip in one of the first qubits occurs in, say, the state $|AAA\rangle$. Clearly, this phase flip will result in the state $|BAA\rangle$. When we run the circuit on this state, the first three CNOT gates applied to the first ancillary qubit will give $|1\rangle$. The second set of three CNOT gates will not modify this state any more, so the first ancillary qubit will be $|1\rangle$. The same holds for the second ancillary qubit, and so the two ancillary qubits will be $|11\rangle$. If a phase flip occurs in the second block, we get the state $|ABA\rangle$. Now the first three CNOT gates leave the first ancillary qubit in state $|0\rangle$, whereas the second three

CNOT gates invert it, so that the first ancillary qubit will be $|1\rangle$. However, the second ancillary qubit will remain in state $|0\rangle$, so that the combinated state will be $|10\rangle$. Finally, the state $|AAB\rangle$ which is the result of phase flip in the third block will give us $|01\rangle$. A similar reasoning applies to a phase flip on the state $|BBB\rangle$. The upshot of this discussion is that very similar to the three qubit code, the value of the two ancillary qubits can be measured and will tell us in which block a phase flip occurred. Changing the phase of any of the qubits in this block will then correct the error. This shows that the Shor code can also be used to detect and correct phase flip errors, as long as only one bit flips.

Similarly, one can show that also a combination of phase flip and bit flip errors, i.e. the result of applying XZ, can be detected. Thus this code is able to detect all discrete errors and therefore - by the principle of the digitalization of errors - all errors as long as only one qubit is affected.

Let us now summarize some of the formal properties that this code has. First, the encoding can be described as identifying a 2-dimensional subspace C, called the **code space**, in the larger space spanned by all (in this case nine) qubits. More generally, the space C can have dimension 2^n and the full Hilbert space can have dimension 2^k, in which case we can encode n qubits in a larger set of k qubits (in the nine qubit code example, k = 9 and n = 1).

Hence the code space is spanned by a set of 2^n basis vectors $|\psi_i\rangle$ that we call **code words**. In the case n = 1, it is common practice to denote the codewords by $|0\rangle_L$ and $|1\rangle_L$ to indicate that they represent a **logical qubit** encoded using k **physical qubits**.

In addition, there is a set of **error operators**, i.e. a finite set of operators $\{E_\alpha\}$ acting on the larger Hilbert space. These operators represent the impact of discretized noise.

Now what are the conditions on C and the E_α that allow us to perform error correction? A first condition is that no matter which errors occur, we can still tell the code words apart. This is guaranteed if the various error operators map different code words onto mutually orthogonal subspaces, in other words if

$$\langle\psi_i|E_\alpha^+ E_\beta|\psi_j\rangle = 0$$

whenever $i \neq j$ and for all α, β. Thus, even in the presence of errors, the different code words will never overlap.

What about the case that both code words are the same? It is tempting to ask for the condition

$$\langle\psi_i|E_\alpha^+ E_\beta|\psi_i\rangle = \delta_{\alpha\beta}$$

i.e. to require that different errors map the same codeword to orthogonal subspaces. This would make recovery very easy. We could perform the measurements that correspond to projections onto these subspaces to detect the error and correct them by applying the inverse of the error operator. However, this condition is too restrictive. In the case of the nine qubit code, for example, a phase flip Z_1 and a phase flip Z_2 in the second qubit both map the state $|AAA\rangle$ to the same state, namely $|BAA\rangle$. However, the same applies for the correction, i.e. we do not have to distinguish between these two errors as the act similarly on the code space. In [14], a more general condition is presented which captures this case (see also [3]):

$$\langle\psi_i|E_\alpha^+ E_\beta|\psi_j\rangle = C_{\alpha\beta}\delta_{ij} \tag{1}$$

for a hermitian matrix $C_{\alpha\beta}$. If the matrix has full rank, the code is called **non-degenerate**, otherwise - as in the case of the nine qubit code - the code is called **degenerate**.

Of course this encoding generates some overhead. To represent one logical qubit, we need more than one physical qubit. For the Shor code, we have to use nine physical qubits to encode one logical qubit. It is natural to ask what the minimum overhead is that we need. In 1996, Steane discovered a code ([11]) that requires only seven physical qubits. In the same year, a code that requires only five qubits was presented ([15]) and it was shown that this is a lower bound, i.e. there is no error correction code that requires less than five qubits.

4 The stabilizer formalism

The formalism of stabilizer codes has been introduced by D. Gottesman in his thesis ([5]) and is very useful to identify and describe a large class of quantum error correction codes. To get the idea, let us first consider the action of products of Pauli matrices on n-qubit systems. In what follows, we will use an upper index to indicate the qubit on which an operator acts in a multi-qubit system. First, let us do a short calculation and determine the action of the product of two Pauli Z-operators on a two-qubit Hilbert space.

$$\begin{aligned}\sigma_z^1\sigma_z^2|00\rangle &= |00\rangle\\ \sigma_z^1\sigma_z^2|11\rangle &= |11\rangle\\ \sigma_z^1\sigma_z^2|10\rangle &= -|10\rangle\\ \sigma_z^1\sigma_z^2|01\rangle &= -|01\rangle\end{aligned}$$

Thus the hermitian operator $\sigma_z^1\sigma_z^2$ is able to compare the signs of the two qubits. A state in which these two qubits have the same sign is in the eigenspace with eigenvalue one, a state in which the signs differ is in the eigenspace with eigenvalue minus one.

Looking again at the Shor nine qubit code, this implies that the process of detecting a bit flip error can be implemented using measurements corresponding to these operators. In order to detect a bit flip error in the first block of three qubits, we can measure $\sigma_z^1\sigma_z^2$ and $\sigma_z^1\sigma_z^3$. The outcomes of these two measurements then correspond to the values of the ancillary qubits used in our syndrome measurement circuits and tell us whether a bit flip occurred and where - the same is true for the other blocks.

A similar observation holds for phase flip errors. In fact, it is obvious that - using the same notation as before - we have

$$\begin{aligned}\sigma_x^1\sigma_x^2\sigma_x^3|A\rangle &= |A\rangle\\ \sigma_x^1\sigma_x^2\sigma_x^3|B\rangle &= -|B\rangle\end{aligned}$$

Thus a measurement corresponding to the operator $\sigma_x^1\sigma_x^2\sigma_x^3$ can be used to distinguish between the states $|A\rangle$ and $|B\rangle$. Consequently, the syndrome measurement circuit for the phase flip error corresponds to measurements of the operators $\sigma_x^1\sigma_x^2\sigma_x^3\sigma_x^4\sigma_x^5\sigma_x^6$ and $\sigma_x^1\sigma_x^2\sigma_x^3\sigma_x^7\sigma_x^8\sigma_x^9$

As described in [5], there is a total of eight operators $M_1, \ldots, M_8$ of this type that we have to measure to be able to detect any possible one qubit phase flip or bit flip error. Each of these operator stabilizes the two logical one-qubit states $|0\rangle_L$ and $|1\rangle_L$ of the Shor code, i.e. these states are eigenstates with eigenvalue one. Also note that the matrices $M_1, \ldots, M_8$ all commute and therefore span an abelian group. This is essential, as the fact that they commute means that we can perform measurements that do not conflict with each other. So we have found that in the case of the Shor code, we can detect errors by applying a set of eight compatible measurements.

Let us now try to formalize and generalize this observation. We consider a system with k qubits and the Pauli group $\mathcal{G} = \mathcal{G}_k$. Within the Pauli group, we consider an abelian subgroup S called the **stabilizer**. We can then look at the linear subspace T_S which is the intersection of all $+1$ eigenspaces of the elements in S, i.e.

$$T_S = \{|\psi\rangle \,\text{such that}\, s|\psi\rangle = |\psi\rangle \forall s \in S\}$$

The idea of stabilizer codes is now to use a subspace of the form T_S as code space for a quantum error correction code. In general, the stabilizer S is described in terms of l independent generators, i.e. $S = \langle g_1, \ldots, g_l \rangle$. One can then show (see A.2) that the dimension of the subspace T_S is equal to 2^{k-l}. Intuitively, each additional stabilizer adds one more constraint, namely being in the $+1$-eigenspace of that stabilizer, that divides the dimension of the code space by two. In the example of the nine qubit code, we are considering the subgroup S generated by the elements $M_1, \ldots, M_8$ which turn out to be independent. The overall Hilbert space has dimension $2^k = 512$ as $k = 9$. Therefore the subspace T_S has dimension $2^{9-8} = 2$ and therefore describes one logical qubit.

Now let us turn to the question which errors a code provided in this way can correct. Assume that we are given a set of error operators $\{E_\alpha\} \subset \mathcal{G}$ as before. For any two indices, we can then consider the operator $E_{\alpha\beta} = E_\alpha^+ E_\beta$. Given any $E_{\alpha\beta}$ and any operator $M \in S$, these two operators either commute or anti-commute, as they are in $\mathcal{G}$.

Let us first suppose that M and $E_{\alpha\beta}$ anti-commute. Then, given any two code words $|\psi_i\rangle$ and $|\psi_j\rangle$, we can calculate

$$\langle\psi_i|E_\alpha^+ E_\beta|\psi_j\rangle = \langle\psi_i|ME_{\alpha\beta}|\psi_j\rangle = -\langle\psi_i|E_{\alpha\beta}M|\psi_j\rangle = \langle\psi_i|E_\alpha^+ E_\beta|\psi_j\rangle$$

where we have used that $M|\psi_i\rangle = M^+|\psi_i\rangle = |\psi_i\rangle$ as M is either plus or minus M^+. This is only possible if

$$\langle\psi_i|E_\alpha^+ E_\beta|\psi_j\rangle = 0$$

This is in line with our earlier condition (1)! Similarly, if $E_{\alpha\beta}$ itself is in S, we obtain that

$$\langle\psi_i|E_\alpha^+ E_\beta|\psi_j\rangle = \langle\psi_i|E_{\alpha\beta}|\psi_j\rangle = \langle\psi_i|\psi_j\rangle\delta_{ij}$$

which again matches condition (1). Also note that due to the fact that E_α is either hermitian or anti-hermitian, we have $E_\alpha^+ E_\beta = \pm E_\alpha E_\beta$. Thus we have shown

THEOREM 4.1 (see [5]). *Suppose that S is an abelian subgroup of the Pauli group $\mathcal{G}$ with a non-trivial invariant subspace T_S. If $\{E_\alpha\}$ is a set of error operators in $\mathcal{G}$ such that for any two indices α, β, the operator $E_\alpha E_\beta$ either anti-commutes with some element of S or is itself in S, then T_S defines an error correction code that is able to correct the errors $\{E_\alpha\}$.*

Let us again consider the nine qubit Shor code as an example, and let $E_1 = \sigma_x^1$ and $E_2 = 1$. Then $E_{12} = \sigma_x^1$ which clearly anticommutes with $M_1 = \sigma_z^1\sigma_z^2$. Thus we can detect and correct single bit flip errors in the first qubit (and similarly in all other qubits). Similarly, if $E_1 = \sigma_z^1$, the operator E_{12} will anti-commute with the operator $M_7 = \sigma_x^1\sigma_x^2\sigma_x^3\sigma_x^4\sigma_x^5\sigma_x^6$. Thus we are also able to correct phase flip errors in the first qubit. A similar reasoning applies to all other one qubit errors, thus we again find that the Shor code can correct them all (see [5]).

To find the possible set of error operators, we therefore need to take a look at the **centralizer** $C(S)$ which is defined to be the set of group elements in $\mathcal{G}$ that commute with every element of S. Now, in our case, the centralizer is in fact equal to the **normalizer** $N(S)$, i.e. the set of all group elements that fix S under conjugation.

To see this, let us first assume that we are given some element A of the Pauli group such that $A \in N(S)$. Then, for any $s \in S$, we have

$$A^+ sA \in S$$

by definition of the normalizer. But being elements of the Pauli group, s and A either commute or anti-commute. We claim that they commute, which would prove that $A \in S$. In fact, if A and s anti-commute, then

$$A^+ sA = -A^+ As = -s$$

so $-s$ would be in S. But $s \in S$, so that this would imply that $-1 \in S$. But of course -1 only acts trivially on the zero vector, so that T_S would be trivial which we have excluded. This proves that $N(s) \subset C(S)$. Conversely, it is clear that $C(S) \subset N(S)$.

In light of this identity, we can now rephrase the condition expressed in theorem 4.1: the code given by the stabilizer S corrects the error operators $\{E_\alpha\}$ if for any two indices Eα, β, we have

$$E_\alpha E_\beta \in S \cup (\mathcal{G} - N(S))$$

For the elements in the normalizer $N(S)$, there is a different story to tell. First, if $n \in N(S)$, the matrix n maps T_S to itself. In fact, if $|\psi\rangle \in T_S$, we have

$$n|\psi\rangle = ns|\psi\rangle = sn|\psi\rangle$$

for every $s \in S$. This shows that $n|\psi\rangle$ is fixed by all $s \in S$ and therefore in T_S. Thus the elements of $N(S)/S$ define an automorphism of T_S. We can interpret these matrices as quantum gates that now act on the *logical qubits* in the code space.

Let us consider an example to see how this works in general. We consider the following four generators, acting on a quantum register with five qubits (and therefore providing a two-dimensional invariant subspace T_S, i.e. one logical qubit).

$$
\begin{aligned}
S_1 &= \sigma_x^1\sigma_z^1\sigma_z^2\sigma_z^4\sigma_x^5\sigma_z^5 \\
S_2 &= \sigma_x^2\sigma_z^3\sigma_z^4\sigma_x^5 \\
S_3 &= \sigma_z^1\sigma_z^2\sigma_x^3\sigma_x^5 \\
S_4 &= \sigma_z^1\sigma_z^3\sigma_x^4\sigma_z^4\sigma_x^5\sigma_z^5
\end{aligned}
$$

It is a bit tedious, but not difficult, to verify that these generators commute. For instance, if we move S_2 past S_1, we pick up one minus one from the second qubit and one from the last qubit. When we commute S_3 past S_3, we pick up a sign from the second qubit and a sign from the third qubit which is again an even number and so forth.

Let us now in addition introduce two additional operators that we call $\bar{X}$ and $\bar{Z}$ which are defined as follows.

$$
\begin{aligned}
\bar{X} &= \sigma_z^1\sigma_z^4\sigma_x^5 \\
\bar{Z} &= \sigma_z^1\sigma_z^2\sigma_z^3\sigma_z^4\sigma_z^5
\end{aligned}
$$

Again, a short calculation shows that each of these operators commutes with all the S_i. However, $\bar{X}$ and $\bar{Z}$ do not commute, but anti-commute! Thus the algebra of these two matrices is exactly as that of σ_x and σ_z. In other words, we obtain an action of the Pauli group $\mathcal{G}_1$ on the space T_S, i.e. on the logical qubit described by this code.

Looking at this code, it appears to be an extremely clever combination of Pauli matrices which is nearly impossible to find. However, the real beauty of the stabilizer formalism is that it reduces the problem of finding a code like this one - and the operators $\bar{X}$ and $\bar{Z}$ - to linear algebra over the field $\mathbb{Z}_2$. This is accomplished using the so called *check matrix representation* or *binary representation* of the stabilizer group. We refer to the appendix as well as to the original work [5] and the treatment in [1] for more details.

5 Preparing states and correcting errors

In the previous section, we have developed the stabilizer formalism to describe quantum error correction codes. Let us now take a more detailed look at how we can actually prepare the logical states and correct errors in that formalism.

Let us first explain how an error correction proceeds when we are using a stabilizer code. Suppose as usual that our stabilizer S is spanned by the independent elements $S_1, \ldots, S_k$, and suppose that $E \in \mathcal{G}_n$ is an error operator.

The first thing to note that if $|c\rangle \in T_s$ is a code word, and S_i commutes with E, then

$$S_iE|c\rangle = ES_i|c\rangle = E|c\rangle$$

Similarly, if S_i anti-commutes with E, then

$$S_iE|c\rangle = -ES_i|c\rangle = -E|c\rangle$$

Thus the state $E|c\rangle$ is, in any case, in an eigenspace of S_i. Consequently, we can measure all the S_i without changing the state! For each index i, the measurement either gives us $+1$ - this will be the case if and only if E and S_i

commute - or -1 - this will be the case if E and S_i anti-commute. Combining the results of the measurements, we obtain a vector e called the **error syndrome**.

As the syndrome only depends on the commutation relation between E and the S_i (and not on the code word), the syndrome gives us a mapping from the set of correctable errors to the set of all possible error syndromes. If this mapping were one-to-one, the error syndrome would tell us exactly which error occured, and we could apply the inverse of the error operator to correct the state.

However, error correction also works if this mapping is not one-to-one. To see this, suppose we are given two error operators E_1 and E_2 that have the same error syndrome. Thus E_1 commutes with S_i if and only if E_2 does. But this implies that the operator

$$E = E_1^+ E_2$$

commutes with all the S_i. In fact, in the expression

$$ES_iE^+ = E_1^+ E_2 S_i E_2^+ E_1^+$$

moving S_i to the right gives either two times minus one or no sign at all, thus the result will always be S_i. Thus we find that E is in the normalizer $N(S)$. However, by the error correction condition in Theorem 4.1, a product of two errors is either in S or in the complement of $N(S)$. Therefore we find that $E = E_1^+ E_2 \in S$. Thus if one of the two errors occurs for a code word $|c\rangle$, say E_1, and we apply E_2^+ to correct it, the resulting state will be

$$E_2^+ E_1 |c\rangle = E|c\rangle = |c\rangle$$

as if we had applied E_1^+. The short summary is that even though the error syndrome does not tell us whether the error is E_1 or E_2, it does not matter - applying *any* of the inverses will still correct the error.

So once we have an encoded state, we can measure the error syndrome, i.e. measure the S_i (which are Pauli operators and can therefore be measured easily) to obtain all the information that we need to correct the error. But how do we actually prepare an encoded state in the first place?

Again, this turns out to be comparatively easy. We start with some initial state, say the state $|0 \dots 0\rangle$ of the computational basis. Then we measure each of the operators S_i in turn. By the rules of quantum mechanics, the measurement will project the state into an eigenspace of S_i.

If all the eigenvalues are $+1$, we are done - our resulting state will then be in the code space T_S. If one of the eigenvalues is -1, we need to apply a correction. For that purpose, we treat the state as if an error with the respective error syndrome had occurred. Thus, we apply an error operator that anti-commutes with those S_i for which the eigenvalue is minus one and commutes with all the others. This will force the state into a simultaneous $+1$-eigenstate of all the S_i and therefore into the state T_S (that such an error operator exists can again be shown using the check matrix formalism and linear algebra, see Lemma 1 in [3]).

6 Fault-tolerant quantum computing

Let us briefly recall what we have achieved so far. We have considered an error model where errors occur during a phase which can be though of as a

"transmission" or simply a certain period of time during which a quantum state is stored in a quantum register. We have seen codes that are able to protect against certain types of errors, most notable against single qubit errors. However, our treatment still has a few serious limitations.

First, we have not been able to protect against all errors. Instead, we have only been able to detect and correct single qubit errors. This can be generalized further, but in general we will only be able to protect against errors that affect only a small number of qubits. When we assume that errors on individual qubits occur independently, this is not yet a full protection, but as errors on more than one qubit are less likely in such a model we have been able to reduce the probability of an error.

Further, we have restricted ourselves to errors that occur while we *store* a state. Obviously, when we *act* on a state, i.e. when we apply quantum gates, errors can occurs as well. Even worse, these errors can propagate. Assume for instance that we have a bit flip in the control qubit of a CNOT gate. If the bit flips from $|0\rangle$ to $|1\rangle$, also the target qubit will flip, thus the error has been duplicated.

And finally, the additional circuitry that we need to detect and correct errors is itself prone to errors. Thus we could detect errors which are not there, oversee other errors or perform wrong corrections - another source of trouble that we need to control.

Fault-tolerant quantum computing refers to a set of techniques and theoretical results that address these shortcomings. Following [17], the required capabilities can be described as follows.

1. We need to be able to prepare our initial state, i.e. in most cases the logical state $|0\rangle_L$, in a fault-tolerant manner

2. We need to find a way to apply quantum gates to encoded states. The obvious approach - decoding, applying a quantum gate directly and encoding again - could lead to uncontrollable error propagation, as an error in the quantum gate affects all qubits after encoding. Therefore, one typically applies **logical gates** to the encoded states, and we need to be able to do this in a fault-tolerant way

3. After applying a logical gates, we need to be able to detect and correct errors, and again this needs to be done in a fault-tolerant way, as the circuits used for error correction might themselves create errors

4. Finally, our measurements must not introduce an uncontrollable number of new errors and must reach a certain a certain precision

Fault-tolerant quantum computation is a highly non-trivial subject, and we can only provide a few examples to illustrate the types of techniques and arguments used. We refer the reader to the many excellent papers on the subject (including original work like [2] or [5] and reviews like [17], [13] and the respective sections in [16] and [1]).

As a first example, let us see how we might detect errors in a fault-tolerant way. We again use the three-bit code as an example to keep things simple, even though this code does not protect against phase flip errors.

Let us take a look at the circuit displayed in figure 2 again, that we have used before to measure the error syndrome for the three-bit code. This circuit works,

but it is not fault-tolerant. The reason is that the CNOT gate can propagate errors in an uncontrolled way.

To see this, suppose in general that we are given a unitary transformation U and an error E. If the error occurs before applying the gate, the resulting state will be

$$UE|\psi\rangle >= UEU^{+}U|\psi\rangle >= (UEU^{+})U|\psi\rangle$$

In other words, the result of the operation is the same as if we had applied U to the correct state and then applied the error UEU^{+} after passing the gate. Now let us assume that U is a CNOT operation and we apply a phase error to the target bit. Then the resulting error operator will be

$$UZ_2U^{+} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = Z_1Z_2$$

In other words, a phase flip error in the target qubit propagates through the CNOT gate as a phase flip error in *both* involved qubits, target and control. This is a typical example of unwanted error propagation - an error in one qubit spreads across other qubits and potentially the entire circuit.

Returning to our syndrome measurement circuit, we see that an error in the ancillary qubits can propagate into the code qubits. Even worse, an error in one of the ancillary qubits can propagate into *two* of the code qubits, destroying the encoded state beyond repair.

A similar problem occurs for the Shor nine qubit code, and in [2], Shor proposed a solution for the problem (which we replicate here for the three qubit code to keep the diagrams simple, even though the basic idea is the same). Shor replaces each ancillary qubit with two ancillary qubits, each of which is connected to one qubit of the code word by a CNOT. Each pair of ancillas is then prepared in the state

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

i.e. a superposition of all even two qubit states. This state is kept stable by two or more generally an even number of inversions. However, if we apply an inversion to only one of the qubits, we get a superposition of states with odd parity. Thus we can use the circuit in 4 to determine the error syndrome as a measurement of the ancillary qubits will tell us whether a bit flip error has occurred.

Now this circuit is more tolerant, since an error in one of the ancillary qubits will affect only one of the code word qubits. Thus if that particular case of error can be corrected if only one qubit is affected, we are safe (again note that this is not the case for our particular example, but for Shor's code).

There is of course more that can go wrong. First, the state of the ancillary qubits itself can be faulty. To protect against this, we need a mechanism to validate the ancillary qubits once they have been prepared and repeat the preparation if needed. Second, the measurement of the ancillas can give a wrong result, but we can increase the precision of the measurement by repeating it several times, as discussed in [2]. Overall, these ideas will give us a sufficiently fault tolerant way to measure the error syndrome. Of course the error correction step

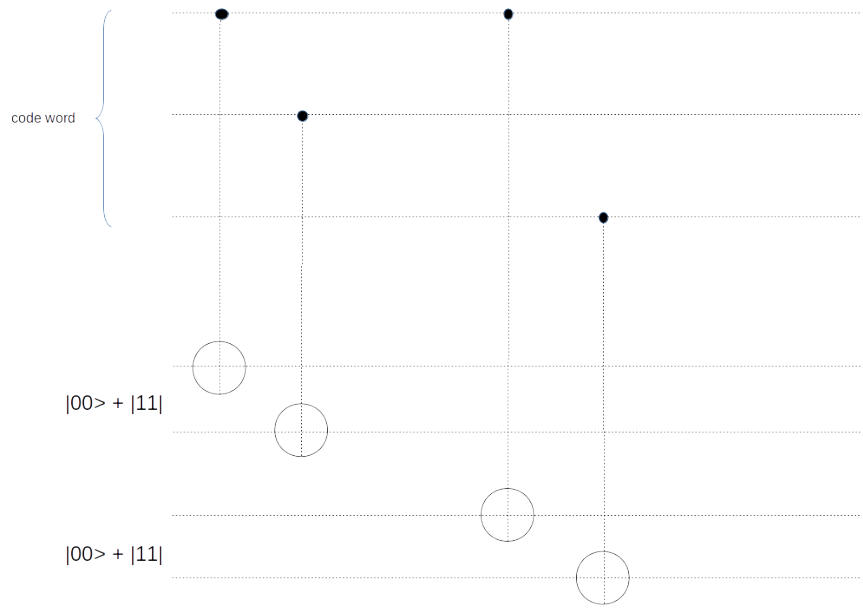

Figure 4: Fault tolerant syndrome measurement

itself can still go wrong, but the entire procedure will only fail if two independent errors occur (one initial error and one error during the correction) and we will be able to fully recover if only one error occurs.

Having discussed error correction, let us now take a brief look at how we can compute with encoded states, i.e. how we can apply logical gates to the code words. Again, the main challenge will be to do this in a fault-tolerant way, and we will discuss only one example to illustrate the basic idea.

Let us assume we wanted to implement a CNOT gate that operates on two logical qubits using the three-qubit code, i.e. on six physical qubits. A straightforward implementation of such a gate could be as in figure 5.

Before discussing error propagation for this example, let us see why the circuit works. If the control state is

$$|1\rangle_L = |111\rangle$$

then each of the CNOT gates will flip a qubit in the target word, and the target state will be flipped logically. If the control state is $|0\rangle_L$, no bit flip occurs and the target state is unchanged.

Unfortunately, this circuit is again not fault tolerant - a bit flip error in the first qubit of the control word will propagate into all three qubits of the target word and create an unrecoverable error. The reason for this behaviour is that one qubit in the first code block is connected to more than one - in this case three - qubits in the second code word.

For a better approach, let us take a look at the circuit shown in figure 6. This circuit also realizes a CNOT on the level of logical states. However, for

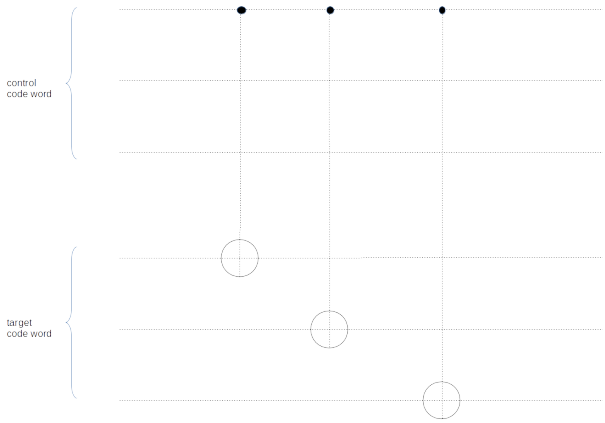

Figure 5: Not fault tolerant encoded CNOT

this circuit, an error in one of the qubits of the first block can only propagate into one qubit in the second block. So if we are able to fully recover from one qubit errors, this circuit works - again as long as only one error occurs. This connectivity property of a circuit is sometimes called **transversality**.

Transversal gates are fault-tolerant, as an error in one qubit can only propagate into one qubit of any other code block, a situation from which we can recover. For many codes, many basic gates can be implemented qubit-wise and therefore transversally on the level of logical qubits. Unfortunately, a full set of universal quantum gates cannot be constructed in such a way - typically one gate remains for which no transversal implementation can be found. Fortunately, there are other (considerable more complicated) methods available to construct fault-tolerant logical gates. In [2], Shor has given the first fully universal set of fault-tolerant logical gates for his nine qubit code.

7 The threshold theorem

We have now discussed fault-tolerant error correction and fault-tolerant computation on encoded states. Let us now see how this fits into the overall picture of fault-tolerant quantum computation.

Suppose we are given a circuit C that describes a quantum computation. We can think of the overall computation as being organized in individual time steps. At each time step, every qubit in the circuit takes part in a basic set of possible **operations** (or, to mention a different term that is sometimes used, **locations**). This can be a gate, the preparation of a qubit in a initial state, a measurement or a "wait step", i.e. the identity operation leaving the involved qubits unchanged. At each time step, a qubit is involved in exactly one of these operations.

Given such a circuit, we can now try to build a fault tolerant circuit performing the same computation as follows. We replace each qubit in the circuit C by a block of qubits encoding a logical qubit. Similarly, we replace each operation

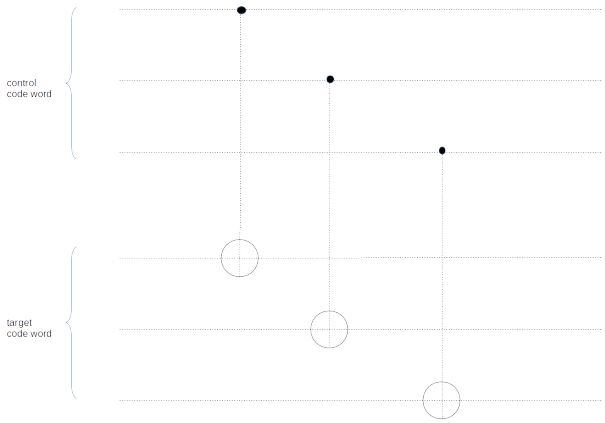

Figure 6: Fault tolerant encoded CNOT

in C by the corresponding fault tolerant equivalent, i.e. we replace each gate by the corresponding logical gate, a measurement by a fault tolerant measurement, and a preparation by a fault tolerant state preparation. In addition, we place a (fault tolerant) error correction circuit in each block which is the output of a logical gate. The result is called a **simulation** of the original circuit. An example is displayed in figure 7, using (roughly) the graphical language described in [3].

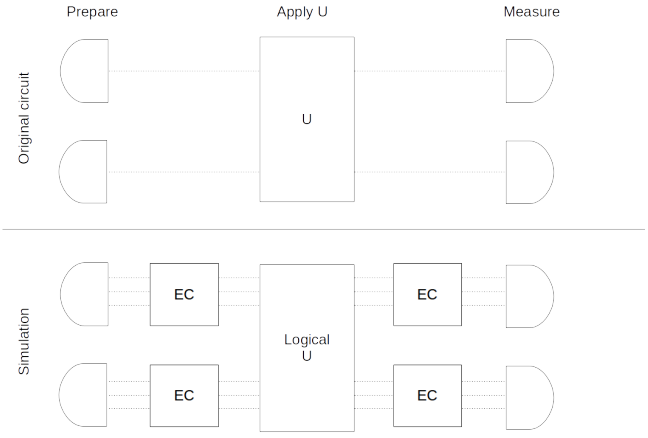

Figure 7: Simulation

The upper part of this diagram shows an original circuit, in which we prepare two qubits in a known state, then apply the unitary transformation U and finally measure both qubits. The result of the replacement procedure described above is displayed in the lower part of the picture, using again the (unrealistic) example of a three qubit code. Here we have replaced each qubit by a corresponding block

of three qubits holding the code word. The operation U has been replaced by its logical implementation, acting on the encoded states, and both after the preparation and after applying U, we add a circuit performing error correction.

We have already seen that the circuits used for error correction, state preparation, logical gates and measurement need to be designed with care and in a fault-tolerant way, which essentially means that errors do not propagate in an uncontrolled manner. But even if we do this, it is not clear that the error rate in the simulation is really smaller. The reason is that even though we have added error correction, we have also added many additional operations acting the physical qubits, i.e. more things that can go wrong. Thus passing to the simulation will introduce additional errors, and it is not a priori clear - and in fact not true - that the error correction will be able to compensate for this so that the net effect is actually an increased reliability.

However, it turns out that you win the race between additional errors that error correction circuitry introduce and the effect of the correction itself provided that the fidelity of the underlying physical operation is below a certain threshold. This is the content of the famous **threshold theorem** - you can correct errors more errors that you create ([18], [19], [20]).

The proof of this theorem is long and complicated, so we restrict ourselves to an oversimplified example. Let us again look at the fault-tolerant implementation of a CNOT gate shown in figure 6. To analyze the probability that this circuit fails, let us assume that the only error that can occur is a bit flip error in the target qubit of each physical CNOT gate and that this error occurs with probability p. Thus we assume that error correction, state preparation and measurement work in an idealized, error free way.

Let us now look at the different combinations of faults that can occur during a specific run of the simulation circuit, i.e. a different **fault paths**. A first fault path is given a single bit flip error in one of the target qubits after executing the CNOT operation. However, this affects only one qubit and can therefore be corrected by the error correction circuit. Thus, for this fault path, the simulation works correctly.

There is, however, a different fault path, given by a fault in target qubit one and target qubit two, whereas the CNOT acting on target qubit three operates correctly. Assuming that the individual faults are statistically independent, the probability for this fault path is

$$p \cdot p \cdot (1-p) = p^2 - p^3$$

This would introduce an error from which we could not recover, as our code is only able to correct errors in one qubit. But this is not yet the overall probability of failure for our new circuit, as there is a second fault path that we need to consider - we could as well have a failure in the first and the third instead of the first and second qubit. Obviously, the probability for this fault path is the same. And there is a third fault path with two errors, given by an error in the second and third target qubit. These three different fault paths, all with two errors, correspond to the

$$\binom{3}{2} = 3$$

possibilities to choose two error locations out of the three target qubits.

And finally, it might happen that faults occur in all three locations. Thus the overall probability to have at least two faults is

$$3(p^2 - p^3) + p^3 = 3p^2 - 2p^3 \leq 3p^2$$

Thus the overall probability that the circuit does not operate correctly does not reduce from p - for the physical CNOT operation without error protection - to p^2, but to $3p^2$, where the factor 3 reflects the growth of the circuit when passing from the physical circuit to the simulation, i.e. the impact of the additional gates.

Of course this is a very naive model, but it turns out that a similar result can be obtained with more realistic error models - this is the work done in [18] and [19] and more recently in [20] using a different formalism that is better suited to cover the case of a code that can correct only one error. The upshot of the discussion is that when passing from a physical circuit with error rate p per operation to the simulation and applying the principles of fault-tolerant quantum computation explained in the previous section, the effective error rate changes according to the rule

$$p \mapsto cp^{t+1}$$

where t is the number of errors that the used code can correct and c is a constant that reflects the growth of the circuit when passing to the simulation and depends on the code and the used fault-tolerant implementations of gates, measurements and state preparations.

Of course this is not necessarily better than p, unless $cp^t < 1$, i.e. unless the error rate p of the physical operations is below a critical threshold given by

$$p < p_{crit} = c^{-\frac{1}{t}}$$

Now the point is that if we are below this threshold, we can improve further by repeating this procedure. Thus we start with a circuit C_0, pass to the simulation C_1 which is again made up of physical gates and qubits, and then replace each of these qubits and gates by their encoded equivalents to obtain another circuit C_2 which simulates C_1 and so forth. On the level of codes, this means that in each step, we encode each of the qubits that make up the code word further using the same code - a procedure known as **concatenation**.

Let us see how the probability for a fault develops when we do this. For simplicity, let us consider the case $t = 1$. We have already seen that passing to the simulation amounts to replacing p by

$$cp^2 = p_{crit}^{-1} pp = \epsilon p$$

where $\epsilon = pp_{crit}{}^{-1} = cp < 1$. If we now iterate this procedure, it is easy to show by induction that after the k-th step, the overall fault probability will be

$$\epsilon^{2^{k+1}-1} p$$

As $\epsilon < 1$, this quickly goes to zero, and we can therefore reach any desired precision, whereas if $\epsilon > 1$, the probability blows up exponentially when we iterate through this procedure. Thus we have shown that *given physical operations with error rate less than p_{crit}, we can implement each quantum circuit in a fault-tolerant way with arbitrary reliability.*

Unfortunately, the exact value of the threshold is not known. The initial estimate provided in [18] was in the range of $p_{crit} = 10^{-6}$. Later, this was improved by several authors, and is mostly assumed to be in the range between 10^{-4} and 10^{-2} (see [13] for an overview of some results in this direction and [6] for the error threshold for surface codes).

As c is in the order of p_{crit}^{-1}, this implies that we need as many as a few hundred to a few thousand physical gates and qubits to simulate one logical qubit. Note that in general, the overhead depends on the desired accuracy and the level of simulations needed as well as on the factors entering the constant c, i.e. the encoding, the architecture of the logical gates, as well as on the quotient ϵ that measures how close the actual physical error rate is compared to the threshold. Improving the threshold and thereby reducing the overhead remains one of the most active areas of current research in quantum computing and is essential for paving the road towards a large-scale fully fault tolerant universal quantum computer.

8 Toric codes

So far we have studied stabilizer codes on abstract Hilbert spaces, assuming no additional relations between the involved qubits. However, in reality, qubits are typically organized on a surface or in a three-dimensional structure. In order to be able to apply a stabilizer code, qubits need to interact, and this is in general not possible between any two qubits in a real world quantum computer, but only between qubits which are direct neighbors. Roughl¡ speaking, **topological codes** are quantum error correction codes that take this additional structure into account, and **toric codes**, which we describe in this section, have been the first example of topological codes.

To describe toric codes, let us first assume that we have organized our qubits on a two-dimensional lattice L. More precisely, we assume that each edge of a regular two-dimensional lattice carries exactly one qubit. We identify opposite edges of the lattice so that alternatively, we could think of the lattice as a cell decomposition of the 2-dimensional torus $T^2 = S^1 \times S^1$.

Recall that the group $C_1(L; \mathbb{Z}_2)$ of 1-chains for that cell decomposition is the set of all formal linear combinations

$$c = \sum_{\text{edges } e} c(e)e$$

of edges, where we choose the coefficients (e) to be in $\mathbb{Z}_2$. If E denotes the number of edges, then we can clearly identify this group with the group $\mathbb{Z}_2^E$, i.e. with the set of binary vectors of length E. But the edges are also in one-to-one correspondence to the qubits, so that binary strings of length E are also in one-to-one correspondence with elements of the computational basis. Thus we have

$$\mathbb{Z}_2^E \simeq C_1(L; \mathbb{Z}_2) \simeq \text{computational basis}$$

and we will switch back and forth between these views as needed. In particular, given a binary vector c of length E, we can consider c as an element of the group of one-chains $C_1(L; \mathbb{Z}_2)$ and as a member $|c\rangle$ of the computational basis.

Similary, given a binary vector c of length E with coefficients c_i, where i ranges over the qubits / edges, we can form operators

$$X_c = \prod_i (\sigma_x^i)^{c_i}$$

and

$$Z_c = \prod_i (\sigma_z^i)^{c_i}$$

where again σ_x^i is the Pauli matrix σ_x acting on qubit i and similarly for σ_z^i. As the Pauli matrices square to one, this gives us two group homomorphisms

$$X, Z \colon C_1(L; \mathbb{Z}_2) \to \mathcal{G}_E$$

into the Pauli group of E qubits. Let us study these homomorphism in a bit more detail.

LEMMA 8.1. *The homomorphisms X and Z from the group $C_1(L; \mathbb{Z}_2)$ into the Pauli group have the following properties.*

1. *For a chain c and a binary vector b, we have $X_c|b\rangle = |b \oplus c\rangle$ where $\oplus$ denotes bitwise addition*
2. *The homomorphism X is one-to-one*
3. *X and Z are related by $Z_c = WX_cW$ where W is the Hadamard-Walsh operator, i.e. the bitwise Hadamard operator*
4. *Z is one-to-one*
5. *Letting $r \colon \mathcal{G}_E \to \mathbb{Z}_2^{2E}$ denote the homomorphism introduced as part of the check matrix formalism in appendix B, we have $r(X_c) = (c; 0)$ and $r(Z_c) = (0; c)$, where we again identify 1-cycles and binary vectors of length E.*

Proof. Given a chain c and a a binary vector b, X_c acts on the computational basis element $|b\rangle$ by flipping those bits for which c_i is not zero, i.e. the qubits at those edges contained in c. In terms of the computational basis, this corresponds to bitwise addition of c, proving our first claim. This implies immediately that $X_c = 1$ can only happen if $|b \oplus c\rangle = c$ for all b, which is only possible if $c = 0$, so X is one-to-one. The third property follows immediately from the fundamental relation $H\sigma_x H = \sigma_z$, and in turn implies that Z is one-to-one. Finally, the statement about the r-homomorphism is clear from the definitions. □

Our next goal is to define an abelian group S that will act as the stabilizer of our code. First, suppose that f is a face, so that $\partial f \in C_1(L; \mathbb{Z}_2)$. We can then apply the Z-homomorphism to ∂f to obtain the operator

$$Z_f = Z_{\partial f} = \prod_{e \in \partial f} Z_e$$

where the second equality is obtained by simply unraveling the definition of Z. Geometrically, the operator Z_f will apply the Pauli Z-operator σ_z to all qubits that are located on an edge in the boundary of the face f. In figure 8, we have

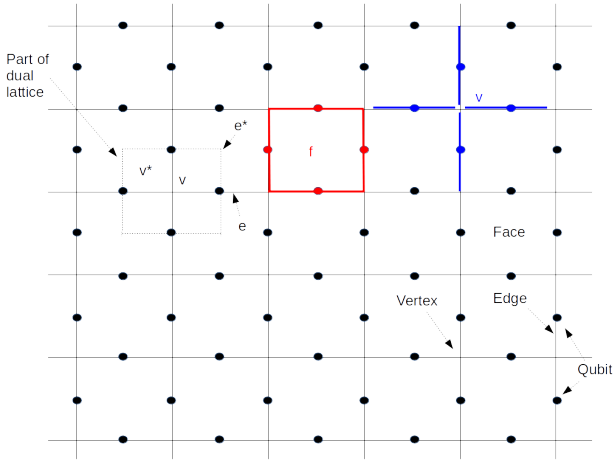

Figure 8: Toric code

marked a face in red, and the operator Z_f will flip the phases of the qubits on its boundary (marked in red as well) and leave all other qubits alone.

A similar construction can be done for vertices. Suppose that v is a vertex. Then there are exactly four edges e that touch v, in the sense that $v \in \partial e$. For each of these edges, we have an operator X_e and we can multiply them to obtain an operator

$$X_v = \prod_{v \in \partial e} X_e$$

In figure 8, the operator X_v for the blue vertex v will apply a Pauli X-operator σ_x to qubits attached to vertices touching v, which are marked in blue as well.

DEFINITION 8.1. *The stabilizer group S is defined to be the subgroup of the Pauli group generated by the elements Z_f and X_v, where f runs over all faces and v runs over all vertices.*

Before studying the group S further, it is useful to expand on the apparent symmetry between Z_f and X_v. To do this, let us consider the dual lattice L^*. This lattice is defined as follows. For each face of L, we introduce a vertex, sitting in the center of the face. For each edge e of L, we introduce another edge which is perpendicular to e and joins the centers of the two faces that meet in e. Finally, for each vertex, we introduce a face, centered at this vertex. In figure 8, a part of the dual cell complex is indicated through dotted lines on the left hand side. It is obvious that the cell complex of L and that of L^* are related by

$$C_2^*(L^*; \mathbb{Z}_2) \sim C_0(L; \mathbb{Z}_2)$$
$$C_0^*(L^*; \mathbb{Z}_2) \sim C_2(L; \mathbb{Z}_2)$$
$$C_1^*(L^*; \mathbb{Z}_2) \sim C_1(L; \mathbb{Z}_2)$$

We can now define a co-boundary ∂^* on C_0 as the composition

$$C_0(L; \mathbb{Z}_2) \to C_2^*(L^*; \mathbb{Z}_2) \to C_1(L^*; \mathbb{Z}_2) \to C_1(L; \mathbb{Z}_2)$$

This composition takes a vertex v to the corresponding face of the dual lattice, then applies the boundary operator to obtain a one-cycle in the dual lattice and

finally switches back to the original lattice. From figure 8, it is obvious that the co-boundary $\partial^* v$ for a vertex will be exactly the linear combination of all edges that touch v. With this, we can write our X-operator as

$$X_v = X_{\partial^* v}$$

As we have identified the chains of the lattice L with binary vectors, the same identification can be done for the chains of the lattice L^*, and similarly with elements of the computational basis, and we will do this in the sequel.

LEMMA 8.2. *The group S as defined above has the following properties.*

1. *S is abelian*
2. *All elements in S square to one*
3. *$-1 \notin S$*
4. *S is generated by $V+F-2$ independent generators, where V is the number of vertices and F the number of faces*
5. *The code space T_S, i.e. the subspace on which all elements of S act trivially, has dimension 2^{b_1}, where b_1 is the first Betti number of the lattice L*
6. *The first Betti number of the lattice is 2 and therefore the dimension of T_S is four*

Proof. First, let us prove that S is abelian. For this, it is obviously sufficient to show that all the Z_f and X_v commute. The only non-trivial part is that X_v and Z_f commute for all choices of v and f. Now from the definition of Z_f and X_v, it is clear that the supports of these two operators only overlap if v is touching f. But in this case, there are actually two qubits on which X_v and Z_f both operate (those qubits on the edges touching both v and f). Each of those two qubits gives a minus one if we commute the operators, so the result will be plus one and the operators commute. The second statement immediately follows from the fact that the group is abelian and all generators square to one.

Next let us show that $-1 \notin s$. For that purpose, assume that $-1 \in S$. This would give a relation

$$\prod_f Z_{\partial f} \prod_v X_{\partial^* v} = -1$$

for some vertices v and some faces f. Now let both sides act on the state $|0 \dots 0\rangle$. As each Z acts trivially on this state, we obtain

$$\prod_v X_{\partial^* v} \prod_f Z_{\partial f} |0 \dots 0\rangle = \prod_v X_{\partial^* v} |0 \dots 0\rangle = |\partial^* \sum_v v\rangle = -|0\rangle$$

This would not only imply $\partial^* \sum_v v = 0$ (which can happen and will actually happen if the set of vertices is equal to the set of all vertices) but also $|0\rangle = -|0\rangle$ (which can never happen) and is therefore a contradiction.

To find the number of independent generators, we can now apply the check matrix formalism described in appendix B. This matrix has $2E$ columns and $V + F$ rows. The first V rows are given by

$$r(X_v) = r(X_{\partial^* v}) = (\partial^* f; 0)$$

where v runs over all vertices. Similarly, the last F rows are given by the vectors $(0; \partial f)$. In a slight abuse of notation, the check matrix is therefore

$$\mathcal{C} = \begin{pmatrix} \partial^* & 0 \\ 0 & \partial \end{pmatrix}$$

Thus the rank of this matrix, i.e. the dimension of its image, is given as the sum

$$\dim \text{range}(\partial) + \dim \text{range}(\partial^*)$$

But the dimension of the range of ∂ is - by elementary linear algebra - the dimension of the domain, i.e. F, minus the dimension of the kernel. The dimension of the kernel is the dimension of the set of 2-cycles, which, by algebraic topology, is the dimension of the second homology group $H_2(L; \mathbb{Z}_2)$, i.e. the second Betti number $b_2(L)$. Similarly, by duality, the dimension of the kernel of ∂^* is the zero-th betti number b_0. Thus the rank is

$$F - b_2 + V - b_0 = (F + V - E) + E - (b_0 + b_2 - b_1) - b_1 = E - b_1$$

where we have used that the Euler number of the lattice is given by

$$\chi = F + V - E = b_0 + b_2 - b_1$$

Now our lattice is topologically a torus, and therefore the first Betti number if $b_1 = 2$. Thus the rank is $E - 2$. If we compare this to $F + V$, using that the Euler characteristic of the lattice is that of the torus and hence zero, we find that the rank is $V + F - 2$. By the check matrix formalism, we can therefore conclude that there are $F + V - 2$ independent generators as claimed. Note that the two missing dimensions correspond to those chains which are in the kernel of ∂ and ∂^*, i.e. to the relations that the product of all the X_v and the product of all the Z_f are both the identity. Consequently the dimension of the code space is given (see Lemma A.2 in the appendix) by

$$2^{E-(E-b_1)} = 2^{b_1} = 4$$

as the Betti number of the torus is two. □

Let us now study the stabilizer $N(S)$ and try to find a geometric interpretation for it. We have seen that we can, to any one-cycle c, associate operators X_c and Z_c. To further emphasize the duality between X and Z, let us consider the set of operators obtained as

$$Z_c\,,\; X_{c^*} \quad \text{for} \quad c \in C_1(L; \mathbb{Z}_2),\; c^* \in C_1(L^* \mathbb{Z}_2)$$

i.e. we associate Z-operators with chains and X-operators with chains in the dual lattice (the reader more familiar with algebraic topology will recognize the chains of the dual lattice as the co-chains, identified using the intersection product). It is clear that, up to a phase, every Pauli generator can be obtained in this way, i.e. these operators plus i generate the entire Pauli group acting on the E qubits.

There is one more property of these operators that we will need - they allow us to represent commutation relations geometrically in terms of the so-called intersection product. Given a chain c in L and a chain c^* in the dual lattice, we

can form the intersection number $c \cdot c^*$, which is a number in $\mathbb{Z}_2$ and counting the number of intersection points modulo two. This number is called the intersection number or intersection product. If i is a qubit on which both Z_c and X_{c^*} act, then this qubit is sitting at an intersection of c and c^*. This implies that Z_c and X_{c^*} commute if and only if the intersection number $c \cdot c^*$ is zero.

What does it mean for those operators to be in the normalizer $N(S)$? By the general theory, we know that the normalizer consists of all those operators that commute with all elements in S, so we have to express commutation relations between the Z_c and X_{c^*} with the elements in S. Clearly, and two X-operators and any two Z-operators commute. The remaining commutation relation turn out to be given as follows.

LEMMA 8.3. *Given a vertex v, a chain c and a face f, we have the following commutation relations.*

$$[Z_c, X_v] = 0 \Leftrightarrow v \notin \partial c$$
$$[Z_f, X_{c^*}] = 0 \Leftrightarrow \partial f \cdot c^* = 0$$

Proof. Let us start with the first relation. By definition, X_v is the operator that acts as a σ_x on all the qubits which are on edges adjacent to the vertex v. When we commute this past Z_c for a chain c, we will pick up a minus one for every component of c which is one of those four edges. Therefore the overall sign can only be minus one if c or a component of c ends at v, i.e. if $v \in \partial c$.

For the second line, note that Z_f contains one Z-operator for each edge in the boundary of f. The two operators will commute if and only if c^* contains an even number of these operators. Every operator that occurs in both Z_f and X_{c^*} corresponds to one intersection of c^* and ∂f. Thus the operators commute if and only if that number if even which is the case if and only if the $\mathbb{Z}_2$ intersection product $\partial f \cdot c^*$ is zero. □

This lemma seems to break the symmetry between the X and Z operators, but is does not - observe that we could rewrite the first condition as $\partial^* v \cdot c = 0$ and therefore as an intersection product as well. In fact, we could also prove the first line similar to the second one by working in the dual lattice.

We are now in a position to give a geometrical description of the normalizer $N(S)$. First, note that if c is a cycle and c^* a dual cycle, then Z_c and X_{c^*} commute with all the Z_f and X_v (note that $\partial f \cdot c^* = f \cdot \partial c^*$ and therefore zero if c^* is a co-cycle). Hence every element of $Z_1(L; \mathbb{Z}_2) \times Z_1(L^*; \mathbb{Z}_2)$ gives us an element in the normalizer, where Z_1 denotes the group of cycles.

It is also clear that if $c = \partial f$ is a boundary, then $Z_c = Z_{\partial f} = Z_f$ is in S. Similarly, if c^* is a co-boundary, then c^* is dual to $\partial^* v$ for some vertex v, and hence $X_{c^*} = X_v \in S$. So under this mapping, elements in the group of boundaries go to S. Therefore only non-trivial elements in the product

$$H_1(L; \mathbb{Z}_2) \times H_1(L^*; \mathbb{Z}_2)$$

will give us non-trivial elements of $N(S)/S$. Geometrically speaking, every closed loop on the lattice will give us a Z-operator in the normalizer and every closed loop in the dual lattice will give us an X-operator. Moreover, homologically equivalent loops describe the same elements in the normalizer.

Let us now identify four operators in $N(S)$ that are not in S and intuitively speaking span $N(S)/S$ up to a phase. In figure 9, we have indicated two cycles

c_1, c_2 and two cycles c_1^*, c_2^* in the dual lattice. Both c_1 and c_2 wind around the torus once and are homologically non-trivial. Their mutual intersection number is one, so they generate the group $H_1(L; \mathbb{Z}_2)$. The same holds for the two dual cycles on the dual lattice. Note that $c_1 \cdot c_1^* = 0 = c_2 \cdot c_2^*$ while $c_1 \cdot c_2^* = 1 = c_1^* \cdot c_2$.

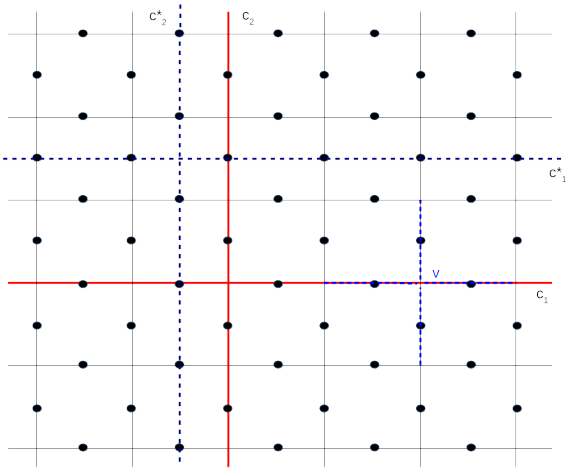

Figure 9: Logical Pauli operators in the toric code

We can now associate operators with these cycles and dual cycles as before. Specifically, let

$$\bar{X}_i = X_{c_i^*}$$
$$\bar{Z}_i = Z_{c_i}$$

for $i = 1, 2$. By what we have seen above, these four operators are all in $N(S)$ (since the are constructed from cycles). Their mutual commutation relations are

$$[\bar{X}_i, \bar{X}_j] = 0$$
$$[\bar{Z}_i, \bar{Z}_j] = 0$$
$$[\bar{X}_i, \bar{Z}_j] = 0 \,\text{if}\, i \neq j$$
$$\{\bar{X}_i, \bar{Z}_i\} = 0$$

Thus, these operators act as **logical Pauli operators** on the code space T_S. As for any stabilizer code, we can use the commutation relations again to build up the entire code space from a ground state $|0\rangle_L$ by repeatedly applying the logical Pauli X operators. This will give us three additional, linearly independent states which, together with the ground state, span the entire code space.

The commutator relations that we have derived can also be used to obtain a prescription how to construct states in the code space T_S explicitly. To see how this works, let us first start with a state $|\zeta\rangle$ that is left invariant by all Z-operators - this could for instance be the state $|0 \ldots 0\rangle$. Now pick a cycle c^* in the dual lattice and consider the state $X_{c^*}|\zeta\rangle$. Let us see how a face operator Z_f acts on this state. As c^* is a cycle, we have $\partial f \cdot c^* = 0$, and therefore

$$Z_f X_{c^*}|\zeta\rangle = X_{c^*} Z_f |\zeta\rangle = X_{c^*}|\zeta\rangle$$

So we have found a state that is invariant under all the Z_f. Of course this state is not invariant under all X_v, but we can now use the usual averaging trick. Consider the state

$$\sum_{b \in B_1^*} X_{c^*+b} |\zeta\rangle$$

where B_1^* is the group of all one-cycles in the dual lattice which are a boundary. As any $c^* + b$ is again a cycle in the dual lattice, each of the states $X_{c^*+b}|\zeta\rangle$ is left invariant by all the Z_f by our previous argument, so the same is true for the sum. But this sum is all invariant under all X_v, as $X_v = X_{\partial^* v}$ and $\partial^* v \in B_1^*$, so letting this operator act will just shuffle the elements of the sum. So we have found a state in the code space.

A similar construction works when we start with a state $|\xi\rangle$ which is left invariant by all X-operators. Then an argument along the same lines shows that for every 1-cycle c, the state

$$\sum_{b \in B_1} Z_{c+b} |\xi\rangle$$

is in the code space.

9 Error correction in toric codes

Having described toric codes as special cases of the more general formalism of stabilizer codes, let us now see how we can detect and correct errors. The stabilizer of our code space is given by all operators X_v and Z_f. Both for error detection as well as for state preparation, we need to be able to measure these operators.

To understand how this can be done, let us first recall a general trick to measure eigenvalues of unitary operators. So suppose that U is an operator that is unitary as well as hermitian, so that its eigenvalues can only be ± 1. Consider the circuit shown in figure 10.

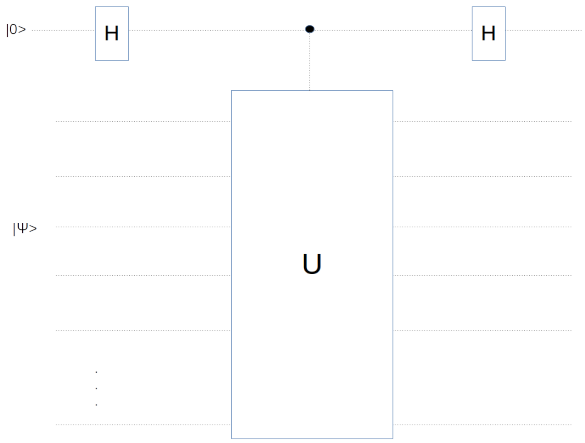

Figure 10: Measurement as a controlled-U operation

Let us see how this circuit acts on a state $|\psi_+\rangle$ which is an eigenstate of U for the eigenvalue $+1$. In the first time step, the state is mapped to

$$H|0\rangle \otimes |\psi_+\rangle = \frac{1}{\sqrt{2}}\big[|0\rangle \otimes |\psi_+\rangle + |1\rangle \otimes |\psi_+\rangle\big]$$

In the second step, we apply the controlled U-operation, and get

$$\frac{1}{\sqrt{2}}\big[|0\rangle \otimes |\psi_+\rangle + |1\rangle \otimes |\psi_+\rangle\big]$$

as the U-operation acts trivially in this case. So the result is again

$$H|0\rangle \otimes |\psi_+\rangle$$

so that the final Hadamard returns the state to its original state. Thus the gate acts trivially on $|\psi_+\rangle$, as expected. If, however, we apply the gate to an eigenstate $|\psi_-\rangle$ for the eigenvalue minus one, the same calculation shows that the result of the controlled U-operation is

$$\frac{1}{\sqrt{2}}\big[|0\rangle \otimes |\psi_-\rangle - |1\rangle \otimes |\psi_-\rangle\big] = H|1\rangle \otimes |\psi_-\rangle$$

so that the final Hadamard results in $|1\rangle \otimes |\psi_-\rangle$. Therefore a general state

$$|\psi\rangle = \alpha|\psi_+\rangle + \beta|\psi_-\rangle$$

is mapped to

$$\alpha|0\rangle \otimes |\psi_+\rangle + \beta|1\rangle \otimes |\psi_-\rangle$$

If we now apply a measurement to the control qubit, which serves as an ancilla here, the state is projected onto either $|1\rangle \otimes |\psi_-\rangle$ or $|0\rangle \otimes |\psi_+\rangle$, with probabilities $|\beta|^2$ and $|\alpha|^2$. The remaining qubits end up in the state $|\psi_-\rangle$ or $|\psi_+\rangle$, i.e. in an eigenstate. Thus applying this circuitry and measuring the ancilla has the same implications as measuring the operator U.

Let us now apply this idea to the problem of measuring one of the operators X_v. This operator is a tensor product of four σ_x operators, i.e. inversion operators. Therefore a controlled-X_v circuit is nothing but a sequence of controlled inversions, i.e. CNOT gates. Thus we can measure a vertex operator using the circuit in diagram 11.

Here the four qubits at the bottom correspond to the four edges meeting the vertex v. So we need one ancilla qubit for doing the measurement, and we need to be able to combine it in gates with all the four qubits sitting on the edges meeting v. For the physical implementation, the easiest approach is to place this additional qubit called **measurement qubit** in the vertex itself, in contrast to the **data qubits** sitting on the edges.

A circuit along the same lines can be constructed to measure Z_f for a face f. Using that $H\sigma_x H = \sigma_z$, we can construct such a circuit as follows. In the first time step, apply a Hadamard to both the ancilla and all target qubits. Then apply CNOT operators to all target qubits, controlled by the ancilla, and then apply Hadamard operators again. Now, it is well known that conjugating the CNOT with Hadamard gates switches the roles of target qubits and control qubits. This gives us the circuit displayed in figure 12.

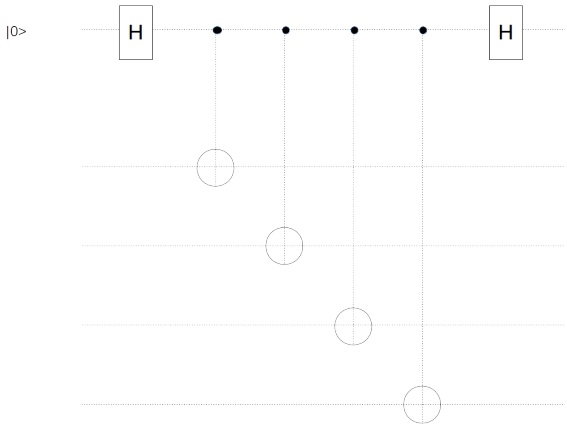

Figure 11: Measuring a vertex operator X_v

Again, for each vertex, we need an ancilla qubit to measure Z_f. This qubit will have to interact with all data qubits on the edges bounding this face, so it is easiest to position it in the center of that face.

Overall, we have now added $F + V$ measurement qubits to our original E data qubits. Now the Euler characteristic of the torus is zero, and therefore $F+V = E$, showing that this procedure actually doubles the number of physical qubits.

We have now arrived at a description of the toric code (and its cousin the surface code) that one can often find as a definition in the literature but which somehow obscures the relation to the surface topology a bit.

1. We start with a lattice L with E edges and identify opposite borders
2. On each vertex, we place a data qubit
3. On each face, we place a measurement qubit called a Z-qubit
4. On each vertex, we place a measurement qubit called an X-qubit
5. To each X-qubit, we associate the operator that acts as a bit flip on the four data qubits next to the measurement qubit
6. To each Z-qubit, we associate the operator that acts a a phase flip on the four data qubits next to the measurement qubit

Thus every data qubit is acted on by four operators, the two Z-operators corresponding to the adjacent faces and the two X-operators corresponding to the adjacent vertices. The additional measurement qubits - indicated by white circles in contrast to the dark circles being the data qubits - are shown in figure 13.

For one face and one vertex, we have indicated the associated operators X_v and Z_f by adding letters X and Z to show how these operators act on the nearby data qubits. This is the graphical representation typically used in the literature (like [6]). Note that some authors use different conventions and

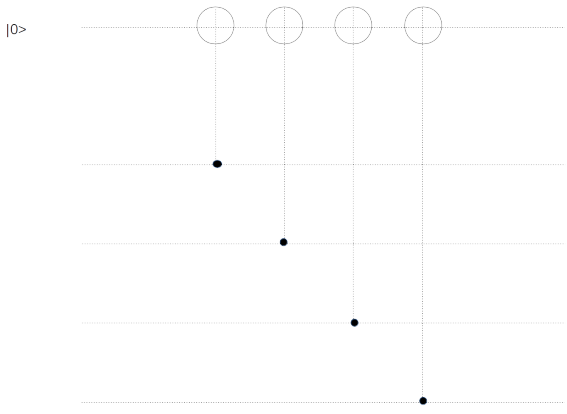

Figure 12: Measuring a face operator Z_f

associate Z-operators with faces and X-operators with vertices, but this is of course completely equivalent due to the duality relations that we have already used several times.

Which errors can we correct using this code? First, each Pauli operator of weight one (i.e. acting on a single qubit) anti-commutes with one of the X_v and Z_f. Take, for instance, the operator σ_z acting on a qubit sitting on the edge e. Then this operator anti-commutes with all the operators X_v for a vertex v adjacent to e. Similarly, an operator σ_x acting on this edge anti-commutes with Z_f if f is a face adjacent to that edge. Thus our code is able to protect against single bit flip and phase flip errors.

To discuss more general errors, recall the operators Z_c and X_{c^*}. It is obvious that up to a phase, every Pauli operator can be expressed as a product of such operators (in fact, for an edge e, Z_e is simply the σ_z operator for the corresponding qubit, and if $e^\sharp$ denotes the dual edge, $X_{e^\sharp}$ is the σ_x operator for this qubit). Thus every Pauli operator $g \in \mathcal{G}_E$ can be written as

$$g = i^a Z_c X_{c^*}$$

for a chain c and a co-chain c^*. The weight of that operator is at least the minimum length of c and c^*. Geometrically, we can think of the cycles as strings (or collections of strings) running on the lattice itself (c) or on the dual lattice (c^*).

By our previous discussion on commutators, the element g will be in S if and only if c and c^* are both boundaries, and it will not be in $N(S)$ if one of the strings is open, i.e. has a boundary. In both cases, the general theory of stabilizer codes tells us that the error can be corrected. So the problematic operators that correspond to products of non-correctable errors are those given by closed strings that are no boundaries, i.e. homologically non-trivial. If l is the minimum length of such a string, then, consequently, the distance of the code is the size of the smallest non-trivial loop in L or L^*. Of course, in our toroidal geometry, l is just the smaller of height and weight of the lattice. Therefore the larger the lattice, the higher the ability of the code to correct errors, but of

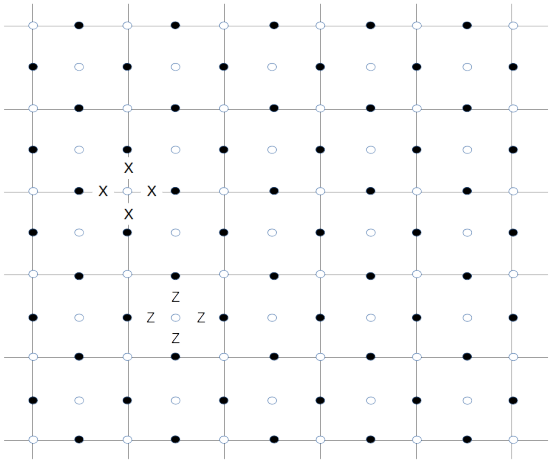

Figure 13: Data and measurement qubits

course the overhead increases as well with the size of the lattice.

There is also a nice geometric interpretation of the error syndrome. Suppose that an error $Z_c X_{c^*}$ occurs. As usual with stabilizer codes, we can then determine the error syndrome by measuring all the generators of the stabilizer. If we measure Z_f, the result will be minus one for exactly those faces f for which $\partial f \cdot c^* = 1$, i.e. for those faces in which the string c^* starts or ends. Similarly, if we measure the eigenvalue of an X_v, then the result will be minus one for those vertices v that are in the boundary of the string c. Thus the syndrome measurement will allow us to reproduce the *boundaries* of the strings c and c^*. To correct, for instance, a Z-error given by a string c, we need to choose a string connecting the two vertices for which the error syndrome is not one. In practice, we might want to choose a path between these two vertices of smallest weight. This choice is not unique, but if the vertices are sufficiently close to another, then a different choice will give a homologically equivalent path, and the difference will be a closed, homologically trivial loop, corresponding to an element of S, so that both choices give the same correction. Only if the length of the error string exceed a certain value, corresponding to half of the minimum length of a non-trivial loop, will this procedure be at risk to give a wrong answer.

Finally let us see how state preparation works on that lattice. By the general theory of stabilizer codes, one approach is as follows. We first bring the system into an arbitrary state and measure all the X_v and Z_f. This gives us an error syndrome, describing the outcomes of these $F + V = E$ measurements. For these positions in the error syndrome that are not one, we then apply a bit flip or phase flip. This will put the system into a state in the code space. We can now apply the logical X-operators to flip logical qubits if needed to arrive at the state $|00\rangle_L$. In fact, as explained in [6], we usually apply several rounds of stabilizer measurements to protect against measurement errors and, if we find an error, do not actually correct it but keep track of the correction factor in software and commute it through the remainder of the circuitry until we arrive at the measurement.

10 The particle interpretation

So far our description of the toric code and the code space has been rather unphysical. In this section, we will provide a more physically motivated description of the code and the error correction process. For that purpose, consider the operator

$$H = -\sum_v X_v - \sum_f Z_f$$

which is simply the sum of all face and vertex operators. Clearly, this operator is hermitian, being a sum of hermitian operators. We can therefore interpret it as the Hamiltonian of a physical system.

Let us try to determine the eigenstates of the Hamiltonian, i.e. the energy levels of our system. As the X_v and the Z_f all commute, we can find a simultaneous basis of eigenstates that will diagonalize the Hamiltonian. Let us call this basis $\{|\psi_i\rangle\}$. Each of these vectors is an eigenstate of all X_v and all Z_f, and we denote the corresponding eigenvalues by x_{vi} and z_{fi}. Clearly, $|\psi_i\rangle$ is then an eigenvector of H with eigenvalue

$$-\sum_f z_{fi} - \sum_v x_{vi}$$

This is minimal if and only if all the x_{vi} and z_{vi} are plus one, which is the case if $|\psi_i\rangle$ is an element of the code space. In this case, the energy is

$$-(F+V) = -(F+V-E) - E = -\chi(S^1 \times S1^{)} - E = -E$$

and we see that the elements of the code space can be identified as the *ground states* of our system. We also see that the ground state is degenerate, as the dimension of the code space is not one but four.

Now let us introduce an error into our system. Thus, we start with a ground state $|g\rangle$ and, say, apply an operator Z_c for some 1-chain c. We have already seen that the error syndrome will be localized on those two vertices which form the boundary of the chain c. The energy of that state is now no longer $-E$. Instead, if we expand the state into a sum of eigenstates and calculate the expectation value of H, we see that two of the coefficients in the expansion have flipped, and this state will have energy $-E+4$. Thus, we have found an excitation which is localized on two locations on our lattice. Is is tempting to call this excitation a **particle** (similar to, for instance, quantum field theory, where the elementary excitations of a field are considered as the particles described by the theory).

In fact, our model contains two types of particles. We have particles created by applying the operator Z_c for a string c, which correspond to excitations localized on two vertices of the lattice - these are called **electric charges** or e-particles in [22]. And there are particles created by an operator X_{c^*} for some string c^* in the dual lattice, which are called **magnetic vortices** or m-particles. These particles live on the faces of the lattice, namely those faces at which c^* starts and ends. Thus we can interpret errors as excitations or particles, located at the points of the lattice and dual lattice with a non-trivial error syndrome.

Figure 14 shows an example of m- and e-particles. Here the solid red line is a chain c, and the endpoints of this line are two e-particles. Similarly, the solid blue line is a chain c^* and its two endpoints are two m particles.

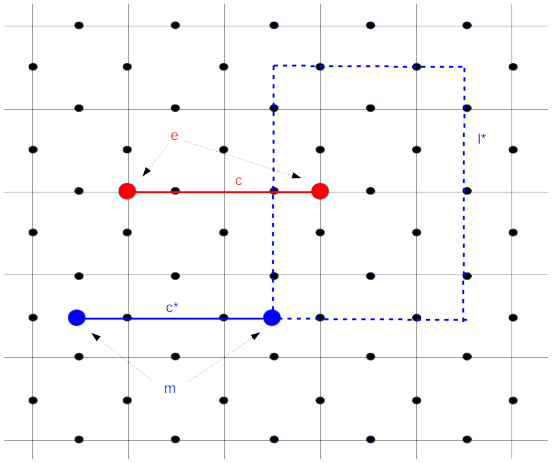

Figure 14: Quasi-particles in the toric code

Note that if we apply an operator $X_{c'}$ to the state representing the two e-particles and c' is a chain continuing c (i.e. one of the boundary points of c' is one of the boundary points of c), then the result of applying $X_{c'}$ will be the same as if we had applied $X_{c+c'}$ to the ground state - an e-particle located at the boundary points of $c + c'$. Thus we can interpret the process of applying $X_{c'}$ as **moving** the e-particle which is on the boundary of c' along the chain c' (realized by a small perturbation of the type $\lambda X_{c'}$ of the Hamiltonian). In particular, applying X_c once more - which will of course give us the ground state as $X_c^2 = 1$ - can be interpreted as moving the two e-particles towards each other along the chain c until they annihilate.

The same process, of course, can be applied to m-particles - they move along chains in the dual lattice. Let us now look at a specific example of such a movement. In figure 14, the path marked with a dashed blue line is a chain l^* in the dual lattice which is closed. We consider the following process.

1. We start with a ground state $|g\rangle$
2. We apply the operators Z_c and X_{c^*} to create two e-particles and two m-particles - let us call this state $|\psi\rangle$
3. We then apply X_{l^*} to move one of the m-particles once around one of the e-particles as indicated in figure 14

Clearly, the state $|\psi\rangle$ after completing the first two steps of this procedure is given by

$$|\psi\rangle = Z_c X_{c^*} |g\rangle$$

Let us now compute the state after the third step, again employing our commutation relations. The closed cycle l^* intersects c once, so X_{l^*} and Z_c anti-commute. Thus

$$X_{l^*}|\psi\rangle = X_{l^*} Z_c X_{c^*} |g\rangle = -Z_c X_{c^*} X_{l^*} |g\rangle$$

But l^* is a cycle, and in fact it is a homologically trivial cycle, i.e. a boundary. Therefore X_{l^*} is in the stabilizer, and maps the ground state to itself. We therefore end up with

$$X_{l^*}|\psi\rangle = -Z_c X_{c^*}|g\rangle = -|\psi\rangle$$

Thus moving one of the m-particles around one of the e-particles introduces a phase factor minus one. This is the reason why Kitaev calls these states *anyons* in [22] (which is not fully precise, as each individual particle is easily seen to be a boson, but their mutual statistics is that of an anyon, see [26] for a detailed introduction into this field).

Exchanging these particles by moving them around each other can be described as an action of the pure braid group, i.e. as an action of the group of braids that return each particle to its original position (this subgroup, the kernel of the natural homomorphism from the braid group to the permutation group, is sometimes called the group of colored braids). If we perturb a braid slightly, the action will remain unchanged, so the unitary operators which are given by that action are by construction rather stable in the presence of noise. In [22], Kitaev developed the idea to use braid group representations to implement quantum gates. Unfortunately, the set of operators that we can obtain in this way is not sufficient to implement a universal quantum computer - we will get back to this point in a later section when we look at the actual procedures used to perform computations in a toric or surface code. In [22], certain hypothetical particles called non-abelian anyons are described which, if they can be physically realized, provide sufficiently rich unitary operations. This approach is called *topological quantum computation*. We will not go into further details but refer to the original work [22].

11 Planar codes

So far we have worked with qubits organized on a lattice with a toroidal geometry, i.e. a cellular decomposition of the torus. In practice, however, it would be much easier to produce such a lattice with a planar geometry, also because that makes it easier to combine several lattices by stacking them. In this section, we explain how the toric code can be modified to obtain a planar version called **surface code**.

Our first attempt could be to start with a lattice L as before, but without performing the identification of the boundaries that introduces the toroidal geometry. We could then still place qubits on the edges and define our operators X_v and Z_f as before. The dimension of the code space would then again be given by $2^{b_1(L)}$. Unfortunately, such a lattice is topologically equivalent to a disk and therefore has vanishing first homology, so that our code space would be one-dimensional and not be able to support a logical qubit. So we need to modify the formalism to obtain a non-trivial first homology group. The idea presented in [27] is to work in the homology group relative to a subset of the boundary.

To illustrate this, let us take a look at the lattice in figure 15. The solid lines define a planar lattice L. As before, we associate qubits with the edges of the lattice. However, there is one difference - we only place qubits on a subset of the edges. Specifically, we split the boundary of the lattice into two pieces.

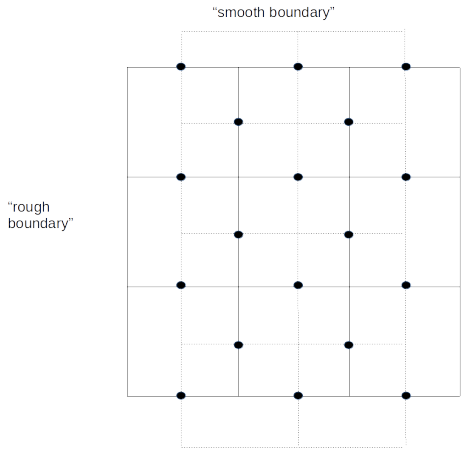

Figure 15: Planar code - all edges shown

One part, called the "smooth boundary" (in our figure, this is the boundary at the top and bottom part of the lattice) carries qubits as usual. The other two boundary pieces on the left and the right of the lattice are called "rough boundary" and do not carry qubits.

Let us denote the rough boundary by V. We can then define a group of one chains $C_1(L)$ and a group of one chains $C_1(V)$, with a natural inclusion $C_1(V) \to C_1(L)$. The quotient

$$C_1(L, V; \mathbb{Z}_2) = C_1(L; \mathbb{Z}_2)/C_1(V; \mathbb{Z}_2)$$

is called the group of *relative one chains*. Similarly, we can consider the group $C0_(L, V; \mathbb{Z}_2)$ - here it is important to note that we consider all vertices to be part of V that are in the boundary of V. Thus, in our example, the four vertices located at the corners of the lattice are all in V. Finally, there is a group $C_2(L, V; \mathbb{Z}_2)$ which is of course equal to $C_2(L; \mathbb{Z}_2)$ as V does not contain any faces.

As $C_1(L; \mathbb{Z}_2)$ is the vector space over $\mathbb{Z}_2$ generated by the edges of L and $C_1(V; \mathbb{Z}_2)$ is the subspace generated by the edges in V, it is clear that the quotient is isomorphic to the subspace spanned by the edges not in V alone, and we will switch forth and back between the description as a quotient and a subspace as needed.

As before, any chain $c \in C_1(L; \mathbb{Z}_2)$ gives us operators X_c and Z_c, acting on the qubits placed on the lattice. Of course, the chains supported in V act trivially, as there are no qubits on the edges which are part of V. Therefore these maps factor to group homorphisms

$$X, Z \colon C_1(L, V; \mathbb{Z}_2) = C_1(L; \mathbb{Z}_2)/C_1(V; \mathbb{Z}_2) \to \mathcal{G}$$

where $\mathcal{G}$ again denotes the Pauli group acting on all the qubits.

We can again form a dual lattice L^*. Specifically, we place an edge of the dual lattice at each qubit, perpendicular to the edge supporting the qubit, and add edges at the top and the bottom to close the lattice, as shown in figure 15. The

boundary of the dual lattice again contains a part which we call rough boundary which does not contain any qubits and which we denote by V^*. Note that the rough boundary of the original lattice gives raise to the smooth boundary of the dual lattice and vice versa. We again have relative chain groups $C_*(L^*, V^*; \mathbb{Z}_2)$, and our conventions make sure that there is still a duality between those chain groups and that of L. In fact, $C_1(L, V; \mathbb{Z}_2)$ is the vector space generated by all edges of L that carry a qubit. These edges are in a one-to-one correspondence to those edges of L^* that carry a qubit, and those in turn span the chain group $C_1(L^*, V^*; \mathbb{Z}_2)$ which gives raise to an isomorphism

$$C_1(L, V; \mathbb{Z}_2) \simeq C_1(L^*, V^*; \mathbb{Z}_2)$$

which geometrically is again described by the intersection product. Similarly, the group $C_2(L^*, V^*; \mathbb{Z}_2)$ is spanned by the faces of the dual lattice which are in one-to-one correspondence to those vertices of the lattice L not in V, so that

$$C_0(L, V; \mathbb{Z}_2) \simeq C_2(L^*, V^*; \mathbb{Z}_2)$$

And finally, the vertices of the dual lattice which are not in V^* are in one-to-one correspondence with the faces of L.

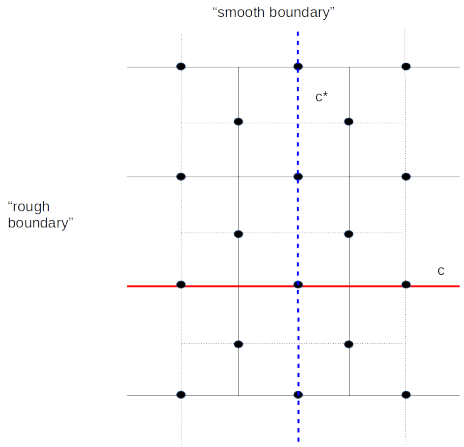

Figure 16: Planar codes - usual visualization

In most treatments, the lattices L and L^* are visualized a bit differently, namely with the rough boundaries V and V^* removed, as in figure 16. This visualization makes the choice of the names more transparent and has the additional advantage that all edges do now again carry a qubit. However, to work with the relative chain complex and its homology groups, it is more convenient to consider the boundaries V and V^* as part of the respective lattice, and we will do that in the sequel (at least implicitly).

Now we can define a stabilizer code as we did it before in the toric geometry. For a face f, we can again define a face operator

$$Z_f = \prod_{e \in \partial f} Z_e = \prod_{e \in \partial f \setminus V} Z_e$$

with the understanding that those edges that are in the rough boundary act trivially. Thus the face operator associated with a face in the interior will consist of four Z-operators. A face operator for a face that touches the boundary along the smooth part only also has four Pauli operators, whereas for a face f touching the rough boundary, Z_f will be a product of three Pauli Z operators only. Similarly, given a vertex v, we can define the vertex operator

$$X_v = \prod_{e \in \partial^* v} X_e = \prod_{e \in \partial^* v \setminus V} X_e$$

Note that a vertex on the smooth boundary will give raise to a vertex operator X_v that is a product of three Pauli X operators only. For a vertex in the rough boundary, the vertex operator is not defined, as the co-boundary operator ∂^* is only defined on the relative chain group $C_0(L, V\mathbb{Z}_2)$.

Having defined face and vertex operators, it is again easy to see that all these operators commute and square to one. Thus they generate an abelian group S. Using arguments similar to the one used in the proof of lemma 8.2, it is not difficult to see (you will have to apply the general fact that the Euler characteristic of a chain complex over a field is equal to that of its homology) that the dimension of the code space is given by

$$\dim T_S = 2^{b_1(L,V)}$$

were now the first Betti number is that of the relative homology. Now the first homology group of L relative to V is easily seen to be one-dimensional, generated by the cycle running from the left to the right through the lattice and connecting the two components of the rough boundary. Therefore the first Betti number is one and the code space is two-dimensional, i.e. the code space encodes a single logical qubit.

The commutator relations between Z and X operators are also as in lemma 8.3, where now the boundary operators ∂ and ∂^* are to be taken relative to the rough boundary V. If, for instance, c is a chain ending and starting at the rough boundary, i.e. a cycle relative to V, then Z_c will commute with any X_v. Similarly, if c^* is a cycle relative to V^*, then X_{c^*} will commute with all the Z_f. This can again be used to construct logical Pauli operators that act on the code space T_S.

In fact, consider the chains c (in the lattice) and c^* (in the dual lattice) marked in figure 16. The resulting operator X_{c^*} and Z_c anti-commute, as the intersection number between c and c^* is one. Clearly, c is a cycle relative to V, and c^* is a cycle relative to V^*. Hence these two operators act on the code space T_S and therefore provide logical Pauli operators

$$\bar{X} = X_{c^*}$$

and

$$\bar{Z} = Z_c$$

This is very similar to the toric case, with the difference that we now have only one pair of logical Pauli operators, corresponding to the fact that our code only encodes one logical qubit.

Finally, let us see how we can measure the error syndromes. Essentially, the idea to do this is as in the toric case, but again we need to be a bit careful at

the boundary. Let us start with the face operators. If a face operator consists of four Pauli operators, we again place a measurement qubit in the centre of the face that we can use for all four CNOT operations, as we did it in the toric code. However, if the face is adjacent to the rough boundary, the face operator will be a product of three Pauli operators only. We again place a measurement qubit inside the face, but this time, the CNOT operations only take place between this measurement qubit and the three data qubits adjacent to them.

Similarly, we need measurement qubits for the vertex operators. This time, the exception applies to the vertices along the smooth boundary. Here one adjacent data qubit is missing, and therefore our measurement qubit interacts with three data qubits only. The entire scheme is shown in figure 17 - this visualization is essentially that of [6], with the roles of filled and open circles exchanged.

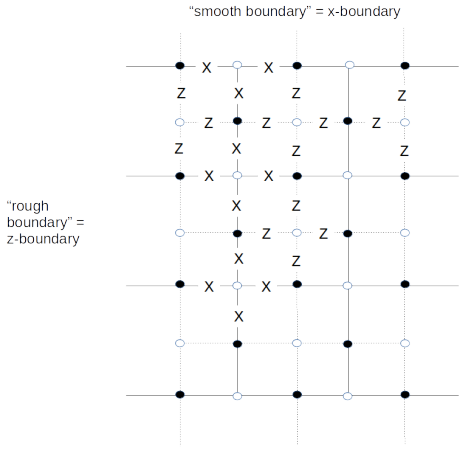

Figure 17: Measurement qubits in the planar code

With that understanding, we can now use the surface code as an error correction code. We can measure the stabilizers and perform individual bit flips and phase flips to create an initial state, apply logical Pauli operators and perform measurements to detect and correct errors. Thus we could apply toric codes as any other codes, maybe as part of a concatenated code. However, one reason why surface codes have become so popular is that there is in fact a different approach to performing quantum computation with surface codes that avoids the use of concatenation and that we will describe in the next section.

12 Creating logical qubits

So far, we have been able to construct a planar code for one logical qubit. If we wanted to combine more than one of these logical qubits, we could start to stack the surfaces on which our qubits are located, so that nearby qubits on different surfaces can interact. However, there is a different approach which allows us to stick to a two-dimensional geometry - creating additional qubits in the surface code.

The code space of our code is the intersection of the plus one eigenspaces of all the stabilizers. Each stabilizer cuts down the original space by a factor two. The idea to creating additional logical qubits is now to simply drop some of the stabilizers in order to enlarge the code space.

To understand how this works, we first have to understand how the surface code is usually used in practice (see [6]). We first put the system into an initial, potentially unknown state. We then perform a series of measurements of all stabilizers. This will force the state into a simultaneous eigenstate of all the stabilizer operators.

Now, the eigenvalues will typically not all be one. We would now have to apply corrections to move our state into the code space. However, instead of doing this, we could as well keep track of the error syndrome and perform the corrections in software when we measure.

We now start our quantum algorithm. After each step of the algorithm, we perform another round of measurements of the stabilizer to detect any errors that might have occurred. Again, we could now correct the errors, but alternatively keep track of them as well and correct the final measurements if needed. In this way, a computation with a surface code is essentially a periodic measurement of all stabilizers in repeating cycles, and logical operations executed between any two subsequent cycles.

Now let us see what happens when we modify our code by removing a face operator Z_f from the set of stabilizers. Thus we perform some initial cycles to put the system into a ground state $|g\rangle$ (here and in the sequel, we assume that errors are actually corrected and not just detected to simplify our calculations) on which all stabilizers act trivially, and starting with the next round of measurements, we exclude a Z_f operator from the measurements. Of course, this will enlarge the code space.

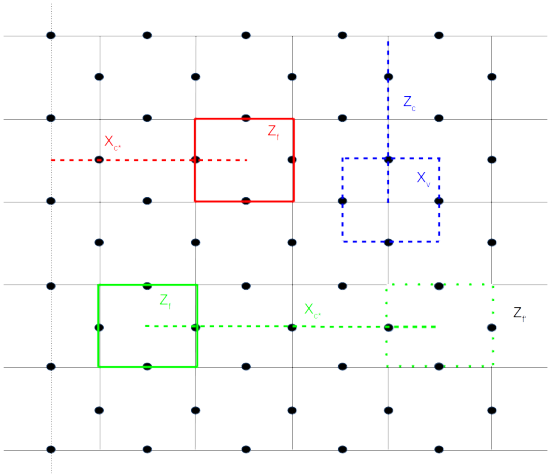

Figure 18: Generating logical qubits with holes

Specifically, let us take a look at the upper left corner of figure 18. Here we have marked a face f and an operator X_{c^*} coming from a string connecting this face with the rough boundary. Then the boundary of c^* consists of one point in the dual lattice, and hence every face operator commutes with X_{c^*}, except

Z_f which anti-commutes. If we now define our code space to be the space fixed by all the X_v and all face operators *except* Z_f, then X_{c^*} and Z_f map that code space to itself. As c^* intersects the boundary of f in one point, Z_f and X_{c^*} anti-commute and therefore define a set of logical Pauli operators, acting on the code space. Thus we have constructed a logical qubit!

There are two things that should be noted. First, we have not changed in any way the physical layout of the surface or added or removed any physical qubits - we have only changed the code (this approach is sometimes called **code deformation**). Second, the global Pauli operators shown in figure 16 still exist and act on our new code space. In fact, our new code space is four-dimensional, but we ignore half of the dimensions and use it only to represent one logical qubit, operated on by the two logical Pauli matrices

$$\bar{X} = X_{c^*}$$

and

$$\bar{Z} = Z_f$$

The logical zero state $|0\rangle_L$ is equal to the original state before turning off that stabilizer, i.e. to the logical zero state of the full surface code. The logical one state $|1\rangle_L$ is then equal to $X_{c^*}|0\rangle_L$.

Note that this construction has a nice physical interpretation. Suppose we create a pair of quasi-particles as before, supported on Z_f and a second face close to the rough boundary. We have seen that we can move these particles by acting with edge operators X_e on them. Thus, we can move the second face on which the particle lives across the rough boundary and now obtain a particle supported on one face only. This is exactly the configuration which is obtained by acting with the logical Pauli X operator of our newly created logical qubit on the ground state, i.e. these logical qubits correspond to pairs of particles where one particle has left the surface.

As usual, this construction has a counterpart in the dual lattice, as shown on the right of figure 18. The configuration marked in blue consists of a vertex operator X_v and a string c connecting that vertex to the smooth boundary. We can now remove X_v from the stabilizer, and obtain a pair of logical operators

$$\bar{X} = X_v$$

and

$$\bar{Z} = Z_c$$

which again anti-commute and operate on our new, enlarged code space. Note that now, the logical ground state can be obtained as

$$|0\rangle_L = \frac{1}{2}(1 + Z_c)|g\rangle$$

where $|g\rangle$ is a ground state (i.e a logical zero) for the full surface code. This type of logical qubit is called a **single X-cut** in [6], whereas the first type of qubit considered is a **single Z-cut** - other authors use the term **defects** for these logical qubits.

This technique allows us in theory to encode a large number of logical qubits in one surface. However, in practice, the restriction that we need to reach the

boundary from each of the faces and vertices that we switch off is restricting our layout options a bit. Fortunately, it turns out that this is not even necessary.

In fact, coming back to the analogy of the quasi-particles, nobody forces us to move the second particle off the surface. To see this, consider the configuration marked in green in the lower part of figure 18. Here we have removed two stabilizers from the code, the face operator Z_f corresponding to the face on the left hand side, and the face operator $Z_{f'}$ corresponding to the face on the right hand side. This will enlarge our code space by a space of dimension four. However, we now gracefully ignore half of that space and instead work with the space spanned by the Pauli operators

$$\bar{Z} = Z_f$$

and

$$\bar{X} = X_{c^*}$$

Because Z_v commutes with every X_v and X_{c^*} commutes with every face operator except Z_f and $Z_{f'}$ (i.e. those that we have removed from the stabilizer), those two operators map the stabilized subspace onto itself. They also anti-commute and thus span a two-dimensional subspace, i.e. a logical qubit. More precisely, we can choose as a basis the states

$$|0\rangle_L = |g\rangle$$

and

$$|1\rangle_L = X_{c^*}|g\rangle$$

where again $|g\rangle$ is a ground state of the original code, i.e. a state on which all face and vertex operators act trivially.

A logical qubit obtained by this construction is called a **double Z-cut**. Of course, we can again repeat this construction in the dual lattice and obtain a **double X-cut**.

13 Moving and braiding logical qubits

Having described individual logical qubits, let us now see how we can manipulate them to perform actual calculations. The main ingredient to this is the ability to **move** logical qubits along the surface.

Let us describe this process for a single Z-cut logical qubit (the other cases are similar). Consider the situation shown in the upper part of figure 19.

Here we consider a logical qubit described by the Pauli operators X_{c^*} and Z_f, i.e the face operator Z_f has been removed from the set of stabilizers. Adjacent to the face f, there is a face that we denote by $\bar{f}$. By removing $Z_{\bar{f}}$ (and adding Z_f again to the set of stabilizers), we would obtain a different code space - we can again think of this different code space as a deformation of the code space given by removing Z_f. We want to describe a process that starts with a state in the code space given by Z_f and ends with a state in the code space given by $Z_{\bar{f}}$ (formally, we could define a deformation to be a chain of code spaces, connected by unitary maps that respect the logical Pauli operators, and we want to describe this map).

To do this, we manipulate the original state in several steps. First, we measure the Pauli X operator X_e for the physical qubit sitting on the edge e of

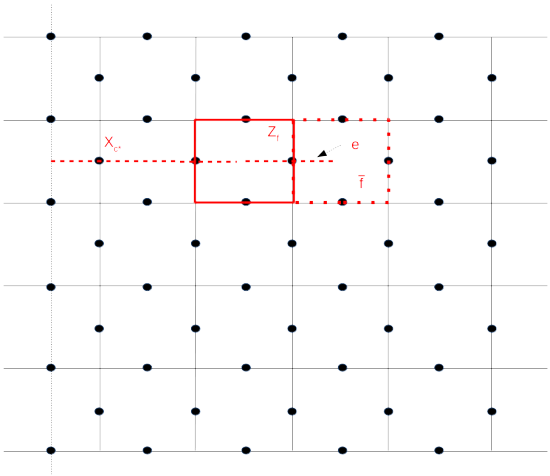

Figure 19: Elementary logical qubit moves

the dual lattice that connects the centre of f with that of $\bar{f}$ - this is the qubit on which both $Z_{\bar{f}}$ and Z_f act. Let us assume for a moment that the outcome of this measurement is one. When the system was initially in the state $|\psi\rangle$, the system is now in the state

$$\frac{1}{2}(1 + X_e)|\psi\rangle$$

up to a normalization factor. Next, we do another measurement - this time we measure Z_f. This will force the system into an eigenstate of Z_f. Let us assume again that the outcome of this measurement is plus one, so that the system is now in the new code space obtained by removing $Z_{\bar{f}}$ instead of Z_f from the stabilizer. Then the entire process is described by a normalization factor times the operator

$$P = \frac{1}{4}(1 + Z_f)(1 + X_e)$$

Knowing that X_e and Z_f anti-commute, we can write this as

$$P = \frac{1}{4}\big[1 + Z_f + X_e + Z_f X_e\big] = \frac{1}{4}\big[(1 + Z_f) + X_e(1 - Z_f)\big]$$

From this, we can easily read off how the operator acts on the eigenstates of Z_f. We find that, after normalization,

$$|0\rangle_L \mapsto |0\rangle_L$$
$$|1\rangle_L \mapsto X_e|1\rangle_L = X_e X_{c^*}|0\rangle_L = X_{c^*+e}|0\rangle_L$$

Thus we find that this process maps the logical state $|0\rangle_L$ to the logical zero state of the new code, and the same is true for the logical one state. In other words, this operation is moving a particle from f to $\bar{f}$ while preserving the logical information encoded in the state. This is what we mean when we say that we move a logical qubit around. Note that (because $\partial^* e = f + \bar{f}$) the logical Pauli operators for the old and the new code are related by

$$\bar{X} \mapsto X_e \bar{X}$$
$$\bar{Z} \mapsto Z_{\partial^* e} \bar{Z}$$

In this case, it is obvious what the new logical operators are, but we could as well determine them by the requirement that the logical X-operator followed by P should be the same as first applying P and then the logical X-operator, and the same for the logical Z-operator. From this requirement, we could then derive the correct logical states, identified as eigenvalues of the logical Z-operators (and determined up to a phase by this).

Let us quickly see how this calculation changes when the measurements have different outcomes. First, let us assume that the first measurement - the measurement of X_e - yields a minus one, whereas the second one still yields a one. Then

$$P = \frac{1}{4}\big[1 + Z_f - X_e - Z_f X_e\big] = \frac{1}{4}\big[(1 + Z_f) - X_e(1 - Z_f)\big]$$

from which we can again read off that

$$\begin{aligned} &|0\rangle_L \mapsto |0\rangle_L \\ &|1\rangle_L \mapsto X_e|1\rangle_L = -X_e X_{c^*}|0\rangle_L = -X_{c^*+e}|0\rangle_L \end{aligned}$$

This is again in the code space, but to make sure that our requirements above hold, we need to redefine the logical X-operator by adding a minus sign:

$$\begin{aligned} \bar{X} &\mapsto -X_e \bar{X} \\ \bar{Z} &\mapsto Z_{\partial^* e} \bar{Z} \end{aligned}$$

Things are a bit more complicated if the outcome of the second measurement (that of Z_f) is minus one. Let us consider the example that this is combined with a first measurement with outcome one. Then

$$P = \frac{1}{4}(1 - Z_f)(1 + X_e) = \frac{1}{4}\big[1 - Z_f + X_e + X_e Z_f\big] = \frac{1}{4}\big[(1 - Z_f) + X_e(1 + Z_f)\big]$$

so that

$$\begin{aligned} |0\rangle_L &\mapsto X_e|0\rangle_L \\ |1\rangle_L &\mapsto |1\rangle_L \end{aligned}$$

The problem with this is that this it not even in the new code space - which we expect, because we know that after the last measurement, our state is in a -1 eigenspace of Z_f, not a $+1$ eigenspace. So we need to adjust our procedure a bit and act once with X_e after the second measurement to get into a $+1$-eigenspace again - effectively, this is an error correction to get into the new code space. Thus our new version of P is

$$P = \frac{1}{4} X_e \big[(1 - Z_f) + X_e(1 + Z_f)\big] = \frac{1}{4}\big[(1 + Z_f) + X_e(1 - Z_f)\big]$$

which is the same as in the case considered (both measurements plus one). Thus our procedure to move a logical qubit in the direction of an edge e in the dual lattice is now

1. Measure X_e and let ϵ denote the result
2. Measure Z_f

3. If the result of this measurement is -1, then apply X_e
4. Replace the logical Pauli operators $\bar{X}$ and $\bar{Z}$ by $\epsilon X_e \bar{Z}$ and $Z_{\partial^* e}\bar{Z}$.

Note that in [6], this correction is not done physically by applying a bit flip, but done in software, i.e. by keeping track of the sign changes in the control software.

Of course this procedure can be iterated to move a Z-type logical qubit along a string c. If c is a closed loop, we eventually return to our original location and our original code space, but have transformed the set of Pauli operators, corresponding to a transformation of the logical qubit. This is especially interesting if two qubits are involved. In this case, we can obtain a transformation by braiding the qubits, i.e. moving one qubit around another, and eventually returning to the original locations. This is very similar to the idea of encoding quantum information by braiding anyons.

We will not go into all the details, but briefly describe one example and refer the reader to [6] for the details. We consider two logical qubits, one obtained as a Z-cut, the other obtained as an X-cut (it does not matter whether these qubits are single cut or double cut qubits).

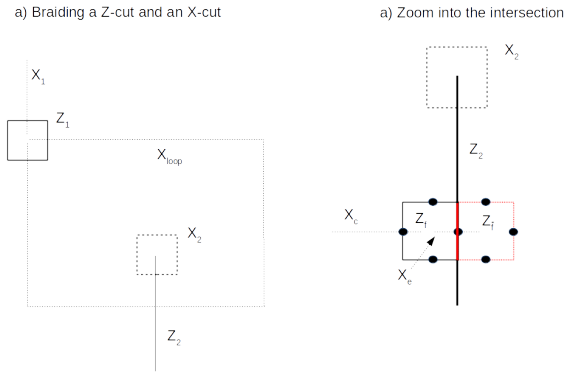

Figure 20: Moving a Z-cut around an X-cut

The process is illustrated on the left hand side of figure 20. We start with a pair of logical qubits given by an X-cut and a Z-cut. We then move the face corresponding to the Z-cut around the vertex corresponding to the X-cut using elementary moves as described above. The loop along which we move - X_{loop} - is winding once around the vertex at which the X-cut has been carried out.

Let us now see how the 2-qubit system spanned by the Pauli operators Z_i and X_i transforms if we carry out the move. We start with a ground state $|g\rangle$ of the full code (and therefore of Z_1 which is a face operator), so that the state

$$|\zeta\rangle = \frac{1}{2}(1 + Z_2)|g\rangle$$

is the logical state $|00\rangle$ of our 2-qubit system, i.e. a simultaneous $+1$-eigenstate

of Z_1 and Z_2. We can then form the states

$$|01\rangle = X_2|00\rangle$$
$$|11\rangle = X_1X_2|00\rangle$$
$$|10\rangle = X_1|00\rangle$$

which together with $|00\rangle$ span the entire code space. Let us now see how these states transform under the move.

As long as we operate sufficiently far away from the string defining Z_2, the action on the two logical qubits is completely decoupled, so we know from what we have discussed how the states and logical operators transform. However, the situation is a bit more subtle when we reach the string defining Z_2. The critical point is displayed on the right hand side of figure 20. So we have reached the vertex f, i.e. removed the face operator Z_f from the set of stabilizers, and now want to move on to the face $\bar{f}$.

Thus as before, we first measure the X-operator of the qubit located on the edge where those two faces meet, i.e. the operator X_e. For simplicity, let us assume that the outcome of this measurement is one. We then measure $Z_{\bar{f}}$, and again let us assume that the outcome is one. Thus we apply the projection

$$P = \frac{1}{4}\big[(1+Z_f)(1+X_e)\big] = \frac{1}{4}\big[(1+Z_f)+X_e(1-Z_f)\big]$$

It is again easy to calculate how this operator maps (up to a normalization) the eigenstates of the current code space.

$$|00\rangle_L = |\zeta\rangle \mapsto |\zeta\rangle = |00\rangle$$
$$|01\rangle_L = X_2|\zeta\rangle \mapsto X_2|\zeta\rangle = |01\rangle_L$$
$$|10\rangle_L = X_1X_c|\zeta\rangle \mapsto X_1X_{c+e}|\zeta\rangle$$
$$|11\rangle_L = X_1X_cX_2|\zeta\rangle \mapsto X_1X_{c+e}X_2|\zeta\rangle$$

from which we can immediately read off the new logical states.

$$|00\rangle_{L'} = |00\rangle_L$$
$$|10\rangle_{L'} = X_e|10\rangle_L$$
$$|11\rangle_{L'} = X_e|11\rangle_L$$
$$|01\rangle_{L'} = |01\rangle_L$$

This is still what we expect. However, there is a twist with the new logical operators of our code. In fact, Z_2 is no longer the correct Pauli operator for the second qubit. The reason is that Z_2 and X_e anti-commute, so that

$$Z_2|10\rangle_{L'} = Z_2X_e|10\rangle_L = -X_eZ_2|10\rangle_L = -X_e|10\rangle_L = -|10\rangle_{L'}$$

Thus there is a minus one which should not be there! Alternatively, the problem can also be seen from the fact that Z_2 and our tentative new X-Pauli operator X_1X_{c+e} for the first qubit anti-commute, but should commute. It is not difficult, though, to guess what the correct Pauli operator is - $Z_2Z_{\bar{f}}$. In fact, this is an operator acting on the code space. It commutes with our new logical X-Pauli operator (because both factors anti-commute with it), and we can check

explicitly that this operator acts correctly on the basis states described above. So we find that the move will transform our logical Pauli operators as follows.

$$\begin{aligned}\bar{X}_1 &\mapsto X_e \bar{X}_1\\ \bar{Z}_1 &\mapsto Z_{\partial^* e} \bar{Z}_1\\ \bar{X}_2 &\mapsto \bar{X}_2\\ \bar{Z}_2 &\mapsto Z_{\bar{f}} \bar{Z}_2\end{aligned}$$

Geometrically, we have replace Z_2 by the operator given by the string which is obtained from the Z_2 string by going around the critical qubit on the right hand side of the face $\bar{f}$. In figure 20, this amounts to replacing the red line by the dotted red line, i.e. we push the string defining the Pauli operator for the second qubit to the right.

Let us see what happens if we do the next move. Let f' denote the face on the right of $\bar{f}$ which we would visit next. Overall, the move from f to $\bar{f}$ and the move from $\bar{f}$ to f' would multiply the logical Z-operator for the second qubit by $Z_{\bar{f}} Z_{f'}$. But on the new code space, $Z_{\bar{f}}$ acts as a stabilizer (we have removed $Z_{f'}$ and added $Z_{\bar{f}}$ again to the set of stabilizers). Therefore, on the code space, this is the same as multiplying by $Z_{f'}$ only. Thus we pick up a copy of the face operator at the head of the chain of faces once we have crossed Z_2.

When we complete the loop and reach the face on which Z_1 lives again, we will therefore overall pick up a factor Z_1. Thus after closing the loop, the Pauli operators transform as follows (ignoring potential signs from X-measurements for a moment)

$$\begin{aligned}X_1 &\mapsto X_{loop} X_1\\ Z_1 &\mapsto Z_1\\ X_2 &\mapsto X_2\\ Z_2 &\mapsto Z_1 Z_2\end{aligned}$$

Now there is one more observation we can make. The loop around which we have moved the qubit is a loop in the dual lattice that can be contracted to X_v without intersecting any other parts of the configuration. Therefore the operator X_{loop} act identically to X_2 on the code space. This gives us our final transformation rule for the braid move.

$$\begin{aligned}X_1 &\mapsto X_1 X_2\\ Z_1 &\mapsto Z_1\\ X_2 &\mapsto X_2\\ Z_2 &\mapsto Z_1 Z_2\end{aligned}$$

Let us now try to understand what all this means. As our loop comes back to the code space, the process of moving the logical qubit around the loop gives raise to a unitary transformation U on the code space. From our previous calculations for each individual step in the move, we know that a state vector of the form $|0\rangle \otimes |x\rangle$ remains unchanged, whereas states of the form $|1\rangle \otimes |x\rangle$

receive a factor X_e during each move, which combines to an overall factor X_{loop}. As X_{loop} is identical to X_2 on the code space, we find that

$$U(|0\rangle \otimes |x\rangle) = |0\rangle \otimes |x\rangle$$
$$U(|1\rangle \otimes |x\rangle) = |1\rangle \otimes X_2|x\rangle$$

In other words, the transformation obtained by moving the qubit around acts a a conditional X_2, i.e. a CNOT gate!

We could - at least up to a phase factor- derive the same result from the transformation rules for the operators. Suppose, for instance, we wanted to understand what U does to $|10\rangle$. During each move, an old state is mapped to a new state and an old set of logical Pauli operators is mapped to a new set, but the defining relations between states and operators remain unchanged. In other words, if a state is an eigenstate of one of the Pauli operators, the transformed state is an eigenstate of the transformed Pauli operator for the same eigenvalue. Therefore $U|10\rangle$ is an eigenstate of the transformed operator $Z_1' = Z_1$ with eigenvalue minus one and an eigenstate of the transformed operator $Z_2' = Z_1 Z_2$ with eigenvalue one. This is only possible if it is an eigenvalue of Z_2 alone with eigenvalue minus one. These properties determine the state up to a phase, and we can conclude that $U|10\rangle = |11\rangle$ up to a phase, giving the same result as our direct argument working with states below. Essentially, this line of arguments employs the Heisenberg picture of quantum mechanics, whereas our argument based on the evolution of the states utilizes the (equivalent) Schrödinger picture.

To summarize, we have found that the operation of moving a logical Z-qubit around a logical X-qubit creates a logical CNOT operation on the code space. This process can be represented by a two-strand braid, where the two strands wrap around each other twice so that their ends return to their original position, as indicated in figure 21.

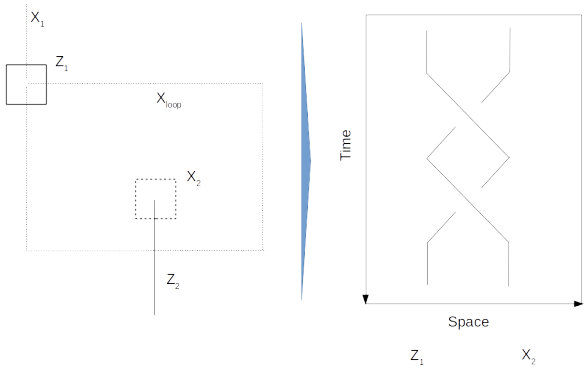

Figure 21: Moving qubits as a braiding operation

We have also seen that the transformation given by such a braiding operation is of a topological nature, i.e. a slightly deformed braid or path gives the same result. Therefore this operation has a certain natural protection againsts faults,

it is **topologically protected**. Unfortunately, as remarked already in [22], the toric code does not allow a representation of a universal set of quantum gates by braids - doing this requires the use of a more sophisticated physical equivalent called **non-abelian anyons**. Still, most operations on a surface code are topologically protected, and it turns out that even without concatenation, a surface code has an accuracy threshold which is somewhere in range of one percent (see [23] or [6]). Therefore surface codes remain a promising technology to implement universal fault-tolerant quantum computation, and are one of the most active areas of current research.

At the same time, a purely topological quantum computer in which all required gates can be implemented by braiding remains an aspirational but desirable goal. The fractional Quantum Hall effect seems to be a good candidate for a physical realization of non-abelian anyons, but other candidates like the so-called nanowires exist. We will not go further into this topic, but refer the reader to [24]or [25] for recent surveys.

A The Pauli matrices

In this section, we fix some notation and recall the most important properties of the Pauli matrices, for which different conventions are in place. We call the matrix

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

the bit-flip operator or inversion operator, as it switches the two basis vectors in the standard basis. Similarly, the matrix

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

flips the relative phase of the vectors in the standard basis and is therefore called the phase flip operator. The product

$$Y = ZX = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

is denoted by Y (whereas other authors use the letter Y differently). The matrices X and Z are hermitian, whereas Y is anti-hermitian. To make contact with the notation usually used in quantum physics, we now introduce the notation

$$\sigma_x = X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \qquad \sigma_y = -iY = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \qquad \sigma_z = Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

These matrices are usually called the *Pauli matrices*. They are hermitian and unitary and have square 1. Looking at the linear combinations $\sigma_z \pm 1$ and $\sigma_x \pm i\sigma_y$, it is also not difficult to see that the three Pauli matrices together with the identity matrix form a complex basis for the set of complex 2x2 matrices, a fact that is exploited heavily in quantum error correction.

Let us now study the multiplicative subgroup generated by the Pauli matrices and the identity. We have the relations

$$\sigma_x \sigma_y = i\sigma_z \qquad \text{plus cyclic}$$

and any two Pauli matrices anti-commute. Thus in order to form a group, we will have to add the scalars $\{\pm 1, \pm i\}$. The resulting group that consists of the sixteen matrices

$$\mathcal{G} = \{\pm 1, \pm\sigma_x, \pm\sigma_y, \pm\sigma_z, \pm i, \pm i\sigma_x, \pm i\sigma_y, \pm i\sigma_z\}$$

is called the **Pauli group**. This group is sometimes denoted by $\mathcal{G}_1$ to indicate that it consists of operators acting on one qubit.

For n qubits, we can as well form the Pauli group $\mathcal{G}_n$ that is generated by tensor products of the Pauli matrices acting on one qubit only. In general, the order of the Pauli group is 2^{2n+2}, where the factor $2^{2n} = 4^n$ is given by the pure products of Pauli matrices and the identity and the factor of $2^2 = 4$ is added by $\{\pm 1, \pm i\}$.

Let us now collect a free remarks on stabilizer subgroups of the Pauli group. The first observation is as follows.

LEMMA A.1. *Let S be a subgroup of $\mathcal{G}$ and T be a non-trivial subspace fixed by all elements in S. Then $-1 \notin S$ and S is abelian.*

Proof. Is is clear that $-1 \notin S$, as -1 does not stabilize any non-trivial subspace. Now assume that S is not abelian, i.e. that there are elements A and B that do not commute. Any two elements of the Pauli group either commute or anti-commute, thus $AB = -BA$. But if x in some non-zero element of T, this implies

$$x = (AB)(x) = -(BA)(x) = -x$$

which is a contradiction. □

So in order to stabilize a non-trivial subspace, the subgroup S needs to be abelian and must not include -1. It is instructive to note that a subgroup of $\mathcal{G}_n$ that does not contain -1 is automatically abelian. If fact, given two elements a, b of such a group, both elements square to one (as every element of the Pauli group squares to one or minus one and minus one is not allowed). Suppose that the elements anti-commute (again, any two elements of the Pauli group either commute or anti-commute). Then

$$ab = -ba$$

which, by multiplying from the right first by a and then by b, implies

$$abab = -1$$

which is a contradiction as minus one is not in our subgroup.

Let us study the structure of such subgroups of $\mathcal{G}_n$ in more detail. First, every element of $\mathcal{G}_n$ has square ± 1. As we assume that -1 is not in S, each element of S therefore squares to one, in other words each non-trivial element of S has order two. Further, S is a finite abelian group. By the structure theory for finite abelian groups, we can therefore conclude that

$$S = \mathbb{Z}_2 \oplus \mathbb{Z}_2 \cdots \oplus \mathbb{Z}_2$$

Let us denote the number of factors by l. Then the order of S is 2^l. As t elements of order t can at most generate a subgroup of size 2^t, it also follows

that the number of factors l is equal to the minimum number of generators that we need to generate the group S. Thus any set of independent generators will have length l.

Under these conditions, one can also relate the dimension of the subspace to the number of generators of the group. In fact, we have the following

LEMMA A.2 (see [4] or [16], section 11.4.2). *Suppose that S is an abelian subgroup of the Pauli group $\mathcal{G}_n$ such that S is generated by k independent elements and $-1 \notin S$. Then the subspace T_S of all vectors that are fixed by all elements of S has dimension 2^{n-k}. The projection onto this subspace is given by the product*

$$\frac{1}{2^k}\prod_{i=1}^{k}(1+S_i)$$

Proof. Let us first prove the following claim: if $S_1, \ldots, S_k$ are independent generators of S, the projection onto T_S is given by the product

$$\frac{1}{2^k}(1+S_k)(1+S_{k-1})\cdots(1+S_1)$$

To see this, we use induction on k. For any $r \leq k$, let us denote by T_r the subspace which is fixed by all generators S_1, S_r. As S_{r+1} commutes with every S_i, it is clear that S_{r+1} is mapping T_r to itself. Further, we have

$$T_{r+1} = \{|\psi\rangle \in T_r \,\text{such that}\, S_{r+1}|\psi\rangle = |\psi\rangle\}$$

In other words, T_{r+1} is the $+1$ eigenspace of S_{r+1} restricted to T_r.

Let P_r be the orthogonal projection onto T_r. To prove our claim by induction, we have to show that the operator

$$\frac{1}{2}(1+S_{r+1})P_r$$

is the orthogonal projection onto T_{r+1}, i.e. the unique idempotent hermitian operator with range T_{r+1}. Now each S_{r+1} is hermitian (as it is unitary with square one), and P_r is hermitian, so this operator is clearly hermitian. Using the induction hypothesis, it is also clear that P_r commutes with S_{r+1} and a short calculation using this shows that our operator squares to one. So we only have to show that its range is equal to T_{r+1}.

To do this, let us first show that its range is contained in T_{r+1}. As S_{r+1} acts on T_r, a vector of the form

$$|x\rangle = \frac{1}{2}(1+S_{r+1})P_r|y\rangle$$

is of course contained in T_r. Now

$$S_{r+1}(1+S_{r+1}) = S_{r+1}+1 = 1+S_{r+1}$$

and hence

$$S_{r+1}|x\rangle = S_{r+1}\frac{1}{2}(1+S_{r+1})P_r|y\rangle = \frac{1}{2}(1+S_{r+1})P_r|y\rangle = S_{r+1}|x\rangle$$

which shows that the range is contained in T_r. To show the converse, assume we are given an element in $|x\rangle \in T_{r+1}$. Then S_{r+1} fixes this element, and the same is true for P_r, so we have

$$|x\rangle = \frac{1}{2}(1 + S_{r+1})P_r|x\rangle$$

showing that $|x\rangle$ is in the range. This completes the proof of our claimed formula for the projection P_{r+1}.

Now let us turn to proving the main statement of the Lemma. Again, we do this by induction. Suppose we have already shown that $\dim T_r = 2^{n-r}$. We then only have to show that the eigenspace of S_{r+1} with eigenvalue one cuts out half of T_r to complete the induction step. This is equivalent to showing that the dimensions of the plus one and minus one eigenspaces of $S_{r+1}P_r$ are equal, which again is (as one can see in a basis of eigenvectors) equivalent to the statement that the trace of $S_{r+1}P_r$ is zero. But by the formula for the projection, we see that

$$S_{r+1}P_r = S_{r+1}(1 + S_r)(1 + S_{r-1}) \cdots (1 + S_1)$$

This is a sum of terms each of which is a product of elements of $\mathcal{G}_n$ different from the identity. Each such product has trace zero (to see this, use the fact that the trace of a tensor product is the product of the traces), and therefore the trace of the sum is zero and our proof is complete. □

LEMMA A.3. *Assume that S is an abelian subgroup of the Pauli group $\mathcal{G}_n$ and that the subspace*

$$T_S = \{|\psi\rangle \text{ such that } s|\psi\rangle = |\psi\rangle \forall s \in S\}$$

is non-trivial. Then the subgroup S is equal to the full stabilizer of T_S, i.e. if g is an element of $\mathcal{G}$ that fixes all elements of T_S, then $g \in S$.

Proof. Pick a set of l elements $s_1, \ldots, s_l$ that are independent and generate S. Let $\bar{S} = S(T)$ denote the full stabilizer of T_S, i.e.

$$\bar{S} = \{g \in \mathcal{G}_n \text{ such that } gx = x \forall x \in T_S\}$$

Clearly, $S \subset \bar{S}$. We want to show that $\bar{S} = S$. Suppose that this is not the case, i.e. that we can find an element $s \in \bar{S}$ which is not already in S. Let us then consider the group S' generated by the set $s, s_1, \ldots, s_l$. Clearly, $T_S = T_{S'}$, and the set of generators $s, s_1, \ldots, s_l$ is independent. But given Lemma A.2, this would imply that

$$\dim T_S = 2^{n-l} = \dim T_{S'} = 2^{n-l+1}$$

which is a contradiction. □

B The check matrix formalism

In this section, we will shortly describe the check matrix formalism and show for a few examples how it can be applied to stabilizer codes. Again, our setup will be as follows. We are given n physical qubits and $k < n$ indepenent elements $S_1, \ldots, S_k$ of the n-dimensional Pauli group $\mathcal{G}_n$ which commute. We consider

the abelian subgroup $S = \langle S_1, \ldots, S_k \rangle$ generated by these elements and assume that $-1 \notin S$. By what we have seen in the previous section, we therefore know that the code space T_S is non-trivial and has dimension 2^{n-k}.

Now assume that we are given an element g of the Pauli group $\mathcal{G}_n$. As, up to a phase, σ_y is a product of σ_x and σ_z, we can write this as

$$g = \mu(\sigma_x^1)^{a_1}(\sigma_z^1)^{b_1} \cdots (\sigma_x^n)^{a_n}(\sigma_z^n)^{b_n}$$

with an overall phase factor $\mu \in \{\pm 1, \pm i\}$. To such an element, we then associate the vector (following the conventions in [5] which are similar to ours)

$$r(g) = (a, b) \in \mathbb{Z}_2^n \times \mathbb{Z}_2^n = \mathbb{Z}_2^{2n}$$

Thus the vector a has a one at position i if σ_x occurs in the i-th qubit, and b has a one at position i if σ_z occurs at the i-th qubit. The vector a is sometimes called the x-vector and b is sometimes called the z-vector of g, together they form the 2n-dimensional binary vector $r(g)$.

The mapping from $\mathcal{G}_n$ to $\mathbb{Z}_2^{2n}$ is easily seen to be a group homomorphism, i.e. $r(gg') = r(g) + r(g')$. Furthermore, the kernel is exactly the subgroup of order four generated by i. In particular the restriction of r to S is one-to-one, as -1 is not in S and therefore i and $-i$ are not in S as well.

This homomorphism is extremely useful to relate statements about elements of the Pauli group to linear algebra over the field $\mathbb{Z}_2$. As an example, let us prove the following fact.

LEMMA B.1. *Suppose $S_1, \ldots, S_k$ are elements of the Pauli group $\mathcal{G}_n$ such that $-1 \notin \langle S_1, \ldots, S_k \rangle$. Then the S_i are independent if and only if the binary vectors $r(S_i)$ are linearly independent.*

Proof. First, suppose that the $r(S_i)$ are linearly independent and assume that we have a relation among the S_i, which (as the S_i square to one and commute) we can always arrange to be in the form

$$1 = \prod_i S_i^{\alpha_i}$$

with $\alpha_i \in \mathbb{Z}_2$. Applying the r-homomorphism yields

$$0 = r(1) = \sum_i \alpha_i r(S_i)$$

As the $r(S_i)$ were assumed to be linearly independent, this implies that all coefficients α_i are zero.

Conversely, suppose we are given a relation between the $r(S_i)$. Then the same calculation shows that

$$0 = r(\prod_i S_i^{\alpha_i})$$

which shows that this product is in the kernel of r, i.e. it shows that

$$\prod_i S_i^{\alpha_i} \in \{\pm 1, \pm i\}$$

But as -1 is not in the group generated by the S_i, the only value that this product can take is one. Therefore this gives us a relation between the S_i, and as those are independent, we can again conclude that the α_i all vanish. □

Now let us try to find a condition on the $r(g)$ that tells us whether two elements commute or anti-commute. Suppose we are given two group elements g and g' and

$$r(g) = (a, b)$$
$$r(g') = (c, d)$$

Then the binary scalar product ad measures in how many qubits a σ_x in g and a σ_z in g' pair up. Each of these pairs gives us a factor minus one if we commute g' by g. Similarly, cb measures in how many qubits a σ_x in g' meets a σ_z in g. Thus we see that g and g' commute if and only if

$$ad + bc = 0$$

and anti-commute if and only if

$$ad + bc = 1$$

This formula defines a **symplectic product** on the vector space $\mathbb{Z}_2^{2n}$, which is of course simply the standard symplectic product defined on any even-dimensional vector space. To write down this product, we can introduce the $2n \times 2n$ matrix

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

which switches a and b when acting on a vector (a, b). Thus we have shown:

LEMMA B.2. *Two elements g and g' of the Pauli group commute if and only if*

$$r(g')\Lambda r(g)^T = 0$$

Finally, it is sometimes useful to understand how an operator g acts on an element $|a\rangle$ of the standard basis in the space of n qubits. To see this, write a as as a binary string

$$|a\rangle = |a_1 a_2 \dots a_n\rangle$$

Now suppose we are given an element g of the Pauli group, again written without any σ_y terms. Then each σ_x acts by flipping a qubit, whereas each σ_z acts by adding a phase. Thus if the i-th position of $r(g)$ is one, a bit is flipped, otherwise a phase is picked up. Let us denote by $r_1(g)$ the projection of $r(g)$ to the first n coordinates. Then, in a hopefully readable notation,

$$g|a\rangle = \mu|a \oplus r_1(g)\rangle$$

with a phase μ, where the $\oplus$ denotes bitwise addition.

After these preliminaries, we can now introduce check matrices. Suppose we are in the standard situation considered in this section, i.e. we have an abelian group S with independent generates $S_1, \dots, S_k$ that stabilizes a non-trivial subspace T_S.

For each of the generators S_i, we can again consider the 2n-dimensional binary vector $r(S_i)$. Let us combine these vectors into a matrix $\mathcal{C}$ called the **check matrix**. Thus if

$$r(S_i) = (a_i, b_i)$$

then the check matrix is the matrix

$$\mathcal{C} = \big(A \mid B\big)$$

with k rows and 2n columns, where we use the vertical bar in the middle to separate the first n columns from the second n columns. As the elements S_i are independent, we know from our previous discussion that the matrix has full rank r.

To illustrate how the check matrix can be used to translate group theoretical questions into ordinary linear algebra, let us prove the following statement.

LEMMA B.3. *The normalizer $N(S)$ has $4 \cdot 2^{2n-k}$ elements.*

Proof. We know that an element g is in the normalizer of S if and only if it commutes with every generator S_i, i.e. if

$$r(S_i)\Lambda r(g)^T = 0$$

for all i. Now the $r(S_i)$ are the rows of $\mathcal{C}$. Combining these k equations is therefore equivalent to the condition

$$\mathcal{C}\Lambda r(g)^T = 0$$

As Λ is an isomorphism, the kernel of $\mathcal{C}\Lambda$ has the same dimension as the kernel of $\mathcal{C}$. But the dimension of this kernel is $2n-k$, as the rank of the check matrix is k.

So the normalizer is the pre-image of an linear subspace of dimension $2n-k$ under the homomorphism r. The kernel of this homomorphism is the group $Z = \{\pm 1, \pm i\}$ which is in the center of the normalizer. This gives an additional factor four and shows that the cardinality of the normalizer is $4 \cdot 2^{2n-k}$ as claimed. □

But we can do even more when we bring the check matrix into a standard form. In fact, we can start to apply the basic steps of a Gauss-Jordan elimination to it, more specifically to its left side! Let us see how these steps are reflected in the group S.

First, we can of course re-label the qubits at will. This will simply exchange two columns of the left side (and the corresponding columns on the right side as well). Next, we can re-label the generators. This simply corresponds to a permutation of the rows. And finally, we can pick an index i and multiply all generators except S_i by S_i, i.e. pass to the set $\{S_1S_i, S_2S_i \ldots, S_i, \ldots, S_kS_i\}$. This corresponds to adding row i to all other rows of the matrix. Thus all operations in the Gauss-Jordan algorithm (over $\mathbb{Z}_2$) are reflected by operations on the group level that do not change the stabilizer group S.

Thus, starting with any check matrix, we can apply Gauss-Jordan elimination steps, i.e. change the set of generators by the operations explained above, to the left hand side of the matrix and end up with a check matrix in the following structure.

$$\mathcal{C} = \left(\begin{array}{cc|cc} 1 & A & B & C \\ 0 & 0 & D & E \end{array}\right) \tag{2}$$

Here the unit matrix in the upper left corner has dimension r, where r is the rank of the left hand side of the check matrix.

As an example how useful this representation can be, let us now see how we can systematically construct vectors in the code space T_S, i.e. vectors invariant under S. The obvious approach which readers familiar with group theory will immediately recognize is averaging, i.e. starting with some element, say $|0\dots0\rangle$, and then looking at the sum

$$|x\rangle = \sum_{s\in S} s|0\dots0\rangle$$

As multiplying by an element in S simply reorders the sum, it is clear that this element is invariant. However, it is not clear that it is not zero (and we have not used the necessary condition that $-1 \notin S$ so far). However, with the standard assumptions that we usually make, this is true, and the check matrix standard form can be used to prove it.

LEMMA B.4. *Suppose that the abelian group S has a non-trivial invariant subspace T_S. Then the invariant element*

$$\sum_{s\in S} s|0\dots0\rangle$$

is not zero.

Proof. So far we have not chosen any generators. Let us do this now, i.e. we pick independent generators $S_1,\dots,S_k$. As we assume that the code space is non-trivial, we already know that $k < n$ and that $-1 \notin S$. By the arguments above, we can also assume that the generators are chosen such that the check matrix is in the standard form (2) described above.

We already know (and there the condition that $-1 \notin S$ comes into play) that the group S is a product $(\mathbb{Z}_2)^k$. Using this, it is not difficult to show that

$$\sum_{s\in S} s|0\dots0\rangle = \frac{1}{2^k}\prod_{i=1}^{k}(1+S_i)|0\dots0\rangle$$

and therefore be expressed by the generators alone. Now let again r denote the rank of the left hand side of the check matrix. Then all S_i with $i > r$ do not contain any σ_x matrices, as the rows of the check matrix below index r are zero in the first n positions. As σ_z acts trivially on $|0\dots0\rangle$, we can ignore these factors (as they only give a factor two) and see that

$$\sum_{s\in S} s|0\dots0\rangle = \frac{1}{2^k}\prod_{i=1}^{r}(1+S_i)|0\dots0\rangle$$

which expresses our element by the first r generators only (these generators are called type I generators in [5]). Reversing the order, we can write the same expression as

$$\sum_{s\in S} s|0\dots0\rangle = \frac{1}{2}(1+S_r)\dots\frac{1}{2}(1+S_1)|0\dots0\rangle$$

Now let us take a look at the check matrix in the standard form to see how S_1 acts. We focus on the first r bits and write a binary vector of length n as

$$|x;?\rangle$$

with a binary string x of length r and the question mark indicating that we do not care about the other positions. Then, obviously

$$S_1|0\dots0\rangle = \mu_1|10\dots0;?\rangle$$

as S_1 only flips the first qubit, where μ_1 is a phase factor. Thus

$$(1+S_1)|0\dots0\rangle = (|0\dots0;?\rangle + \mu_1|10\dots0;?\rangle)$$

If we now apply S_2 to this, then S_2 will only flip the second qubit and let the bits among the first r bits alone (and it might act on the qubits after position r). Thus

$$S_2(1+S_1)|0\dots0\rangle = (\mu_2|01\dots0;?\rangle + \mu_3|110\dots0;?\rangle)$$

with new phase factors μ_2, μ_3. Therefore

$$\begin{aligned}(1+S_2)(1+S_1)|0\dots0\rangle =& |0\dots0;?\rangle + \mu_1|10\dots0;?\rangle \\ &+ \mu_2|01\dots0;?\rangle + \mu_3|110\dots0;?\rangle\end{aligned}$$

which already has four components. Using again the standard shorthand notation (replacing binary strings by their decimal value), we could write this as well as

$$(1+S_2)(1+S_1)|0\dots0\rangle = |0;?\rangle + \mu_1|1;?\rangle + \mu_2|2;?\rangle + \mu_3|3;?\rangle$$

Now we can continue like this, exploiting the fact that each new S_i will only act on bits among the first r bits which all the previous operators have left alone. We therefore see that

$$\sum_{s\in S} s|0\dots0\rangle = \frac{1}{2^k}\sum_{i=0}^{r}\mu_i|i;?\rangle$$

where all the μ_i are non-zero. Thus this is a sum of non-zero coefficients times different vectors of the standard basis and therefore non-trivial. □

One can use similar arguments and a more refined standard form to obtain much more information on the structure of the normalizer $N(S)$. This is carried out in section 4.1 of [5]. The main result is that it is possible to find a set of $n-k$ group elements $\bar{X}_i, \bar{Z}_i$ in the normalizer $N(S)$ with the following properties.

1. All the $\bar{X}_i$ commute with each other and with any element in S
2. All the $\bar{Z}_i$ commute with each other and with any element in S
3. $\bar{X}_i$ and $\bar{Z}_j$ commute for $i \neq j$
4. $\bar{X}_i$ and $\bar{Z}_i$ anti-commute
5. The $\bar{Z}_i$ operators are built of σ_z matrices only

Clearly, the $\bar{Z}_i$ are hermitian, as they are a product of σ_z-operators. As the properties above are not changed if we multiply any of the $\bar{X}_j$ with i, we can also assume that the $\bar{X}_i$ are hermitian. Further, none of the $\bar{X}_i$ or $\bar{Z}_i$ can be in S,

as each of them anti-commutes with some element in $N(S)$ ($\bar{Z}_i$ anti-commutes with $\bar{X}_i$).

Let us see what else these properties imply. First, let us revisit the (non-zero) state which is obtained by averaging over the full group - we will denote this state by $|\bar{0}\rangle$ in the sequel:

$$|\bar{0}\rangle = \sum_{s\in S} s|0\dots 0\rangle$$

As any $\bar{Z}_i$ can be commuted past the $s \in S$ and acts trivially on the state $|0\dots 0\rangle$ (being composed of σ_z's only), we find that

$$\bar{Z}_i|\bar{0}\rangle = \bar{Z}_i \sum_{s\in S} s|0\dots\rangle = \sum_{s\in S} s\bar{Z}_i|0\dots\rangle = |\bar{0}\rangle$$

So $|\bar{0}\rangle$ is a $+1$ eigenstate of all the $\bar{Z}_i$, very similar to the state $|0\rangle$ being an eigenstate of σ_z. Now we can apply the type of arguments that physically inclined readers will know from the algebraic treatment of the quantum harmonic oscillator to learn more about the action of the $\bar{X}_i$ on the "ground state" $|\bar{0}\rangle$. For instance

$$\langle\bar{0}|\bar{X}_i|\bar{0}\rangle = \langle\bar{0}|\bar{X}_i\bar{Z}_i|\bar{0}\rangle = -\langle\bar{0}|\bar{Z}_i\bar{X}_i|\bar{0}\rangle = -\langle\bar{0}|\bar{X}_i|\bar{0}\rangle$$

which implies that $\bar{X}_i|\bar{0}\rangle$ is orthogonal to $|\bar{0}\rangle$ (and of course not zero, as $\bar{X}_i$ is unitary). Similar arguments show that given any two different bit-strings $(a_1,\dots,a_n)$ and $(b_1,\dots,b_n)$, the two vectors

$$\bar{X}_1^{a_1}\cdots\bar{X}_n^{a_n}|\bar{0}\rangle$$

and

$$\bar{X}_1^{b_1}\cdots\bar{X}_n^{b_n}|\bar{0}\rangle$$

are orthogonal. Thus the encoding

$$(a_1,\dots,a_{n-k}) \mapsto \bar{X}_1^{a_1}\cdots\bar{X}_n^{a_n}|\bar{0}\rangle$$

creates an orthonormal basis of the vector space T_S, presenting this vector space as a tensor product of $n-k$ copies of a two-dimensional Hilbert space and thereby identifying the quotient $N(S)/S$ with the Pauli group $\mathcal{G}_{n-k}$.

Thus the action of $N(S)/S$ on T_S can be identified with the action of the Pauli group on a standard quantum register with $n-k$ qubits. We have therefore not only encoded $n-k$ logical qubits in n physical qubits, but also identified unitary operators, i.e. quantum gates, that act on the encoded qubits similar to the action of the Pauli matrices on the physical qubits. This is one of the advantages of the stabilizer formalism - it does not only provide a code, but also a set of Pauli operators on the encoded states.

To conclude this section, let us now verify that instead of starting with a group S defined by independent generators S_i, we could as well choose a check matrix as starting point and construct the code from there. Specifically, we have the following result.

LEMMA B.5. *Assume that $\mathcal{C}$ is a $r\times 2n$ matrix with independent rows such that the symplectic product of any two rows is zero. Then there are commuting and independent hermitian group elements $S_1,\dots,S_k$ with $S_i^2 = 1$ such that the group S generated by the S_i does not contain -1.*

Proof. First, let us choose group elements S_i such that $r(S_i)$ is the i-th row of the given check matrix $\mathcal{C}$. Then, according to our assumptions about this matrix, the S_i commute. Being an element of the Pauli group, each S_i is either hermitian or anti-hermitian, and each S_i is unitary. Thus the condition $S_i^2 = 1$ is equivalent to S_i being hermitian. If any of the S_i is not hermitian, then we can replace it by iS_i, and therefore we can find the S_i such that they are all hermitian and have square $+1$. Let S denoote the group generated by the S_i which is clearly abelian.

The only thing that is left to show is that $-1 \notin S$. To see this, consider the map

$$s\colon \mathbb{Z}_2^k \to S$$

that maps a vector $a = (a_1, \ldots, a_k)$ to the group element

$$s(a) = \prod_i S_i^{a_i}$$

As the S_i commute and have square $+1$, this map is well-defined and a group homomorphism. By the definition of S as the group generated by the S_i, it is onto.

Let us now assume that $-1 \in S$. Then we could find a vector $a \in \mathbb{Z}_2^k$ such that $s(a) = 1$. Applying the homomorphism r on both sides yields

$$0 = r(-1) = r(s(a)) = \sum_i a_i r(S_i)$$

But we have assumed that the $r(S_i)$ are linearly independent, so this implies that $a = 0$. But $s(0) = 1 \neq -1$, so that we have found a contradiction and our proof is complete. □

References

[1] M.A. Nielsen, I.L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge, 2010

[2] P. Shor, *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*, SIAM J.Sci.Statist.Comput. Vol. 26 Issue 5 (1997), pp. 1484–1509, available as arXiv:quant-ph/9508027v2

[3] D. Gottesman, *An Introduction to Quantum Error Correction and Fault-Tolerant Quantum Computation*, arXiv:0904.2557v1

[4] D. Gottesman, *A Class of Quantum Error-Correcting Codes Saturating the Quantum Hamming Bound*, arXiv:quant-ph/9604038

[5] D. Gottesman, *Stabilizer Codes and Quantum Error Correction*, Caltech Ph.D. Thesis, available as arXiv:quant-ph/9705052

[6] A.G. Fowler, M. Mariantoni, J.M. Martinis, A.N. Cleland, *Surface codes: Towards practical large-scale quantum computation*, arXiv:1208.0928v2

[7] K. Fujii, *Quantum Computation with Topological Codes - from qubit to topological fault-tolerance*, arXiv:1504.01444

[8] D. Dieks, *Communication by EPR devices*, Physics Letters A, Vol. 92 Issue 6 (1982), 271–272

[9] W.K. Wooters, W.H. Zurek, *A single quantum cannot be cloned*, Nature Vol. 299 (1982), 802–803

[10] A.M. Steane, *A tutorial on quantum error correction*, Proceedings of the International School of Physics Enrico Fermi, course CLXII, Quantum Computers, Algorithms and Chaos, G. Casati, D. L. Shepelyansky and P. Zoller, eds., pp. 132 (IOS Press, Amsterdam 2006)

[11] A.M. Steane, *Multiple Particle Interference and Quantum Error Correction*, arXiv:quant-ph/9601029

[12] P. Shor, *Scheme for reducing decoherence in quantum computer memory*, Phys. Rev. A Vol. 52 (1995) Issue 4, pp. 2493–2496

[13] S. J. Devitt, W. J. Munro, K. Nemoto, *Quantum Error Correction for Beginners*, arXiv:0905.2794v4

[14] E. Knill, R. Laflamme, *A Theory of Quantum Error-Correcting Codes*, arXiv:quant-ph/9604034

[15] R. Laflamme, C. Miquel, J.P. Paz, W.H. Zurek, *Perfect Quantum Error Correction Code*, arXiv:quant-ph/9602019

[16] E. Rieffel, W. Polak, *Quantum computing - a gentle introduction*, MIT Press, Cambridge 2011

[17] J. Kempe, *Approaches to quantum error correction*, Séminaire Poincaré 1 (2005), pp. 65–93, available as arXiv:quant-ph/0612185

[18] D. Arahonov, M. Ben-Or, *Fault Tolerant Quantum Computation with Constant Error*, arXiv:quant-ph/9611025

[19] E. Knill, R. Laflamme, W.H. Zurek, *Threshold accuracy for quantum computation*, arXiv:quant-ph/9610011,

[20] P. Aliferis, D. Gottesman, J. Preskill, *Quantum accuracy threshold for concatenated distance-3 codes*, arXiv:quant-ph/0504218v3

[21] H. Bombin, *An introduction to topological quantum codes*, in *Quantum error correction*, edited by D.A. Lidar and T.A. Brun, Cambridge University Press, New York, 2013, available as arXiv:quant-ph/1311.0277

[22] A.Yu. Kitaev, *Fault-tolerant quantum computation by anyons*, arXiv:quant-ph/9707021

[23] E. Dennis, A. Kitaev, A. Landahl, J. Preskill, *Topological quantum memory*, arXiv:quant-ph/0110143

[24] V. Lahtinen, J.K. Pachos, *A Short Introduction to Topological Quantum Computation*, SciPost Phys. 3, 021 (2017) and arXiv:1705.04103

[25] B. Field, T. Simula, *Introduction to topological quantum computation with non-Abelian anyons*, arXiv:1802.06176

[26] S. Rao, *Introduction to abelian and non-abelian anyons*, arXiv:1610.09260

[27] S.B. Bravyi, A.Yu. Kitaev, *Quantum codes on a lattice with boundary*, arXiv:quant-ph/9811052