CSS code surgery as a universal construction

Alexander Cowtan[1,2] and Simon Burton[2]

[1]Dept. of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford OX1 3QD, UK
[2]Quantinuum, Terrington House, 13-15 Hills Road, Cambridge CB2 1NL, UK

arXiv:2301.13738v6 [quant-ph] 7 May 2024

We define code maps between Calderbank-Shor-Steane (CSS) codes using maps between chain complexes, and describe code surgery between such codes using a specific colimit in the category of chain complexes. As well as describing a surgery operation, this gives a general recipe for new codes. As an application we describe how to 'merge' and 'split' along a shared $\overline{X}$ or $\overline{Z}$ operator between arbitrary CSS codes in an error-corrected manner, so long as certain technical conditions concerning gauge fixing and code distance are satisfied. We prove that such merges and splits on LDPC codes yield codes which are themselves LDPC.

1 Introduction

Quantum computers have become larger and more sophisticated in recent years [1, 13], but fault-tolerance is necessary to perform many practically relevant quantum algorithms. Qubit stabiliser error-correction codes are a well-studied approach to fault-tolerant quantum computing [23] and are favourable both for their practicality and theoretical simplicity. Such codes store logical data using entangled states of physical qubits and repeated many-body measurements, and so long as the physical errors on the qubits stay below a certain threshold the logical data is protected.

The most well-known example of a qubit stabiliser code is the toric code, in which qubits are embedded on the surface of a torus, and properties of the logical space are determined by the topology of the surface [18, 31]. This is a basic example of a qubit Calderbank-Shor-Steane (CSS) code; there are several equivalent ways of defining CSS codes, but for our purposes we shall describe them as codes which are all *homological* in a suitable sense [5, 7].

This means that we can study CSS codes using the tools of homological algebra [45]. This approach has recently seen much success, for example in the construction of so-called good quantum low-density parity check (LDPC) code families using a lifted product of chain complexes [40]. Such code families have an encoding rate k/n of logical to physical qubits which is constant in the code size, while maintaining a linear code distance d, a substantial asymptotic improvement over simpler examples such as the toric code. The main caveat is, informally, that the connectivity between physical qubits is non-local. This complicates the architecture of the system, and also complicates the protocols for performing logical gates.

Alexander Cowtan: akcowtan@gmail.com
Simon Burton: simon.burton@quantinuum.com

There have been several recent works on protocols for logical gates in CSS codes [32, 12, 9, 42, 27], of varying generality. Here, we build on this work by defining surgery, in the abstract, using arbitrary CSS codes which form a categorical span, although the practical implementation of such surgery has several important caveats. The idea is that merging two codes works by identifying a common structure in each code and quotienting it out. CSS code surgery is particularly convenient when the CSS codes are *compatible*, in the sense that they have at least one identical $\overline{Z}$ or $\overline{X}$ logical operator. In this case, the common structure being quotiented out is the logical operator. In order to formalise this, we take a step back and look at the category of chain complexes $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$.

We start by giving a recap of the relevant categorical background of chain complexes, and the view of classical linear binary codes and qubit CSS codes using chain complexes. We then define code maps between CSS codes using morphisms between chain complexes. These are maps which send X-checks to X-checks and Z-checks to Z-checks in a coherent way, and have a convenient presentation as phase-free ZX diagrams, which we prove in Proposition 4.11.

We believe that code maps crop up throughout the CSS code literature. We see 3 primary use-cases for code maps:

1. Encoders/decoders [19, 17, 25].
2. Constructing new codes.
3. Designing fault-tolerant logical operations.

We intend to expound on code maps in future work, but presently we focus on items 2 and 3.

We define CSS code merges as a colimit – specifically, a coequaliser/pushout – in the category of chain complexes. Not only does the construction describe a surgery operation, but it also gives a general recipe for new codes. An application of our treatment is the description of certain classes of code surgery whereby the codes are merged or split along a $\overline{Z}$ or $\overline{X}$ operator. This is closely related to the notion of 'welding' in [37], and generalises the cases for 2D topological codes given in [26, 34, 38]. We prove that merging two LDPC codes in such a manner still yields an LDPC code. We give a series of examples, including the specific case of lattice surgery between surface codes. Lastly, we discuss how to apply such protocols in practice. We prove that when two technical conditions are satisfied then code surgery can be performed while maintaining the error-correcting properties of the code, allowing us to perform logical parity measurements on codes.

1.1 Guide to reading the paper

Section 2 gives a bird's eye view of category theory and universal constructions, which will be useful later on. Section 3 describes the category of chain complexes with morphisms as matrices over $\mathbb{F}_2$. Category theorists may wish to skip past these sections.

We then give a rundown of CSS codes viewed as chain complexes in Section 4. Readers familiar with basic category theory and this perspective of CSS codes can safely skip to Section 4.3, where we introduce the notion of *code maps*, that is coherent transforms between codes.

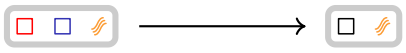
We introduce surgery of codes as a colimit in Section 5. This is when the notion of 'gluing' codes together comes in, and we prove several results about these codes when the colimit uses logical $\overline{Z}$ or $\overline{X}$-operators. Lastly, we introduce a protocol for performing logical $\overline{Z} \otimes \overline{Z}$ and $\overline{X} \otimes \overline{X}$ measurements in Section 6.

2 Universal constructions

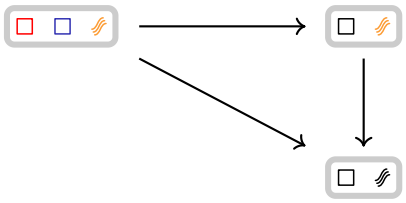
In this section we provide a cartoon introduction to category theory and universal constructions. See [33] for a more in-depth introduction.

A *category* is a collection of *objects* and *morphisms*. We will begin by drawing an object as a box with a decoration, such as

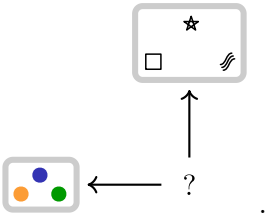
Morphisms are arrows between objects, like this



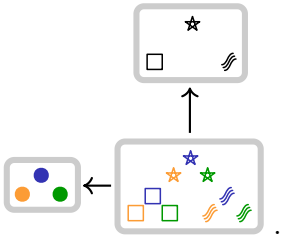
The arrow notation suggests that we can *compose* these.



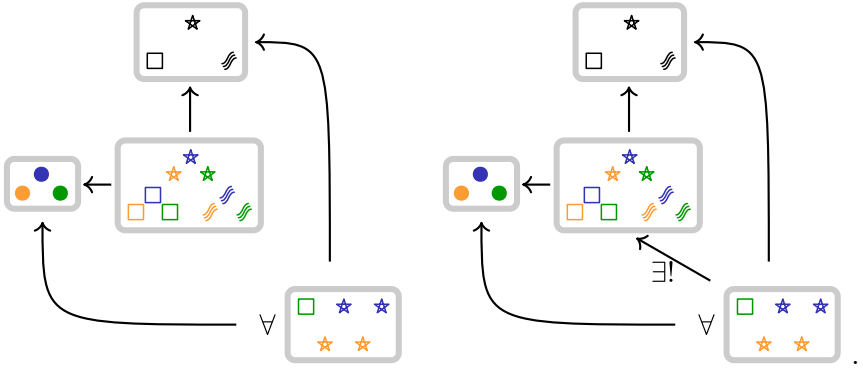
The *product* of two objects in a category is an object, together with two arrows,



The product decoration combines the two decorations,

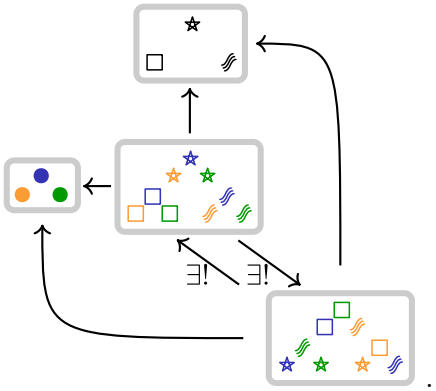


The product also must satisfy a *universal property*. This states that any other object that also combines the two decorations is already compatible with the product object in a unique way. In other words, for all test objects there exists a unique *comparison* morphism:

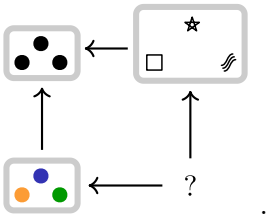


The real product is the minimal object that projects down to the factors. Any other test object lives over the real product.

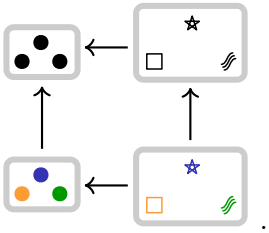
This universal property has the immediate consequence that any other object that satisfies all these requirements, will be isomorphic via a unique isomorphism that commutes with the other morphisms,



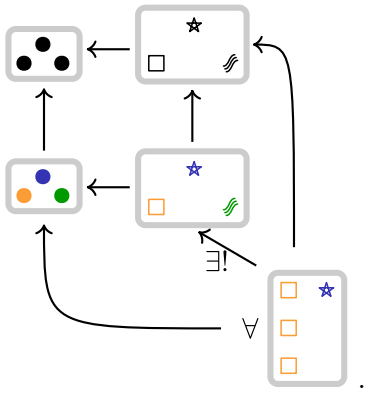
A *pullback* is a product with constraints:



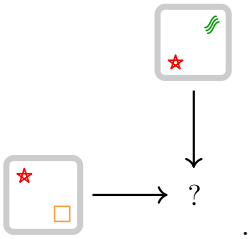
The resulting square should *commute*: if we compose any two paths of arrows with the same source object and the same target object then these paths should be equal.



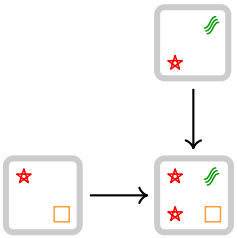
As with products, we also require the pullback to satisfy a universal property.



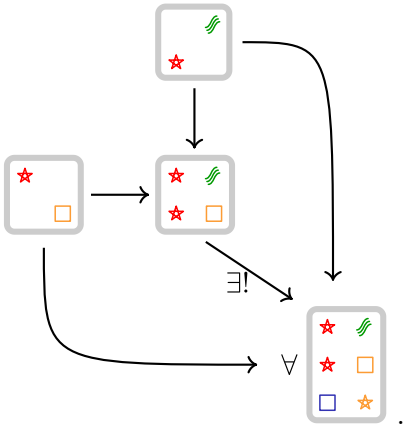
All of these statements have *dual* statements, which we get by reversing all the arrows. When we do this we sometimes put a *co-* prefix on the terminology. For example, a *coproduct*, which would normally be called a sum, looks like this



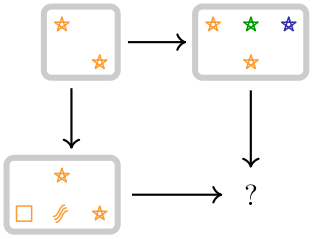
Once again, we require any such candidate coproduct



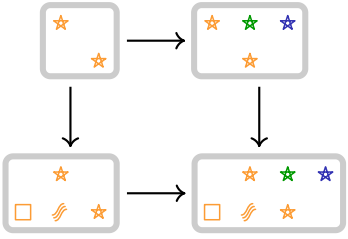
to satisfy a universal property



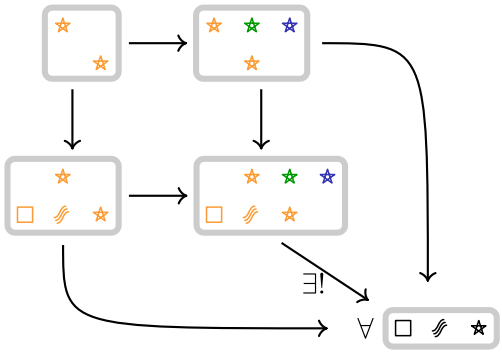
We think of a coproduct as a way of gluing together objects. By adding constraints we can express where we wish to glue



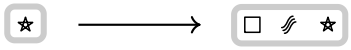
The answer to this question is called a *pushout*: it is an object together with two morphisms,



that satisfies a universal property,

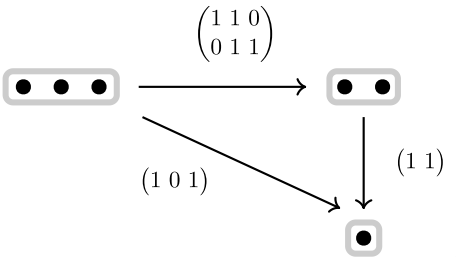


We have purposefully avoided describing the decorations in these diagrams: how they work, what they mean. A more in-depth introduction to category theory would describe these systematically, possibly mentioning the category of *finite sets and functions*. In this case, objects are sets, with *elements*, and we can combine these in various ways to make other sets. Instead of telling this story, we skip to the punchline, which is that there are no elements, or rather, an element of an object is really a morphism into that object:



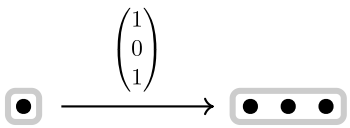
the star is an element of

To push this idea home, and also move toward the goals of this paper, we consider the category $\mathtt{Mat}_{\mathbb{F}_2}$ of finite dimensional matrices over the field $\mathbb{F}_2$. This has as objects the natural numbers and matrices over $\mathbb{F}_2$ as morphisms. Composition of morphisms is matrix multiplication. We will show each object as a box with dots. For example, here is a composition of two morphisms in this category



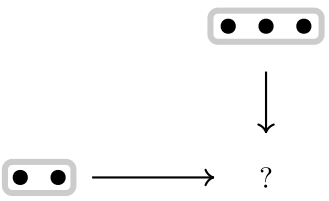
.

The objects serve only as anchors for the morphisms (matrices) where all the action is taking place. The vector elements of a vector space have vanished into the morphisms:

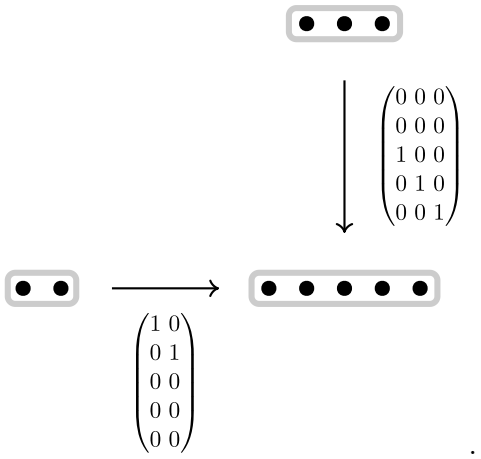


the vector in a vector space

A coproduct (sum) in this category is an object together with two morphisms

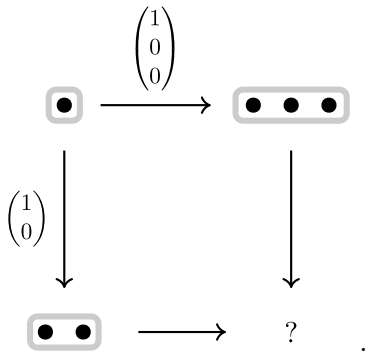


satisfying the universal property of coproducts. Here is one candidate:

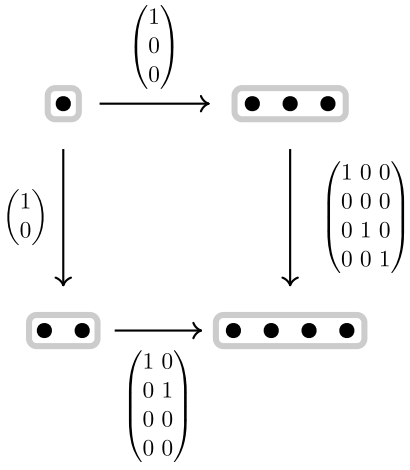


This coproduct will not be unique (except for some degenerate cases), but the universal property of the coproduct guarantees it is unique up to unique isomorphism. We have reinvented the *direct sum* of vector spaces.

For a pushout of vector spaces

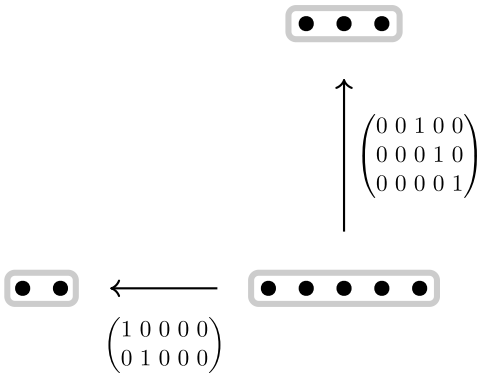


we get



This is gluing of a two dimensional vector space and a three dimensional vector space along a one dimensional vector space.

But what about products? A curious thing happens in the category $\mathtt{Mat}_{\mathbb{F}_2}$; we can get the dual universal construction by transposing matrices. For example, the above coproduct becomes the product:



and similarly with pullbacks. The transpose duality of $\mathtt{Mat}_{\mathbb{F}_2}$ will follow us throughout the rest of this paper.

Here we have been taking the objects of $\mathtt{Mat}_{\mathbb{F}_2}$ to be just natural numbers. In the rest of the paper we will use a slightly different definition for the objects: each natural number n is replaced by a basis set of size n for an n-dimensional vector space.

3 Chain complexes

We now recap some elementary homological algebra. All of this section is known [45], but we fix notation and look explicitly at the particular category of interest.

Let $\mathtt{Mat}_{\mathbb{F}_2}$ be the category which has as objects linearised finite sets over $\mathbb{F}_2$, so each object is a vector space V equipped with a specified basis $\tilde{V}$ such that $V = \mathbb{F}_2\tilde{V}$. Each element of $\tilde{V}$ corresponds to an entry for vectors in V. Importantly, we require the vector spaces to have this form on the nose. Throughout, we will call these *based vector spaces*. A morphism $f : V \to W$ in $\mathtt{Mat}_{\mathbb{F}_2}$ is a $\dim W \times \dim V$ matrix valued in $\mathbb{F}_2$.

Let $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$ be the category of bounded chain complexes in $\mathtt{Mat}_{\mathbb{F}_2}$. We now recap some of the basic properties of this category. A chain complex $C_\bullet$ looks like this:

$$\cdots \longrightarrow C_{n+1} \xrightarrow{\partial_{n+1}} C_n \xrightarrow{\partial_n} C_{n-1} \longrightarrow \cdots$$

where each component C_i is a based vector space and $n \in \mathbb{Z}$ is called the degree of the component in $C_\bullet$. $C_\bullet$ has $\mathbb{F}_2$-matrices as differentials $\partial_{n+1} : C_{n+1} \to C_n$ such that $\partial_n \circ \partial_{n+1} = 0 \pmod 2$, $\forall n \in \mathbb{Z}$. To disambiguate differentials between chain complexes we will use $\partial_n^{C_\bullet} := \partial_n \in C_\bullet$ when necessary.

All our chain complexes are bounded, meaning there is some $k \in \mathbb{Z}$ such that $C_{n>k} = 0$ and $l \in \mathbb{Z}$ such that $C_{n<l} = 0$, i.e. it is bounded above and below. We call $k - l$ the length of $C_\bullet$ for k and l the smallest and largest possible values respectively.

Definition 3.1. *Given a chain complex $C_\bullet$ we let*

$$Z_n(C_\bullet) = \ker(\partial_n); \quad B_n(C_\bullet) = \mathrm{im}(\partial_{n+1})$$

and call Z_n, B_n the n-cycles and n-boundaries. We also define a quotient $H_n(C_\bullet) = Z_n(C_\bullet)/B_n(C_\bullet)$, and call H_n the nth homology space of $C_\bullet$.

Recall that $\dim(\ker(\partial_n)) = \mathrm{null}(\partial_n) = \dim C_n - \mathrm{rank}(\partial_n)$. Throughout we sometimes use $\ker(f)$ of a matrix f to mean the kernel object, i.e. subspace, and sometimes the kernel morphism, i.e. inclusion map. It should be clear from context which is meant.

Example 3.2. *Let $\Gamma = (V, E)$ be a finite simple undirected graph. We can form the incidence chain complex $C_\bullet$ of Γ, which has $C_0 = \mathbb{F}_2 V$, $C_1 = \mathbb{F}_2 E$. All other components are zero. The sole nonzero differential ∂_1 is the incidence matrix of Γ, with $(\partial_1)_{ij} = 1$ if the jth edge is attached to the ith vertex, and 0 otherwise. $H_1(C_\bullet)$ is determined by the graph homology of Γ [45].*

Definition 3.3. *A morphism $f_\bullet : C_\bullet \to D_\bullet$ in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$ is called a chain map, and consists of a collection of matrices $\{f_i : C_i \to D_i\}_{i\in\mathbb{Z}}$ such that each resultant square of maps commutes:*

$$\begin{array}{ccccccccc}
\cdots & \longrightarrow & C_{n+1} & \xrightarrow{\partial^{C_\bullet}_{n+1}} & C_n & \xrightarrow{\partial^{C_\bullet}_{n}} & C_{n-1} & \longrightarrow & \cdots \\
 & & \downarrow{\scriptstyle f_{n+1}} & & \downarrow{\scriptstyle f_n} & & \downarrow{\scriptstyle f_{n-1}} & & \\
\cdots & \longrightarrow & D_{n+1} & \xrightarrow{\partial^{D_\bullet}_{n+1}} & D_n & \xrightarrow{\partial^{D_\bullet}_{n}} & D_{n-1} & \longrightarrow & \cdots
\end{array}$$

As we specified *bounded* chain complexes only a finite number of the f_i matrices will be non-zero. A chain map $f_\bullet$ is an isomorphism in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$ iff all f_i are invertible, in which case one can think of the isomorphism as being a 'change of basis' for all components, which thus transforms the differential matrices appropriately. Every pair of chain complexes has at least two chain maps, the zero chain maps, between them, given by a collection of entirely zero matrices either way.

Definition 3.4. *We say that a chain map is* basis-preserving *when every matrix at each component maps basis elements to basis elements.*

This does not require that the map is either monic or epic, and is a property associated only with based vector spaces, as it evidently does not arise with abstract vector spaces. We can equivalently see this property as the case when the chain map is a collection of functions between the underlying sets $\{\tilde{f}_i : \tilde{C}_i \to \tilde{D}_i\}_{i\in\mathbb{Z}}$.

Lemma 3.5. *A chain map at a component $f_n : C_n \to D_n$ lifts to a matrix $H_n(f_\bullet) : H_n(C_\bullet) \to H_n(D_\bullet)$.*

Proof. It is easy to check that f_n induces matrices from $Z_n(C_\bullet) \to Z_n(D_\bullet)$ and the same for B_n. □

This lemma is equivalent to saying that $H_n(-)$ is a functor from $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2}) \to \mathtt{Mat}_{\mathbb{F}_2}$.

$\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$ has several known categorical properties which will be useful to us. One way to see a chain complex $C_\bullet$ in $\mathtt{Mat}_{\mathbb{F}_2}$ is as a $\mathbb{Z}$-graded $\mathbb{F}_2$-vector space, with a specified basis and a distinguished map $\partial : C_\bullet \to C_\bullet$ with components $\partial_i : C_{i+1} \to C_i$, such that $\partial \circ \partial = 0$. Many of the properties of $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$ are inherited directly from those of $\mathbb{Z}$-graded $\mathbb{F}_2$-vector spaces.

$\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$ is an additive category, i.e. it has all finite biproducts, i.e. direct sums. These have components

$$(C \oplus D)_n = C_n \oplus D_n$$

and the same for differentials. This is both a categorical product and coproduct. Homology preserves direct sums: given chain complexes $C_\bullet$ and $D_\bullet$,

$$H_n((C \oplus D)_\bullet) \cong H_n(C_\bullet) \oplus H_n(D_\bullet)$$

This is obvious, considering the blocks of each differential in $(C \oplus D)_\bullet$.

A co-chain complex is similar to a chain complex, with some notational differences.

Definition 3.6. *Given a chain complex $C_\bullet$, the co-chain complex $C^\bullet$ has components*

$$C^n = C_n$$

and differentials

$$\delta^n = \partial_{n-1}^\intercal.$$

We also have co-cycles $Z^n = \ker(\delta^n)$, co-boundaries $B^n = \mathrm{im}(\delta^{n-1})$ and co-homology spaces $H^n = Z^n/B^n$. It is easy to show that $H_n(C_\bullet) \cong H^n(C^\bullet)$.

A co-chain map is defined similarly to a chain map:

$$\begin{array}{ccccccccc}
\cdots & \longrightarrow & C^{n-1} & \xrightarrow{\delta^{n-1}_{C^\bullet}} & C^n & \xrightarrow{\delta^{n}_{C^\bullet}} & C^{n+1} & \longrightarrow & \cdots \\
 & & \downarrow{\scriptstyle f^{n-1}} & & \downarrow{\scriptstyle f^{n}} & & \downarrow{\scriptstyle f^{n+1}} & & \\
\cdots & \longrightarrow & D^{n-1} & \xrightarrow{\delta^{n-1}_{D^\bullet}} & D^n & \xrightarrow{\delta^{n}_{D^\bullet}} & D^{n+1} & \longrightarrow & \cdots
\end{array}$$

and these are morphisms in the category $\mathtt{Coch}(\mathtt{Mat}_{\mathbb{F}_2})$. There is a functor $H^n(-) : \mathtt{Coch}(\mathtt{Mat}_{\mathbb{F}_2}) \to \mathtt{Mat}_{\mathbb{F}_2}$. We can also ask for co-chain maps to be basis-preserving. Given a chain map $f_\bullet$, we automatically also have the cochain map $f^\bullet$, with components $f^n = f_n^\intercal$.

4 Quantum codes

Here we introduce classes of both classical and quantum codes as chain complexes. We give easy examples such as the surface and toric codes. Up until Section 4.3, this part is also well-known, although we describe the relationship between Z and X operators in greater detail than we have found elsewhere.

4.1 Codes as chain complexes

Binary linear classical codes which encode k bits using n bits can be described by a $m \times n$ parity check $\mathbb{F}_2$-matrix P. The parity check matrix P, when applied to any codeword of length n, gives $Pc = 0$, and thus $k = \dim \ker(P)$; if the result is non-zero then an error has been detected, and under certain assumptions can be corrected. The distance d of a binary linear classical code is the minimum Hamming weight of its nonzero codewords, and one characterisation of codes is by the parameters $[n, k, d]$. We may trivially view a binary linear classical code as a length 1 chain complex, with indices chosen for convenience:

$$C_\bullet = C_1 \xrightarrow{\partial_1} C_0$$

where $C_1 = \mathbb{F}_2^n$, $C_0 = \mathbb{F}_2^m$, and $\partial_1 = P$, the chosen $m \times n$ parity check matrix. Then we have $k = \dim H_1(C_\bullet) = \dim Z_1(C_\bullet)$, where $Z_1(C_\bullet)$ is the codespace.

Example 4.1. *Let $C_\bullet$ be a $[3, 1, 3]$ repetition code, encoding 1 bit into 3 bits. In this case, let*

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

Example 4.2. *Let $C_\bullet$ be the $[7, 4, 3]$ Hamming code. Then let*

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

We now move on to quantum codes. Qubit Calderbank-Shor-Steane (CSS) codes are a type of stabiliser quantum code. Let $\mathscr{P}_n = \mathscr{P}^{\otimes n}$ be the Pauli group over n qubits. Stabiliser codes start by specifying an Abelian subgroup $\mathscr{S} \subset \mathscr{P}_n$, called a stabiliser subgroup, such that the codespace $\mathscr{H}$ is the mutual $+1$ eigenspace of all operators in $\mathscr{S}$. That is,

$$U|\psi\rangle = |\psi\rangle \quad \forall U \in \mathscr{S}, |\psi\rangle \in \mathscr{H}$$

We then specify a generating set of $\mathscr{S}$, of size m. For CSS codes, this generating set has as elements tensor product strings of either $\{I, X\}$ or $\{I, Z\}$ Pauli terms, with no scalars other than 1. One can define two parity check $\mathbb{F}_2$-matrices P_X, P_Z, for the Xs and Zs, which together define a particular code. Each column in P_X and P_Z represents a physical qubit, and each row a measurement/stabiliser generator. P_X and P_Z thus map Z and X operators on physical qubits respectively to sets of measurement outcomes, with a 1 outcome if the operators anticommute with a given stabiliser generator, and 0 otherwise; these outcomes are also called *syndromes*. P_X is a $m_X \times n$ matrix, and P_Z is $m_Z \times n$, with m_X, m_Z marking the division of the generating set into Xs and Zs respectively, satisfying $m = m_X + m_Z$. We do not require the generating set to be minimal, and hence P_X and P_Z need not be full rank.

Definition 4.3. *We say that w^Z is the maximal weight of all Z-type generators and w^X the same for the X-type generators. These are the highest weight rows of P_Z and P_X respectively. Similarly, we say that q^Z, q^X is the maximal number of Z, X generators sharing a single qubit. These are the highest weight columns of P_Z and P_X.*

CSS codes are described by parameters $[\![n, k, d]\!]$, with k the number of encoded qubits and d the code distance, which we define presently.

That the stabilisers must commute is equivalent to the requirement that $P_X P_Z^\intercal = P_Z P_X^\intercal = 0$. We may therefore view these matrices as differentials in a length 2 chain complex:

$$C_\bullet = C_2 \xrightarrow{\partial_2} C_1 \xrightarrow{\partial_1} C_0$$

where $\partial_2 = P_Z^\intercal$ and $\partial_1 = P_X$, or the other way round ($\partial_2 = P_X^\intercal, \partial_1 = P_Z$) if desired, but we use the former for consistency with the literature. The quantum code then has $C_1 = \mathbb{F}_2^n$, and thus:

$$C_\bullet = \mathbb{F}_2^{m_Z} \xrightarrow{P_Z^\intercal} \mathbb{F}_2^n \xrightarrow{P_X} \mathbb{F}_2^{m_X}$$

The code also has $k = \dim\ H_1(C_\bullet)$. To see this, observe first that C_1 represents the space of Z Paulis on the set of physical qubits, with a vector being a Pauli string e.g. $v = \begin{pmatrix}1 & 0 & 1\end{pmatrix}^\intercal \rightsquigarrow Z \otimes I \otimes Z$. Each vector in $H_1(C_\bullet)$ can be interpreted as an equivalence class $[v]$ of Z operators on the set of physical qubits, modulo Z operators which arise as Z stabilisers. That this vector is in $Z_1(C_\bullet)$ means that the Z operators commute with all X stabilisers, and when the vector is not in $[0] = B_1(C_\bullet)$ it means that the Z operators act nontrivially on the logical space. A basis of $H_1(C_\bullet)$ constitutes a choice of individual logical Paulis $\overline{Z}$, that is a tensor product decomposition of the space of logical Z operators, and we set $\overline{Z}_1 = \overline{Z} \otimes \overline{I} \cdots \otimes \overline{I}$ on *logical* qubits, $\overline{Z}_2 = \overline{I} \otimes \overline{Z} \cdots \otimes \overline{I}$ etc. There is a logical qubit for every logical Z, hence $k = \dim H_1(C_\bullet)$.

To get the logical X operators, consider the cochain complex $C^\bullet$. The vectors in $H^1(C^\bullet)$ then correspond to $\overline{X}$ operators in the same manner. As $H_i(C_\bullet) \cong H^i(C^\bullet)$ there must be an $\overline{X}$ operator for every $\overline{Z}$ operator and vice versa.

Lemma 4.4. *A choice of basis $\{[v]_i\}_{i \leq k}$ for $H_1(C_\bullet)$ implies a choice of basis $\{[w]_j\}_{j \leq k}$ for $H^1(C^\bullet)$.*

Proof. First, recall that we have the nondegenerate bilinear form

$$\cdot : \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2; \quad u \cdot v = u^\intercal v$$

which is equivalent to $\cdot : C_1 \times C^1 \to \mathbb{F}_2$; computationally, this tells us whether a Z operator commutes or anticommutes with an X operator. Now, let $u \in Z_1(C_\bullet)$ be a (possibly trivial) logical Z operator, and $v \in B^1(C^\bullet)$ be a product of X stabilisers. Then $P_X u = 0$, and $v = P_X^\intercal w$ for some $w \in C^2$. Thus $u \cdot v = u^\intercal v = u^\intercal P_X^\intercal w = (P_X u)^\intercal w = 0$, and so products of X stabilisers commute with logical Z operators. The same applies for Z stabilisers and logical X operators.

As a consequence, $v \cdot w = (v+s) \cdot (w+t)$ for any $v \in Z_1(C_\bullet)$, $w \in Z^1(C^\bullet)$, $s \in B_1(C_\bullet)$, $t \in B^1(C^\bullet)$, and so we may define $[v] \cdot [w] = v \cdot w$ for any $[v] \in H_1(C_\bullet)$, $[w] \in H^1(C^\bullet)$ with representatives v, w. The duality pairing of C_1, C^1 thus lifts to $H_1(C_\bullet), H^1(C^\bullet)$, and a choice of basis $\{[v]_i\}_{i \leq k}$ for $H_1(C_\bullet)$ implies a choice of basis of $H^1(C^\bullet)$, determined uniquely by $[v]_i \cdot [w]_j = \delta_{i,j}$. □

The above lemma ensures that picking a tensor product decomposition of logical Z operators also entails the same tensor product decomposition of logical X operators, so that $\overline{X}_i \overline{Z}_j = (-1)^{\delta_{i,j}} \overline{Z}_j \overline{X}_i$, for operators on the ith and jth logical qubits.

Let

$$d^Z = \min_{v \in Z_1(C_\bullet) \setminus B_1(C_\bullet)} |v|; \quad d^X = \min_{w \in Z^1(C^\bullet) \setminus B^1(C^\bullet)} |w|$$

where $|\cdot|$ is the Hamming weight of a vector, then the code distance $d = \min(d^Z, d^X)$. d^Z and d^X are called the systolic and cosystolic distances, and represent the lowest weight nontrivial Z and X logical operators respectively.

As all the data required for a CSS code is contained within the chain complex $C_\bullet$ – and potentially a choice of basis of $H_1(C_\bullet)$ – then we could define a CSS code as just the single chain complex, but it will be convenient to have direct access to the cochain complex as well.

Definition 4.5. *A CSS code is a pair $(C_\bullet, C^\bullet)$, with $C_\bullet$ a length 2 chain complex centred at degree 1, so we have:*

$$C_\bullet = C_2 \xrightarrow{P_Z^\intercal} C_1 \xrightarrow{P_X} C_0 \ ; \qquad C^\bullet = C^0 \xrightarrow{P_X^\intercal} C^1 \xrightarrow{P_Z} C^2$$

We call the first of the pair the Z-type complex, as vectors in C_1 correspond to Z-operators, and the second the X-type complex. A based *CSS code additionally has a choice of basis for $H_1(C_\bullet)$, and hence for $H^1(C^\bullet)$.*

Employing the direct sum $(C \oplus D)_\bullet$ of chain complexes we have the CSS code $((C \oplus D)_\bullet, (C \oplus D)^\bullet)$, which means the CSS codes $(C_\bullet, C^\bullet)$ and $(D_\bullet, D^\bullet)$ perform in parallel on disjoint sets of qubits, without any interaction. The Z and X operators will then be the tensor product of operators in each.

In summary, there is a bijection between length 1 chain complexes in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$ and binary linear classical codes, and between length 2 chain complexes in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$ and CSS codes. There are CSS codes for higher dimensional qudits, but for simplicity we stick to qubits.

Rather than just individual codes we tend to be interested in families of codes, where n, k, d scale with the size of code in the family. Of particular practical interest are quantum *low density parity check* (LDPC) CSS codes, which are families of codes where all w^Z, w^X, q^Z and q^X in the family are bounded from above by a constant. Equivalently, this means the Hamming weight of each column and row in each differential is bounded by a constant.

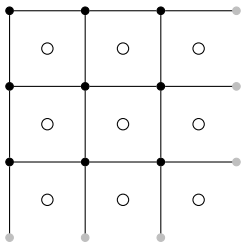
Example 4.6. *Let $(C_\bullet, C^\bullet)$ be the $[\![9,1,3]\!]$ Shor code, so we have $C_2 = \mathbb{F}_2^2$, $C_1 = \mathbb{F}_2^9$, $C_0 = \mathbb{F}_2^6$. The parity check matrices are given by*

$$P_X = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}; \quad P_Z = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix}$$

We then have $\dim Z_1(C_\bullet) = \dim C_1 - \operatorname{rank}(P_X) = 9-2 = 7$ *and* $\dim B_1(C_\bullet) = \operatorname{rank}(P_Z^\intercal) = 6$. *Thus* $k = \dim H_1(C_\bullet) = 1$. *There is a single nonzero equivalence class* $[v] \in H_1(C_\bullet)$, *with a representative* $v = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}^\intercal$. *Similarly there is the nonzero vector* $w = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}^\intercal$, *which is a representative of* $[w] \in H_1(C_\bullet)$. *Hence, we have two logical operators* $\overline{Z} = \bigotimes_i^9 Z_i$, $\overline{X} = \bigotimes_i^9 X_i$ *with* Z_i *on the* i*th qubit and the same for* X_i. *We equally have, say,* $\overline{Z} = Z_1 \otimes Z_4 \otimes Z_7$ *and* $\overline{X} = X_1 \otimes X_2 \otimes X_3$ *in the same equivalence classes as those above,* $[v]$ *and* $[w]$.

We now consider two examples which come from square lattices. This can be done much more generally. In Appendix C we formalise categorically the procedure of acquiring chain complexes – and therefore CSS codes – from square lattices, which are a certain type of cell complex.

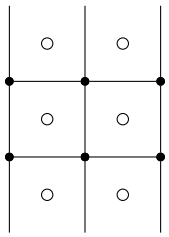
Example 4.7. *Consider the following square lattice:*



Edges in the lattice are qubits, so $n = 18$, the 9 X-checks are associated with vertices and the 9 Z-checks are associated with faces, which are indicated by white circles. Grey vertices indicate periodic boundary conditions, so the lattice can be embedded on a torus. This is an instance of the standard toric code [31].

The abstracted categorical homology from before is now the homology of the tessellated torus, with cycles, boundaries etc. having their usual meanings. $k = \dim H_1(C_\bullet) = 2$, and (co)systolic distances are the lengths of the essential cycles of the torus.

Example 4.8. *Now consider a different square lattice:*



This represents a patch of surface code $(D_\bullet, D^\bullet)$, where we have two smooth sides, on the left and right, and two rough sides to the patch, on the top and bottom. There are 'dangling' edges at the top and bottom, which do not terminate at vertices. We have

$$\dim D_2 = \dim D_0 = 6; \quad n = \dim D_1 = 13; \quad k = \dim H_1(D_\bullet) = 1$$

The systolic distance is 3*, the length of the shortest path from the top to bottom boundary, and the cosystolic distance* 3*, the same but from left to right.*

4.3 Code maps

One may wish to convert one code into another, making a series of changes to the set of stabiliser generators to be measured, and potentially also to the physical qubits. The motivation behind such protocols is typically to perform logical operations which are not available natively to the code; not only might the target code have other logical operations, but the protocol is itself a map between logical spaces when chosen carefully. An example of a change to the measurements and qubits is code deformation. We do not formalise code deformation here, as that has some specific connotations [44]. Instead we define a related notion, called a *code map*, which has some overlap. A code map is also related to, but not the same as, the 'homomorphic gadgets' from [27].

Definition 4.9. *A $\overline{Z}$-preserving code map $\mathcal{F}_{\overline{Z}}$ from a CSS code $(C_\bullet, C^\bullet)$ to $(D_\bullet, D^\bullet)$ is a paired chain map and cochain map $(f_\bullet, f^\bullet)$, for $f_\bullet : C_\bullet \to D_\bullet$ and $f^\bullet : D^\bullet \to C^\bullet$.*

$$\mathcal{F}_{\overline{Z}} \Downarrow \qquad \begin{array}{c} (C_\bullet, C^\bullet) \\ f_\bullet \downarrow \quad \uparrow f^\bullet \\ (D_\bullet, D^\bullet) \end{array}$$

Note that the cochain map is strictly speaking obsolete, as all the data is contained in a single chain map $f_\bullet$, but as with CSS codes it will be handy to keep both around.

Let us unpack this definition. $\mathcal{F}_{\overline{Z}}$ first maps Z-operators in C_1 to Z-operators in D_1, using f_1. It may map a single Z on a qubit to a tensor product of Zs, or to I. It then has a map f_2 on Z generators, and another f_0 on X checks. Recalling Definition 3.3, we have:

$$\begin{array}{ccccc} C_2 & \xrightarrow{\partial_2^{C\bullet}} & C_1 & \xrightarrow{\partial_1^{C\bullet}} & C_0 \\ {\scriptstyle f_2}\downarrow & \mathrm{I} & \downarrow{\scriptstyle f_1} & \mathrm{II} & \downarrow{\scriptstyle f_0} \\ D_2 & \xrightarrow[\partial_2^{D\bullet}]{} & D_1 & \xrightarrow[\partial_1^{D\bullet}]{} & D_0 \end{array}$$

With two commuting squares labelled I and II. I stipulates that applying products of Z stabiliser generators on the code and then performing the code map should be equivalent to performing the code map and then applying products of Z stabiliser generators, i.e. $f_1 \circ \partial_2^{C\bullet} = \partial_2^{D\bullet} \circ f_2$. II stipulates that performing the X measurements and then mapping the code should be equivalent to mapping the code and then performing X measurements, so there is a consistent mapping between all measurement outcomes, i.e. $f_0 \circ \partial_1^{C\bullet} = \partial_1^{D\bullet} \circ f_1$.

Then there is the cochain map $f^\bullet$. This has the component $f^1 = f_1^\intercal : D^1 \to C^1$, which maps an X-operator in D^1 back to an X-operator in C^1. Similarly for f^0 and $f_2^\intercal$, each of which come with commuting squares which are just the transposed conditions of those in $f_\bullet$, so they say nothing new. This is not surprising, as all the data for $f^\bullet$ is given by $f_\bullet$ already.

We now show that this definition entails some elementary properties. For a start, Lemma 3.5 implies that a code map gives a map from a $\overline{Z}$ operator in $H_1(C_\bullet)$ to $\overline{Z}$s in $H_1(D_\bullet)$; this can also map to a tensor product of logical $\overline{Z}$s, and in particular map $\overline{Z}$ to zero i.e. $\overline{I}$, but it must not map a $\overline{Z}$ to an operator which can be detected by the X stabiliser measurements. Hence $(f_\bullet, f^\bullet)$ preserves the fact that any $\overline{Z}$ is an undetectable operator on the codespace. A similar requirement holds for $\overline{X}$ operators, but this time the condition is inverted. Every $\overline{X}$ in $H^1(D^\bullet)$ must have a map only to logical operators in $H^1(C^\bullet)$, but the other way is not guaranteed.

Let n_C and n_D be the number of physical qubits in codes $(C_\bullet, C^\bullet)$ and $(D_\bullet, D^\bullet)$ respectively. We may interpret $\mathcal{F}_{\overline{Z}}$ as a $\mathbb{C}$-linear map M in `FHilb`, the category of Hilbert spaces. This $\mathbb{C}$-linear map has the property that $MU_Z = U'_Z M$, where U_Z is a tensor product of Z Paulis on n_C qubits and U'_Z is a tensor product of Z Paulis on n_D qubits. In particular, given any U_Z we have a specified U'_Z. The same is not true the other way round, as the map f_1 is not necessarily injective or surjective. Similarly, $MU_X = U'_X M$. This time, however, given any unique U'_X on n_D qubits we have a specified U_X but vice versa is not guaranteed, depending on f_1^T.

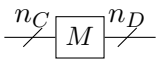
As a consequence, the linear map M is *stabiliser*, in the sense that it maps Paulis to Paulis, but not *unitary* in general. M is unitary iff f_1 is invertible.

If M is not even an isometry, it cannot be performed deterministically, and the code map must include measurements on physical qubits. There will in general be Kraus operators corresponding to different measurement outcomes which will determine whether the code map has been implemented as desired; for now we assume that M is performed deterministically, and leave this complication for Section 6. Similarly, while the code map can be interpreted as a circuit between two codes, we do not claim that such a circuit can be performed fault-tolerantly in general.

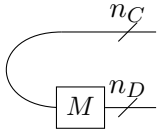
Remark 4.10. *For the following proposition, and at various points throughout the rest of the paper, we will use the ZX-calculus, a formal graphical language for reasoning about computation with qubits. We do not give a proper introduction to this calculus for brevity, but Sections 1-3 of [43] are sufficient for the interested reader. Our use of ZX diagrams is unsophisticated, and primarily for convenience.*

Proposition 4.11. *Let $\mathcal{F}_{\overline{Z}}$ be a $\overline{Z}$-preserving code map between codes $(C_\bullet, C^\bullet)$ and $(D_\bullet, D^\bullet)$ with qubit counts n_C and n_D. The interpretation of $\mathcal{F}_{\overline{Z}}$ as a $\mathbb{C}$-linear map M in* `FHilb` *has a presentation as a circuit with gates drawn from* $\{\mathrm{CNOT}, |+\rangle, \langle 0|\}$.

Proof. We start with the linear map $M : (\mathbb{C}^2)^{\otimes n_C} \to (\mathbb{C}^2)^{\otimes n_D}$:

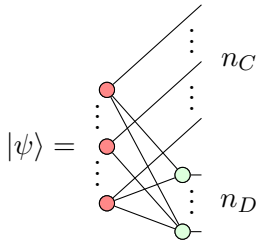


By employing the partial transpose in the computational basis we convert it into the state

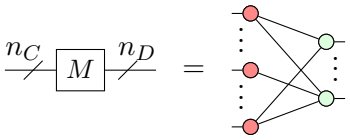
$|\psi\rangle =$ 

i.e. inserting n_C Bell pairs. By the definition of f_1 we know that this has an independent stabiliser, with one Z and $n_C - 1$ Is followed by some n_D-fold tensor product of Z and I, for each of the n_C qubits. From f_1^T it also has an independent stabiliser, with some n_C-fold tensor product of X and I followed by $n_D - 1$ Is and one X, for each of the n_D qubits.

$|\psi\rangle$ is therefore a stabiliser state. Further, from Theorem 5.1 of [29] it has a presentation as a 'phase-free ZX diagram', of the form



where the top n_C qubits do not have a green spider. We perform the partial transpose again to convert the state $|\psi\rangle$ back into the map M, which has the form



Any ZX diagram of this form can be expressed as a matrix over $\mathbb{F}_2$, mapping X-basis states from $(\mathbb{C}^2)^{\otimes n_C}$ to $(\mathbb{C}^2)^{\otimes n_D}$. The example above, ignoring the ellipses, has the matrix

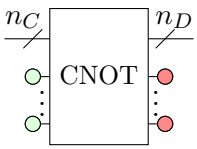
$$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

which is equal to f_1; the point of the above rigmarole is thus to say that f_1 is precisely a linear map between X-basis states, which one can check easily. One can explicitly calculate M as a matrix in the X-basis in `FHilb`. For the first column, we compute $f_1 \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, so $M|+\rangle|+\rangle|+\rangle = |+\rangle|+\rangle$. Overall, we have

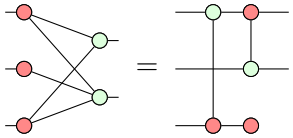
$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

which again is in the X-basis, not the (computational) Z-basis.

Returning to f_1, we can perform Gaussian elimination, performing row operations, which produce CNOTs on the r.h.s. of the diagram in the manner of [30], until the matrix is in reduced row echelon form. We then perform column operations producing CNOTs on the l.h.s. of the diagram, until the matrix has at most one 1 in each row and column. This can be performed using the leading coefficients to remove all other 1s in that row. The final matrix just represents a permutation of qubits with some states and effects. An empty column corresponds to a $\langle 0|$ effect, and an empty row a $|+\rangle$ state. We thus end up with a presentation of M in the form



On our example, this is then



which one can check maps $Z \otimes I \otimes I \mapsto Z \otimes Z$ etc. □

As a consequence $\bar{M} = M$, i.e. the conjugate of M is just M.

Corollary 4.12. *If $n_C = 0$ then the map M is actually a stabiliser state of the form $M = |+\rangle^{\otimes n_D}$. When $n_D = 0$ then $M = \langle 0|^{\otimes n_C}$.*

Proof. When $n_C = 0$ we see that M has exactly n_D independent stabilisers with 1 X and $n_D - 1$ Is, for each qubit to put X on. The flipped argument applies when $n_D = 0$. □

Definition 4.13. *An $\overline{X}$-preserving code map $\mathcal{F}_{\overline{X}}$ from a CSS code $(D_\bullet, D^\bullet)$ to $(C_\bullet, C^\bullet)$ is a paired chain map and cochain map $(f_\bullet, f^\bullet)$, for $f_\bullet : C_\bullet \to D_\bullet$ and $f^\bullet : D^\bullet \to C^\bullet$.*

$$\begin{array}{ccc} & & (C_\bullet, C^\bullet) \\ \mathcal{F}_{\overline{X}} \Uparrow & & f_\bullet \downarrow \quad \uparrow f^\bullet \\ & & (D_\bullet, D^\bullet) \end{array}$$

So $\mathcal{F}_{\overline{X}}$ is just mapping in the other direction to $\mathcal{F}_{\overline{Z}}$ from before, and we say that $\mathcal{F}_{\overline{X}}$ is *opposite* to $\mathcal{F}_{\overline{Z}}$. In this case, when we interpret $\mathcal{F}_{\overline{X}}$ as a $\mathbb{C}$-linear map L, it has the property that $LU_X = U'_X L$ and that any U_X gives a specified U'_X, and $LU_Z = U'_Z L$, but that any U'_Z gives a specified U_Z but not vice versa.

By inspecting the stabilisers we see that, for $\mathcal{F}_{\overline{Z}}$ with interpretation M and $\mathcal{F}_{\overline{X}}$ with interpretation L, $L = M^\dagger = M^\intercal$.

Corollary 4.14. *Let $\mathcal{F}_{\overline{X}}$ be an $\overline{X}$-preserving code map between codes $(D_\bullet, D^\bullet)$ and $(C_\bullet, C^\bullet)$ with qubit counts n_D and n_C. The interpretation of $\mathcal{F}_{\overline{X}}$ as a $\mathbb{C}$-linear map L in* `FHilb` *has a presentation as a circuit with gates drawn from $\{\mathrm{CNOT}, |0\rangle, \langle +|\}$.*

Corollary 4.15. *If $n_D = 0$ then $L = |0\rangle^{\otimes n_C}$, and if $n_C = 0$ then $L = \langle +|^{\otimes n_D}$.*

Corollary 4.16. *The restrictions of $\mathcal{F}_{\overline{Z}}$ and $\mathcal{F}_{\overline{X}}$ to use only $H_1(f_\bullet)$ and $H^1(f^\bullet)$ also have interpretations as $\mathbb{C}$-linear maps on logical qubits in the same way, and Proposition 4.11 and Corollary 4.14 also apply to such interpretations.*

While our definitions are for chain complexes of length 2, in principle one can map between any two codes with an arbitrary number of meta-checks, or between a classical code and quantum code, which could be interpreted as 'switching on/off' either X or Z stabiliser measurements.

Code maps are related to code deformations, but we are aware of code deformation protocols which do not appear to fit in the model of chain maps described. For example, when moving defects around on the surface code for the purpose of, say, defect braiding [21], neither $\overline{Z}$ nor $\overline{X}$ operators are preserved in the sense we give here.

5 CSS code surgery

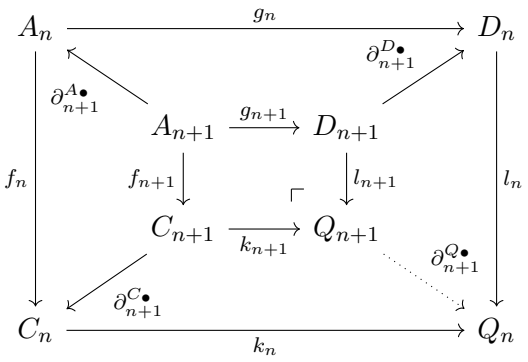
5.1 Colimits in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$

To understand code surgery we require some additional chain complex technology, namely colimits. Coproducts, pushouts and coequalisers are directly relevant for our applications. Coproducts are just direct sums, so we describe pushouts and coequalisers here.

Definition 5.1. *The pushout of chain maps $f_\bullet : A_\bullet \to C_\bullet$ and $g_\bullet : A_\bullet \to D_\bullet$ gives the chain complex $Q_\bullet$, where each component is the pushout Q_n of f_n and g_n. The differentials $\partial_n^{Q\bullet}$ are given by the unique mediating map from each component's pushout. Specifically, if we have the pushout*

$$\begin{array}{ccc} A_\bullet & \xrightarrow{g_\bullet} & D_\bullet \\ {\scriptstyle f_\bullet}\downarrow & \ulcorner & \downarrow{\scriptstyle l_\bullet} \\ C_\bullet & \xrightarrow[k_\bullet]{} & Q_\bullet \end{array}$$

then for degrees $n, n+1$ we have



where

$$Q_n = (C \oplus D)_n / f_n \sim g_n; \quad k_n(c) = [c] \in Q_n; \quad l_n(d) = [d] \in Q_n.$$

with $[c]$ being the equivalence class in Q_n having c as a representative, and the same for $[d]$. As $k_n \circ \partial^{C\bullet}_{n+1} \circ f_{n+1} = l_n \circ \partial^{D\bullet}_{n+1} \circ g_{n+1}$ and the inner square is a pushout in $\mathtt{Mat}_{\mathbb{F}_2}$, there is a unique matrix $\partial^{Q\bullet}_{n+1}$. The differentials satisfy $\partial^{Q\bullet}_n \circ \partial^{Q\bullet}_{n+1} = 0$, and one can additionally check that this is indeed a pushout in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$ by considering the universal property at each component.

Definition 5.2. *The coequaliser of chain maps $C_\bullet \underset{g}{\overset{f}{\rightrightarrows}} D_\bullet$ is a chain complex $E_\bullet$ and chain map $coeq(f,g)_\bullet : D_\bullet \to E_\bullet$, which we will just call $coeq_\bullet$. We have $E_n = D_n / f_n \sim g_n$ and $coeq_n(d) = [d]$.*

Doing some minor diagram chasing one can check that this is indeed a coequaliser in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$.

Remark 5.3. *We can view the pushout*

$$\begin{array}{ccc} A_\bullet & \xrightarrow{g_\bullet} & D_\bullet \\ {\scriptstyle f_\bullet}\downarrow & \ulcorner & \downarrow{\scriptstyle l_\bullet} \\ C_\bullet & \xrightarrow[k_\bullet]{} & Q_\bullet \end{array}$$

as the coequaliser of $A_\bullet \underset{\omega_\bullet \circ g_\bullet}{\overset{\tau_\bullet \circ f_\bullet}{\rightrightarrows}} (C \oplus D)_\bullet$ for the inclusion maps $C_\bullet \xhookrightarrow{\tau_\bullet} (C \oplus D)_\bullet$, $D_\bullet \xhookrightarrow{\omega_\bullet} (C \oplus D)_\bullet$. The difference is that the pair of chain maps $k_\bullet, l_\bullet$ have been replaced with the single map $coeq_\bullet$, so we have

$$A_\bullet \underset{\omega_\bullet \circ g_\bullet}{\overset{\tau_\bullet \circ f_\bullet}{\rightrightarrows}} (C \oplus D)_\bullet \xrightarrow{coeq_\bullet} Q_\bullet$$

We can view coequalisers as instances of pushouts as well, doing a sort of reverse of the procedure above.

As with all colimits, those above are defined by the category theory only up to isomorphism. Because we are working over a field, the isomorphism class of a chain complex $Q_\bullet$ is completely determined by the dimensions of the underlying vector spaces $\{\dim Q_i\}_i$ and its *Betti numbers*, which is the set $\{\dim H_i(Q_\bullet)\}_i$ of dimensions of the homology spaces. This is a homological version of the rank-nullity theorem. These are very large iso-classes, and we require more fine-grained control over which chain complexes are chosen by the colimits.

One way to choose a specific pushout of chain maps is via an explicit definition of the coequaliser of two based linear maps. For this we need not just a basis for our vector spaces, but an ordered basis. Using these coequalisers we can then construct pushouts of linear maps and their universal arrows, which is used in turn to define the pushout of chain maps. For the coequaliser of linear maps $r, s : V \to W$, we may take $s = 0$ by linearity. Then we let r^+ be the reflexive generalised inverse of r, which always exists by [46], and see that $P = I - rr^+$ is a projector $P : W \to W$ that coequalizes $r, 0 : V \to W$. To make this a universal projector we need to row-reduce the matrix of P, i.e. put P into row echelon form and remove all-zero rows, which is where we use the order on the basis of W. This row-reduced matrix will then have full rank and will be a universal coequaliser.

Alternatively, there is a straightforward way to choose the representatives we want from these iso-classes when the chain maps $f_\bullet, g_\bullet$ in the span are basis-preserving in the sense of Definition 3.4.

Lemma 5.4. *Let the chain maps $f_\bullet, g_\bullet$ be basis-preserving. Then there is always a choice of pushout such that the chain maps $k_\bullet, l_\bullet$ are also basis-preserving.*

Proof. Let $Q_n = (C \oplus D)_n / f_n \sim g_n$. Then, for any basis element $a \in A_n$, the elements $f_n(a)$ and $g_n(a)$ are mapped by k_n and l_n respectively to the same basis element in Q_n. For basis elements in C_n, D_n which are not in the images of f_n and g_n, k_n and l_n will map them to distinct basis elements in Q_n. So this choice of representative of the isomorphism class of pushouts has $k_\bullet$, $l_\bullet$ being basis-preserving. □

We can think of the pushout at each component as being a pushout in the category $\mathtt{Set}$, freely promoted to $\mathtt{Mat}_{\mathbb{F}_2}$. The differentials are then defined by the universal property. All of the choices of pushout such that $k_\bullet$, $l_\bullet$ are basis-preserving are equivalent up to a relabelling of basis elements in each component, hence (coherent) row and column permutation of the differential matrices. Computationally, this means that the choice of pushout is defined up to relabelling qubits, Z-checks and X-checks, and hence properties like code distance and being LDPC are well-defined for such pushouts. This will be important later.

Lemma 5.4 also applies to coequalisers, following Remark 5.3.

Lemma 5.5. $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$ *is an Abelian category, and thus is finitely complete and cocomplete, meaning that it has all finite limits and colimits.*

While this is well-known, we sketch a proof of this lemma in Appendix A. There we also sketch some additional limits for completeness, but we only need colimits in this work. This lemma does not mean that every span has a *basis-preserving* pushout, but whenever there is a basis-preserving span there is a basis-preserving pushout. The same applies to coequalisers. From now on every example of a span we use will be basis-preserving, so we assume that the pushouts and coequalisers are also basis-preserving and will no longer mention this property.

5.2 Generic code surgery

We now give a general set of definitions for surgery between arbitrary compatible CSS codes; the condition for compatibility is very weak here. Working at this level of generality means that we cannot prove very much about the output codes or relevant logical maps. As a consequence, we will then focus on particular surgeries which make use of 'gluing' or 'tearing' along logical $\overline{Z}$ or $\overline{X}$ operators in Section 5.3.

Definition 5.6. *Let $(C_\bullet, C^\bullet)$, $(D_\bullet, D^\bullet)$ and $(A_\bullet, A^\bullet)$ be CSS codes, such that there is a basis-preserving span of chain complexes*

$$\begin{array}{ccc} A_\bullet & \xrightarrow{g_\bullet} & D_\bullet \\ {\scriptstyle f_\bullet}\downarrow & & \\ C_\bullet & & \end{array}$$

The Z-type merged code of $(C_\bullet, C^\bullet)$ and $(D_\bullet, D^\bullet)$ along $f_\bullet, g_\bullet$ is the code $(Q_\bullet, Q^\bullet)$ such that $Q_\bullet$ is the pushout of the above diagram.

Recall from Remark 5.3 that we can view any pushout as a coequaliser. We thus have

$$A_\bullet \underset{\iota_D \circ g_\bullet}{\overset{\iota_C \circ f_\bullet}{\rightrightarrows}} (C \oplus D)_\bullet \xrightarrow{coeq_\bullet} Q_\bullet$$

and we call $coeq_\bullet$ the Z-merge chain map. We can bundle this up into a Z-merge code map:

$$\mathcal{F}_{\overline{Z}} \Downarrow \quad \begin{array}{c} ((C \oplus D)_\bullet, (C \oplus D)^\bullet) \\ {\scriptstyle coeq_\bullet}\downarrow \quad \uparrow{\scriptstyle coeq^\bullet} \\ (Q_\bullet, Q^\bullet) \end{array} \tag{1}$$

We then call $coeq^\bullet : Q^\bullet \to (C \oplus D)^\bullet$ an X-split cochain map, and hence we have an X-split code map too:

$$\mathcal{F}_{\overline{X}} \Uparrow \quad \begin{array}{c} ((C \oplus D)_\bullet, (C \oplus D)^\bullet) \\ {\scriptstyle coeq_\bullet}\downarrow \quad \uparrow{\scriptstyle coeq^\bullet} \\ (Q_\bullet, Q^\bullet) \end{array} \tag{2}$$

Definition 5.7. *Let $(C_\bullet, C^\bullet)$, $(D_\bullet, D^\bullet)$ and $(A_\bullet, A^\bullet)$ be CSS codes, such that there is a span of cochain complexes*

$$\begin{array}{ccc} A^\bullet & \xrightarrow{g^\bullet} & D^\bullet \\ {\scriptstyle f^\bullet}\downarrow & & \\ C^\bullet & & \end{array}$$

the X-type merged code of $(C_\bullet, C^\bullet)$ and $(D_\bullet, D^\bullet)$ along $f^\bullet, g^\bullet$ is the code $(Q_\bullet, Q^\bullet)$ such that $Q^\bullet$ is the pushout of the above diagram.

We have an X-merge cochain map and thus X-merge code map using the coequaliser picture, so

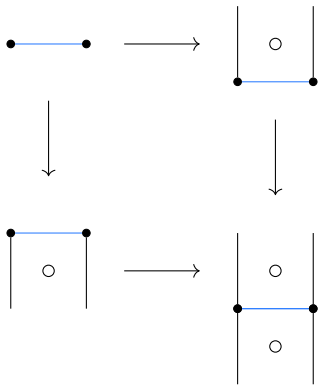
$$\mathcal{E}_{\overline{X}} \Uparrow \quad \begin{array}{c} (Q_\bullet, Q^\bullet) \\ {\scriptstyle coeq_\bullet}\downarrow \quad \uparrow{\scriptstyle coeq^\bullet} \\ ((C \oplus D)_\bullet, (C \oplus D)^\bullet) \end{array}$$

We also have a Z-split chain map and the Z-split code map $\mathcal{E}_{\overline{Z}}$ by taking the opposite.

$$\begin{array}{ccc} & & (Q_\bullet, Q^\bullet) \\ \mathcal{E}_{\overline{Z}} \Downarrow & & {\scriptstyle coeq_\bullet}\downarrow \quad \uparrow{\scriptstyle coeq^\bullet} \\ & & ((C \oplus D)_\bullet, (C \oplus D)^\bullet) \end{array}$$

This is rather abstract, so let's see a small concrete example.

Example 5.8. *Consider the following pushout of square lattices:*



We have not properly formalised pushouts of square lattices in the main body for brevity, but we do so in Appendix C. Informally, we are just 'gluing along' the graph in the top left corner, where the edges to be glued are coloured in blue.

We can consider this pushout to be in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$ [1]*, giving the pushout:*

$$\begin{array}{ccc} A_\bullet & \xrightarrow{g_\bullet} & D_\bullet \\ {\scriptstyle f_\bullet}\downarrow & \ulcorner & \downarrow{\scriptstyle q_\bullet} \\ C_\bullet & \xrightarrow[p_\bullet]{} & Q_\bullet \end{array}$$

with

$$A_\bullet = \mathbb{F}_2 \xrightarrow{\partial_1^{A\bullet}} \mathbb{F}_2^2$$

$$C_\bullet = \mathbb{F}_2 \xrightarrow{\partial_2^{C\bullet}} \mathbb{F}_2^3 \xrightarrow{\partial_1^{C\bullet}} \mathbb{F}_2^2$$

$$D_\bullet = \mathbb{F}_2 \xrightarrow{\partial_2^{D\bullet}} \mathbb{F}_2^3 \xrightarrow{\partial_1^{D\bullet}} \mathbb{F}_2^2$$

and

$$\partial_1^{A\bullet} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}; \quad \partial_2^{C\bullet} = \partial_2^{D\bullet} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}; \quad \partial_1^{C\bullet} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}; \quad \partial_1^{D\bullet} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

One can see from the cell complexes that we have

$$Q_\bullet = \mathbb{F}_2^2 \xrightarrow{\partial_2^{Q\bullet}} \mathbb{F}_2^5 \xrightarrow{\partial_1^{Q\bullet}} \mathbb{F}_2^2$$

[1] Categorically, this is because there is a cocontinuous functor from the appropriate category of square lattices to $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$.

with

$$\partial_2^{Q\bullet} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}; \quad \partial_1^{Q\bullet} = \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

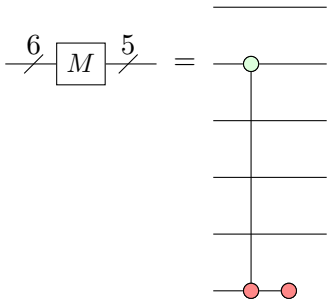
Rather than compute the pushout maps, let us instead give the coequaliser $coeq_\bullet$*:*

$$\begin{array}{ccccc} (C\oplus D)_2 & \xrightarrow{\partial_2^{(C\oplus D)\bullet}} & (C\oplus D)_1 & \xrightarrow{\partial_1^{(C\oplus D)\bullet}} & (C\oplus D)_0 \\ \downarrow{\scriptstyle coeq_2} & & \downarrow{\scriptstyle coeq_1} & & \downarrow{\scriptstyle coeq_0} \\ Q_2 & \xrightarrow{\partial_2^{Q\bullet}} & Q_1 & \xrightarrow{\partial_1^{Q\bullet}} & Q_0 \end{array}$$

We immediately see that $coeq_2 = \mathrm{id}$*. For the other two surjections we have*

$$coeq_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}; \quad coeq_0 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

Finally we interpret all the chain complexes in this pushout as being the Z-type complexes of CSS codes $(A_\bullet, A^\bullet)$*,* $(C_\bullet, C^\bullet)$ *etc. Thus we have a Z-merge code map* $\mathcal{F}_{\overline{Z}}$*, with an interpretation M as a* $\mathbb{C}$*-linear map, using* $coeq_1$ *and* $coeq_1^\intercal$*. We refrain from writing out the full* 32*-by-*64 *matrix, but as a ZX-diagram using gates from* $\{\mathrm{CNOT}, |+\rangle, \langle 0|\}$ *we have simply*



We know from Lemma 3.5 that this map must restrict to a map on logical qubits. However, easy calculations show that $H_1((C\oplus D)_\bullet) = 0$*, while* $H_1(Q_\bullet) = 1$*. That is, in the code* $((C\oplus D)_\bullet, (C\oplus D)^\bullet)$ *there are no logical qubits – there are still operators which show up as errors and some which don't, but all of those which don't are products of Z or X stabiliser generators. By Corollary 4.12 and Corollary 4.16 the logical map in* `FHilb` *is then just* $|+\rangle$*. This trivially preserves both* $\overline{Z}$ *and* $\overline{X}$ *operators, although its opposite code map* $\mathcal{F}_{\overline{X}}$ *does not preserve* $\overline{Z}$ *operators.*

This example was very simple, but the idea extends in quite a general way. To give an idea of how general this notion of CSS code surgery is, consider the balanced product codes from [4, 40, 41]. The balanced product of codes is by definition a coequaliser in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$, and so we can convert it into a pushout using routine category theory. The coequaliser is

$$(C\otimes A\otimes D)_\bullet \underset{g_\bullet}{\overset{f_\bullet}{\rightrightarrows}} (C\otimes D)_\bullet \xrightarrow{coeq_\bullet} (C\otimes_A D)_\bullet$$

where $g_\bullet$ and $f_\bullet$ represent left and right actions of $A_\bullet$ respectively; in all the cases from [4, 40, 41] these actions are basis-preserving. We have not explicitly defined the tensor product of chain complexes in the main body for brevity, but see Appendix B. Then to this coequaliser we can associate a pushout,

$$\begin{array}{ccc} ((C\otimes A\otimes D)\oplus(C\otimes D))_\bullet & \xrightarrow{(g_\bullet \mid \mathrm{id}_\bullet)} & (C\otimes D)_\bullet \\ {\scriptstyle (f_\bullet \mid \mathrm{id}_\bullet)}\downarrow & \ulcorner & \downarrow{\scriptstyle l_\bullet} \\ (C\otimes D)_\bullet & \xrightarrow[k_\bullet]{} & (C\otimes_A D)_\bullet \end{array}$$

where one can check that the universal property is the same in both cases. Thus we can think of a balanced product as a merge of tensor product codes, with the apex being two adjacent tensor product codes. As the maps in the span are evidently not monic, the merge is of a distinctly different sort from Example 5.8, and also the $\overline{Z}$- and $\overline{X}$-merges we will describe in Section 5.3.

It would be convenient if we could guarantee some properties of pushouts in general; for example, if the pushout of LDPC codes was also LDPC, or if the homologies were always preserved. Unfortunately, the definition is general enough that neither of these are true. We discuss this in slightly greater detail in Appendix D, but the gist is that we need to stipulate some additional conditions to guarantee bounds on these quantities.

5.3 Surgery along a logical operator

The procedure of merging here is closely related to that of 'welding' in [37]. Our focus is not just on the resultant codes, but the maps on physical and logical data. On codes generated from square lattices, the merges here will correspond to a pushout along a 'string' through the lattice.

Definition 5.9. *Let* $C_\bullet = C_2 \xrightarrow{\partial_1} C_1 \xrightarrow{\partial_1} C_0$ *be a length 2 chain complex. Let* $v \in C_1$ *be a vector such that* $v \in \ker(\partial_1^{C\bullet})\backslash\mathrm{im}(\partial_2^{C\bullet})$*. We now construct the* logical operator subcomplex $V_\bullet$*. This has:*

$$\tilde{V}_1 = \mathrm{supp}\ v; \quad \partial_1^{V\bullet} = \partial_1^{C\bullet}\restriction_{\mathrm{supp}\ v}; \quad \tilde{V}_0 = \bigcup_{u\in\mathrm{im}(\partial_1^{V\bullet})} \mathrm{supp}\ u$$

where supp v *is the set of basis vectors in the support of* v*, and* $\partial_i\restriction_S$ *is the restriction of a differential to a subset* S *of its domain. All other components and differentials of* $V_\bullet$ *are zero.*

There is a monic $f_\bullet : V_\bullet \hookrightarrow C_\bullet$ given by the inclusion maps of $V_1 \subseteq C_1$ etc.

Definition 5.10. *Let* $V_\bullet$ *be a logical operator subcomplex of two chain complexes*

$$C_\bullet = C_2 \xrightarrow{\partial_2} C_1 \xrightarrow{\partial_1} C_0$$

and

$$D_\bullet = D_2 \xrightarrow{\partial_2} D_1 \xrightarrow{\partial_1} D_0$$

simultaneously, so there is a basis-preserving monic span

$$\begin{array}{ccc} V_\bullet & \xhookrightarrow{g_\bullet} & D_\bullet \\ {\scriptstyle f_\bullet}\downarrow & & \\ C_\bullet & & \end{array}$$

This monic span has the pushout

$$\begin{array}{ccc} V_\bullet & \xrightarrow{g_\bullet} & D_\bullet \\ {\scriptstyle f_\bullet}\downarrow & \ulcorner & \downarrow{\scriptstyle q_\bullet} \\ C_\bullet & \xrightarrow[p_\bullet]{} & Q_\bullet \end{array}$$

with components

$$Q_2 = C_2 \oplus D_2; \quad Q_1 = C_1 \oplus D_1/(\text{supp } v \sim \text{supp } w); \quad Q_0 = C_0 \oplus D_0/(\tilde{f}_0 \sim \tilde{g}_0),$$

where w is the logical operator associated to $\text{im}(g_1) \in D_1$.

The construction here is inspired by [12].

Definition 5.11. *Let $(C_\bullet, C^\bullet)$ and $(D_\bullet, D^\bullet)$ be CSS codes. Let $(V_\bullet, V^\bullet)$ be a CSS code such that $V_\bullet$ is a logical operator subcomplex of $C_\bullet$ and $D_\bullet$; this means that $(V_\bullet, V^\bullet)$ can be seen as merely a classical code, as $V_2 = 0$. Then the Z-type merged CSS code $(Q_\bullet, Q^\bullet)$ is called the* $\overline{Z}$-merged code *of $(C_\bullet, C^\bullet)$ and $(D_\bullet, D^\bullet)$ along $(V_\bullet, V^\bullet)$.*

Definition 5.12. (Separation) *Let $V_\bullet$ be a logical operator subcomplex such that for the inclusion maps $f_\bullet$ and $g_\bullet$,* $\text{im}(f_1)$ *and* $\text{im}(g_1)$ *contain only one operator in $Z_1(C_\bullet)$, $Z_1(D_\bullet)$ respectively, with those operators being v, w. Then we say that these operators are separated, as they contain no other logicals in their support, and the pushout satisfies the separation property.*

The intuition here, following [12], is that it is convenient when the logical operators we glue along do not themselves contain any nontrivial logical operators belonging to a different logical qubit; if they do, the gluing procedure may yield a more complicated output code, as we could be merging along multiple logical operators simultaneously. In Appendix E we demonstrate that it is possible for this condition to not be satisfied, using a patch of octagonal surface code. Additionally, we do not want the gluing procedure to send any logical $\overline{Z}$ operators to stabilisers.

We would like to study not only the resultant code given some $\overline{Z}$-merge, but also the map on the logical space. We can freely switch between pushouts and coequalisers. Recall the Z-merge code map $\mathcal{F}_{\overline{Z}}$ from Equation 1. We call this a $\overline{Z}$-merge code map when the merge is along a $\overline{Z}$-operator as above, and from now on we assume that all merges are separated.

Lemma 5.13. *Let $(Q_\bullet, Q^\bullet)$ be a separated $\overline{Z}$-merged code with parameters $[\![n_Q, k_Q, d_Q]\!]$, and let $[\![n_C, k_C, d_C]\!]$, $[\![n_D, k_D, d_D]\!]$ be the parameters of $(C_\bullet, C^\bullet)$ and $(D_\bullet, D^\bullet)$ respectively. Let $n_V = \dim V_0$. Then*

$$n_Q = n_C + n_D - n_V; \quad k_Q \geq k_C + k_D - 1$$

Further, let $\{[u]_i\}$ and $\{[v]_j\}$ be the bases for $H_1(C_\bullet)$ and $H_1(D_\bullet)$ respectively, and say w.l.o.g. that $u \in [u]_1$ and $v \in [v]_1$ are the vectors quotiented by the pushout. Then $H_1(Q_\bullet)$ has a basis $\{[w]_l\}$ for $l \leq k_Q$, where $[w]_1 = [u]_1 = [v]_1$, $[w]_l = [u]_l$ when $1 < l \leq k_C$ and $[w]_l = [v]_{l-k_C+1}$ for $k_C < l \leq k_C + k_D - 1$.

Proof. n_Q is immediate by the definition. Given $u \in [u]_1$ and $v \in [v]_1$, any other representatives $y \in [u]_1$, $x \in [v]_1$ belong to the same equivalence class in $H_1(Q_\bullet)$, as $y \sim u \sim v \sim x$.

All other equivalence classes remain distinct, as they would be in $((C \oplus D)_\bullet, (C \oplus D)^\bullet)$.

However, it is possible to introduce new equivalence classes, without a preimage in $H_1((C \oplus D)_\bullet)$. Despite $coeq_\bullet$ being surjective, the lift $H_1(coeq_\bullet)$ is not always surjective, as the restriction of $coeq_1$ to $\ker(\partial_1^{(C\oplus D)_\bullet})$ is not always surjective. □

This last case is subtle, and rarely occurs with small codes or topological codes. We present an explicit example in Appendix G. Should it be useful, we can swap to subsystem codes after the merge, and not store any data in the new logical qubits, relegating them to gauge qubits, as done in [12].

Lemma 5.14. *Let the $\overline{Z}$-merge code map*

$$\begin{array}{ccc} & & ((C\oplus D)_\bullet, (C\oplus D)^\bullet) \\ \mathcal{F}_{\overline{Z}} \Big\Downarrow & & {\scriptstyle coeq_\bullet}\downarrow \quad \uparrow{\scriptstyle coeq^\bullet} \\ & & (Q_\bullet, Q^\bullet) \end{array}$$

of a separated $\overline{Z}$-merged code have its interpretation M as a $\mathbb{C}$-linear map. Then M acts as

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

on each pair of qubits in $((C\oplus D)_\bullet, (C\oplus D)^\bullet)$ which are equivalent in $(Q_\bullet, Q^\bullet)$ and M acts as identity on all other qubits.

Proof. M must have the following maps on Paulis on each pair of qubits being merged:

$$Z\otimes I \mapsto Z; \quad I\otimes Z \mapsto Z; \quad X\otimes X \mapsto X$$

which uniquely defines the matrix above. In other words we have $|00\rangle \mapsto |0\rangle$, $|11\rangle \mapsto |1\rangle$, $|01\rangle \mapsto |0\rangle, |10\rangle \mapsto |0\rangle$ etc, which has the convenient presentation as the ZX diagram on the left above. □

Lemma 5.15. *Let $(Q_\bullet, Q^\bullet)$ be a separated $\overline{Z}$-merged code of $(C_\bullet, C^\bullet)$ and $(D_\bullet, D^\bullet)$ along $(V_\bullet, V^\bullet)$. Call $f = H_1(coeq_\bullet)$. Then*

$$f([u]_i + [v]_j) = [w]_l$$

where $[w]_l$ was defined in Lemma 5.13.

This is obvious by considering the surjection in question and using Lemma 5.13. It essentially says that on the pair of logical operators in $((C\oplus D)_\bullet, (C\oplus D)^\bullet)$ which are being quotiented together, $\mathcal{F}_{\overline{Z}}$ acts as:

$$\overline{Z}\otimes\overline{I} \mapsto \overline{Z}; \quad \overline{I}\otimes\overline{Z} \mapsto \overline{Z}; \quad \overline{X}\otimes\overline{X} \mapsto \overline{X}$$

where the map on Xs is inferred from the dual. In the case where new logical qubits are introduced, as described in Lemma 5.13, it can be easily checked that these are initialised in the logical $|+\rangle$ state, as they are not in the image of the $H_1(coeq_\bullet)$.

Lemma 5.16. *Let the merged code have no new logical qubits, i.e. $k_Q = k_C + k_D - 1$. Then,*

$$d_Q^X \geq \min(d_C^X, d_D^X)$$

Proof. By considering the code map $\mathcal{F}_{\overline{Z}}$, we see that any $\overline{X}$ logical operator u in $(Q_\bullet, Q^\bullet)$ has a preimage w which is also an $\overline{X}$ logical operator in $((C\oplus D)_\bullet, (C\oplus D)^\bullet)$, s.t. $|w| \leq |u|$. This is because $coeq^\bullet$ can be restricted to $f^\intercal = H^1(coeq^\bullet)$, and any logical in $((C\oplus D)_\bullet, (C\oplus D)^\bullet)$ has sublogicals in $(C_\bullet, C^\bullet)$ and $(D_\bullet, D^\bullet)$. □

The proof fails when the merged code has new logical qubits, as there can be a $\overline{X}$ logical operator u whose image under $coeq^\bullet$ is in [0], which gives no bound on the weight of u.

Remark 5.17. *Note that we do not in general have a lower bound on d_Q^Z in terms of d_C^Z and d_D^Z. We can see this from the discussion in Section 4.3. Given the code map $\mathcal{F}_{\overline{Z}}$, the chain map $f_1 : (C \oplus D)_1 \to Q_1$ restricts to $H_1(f)$, but this does not preclude there being other vectors in $(C \oplus D)_1 \setminus \ker \partial_1^{(C\oplus D)_\bullet}$ which are mapped into one of the equivalence classes in $H_1(Q_\bullet)$. In computational terms, while we cannot have detectable X operators in the initial codes which are mapped to logicals by the code map $\mathcal{F}_{\overline{Z}}$, this is unfortunately possible with detectable Z operators. We illustrate this with an example in Appendix F.*

We now show that, if we consider two codes to be merged as instances of LDPC families, their combined $\overline{Z}$-merged code code is also LDPC. Recall Definition 4.3.

Lemma 5.18. (LDPC) *Say our input codes $(C_\bullet, C^\bullet)$, $(D_\bullet, D^\bullet)$ have maximal weights of generators labelled w_C^Z, w_C^X and w_D^Z, w_D^X respectively. Let $(Q_\bullet, Q^\bullet)$ be a separated $\overline{Z}$-merged code of $(C_\bullet, C^\bullet)$ and $(D_\bullet, D^\bullet)$ along $(V_\bullet, V^\bullet)$. Then*

$$w_Q^Z = \max(w_C^Z, w_D^Z); \quad w_Q^X < w_C^X + w_D^X.$$

Similarly, letting the input codes have maximal number of shared generators on a single qubit q_C^Z, q_C^X and q_D^Z, q_D^X we have

$$q_Q^Z \leq q_C^Z + q_D^Z; \quad q_Q^X = \max(q_C^X, q_D^X)$$

Proof. None of the Z-type generators are quotiented by a $\overline{Z}$-merge map, so $w_Q^Z = w_{(C\oplus D)}^Z = \max(w_C^Z, w_D^Z)$. For the X-type generators, in the worst case the two generators which are made to be equivalent by the merge are the highest weight ones. For these generators to appear in V_0 they must have at least two qubits in each of their support which is in V_1, and thus these qubits are merged together, so $w_Q^X < w_C^X + w_D^X$.

Next, using again the fact that none of the Z-type generators are quotiented, a single qubit could in the worst case be the result of merging two qubits in $(C_\bullet, C^\bullet)$ and $(D_\bullet, D^\bullet)$ which each have the maximal number of shared Z-type generators, so $q_Q^Z \leq q_C^Z + q_D^Z$. For the X case, if a qubit is in V_1 then all X-type generators it is in the support of must appear in V_0. Therefore, when any two qubits are merged all of their X-type generators are also merged. Thus $q_Q^X = q_{(C\oplus D)}^X = \max(q_C^X, q_D^X)$. □

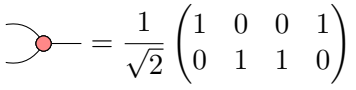
Note that as w^Z, w^X and q^Z, q^X are at worst additive in those of the input codes, the $\overline{Z}$-merge of two LDPC codes is still LDPC, assuming the pushout is still well-defined using matching $\overline{Z}$ operators for each member of the code families. Next, we dualise everything, and talk about $\overline{X}$-merges.

Definition 5.19. *Let $(C_\bullet, C^\bullet)$ and $(D_\bullet, D^\bullet)$ be CSS codes. Let $(V_\bullet, V^\bullet)$ be a CSS code such that $V^\bullet$ is a logical operator subcomplex of $C^\bullet$ and $D^\bullet$, and $Q^\bullet$ is the merged complex along $V^\bullet$. Then the CSS code $(Q_\bullet, Q^\bullet)$ is called the $\overline{X}$-*merged code *of $(C_\bullet, C^\bullet)$ and $(D_\bullet, D^\bullet)$ along $(V_\bullet, V^\bullet)$.*

In this case we glue along an $\overline{X}$ logical operator instead. The notions of separation, Lemma 5.13 and Lemma 5.18 carry over by transposing appropriately.

An $\overline{X}$-merge map $\mathcal{E}_{\overline{X}}$ can be defined similarly, and a similar result as Lemma 5.14 applies to separated $\overline{X}$-merged codes.

Lemma 5.20. *Let the $\overline{X}$-merge code map of a separated $\overline{X}$-merged code have its interpretation L as a $\mathbb{C}$-linear map. Then L acts as*



$$= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

on each pair of qubits in $((C \oplus D)_\bullet, (C \oplus D)^\bullet)$ which are equivalent in $(Q_\bullet, Q^\bullet)$, i.e. $|{+}{+}\rangle \mapsto |{+}\rangle$, $|{-}{-}\rangle \mapsto |{-}\rangle$, and L acts as identity on all other qubits.

Proof. This time, L must have the maps

$$X \otimes I \mapsto X; \quad I \otimes X \mapsto X; \quad Z \otimes Z \mapsto Z$$

□

Similarly, the maps on logical operators are

$$\overline{X} \otimes \overline{I} \mapsto \overline{X}; \quad \overline{I} \otimes \overline{X} \mapsto \overline{X}; \quad \overline{Z} \otimes \overline{Z} \mapsto \overline{Z}$$

and, when new logical qubits are generated, they are initialised in the $|0\rangle$ state.

Having discussed $\overline{Z}$- and $\overline{X}$-merged codes, we briefly mention splits. These are just the opposite code maps to $\mathcal{F}_{\overline{Z}}$ and $\mathcal{E}_{\overline{X}}$. In both cases, all the mappings are determined entirely by Lemma 5.13 by taking transposes or adjoints when appropriate.

Remark 5.21. *In practice, when the CSS codes in question hold multiple logical qubits it may be preferable to merge/split along multiple disjoint $\overline{Z}$ or $\overline{X}$ operators at the same time. Such a protocol is entirely viable within our framework, and requires only minor tweaks to the above results. The same is true should one wish to merge/split along operators within the same code.*

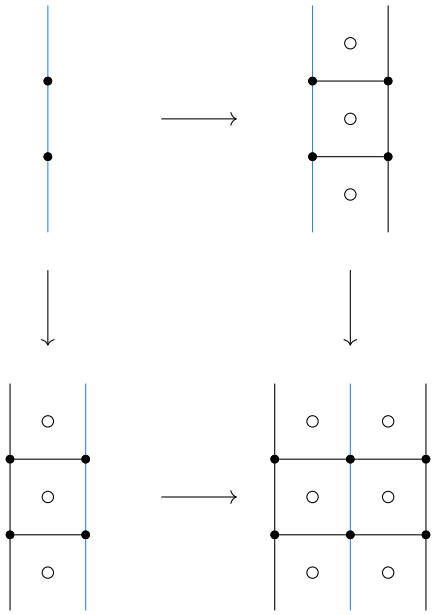
We now look at a short series of examples.

5.4 Examples of surgery

5.4.1 Lattice surgery

Lattice surgery is the prototypical instance of CSS code surgery. It starts with patches of surface code and then employs separated splits and merges to perform non-unitary logical operations [26]. The presentation we give of lattice surgery is idiosyncratic, in the sense that we perform the merges on physical edges/qubits, whereas the standard method is to introduce additional edges between patches to join them together. We remedy this in Section 6.

Consider the pushout of cell complexes below:



As before, we informally consider this to be 'gluing along' the graph in the top left, but for completeness it is formalised in Appendix C. By considering the pushout to be in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$, we have:

$$\begin{array}{ccc} V_\bullet & \xhookrightarrow{g_\bullet} & D_\bullet \\ {\scriptstyle f_\bullet}\downarrow & \ulcorner & \downarrow{\scriptstyle q_\bullet} \\ C_\bullet & \xrightarrow[p_\bullet]{} & Q_\bullet \end{array}$$

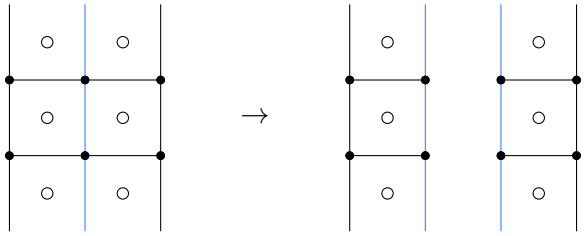
Letting $coeq_\bullet : (C \oplus D)_\bullet \to Q_\bullet$ be the relevant coequaliser map, we see that $\mathcal{F}_{\overline{Z}} = (coeq_\bullet, coeq^\bullet)$ constitutes a separated $\overline{Z}$-merge map. In particular, observe that $\mathcal{F}_{\overline{Z}}$ sends the logical operators:

$$\begin{aligned} \overline{Z} \otimes \overline{I} &\mapsto \overline{Z} \\ \overline{I} \otimes \overline{Z} &\mapsto \overline{Z} \\ \overline{X} \otimes \overline{X} &\mapsto \overline{X} \end{aligned}$$

as predicted by Lemma 5.15.

The first two give $H_1(coeq_\bullet) = \begin{pmatrix} 1 & 1 \end{pmatrix}$ and the last $H^1(coeq^\bullet) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$. $\mathcal{F}_{\overline{Z}}$ is evidently $\overline{Z}$-preserving but not $\overline{X}$-preserving, as $\overline{X} \otimes \overline{I}$ is taken to an operation which is detected by the Z stabilisers. Observe that we end up with a greater cosystolic distance of $(Q_\bullet, Q^\bullet)$ than we started with in $((C \oplus D)_\bullet, (C \oplus D)^\bullet)$.

If we instead consider the pair $(coeq_\bullet, coeq^\bullet)$ as an $\overline{X}$-preserving code map $\mathcal{F}_{\overline{X}}$, then it is a separated $\overline{X}$-split map. In terms of cell complexes we would have [2]

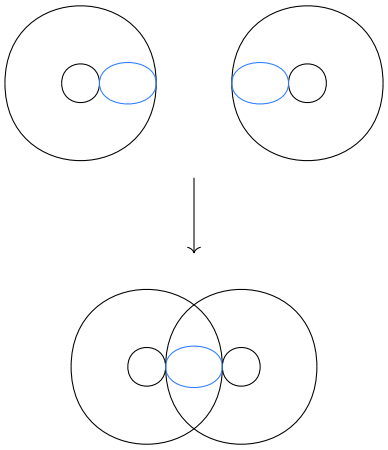


[2] Pedantically, this is a morphism in the opposite category of cell complexes $\mathtt{OACC}^{\mathrm{op}}$.

We similarly have a separated $\overline{X}$-merge map and separated $\overline{Z}$-split map with the obvious forms by dualising appropriately.

Remark 5.22. *While it is convenient to choose logical operators along patch boundaries to glue along, so that the complexes can all be embedded on the 2D plane, this is not necessary. One could intersect two patches along any matching operator.*

Recall the toric code from Example 4.7. We can merge two copies of the code along a logical $\overline{Z}$ operator, which corresponds to an essential cycle of each torus. The resultant code will then look like two tori intersecting, depending somewhat on the choices of essential cycle:



The $\overline{Z}$-merge map on logical qubits will be the same as for patches.

5.4.2 Shor code surgery

Of course, the pushout we take does not have to come from square lattices. Let $C_\bullet$ and $D_\bullet$ be two copies of Shor codes from Example 4.6.[3] We can perform merges between them. We give two examples. First, for a $\overline{Z}$-merge, we take the logical $\overline{Z}$ operator $\overline{Z} = \bigotimes_i^8 Z_i$ and apply Definition 5.9 to get the logical operator subcomplex:

$$V_\bullet = V_1 \xrightarrow{P_X} V_0$$

with $V_1 = \mathbb{F}_2^9$, $V_0 = \mathbb{F}_2^2$, and all other components zero. This is just $C_\bullet$ from Example 4.6 truncated to be length 1, as this logical $\overline{Z}$ operator has support on all physical qubits. The monic chain map $f_\bullet$ given by inclusion into the Shor code is just

$$\begin{array}{ccccc} 0 & \xrightarrow{0} & V_1 & \xrightarrow{P_X} & V_0 \\ \downarrow{\scriptstyle 0} & & \downarrow{\scriptstyle \mathrm{id}} & & \downarrow{\scriptstyle \mathrm{id}} \\ C_2 & \xrightarrow{P_Z^\intercal} & C_1 & \xrightarrow{P_X} & C_0 \end{array}$$

and the same for $g_\bullet$. The pushout of

$$\begin{array}{ccc} V_\bullet & \xhookrightarrow{g_\bullet} & D_\bullet \\ {\scriptstyle f_\bullet}\downarrow & & \\ C_\bullet & & \end{array}$$

[3] The Shor code can be constructed as a cellulation of the projective plane, so it is actually not wholly dissimilar from the lattice codes [22].

will then be

$$Q_\bullet = \mathbb{F}_2^{12} \xrightarrow{\partial_2^{Q\bullet}} \mathbb{F}_2^{9} \xrightarrow{\partial_1^{Q\bullet}} \mathbb{F}_2^{2}$$

where $\partial_1^{Q\bullet} = P_X$ and $\partial_2^{Q\bullet} = \left(P_Z^\intercal | P_Z^\intercal\right)$. We have ended up with virtually the same code as the Shor code, except that we have a duplicate for every Z-type generator, i.e. every measurement of Z stabilisers is performed twice and the result noted separately. While this example is very simple, it highlights that the result of a merge can have somewhat subtle features, such as duplicating measurements, which the two input codes do not. This merge did not use separated logical operators.

For our second case, we use a different (but equivalent) logical operator, $\overline{Z} = Z_1 \otimes Z_4 \otimes Z_7$. We still glue two copies of the Shor code, but now we have $V_1 = \mathbb{F}_2^3$, $V_0 = \mathbb{F}_2^2$ and $\partial_1^{V\bullet} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$. That is, our logical operator subcomplex is just the repetition code from Example 4.1. We then have

$$\begin{array}{ccccc} 0 & \xrightarrow{0} & V_1 & \xrightarrow{\partial_1^{V\bullet}} & V_0 \\ \downarrow{\scriptstyle 0} & & \downarrow{\scriptstyle f_1} & & \downarrow{\scriptstyle \mathrm{id}} \\ C_2 & \xrightarrow{P_Z^\intercal} & C_1 & \xrightarrow{P_X} & C_0 \end{array}$$

where

$$f_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

and the same for g_1, forming again a monic span of chain complexes. The resultant $\overline{Z}$-merged code is then

$$Q_\bullet = \mathbb{F}_2^{12} \xrightarrow{\partial_2^{Q\bullet}} \mathbb{F}_2^{15} \xrightarrow{\partial_1^{Q\bullet}} \mathbb{F}_2^{2}$$

and the large matrices $\partial_2^{Q\bullet}$ and $\partial_1^{Q\bullet}$ are easily obtained by quotienting out rows and columns from $\partial_2^{C\bullet} \oplus \partial_2^{D\bullet}$ and $\partial_1^{C\bullet} \oplus \partial_1^{D\bullet}$.

Remark 5.23. *The protocols for performing $\overline{X}$ and $\overline{Z}$ measurements by generalised lattice surgery in [12] can be seen as using separated $\overline{Z}$- and $\overline{X}$-merged codes, with the caveat that they don't perform the merge maps; instead they initialise fresh qubits in the ancillary hypergraph code and measure all stabiliser generators. The present work has overlap with their protocols, but we do not subsume them; for example their $\overline{X} \otimes \overline{Z}$ and $\overline{Y}$ measurement methods are outside of our formalism, as they lead to non-CSS codes.*

6 Error-corrected logical operations

We now describe how our abstract formalism leads to a general set of error-corrected logical operations for CSS codes. We consider this to be a good application of the homological algebraic formalism, as we suspect these logical operations would be challenging to derive

without the machinery of $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$. [4] So far in our description of code maps there are two main assumptions baked in: that one can perform linear maps between CSS codes (a) deterministically and (b) while maintaining error-correction, both of which are desired for performing quantum computation.

For assumption (a), we can only implement code maps which are interpreted as an isometry deterministically. If they are not, instead we must perform measurements on physical qubits. Recall from Proposition 4.11 that every code map has an interpretation constructed from CNOTs and some additional states and effects taken from $\{|+\rangle, \langle 0|\}$ for a $\overline{Z}$-preserving code map or $\{\langle +|, |0\rangle\}$ for an $\overline{X}$-preserving code map. This means that in order to implement the code map non-deterministically, one need only apply CNOTs and measure some qubits in the Z-basis (for a $\overline{Z}$-preserving code map) or the X-basis ($\overline{X}$-preserving code map). Of course, should we acquire the undesired measurement result, we induce errors in our code map. There is no protocol for correcting these errors in all generality. For assumption (b), there is no protocol for performing arbitrary CNOT circuits on physical qubits in a code fault-tolerantly. However, when performing CSS code surgery which is a separated $\overline{Z}$- or $\overline{X}$-merge, we have a protocol which addresses both (a) and (b).

6.1 Procedure summary

Our procedure for performing an error-corrected $\overline{Z} \otimes \overline{Z}$ measurement is as follows:

1. Find a matching $\overline{Z}$ logical operator which belongs to both initial codes, in the sense of Definition 5.9.
2. Verify that this logical operator satisfies the separation property of Definition 5.12 in both codes.
3. Verify that the merge is bounded below, in the sense of Definition 6.8 below.
4. Perform the merge as described in Proposition 6.10.

We do not know how difficult it will be in general to perform the verification in steps (2) and (3) for codes (or families of codes) of interest.

6.2 Full description of procedure

We will first describe gauge fixing. Luckily this does not become an additional condition, as we will show it coincides precisely with separation. For reasons of brevity we do not describe the connection between lattice surgery and gauge fixing, but refer the interested reader to [44]. Briefly, we will consider the whole system to be a subsystem code, and fix the gauges of the $\overline{Z}$ operators we are gluing along.

Definition 6.1. *Let $C_\bullet$ be a chain complex and u be a representative of the equivalence class $[u] \in H_1(C_\bullet)$, which is a basis vector of $H_1(C_\bullet)$. Let x be a vector in C_1 such that $|x| = 1$ and $x \cdot u = 1$. We say that x is a qubit in the support of u. Recall from Lemma 4.4 that u has a unique paired basis vector $[v] \in H^1(C^\bullet)$ such that $[u] \cdot [v] = 1$. It is possible to* safely correct *a qubit x when there is a vector $v \in [v]$ such that $x \cdot v = 1$ and $y \cdot v = 0$ for all other qubits y in the support of u. We say that u is* gauge-fixable *when it is possible to safely correct all qubits in the support of u.*

[4] An alternative approach could be to use Tanner graphs.

Example 6.2. *Consider the Shor code from Example 4.6 and Section 5.4.2. The $\overline{Z}$ operator*

$$v = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}^{\mathsf{T}}$$

has qubits in its support for which it is not possible to safely correct, as there are only 4 representatives of the nonzero equivalence class $[w] \in H^1(C^\bullet)$ but 9 qubits for which being able to safely correct is necessary. However, it is possible to safely correct all qubits in the support of the $\overline{Z}$ operator

$$u = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}^{\mathsf{T}},$$

where $u \in [v]$, with the fixing operators:

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^{\mathsf{T}}; \quad \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix}^{\mathsf{T}}; \quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}^{\mathsf{T}}$$

The same definition of gauge-fixability applies if we exchange X and Z appropriately.

Lemma 6.3. *Every separated logical operator is also gauge-fixable, and vice versa.*

Proof. See Appendix H. □

Next we will require the tensor product of chain complexes, for which see Appendix B.

Definition 6.4. *Let $V_\bullet = \; V_1 \longrightarrow V_0$ and $P_\bullet = \; P_1 \xrightarrow{\begin{pmatrix} 1 \\ 1 \end{pmatrix}} P_0$ be length 1 chain complexes. Then we can make the tensor product chain complex $W_\bullet = (P \otimes V)_\bullet$. Explicitly,*

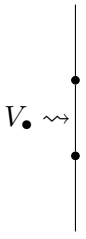
$$W_\bullet = \; W_2 \longrightarrow W_1 \longrightarrow W_0$$

with

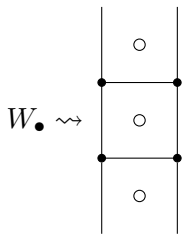
$$W_2 = P_1 \otimes V_1 = V_1; \quad W_1 = (P_0 \otimes V_1) \oplus (P_1 \otimes V_0) = (\mathbb{F}_2^2 \otimes V_1) \oplus V_0; \quad W_0 = P_0 \otimes V_0 = \mathbb{F}_2^2 \otimes V_0$$

Also, $\partial_2^{W_\bullet} = \begin{pmatrix} \mathrm{id}_{V_1} \\ \mathrm{id}_{V_1} \\ \partial_1^{V_\bullet} \end{pmatrix}$ and $\partial_1^{W_\bullet} = \begin{pmatrix} \mathrm{id}_{\mathbb{F}_2^2} \otimes \partial_1^{V_\bullet} & \partial_1^{P_\bullet} \otimes \mathrm{id}_{V_0} \end{pmatrix} = \begin{pmatrix} \partial_1^{V_\bullet} & 0 & \mathrm{id}_{V_0} \\ 0 & \partial_1^{V_\bullet} & \mathrm{id}_{V_0} \end{pmatrix}$.

In the case where $V_\bullet$ is a string along a patch of surface code, say of the form:



then $W_\bullet$ will be of the form



as a square lattice, see Definition C.8. We can see this as the 'intermediate section' used to perform lattice surgery.

Lemma 6.5. *Let $V_\bullet$ be a $\overline{Z}$ logical operator subcomplex of a chain complex $C_\bullet$, and let $V_\bullet$ satisfy the separation property from Definition 5.12. Then*

$$w_W^X = w_C^X + 1; \quad w_W^Z = q_C^X + 2; \quad q_W^X = \max(q_C^X, 2); \quad q_W^Z = w_C^X$$

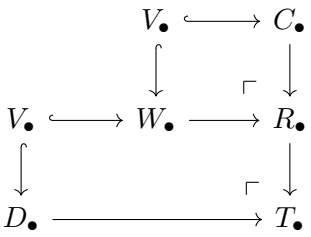
and $\dim H_1(W_\bullet) = \dim H_1(V_\bullet) = 1$.

Proof. Observe that $\partial_1^{V_\bullet}$ has maximum row weight w_C^X and column weight q_C^X. Then inspect the matrices $\partial_2^{W_\bullet}$ and $\partial_1^{W_\bullet}$ from Definition 4.3. For $\dim H_1(W_\bullet)$, we use the Künneth formula, for which see Lemma B.3, which in this case says $H_1((P \otimes V)_\bullet) = (H_0(P_\bullet) \otimes H_1(V_\bullet)) \oplus (H_1(P_\bullet) \otimes H_0(V_\bullet))$. We then have

$$\dim H_0(P_\bullet) = 1; \quad \dim H_1(P_\bullet) = 0; \quad \dim H_0(V_\bullet) = 0; \quad \dim H_1(V_\bullet) = 1$$

where the last comes from the fact that $V_0 = \mathrm{im}(\partial_1^{V_\bullet})$, using Definition 5.9. $\dim H_1(V_\bullet) = 1$ as $B_1(V_\bullet) = 0$ and $Z_1(V_\bullet) = 1$. □

Definition 6.6. *Let $V_\bullet$ be a simultaneous $\overline{Z}$ logical operator subcomplex of both $C_\bullet$ and $D_\bullet$, satisfying the separation property. Then define the 'sandwiched code' $(T_\bullet, T^\bullet)$, with $T_\bullet$ as the pushout of a pushout:*



where the middle term is $W_\bullet = (P \otimes V)_\bullet$ from Definition 6.4 above, and the two inclusion maps $V_\bullet \hookrightarrow W_\bullet$ map V_1 into each of the copies of V_1 in W_1, and the same for V_0. All maps in the pushouts are basis-preserving, and one can check that they are all monic.

Colloquially, we are gluing first one side of the code $W_\bullet$ to $C_\bullet$, and then the other side to $D_\bullet$. [5]

Lemma 6.7. *The 'sandwiched code' $(T_\bullet, T^\bullet)$ has*

$$n_T = n_C + n_D + r; \quad k_T \geq k_C + k_D - 1$$

and

$$w_T^X \leq w_{C\oplus D}^X + 1; \quad w_T^Z \leq \max(w_{C\oplus D}^Z, q_{C\oplus D}^X + 2); \quad q_T^Z \leq q_{C\oplus D}^Z + w_{C\oplus D}^X; \quad q_T^X = \max(q_{C\oplus D}^X, 2).$$

If $k_T = k_C + k_D - 1$ then $d_T^X \geq \min(d_C^X, d_D^X)$.

Proof. For n_T, just apply Lemma 5.13 twice. For k_T, use Lemma 6.5 and apply Lemma 5.13 twice.

For d_T^X, we first show that $(R_\bullet, R^\bullet)$ has $d_R^X \geq d_C^X$. Every $\overline{X}$ operator in $(W_\bullet, W^\bullet)$ must anticommute with the $\overline{Z}$ operator used to construct $V_\bullet$, and thus must have support on those qubits. In addition, it must have a matched $\overline{X}$ operator in $(C_\bullet, C^\bullet)$, which also has support on those qubits. As the only other $\overline{X}$ operators in $(R_\bullet, R^\bullet)$ are those

[5] We could equally do it the other way, in which case the two pushouts would be flipped, but this does not change $T_\bullet$.

in $(C_\bullet, C^\bullet)$ which are unaffected by the merge, having no support on the qubits being merged, $d_R^X \geq d_C^X$. Then, if $k_T = k_C + k_D - 1$, $d_T^X \geq \min(d_C^X, d_D^X)$ using Lemma 5.16.

For w_T^X, the pushouts will glue each X type stabiliser generator in $W_\bullet$ into those in $C_\bullet$ and $D_\bullet$ in such a way that they will have exactly one extra qubit in the support, by the product construction of $W_\bullet$; we can see this from $\partial_1^{W_\bullet}$ in Definition 6.4, as there is exactly a single 1 which is not part of the $\partial_1^{V_\bullet}$ in any given row of the matrix.

For w_T^Z, q_T^Z and q_T^X we just use Lemma 6.5 and apply Lemma 5.18 twice. □

The intuition here is that rather than gluing two codes $(C_\bullet, C^\bullet)$ and $(D_\bullet, D^\bullet)$ together directly along a logical operator, we have made a low distance hypergraph code $(W_\bullet, W^\bullet)$ and used that to sandwich the codes. A consequence of the above lemma is that this 'sandwiching' procedure maps LDPC codes to LDPC codes. Importantly, under suitable conditions the two pushouts let us perform a code map on logical qubits in an error-corrected manner.

Definition 6.8. *Let $(T_\bullet, T^\bullet)$ have no logical $\overline{Z}$ operators with weight lower than $d_{C\oplus D}$. Then we say that the merged code has* distance bounded below*.*

Remark 6.9. *Note that the only Z operators which can lower the distance are those with support on the logical $\overline{Z}$ which is used to construct $V_\bullet$, as all others will be unchanged by the quotient. The condition for a merge to have distance bounded below is quite a tricky one, as we do not know of a way to check this easily. Because of Lemma 6.7, this problem is isolated to $\overline{Z}$ operators, as the distance is guaranteed to be bounded below for $\overline{X}$ operators.*

Proposition 6.10. *Let $(C_\bullet, C^\bullet)$ and $(D_\bullet, D^\bullet)$ be CSS codes which share a separated $\overline{Z}$ operator on m physical qubits and r X-type stabiliser generators each; let the relevant logical qubits be i and j, and let $V_\bullet$ be the logical operator subcomplex of $C_\bullet$ and $D_\bullet$ such that the codes admit a separated $\overline{Z}$-merge. Further, let d be the code distance of $((C \oplus D)_\bullet, (C \oplus D)^\bullet)$, and let the merged code $(T_\bullet, T^\bullet)$ have distance bounded below. Then there is an error-corrected procedure with distance d for implementing a $\overline{Z} \otimes \overline{Z}$ measurement on the pair i, j of logical qubits, which gives the code $(T_\bullet, T^\bullet)$. This procedure requires r auxiliary clean qubits and an additional m Z-type stabiliser generators.*

Proof. We aim to go from the code $((C \oplus D)_\bullet, (C \oplus D)^\bullet)$ to $(T_\bullet, T^\bullet)$. The code map we apply to physical qubits is as follows. We call the physical qubits in the support of the logical operators to be glued together the *participating* qubits. We initialise a fresh qubit in the $|+\rangle$ state for each pairing of X-measurements on the two logical operators of qubits i and j, that is for each qubit in $(W_\bullet, W^\bullet)$ which is not glued to a qubit in $(C_\bullet, C^\bullet)$ or $(D_\bullet, D^\bullet)$.

We now modify the stabilisers to get to $(T_\bullet, T^\bullet)$. To start, change the X stabiliser generators with support on the participating qubits to have one additional fresh qubit each, so that each pairing of X-measurements shares one fresh qubit. We add a new Z stabiliser generator with weight $a + 2$ for each participating qubit in one of the logical operators to be glued, where a is the number of X type generators of which that physical qubit is in the support. One can see this using Definition 6.4, as on the middle code $(W_\bullet, W^\bullet)$ we have

$$P_Z = (\partial_2^{W_\bullet})^\intercal = \begin{pmatrix} \mathrm{id}_{\mathbb{F}_2^m} & \mathrm{id}_{\mathbb{F}_2^m} & (\partial_1^{V_\bullet})^\intercal \end{pmatrix}$$

We then measure d rounds of all stabilisers. All of the qubits in the domain of the last block of P_Z above are those which were initialised to $|+\rangle$. The only other qubits which contribute to the new Z stabiliser generators are those on either side of the sandwiched code, i.e. those along the $\overline{Z}$ logical operators of qubits i and j. Each of the physical qubits

in the support of these logical operators is measured exactly once by the new Z stabiliser generators, and they are measured in pairs, one from each side; therefore performing these measurements and recording the total product is equivalent to measuring $\overline{Z} \otimes \overline{Z}$. We will now check this, and verify that it maintains error-correction.

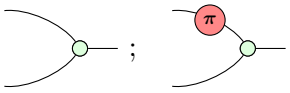
Let the outcome of a new Z-type measurement be $c_\lambda \in \{1, -1\}$, and the overall outcome $c_L = \prod_{\lambda \leq m} c_\lambda$. Whenever $c_\lambda = -1$ we apply the gauge fixing operator $X_\lambda = \bigotimes_{(i \in v \mid v_i = 1)} X_i$ for the specified $v \in C^0$ (or one could choose a gauge fixing operator using D^0 instead). We let $X_{c_L} = \prod_{(\lambda \mid c_\lambda = -1)} X_\lambda$. On participating physical qubits, the merge is then

$$X_{c_L} \prod_\lambda \frac{I + c_\lambda Z}{2} = \prod_\lambda \frac{I + Z}{2} X_{c_L}$$

where we abuse notation somewhat to let I and Z here refer to tensor products thereof. As each X_λ belongs to the same equivalence class of logical $\overline{X}$ operators in $H_1(C_\bullet)$, if $c_L = 1$ then X_{c_L} acts as identity on the logical space; if $c_L = -1$ then X_{c_L} acts as $\overline{X}$ on logical qubit i in the code before merging. One can then see that these two branches are precisely the branches of the logical $\overline{Z} \otimes \overline{Z}$ measurement. As the measurements were performed using d rounds of stabilisers, and the gauge fixing operators each have support on at least d qubits, the overall procedure is error-protected with code distance d.

We also check that the procedure is insensitive to errors in the initialisation of fresh qubits. If a qubit is initialised instead to $|-\rangle$, or equivalently suffers a Z error, then the new Z stabiliser measurements are insensitive to this change, and it will just show up at the X measurements on either side of the fresh qubit. If it suffers some other error, say sending it to $|1\rangle$, then each new stabiliser measurement with that qubit in its support may have its result flipped. By construction of $V_\bullet$, each fresh qubit is in the support of an even number of new Z stabiliser measurements, and so initialising the fresh qubits incorrectly will not change c_L. □

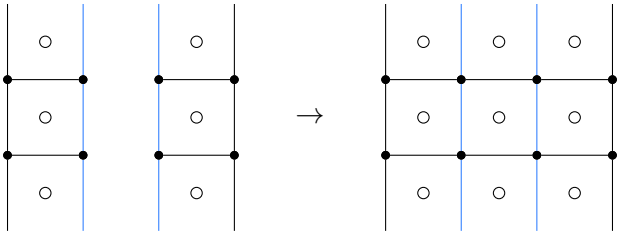
As ZX diagrams, the branches are:



on logical qubits i and j, and all other logical qubits in the code are acted on as identity. We can freely choose which logical qubit may have the red π spider, as it will differ only up to a red π – i.e. a logical $\overline{X}$ – on the output logical qubit. In practice, depending on the code there will typically be cheaper ways of fixing the gauges than using an $\overline{X}$ logical operator for each -1 outcome, as there could be an $\overline{X}$ logical operator which has support on multiple of the qubits belonging to new stabilisers. Moreover, one can just update the Pauli frame rather than apply any actual $\overline{X}$ logical operators. The ability to do so is necessary, however, so that the -1 outcome is well-defined.

The protocol obviates the problem of performing the code map on *physical* qubits deterministically, as the only non-isometric transformations we perform are measurements of stabiliser generators. However, the code map on *logical* qubits is still not isometric, hence we have a logical measurement.

For the prototypical example of lattice surgery we then have:

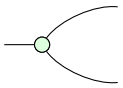


We also look at a less obvious example, that of error-corrected surgery of the Shor code, in Appendix I.

By dualising appropriately one can perform an $\overline{X}$-merge by sandwiching in a similar manner. We can also do the 'inverse' of the merge operation:

Corollary 6.11. *Let $(T_\bullet, T^\bullet)$ be a CSS code formed by sandwiching codes $(C_\bullet, C^\bullet)$ and $(D_\bullet, D^\bullet)$ together along a $\overline{Z}$ operator. Then there is an error-corrected procedure to implement a code map on logical qubits $\mathcal{E}_{\overline{X}}$ from $(T_\bullet, T^\bullet)$ to $((C \oplus D)_\bullet, (C \oplus D)^\bullet)$.*

Proof. As the initial code is already a sandwiched code we can just take the opposite of sandwiching. We delete the qubits belonging to the intermediate code $(W_\bullet, W^\bullet)$ but not $(C_\bullet, C^\bullet)$ or $(D_\bullet, D^\bullet)$ by measuring them out in the X-basis. The code map $\mathcal{E}_{\overline{X}}$ on participating logical qubits is



by following precisely the same logic as for traditional lattice surgery [26]. □

Again, by dualising appropriately we get the last split operation.

Given a procedure for making $\overline{Z} \otimes \overline{Z}$ and $\overline{X} \otimes \overline{X}$ logical measurements and the isometries from splits, one can easily construct a logical CNOT between suitable CSS codes following, say, [6] and observing that the same ZX diagrammatic arguments apply. Augmented with some Clifford single-qubit gates and non-stabiliser logical states one can then perform universal computation. As opposed to some other methods of performing entangling gates with CSS codes, e.g. transversal 2-qubit gates, the schemes above require only the m qubits from the respective $\overline{Z}$ or $\overline{X}$ operators to participate, and we expect $m \ll n$ for practical codes. Unlike that of [12], our method does not require a large ancillary hypergraph product code, which can have significantly worse encoding rate and code distance scaling than the LDPC codes holding data. Our method does not require the code to be 'self-ZX-dual' in the sense of [9], and unlike [27] our method does not require the code to be defined on any kind of manifold, and is purely algebraic in description; moreover, these works study single qubit gates and measurements rather than entangling gates.

7 Conclusions and further work

We believe our constructions are flexible and conceptually quite simple. The immediate next step is to benchmark our CSS code surgery against other methods of performing entangling logical gates and characterise which CSS codes admit separated logical operators such that the merges have distance bounded below.

The pushouts we gave along logical operators are the most obvious cases. By taking pushouts of more interesting spans other maps on logical data can be obtained, although

by Proposition 4.11 and Corollary 4.16 all code maps as we defined them are limited and do not allow for universal quantum computation; we also do not know whether other pushouts would allow the maps on logical data to be performed fault-tolerantly.

In this paper we assumed that the two codes being 'glued' are different codes, but the same principles apply if we have only one code we would like to perform internal surgery on. In this case, the correct universal construction to use should be a coequaliser. There may be other uses of colimits in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$. The methods of constructing families of quantum LDPC codes in [40, 41, 4] use a balanced product, which is also a coequaliser, and the instances used there could be generalised. The quotient is with respect to actions of a finite group G or ring R. This could be changed to some other differential graded algebra, for example the algebras studied in quantum Riemannian geometry [3]. Therein, for a first-order differential calculus $A_\bullet$ one has, say, a group algebra $\mathbb{F}_2G$ as the A_0 component, but then the A_1 component is an $\mathbb{F}_2G$-bimodule. The initial codes $C_\bullet$ and $D_\bullet$ are then right- and left-modules over $A_\bullet$, which in the language of noncommutative geometry are sections of a vector bundle. One must ensure that the actions of $A_\bullet$ on the initial codes are basis-preserving in the sense of Definition 3.4 for the resultant balanced product chain complexes to have a reasonable interpretation as quantum codes. This is always true for free modules, but can also hold for non-free modules. A balanced product code of this form cannot have asymptotically better $[\![n, k, d]\!]$ parameters than those of [40], up to constant factors, as their construction already saturates the relevant bounds. However, it would be interesting to see if one can obtain better parameters for concrete instances. Calculating the homologies of such codes is likely to require tools such as spectral sequences. This area is largely unexplored, as basis-dependent notions like code-distance are not typically of interest to mathematical physicists studying noncommutative geometry, and the process of selecting an instance of a code which has a consistent definition of code distance from an iso-class is categorically 'evil'.

It should be possible to extend the definitions of $\overline{X}$- and $\overline{Z}$-merges straightforwardly to include metachecks [11], say by specifying that the logical operator subcomplex $V_\bullet$ now runs from V_1 to V_{-1}, so it has X-checks and then metachecks on X-checks, but we have not proved how this affects metachecks in the merged code.

There are several ways in which our constructions could be generalised to other codes. The obvious generalisation is to qudit CSS codes. For qudits of prime dimension q, everything should generalise fairly straightforwardly using a different finite field $\mathbb{F}_q$ but in this case the cell complexes will require additional data in the form of an orientation on edges, as is familiar for qudit surface codes. When q is not prime, one formalism for CSS codes with dimension q is chain complexes in $\mathbb{Z}_q$-$\mathtt{FFMod}$, the category of free finite modules over the ring $\mathbb{Z}_q$. As $\mathbb{Z}_q$ is not generally a domain this complicates the homological algebra.

Second, if we wish to upgrade to more general stabiliser codes we can no longer use chain complexes. The differential composition $P_X P_Z^\intercal$ is a special case of the symplectic product $\omega(M, N) = M\omega N^\intercal$ for $\omega = \begin{pmatrix} 0_n & I_n \\ -I_n & 0_n \end{pmatrix}$ [24], but by generalising to such a product we lose the separation of Z and X stabilisers to form a pair of differentials. It is unclear what the appropriate notion of a quotient along an $\overline{X}$ or $\overline{Z}$ operator is for such codes.

For quantum codes which are not stabiliser but are based on cell complexes, such as the Kitaev model [31], there are no stabiliser generators, but the codes are still 'CSS-like', in the sense that vertices correspond to actions of the group algebra $\mathbb{C}G$ and faces actions of the function algebra $\mathbb{C}(G)$, with each measurement outcome corresponding to an irreducible representation of the quantum double $D(G) = \mathbb{C}(G) \bowtie \mathbb{C}G$. More generally we can replace $\mathbb{C}G$ and $\mathbb{C}(G)$ with H and H^* for any semisimple Hopf algebra H [36, 15] while

retaining the relevant features of the model. Just as there are no stabiliser generators, there are no longer Z and X-operators, but there are ribbon operators. As special cases there are ribbon operators which correspond to actions of only $\mathbb{C}G$ or $\mathbb{C}(G)$. The first author recently generalised lattice surgery to Kitaev models [16], albeit with some caveats. In the same way that CSS codes generalise stabiliser codes based on cell complexes, there should be a general class of commuting projector models using the quantum double, which are not necessarily defined on a tessellated manifold. We speculate that the notion of 'gluing' along, say, a $\mathbb{C}G$ operator could work for such commuting projector models. These would give merges and splits which correspond to the multiplications and comultiplications of the relevant Hopf algebras.

8 Acknowledgements

AC thanks Aleks Kissinger for helpful discussions about Lemma 4.4, and both Aleks Kissinger and John van de Wetering for helpful discussions about Proposition 4.11. AC also thanks the Wolfson Harrison UK Research Council Quantum Foundation Scholarship for making this work possible.

We are grateful to Christophe Vuillot for spotting a crucial error in an earlier version of this paper, and for providing us with the illustrative example in Appendix F. We are also grateful to anonymous reviewers, whose thoughtful and rigorous feedback helped improve the paper substantially.

References

[1] F. Arute, K. Arya, R. Babbush et al., Quantum supremacy using a programmable superconducting processor. Nature 574, 505–510 (2019), https://doi.org/10.1038/s41586-019-1666-5

[2] B. Audoux and A. Couvreur, On tensor products of CSS Codes, Ann. Inst. Henri Poincaré Comb. Phys. Interact. 6 (2019), no. 2, pp. 239–287, https://doi.org/10.4171/aihpd/71

[3] E. J. Beggs and S. Majid, Quantum Riemannian Geometry, Springer International Publishing, 1 Feb 2020, https://doi.org/10.1007/978-3-030-30294-8

[4] N. P. Breuckmann and J. N. Eberhardt, Balanced Product Quantum Codes, IEEE Transactions on Information Theory 2021, https://doi.org/10.1109/TIT.2021.3097347

[5] N. P. Breuckmann and J. N. Eberhardt, Quantum Low-Density Parity-Check Codes, PRX Quantum 2 (4), 040101, 2021, https://doi.org/10.1103/PRXQuantum.2.040101

[6] N. de Beaudrap and D. Horsman, The ZX calculus is a language for surface code lattice surgery, Quantum 4, 218 (2020), https://doi.org/10.22331/q-2020-01-09-218

[7] H. Bombin and M. A. Martin-Delgado, Homological error correction: Classical and quantum codes, Journal of Mathematical Physics, vol. 48, no. 5, p. 052105 (2007), https://doi.org/10.1063/1.2731356

[8] S. Bravyi, B. M. Terhal and B. Leemhuis, Majorana fermion codes, New Journal of Physics, vol. 12, no. 8, p. 083039 (2010), https://doi.org/10.1088/1367-2630/12/8/083039

[9] N. P. Breuckmann and S. Burton, Fold-Transversal Clifford Gates for Quantum Codes, arXiv:2202.06647 [quant-ph], https://doi.org/10.48550/arXiv.2202.06647

[10] N. P. Breuckmann, C. Vuillot, E. Campbell, A. Krishna and B. M. Terhal, Hyperbolic and Semi-Hyperbolic Surface Codes for Quantum Storage, Quantum Science and Technology, Volume 2, Number 3, 2017, https://doi.org/10.1088/2058-9565/aa7d3b

[11] E. T. Campbell, A theory of single-shot error correction for adversarial noise, Quantum Science and Technology 4, 025006 (2019), https://doi.org/10.1088/2058-9565/aafc8f

[12] L. Z. Cohen, I. H. Kim, S. D. Bartlett and B. J. Brown, Low-overhead fault-tolerant quantum computing using long-range connectivity, Sci. Adv. 8, eabn1717 (2022), https://doi.org/10.1126/sciadv.abn1717

[13] C. Ryan-Anderson, N. C. Brown, M. S. Allman et al., Implementing Fault-tolerant Entangling Gates on the Five-qubit Code and the Color Code, arXiv:2208.01863 [quant-ph], https://doi.org/10.48550/arXiv.2208.01863

[14] A. Cowtan, Qudit lattice surgery, In Proceedings QPL 2022, arXiv:2204.13228 [quant-ph], https://doi.org/10.48550/arXiv.2204.13228

[15] A. Cowtan and S. Majid, Quantum double aspects of surface code models, J. Math. Phys. 63 042202 (2022), https://doi.org/10.1063/5.0063768

[16] A. Cowtan and S. Majid, Algebraic aspects of boundaries in the Kitaev quantum double model, J. Math. Phys. 64, 102203 (2023), https://doi.org/10.1063/5.0127285

[17] N. Delfosse, Decoding color codes by projection onto surface codes, Phys. Rev. A 89, 012317 (2014), https://doi.org/10.1103/PhysRevA.89.012317

[18] E. Dennis, A. Kitaev, A. Landahl and J. Preskill, Topological quantum memory, J. Math. Phys. 43, 4452-4505 (2002), https://doi.org/10.1063/1.1499754

[19] G. Duclos-Cianci and D. Poulin, A renormalization group decoding algorithm for topological quantum codes, Information Theory Workshop (ITW), 2010 IEEE, pp.1-5, Aug. 30 2010-Sept. 3 2010, https://doi.org/10.1109/CIG.2010.5592866

[20] D. S. Farley, Finiteness and CAT(0) properties of diagram groups, Topology, Vol. 42, Issue 5 (2003) pp. 1065-1082, https://doi.org/10.1016/S0040-9383(02)00029-0

[21] A. G. Fowler, M. Mariantoni, J. M. Martinis and A. N. Cleland, Surface codes: Towards practical large-scale quantum computation, Phys. Rev. A 86 (2012), https://doi.org/10.1103/PhysRevA.86.032324

[22] M. H. Freedman and D. A. Meyer, Projective plane and planar quantum codes, Foundations of Computational Mathematics 1, 325 (2001), https://doi.org/10.1007/s102080010013

[23] D. Gottesman, Stabilizer Codes and Quantum Error Correction, https://doi.org/10.48550/arXiv.quant-ph/9705052

[24] J. Haah, Algebraic Methods for Quantum Codes on Lattices, Revista Colombiana de Matemáticas, 50(2), 299-349 (2016), https://doi.org/10.15446/recolma.v50n2.62214

[25] O. Higgott, M. Wilson, J. Hefford, J. Dborin, F. Hanif, S. Burton and D. E. Browne, Optimal local unitary encoding circuits for the surface code, Quantum 5, 517 (2021), https://doi.org/10.22331/q-2021-08-05-517

[26] D. Horsman, A. G. Fowler, S. Devitt and R. Van Meter, Surface code quantum computing by lattice surgery, New J. Phys. 14 (2012) 123011, https://doi.org/10.1088/1367-2630/14/12/123011

[27] S. Huang, T. Jochym-O'Connor, T. J. Yoder, Homomorphic Logical Measurements, PRX Quantum 4, 030301 (2023), https://doi.org/10.1103/PRXQuantum.4.030301

[28] G. Hahn and G. Sabidussi, Graph symmetry: algebraic methods and applications, NATO Advanced Science Institutes Series, vol. 497, Springer, p. 116 (1997) ISBN 978-0-7923-4668-5, https://doi.org/10.1007/978-94-015-8937-6

[29] A. Kissinger, Phase-free ZX diagrams are CSS codes (...or how to graphically grok the surface code), In Proceedings QPL 2022, arXiv:2204.14038 [quant-ph], https://doi.org/10.48550/arXiv.2204.14038

[30] A. Kissinger, A. Meijer-van de Griend, CNOT circuit extraction for topologically-constrained quantum memories, Quantum Information and Computation, 20, 7& 8, (2020), https://doi.org/10.26421/QIC20.7-8

[31] A. Kitaev, Fault-tolerant quantum computation by anyons, Ann. Phys. 303 (2003) 3–20, https://doi.org/10.1016/S0003-4916

[32] Al. Krishna and David Poulin, Fault-tolerant gates on hypergraph product codes, Phys. Rev. X 11, 011023 (2021), https://doi.org/10.1103/PhysRevX.11.011023

[33] T. Leinster, Basic Category Theory, Cambridge Studies in Advanced Mathematics, Vol. 143, Cambridge University Press, 2014, https://doi.org/10.1017/CBO9781107360068

[34] A. J. Landahl and C. Ryan-Anderson, Quantum computing by color-code lattice surgery, arXiv:1407.5103 [quant-ph], https://doi.org/10.48550/arXiv.1407.5103

[35] Math stackexchange, https://math.stackexchange.com/questions/1046209/pullbacks-and-pushouts-in-the-category-of-graphs, accessed 25/10/22

[36] C. Meusburger, Kitaev lattice models as a Hopf algebra gauge theory, Commun. Math. Phys. 353 (2017) 413–468, https://doi.org/10.1007/s00220-017-2860-7

[37] K. P. Michnicki, 3D Topological Quantum Memory with a Power-Law Energy Barrier, Phys. Rev. Lett. 113, 130501, https://doi.org/10.1103/PhysRevLett.113.130501

[38] H. P. Nautrup, N. Friis and H. J. Briegel, Fault-tolerant interface between quantum memories and quantum processors, Nat. Commun. 8, 1321 (2017), https://doi.org/10.1038/s41467-017-01418-2

[39] J. Old, M. Rispler and M. Müller, Lift-Connected Surface Codes, arXiv:2401.02911 [quant-ph], https://doi.org/10.48550/arXiv.2401.02911

[40] P. Panteleev and G. Kalachev, Asymptotically Good Quantum and Locally Testable Classical LDPC Codes, STOC 2022: Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing, https://doi.org/10.1145/3519935.3520017

[41] P. Panteleev and G. Kalachev, Quantum LDPC Codes With Almost Linear Minimum Distance, in IEEE Transactions on Information Theory, vol. 68, no. 1, pp. 213-229, Jan. 2022, https://doi.org/10.1109/TIT.2021.3119384

[42] A. O. Quintavalle, P. Webster and M. Vasmer, Partitioning qubits in hypergraph product codes to implement logical gates, Quantum 7, 1153 (2023), https://doi.org/10.22331/q-2023-10-24-1153

[43] J. van de Wetering, ZX-calculus for the working quantum computer scientist, arXiv:2012.13966 [quant-ph], https://doi.org/10.48550/arXiv.2012.13966

[44] C. Vuillot, L. Lao, B. Criger, C. G. Almudéver, K. Bertels and B. M. Terhal, Code deformation and lattice surgery are gauge fixing, New J. Phys. 21 033028 (2019), https://doi.org/10.1088/1367-2630/ab0199

[45] C. A. Weibel, An Introduction to Homological Algebra (Cambridge Studies in Advanced Mathematics), Cambridge University Press (1994), https://doi.org/10.1017/CBO9781139644136

[46] Chuan-Kun Wu and Ed Dawson, Existence of generalized inverse of linear transformations over finite fields, Finite Fields and Their Applications 4 (1998) 307–315, https://doi.org/10.1006/ffta.1998.0215

A Limits and colimits in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$

Lemma A.1. $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$ *has all kernels and cokernels.*

Proof. Recall that $\mathtt{Mat}_{\mathbb{F}_2}$ has all kernels and cokernels, i.e. subspaces and quotient spaces. Then given a chain map $f : C_\bullet \to D_\bullet$ we define $\ker(f)$ with

$$\begin{array}{ccccccc}
\cdots & \cdots\!\!\rightarrow & K_{n+1} & \xrightarrow{\partial_n^{K\bullet}} & K_n & \cdots\!\!\rightarrow & \cdots \\
 & & \downarrow{\scriptstyle \ker(f_{n+1})} & & \downarrow{\scriptstyle \ker(f_n)} & & \\
\cdots & \longrightarrow & C_{n+1} & \xrightarrow{\partial_n^{C\bullet}} & C_n & \longrightarrow & \cdots \\
 & & \downarrow{\scriptstyle f_{n+1}} & & \downarrow{\scriptstyle f_n} & & \\
\cdots & \longrightarrow & D_{n+1} & \xrightarrow{\partial_n^{D\bullet}} & D_n & \longrightarrow & \cdots
\end{array}$$

where $\partial_n^{K\bullet}$ always exists and is uniquely defined, because

$$f_n \circ \partial_n^{C\bullet} \circ \ker(f_{n+1}) = \partial_n^{D\bullet} \circ f_{n+1} \circ \ker(f_{n+1}) = 0$$

and so by the universal property of $\ker(f_n)$ there is a unique matrix $\partial_n^{K\bullet} : K_{n+1} \to K_n$. These satisfy $\partial_n^{K\bullet} \circ \partial_{n+1}^{K\bullet} = 0$ as

$$\ker(f_n) \circ \partial_n^{K\bullet} \circ \partial_{n+1}^{K\bullet} = \partial_n^{C\bullet} \circ \partial_{n+1}^{C\bullet} \circ \ker(f_{n+2}) = 0$$

and then kernels are monic. $K_n = \{v \in C_n \mid f_n(v) = 0\}$ by the definition of kernels in $\mathtt{Mat}_{\mathbb{F}_2}$. Given the correct choice of basis, $\partial_n^{K\bullet}$ is thus just $\partial_n^{C\bullet} \circ \ker(f_{n+1})$ as a matrix but without the all-zero rows which map into C_n/K_n.

That $\ker(f)$ is a genuine kernel in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$ is straightforward to check but we do not give further details.

The reversed argument applies for cokernels, giving quotient complexes $D_\bullet/\mathrm{im}(f)$ with components $D_n/\mathrm{im}(f_n)$ etc. □

Remark A.2. *As* $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$ *is additive, equalisers and coequalisers can be seen as special cases of kernels and cokernels by defining* $\mathrm{eq}(f,g) = \ker(f-g)$ *and* $\mathrm{coeq}(f,g) = \mathrm{coker}(f-g)$*, for* $f, g : C_\bullet \to D_\bullet$*. For the chain complex part* $E_\bullet$ *of an equaliser we have components* $E_n = \{c \mid f(c) = g(c)\} \subseteq C_n$*. For the chain complex part* $F_\bullet$ *of a coequaliser, we have components* $F_n = D_n/f(c) \sim g(c)$*, for* $c \in C_n$*.*

We now sketch a proof of Lemma 5.5.

Proof. Recall that an Abelian category is an additive category such that:

1. Every morphism has a kernel and cokernel.
2. Every monomorphism is the kernel of its cokernel.
3. Every epimorphism is the cokernel of its kernel.

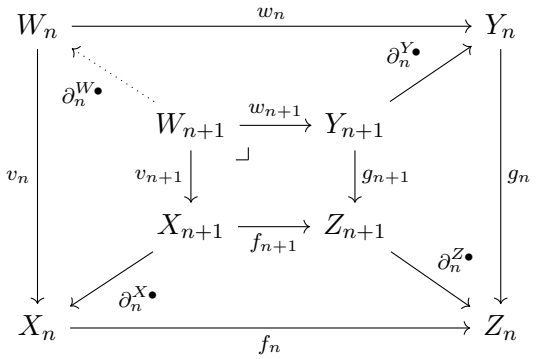
The first is just Lemma A.1, and the other two follow using the fact that they hold degree-wise in $\mathtt{Mat}_{\mathbb{F}_2}$. □

We will now spell out pullbacks. While they can be defined using equalisers and products we construct them explicitly, as it is easy to do so.

Definition A.3. *The pullback of chain maps $f : X_\bullet \to Z_\bullet$ and $g : Y_\bullet \to Z_\bullet$ gives the chain complex $W_\bullet$, where each component is the pullback W_n of f_n and g_n. The differentials $\partial_n^{W_\bullet}$ are given by the unique map from each component's pullback. Specifically, if we have the pullback*

$$\begin{array}{ccc} W_\bullet & \xrightarrow{w} & Y_\bullet \\ {\scriptstyle v}\downarrow & \lrcorner & \downarrow{\scriptstyle g} \\ X_\bullet & \xrightarrow[f]{} & Z_\bullet \end{array}$$

then for degrees $n, n+1$ we have



where

$$W_n = \{(x, y) \mid f_n(x) = g_n(y)\} \subseteq X_n \oplus Y_n; \quad v_n(x, y) = x \in X_n; \quad w_n(x, y) = y \in Y_n.$$

As $f_n \circ \partial_n^{X_\bullet} \circ v_{n+1} = g_n \circ \partial_n^{Y_\bullet} \circ w_{n+1}$ and the outer square is a pullback, there is a unique matrix $\partial_n^{W_\bullet}$. One can check by diagram chasing that the differentials $\partial_n^{W_\bullet} \circ \partial_{n+1}^{W_\bullet} = 0$, and then that this is indeed a pullback in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$.

B Tensor structure of $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$

Definition B.1. *[45, Sec. 2.7] Let $C_\bullet, D_\bullet$ be chain complexes in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$. Define $(C \otimes D)_\bullet$ with components*

$$(C \otimes D)_n = \bigoplus_{i+j=n} C_i \otimes D_j$$

where the latter tensor product is the normal tensor product in $\mathtt{Mat}_{\mathbb{F}_2}$. Differentials between components are given by

$$\partial_n^{(C\otimes D)_\bullet} =_{__\, i+j=n} \left(\mathrm{id}_{C_i} \otimes \partial_j^{D_\bullet} | \partial_i^{C_\bullet} \otimes \mathrm{id}_{D_j}\right)$$

where the horizontal line __ indicates that all these matrices are stacked vertically, which we illustrate in an example below. One can check that $\partial_n^{(C\otimes D)_\bullet} \circ \partial_{n+1}^{(C\otimes D)_\bullet} = 0 \pmod 2$*, as desired.*

Also define the object $\mathbf{1}_\bullet \in \mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$ *as*

$$\mathbf{1}_\bullet = \cdots \longrightarrow 0 \longrightarrow \mathbf{1}_0 \longrightarrow 0 \longrightarrow \cdots$$

where $\mathbf{1}_0 = \mathbb{F}_2$*, and all other* $\mathbf{1}_i$ *are* 0*.*

One can check that $(C \otimes D)_\bullet$ is a $\mathbb{F}_2$-linear monoidal product $\otimes$ in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$, which follows from associativity and distributivity of $\oplus$ and $\otimes$ in $\mathtt{Mat}_{\mathbb{F}_2}$. For the unit, observe that

$$(C \otimes \mathbf{1})_n = C_n \otimes 1 = C_n; \quad \partial_n^{(C\otimes \mathbf{1})_\bullet} = \left(\mathrm{id}_{C_n} \otimes \partial_0^{\mathbf{1}_\bullet} | \partial_n^{C_\bullet} \otimes \mathrm{id}_{\mathbf{1}_0}\right) = \partial_n^{C_\bullet}.$$

Example B.2. *Consider two chain complexes of length 1:*

$$C_\bullet = \cdots \longrightarrow 0 \longrightarrow C_1 \xrightarrow{\partial_1^{C_\bullet}} C_0 \longrightarrow 0 \longrightarrow \cdots$$

$$D_\bullet = \cdots \longrightarrow 0 \longrightarrow D_1 \xrightarrow{\partial_1^{D_\bullet}} D_0 \longrightarrow 0 \longrightarrow \cdots$$

In this case we have

$$(C \otimes D)_0 = C_0 \otimes D_0; \quad (C \otimes D)_1 = (C_1 \otimes D_0) \oplus (C_0 \otimes D_1); \quad (C \otimes D)_2 = C_1 \otimes D_1$$

for nonzero components, and

$$\partial_1^{(C\otimes D)_\bullet} = (\mathrm{id}_{C_0} \otimes \partial_1^{D_\bullet} | \partial_1^{C_\bullet} \otimes \mathrm{id}_{D_0}); \quad \partial_2^{(C\otimes D)_\bullet} = \left(\frac{\partial_1^{C_\bullet} \otimes \mathrm{id}_{D_1}}{\mathrm{id}_{C_1} \otimes \partial_1^{D_\bullet}}\right)$$

for nonzero differentials. Then

$$\partial_1^{(C\otimes D)_\bullet} \circ \partial_2^{(C\otimes D)_\bullet} = \partial_1^{C_\bullet} \otimes \partial_1^{D_\bullet} + \partial_1^{C_\bullet} \otimes \partial_1^{D_\bullet} = 0 \pmod 2$$

as the matrix partitions factor upon multiplication.

This example illustrates an interesting property of $\otimes$ in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$: both $C_\bullet, D_\bullet$ have only one nonzero differential, but $(C \otimes D)_\bullet$ has two. It is easy to see that given two complexes of lengths s, t the tensor product will have length $s + t$.

Lemma B.3. *[45]*

$$H_n((C \otimes D)_\bullet) \cong \bigoplus_{i+j=n} H_i(C_\bullet) \otimes H_j(D_\bullet)$$

That is, the homology subspaces factor through the tensor product conveniently. This is also called the Künneth formula. The manner in which the homology factors through does not make $H_n(-)$ a monoidal functor with respect to the tensor product.

The tensor product is used to build codes from other CSS codes [2].

C Graphs and cell complexes

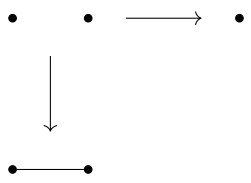
In this appendix we give some categorical background on abstract cell complexes. This is not necessary to define CSS code surgery, but codes obtained from cell complexes are an important motivating example, as they include surface codes, toric codes [31], hyperbolic codes [10] and the expander lifted product codes from [40]. In general, if a CSS code comes from tessellating a manifold, it is likely to use cell complexes. Cell complexes are important in the study of topological spaces, and many of the constructions of CSS codes, such as balanced/lifted products, can also be phrased in the language of topology, but we stick to cell complexes for brevity. As a warm-up, we describe certain categories of graphs, and then move on to a specific kind of cell complex.

Let Γ be a finite simple undirected graph. Recall that as a simple graph, Γ has at most one edge between any two vertices and no self-loops on vertices. Γ can be defined as a pair of sets, $V(\Gamma)$ and $E(\Gamma)$, with $E(\Gamma) \subseteq 2^{V(\Gamma)}$, the powerset of vertices, where each $e \in E(\Gamma)$ has 2 elements i.e. it can be expressed as $e = \{v_1, v_2\}$. An example of a graph is $\mathcal{C}_n$, the cycle graph with n vertices and edges. We will also use $\mathcal{P}_n$, the path graph with n edges and $n+1$ vertices.

Definition C.1. *Let* `Grph` *be the category of finite simple undirected graphs. A morphism* $\Gamma \to \Delta$ *in* `Grph` *is a function* $f : V(\Gamma) \to V(\Delta)$ *such that* $\{v_1, v_2\} \in E(\Gamma) \implies \{f(v_1), f(v_2)\} \in E(\Delta)$*, i.e. the function respects the incidence of edges.*

`Grph` has several different products and other categorical features. We are particularly interested in colimits. `Grph` has a coproduct $\Gamma + \Delta$ being the disjoint union, with $V(\Gamma + \Delta) = V(\Gamma) \sqcup V(\Delta)$ and $E(\Gamma + \Delta) = E(\Gamma) \sqcup E(\Delta)$. It also has an initial object I given by the empty graph. However, `Grph` is not cocomplete, as it does not have all pushouts.

Example C.2. *As a counterexample [35], given the diagram*



no cocone exists, as the graphs are not allowed self-loops. Therefore, no pushout exists.

One can easily see that there are diagrams for which pushouts do exist, though.

More than just graphs, we would like to allow for *open* graphs, i.e. graphs which may have edges which connect to only one vertex, but are not self-loops. For example,

We call $\mathcal{G}_3$ the 3rd *open path graph*, where the nth open path graph $\mathcal{G}_n$ has n edges in a line with $n-1$ vertices between them. We now give a particular formalisation of open graphs.

Definition C.3. *Let* Γ *be a finite simple undirected graph with two disjoint vertex sets* $V(\Gamma)$ *and* $B(\Gamma)$*, where* $E(\Gamma) \subseteq 2^{V(\Gamma) \cup B(\Gamma)}$*. We then say that* Γ *is an open graph. We call* $V(\Gamma)$ *the internal vertices and* $B(\Gamma)$ *the boundary vertices.*

So in the picture of $\mathcal{G}_3$ above there are vertices at either end of the open wires, but they are considered 'invisible', i.e. they belong to $B(\Gamma)$.

Definition C.4. *Let* `OGrph` *be the category of open graphs. A morphism* $\Gamma \to \Delta$ *in* `OGrph` *is a function* $f : V(\Gamma) \cup B(\Gamma) \to V(\Delta) \cup B(\Delta)$ *such that* $\{v_1, v_2\} \in E(\Gamma) \implies \{f(v_1), f(v_2)\} \in E(\Delta)$ *and* $f(x) \in V(\Delta) \iff x \in V(\Gamma)$.

This restriction disallows internal vertices from being 'created' or 'deleted' by a graph morphism by converting them to boundary vertices. `OGrph` has very similar properties to `Grph`. Its initial object is the empty open graph. `OGrph` has a coproduct, where $V(\Gamma + \Delta) = V(\Gamma) \sqcup V(\Delta)$ and $B(\Gamma + \Delta) = B(\Gamma) \sqcup B(\Delta)$. Like `Grph`, `OGrph` is not cocomplete, as Example C.2 also works in the setting of open graphs. It is obvious that `Grph` is a subcategory of `OGrph`.

We now move on to cell complexes, in particular abstract cubical complexes. These are abstract cell complexes which are 'square', unlike their 'triangular' relatives simplicial complexes.

Definition C.5. *The abstract* d*-cube is the set* $\{0,1\}^d$*, with the* 0*-cube* $\{0,1\}^0 := \{0\}$*. A face of the abstract* d*-cube is a product* $A_1 \times \cdots \times A_d$*, where each* A_i *is a nonempty subset of* $\{0,1\}$.

Definition C.6. *[20] Let* S *be a finite set and let* Ω *be a collection of nonempty subsets of* S *such that:*

- Ω *covers* S.
- *For* $X, Y \in \Omega$, $X \cap Y \in \Omega$ *or* $X \cap Y = \emptyset$.
- *For each* $X \in \Omega$*, there is a bijection from* X *to the abstract* d*-cube for some choice of* d*, such that any* $Y \subset X$ *is in* Ω *iff it is mapped to a face of the* d*-cube.*

Then Ω *is an abstract cubical complex.*

Abstract cubical complexes are combinatorial versions of cubical complexes, meaning they are stripped of their associated geometry. The elements in Ω are still called *faces*. We can consider Ω to be a graded poset, with subset inclusion as the partial order, and the grading $\dim(X) = \log_2 |X|$. We also call this grading the dimension d of X, and we call X a d-face. The set of d-faces in Ω is called Ω_d. There is a relation $\Omega_d \to \Omega_{d-1}$ taking a d-face to its $(d-1)$-face subsets.

We call the vertex set $V(\Omega) = S = \Omega_0$, and also define the dimension of a cubical complex

$$\dim(\Omega) = \max_{X \in \Omega} \dim(X)$$

The d-skeleton of Ω is the maximal subcomplex $\Upsilon \subseteq \Omega$ such that $\dim(\Upsilon) = d$. The 1-skeleton of an abstract cubical complex is a finite simple undirected graph. The 2-skeleton of an abstract cubical complex is 'like' a square lattice, in that it has 2-faces which each have 4 0-faces as subsets and 4 1-faces.

Definition C.7. *Let* `ACC` *be the category of abstract cubical complexes. A morphism* $f : \Omega \to \Upsilon$ *in* `ACC` *is a function* $f : V(\Omega) \to V(\Upsilon)$*, such that* $\{x, \cdots, y\} \in \Omega_d \implies \{f(x), \cdots, f(y)\} \in \Upsilon_d$*, i.e. incidence is preserved at each dimension.*

Similar to `Grph`, `ACC` has coproduct given by $(\Omega + \Upsilon)_i = \Omega_i \sqcup \Upsilon_i$ and an initial object $I = \emptyset$, and does not generally have pushouts, where we can reuse the same counterexample as `Grph`. Another categorical property we highlight here is that `ACC` has a monoidal product called the *box product*.

Definition C.8. *Let $\Upsilon \square \Omega$ be the box product of abstract cubical complexes. Then*

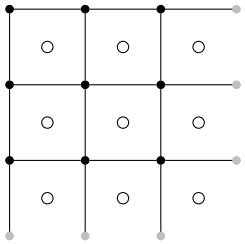
$$(\Upsilon \square \Omega)_n = \sum_{i+j=n} \Upsilon_i \times \Omega_j.$$

We now check that $\Upsilon \square \Omega$ is indeed an abstract cubical complex.

Proof. First, it has a vertex set $V(\Upsilon \square \Omega) = V(\Upsilon) \times V(\Omega)$, and thus trivially covers $\Upsilon_0 \times \Omega_0$. Second, let $X \times Y \in \Upsilon_i \times \Omega_j$ and $T \times U \in \Upsilon_k \times \Omega_l$. This has $(X \times Y) \cap (T \times U) = (X \cap T) \times (Y \cap U)$ which is either in $\Upsilon_m \times \Omega_n$ for some $m \leq i, m \leq k$ and $n \leq j, n \leq l$, and thus $(X \cap T) \times (Y \cap U) \in \Upsilon \square \Omega$, or $(X \cap T) \times (Y \cap U) = \emptyset$. Third, if X and Y each have a bijection to an i-cube and j-cube respectively, then $X \times Y$ has a bijection to an $(i+j)$-cube. Any $W \subset X \times Y$ can be expressed as $T \times U$, for $T \subset X$ and $U \subset Y$. Then W is in $\Omega \square \Upsilon$ iff T is mapped to a face of the i-cube and U to a face of the j-cube, thus W to a face of the $(i+j)$-cube. □

Let us compile this into a more digestible form for the case when Υ and Ω are both graphs. Given vertices (u, u') and (v, v') in $V(\Upsilon \square \Omega)$, the 1-face $\{(u, u'), (v, v')\} \in (\Upsilon \square \Omega)_1$ iff $(u = v \ \& \ (u', v') \in \Omega)$ or $((u, v) \in \Upsilon \ \& \ u' = v')$. Then $(\Upsilon \square \Omega)_2 \cong E(\Upsilon) \times E(\Omega)$. The 1-skeleton of $\Upsilon \square \Omega$ is just the normal box product of graphs [28].

Example C.9. *Let $\mathcal{C}_m$ and $\mathcal{C}_n$ be cycle graphs with m and n vertices respectively, considered as abstract cubical complexes. Then $\mathcal{T} = \mathcal{C}_m \square \mathcal{C}_n$ admits an embedding as a square lattice on the torus, and has $\dim(\mathcal{C}_m \square \mathcal{C}_n) = 2$. Setting $m = n = 3$ we have*



where the grey dots indicate periodic boundary conditions and the white circles specify 2-faces. This example comes up in the form of the toric code in Section 4.

Obviously, `Grph` is a subcategory of `ACC`.

We are also interested in *open* abstract cubical complexes.

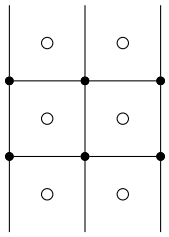
Definition C.10. *Let Υ be an open abstract cubical complex. Υ is an abstract cubical complex where Υ_0 is divided into two disjoint vertex sets $V(\Upsilon)$ and $B(\Upsilon)$.*

The 1-skeleton of an open abstract cubical complex is an open graph.

Definition C.11. *Let `OACC` be the category of open abstract cubical complexes. A morphism $f : \Omega \to \Upsilon$ in `OACC` is a function $f : V(\Omega) \cup B(\Omega) \to V(\Upsilon) \cup B(\Upsilon)$ such that $f(x) \in V(\Upsilon) \iff x \in V(\Omega)$ and $\{x, \cdots, y\} \in \Omega_d \implies \{f(x), \cdots, f(y)\} \in \Upsilon_d$.*

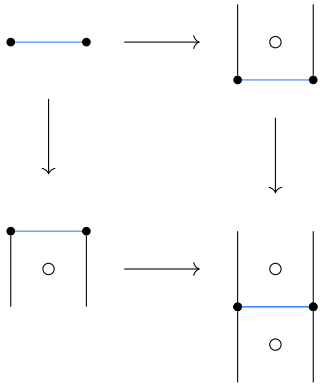
As in our previous examples, `OACC` has the obvious coproduct and initial object, and does not have pushouts in general.

Example C.12. *Let Υ be a 'patch', a square lattice with two rough and two smooth boundaries:*



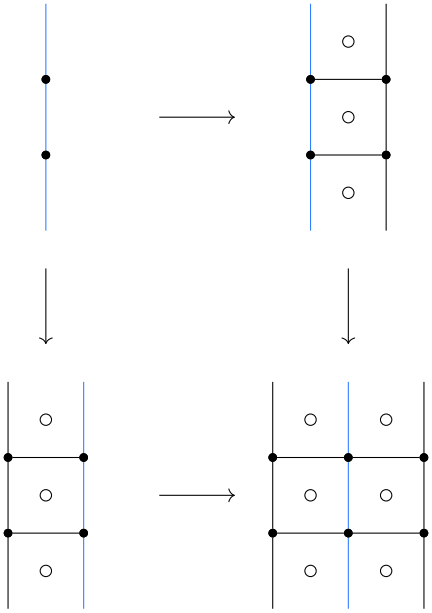
This patch has 6 2-faces, 13 1-faces and 6 0-faces.

Example C.13. *We can perform the pushout of two smaller open abstract cubical complexes to acquire a patch:*



where the apex is $\mathcal{P}_1$, the blue edge indicates where the apex is mapped to, and the bottom right open abstract cubical complex is the object of the pushout.

Example C.14. *Let $\mathcal{G}_3$ be the open path graph, and let Ω be a patch. Then we have a pushout*



This example comes up in the context of lattice surgery on surface codes. Evidently, both `OGrph` and `ACC` are subcategories of `OACC`, and one can define a box product for `OACC` in the same way as we did for `ACC` in Definition C.8.

One can define quantum codes using abstract cell complexes more generally, but abstract cubical complexes are the specific type which we make use of in examples in Section 4 and onwards. We now relate the above cell complexes to chain complexes by way of functors.

Definition C.15. *Given an abstract cubical complex Ω we can define the incidence chain complex $C_\bullet$ in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$, where each nonzero component has a basis $\tilde{C}_n = \Omega_n$, and each nonzero differential $\partial^{C_\bullet}_{n+1}$ takes an $n+1$-face to its n-dimensional subsets. The differential is thus a matrix with a 1 where an n-face is contained within an $(n+1)$-face, and 0 elsewhere. It is an elementary fact that every $(d-2)$-face in a d-face is the intersection of exactly 2 $(d-1)$-faces, thus $\partial^{C_\bullet}_{n-1} \circ \partial^{C_\bullet}_n = 0 \mod 2$. Clearly, the incidence chain complex of a dimension 1 abstract cubical complex is just the incidence matrix of a simple undirected graph.*

We can do essentially the same thing given an open abstract cubical complex Υ. In this case, each nonzero component has a basis $\tilde{C}_n = \{X \in \Omega_n \mid X \not\subseteq B(\Omega)\}$, that is we ignore all faces which are made up only of boundary vertices, and differentials are the same matrices as above, with a 1 where an n-face which is not a subset of $B(\Omega)$ (and therefore would be 'invisible') is contained in an $(n+1)$-face. It is easy to see that we still have $\partial^{C_\bullet}_n \circ \partial^{C_\bullet}_{n+1} = 0 \mod 2$, as making vertices 'invisible' corresponds to deleting rows in $\partial^{C_\bullet}_1$, edges rows in $\partial^{C_\bullet}_2$ etc. The incidence chain complex of a dimension 1 open abstract cubical complex is the incidence matrix of an open graph.

Definition C.16. *Let $C_\bullet$ and $D_\bullet$ be the incidence chain complexes of two abstract cubical complexes Ω and Υ with a morphism $f : \Omega \to \Upsilon$, and set $\tilde{C}_0, \tilde{D}_0$ as $V(\Omega), V(\Upsilon)$ respectively. This induces a chain map $g_\bullet : C_\bullet \to D_\bullet$, with the matrix g_1 given by f, and all matrices on higher components generated inductively. Degrees $i < 1$ are assumed to be zero.*

As a consequence, we can define a functor $\varphi : \mathtt{ACC} \to \mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$, sending each abstract cell complex to its free chain complex as described in Definition C.15. One can check that $\varphi(f) \in \mathrm{Hom}(\varphi(\Omega), \varphi(\Upsilon))$ for any morphism $f : \Omega \to \Upsilon$ between abstract cubical complexes. φ is faithful but not full, as there exist morphisms, such as the zero morphism, which are not in the image of φ.

Definition C.17. *There is also a functor $\vartheta : \mathtt{OACC} \to \mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$. On objects, this again follows Definition C.15. On morphisms this is the same as φ except it must obviously ignore maps between boundary vertices everywhere. Thus ϑ is not faithful.*

Example C.18. *Let Ω and Υ be two abstract cubical complexes. Then $\varphi(\Omega + \Upsilon) = \varphi(\Omega) \oplus \varphi(\Upsilon)$, which is easy to check. Similarly, $\varphi(\emptyset) = \mathbf{0}_\bullet$. The same is true of ϑ, except that $\vartheta(\Xi) = \mathbf{0}_\bullet$ for any Ξ with $V(\Xi) = \emptyset$.*

Lemma C.19. *The functors φ and ϑ are cocontinuous i.e. they preserve colimits.*

Proof. We give a proof sketch here. We know already that φ preserves coproducts so it is sufficient to check that it preserves pushouts. Let

$$\begin{array}{ccc} \Xi & \xrightarrow{g} & \Upsilon \\ {\scriptstyle f}\downarrow & \ulcorner & \downarrow{\scriptstyle l} \\ \Omega & \xrightarrow[k]{} & \chi \end{array}$$

be a pushout in $\mathtt{ACC}$. Then $\chi_0 = \Omega_0 \sqcup \Upsilon_0 / f \sim g$, and we have elements in χ_n of the form $([x], \cdots, [y])$, which can be seen as pushouts at each dimension. Also, $(x, \cdots, y) \in$

$\Omega_n \implies ([x], \cdots, [y]) \in \chi_n$, and the same for Υ_n. Then $\varphi(\tilde{\chi})_0 = \chi_0$. We then have basis elements of the form $([x], \cdots, [y]) \in \varphi(\tilde{\chi})_n$, and differentials have their obvious form. If we take the diagram in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$:

$$\begin{array}{ccc} \varphi(\Xi) & \xrightarrow{\varphi(g)} & \varphi(\Upsilon) \\ {\scriptstyle \varphi(f)}\downarrow & & \\ \varphi(\Omega) & & \end{array}$$

Then we have $Q_\bullet$ as the pushout. Basis elements in Q_n are then also of the form $([x], \cdots, [y])$ for $[x], [y] \in \chi_n$. The differentials also match up correctly, and so $Q_\bullet = \varphi(\chi)$.

The same checks apply if we take $\vartheta : \mathtt{OACC} \to \mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$ instead. Observe that in this case f and g may have images only in $B(\Omega)$ and $B(\Upsilon)$, in which case Ξ must have empty $V(\Xi)$. Then the pushout in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$ will just be a direct sum, i.e. the pushout with $\vartheta(\Xi) = \mathbf{0}_\bullet$ as the apex.

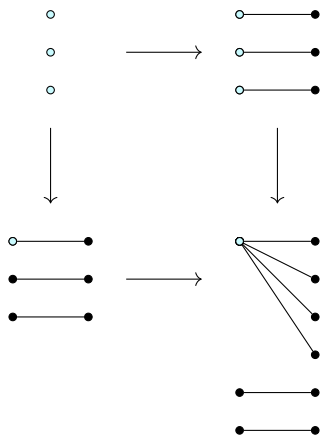
Recall that $\mathtt{ACC}$ and $\mathtt{OACC}$ do not themselves have all pushouts, and therefore all colimits, but φ and ϑ preserve those which they do have. □

Definition C.20. *For any chain complex $C_\bullet$ we have also the pth translation $C[p]_\bullet$, where all indices are shifted down by p, i.e. $C[p]_n = C_{n+p}$ and $\partial_n^{C[p]_\bullet} = \partial_{n+p}^{C_\bullet}$. This extends to an invertible endofunctor $p : \mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2}) \to \mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$ in the obvious way.*

Lemma C.21. *Let Υ and Ω be two open abstract cubical complexes. Recalling the functor $\vartheta : \mathtt{OACC} \to \mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$ from Definition C.16, we have $\vartheta(\Upsilon \square \Omega) = \vartheta(\Upsilon) \otimes \vartheta(\Omega)$, so ϑ is a monoidal functor, but for moral purposes it is.*

D Pushouts and properties of codes

Here we describe a few problems with using general pushouts to construct new quantum codes, even when the spans are basis-preserving. First, in a certain sense the pushout of LDPC codes is not necessarily LDPC. To illustrate this, consider the following pushout of graphs:



where the light dots indicate the graph morphisms. As ϑ is cocontinuous this pushout exists also in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$. There, it represents a merge of two binary classical codes, although we can consider a binary linear code to just be a CSS code without any Z measurements. As a consequence, we have two initial codes with P_X having maximal weights 1 each, and the merged code has maximal weight 4. Evidently, one can scale this with the size of the input graphs: here, the input graphs each have 3 edges, but if there are graphs with m

edges each (and weight 1) and the apex with m vertices (and weight 0) then the pushout graph will have maximal weight $m+1$. As a consequence the family of pushout graphs as m scales is not bounded above by a constant, and so the corresponding family of codes is not LDPC.

Conjecture D.1. *Let*

$$\begin{array}{ccc} A_\bullet & \xhookrightarrow{g_\bullet} & D_\bullet \\ {\scriptstyle f_\bullet}\downarrow & & \\ C_\bullet & & \end{array}$$

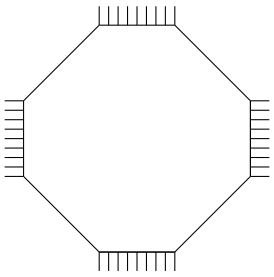
be a basis-preserving monic span in $\mathtt{Ch}(\mathtt{Mat}_{\mathbb{F}_2})$*, and let* $Q_\bullet$ *be the pushout chain complex of this monic span. Further, let the monic span be a representative of a family of monic spans which are parameterised by some* $n \in \mathbb{N}$*, and let* $A_\bullet$*,* $C_\bullet$ *and* $D_\bullet$ *be the* Z*-type complexes of quantum LDPC codes. Then* $(Q_\bullet, Q^\bullet)$ *is also LDPC.*

Formulating this conjecture properly requires specifying what it means for a monic span to be parameterised.

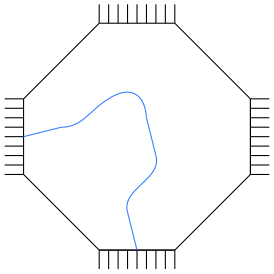
Lastly, taking pushouts evidently preserves neither homologies nor code distances, as easy examples with lattice surgery demonstrate. Moreover, we do not know of a way of giving bounds on these quantities for general pushouts, although again we suspect it should be easier for monic spans.

E Octagonal surface code patch

Consider the following patch of surface code:

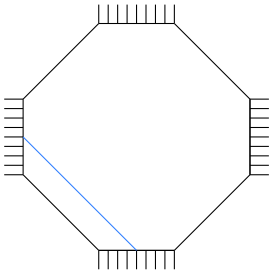
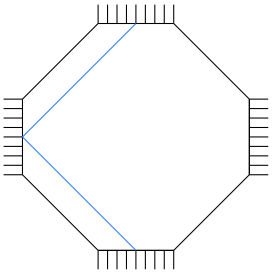
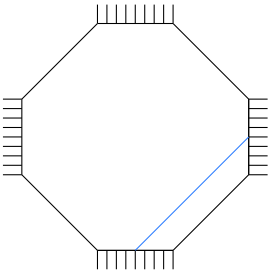


where the bristled edges are rough boundaries, and the diagonal edges are smooth boundaries. We have abstracted away from the actual cell complex as the tessellation is not important. Z-type logical operators take the form of strings extending from one rough boundary to another, e.g.

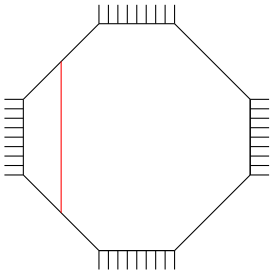
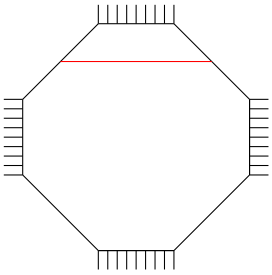
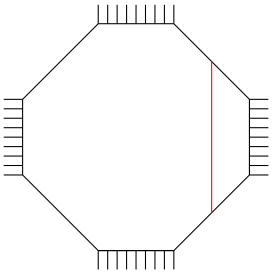


Two strings belong to the same equivalence class iff they are isotopic on the surface, allowing for the endpoints to slide up and down a rough boundary. There are exactly 3 nontrivial such classes out of which all other strings can be composed. As a consequence,

this patch of surface code has logical space V with $\dim V = 2^3 = 8$. [6] We can choose a basis for this logical space, which has logical $\overline{Z}$ operators with representatives:

$\overline{Z}_1 =$  $\overline{Z}_2 =$  $\overline{Z}_3 =$ 

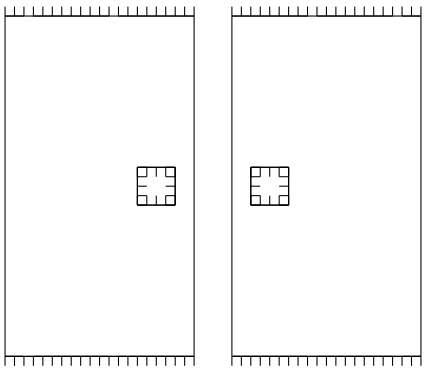
where the middle operator can be smoothly deformed to a vertical line from top to bottom if desired. Recall that on the surface code an $\overline{X}$ operator anticommutes with a $\overline{Z}$ operator iff the strings cross an odd number of times. Thus, given the basis above, the duality pairing of Lemma 4.4 forces a similar basis of $\overline{X}$ operators, with representatives:

$\overline{X}_1 =$  $\overline{X}_2 =$  $\overline{X}_3 =$ 

We see that $\overline{Z}_1$ is contained entirely within $\overline{Z}_2$ on physical qubits. Thus it is possible to construct a $\overline{Z}$ merge which is not separated, in the parlance of Definition 5.12. If we choose a different representative, by deforming $\overline{Z}_2$ to be a vertical line, then we can also perform a separated $\overline{Z}$ merge.

F A $\overline{Z}$-merge map which is not distance preserving

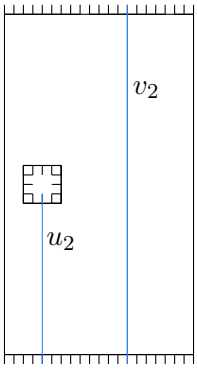
Here we provide an illustrative example to show that it is possible to create $\overline{Z}$ operators in a $\overline{Z}$-merged code which are of lower weight than any logical operator in the initial code. Consider the following surface code patches:



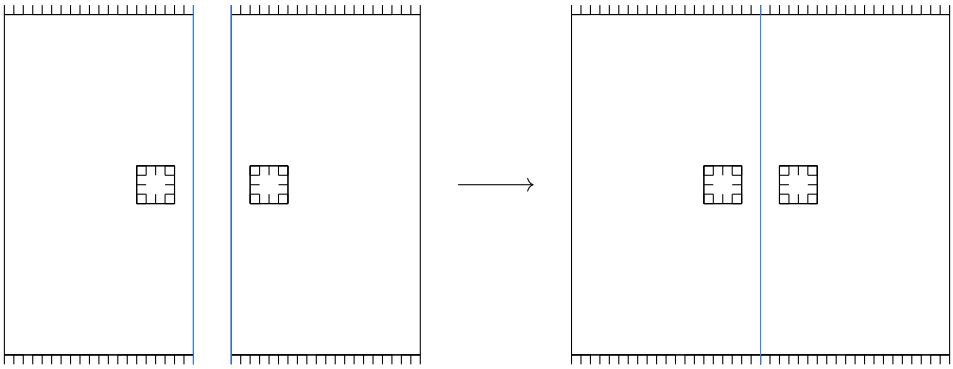
where, as in the previous appendix, bristled edges represent rough boundaries and non-bristled edges represent smooth boundaries. There is a hole in each patch, with bristled

[6] More generally, a patch with $2m$ edges, alternating rough and smooth, has $\dim V = 2^{m-1}$, i.e. the number of edges in a minimal spanning tree on the complete graph with m vertices.

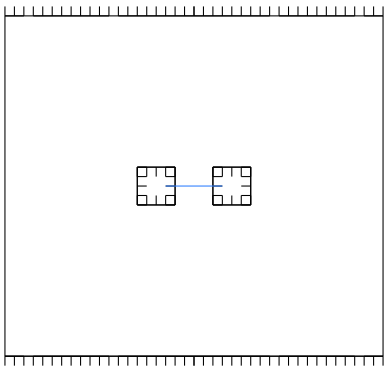
edges around it. As a consequence, each patch has 2 logical qubits. We can assign $\overline{Z}$ logical operators u_2 and v_2, representatives of each equivalence class $[u_2]$ and $[v_2]$ from which all other classes can be composed, like so:



and the same for $[u_1]$, $[v_1]$ on the other patch. We quotient out a $\overline{Z}$ operator in $[v_1]$ and $[v_2]$ going along the right and left boundaries, like so:



leaving a $\overline{Z}$-merged code. This has new $\overline{Z}$ operators, which belong to the equivalence class $[u_1 + u_2]$. These operators are of the form:



to see that these do belong to $[u_1 + u_2]$, label this operator t and see that $t + u_1 + u_2$ is in $[0]$, as it forms a contractible loop. Then $[t] = [-u_1 - u_2] = [u_1 + u_2]$, recalling that we are working over $\mathbb{F}_2$. This new operator t has a weight lower than any of those in the original codes, which one can see from the diagrams.

G A merged code with larger logical space

Take the lift-connected surface (LCS) codes from [39] with $l = 1$, $L = 3$. This is a $[\![15, 3, 3]\!]$ code $C_\bullet$, with the parity-check matrices:

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

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

This code has a separated $\overline{Z}$-logical with support on qubits $(2, 9, 14)$, starting from 0. If we merge two copies of $C_\bullet$ along this logical we obtain a $[\![27, 6, 2]\!]$ code, when naively one would expect a code with 5 logical qubits. The additional logical has appeared because, while quotienting the logicals together in the two codes, we have inadvertently increased the size of $\ker(P_X)$, increasing the size of the logical space.

One explanation for why this can occur is because of the complication of bases for our chain complexes. In Definition 5.9, we set $\tilde{V}_0 = \bigcup_{u \in \mathrm{im}(\partial_1^{V\bullet})} \mathrm{supp}\ u$, so we incorporated all basis elements with support in the image of the logical operator; if we were to instead set $V_0 = \mathrm{im}(\partial_1^{V\bullet})$ we believe that this occurrence would be impossible, as the only quotient would be on precisely those vectors in the image of the logical operator, not those vectors' basis elements. However, we cannot do this in general while keeping all chain maps basis-preserving.

H Separation is gauge-fixability

We will prove in this section that for all CSS codes having separated and gauge-fixable operators are equivalent properties.

Lemma H.1. *A vector $u \in \ker(P_X) \backslash \mathrm{im}(P_Z^\mathsf{T})$ is gauge-fixable iff for every pair (e_i, e_j) of basis vectors in* supp (u) *there is a vector a in $\mathrm{im}(P_X^\mathsf{T})$ such that $u \odot a = e_i + e_j$.*

Proof. If there are two vectors v, w paired with u such that $v \odot u = e_i$ and $w \odot u = e_j$ then $v + w \odot u = e_i + e_j$. As each basis vector in u must be safely correctable, u always has a vector $a = v + w \in \mathrm{im}(P_X^\mathsf{T})$ such that $a \odot u = e_i + e_j$. Going the other way, there must be at least one paired vector v with u such that $u \cdot v = 1$ – we don't assume that $|u \odot v| = 1$, just that the dot product is 1, i.e. they have an odd number of intersecting basis vectors. This can be reduced to an intersection of 1 by applying a vector in $\mathrm{im}(P_X^\mathsf{T})$ corresponding to a pair (e_i, e_j) to each of the pairs of intersecting basis vectors apart from the last one. This single basis vector can then be moved around u by further applications of vectors in $\mathrm{im}(P_X^\mathsf{T})$. □

We find this equivalent definition of gauge-fixing more helpful in practice, as it requires only data about the X stabilisers, rather than paired logical operators, the choice of which depends on the choice of basis of $H_1(C_\bullet)$.

Lemma H.2. *Let $\partial_A : A_1 \to A_0$ be a matrix over $\mathbb{F}_2$. Then for any $v \in \ker(\partial_A)$, either for any pair of basis vectors of A_1 $(e_i, e_j) \in \mathrm{supp}(v)$ there is a vector $b \in \mathrm{im}(\partial_A^\intercal)$ such that $v \odot b = e_i + e_j$ or there is another non-zero vector $u \in \ker(\partial_A)$ such that $\mathrm{supp}(u) \subset \mathrm{supp}(v)$.*

Proof. Define the vector space $S = \{w \odot v : w \in \ker(\partial_A)^\perp\}$. Observe that any vector in S must have even Hamming weight, and that

$$S \cong \{w \odot v : w \in \ker(\partial_A \restriction_{\mathrm{supp}\ v})^\perp\} = \ker(\partial_A \restriction_{\mathrm{supp}\ v})^\perp,$$

as any vectors in $\ker(\partial_A)^\perp$ wholly outside of $\mathrm{supp}(v)$ will not contribute to the Hadamard product, and the isomorphism merely entails chopping off some entries which will always be 0. Now, $\dim(S) \leq |v| - 1$, as $\dim(\ker(\partial_A \restriction_{\mathrm{supp}\ v})) \geq 1$ by definition.

Suppose $\dim(S) = |v| - 1$. Then v has no other vectors in $\ker(\partial_A)$ contained in its support, as then $\dim(\ker(\partial_A \restriction_{\mathrm{supp}\ v})) = 1$, and for any pair of basis vectors of A_1 $(e_i, e_j) \in \mathrm{supp}(v)$ there is a vector $b \in \mathrm{im}(\partial_A^\intercal)$ such that $v \odot b = e_i + e_j$. To see this, view S as the row space of a matrix:

$$\begin{pmatrix} w_1 \odot v \\ w_2 \odot v \\ \vdots \\ w_m \odot v \end{pmatrix}$$

where $m = |v| - 1$, with some chosen basis of S. Then, put the matrix in row echelon form by performing Gaussian elimination:

$$\begin{pmatrix} 1 & * & * & * & \cdots & * \\ 0 & 1 & * & * & \cdots & * \\ & & & \vdots & & \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

where $*$ values are unknown. As $\dim(S) = |v| - 1$, without knowing anything else about the code, there will be exactly 1 row which is indented by 2 from the previous row. But each row must have an even number of 1s in it, including the last row, so the matrix must actually have the row echelon form:

$$\begin{pmatrix} 1 & * & * & \cdots & * & * \\ 0 & 1 & * & \cdots & * & * \\ 0 & 0 & 1 & \cdots & * & * \\ & & \vdots & & & \\ 0 & 0 & 0 & \cdots & 1 & 1 \end{pmatrix}$$

Then, add rows from the bottom to the top as necessary to give

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & * \\ 0 & 1 & 0 & \cdots & 0 & * \\ 0 & 0 & 1 & \cdots & 0 & * \\ & & \vdots & & & \\ 0 & 0 & 0 & \cdots & 1 & 1 \end{pmatrix}$$

i.e. an identity matrix with one column unknown to the right. But once again each row must have an even number of non-zero entries, as $S = \ker(\partial_A \restriction_{\mathrm{supp}\ v})^\perp$, so the column to the right must have all entries equal to 1. Therefore, we have $|v| - 1$ different pairs, and combinations of these suffice to give any pair in $\mathrm{supp}(v)$.

Now, suppose $\dim(S) < |v| - 1$. Then, as $\dim(\ker(\partial_A \restriction_{\mathrm{supp}\ v})^\perp) > 1$, there must be another vector in $\ker(\partial_A)$ contained in $\mathrm{supp}(v)$. □

This means that a vector v in $\ker(\partial_A)$, with no other vectors in $\ker(\partial_A)$ contained in $\mathrm{supp}(v)$, will always have the property that for any pair of basis vectors of A_1 $(e_i, e_j) \in \mathrm{supp}(v)$ there is a vector $b \in \mathrm{im}(\partial_A^\mathsf{T})$ such that $v \odot b = e_i + e_j$.

This lemma implies that separation and gauge-fixing coincide.

I Error-corrected $\overline{Z}$-merge with the Shor code

In this appendix we work through an example explicitly, using the techniques of Section 6 to perform a distance 3 error-corrected $\overline{Z} \otimes \overline{Z}$ measurement between two copies of the Shor code, for which see Example 4.6.

Let us say the two copies are labelled $(C_\bullet, C^\bullet)$ and $(D_\bullet, D^\bullet)$, with

$$C_\bullet = D_\bullet = \; \mathbb{F}_2^6 \xrightarrow{\partial_2} \mathbb{F}_2^9 \xrightarrow{\partial_1} \mathbb{F}_2^2$$

and

$$\partial_2^{C\bullet} = \partial_2^{D\bullet} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}; \quad \partial_1^{C\bullet} = \partial_1^{D\bullet} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

We will use the $\overline{Z}$ operator $Z_1 \otimes Z_4 \otimes Z_7$, denoted $u = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}^\mathsf{T}$, with $u \in C_1$ and $u \in D_1$, to glue along.

The logical operator subcomplex $V_\bullet$ is then

$$V_\bullet = \; \mathbb{F}_2^3 \xrightarrow{\partial_1^{V\bullet}} \mathbb{F}_2^2$$

with $\partial_1^{V\bullet} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$ and all other components of $V_\bullet$ being 0.

We now make the tensor product chain complex $W_\bullet = (P \otimes V)_\bullet$ from Definition 6.4, where $P_\bullet = \; P_1 \xrightarrow{\begin{pmatrix} 1 \\ 1 \end{pmatrix}} P_0$. We have

$$W_\bullet = \; \mathbb{F}_2^3 \xrightarrow{\partial_2^{W\bullet}} \mathbb{F}_2^8 \xrightarrow{\partial_1^{W\bullet}} \mathbb{F}_2^4$$

with

$$\partial_2^{W\bullet} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}; \quad \partial_1^{W\bullet} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

For $T_\bullet$ we take the two pushouts from Definition 6.6. First, we have

$$\begin{array}{ccc} V_\bullet & \xrightarrow{g_\bullet} & C_\bullet \\ {\scriptstyle f_\bullet}\downarrow & \ulcorner & \downarrow{\scriptstyle q_\bullet} \\ W_\bullet & \xrightarrow[p_\bullet]{} & R_\bullet \end{array}$$

Giving

$$R_\bullet = \mathbb{F}_2^9 \xrightarrow{\partial_2^{R_\bullet}} \mathbb{F}_2^{14} \xrightarrow{\partial_1^{R_\bullet}} \mathbb{F}_2^4$$

with $R_2 = W_2 \oplus C_2$, as $V_2 = 0$. The other components of $R_\bullet$ require taking quotients, identifying elements of W_1 and C_1, and the same for W_0 and C_0. One can then use Definition 5.1 to show that

$$\partial_2^{R_\bullet} = \begin{pmatrix} 1&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0 \\ 1&1&0&0&0&0&0&0&0 \\ 1&0&1&0&0&0&0&0&0 \\ 1&0&0&1&1&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0 \\ 0&1&0&0&0&1&1&0&0 \\ 0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0&0 \\ 0&0&1&0&0&0&0&1&1 \\ 0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&1 \end{pmatrix}; \quad \partial_1^{R_\bullet} = \begin{pmatrix} 0&0&0&1&0&1&1&1&1&1&1&0&0&0 \\ 0&0&0&0&1&1&1&1&0&0&0&1&1&1 \\ 1&1&0&1&0&0&0&0&0&0&0&0&0&0 \\ 1&0&1&0&1&0&0&0&0&0&0&0&0&0 \end{pmatrix}.$$

For the second pushout, that is

$$\begin{array}{ccccc} V_\bullet & \hookrightarrow & W_\bullet & \longrightarrow & R_\bullet \\ \downarrow & & & & \downarrow \\ D_\bullet & & \longrightarrow & & T_\bullet \end{array}$$

we then have

$$T_\bullet = \mathbb{F}_2^{15} \xrightarrow{\partial_2^{T_\bullet}} \mathbb{F}_2^{20} \xrightarrow{\partial_1^{T_\bullet}} \mathbb{F}_2^4 \ .$$

The differentials are somewhat unwieldy, but we include them for completeness:

$$\partial_2^{T_\bullet} = \begin{pmatrix}
1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&1&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&1&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&1&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{pmatrix}$$

$$\partial_1^{T_\bullet} = \begin{pmatrix}
1&0&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&1&1&1&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&1&1&1
\end{pmatrix}$$

One can check the various properties of this code. For example, $\mathrm{rank}\partial_1^{T_\bullet} = 4$ and $\mathrm{rank}\partial_2^{T_\bullet} = 15$. Thus $\dim H_1(T_\bullet) = \dim T_1 - 4 - 15 = 1$, and so the code $(T_\bullet, T^\bullet)$ encodes one logical qubit.

We can compare this with $(C \oplus D)_\bullet$ from before the merge:

$$(C \oplus D)_\bullet = \mathbb{F}_2^{12} \xrightarrow{\partial_2^{(C\oplus D)_\bullet}} \mathbb{F}_2^{18} \xrightarrow{\partial_1^{(C\oplus D)_\bullet}} \mathbb{F}_2^{4}$$

where the differentials are easy to see from those of $C_\bullet$ and $D_\bullet$, with $\partial_2^{(C\oplus D)_\bullet} = \partial_2^{C_\bullet} \oplus \partial_2^{D_\bullet}$ etc. Evidently, $((C\oplus D)_\bullet, (C\oplus D)^\bullet)$ encodes 2 logical qubits. As expected, there are 2 new qubits, and 3 new Z-measurements in $(T_\bullet, T^\bullet)$. Each of the 2 new qubits participates in 2 of the new Z-measurements (and no other Z-measurements). We can check that $d_T^Z \geq 3$, i.e. the code has distance bounded below.

For the error-corrected $\overline{Z} \otimes \overline{Z}$ measurement, we therefore start with the code $((C \oplus D)_\bullet, (C \oplus D)^\bullet)$. Recall that this has $d = 3$. We then initialise the 2 new qubits in the $|+\rangle$ state and measure 3 rounds of the stabilisers specified by $\partial_2^{T_\bullet}$ and $\partial_1^{T_\bullet}$. As the 2 new qubits each participate in 2 of the new Z-measurements, the product of the outcomes is insensitive to initialisation errors. We apply the gauge-fixing operators from Example 6.2 to correct for the 3 new Z-measurements which may output the -1 measurement outcome. We end up with the code $(T_\bullet, T^\bullet)$.