

Tutorial on Quantum Error Correcting Codes in the ZX-calculus

Lia Yeh

September 15, 2025

1 Stabilizer formalism

In this section, we introduce the stabilizer formalism for qubit computation.

First, we define the Pauli group, whose elements are called Paulis.

Definition 1 (Pauli group). The *Pauli group* \mathcal{P}_n is the group of n -qubit operators generated by the *Pauli matrices*:

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

Sometimes, the Pauli matrices also includes the identity matrix

$$I = X^2 = Y^2 = Z^2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Some definitions of the Pauli group also include arbitrary global phases $e^{i\theta}$ for any real θ .

The three Pauli matrices X , Y , and Z are also the rotations of angle π about the x , y , and z axes of the Bloch sphere, respectively.

Integer powers of i are always in the Pauli group because

$$I = -iXYZ = -iYZX = -iZXY$$

and so

$$iI = XYZ$$

Remark 2. When operators are written sequentially with symbols as done above, the intended order of composition matches that of matrix multiplication, from right to left.

In quantum information and computation, in many cases this way of writing operators with a space or nothing in between them actually denotes their tensor product, rather than matrix multiplication. Which of the two is being denoted is supposed to be clear from context—for example, when referring to quantum error correcting codes and stabilizer tableaus, tensor product is intended.

We seldom use a semicolon ; to denote composing operators from left to right, which for instances matches how quantum circuits are drawn. For example, for the composition of two operators A and B , B then A is written as

$$AB = B; A$$

As is conventionally defined in the literature, something is *generated* by a set of generators when every element of that something can be expressed by a finite composition of the generators. Generally, the allowed compositions are multiplication and tensor product. That something could be a group, a vector space, the quantum circuits possible over a gate set, a fragment of quantum computation, a fragment of a graphical calculus, or other things, where the or here is not mutually exclusive.

Next, we define the Clifford group, whose elements are called Cliffords.

Definition 3 (Clifford group). The *Clifford group* C_n is the group of n -qubit operators generated by the *Clifford gate set* [Got98b]

$$S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, \quad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad CX = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Sometimes, this definition of the Clifford group also includes arbitrary global phases.

The Pauli group is a subgroup of the Clifford group, which can be seen by observing that all generators of the Pauli group are elements of the Clifford group:

$$Z = S^2, X = HZH, iI = SXSX, Y = iXZ \quad (4)$$

Instead of defining the Clifford group by its generators, we could have also defined it based on the Paulis.

Definition 5 (Clifford unitary). Let U be a unitary acting on n qubits. We say it is *Clifford* when every Pauli is mapped to another Pauli under conjugation by U , i.e. if UPU^\dagger is in the n -qubit Pauli group for any Pauli P . All such unitaries form the Clifford group on n qubits.

A core concept to quantum information that is closely linked to the Paulis and Cliffords, is that of *stabilizer states*. Gottesman defined them as [Got98a]:

Definition 6 (Stabilizer). The *stabilizer* of a state is a generating set of each element of the Pauli group that the state is a $+1$ eigenstate of.

A state which can be completely described by specifying the stabilizer is called a *stabilizer state*.

For any n -qubit stabilizer state, its stabilizer is generated by exactly n independent Pauli strings.


Example 7. $X \otimes X \otimes X$ is a stabilizer of the GHZ state $|GHZ\rangle = \frac{1}{\sqrt{2}}(|000\rangle + |111\rangle)$. A proof in bra-ket notation is

$$(X \otimes X \otimes X) |GHZ\rangle = \frac{1}{\sqrt{2}} ((X|0\rangle) \otimes (X|0\rangle) \otimes (X|0\rangle) + (X|1\rangle) \otimes (X|1\rangle) \otimes (X|1\rangle)) \quad (8)$$

$$= \frac{1}{\sqrt{2}} (|111\rangle + |000\rangle) \quad (9)$$

$$= |GHZ\rangle \quad (10)$$

A proof in the ZX-calculus is



$$\text{Diagram 1} = \text{Diagram 2} = \text{Diagram 3} = \text{Diagram 4} \quad (11)$$

Any operation \mathcal{O} in the stabilizer of a state ψ , leaves it unchanged:

$$\mathcal{O}|\psi\rangle = |\psi\rangle \quad (12)$$

Campbell, Anwar, and Browne defined *stabilizer operations* as follows [CAB12]:

Definition 13. Consider a completely positive map $\mathbb{E}: \mathbb{B}(\mathbb{H}^{d^{in}}) \rightarrow \mathbb{B}(\mathbb{H}^{d^{out}})$. The map is a stabilizer operation if and only if it can be composed from the following elements:

1. Clifford unitaries,
2. measurements and subsequent projections on stabilizer subspaces,
3. preparation of fresh ancilla in a stabilizer state,
4. tracing out of unwanted qudits, and

5. adaptive decision making based both on measurement outcomes and on random coin tosses

We will also introduce how to represent stabilizer states with a *stabilizer tableau*—a binary matrix indicating the Pauli strings that stabilize that state. By representing the application of Clifford gates with tableau operations, any quantum computation in the stabilizer fragment can be efficiently simulated according to the Gottesman-Knill Theorem [Got98a]:

Theorem 14 (Gottesman-Knill). *Any quantum computer performing only:*

- a) *Clifford group gates,*
 - b) *measurements of Pauli group operators, and*
 - c) *Clifford group operations conditioned on classical bits, which may be the results of earlier measurements,*
- can be perfectly simulated in polynomial time on a probabilistic classical computer.*

Therefore, the Clifford group operations used in stabilizer quantum error correction codes, are also operations which are efficiently classically simulable.

However, the Clifford group is not a universal set of quantum states. To be able to approximately generate any quantum gate, at least one non-Clifford gate must be added to the gate set.

Definition 15. For the purposes of this tutorial, we will say that sufficient conditions to call an approximately universal gate set *fault-tolerant* are that:

- it contains gates which generate the Clifford group,
- it contains at least one non-Clifford gate, and
- this non-Clifford gate admits an injection (also referred to as gate teleportation) protocol, which consumes one or more magic states and whose probability of success if multiple rounds of the protocol are necessary converges quickly to one.

As a gate set, note that this definition does not require specification of how these gates are implemented (for instance, a given hardware implementation or noise model). A helpful addition to the above conditions, would be that there is a way to suppress the error of the non-Clifford gate to be arbitrarily low. There are several different proposals for how to implement a Clifford + non-Clifford gate set, such as magic state distillation [BK05], code switching [ADP14; Bom16; KB15], and magic state cultivation [GSJ24].

For now, we introduce magic state distillation and injection, so that the concept of non-Clifford gates can be conveyed. Presume you are sticking to one quantum error correcting code, which can do all Clifford operations fault-tolerantly: these operations in this code don't propagate single-qubit errors to multi-qubit errors before the error correction step. A simple example of a small code with this property, which we will introduce later, is the $[[5, 1, 3]]$ code.

A *magic state* is a non-stabilizer state which can be continually refined to bound its error through a process called *magic state distillation* in a quantum error correcting code, which we will later illustrate.

Example 16. The most well-studied magic state is the $|T\rangle$ state. Injecting one copy of the $|T\rangle$ state into a quantum circuit, lets you implement the most well-studied non-Clifford gate, the single-qubit T gate (also called the $\pi/8$ gate):

$$T = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix} \quad (17)$$

Translating the T magic state injection protocol into the ZX-calculus, we can verify that it implements the T gate deterministi-

cally (up to an irrelevant global phase), regardless of whether the measured bit a was 0 or 1.

$$(18)$$

This is also called a gate teleportation protocol because it teleports a resource state (the T state) in order to enable performing the T gate. In addition to this T state injection and the standard quantum teleportation protocol in the introduction section, many other quantum teleportation protocols admit a clean presentation in the ZX-calculus.

This has the effect of adding the T gate to the Clifford gate set; this combined gate set is called the *Clifford+ T* gate set, and is the most studied gate set in quantum computation. It is (approximately) universal for quantum computation, meaning that any quantum circuit can be approximated to arbitrary precision by a circuit over the Clifford+ T gate set.

Not all non-Clifford gates possess the same viability in a fault-tolerant setting. The Clifford hierarchy is an important concept in characterizing which gates admit magic state injection protocols or transversal implementation on stabilizer codes.

Definition 19 (Clifford hierarchy). The n^{th} level of the Clifford hierarchy C_n [Got99; PRTC20; RCP19; CGK17] is defined inductively as $C_1 := \mathcal{P}$, the Pauli's and

$$C_n := \{\text{unitary } U \mid \forall P \in \mathcal{P} : UPU^\dagger \in C_{n-1}\}. \quad (20)$$

The Pauli gates form C_1 and the Clifford gates form C_2 , while certain non-Clifford gates belong to C_n for higher n . The Clifford hierarchy is significant because of its relation to the correction of Pauli errors. For instance, when implementing a gate via a state injection mechanism [BK05], a gate from C_i requires a correction from C_{i-1} . Thus, implementing a gate from the third level of the hierarchy using state injection requires a Clifford correction.

2 Classical error correcting codes

For those new to quantum error correction, the easiest starting point is likely to be with classical error correcting codes. Consider you have m classical bits, which each have a small but nonzero probability p of experiencing a bit flip error. Therefore, the probability of one of the bits not being flipped is $1 - p$, and the total probability of no errors occurring is $(1 - p)^m$.

In this section, let us consider the three-bit repetition code, which protects against bit flip errors. For each bit of information, encode three copies of it. There are two possible bitstrings that encoded bits can take: 000 and 111—these are called the *codewords* of this code. This code has a *code distance*, or commonly just called *distance*, of $d = 3$ because the minimum Hamming distance between any two codewords is 3. This means that for one codeword to experience so much error that it becomes another codeword, there must have been a minimum of three bits flipping.

If all three of these bits are the same, then either all three bits flipped (which is highly unlikely, with probability p^3), or none of them flipped (with probability $(1 - p)^3$, which is the most likely outcome if $p < 0.25$). If two of the bits are the same, then so long as $p < 0.5$, it is more likely that only one of the bits flipped (with probability $3p(1 - p)^2$), than two of them flipped (with probability $3p^2(1 - p)$). Then we should presume that the more likely outcome was the one which occurred. We look at which value (0 or 1) was the majority of the three bits, and flip whichever of the three bits is not equal to that.

This value of $p = 0.5$ is the *threshold*. If the physical error rate p exceeds this threshold, then the probability that the decoding fails (does not correct the error that occurred) is higher than the probability that it succeeds. This value of 0.5 is the highest threshold possible for this error model—note this dependence of thresholds on the error model. However, this is not considered a good code because its *encoding rate* of $\frac{1}{3}$ is very low, of 1 logical bit (of encoded information) per every 3 physical bits (of bits you actually use to store and compute information).

As the decoding takes place locally to each set of three bits which encode a logical bit, each *code block* consists of three physical bits and one logical bit. More generally, each code block consists of k logical bits and n physical bits, where $n \geq k$. Codes are commonly referred to by the three numbers $[[n, k, d]]$, where d is the code distance—the minimum Hamming distance between codewords, i.e. the minimum number of bitflips to turn one codeword into another.

We can represent classical error correcting codes diagrammatically. This is because any computation on classical bits can be represented diagrammatically, using the generators $\{\text{COPY}, \text{ADD}, \text{MULT}\}$. The *encoder* map of the three-bit repetition code is the 1-input, 3-output COPY operator. Indeed, by the copy rule, inputting the bit 0 into the encoder gives the output 000, and inputting the bit 1 gives the output 111.

(21)

We can also see that the *logical operation* of doing a bit flip on the input bit, is implemented by the physical operation of doing a bit flip on all three physical bits.

(22)

The encoder of any code is an isometry—it is reversible by doing its one-sided inverse, called the *ideal decoder* because it decodes all codewords and annihilates all other inputs to it. For this example code, the ideal decoder is the 3-input, 1-output MATCH operator which sends 000 to 0 and 111 to 1, and annihilates all other inputs.

(23)

Doing the ideal decoder, followed by the encoder, is notably not the identity operation. This follows from the fact that compressing n bits (or qubits) into $k < n$ bits (or qubits) necessarily loses information.

(24)

In fact, it is the projector onto the *code space* of the code, as it is identity on any codewords, and annihilates all three-bit states that are not codewords.

3 A quantum error correcting code

With quantum error correcting codes, there are two main differences with classical bit error correcting codes. First, rather than discrete codewords which are bitstrings, the space of encoded quantum states is continuous. For example, taking the three-bit repetition code from before, we can apply the trivial way to lift a classical code to a quantum code, by replacing each generator in the encoder with its quantum analogue. Replacing 0 with $|0\rangle$; 1 with $|1\rangle$; and the 1-input, 3-output COPY operation with the 1-input, 3-output Z spider: we get the three-qubit bit-flip code.

(25)

In addition to $|0\rangle$ and $|1\rangle$, we can encode any single-qubit state; this is determined by linearity of how the encoder maps $|0\rangle$ and $|1\rangle$. For an arbitrary single-qubit state determined by two complex coefficients c_0 and c_1 such that $|c_0|^2 + |c_1|^2 = 1$, the encoder maps this as:

$$c_0 |0\rangle + c_1 |1\rangle \mapsto c_0 |000\rangle + c_1 |111\rangle \quad (26)$$

The code space of this code is a vector space, which can be fixed by a basis for it—for instance $\{|000\rangle, |111\rangle\}$.

Note that here we adhere to the convention of omitting explicitly denoting the tensor product, e.g.

$$|000\rangle = |0\rangle \otimes |0\rangle \otimes |0\rangle = |0\rangle |0\rangle |0\rangle$$

and

$$IZZZZI = (I \otimes Z \otimes Z)(Z \otimes Z \otimes I) = (IZ) \otimes (ZZ) \otimes (ZI) = Z \otimes I \otimes Z = ZIZ$$

In the classical case we saw in the previous section, we could simply read the bits to see if any error occurred. The problem with doing this for quantum states, is that measuring whether a qubit is in either $|0\rangle$ or $|1\rangle$ necessarily collapses any qubit state that is a superposition of $|0\rangle$ and $|1\rangle$. The way that quantum errors are detectable without collapsing the encoded quantum information, is by measuring the parity of two pairs of the three qubits. A fault-tolerant circuit for measuring one of these pairs is:

$$(27)$$

A proof that this measures the parities of the Z basis components of the input states is:

$$(28)$$

If no bit-flip error occurred, both two-qubit parity measurements should be 0. If any bit-flip error occurred, then at least one of these two parity measurements should be 1. These measurements are called *syndrome measurements*. The measurement outcomes, called *syndromes*, are information necessary for the decoder to determine which error most likely occurred, and hence what correction to apply.

Rather than by giving basis states for the code space, the other, more common way that the code space of a quantum error correcting code is defined, is by defining generators of the *stabilizers* of the code. The stabilizers of the three-qubit bit-flip code being measured in the above circuit are ZZI and IZZ . Note that X spiders (which are diagonal in the X basis) are stabilized by Pauli Z on every leg, while Z spiders (which are diagonal in the Z basis) are stabilized by Pauli X on every leg. This is simply a consequence of how $X^{\otimes n} |\text{GHZ}\rangle = |\text{GHZ}\rangle$, and $Z^{\otimes n} (H^{\otimes n} |\text{GHZ}\rangle) = (H^{\otimes n} |\text{GHZ}\rangle)$. We can see that these are indeed stabilizers for any codeword, by applying them to the encoder. This holds, whether we apply the Pauli operators (exemplified here for ZZI)

$$(29)$$

or their associated projectors (exemplified here for IZZ)

$$(30)$$

These two stabilizers, ZZI and IZZ , are (the only two, as $n - k = 3 - 1 = 2$) *stabilizer generators* of the code, because all other stabilizers of the code are a product of these.

For this code, the only other non-identity stabilizer is their product, $ZIZ = ZZIIZZ = IZZZZI$. We do not need to measure this stabilizer, because (assuming no errors from the measurements themselves) its measured bit is always the XOR of the other two bits which we have already measured.

We can write down these two stabilizer generators as a binary matrix, called the *parity check matrix* H_Z of the code. Each row of H_Z is a Z-type (i.e. over $\{Z, I\}$) stabilizer generator, with a 1 in the i th column if there is a Z on the i th qubit, and 0 otherwise.

$$H_Z = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix} \quad (31)$$

The second main difference of quantum error correcting codes compared to classical bit error correcting codes, is that in addition to bit-flip errors, there are also *phase-flip* errors. A bit-flip error is a single-qubit Pauli X operator, which flips $|0\rangle$ to $|1\rangle$ and vice-versa. While a phase-flip error (a single-qubit Pauli Z operator) does not affect $|0\rangle$ or $|1\rangle$ other than by an irrelevant global phase, it is an error on superpositions of these states. For instance, it flips $|+\rangle$ to $|-\rangle$ and vice-versa. This code protects against bit-flip errors, but not phase-flip errors. We can see this from the fact that a physical phase-flip error always results in a logical phase-flip error.

$$(32)$$

This is not detectable, as any error which is equivalent to a logical operator is not detectable. This is because if enough error occurs that a codeword is transformed into another codeword, then the decoder cannot differentiate such error from a logical operation that did the same transformation.

This means that there is a geometric way to think about code distance, by plotting codewords in n -dimensional space. Consider a simple decoder which always maps a physical state to the nearest codeword. How to design a code maximizing the distance d between codewords is equivalent to a discretized sphere-packing problem with a sphere centered at each codeword, where physical states inside each sphere are corrected to the codeword at the center of the sphere.

For both classical and quantum error correcting codes, the code distance is the minimum number of different ‘letters’ between any two codewords. The distance of a quantum error correcting code is the minimum number of qubits which must experience a physical error (projected by the syndrome measurements to be Paulis), in order to change one codeword into another. For the reason above, this is equivalent to the minimum weight (number of non-identity Pauli operators) of all the logical operators of the code. For this code, the distance is $d = 1$, because the logical Z operator is $\bar{Z} = ZII = IZI = IIZ$, which is a weight 1 operator.

4 Stabilizer quantum error correcting codes

In the previous section, we gave a conceptual overview of quantum error correction (QEC) through toy classical and quantum examples. In this section, we go a step further and present preliminaries of stabilizer codes. This is an extremely general class of quantum error correcting codes; in fact, almost all of the well-studied quantum error correcting codes in the literature are stabilizer codes.

So far, we have discussed how a QEC code is often referred to by $[[n, k, d]]$, where n is the number of physical qubits, k is the number of logical qubits, and d is the code distance. While the following concepts apply similarly when $n = k$, here for consistent terminology we assume $n > k$, as otherwise the code can neither detect nor correct errors. In the explanation below, the depictions are as if $k > 0$, but the definitions hold nonetheless for when $k = 0$ (which is simply a stabilizer state).

The encoder view

We discussed previously how to represent a QEC code by an encoder map $E : \mathbb{C}^{\otimes k} \hookrightarrow \mathbb{C}^{\otimes n}$:

$$\begin{array}{c} \vdots \\ k \\ \vdots \end{array} \begin{array}{|c|} \hline E \\ \hline \end{array} \begin{array}{c} \vdots \\ n \\ \vdots \end{array} \quad (33)$$

from k logical qubits to n physical qubits. E must be an isometry, meaning that it has a one-sided inverse E^\dagger . E^\dagger is also called the *ideal decoder* because it decodes all codewords, and annihilates all physical states outside the code space.

$$\begin{array}{c} \vdots \\ k \\ \vdots \end{array} \begin{array}{|c|} \hline E \\ \hline \end{array} \begin{array}{c} \vdots \\ n-k \\ \vdots \end{array} \begin{array}{|c|} \hline E^\dagger \\ \hline \end{array} \begin{array}{c} \vdots \\ n \\ \vdots \end{array} = \begin{array}{c} \vdots \\ n \\ \vdots \end{array} \quad (34)$$

As $n > k$, applying EE^\dagger is not the identity, but rather is the projector map $\Pi : \mathbb{C}^{\otimes n} \rightarrow \mathbb{C}^{\otimes n}$ onto the code space of the code. Π can be decomposed as the product of $n - k$ linearly independent projectors, each onto the $+1$ eigenspace of a Pauli string s_i . We will shortly define these Pauli strings as the *stabilizer generators* of the code.

$$\begin{array}{c} \vdots \\ s_1 \\ \vdots \end{array} \begin{array}{|c|} \hline s_1 \\ \hline \end{array} \begin{array}{c} \vdots \\ s_2 \\ \vdots \end{array} \begin{array}{|c|} \hline s_2 \\ \hline \end{array} \begin{array}{c} \vdots \\ s_{n-k} \\ \vdots \end{array} \begin{array}{|c|} \hline s_{n-k} \\ \hline \end{array} = \begin{array}{c} \vdots \\ \Pi \\ \vdots \end{array} = \begin{array}{c} \vdots \\ E^\dagger \\ \vdots \end{array} \begin{array}{|c|} \hline E \\ \hline \end{array} \begin{array}{c} \vdots \\ n \\ \vdots \end{array} \quad (35)$$

The conventional terminology of a code would define the code by its code space, which is bijective with the projector Π . Comparing this to the encoder E in the above equation, we can see that there are as many possible encoders for a given code, as there are unitaries on k qubits. The reason for this is that the code space is the image of the encoder map, i.e. the space of possible output states of the encoder. Information about the code space alone does not specify anything about which logical qubit states are encoded into which physical qubit states.

$$\begin{array}{c} \vdots \\ E \\ \vdots \end{array} \begin{array}{|c|} \hline \Pi \\ \hline \end{array} \begin{array}{c} \vdots \\ n \\ \vdots \end{array} = \begin{array}{c} \vdots \\ E \\ \vdots \end{array} \begin{array}{c} \vdots \\ E^\dagger \\ \vdots \end{array} \begin{array}{c} \vdots \\ E \\ \vdots \end{array} \begin{array}{c} \vdots \\ n \\ \vdots \end{array} = \begin{array}{c} \vdots \\ E \\ \vdots \end{array} \begin{array}{c} \vdots \\ n \\ \vdots \end{array} \quad (36)$$

The code space is defined by *stabilizers* of the code. In the case of stabilizer codes, each stabilizer is a tensor product of Pauli operators, which we might refer to as a “Pauli string” for short. The default convention in the literature is that we take the sign, i.e. the eigenvalue upon applying the Pauli string to any codeword, to be +1.

Whether s below is measurement of any stabilizer(s) in the code (refer to the previous section for an example), or the tensor product of the single-qubit Pauli gates of the stabilizer(s), the following equation holds for all stabilizers s in the stabilizer group S :

$$\begin{array}{c} \vdots \\ \vdots \end{array} \begin{array}{|c|} \hline E \\ \hline \end{array} \begin{array}{c} \vdots \\ \vdots \end{array} \begin{array}{|c|} \hline s \\ \hline \end{array} \begin{array}{c} \vdots \\ \vdots \end{array} = \begin{array}{c} \vdots \\ \vdots \end{array} \begin{array}{|c|} \hline E \\ \hline \end{array} \begin{array}{c} \vdots \\ \vdots \end{array} \quad \forall s \in S \quad (37)$$

For any code, a non-unique choice of $n - k$ *stabilizer generators* is a set of linearly independent stabilizers, such that composing the projector for each stabilizer generator gives Π . This composition can be in any order because all stabilizers must commute with each other. If your measurement outcome of any stabilizer generator is +1, then the act of measurement implements the projector for that stabilizer corresponding to post-selecting that measurement; this means that if your physical state had a small amplitude of (continuous) error, the most probable outcome is that these measurements would project your physical state back into the code space. If you measure all stabilizer generators and observe at least one -1 outcome, this means that the measurement projected the state to outside the code space, and at least one error must have occurred. This discretization of continuous quantum errors due to measurement means that we can count the number of errors, as the number of undesirable single-qubit Pauli operators that occurred after measuring all stabilizer generators.

While many properties of codes can be derived without specifying how the logical states are mapped to the physical states, any implementation must make a choice on fixing the space of all *logical operators*. It suffices to specify $2k$ logical operators to fix any encoder map, which we will refer to as *the* logical operators. Conventionally, the easiest choice of a basis for the logical operators is to define physical implementations for Pauli X and Z on each logical qubit. Note that Pauli Y is not necessary, as it is a product of X and Z up to an irrelevant global phase.

We depict the logical operators, drawn here on the first logical qubit, as [Kis22]:

$$\begin{array}{c} \pi \\ \vdots \end{array} \begin{array}{|c|} \hline E \\ \hline \end{array} \begin{array}{c} \vdots \\ \vdots \end{array} = \begin{array}{c} \vdots \\ \vdots \end{array} \begin{array}{|c|} \hline E \\ \hline \end{array} \begin{array}{c} \vdots \\ \vdots \end{array} \begin{array}{|c|} \hline \overline{X}_1 \\ \hline \end{array} \begin{array}{c} \vdots \\ \vdots \end{array} \quad (38)$$

$$\begin{array}{c} \pi \\ \vdots \end{array} \begin{array}{|c|} \hline E \\ \hline \end{array} \begin{array}{c} \vdots \\ \vdots \end{array} = \begin{array}{c} \vdots \\ \vdots \end{array} \begin{array}{|c|} \hline E \\ \hline \end{array} \begin{array}{c} \vdots \\ \vdots \end{array} \begin{array}{|c|} \hline \overline{Z}_1 \\ \hline \end{array} \begin{array}{c} \vdots \\ \vdots \end{array} \quad (39)$$

A helpful way to think about the logical operators is that they are stabilizers of the Choi state of the encoder map. If all of the logical operators are Pauli strings, then the encoder is a *Clifford* isometry (meaning that it can be decomposed as an n -qubit Clifford unitary acting on k identity wires tensored with $n - k$ $|0\rangle$ states); then, the logical operators of the $[[n, k, d]]$ are stabilizer generators of the $[[n + k, 0, \infty]]$ “code”, which is a stabilizer state of $n + k$ qubits and hence is fixed by $n + k = (n - k) + 2k$ stabilizers.

$$\begin{array}{c} \pi \\ \vdots \end{array} \begin{array}{c} \vdots \\ \vdots \end{array} \begin{array}{|c|} \hline E \\ \hline \end{array} \begin{array}{c} \vdots \\ \vdots \end{array} \begin{array}{|c|} \hline \overline{X}_1 \\ \hline \end{array} \begin{array}{c} \vdots \\ \vdots \end{array} = \begin{array}{c} \vdots \\ \vdots \end{array} \begin{array}{|c|} \hline E \\ \hline \end{array} \begin{array}{c} \vdots \\ \vdots \end{array} \quad (40)$$

$$\begin{array}{c} \pi \\ \vdots \end{array} \begin{array}{c} \vdots \\ \vdots \end{array} \begin{array}{|c|} \hline E \\ \hline \end{array} \begin{array}{c} \vdots \\ \vdots \end{array} \begin{array}{|c|} \hline \overline{Z}_1 \\ \hline \end{array} \begin{array}{c} \vdots \\ \vdots \end{array} = \begin{array}{c} \vdots \\ \vdots \end{array} \begin{array}{|c|} \hline E \\ \hline \end{array} \begin{array}{c} \vdots \\ \vdots \end{array} \quad (41)$$

In general, any logical operation in the code is some map L on the logical qubits, which is physically implemented by a map P on

the physical qubits, where

$$\begin{array}{|c|} \hline \vdots \\ \hline L \\ \hline \vdots \\ \hline \end{array} = \begin{array}{|c|} \hline \vdots \\ \hline E \\ \hline \vdots \\ \hline P \\ \hline \vdots \\ \hline E^\dagger \\ \hline \vdots \\ \hline \end{array} \quad (42)$$

When $n > k$, there are many possible choices of P that implement the same L , because any operation in the stabilizer group of the code is identity on the logical qubits.

Equation (42) always holds if the following is satisfied, as E^\dagger can be appended to both sides of the equation below. We call this *pushing through the encoder* [HLY+23].

$$\begin{array}{|c|} \hline \vdots \\ \hline L \\ \hline \vdots \\ \hline \end{array} \begin{array}{|c|} \hline \vdots \\ \hline E \\ \hline \vdots \\ \hline \end{array} = \begin{array}{|c|} \hline \vdots \\ \hline E \\ \hline \vdots \\ \hline P \\ \hline \vdots \\ \hline \end{array} \quad (43)$$

In the other direction,

Lemma 44. Equation (42) implies Equation (43) only if P preserves the code space.

Proof. We can see this by appending E to both sides of Equation (42):

$$L = E; P; E^\dagger \implies L; E = E; P; \Pi \quad (45)$$

Let us compare $E; P; \Pi$ to the right hand side of Equation (43), which is $E; P$. By pre-composing by E^\dagger , these match when $\Pi; P = \Pi; P; \Pi$; in other words, when P preserves the code space. \square

Stabilizer codes admit a nice group-theoretic presentation. Any stabilizer code is equivalent to a *stabilizer group* \mathcal{S} , where $\mathcal{S} \subset \mathcal{P}_n$, the Pauli group on n qubits. The code space is the joint $+1$ eigenspace of \mathcal{S} .

The definition of the stabilizer group implies their commutation relations, due to a well-known lemma:

Lemma 46. Everything in the stabilizer group by definition must commute with all other elements in the stabilizer group.

Proof. This follows from a proof by contradiction: Any two Pauli strings either commute or anticommute. Presume two stabilizers \vec{P}_1, \vec{P}_2 anticommute. Then, their product applied to any state in the code space $|\psi\rangle$ cannot stabilize it, as

$$\vec{P}_1 \vec{P}_2 |\psi\rangle = -\vec{P}_2 \vec{P}_1 |\psi\rangle = -\vec{P}_2 |\psi\rangle = -|\psi\rangle \quad (47)$$

\square

Definition 48. The *centralizer* $\mathcal{C}(\mathcal{S})$ of a subset \mathcal{S} in a group G is the set of elements in G which commute with every element of \mathcal{S} .

For any stabilizer code defined by the stabilizer group \mathcal{S} , all logical operators of the code make up $\mathcal{C}(\mathcal{S}) \setminus \mathcal{S}$, as all logical operators commute with all stabilizers of the code, but not with all other logical operators. For instance, the physical implementations of Pauli X and Z on the same logical qubit by definition must anticommute with each other.

All detectable errors of the code make up $\mathcal{P}_n \setminus \mathcal{C}(\mathcal{S})$, as these are all the Pauli operators which do not commute with at least one stabilizer of the code, and are hence detectable as at least one stabilizer generator must measure -1 . All non-trivial undetectable errors of the code must be logical operators; trivial errors are physical errors that are equal to a stabilizer and hence does not cause a logical error.

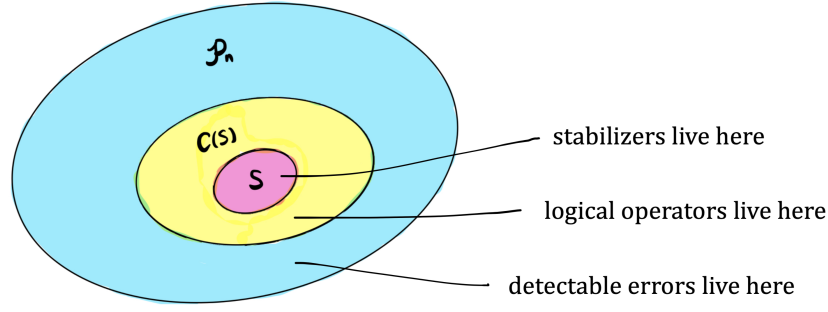


Figure 1: For any stabilizer code, the associated stabilizer group \mathcal{S} , its centralizer $\mathcal{C}(\mathcal{S})$, and the Pauli group on n qubits \mathcal{P}_n satisfy the relation $\mathcal{S} \subset \mathcal{C}(\mathcal{S}) \subset \mathcal{P}_n$. The logical operators of the code make up $\mathcal{C}(\mathcal{S}) \setminus \mathcal{S}$, and the detectable errors of the code make up $\mathcal{P}_n \setminus \mathcal{C}(\mathcal{S})$.

5 CSS codes with the phase-free ZX-calculus

The ZX-calculus is complete for the universal fragment of qubit quantum mechanics. Moreover, the ZX-calculus is complete for the *stabilizer fragment* (i.e. diagrams whose phases are all integer multiples of $\frac{\pi}{2}$) of qubit quantum mechanics [Bac14]. For now, we focus on CSS codes, a class of stabilizer codes with very nice properties [Ste96; CS96]. Precisely,

Definition 49. A CSS code is a stabilizer code whose stabilizer generators can be chosen in such a way, that all stabilizer generators are either over $\{X, I\}^{\otimes n}$ (called *X-type stabilizer generators*) or $\{Z, I\}^{\otimes n}$ (called *Z-type stabilizer generators*).

CSS codes are easy to represent in the ZX-calculus because their syndrome measurements are in the phase-free fragment of the ZX-calculus, i.e. diagrams whose phases are all zero [Kis22]. To start, we can observe that the m -qubit GHZ state (which is the Choi state of any phase-free Z spider) is a sum of two terms in the computational basis:

$$\begin{array}{c} \text{Z spider} \\ \vdots \end{array} = \sum_{j=0}^1 \begin{array}{c} j\pi \\ \vdots \\ j\pi \end{array} = \begin{array}{c} \bullet \\ \vdots \\ \bullet \end{array} + \begin{array}{c} \pi \\ \vdots \\ \pi \end{array} = |0\rangle^{\otimes m} + |1\rangle^{\otimes m} \quad (50)$$

Flipping the colors, we can expand the $H^{\otimes m} |\text{GHZ}\rangle$ state (the Choi state of any phase-free X spider) as a sum of two terms in the X basis:

$$\begin{array}{c} \text{X spider} \\ \vdots \end{array} = \sum_{j=0}^1 \begin{array}{c} j\pi \\ \vdots \\ j\pi \end{array} = \begin{array}{c} \bullet \\ \vdots \\ \bullet \end{array} + \begin{array}{c} \pi \\ \vdots \\ \pi \end{array} = |+\rangle^{\otimes m} + |-\rangle^{\otimes m} \quad (51)$$

Example 52. Consider the leftmost ZX diagram below, whose three Z spiders are respectively identified with the binary vectors

$$\vec{x} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \quad \vec{y} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \quad \vec{z} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} \quad (53)$$

$$\begin{array}{c} \vec{x} \\ \vec{y} \\ \vec{z} \end{array} \begin{array}{c} \text{Z spider} \\ \vdots \end{array} = \sum_{x=0}^1 \sum_{y=0}^1 \sum_{z=0}^1 \begin{array}{c} x\pi \\ x\pi \\ y\pi \\ y\pi \\ z\pi \\ z\pi \end{array} = \sum_{x,y,z=0}^1 \begin{array}{c} (x+z)\pi \\ (x+z)\pi \\ (x+y)\pi \\ (x+y)\pi \\ (y+z)\pi \end{array} = \sum_{x,y,z=0}^1 |x \oplus z\rangle |x \oplus z\rangle |x \oplus y\rangle |y \oplus z\rangle \quad (54)$$

The \vec{x} , \vec{y} , and \vec{z} labels are not necessary in the diagram, and exist simply to denote that the binary vector \vec{x} is equal to the connectivity of that Z spider to the output qubits.

In addition to considering our original diagram expanded in the Z-basis, we can also consider how it acts as parity checks on

X-basis states.

$$(55)$$

In the end, we get three scalars which are each nonzero only when they are of even parity. This is because 0-input, 0-output spiders are the scalar 0 if their phase (modulo 2π) is π , shown below for Z spiders, but the same holds for X spiders.

$$\bigcirc = \bigcirc - \bigcirc = (\sqrt{2}\langle + |)(\sqrt{2}| + \rangle) = 2$$

$$\bigcirc^\pi = \bigcirc - \bigcirc^\pi = (\sqrt{2}\langle - |)(\sqrt{2}| + \rangle) = 0$$

In other words, we have here a system of linear equations, the solutions to which are length 4 bitstrings $q_1 q_2 q_3 q_4$. These correspond to those X-basis states $H^{\otimes 4} |q_1 q_2 q_3 q_4\rangle$ which are not annihilated by the diagram.

$$H_X \vec{q} = \begin{bmatrix} \vec{x}^T \\ \vec{y}^T \\ \vec{z}^T \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \pmod{2} \quad (56)$$

H_X is called the *parity check matrix*, where each row is a binary vector identified with an X-type stabilizer generator.

In fact, observe that the above system of equations is linearly dependent, and so only two of the equations are necessary; the third is redundant. Graphical proof of this is derivable by starting with the bialgebra rule [Kis22]. This enables replacing the biadjacency connectivity of a spider, by a sum of existing such connectivities, as one would do with systems of linear equations. For example, we can add the connectivities denoted by \vec{x} and \vec{y} , to get a connectivity corresponding to $x + y$.

$$(57)$$

We can further show that consecutive diagrams representing the same stabilizer generator are redundant, which follows from $P^2 = P$ for any projector P .

$$(58)$$

Substituting the above two equations, we can reduce our earlier system of equations.

$$(59)$$

The code space of any CSS code expressed in either Z or X basis vectors, is each identifiable with a state in the phase-free ZX-calculus. For instance, the state $|\psi\rangle$ above is identifiable with the code space spanned by $\{XXXXI, IIXXX\}$.

(60)

We can see that this (contrived toy example) code indeed detects phase-flip errors. Say that a phase-flip error occurs on the third qubit between two rounds of syndrome extraction. Presuming the syndrome measurements were not faulty, consider the two consecutive measurements of the $\{XXXXI\}$ stabilizer.

(61)

Utilizing the fact that $ZX = -XZ$, we can proceed with the derivation:

(62)

What we found is that our original diagram, is equal to our original diagram times -1 . How can this be? This can only be the case if the original diagram was equal to zero in the first place. Therefore, the probability of measuring $+1$ for the $\{XXXXI\}$ stabilizer twice in a row, if between these measurements a phase-flip error occurred on the third qubit, is zero.

Remark 63. A helpful notation to streamline proofs like the above is to utilize Pauli webs and detecting regions. The basics are simple: For any spider, you can highlight all of its edges with the opposite color as that spider. You can also highlight any even number of its edges with the same color as that spider. To exemplify these two rules on a Z spider:

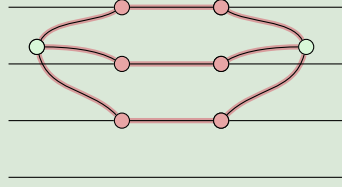
(64)

The name of the game is to find valid highlights, such that for every highlighted edge, both of their vertices (spiders) are consistent with these two rules.

Returning to our earlier example, we invoke these two rules for each spider of the diagram which is part of the $XXXXI$ syndrome measurement.

(65)

The resulting highlighted detecting region is:



(66)

This is a detecting region because it is a Pauli web where no input nor output edges are highlighted.

The previous section covered an example of Z-type stabilizers, which protect against X (i.e. bit-flip) errors. In this section, we looked at X-type stabilizers, which protect against Z (i.e. phase-flip) errors.

CSS codes must protect against both Z and X errors in order to protect against errors projected to the Pauli group.

We can now introduce stabilizer tableaux for CSS codes represented by phase-free ZX diagrams. First, we give the general definition of stabilizer tableaux.

Definition 67. A stabilizer tableau of a stabilizer code is a binary matrix with m rows and $2n$ columns. Each row represents a stabilizer generator, where the left n columns denote which of the n qubits are acted on by Pauli X, and the right n columns denote which of the n qubits are acted on by Pauli Z. As $Y = XZ$ up to a global phase, qubits acted on by Pauli Y have 1s in that qubit's X position and Z position.

For example, the 4-qubit Pauli string $IXYZ$ is represented by 0110|0011.

Remark 68. While the common convention is to take the sign of all stabilizers to be $+1$, more generally, if some of their eigenvalues are -1 , these are indicated by 1s in an additional column of the stabilizer tableau.

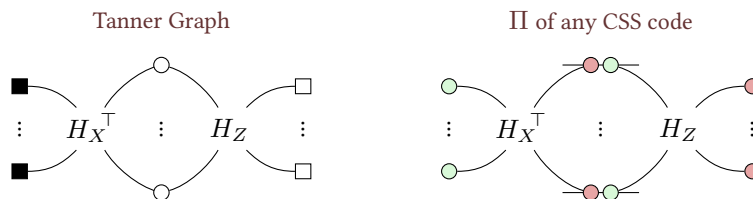
Although not usually used in QEC, note that for some applications such as simulation, it is advantageous to also have twice as many rows to keep track of *destabilizer generators*—a basis of m Pauli strings which anticommute with the stabilizer group, marked by a 1 in an additional column of the tableau [AG04].

Definition 69. For any CSS code, its *stabilizer tableau* can be factored

$$T = \left(\begin{array}{c|c} H_X & 0 \\ \hline 0 & H_Z \end{array} \right) \quad (70)$$

Equivalent to binary matrices, Tanner graphs are widely used for representing classical and CSS codes. A Tanner graph of a CSS code consists of two biadjacency graphs, whose connectivities are precisely H_X^\top and H_Z . This is endowed with additional utility in the ZX-calculus because

Observation 71. The projector onto the code space Π is representable by a ZX diagram of precisely the same connectivity as the Tanner graph.



(72)

Definition 73. For any stabilizer code, its *complete stabilizer tableau* consists of a stabilizer tableau for the code, consisting of $n - k$ rows for the stabilizer generators plus $2k$ rows for the logical operators. For CSS codes, the $n - k$ stabilizer generator rows are factorizable into the above form.

We will also need this definition of a transversal logical operation of a code.

Definition 74. A *transversal* logical operation in a code, is one that does not entangle any physical qubits in the same code block.

6 The Phase-free Z-X Generalized Parity Form (GPF)

In this section, we will introduce the Z-X Generalized Parity Forms. They are the normal forms for phase-free ZX calculus diagrams for an encoder of any CSS code. X-Z Generalized Parity Forms are defined by exchanging the roles of X and Z spiders.

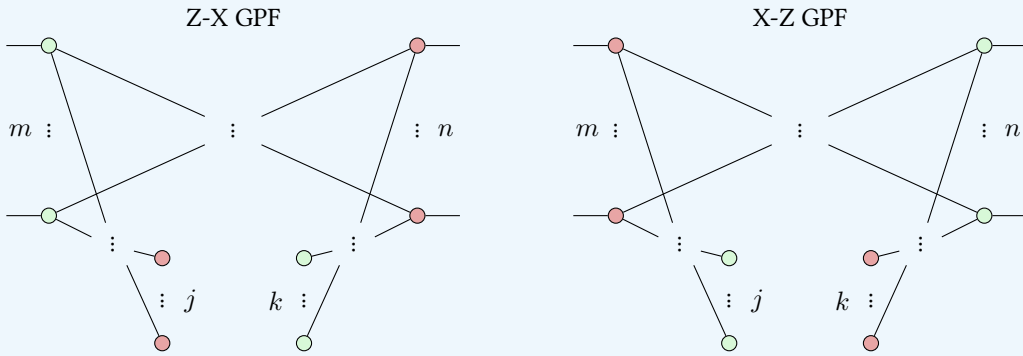
Definition 75. We say a spider is an input spider (or output spider) when it is connected to at least one input (or output).

Any phase-free ZX diagram can be written into Z-X generalized parity form.

Definition 76. We say a phase-free ZX-diagram is in Z-X generalized parity form (GPF) when it is two-coloured and

1. every input is connected to a Z-spider and every output to an X-spider,
2. no two spiders of the same colour are connected,
3. there are no zero-arity (i.e. scalar) spiders, and
4. no non-input Z-spider is connected to a non-output X-spider.

The same phase-free ZX-diagram can be written in X-Z GPF by exchanging the roles of Z and X spiders above. Drawing the above conditions, we get that any phase-free ZX diagram in GPF must look like:



Remark 77. A special case of GPF is for states, i.e. $m = 0$ and $j = 0$.

Remark 78. Another special case of GPF is when the linear map is unitary, i.e. $m = n$, $j = k = 0$ (by the next lemma below), and the biadjacency matrix B is invertible. This corresponds to CNOT circuits, which we will later show can always be extracted from any GPF diagram which is an isometry.

Lemma 79. All Z-X (or X-Z) GPF phase-free ZX diagrams which are isometries, must have $j = 0$.

Proof. This can be concluded through proof by contradiction: Assume that $j \neq 0$. Each of these j red spiders is a post-selection, which annihilates all computational basis state inputs for which that parity is nonzero. However, this contradicts the diagram being an isometry, as isometries cannot annihilate any input states. Therefore, $j = 0$. \square

Theorem 80. Any CSS code admits an encoder map which can be efficiently rewritten to Z-X and X-Z GPF, whose connectivity is in bijection with its complete stabilizer tableau.

Without loss of generality, note that this is a possible choice of encoder for the code, and all other encoders are equivalent to this one up to a k -qubit unitary on the logical qubits. We are making an (initial) choice of logical operators to be either Z-type or X-type. We know from the encoder-from-stabilizer-tableau extraction algorithm that we will present later, that this is always possible. To generalize to any choice of encoder, we can then allow any unitary to be applied on the logical qubits of the chosen phase-free encoder.

Proof. To go from a pair of Z-X and X-Z GPF diagrams for the same isometry to a complete stabilizer tableau, we can simply read off the connectivities. To prove that this indeed is the correct stabilizer tableau, recall from earlier that the logical operators are determined by the connectivity of the input spiders by π -copy, and likewise the stabilizer generators are also determined by their connectivity to the output spiders. Either diagram of the pair fixes the encoder map, so there cannot be more rows of the complete stabilizer tableau than can be written down from these two diagrams' connectivities.

To go from a complete stabilizer tableau of a CSS code where all rows are either Z-type or X-type, to a pair of Z-X and X-Z GPF diagrams for the same isometry, we directly write down the encoder map. How to do this for Z-X GPF is:

1. Draw $k + m_X$ Z spiders on the left, and n X spiders on the right, of a bipartite graph.
2. Draw an output wire for all n X spiders.
3. Draw an input wire to the first k Z spiders, and connect each of them to the X spiders which are supported by $|\overline{Z}_i\rangle$ for $i \in [1, k]$.
4. Connect each of the remaining m_X Z spiders to the X spiders which are supported by $|S_i^X\rangle$ for $i \in [1, m_X]$.

Repeating this procedure but exchanging the roles of Z and X spiders obtains X-Z GPF.

We can note several observations:

- A complete stabilizer tableau fixes an encoder map which must be an isometry.
- Every isometry can be put into both Z-X and X-Z GPF.
- The logical operators and stabilizer generators are bijective with the Z-X and X-Z GPF connectivities.
- We have the correct number of logical operators, which is k Z-type plus k X-type.
- No X-type stabilizer generators can be produced from linear combinations of the Z half of the tableau (as all their X columns are zero), so there are no missing X-type stabilizer generators in the resulting Z-X GPF diagram, and vice versa.

□

This brings us to a definition of a pair of Z-X and X-Z GPF diagrams, the combination of which enables one to read off a complete stabilizer tableau:

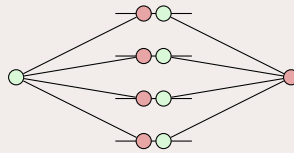
Definition 81. We call a pair of Z-X and X-Z GPF diagrams for the same encoder map, a *presentation* of the code, corresponding to a choice of complete stabilizer tableau in accordance with [Theorem 80](#).

Example 82 (The $[[4, 2, 2]]$ code). The Tanner graph of the $[[4, 2, 2]]$ code, drawn as a ZX-diagram for the projector onto the codespace Π , is

Π of the $[[4, 2, 2]]$ code

$$H_X = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$

$$H_Z = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$



An encoder for this code is

$$\begin{array}{c} \text{Diagram 1: 4 input wires (green) connected to 4 output wires (red) via a network of spiders.} \end{array}
 =
 \begin{array}{c} \text{Diagram 2: 4 input wires (green) connected to 4 output wires (red) via a different network of spiders.} \end{array}
 =
 \begin{array}{c} \text{Diagram 3: A trapezoidal box labeled 'E' with 4 input and 4 output wires.} \end{array}
 =
 \begin{array}{c} \text{Diagram 4: 4 input wires (red) connected to 4 output wires (green) via a network of spiders.} \end{array}
 =
 \begin{array}{c} \text{Diagram 5: 4 input wires (red) connected to 4 output wires (green) via a different network of spiders.} \end{array}
 \quad (83)$$

For any phase-free ZX-diagram which is an isometry, we can show that the physical implementation of Pauli X (or Z) on any logical qubit, is X (or Z) on every physical qubit that that logical qubit's Z (or X) spider in Z-X (or X-Z) GPF is connected to.

As an example, we can show that $\overline{X}_1 = XXII$ by applying the π -copy rule:

(84)

More generally, any ZX-diagram can be *pushed through the encoder* [HLY+23]. This means that given a phase-free ZX-diagram encoder for any CSS code, and any logical map L as a ZX-diagram, we can find always find a ZX-diagram for the physical map P which implements L in that CSS code. This is because we can push through any X spider on the logical qubits by the below steps, to get a Z spider connected to X spiders on all physical qubits in that logical qubit's X support. As an example, in the $[[4, 2, 2]]$ code, for which $\overline{X}_1 = XXII$:

(85)

This can be done for any X spider on any logical qubit, by applying the same procedure but color-reversed. Repeating for each spider, we can push any ZX-diagram on logical qubits through any phase-free CSS code encoder.

As a self-dual code, the $[[4, 2, 2]]$ code has a logical operation whose physical implementation is a transversal Hadamard gate on all physical qubits simultaneously. For this choice of encoder, this logical operation is SWAP; $H \otimes H$:

(86)

7 The RREF_X Normal Form for Complete Stabilizer Tableau Rows

Proposition 87. For any CSS code^a, we can write down an encoder map for it in phase-free Z-X (or X-Z) GPF, such that its biadjacency matrix B is in RREF_X (or RREF_Z) form.

^apresuming some choice of logical operators such that all X logical operators are over $\{X, I\}$ and all Z logical operators are over $\{Z, I\}$

Definition 88. The RREF_X form writes all X stabilizer generator rows ($\{S_X\}$) and X logical operator rows ($\{\overline{X}\}$) of the complete stabilizer tableau of a CSS code (presuming some choice of logical operators such that all \overline{X} are over $\{X, I\}$ and all \overline{Z} are over $\{Z, I\}$) as a binary matrix through the following steps:

1. Reduce $\{S_X\}$ to be in Reduced Row Echelon Form (RREF).
2. Subtract rows of $\{S_X\}$ from $\{\overline{X}\}$, to zero in $\{\overline{X}\}$ all columns which are pivots of $\{S_X\}$.
3. Reduce $\{\overline{X}\}$ to be in RREF.

The resulting binary matrix is of the form below, where the horizontal lines can be any bitstring.

X-type stabilizer generators in RREF

X logical operators in RREF

identity matrix	the rest of RREF
all zeroes	RREF

Likewise, the RREF_Z form can be obtained by applying the above procedure to the Z stabilizer generator rows ($\{S_Z\}$) and Z logical operator rows ($\{\bar{Z}\}$).

Remark 89. This restriction to only X logical operators over $\{X, I\}$ and Z logical operators over $\{Z, I\}$, is nevertheless applicable to any CSS code. This is because a CSS code fixes the *image* of the encoder map, but not the encoder map itself. In fact, changing the choice of logical operators does not change the code distance.

In choosing to presume X logical operators over $\{X, I\}$ and Z logical operators over $\{Z, I\}$, CSS code encoder maps correspond precisely to isometries in the phase-free ZX calculus. Moreover, writing such an encoder map as a state by Choi-Jamiołkowski isomorphism preserves it being a CSS code — to be exact, a $k = 0$ CSS code. This perspective on CSS codes lends itself naturally to techniques developed in the study of the phase-free fragment of the ZX calculus.

We can apply this reasoning to phase-free encoders, to all encoders for CSS codes, by virtue of the fact that any choice of logical operator is always unitarily equal to a phase-free choice of logical operators of the form above.

This is enforced by the property that any quantum error correcting code fixes the image of any encoder map E , such that E^\dagger then E must equal the projector collectively defined by each stabilizer generator.

$$\begin{array}{c} \vdots \\ \text{---} \Pi_{S_i \in S} S_i \text{---} \\ \vdots \end{array} = \begin{array}{c} \vdots \\ \text{---} \Pi \text{---} \\ \vdots \end{array} = \begin{array}{c} \vdots \\ \text{---} E^\dagger \text{---} \vdots \\ \vdots \end{array} \quad (90)$$

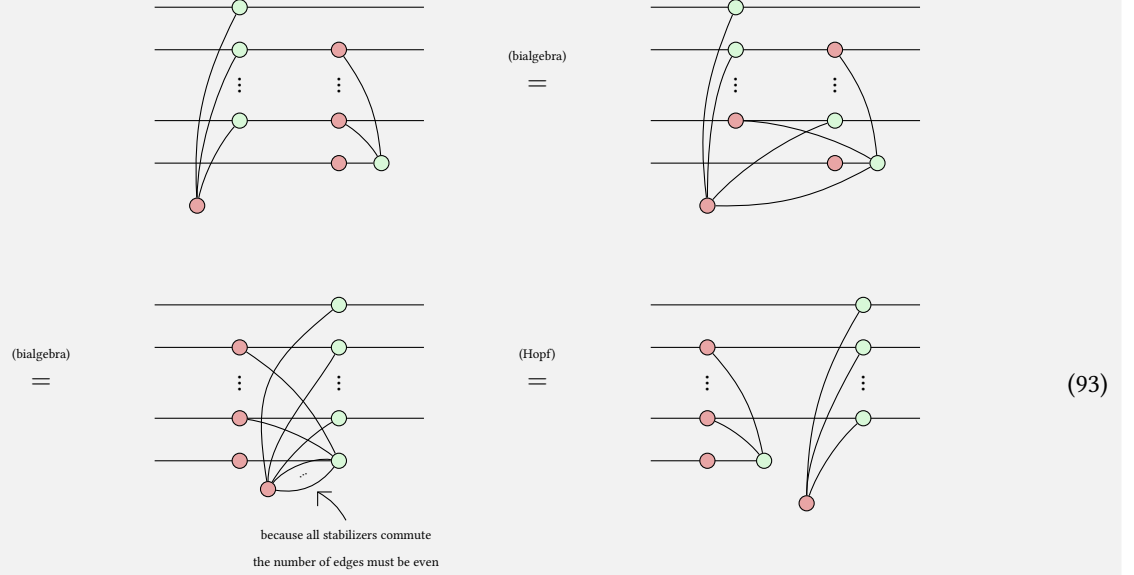
Therefore,

Proposition 91. *Given an incomplete stabilizer tableau for any CSS code, i.e. a complete set of stabilizer generators without specifying the logical operators, we can extract a phase-free encoder map in Z-X GPF. This implies that there always exists a choice of logical operators for any CSS code, such that all X logical operators are over $\{X, I\}$ and all Z logical operators are over $\{Z, I\}$.*

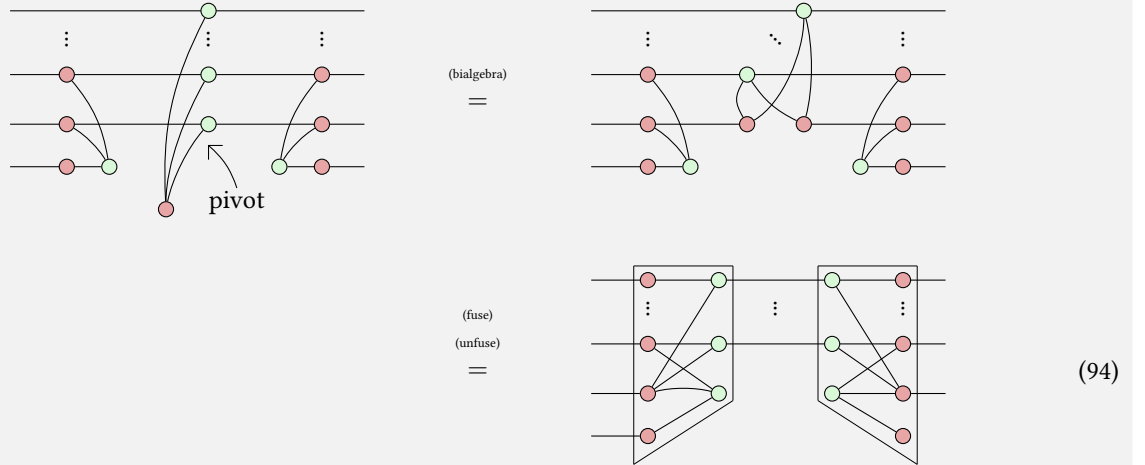
Proof. 1. For each X -type stabilizer generator, apply the bialgebra rule about the spider corresponding to its pivot column in the RREF_X stabilizer tableau.

$$\begin{array}{c} \text{---} \text{red circle} \text{---} \\ \vdots \\ \text{---} \text{red circle} \text{---} \\ \vdots \\ \text{---} \text{red circle} \text{---} \\ \vdots \\ \text{---} \text{green circle} \text{---} \end{array} \xrightarrow{\text{(bialgebra)}} \begin{array}{c} \text{---} \text{red circle} \text{---} \\ \vdots \\ \text{---} \text{red circle} \text{---} \\ \vdots \\ \text{---} \text{red circle} \text{---} \\ \vdots \\ \text{---} \text{green circle} \text{---} \end{array} \xrightarrow{\text{(unfuse) (id)}} \begin{array}{c} \text{---} \text{red circle} \text{---} \\ \vdots \\ \text{---} \text{red circle} \text{---} \\ \vdots \\ \text{---} \text{red circle} \text{---} \\ \vdots \\ \text{---} \text{green circle} \text{---} \end{array} \quad (92)$$

2. Commute all Z-type stabilizer generators to the middle, past all the X-type stabilizer generators.



3. For each Z-type stabilizer generators, apply the bialgebra rule about its pivot in RREF_Z . At the end, fuse all X and Z spiders, then unfuse all middle Z spiders to form E^\dagger on the left and E on the right.



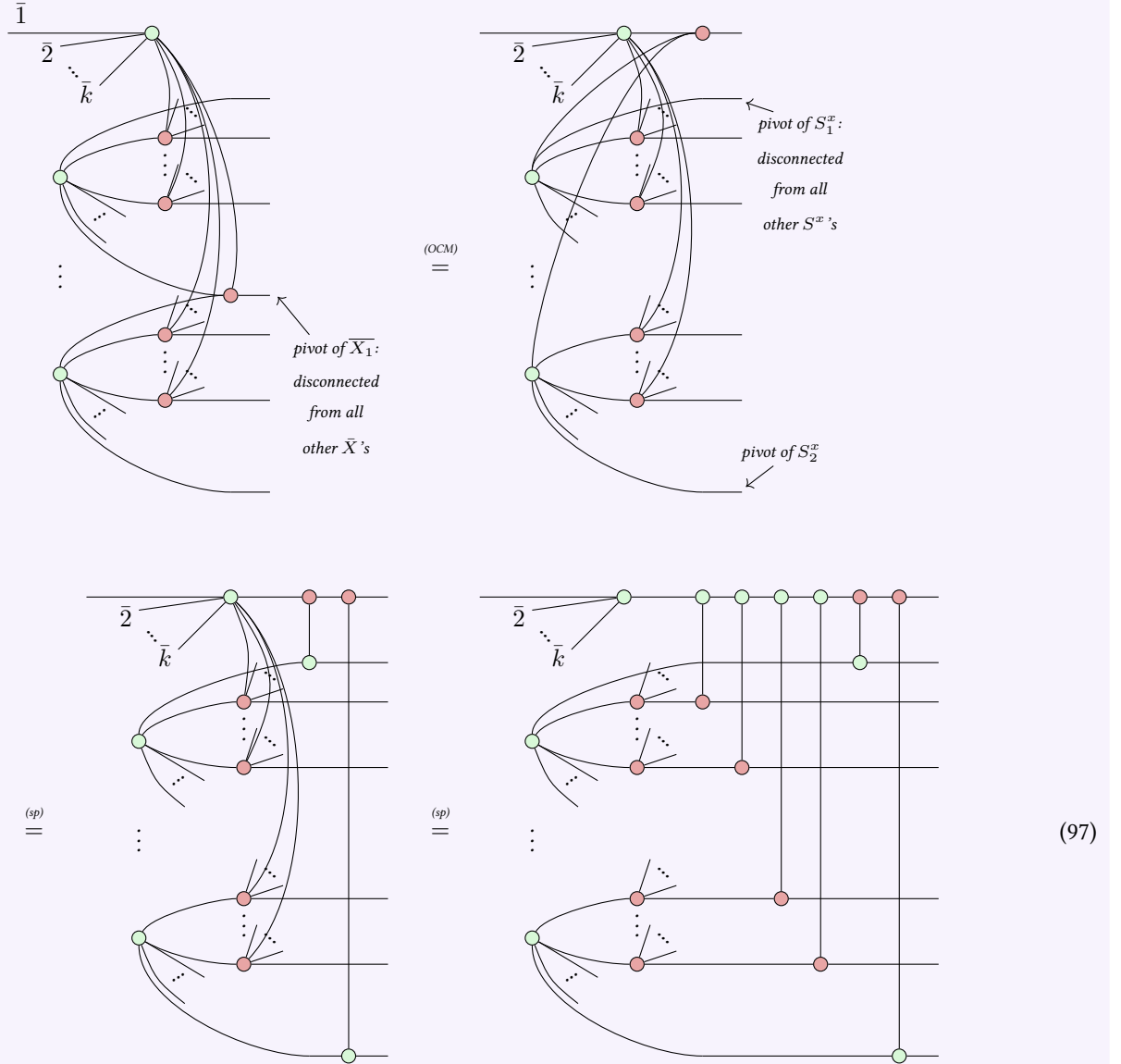
The total number of applications of the bialgebra rule, each of which reduces the number of wires in the middle by one, is equal to the number of stabilizer generators, which is $n - k$. Therefore, the total number of wires in the middle at the end is $n - (n - k) = k$, which is indeed the number of logical qubits. \square

Corollary 95. In the diagrams above, it can be seen that the stabilizer generators fix the image of E , and moreover that there are $k = n - s$ wires in the middle, for k logical qubits. To preserve the equality of Equation 90, the only modifications that can be done to the n input or n output wires of EE^\dagger , are those that preserve its equality to \mathbb{I} . These modify the encoder by prepending the phase-free E by a unitary U , whilst modifying the dagger of the encoder by appending the phase-free E^\dagger by U^\dagger .

8 Circuit Extraction

CSS code encoder maps must be *unitary embeddings*. This means that we can always write them as phase-free ZX unitaries (which are equivalent to CNOT circuits) applied to a tensor product of identity wires, $|0\rangle$ states, and $|+\rangle$ states. To derive this, we present an explicit *circuit extraction* algorithm:

Proposition 96. For any CSS code encoder in phase-free Z-X (or X-Z) GPF, we can always extract a CNOT circuit from it. First, the Z-X GPF must be in $RREF_X$ (or X-Z GPF in $RREF_Z$) form; if not, it can be efficiently rewritten into this form by Gaussian elimination as described in the previous section. The key step of the circuit extraction is given below.



Here, the pivot of each X logical operator is moved as to be on the same horizontal qubit wire as the Z spider for that logical qubit, by permuting the order the outputs are drawn. Following that, CX gates can be unfused for all other X spiders connecting to that Z spider.

The same can be done to unfuse CX gates from each X -type stabilizer generator, such that the control of each new CX gate is on the pivot wire of that stabilizer generator.

The full algorithm iterates the above for each logical operator and for each stabilizer generator. The end result is a CNOT circuit acting upon identity wires, and ancillae initialized to $|0\rangle$ and $|+\rangle$.

Regarding the complexity of this algorithm, reducing the stabilizer tableau rows to $RREF_X$ can be done efficiently, such as by Gaussian elimination. The extraction itself is also efficient because it consists of k initial Only-Connectivity-Matters rewrites, followed by $b - k - g$ CX gate unfusions. Here k is the number of logical qubits of the code, b is the number of edges in the Z-X GPF biadjacency graph, and g is the number of X -type stabilizer generators.

The Scalable ZX-calculus

Next, we introduce the *scalable ZX-calculus* [CHP19]. As is common in quantum circuit notation, rather than having one qubit per wire there can be multiple qubits per wire. We keep the presentation lightweight as we do not need the more heavy lifting capabilities of the scalable ZX-calculus—we just need it to streamline two parts in this tutorial: scalable normal forms for CSS code encoders which is next, and a graphical proof that any CX gate in any CSS code is implementable transversally which is later.

The scalable notation allows representing a register of qubits with a single wire. A bold wire labelled k is a register of k qubits.

$$k \text{ ————— } := \overline{k \begin{matrix} \vdots \\ \vdots \end{matrix}} \quad (98)$$

To easily regroup the registers of qubits, the divide and gather nodes are defined as follows:

$$\begin{array}{ccc} a+b \text{ ————— } \begin{array}{c} \diagup \\ a \\ \diagdown \\ b \end{array} & := & \begin{array}{c} a \text{ ————— } \\ \vdots \\ a \text{ ————— } \\ b \text{ ————— } \\ \vdots \\ b \text{ ————— } \end{array} \end{array} \quad \begin{array}{ccc} \begin{array}{c} a \text{ ————— } \\ \vdots \\ a \text{ ————— } \\ b \text{ ————— } \\ \vdots \\ b \text{ ————— } \end{array} \text{ ————— } a+b & := & \begin{array}{c} a \text{ ————— } \\ \vdots \\ a \text{ ————— } \\ b \text{ ————— } \\ \vdots \\ b \text{ ————— } \end{array} \end{array} \quad (99)$$

Where it is clear from context, we might omit the divide and gather nodes for brevity. The spiders also have scalable versions, now labelled with a vector of phases $\vec{\alpha} = (\alpha_1, \dots, \alpha_k) \in \mathbb{R}^k$.

$$\begin{array}{ccc} \begin{array}{c} \diagup \\ \vdots \\ \alpha \\ \vdots \\ \diagdown \end{array} & := & \begin{array}{c} \begin{array}{c} \vdots \\ \vdots \end{array} \text{ ————— } \begin{array}{c} \alpha_1 \\ \vdots \\ \alpha_k \end{array} \text{ ————— } \begin{array}{c} \vdots \\ \vdots \end{array} \end{array} \end{array} \quad \begin{array}{ccc} \begin{array}{c} \diagup \\ \vdots \\ \alpha \\ \vdots \\ \diagdown \end{array} & := & \begin{array}{c} \begin{array}{c} \vdots \\ \vdots \end{array} \text{ ————— } \begin{array}{c} \alpha_1 \\ \vdots \\ \alpha_k \end{array} \text{ ————— } \begin{array}{c} \vdots \\ \vdots \end{array} \end{array} \end{array} \quad (100)$$

It is easy to see that all the rewrite rules of the ZX-calculus extend to the scalable setting. In addition, the scalable ZX-calculus introduces a generator called the matrix arrow. It allows us to compactly represent a bipartite graph of n input Z spiders connected to m output X spiders. Suppose $A \in \mathbb{F}_2^{m \times n}$ is the biadjacency matrix of such a graph, i.e. $A_{ij} = 1$ if and only if the i th Z spider is connected to the j th X spider. Then the matrix arrow is defined as

$$n \text{ ————— } \xrightarrow{A} m \quad := \quad \begin{array}{c} \begin{array}{c} \vdots \\ \vdots \end{array} \text{ ————— } \begin{array}{c} \alpha_1 \\ \vdots \\ \alpha_k \end{array} \text{ ————— } \begin{array}{c} \vdots \\ \vdots \end{array} \end{array} \xrightarrow{[\cdot]} \sum_{\vec{x} \in \mathbb{F}_2^n} |A\vec{x}\rangle \langle \vec{x}| \quad (101)$$

where $|\vec{x}\rangle$, $\vec{x} \in \mathbb{F}_2^n$ is a computational basis state. As a convention, we omit the matrix arrow label when it is the all-ones matrix. With the matrix arrows, we have additional rewrite rules. Composing matrix arrows multiplies their matrices:

$$\xrightarrow{A} \xrightarrow{B} = \xrightarrow{BA} \quad (102)$$

We can copy matrix arrows through the spiders as shown below:

$$\begin{array}{ccc} \xrightarrow{A} \begin{array}{c} \diagup \\ \vdots \\ \alpha \\ \vdots \\ \diagdown \end{array} & = & \begin{array}{c} \begin{array}{c} \vdots \\ \vdots \end{array} \text{ ————— } \begin{array}{c} \alpha_1 \\ \vdots \\ \alpha_k \end{array} \text{ ————— } \begin{array}{c} \vdots \\ \vdots \end{array} \end{array} \end{array} \quad \begin{array}{ccc} \begin{array}{c} \begin{array}{c} \vdots \\ \vdots \end{array} \text{ ————— } \begin{array}{c} \alpha_1 \\ \vdots \\ \alpha_k \end{array} \text{ ————— } \begin{array}{c} \vdots \\ \vdots \end{array} \end{array} \text{ ————— } \xrightarrow{A} & = & \begin{array}{c} \begin{array}{c} \vdots \\ \vdots \end{array} \text{ ————— } \begin{array}{c} \alpha_1 \\ \vdots \\ \alpha_k \end{array} \text{ ————— } \begin{array}{c} \vdots \\ \vdots \end{array} \end{array} \end{array} \quad (103)$$

This lets us express this corollary of Proposition 96:

Corollary 104. *We can leverage the Scalable ZX-calculus to concisely represent phase-free isometries. Any CNOT circuit can be represented as a map in \mathbb{F}_2 by a reversible binary matrix. Therefore, any CSS code encoder can be drawn in the scalable ZX*

calculus in two different ways—as its Z-X or X-Z GPF biadjacency connectivity [KvdW24], or as a CNOT circuit unitary embedding:

$$\begin{array}{c} \vdots \\ \text{---} \end{array} \left(\begin{array}{c} \vdots \\ E \\ \vdots \end{array} \right) \text{---} = \begin{array}{c} \text{---} \text{---} \\ \text{---} \text{---} \end{array} \begin{array}{c} L_x \\ H_x \end{array} \begin{array}{c} \text{---} \\ \text{---} \end{array} = \begin{array}{c} \text{---} \text{---} \\ \text{---} \text{---} \end{array} \begin{array}{c} L_z \\ H_z \end{array} \begin{array}{c} \text{---} \\ \text{---} \end{array} = \begin{array}{c} \text{---} \\ \text{---} \end{array} \left(\begin{array}{c} \text{---} \\ R \\ \text{---} \end{array} \right) \text{---} \quad (105)$$

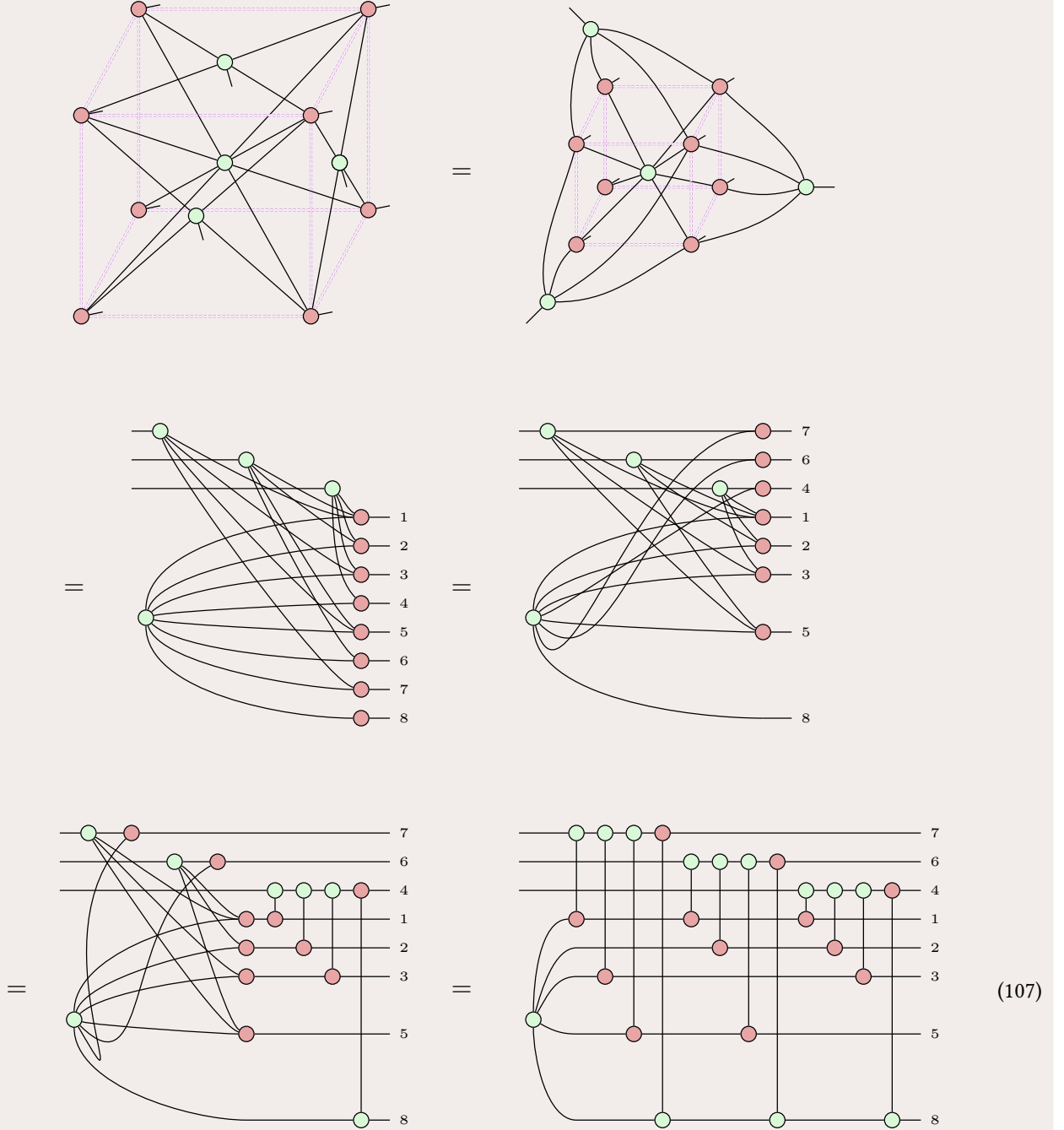
This uses the fact that an arbitrary CNOT circuit can be drawn in the Scalable ZX calculus, as a matrix arrow representing the phase-free ZX parity map, here labelled by the reversible binary function R .

We will shortly find certain scalable ZX calculus rewrite rules essential to succinctly deriving transversality of CX gates for all CSS codes.

First, for an example with more than one logical qubit:

Example 106. Let us apply the above procedure to the $[[8, 3, 2]]$ code encoder. This is commonly drawn as a cube; to illustrate this cube shape, we have superimposed a cube in pink on top of the ZX diagram of its encoder map. The X logical operators

are on three independent faces of the cube, while the weight 8 X-type stabilizer generator is in the volume of the cube.



From here on, the 5-qubit GHZ state can be further decomposed in the typical way: into 4 CNOT gates, all controlled on the same $|+\rangle$ state, and each with a different target qubit in the $|0\rangle$ state.

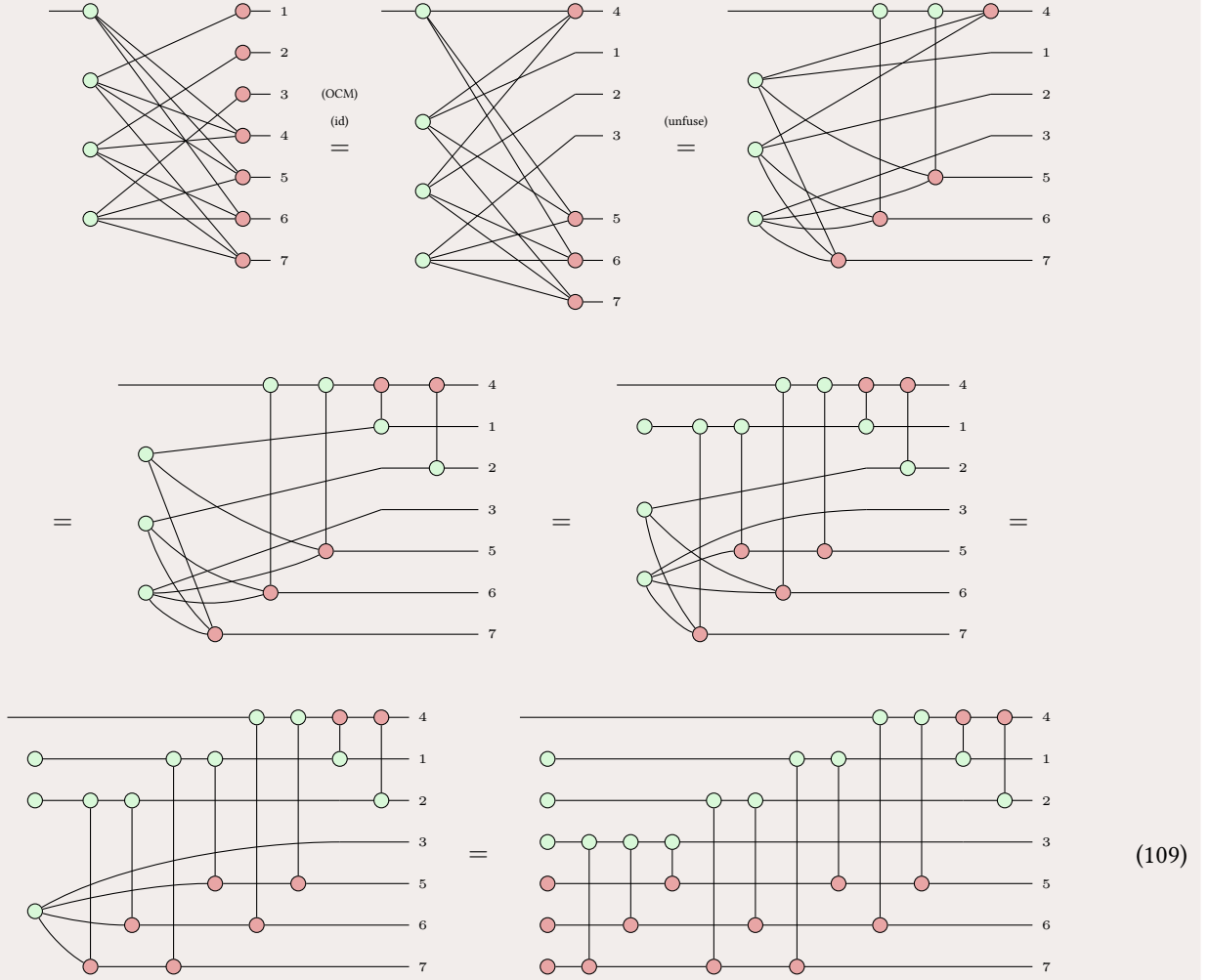
To do an example with more than one X-type stabilizer generator:

Example 108. Let us apply the above circuit extraction procedure to the $[[7, 1, 3]]$ Steane code encoder. This circuit extraction of the $[[7, 1, 3]]$ code from Z-X GPF, has been previously done through a combination of automated rewrites in the Quantomatic

software and human guided proof [DL14]. Its RREF_X matrix is

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

where the first three rows are for the X-type stabilizer generators, and the last row is for the X logical operator. Starting with the Z-X GPF of the above matrix, the circuit extracts to:



9 Transversal Gates and Magic State Distillation

We previously mentioned that transversal operations are those that do not entangle any physical qubits in the same code block.

H gate

Proposition 110. *Any self-dual CSS code (i.e. $H_X = H_Z$) has a logical gate whose physical implementation is a transversal Hadamard gate on all physical qubits simultaneously.*

Proof. Recall a fact we used in Lemma 44, that $\Pi; P = \Pi; P; \Pi$ means that the physical operation P preserves the code space defined by the projector Π . We can see that $P = H^{\otimes n}$ indeed preserves the code space, because

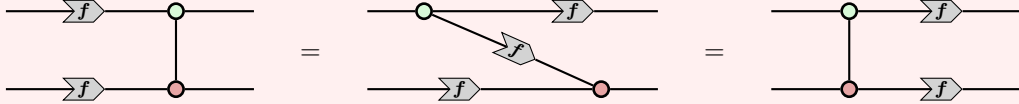
$$\Pi; H^{\otimes n}; \Pi = \Pi; \Pi; H^{\otimes n} = \Pi; H^{\otimes n} \quad (111)$$

The first equality follows from Observation 71, the color change rule, and the fact that $H_X = H_Z$. The second equality follows from $\Pi^2 = \Pi$ for any projector Π . \square

CX gate

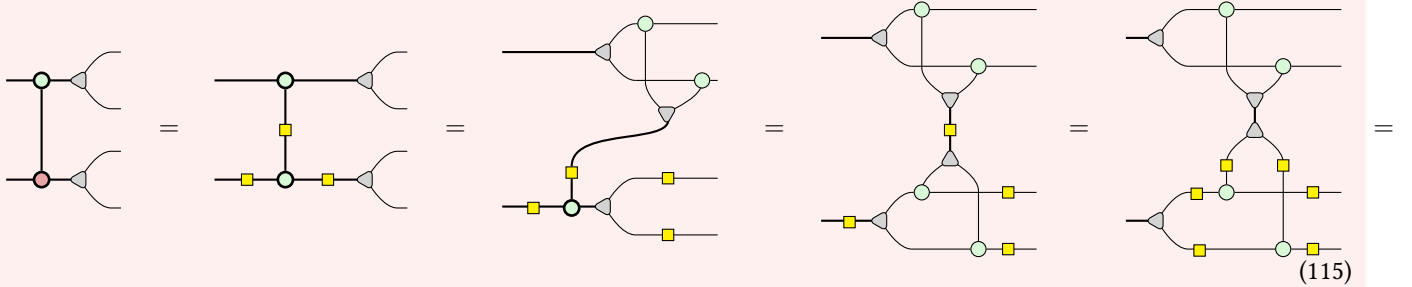
We would like to next prove transversality of any CX gate on any CSS code. In the scalable ZX-calculus, we can commute a pair of the same bijective matrix arrow i.e. equal to the same CNOT circuit, through the controls and the targets respectively of a ladder of transversal CX gates. (Here we switched the look of the divide and gather nodes, but they behave the same as in the overview of scalable ZX-calculus from before.)

Lemma 112. For any matrix arrow $f \in \mathbb{F}_2^{m \times m}$ that is both injective and surjective, i.e. is bijective,


(113)

We also need to commute CX gates with dividers and gatherers, to verify that they act as one would expect.

Lemma 114.


(114)

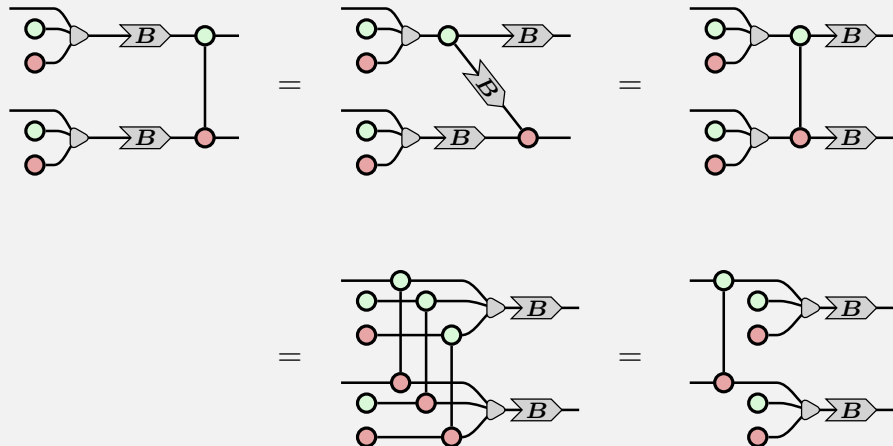
This can be applied iteratively, to show that this indeed implements a CX gate from the i th qubit on the top half to the i th qubit on the bottom half.

Lemma 116. For any CSS code, we can write transversal CX gates between two code blocks physically, is equal to transversal CX gates between two code blocks logically, i.e.

$$\overline{\Pi_{i=1}^k CX_i} = \Pi_{j=1}^n CX_j \quad (117)$$

where CX_i has control on the i th logical qubit of the top block and target on the i th logical qubit of the bottom block, and CX_j has control on the j th physical qubit of the top block and target on the j th physical qubit of the bottom block.

Proof. We apply Lemma 112 to an arbitrary CSS code encoder in the Scalable ZX calculus form of Equation 105.


(118)

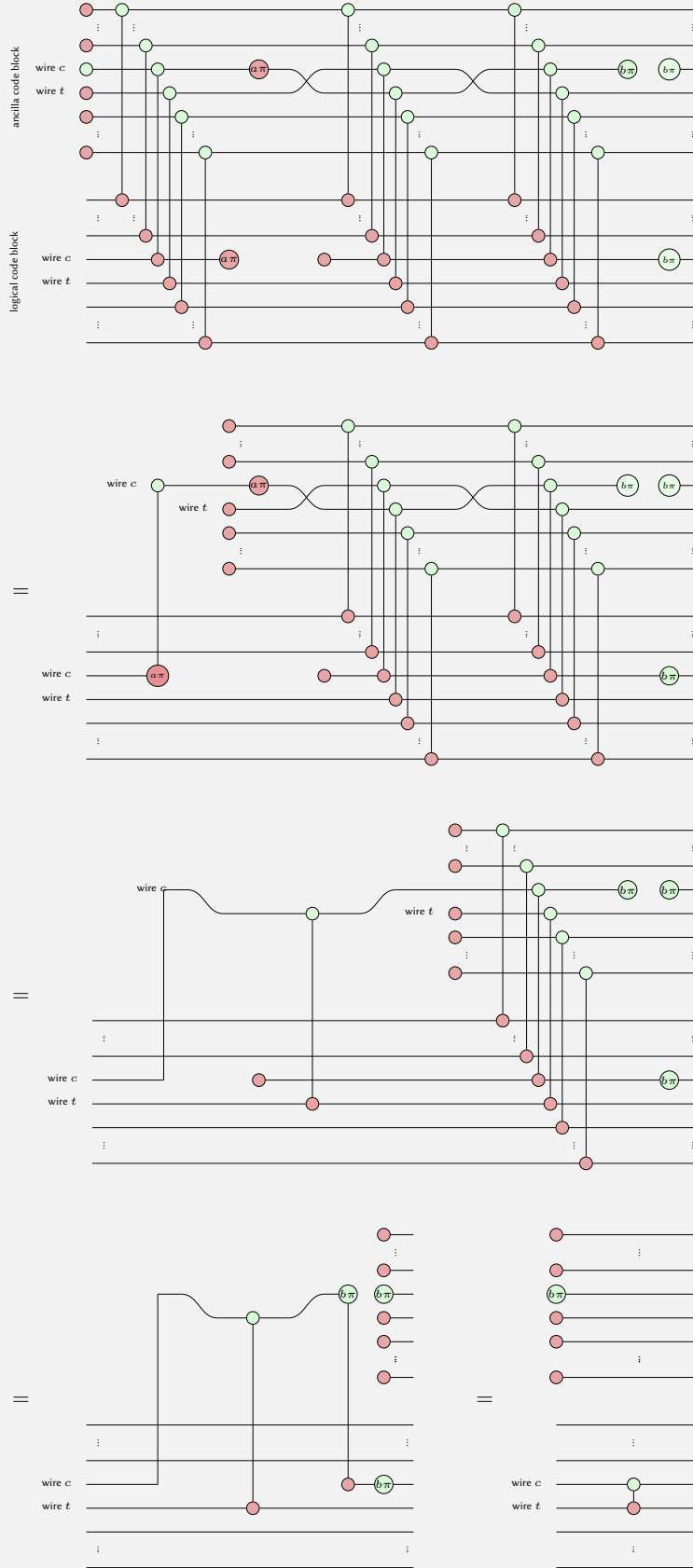
\square

Although the above proof suffices to do any logical CX gate transversally when $k = 1$, we furthermore present a graphical proof to extend this to $k > 1$.

Proposition 119. *Any logical CX gate on any CSS code can be implemented transversally.*

Proof. When $k = 1$, [Lemma 116](#) suffices. For when $k > 1$, we present below a proof entirely in the ZX calculus, of Gottesman’s gate teleportation protocol to implement a transversal logical CX gate between two logical qubits in the same code block [[Got97](#)]. This protocol uses an ancilla code block prepared in the logical state where all logical qubits are initialized to $|0\rangle$, except that the logical qubit having the desired CX gate’s control index c is initialized to $|+\rangle$. Note that below, classical feed-forward of

measurement outcomes labelled by $a, b \in \{0, 1\}$ is necessary for correction after each teleportation.



(120)

The proof proceeds near identically as above for the case where the transversal logical CX gate is between two different code blocks — the only difference, is that the middle ladder of CX gates instead goes from the ancilla code block to the desired CX

gate's target qubit's code block. The protocol still works because it essentially teleports the control qubit from its control code block to the ancilla code block, swaps it to the same index within the code block as that of the target qubit within its own code block, does the CX, then undoes the swap and the teleportation. \square

Diagonal gates

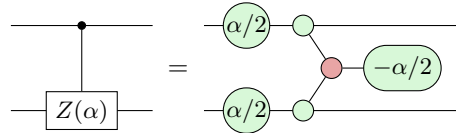
CSS codes with transversal diagonal gates in the third level of the Clifford hierarchy are called *triorthogonal* codes [BH12]. For such codes, H_X is a *triorthogonal* binary matrix, meaning that the Hamming weight of the product of any two rows, as well as the product of any three rows, is 0 mod 2. All such gates admitted by CSS codes were characterized in the scalable ZX-calculus in Ref. [KvdW24], which found that they correspond to *spider nest identities* [dBBW20].

Here, we will demonstrate the well-known example of the transversal CCZ gate of the $[[8, 3, 2]]$ code [Cam16] (also called the smallest interesting color code, an instance of transversal CCZ gates described in Ref. KubicaA2015colorcode), in the ZX-calculus. In addition to serving as an example to gain some intuition before we next introduce magic state distillation, the reason for being interested in this code specifically is that its connection to circuit synthesis will be the inspiration for the proposal at the end of the next chapter, for which we will revisit this example.

Consider the equation $(x - y)^2 = x^2 + y^2 - 2x \cdot y$. For bits x and y , as $x^2 = x$ for $x \in \{0, 1\}$, this equation reduces to $x \oplus y = x + y - 2x \cdot y$. It follows that

$$x \cdot y = \frac{1}{2}(x + y - (x \oplus y)). \quad (121)$$

Importantly, we are considering the $+$ operation here not modulo 2, but just as an action on real numbers, and we are writing \oplus for addition modulo 2. Using this relation we can write $e^{i\alpha(x \cdot y)} = e^{i\frac{1}{2}\alpha(x+y-(x \oplus y))} = e^{i\frac{1}{2}\alpha x} e^{i\frac{1}{2}\alpha y} e^{-i\frac{1}{2}\alpha(x \oplus y)}$. This is the reason the following circuit decomposition holds:



$$\text{Z}(\alpha) = \text{Circuit with two qubits and phase gates } \alpha/2 \text{ and } -\alpha/2. \quad (122)$$

This relation between additive and multiplicative phase gates follows from a Fourier-type duality that exists for semi-Boolean functions, which is explored in detail in Ref. [KvdWK19].

The phase polynomial representation of the CCZ gate is that it acts as $|x, y, z\rangle \mapsto (-1)^{x \cdot y \cdot z} |x, y, z\rangle$, where x, y, z are bits denoting computational basis states. The qubit CCZ gate is identified by the exponent of (-1) being a multiplicative expression $x \cdot y \cdot z$; we can recursively plug in Equation (121) into itself, to obtain an equivalent expression for $x \cdot y \cdot z$ in terms of XOR, additions, and subtractions of x, y , and z . For this reason,

Observation 123. The qubit CCZ gate is equal to the following spider nest of phase gadgets:

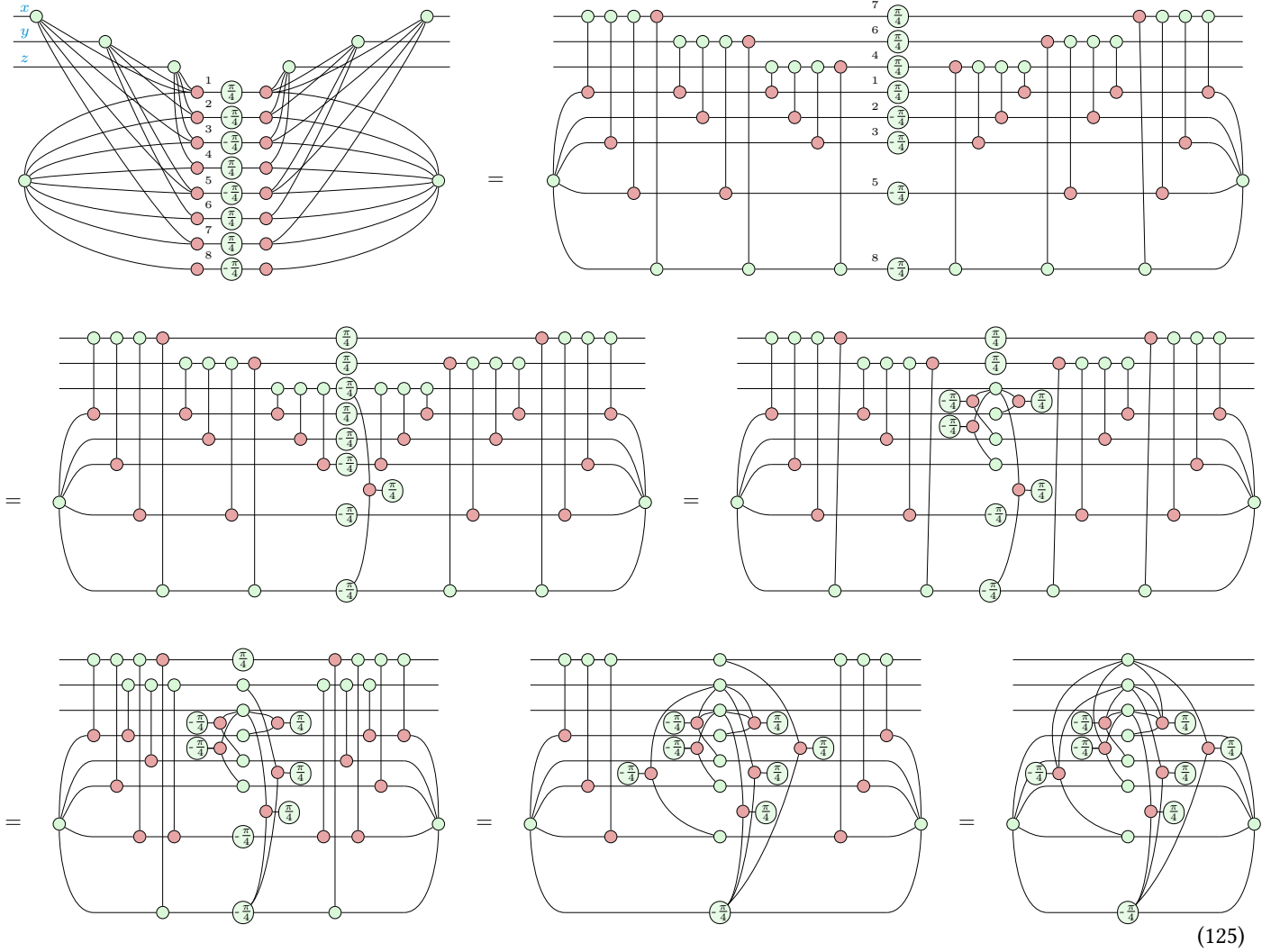


$$\text{CCZ} = \text{Circuit with phase gadgets } \pi/4 \text{ and controlled operations.} \quad (124)$$

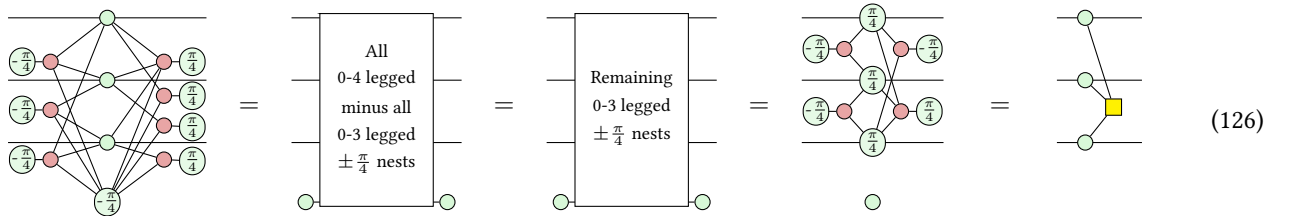
If instead all phases here are $\pm \frac{\pi}{2}$, then instead of the CCZ gate, this is the identity map on three qubits. More generally, for n qubits, having one of all possible connectivity phase gadgets of 0 - n legs, of phases $\pm \frac{\pi}{2^{n-2}}$ where the sign matches the parity of the number of legs, is equal to the n -qubit identity map.

We can substitute the encoder E of the $[[8, 3, 2]]$ code from Equation (107), to verify that $L = E; P; E^\dagger$ where L is the CCZ gate,

and $P = TT^\dagger T^\dagger TT^\dagger TTT^\dagger$.



Continuing from the last step, we now utilize the above observation to recognize that what we have is the spider nest of all 0-4 legged $\mp \frac{\pi}{4}$ phase gadgets, conjugated by all 0-3 legged $\pm \frac{\pi}{4}$ phase gadgets on the upper three qubits. Therefore, we can then apply all 0-4 legged $\mp \frac{\pi}{4}$ phase gadgets, which equals the identity. The end result is precisely the CCZ gate.



Magic state distillation

There is a no-go theorem by Eastin and Knill, which states that no one code can achieve a universal transversal gate set [EK09]. The most well-studied way to circumvent this issue is to use *magic state distillation* and injection [BK05]. Earlier, in Equation (18), we already saw the quantum circuit (at the logical level) for magic state injection of the T state in order to perform the T gate.

Now, we will give a graphical portrayal of magic state distillation. We have just derived that the logical CCZ gate in the $[[8, 3, 2]]$

code is physically implemented by 8 T and T^\dagger gates:

(127)

Taking the conjugate transpose of the above equation and inputting $|+\rangle^{\otimes 8}$, the ideal decoder acts on 8 T and T^\dagger magic states as:

(128)

Substituting in the Z-X GPF of the encoder, the $|+\rangle$ states copy and fuse, leaving just the three-legged H-box.

(129)

In other words, the decoding of 8 T and T^\dagger magic states results in a less noisy CCZ magic state.

10 Code switching

Instead of magic state distillation and injection, an alternative way to achieve universal, fault-tolerant quantum computation is to perform *code switching*. In code switching, you switch between two different quantum error correcting codes, combining the fault-tolerant logical gates of both codes to form a universal gate set.

Applying what we have gone through so far, we will show the most well-studied code switching protocol, which is between the $[[7, 1, 3]]$ code and the $[[15, 1, 3]]$ code [ADP14]. The $[[7, 1, 3]]$ code can do all Clifford gates transversally. On the other hand, the $[[15, 1, 3]]$ code can do the T gate transversally. Therefore, switching between these two codes enables doing the Clifford+ T gate set transversally, which is the most well-studied universal gate set for quantum computing.

First, we give an illustrative high-level overview of this code switching protocol.

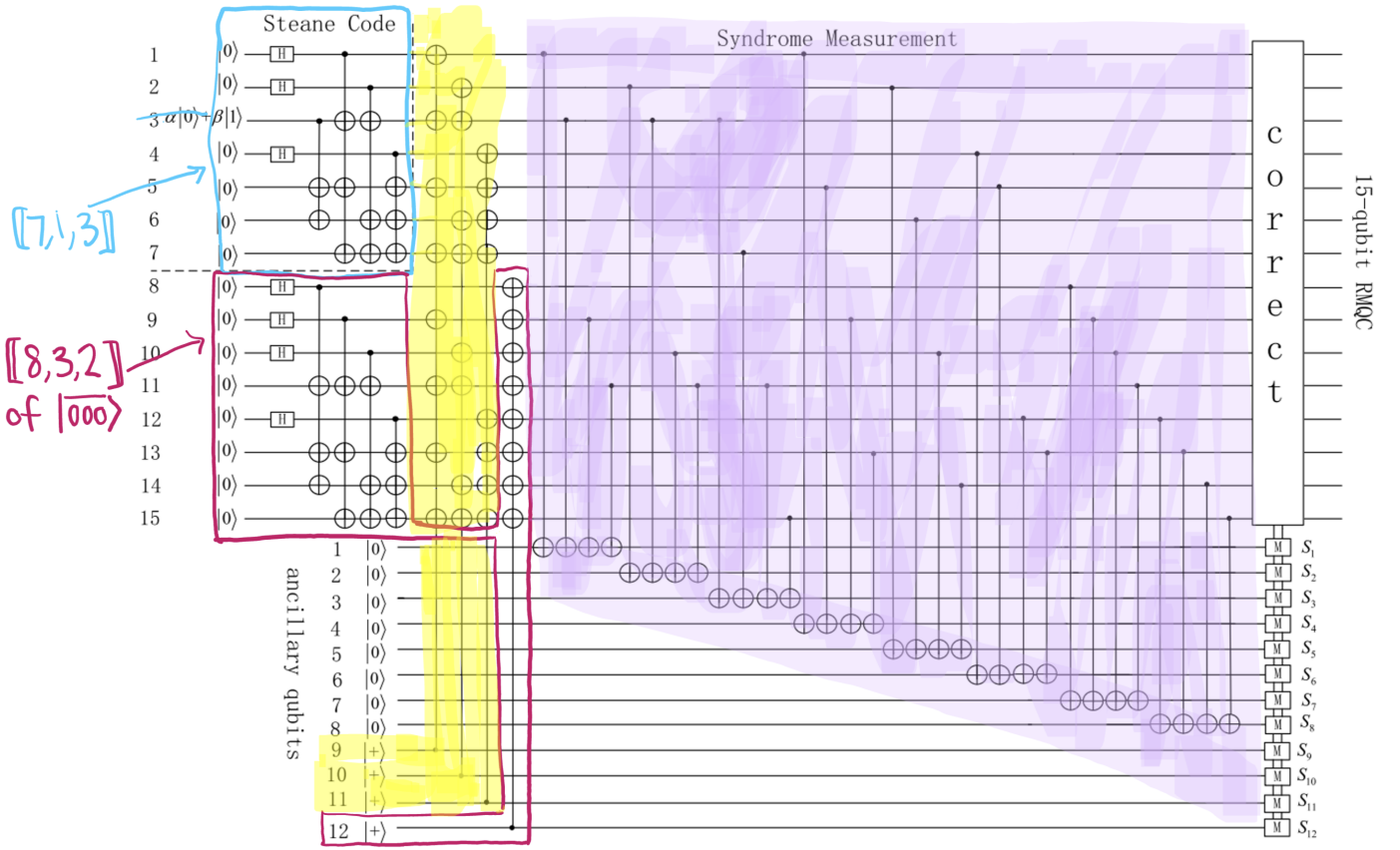


Figure 2: This circuit diagram reprinted from Ref. [QZPS18] is a non-fault-tolerant protocol for code switching from the $[[7, 1, 3]]$ code to the $[[15, 1, 3]]$ code. We have added highlighting and annotations to indicate the presence of smaller QEC codes playing a role in this code switching protocol. Note that as the four measurements to ancilla qubits 9 to 12 mutually commute, the three measurements highlighted in yellow can also be scheduled after the preparation of $|000\rangle$ in the $[[8, 3, 2]]$ code.

A geometric description of this observed in Ref. [WCAB15], is that this code switching protocol adds 3 X-type stabilizer generators, stitching thirds of the $[[7, 1, 3]]$ triangle onto 3 faces of the $[[8, 3, 2]]$ cube. This description can be made more mathematically concrete by drawing the encoders for both of these codes, then adding three weight 8 Z-type stabilizer generators by measuring them as in the yellow subcircuit of Figure 2. Z spider fusion of these three new stabilizers, fuses each Z-type stabilizer generator of the $[[7, 1, 3]]$ with a Z logical operator of the $[[8, 3, 2]]$ code. We can also connect to *code morphing*, a protocol by Vasmer and Kubica [VK22] where they discovered the $[[10, 1, 2]]$ code, the smallest known code that can distill T magic states.

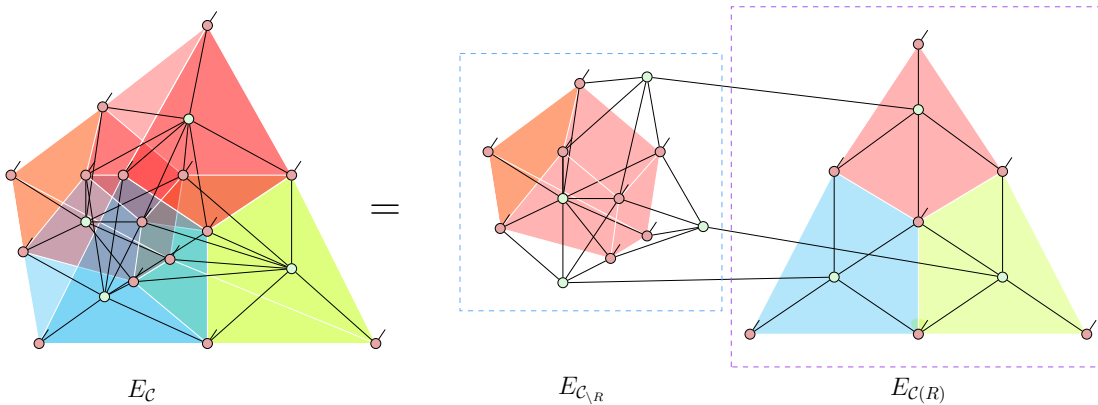
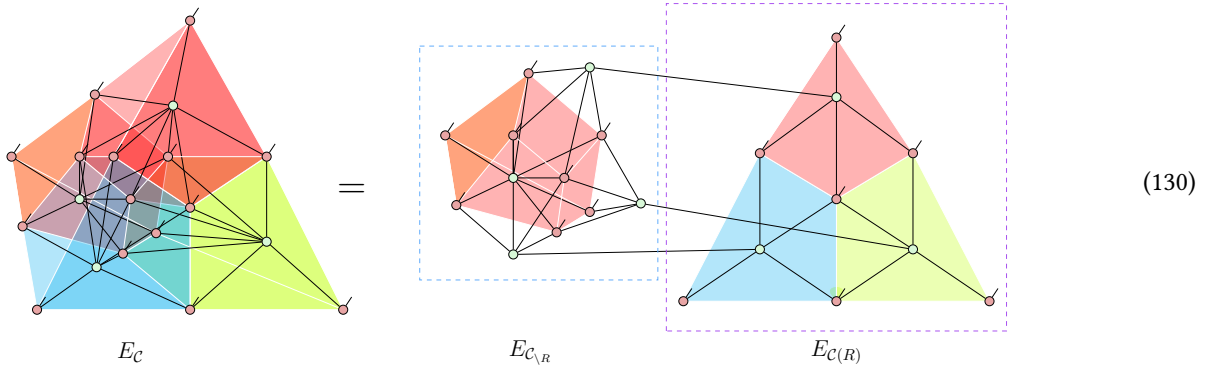


Figure 3: Figure with a minor modification from [HLY+23, Example 4.2]. This depicts in the form of Equation (60) the code morphing protocol from Ref. [VK22] of the $[[15, 1, 3]]$ code as the parent code C and the $[[8, 3, 2]]$ code as the child code $C(R)$, to obtain the morphed code $C \setminus R = [[10, 1, 2]]$. For brevity, X spiders (which are on each vertex of each shaded region) each representing a physical qubit, and Z spiders each representing how X on each logical qubit is encoded, are omitted.

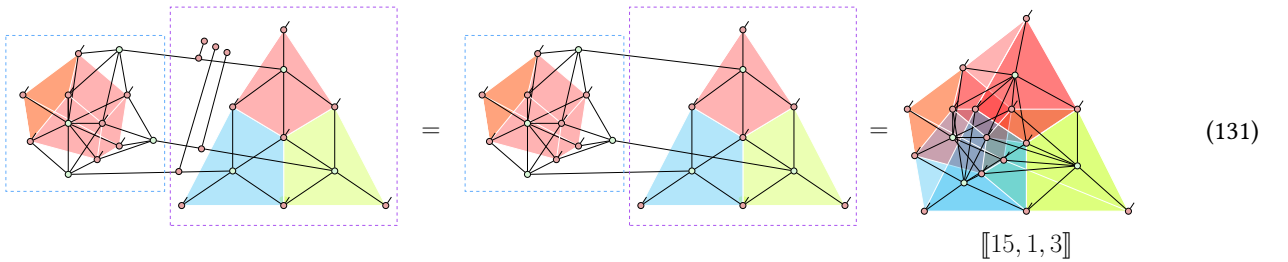
This draws a connection between this code morphing protocol [VK22], and code switching as an instance of *gauge fixing* of a subsystem code [VLC+19]. Performing physical multiqubit measurements changes the logical state $|\psi\rangle$ of the three *gauge logical qubits* of the $[[15, 1, 3, 3]]$ CSS subsystem code. This is a $[[15, 4, 3]]$ code where only one of the four logical qubits stores information, and the fault-tolerant operations that can be done on it are fixed by the state of the other three logical qubits, which are the gauge logical qubits. Measuring the gauge fixing operator L_g^Z fixes $|\psi\rangle$ to be $|000\rangle$ resulting in the $[[15, 1, 3]]$ code, and measuring the gauge fixing operator L_g^X fixes $|\psi\rangle$ to be $|+++\rangle$ resulting in the $[[7, 1, 3]]$ code.

$$\begin{array}{ccc}
 & \text{Measure } (L_g^Z) \nearrow & \llbracket 15, 1, 3 \rrbracket \\
 \text{Measure } (L_g^Z) \nearrow & & \\
 |\Psi\rangle = |0\rangle^{\otimes 3} & & \\
 \llbracket 15, 4, 3 \rrbracket & & \\
 \text{Measure } (L_g^X) \searrow & & \\
 |\Psi\rangle = |+\rangle^{\otimes 3} \searrow & & \llbracket 7, 1, 3 \rrbracket
 \end{array}$$

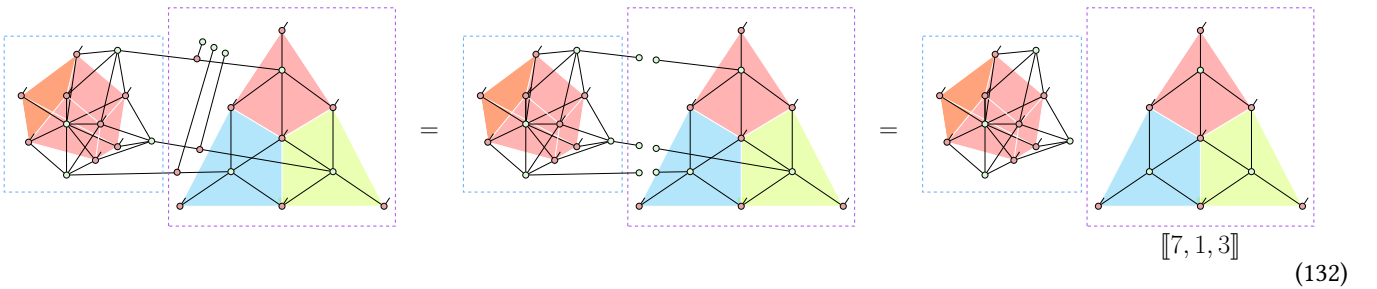
In the $[[15, 1, 3, 3]]$ code, the gauge logical qubit state $|\psi\rangle$ is at



Gauge fixing $|\psi\rangle = |000\rangle$ indeed fuses so the X spiders are identities, resulting in the $[[15, 1, 3]]$ code encoder.



On the other hand, gauge fixing $|\psi\rangle = |+++\rangle$ indeed copies across the X spiders and fuses into the $[[10, 1, 2]]$ code encoder, resulting in the $[[7, 1, 3]]$ code encoder.



References

- [ADP14] Jonas T. Anderson, Guillaume Duclos-Cianci, and David Poulin. “Fault-Tolerant Conversion between the Steane and Reed-Muller Quantum Codes”. In: *Phys. Rev. Lett.* 113 (8 Aug. 2014), p. 080501. DOI: [10.1103/PhysRevLett.113.080501](https://doi.org/10.1103/PhysRevLett.113.080501).
- [AG04] Scott Aaronson and Daniel Gottesman. “Improved simulation of stabilizer circuits”. In: *Physical Review A* 70.5 (2004), p. 052328.
- [Bac14] Miriam Backens. “The ZX-calculus is complete for stabilizer quantum mechanics”. In: *New Journal of Physics* 16.9 (Sept. 2014), p. 093021. DOI: [10.1088/1367-2630/16/9/093021](https://doi.org/10.1088/1367-2630/16/9/093021).
- [BH12] Sergey Bravyi and Jeongwan Haah. “Magic-state distillation with low overhead”. In: *Phys. Rev. A* 86 (5 Nov. 2012), p. 052329. DOI: [10.1103/PhysRevA.86.052329](https://doi.org/10.1103/PhysRevA.86.052329).
- [BK05] Sergey Bravyi and Alexei Kitaev. “Universal quantum computation with ideal Clifford gates and noisy ancillas”. In: *Phys. Rev. A* 71 (2 Feb. 2005), p. 022316. DOI: [10.1103/PhysRevA.71.022316](https://doi.org/10.1103/PhysRevA.71.022316).
- [Bom16] Héctor Bombín. “Dimensional jump in quantum error correction”. In: *New Journal of Physics* 18.4 (Apr. 2016), p. 043038. DOI: [10.1088/1367-2630/18/4/043038](https://doi.org/10.1088/1367-2630/18/4/043038).
- [CAB12] Earl T. Campbell, Hussain Anwar, and Dan E. Browne. “Magic-State Distillation in All Prime Dimensions Using Quantum Reed-Muller Codes”. In: *Phys. Rev. X* 2 (4 Dec. 2012), p. 041021. DOI: [10.1103/PhysRevX.2.041021](https://doi.org/10.1103/PhysRevX.2.041021).
- [Cam16] Earl T. Campbell. *The Smallest Interesting Colour Code*. <https://earltcampbell.com/2016/09/26/the-smallest-interesting-colour-code/>. Accessed: 2025-04-26. Aug. 2016.
- [CGK17] Shawn X. Cui, Daniel Gottesman, and Anirudh Krishna. “Diagonal gates in the Clifford hierarchy”. In: *Phys. Rev. A* 95 (1 Jan. 2017), p. 012329. DOI: [10.1103/PhysRevA.95.012329](https://doi.org/10.1103/PhysRevA.95.012329).
- [CHP19] Titouan Carlette, Dominic Horsman, and Simon Perdrix. “SZX-Calculus: Scalable Graphical Quantum Reasoning”. In: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. DOI: [10.4230/LIPICS.MFCS.2019.55](https://doi.org/10.4230/LIPICS.MFCS.2019.55).
- [CS96] A. R. Calderbank and Peter W. Shor. “Good quantum error-correcting codes exist”. In: *Physical Review A* 54.2 (Aug. 1996), pp. 1098–1105. ISSN: 1094-1622. DOI: [10.1103/physreva.54.1098](https://doi.org/10.1103/physreva.54.1098).
- [dBBW20] Niel de Beaudrap, Xiaoning Bian, and Quanlong Wang. “Fast and Effective Techniques for T-Count Reduction via Spider Nest Identities”. In: *15th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2020)*. Ed. by Steven T. Flammia. Vol. 158. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020, 11:1–11:23. ISBN: 978-3-95977-146-7. DOI: [10.4230/LIPICS.TQC.2020.11](https://doi.org/10.4230/LIPICS.TQC.2020.11).
- [DL14] Ross Duncan and Maxime Lucas. “Verifying the Steane code with Quantomatic”. In: *Electronic Proceedings in Theoretical Computer Science* 171 (Dec. 2014), pp. 33–49. ISSN: 2075-2180. DOI: [10.4204/eptcs.171.4](https://doi.org/10.4204/eptcs.171.4).
- [EK09] Bryan Eastin and Emanuel Knill. “Restrictions on Transversal Encoded Quantum Gate Sets”. In: *Physical Review Letters* 102.11 (Mar. 2009). ISSN: 1079-7114. DOI: [10.1103/physrevlett.102.110502](https://doi.org/10.1103/physrevlett.102.110502).
- [Got97] Daniel Gottesman. “Stabilizer codes and quantum error correction”. PhD thesis. California Institute of Technology, 1997. DOI: [10.48550/arXiv.quant-ph/9705052](https://doi.org/10.48550/arXiv.quant-ph/9705052).
- [Got98a] Daniel Gottesman. “The Heisenberg Representation of Quantum Computers”. In: *Proc. XXII International Colloquium on Group Theoretical Methods in Physics*. Ed. by Stuart P. Corney, Robert Delbourgo, and Peter D. Jarvis. GROUP22. Hobart, Australia: International Press of Boston, 1998, pp. 32–43. ISBN: 9781571460547.
- [Got98b] Daniel Gottesman. “Theory of fault-tolerant quantum computation”. In: *Phys. Rev. A* 57 (1 Jan. 1998), pp. 127–137. DOI: [10.1103/PhysRevA.57.127](https://doi.org/10.1103/PhysRevA.57.127).
- [Got99] Daniel Gottesman. “Fault-Tolerant Quantum Computation with Higher-Dimensional Systems”. In: *Chaos, Solitons & Fractals* 10.10 (Sept. 1999), pp. 1749–1758. ISSN: 0960-0779. DOI: [10.1016/s0960-0779\(98\)00218-5](https://doi.org/10.1016/s0960-0779(98)00218-5).
- [GSJ24] Craig Gidney, Noah Shutty, and Cody Jones. *Magic state cultivation: growing T states as cheap as CNOT gates*. 2024. arXiv: [2409.17595](https://arxiv.org/abs/2409.17595) [quant-ph].
- [HLY+23] Jiaxin Huang, Sarah Meng Li, Lia Yeh, Aleks Kissinger, Michele Mosca, and Michael Vasmer. “Graphical CSS Code Transformation Using ZX Calculus”. In: *Electronic Proceedings in Theoretical Computer Science* 384 (Aug. 2023), pp. 1–19. ISSN: 2075-2180. DOI: [10.4204/eptcs.384.1](https://doi.org/10.4204/eptcs.384.1).
- [KB15] Aleksander Kubica and Michael E. Beverland. “Universal transversal gates with color codes: A simplified approach”. In: *Phys. Rev. A* 91 (3 Mar. 2015), p. 032330. DOI: [10.1103/PhysRevA.91.032330](https://doi.org/10.1103/PhysRevA.91.032330).

- [Kis22] Aleks Kissinger. *Phase-free ZX diagrams are CSS codes (...or how to graphically grok the surface code)*. 2022. arXiv: [2204.14038 \[quant-ph\]](#).
- [KvdW24] Aleks Kissinger and John van de Wetering. “Scalable Spider Nests (...Or How to Graphically Grok Transversal Non-Clifford Gates)”. In: *Electronic Proceedings in Theoretical Computer Science* 406 (Aug. 2024), pp. 79–95. ISSN: 2075-2180. DOI: [10.4204/eptcs.406.4](#).
- [KvdWK19] Stach Kuijpers, John van de Wetering, and Aleks Kissinger. *Graphical Fourier Theory and the Cost of Quantum Addition*. 2019. arXiv: [1904.07551 \[quant-ph\]](#).
- [PRTC20] Tefjol Pllaha, Narayanan Rengaswamy, Olav Tirkkonen, and Robert Calderbank. “Un-Weyl-ing the Clifford Hierarchy”. In: *Quantum* 4 (Dec. 2020), p. 370. ISSN: 2521-327X. DOI: [10.22331/q-2020-12-11-370](#).
- [QZPS18] Dong-Xiao Quan, Li-Li Zhu, Chang-Xing Pei, and Barry C Sanders. “Fault-tolerant conversion between adjacent Reed–Muller quantum codes based on gauge fixing”. In: *Journal of Physics A: Mathematical and Theoretical* 51.11 (Feb. 2018), p. 115305. ISSN: 1751-8121. DOI: [10.1088/1751-8121/aaad13](#).
- [RCP19] Narayanan Rengaswamy, Robert Calderbank, and Henry D. Pfister. “Unifying the Clifford hierarchy via symmetric matrices over rings”. In: *Phys. Rev. A* 100 (2 Aug. 2019), p. 022304. DOI: [10.1103/PhysRevA.100.022304](#).
- [Ste96] A. M. Steane. “Error Correcting Codes in Quantum Theory”. In: *Phys. Rev. Lett.* 77 (5 July 1996), pp. 793–797. DOI: [10.1103/PhysRevLett.77.793](#).
- [VK22] Michael Vasmer and Aleksander Kubica. “Morphing Quantum Codes”. In: *PRX Quantum* 3.3 (Aug. 2022). ISSN: 2691-3399. DOI: [10.1103/prxquantum.3.030319](#).
- [VLC+19] Christophe Vuillot, Lingling Lao, Ben Criger, Carmen García Almudéver, Koen Bertels, and Barbara M Terhal. “Code deformation and lattice surgery are gauge fixing”. In: *New Journal of Physics* 21.3 (Mar. 2019), p. 033028. ISSN: 1367-2630. DOI: [10.1088/1367-2630/ab0199](#).
- [WCAB15] Fern H. E. Watson, Earl T. Campbell, Hussain Anwar, and Dan E. Browne. “Qudit color codes and gauge color codes in all spatial dimensions”. In: *Phys. Rev. A* 92 (2 Aug. 2015), p. 022312. DOI: [10.1103/PhysRevA.92.022312](#).