A Three-Way Normal Form for Stabiliser Codes across ZX Diagrams, Circuits, and Tableaus

Lia Yeh Jiaxin Huang Aleks Kissinger Sarah Meng Li John van de Wetering

Motivation and Background Utility-scale quantum computing relies fundamentally on quantum error-correcting codes (QECCs) to achieve fault-tolerant quantum computation (FTQC) [31, 40, 51, 52]. Most fault-tolerant schemes are based on *stabiliser codes*, defined as the joint $+1$ eigenspace of an abelian subgroup of the Pauli group. By repeatedly measuring these stabilisers, errors on the code space can be detected and corrected. Calderbank-Shor-Steane (CSS) codes form an important subclass constructed from classical codes [7, 8, 52, 54], with generators consisting purely of X-type or Z-type Pauli operators. While this algebraic framework is powerful, it can be rigid and difficult to interpret intuitively. Topological approaches provide geometric intuition for specific code families [2, 24, 42], but they lack a general mechanism for formal logical reasoning.

As a graphical language for representing quantum states and operations [11, 12, 13, 60], ZX-calculus enables formal graphical reasoning to accurately depict linear transformations. Since it provides a precise representation of a QECC encoder [33, 37], any relations proven using the ZX-calculus are true properties of the code. Thanks to this advantage, ZX-calculus is used to graphically reason about QECC properties [20, 25] and tripartite coherent parity check codes [9, 10]. For CSS codes, ZX-calculus is applied to study code transformations [34], code maps and code surgeries [6, 16], surface code constructions [26, 27, 28, 50], and high-dimensional QECCs [15].

These developments show that the stabiliser formalism and the ZX-calculus provide complementary perspectives on the same underlying structures. While the former gives an algebraic description of QECCs in terms of Pauli groups and symplectic linear algebra, the latter offers a compositional graphical language with a complete equational theory for Clifford quantum mechanics. Despite this shared scope, current links between the two remain incomplete: existing correspondences typically apply only to special subclasses of codes or describe codes up to certain equivalences.

In particular, previous approaches to relating stabiliser codes and ZX-diagrams face important structural limitations. Correspondences for CSS codes rely on the block-diagonal form of the stabiliser tableau, where generators separate into purely X-type and purely Z-type operators, as depicted in the unified representations of Steane code in Figure 1. Although effective for CSS constructions, this structure does not extend naturally to general stabiliser codes with mixed generators, and therefore fails to provide a uniform treatment of arbitrary Clifford isometries encountered in fault-tolerant protocols.

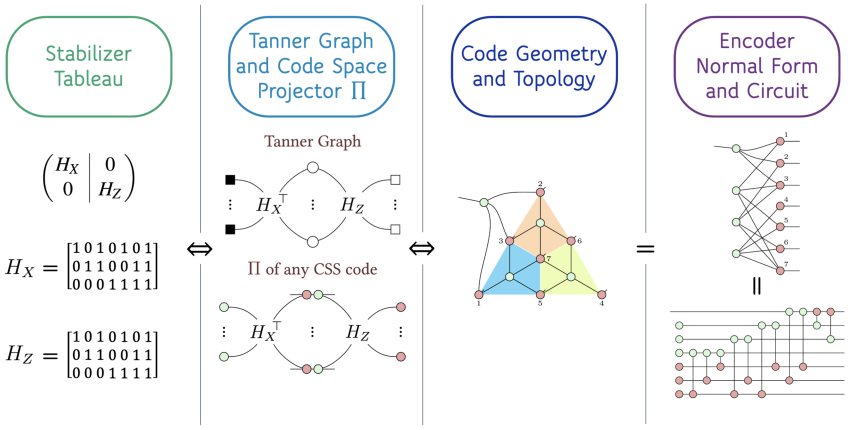


Figure 1: Unify different representations of Stabiliser codes through the lens of ZX-calculus.

Second, graph-state-based normal forms, such as the GSLC (graph state with local Cliffords) form, characterise stabiliser states and codes only up to local Clifford equivalence. Although this is sufficient for many structural

questions, it is inadequate in settings where the precise Clifford structure is operationally relevant. For example, when analysing biased noise models, studying transversal gate implementations, or tracking the action of logical operators through encoders, the distinction between locally Clifford-equivalent representations becomes essential. Identifying codes only up to local equivalence therefore obscures information that is crucial in fault-tolerant contexts.

Our goal is to remove these restrictions by developing an "on-the-nose" normal form that faithfully captures the full stabiliser tableau, rather than merely its equivalence class under local Cliffords. In doing so, we aim to provide a canonical graphical representation that retains the complete algebraic and circuit-level structure of the code.

Contributions In this work, we establish a canonical correspondence between stabiliser tableaus, encoder circuits, and ZX-diagrams. Building on the Affine-with-Phases (AP) normal form, we prove that every stabiliser code admits a unique ZX normal form in one-to-one correspondence with a canonical stabiliser tableau, extending previous results that were limited to CSS codes or to stabiliser states up to local Clifford equivalence. From this correspondence, we derive efficient algorithms to interconvert between graphical, circuit-based, and algebraic representations for arbitrary stabiliser codes. Figure 2 summarises the translation procedures between these representations. The technical manuscript is attached in this submission.

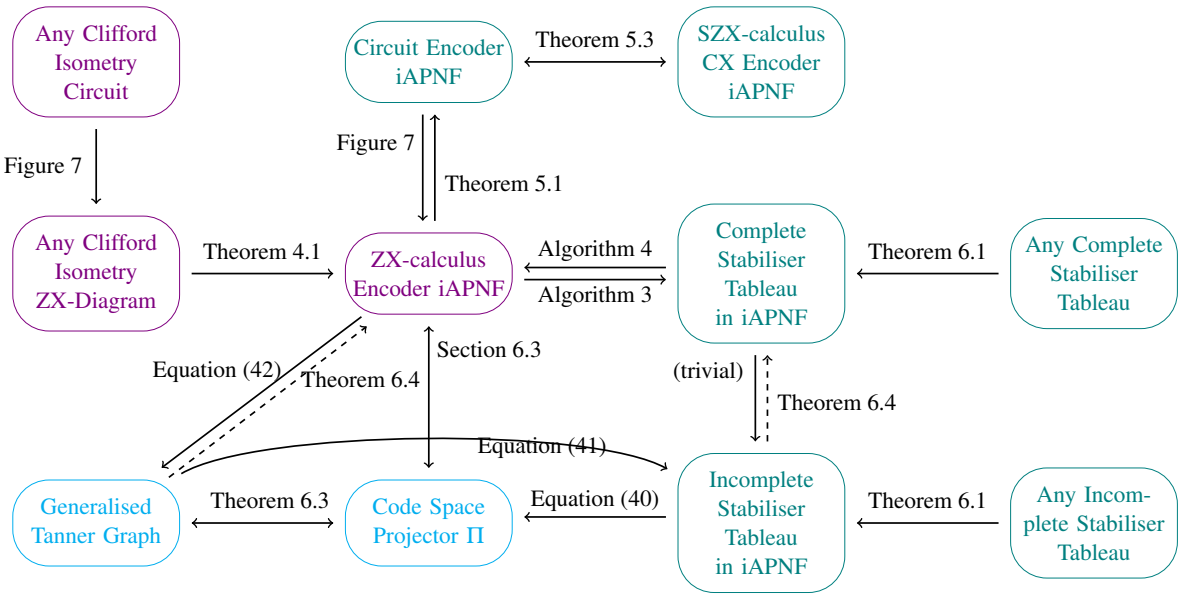


Figure 2: Through the Affine-with-Phases normal form, we reveal the mathematical connection between four prevalent representations of a stabiliser code: ZX-diagrams, quantum circuits, stabiliser tableaus, and what we call generalised Tanner graphs. Every arrow in the diagram corresponds to a procedure and its correctness proof in [61]. The dashed arrows indicate that a choice had to be made to fix some degrees of freedom; for instance, when going from incomplete to complete stabiliser tableaus, a choice of logical operators must be made.

In what follows, we systematically lift diagrammatic correspondences from restricted code families to a fully general, canonical treatment of stabiliser encoders.

From CSS Codes to Phase-Free Normal Forms We begin by revisiting CSS codes and their representation as phase-free ZX-diagrams. Using the Generalised Parity Form (GPF), we show that any CSS encoder can be rewritten into a canonical bipartite diagram whose connectivity directly encodes the X-type and Z-type stabiliser generators. In this setting, reading off the stabiliser tableau amounts to extracting biadjacency matrices from the diagram. Conversely, given the stabiliser tableau of a CSS code, one can reconstruct the corresponding GPF diagram constructively.

This establishes a strict correspondence between CSS tableaus and phase-free ZX normal forms, clarifying and formalising earlier diagrammatic constructions.

Graph Codes and GSLC Form Next, we analyse graph codes and their representation in Graph State with Local Cliffords (GSLC) form. Any stabiliser code is equivalent to a graph code up to local Clifford transformations. In GSLC form, one can read off stabiliser generators by associating each vertex with an X operator on that vertex and Z operators on its neighbours. However, the presence of local Cliffords obscures the precise logical structure and prevents a direct canonical correspondence with tableaus. We demonstrate how to systematically rewrite GSLC diagrams into forms where linearly independent stabiliser generators can be extracted, thereby preparing the ground for a fully general construction.

Affine-with-Phases (AP) Normal Form for Isometries The central technical contribution of this work is the extension of the Affine-with-Phases (AP) normal form from stabiliser states to stabiliser isometries, that is, encoder circuits. The AP normal form represents a stabiliser state as a uniform superposition over an affine subspace together with a diagonal Clifford component. We generalise this construction to the isometry setting, thereby capturing the full encoder of any stabiliser code in a canonical ZX-diagram.

We prove three key results. First, any stabiliser encoder can be efficiently rewritten—using ZX rewrite rules—to AP normal form in polynomial time. Second, the reduced AP normal form is unique. Third, the parameters of the AP form directly determine a canonical stabiliser tableau. Together, these results establish a precise and algorithmic correspondence between encoder circuits, ZX-diagrams, and stabiliser tableaus.

In particular, we show that any stabiliser tableau can be written in the normal form $T = \left(\begin{array}{c|c} H_X & H_X G \\ 0 & H_Z \end{array}\right)$, where H_X and H_Z encode the affine connectivity and G is a symmetric matrix encoding the diagonal Clifford component. Diagrammatically, H_X and H_Z correspond to the bipartite structure of the ZX-diagram, while G specifies a layer of S and CZ gates. This establishes a precise correspondence:

$$\text{ZX AP normal form} \longleftrightarrow \text{canonical stabiliser tableau} \longleftrightarrow \text{encoder circuit}.$$

Three-Way Translation Algorithms Our framework gives explicit and efficient procedures for extracting a complete stabiliser tableau from a ZX-diagram in AP normal form and for constructing a valid encoder circuit from any stabiliser tableau. It provides translations between tableau, circuit, and graphical representations while remaining entirely within the stabiliser fragment. All reductions are implemented via polynomial-time graphical Gaussian elimination procedures expressed as ZX rewrites. Moreover, equality of stabiliser diagrams reduces to comparison of their reduced AP parameters, providing an efficient graphical decision procedure.

Conclusion In this work, we present the first fully canonical ZX normal form for arbitrary stabiliser codes and prove that it is in one-to-one correspondence with a canonical stabiliser tableau, thereby unifying three standard descriptions of stabiliser codes—symplectic stabiliser tableaus, Clifford encoder circuits, and ZX-diagrams—within a single framework. By extending Affine-with-Phases normal form to isometries, this unification holds at the level of encoder maps rather than only states. The resulting normal form has four key properties. First, it is canonical, meaning that it gives a unique representation of the stabiliser code. Second, it is faithful, as it captures the exact Clifford isometry rather than only a local-Clifford equivalence class. Third, it is compositional since it lives entirely within ZX rewriting and is compatible with map-state-duality. Fourth, it is algorithmic because translations can be carried out efficiently and constructively using graphical Gaussian elimination and reductions to the reduced AP form. This diagrammatic framework has several practical uses. It lets us read off logical and stabiliser operators directly from the diagram, which helps in studying transversal gates. It also lets us describe code switching and code deformation as ZX transformations. In addition, it extends structural results from CSS codes to general stabiliser codes, and it fits naturally with automated ZX rewriting for code search, protocol verification, and compilation. More broadly, it shows the potential of diagrammatic methods as a framework for fault-tolerant quantum computing.

A Three-Way Normal Form for Stabiliser Codes across ZX Diagrams, Circuits, and Tableaus

Lia Yeh Jiaxin Huang Aleks Kissinger John van de Wetering
Sarah Meng Li

The ZX-calculus and stabiliser formalism are two ubiquitous multi-tools in quantum computation. Here, we unify these reasoning schemes by establishing a three-way translation between a general stabiliser code, its canonical ZX normal form, and its encoder circuit. We achieve this by extending the Affine-with-Phases normal form from stabiliser states to arbitrary Clifford isometries, and by providing efficient algorithms for constructing it, both graphically in the ZX-calculus and algebraically via tableau operations. This extension yields a canonical normal form for stabiliser tableaus, which corresponds to a factorisation of any stabiliser code into an inner CSS code together with a diagonal Clifford unitary. This unified perspective provides a structurally coherent view of quantum error-correcting codes and paves the way for automated design pipelines for fault-tolerant quantum computation.

Contents

1 Introduction

Motivation and Background Utility-scale quantum computing relies fundamentally on quantum error-correcting codes (QECCs) to achieve fault-tolerant quantum computation (FTQC) [31, 40, 51, 52]. Most fault-tolerant schemes are based on *stabiliser codes*, defined as the joint $+1$ eigenspace of an abelian subgroup of the Pauli group. By repeatedly measuring these stabilisers, errors on the code space can be detected and corrected. Calderbank-Shor-Steane (CSS) codes form an important subclass constructed from classical codes [7, 8, 52, 54], with generators consisting purely of X-type or Z-type Pauli operators. While this algebraic framework is powerful, it can be rigid and difficult to interpret intuitively. Topological approaches provide geometric intuition for specific code families [2, 24, 42], but they lack a general mechanism for formal logical reasoning.

As a graphical language for representing quantum states and operations [11, 12, 13, 60], ZX-calculus enables formal graphical reasoning to accurately depict linear transformations. Since it provides a precise representation of a QECC encoder [33, 37], any relations proven using the ZX-calculus are true properties of the code. Thanks to this advantage, ZX-calculus is used to graphically reason about QECC properties [20, 25] and tripartite coherent parity check codes [9, 10]. For CSS codes, ZX-calculus is applied to study code transformations [34], code maps and code surgeries [6, 16], surface code constructions [26, 27, 28, 50], and high-dimensional QECCs [15].

These developments show that the stabiliser formalism and the ZX-calculus provide complementary perspectives on the same underlying structures. While the former gives an algebraic description of QECCs in terms of Pauli groups and symplectic linear algebra, the latter offers a compositional graphical language with a complete equational theory for Clifford quantum mechanics. Despite this shared scope, current links between the two remain incomplete: existing correspondences typically apply only to special subclasses of codes or describe codes up to certain equivalences.

In particular, previous approaches to relating stabiliser codes and ZX-diagrams face important structural limitations. Correspondences for CSS codes rely on the block-diagonal form of the stabiliser tableau, where generators separate into purely X-type and purely Z-type operators, as depicted in the unified representations of Steane code in Figure 1. Although effective for CSS constructions, this structure does not extend naturally to general stabiliser codes with mixed generators, and therefore fails to provide a uniform treatment of arbitrary Clifford isometries encountered in fault-tolerant protocols.

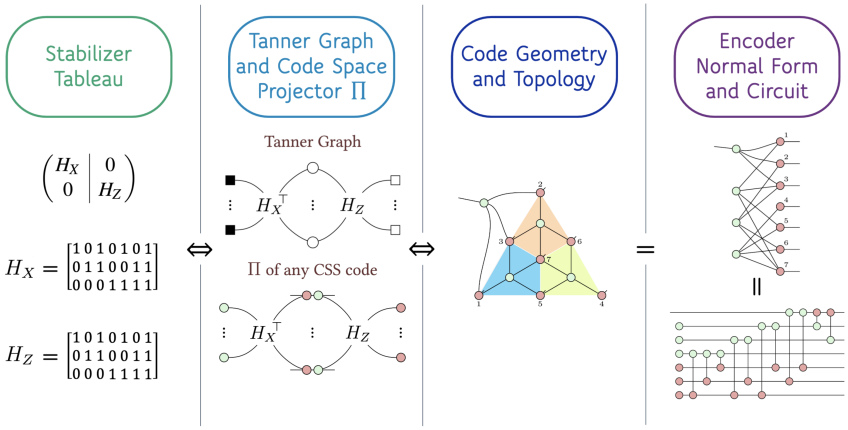


Figure 3: Unify different representations of Stabiliser codes through the lens of ZX-calculus.

Second, graph-state-based normal forms, such as the GSLC (graph state with local Cliffords) form, characterise stabiliser states and codes only up to local Clifford equivalence. Although this is sufficient for many structural questions, it is inadequate in settings where the precise Clifford structure is operationally relevant. For example, when analysing biased noise models, studying transversal gate implementations, or tracking the action of logical operators through encoders, the distinction between locally Clifford-equivalent representations becomes essential. Identifying codes only up to local equivalence therefore obscures information that is crucial in fault-tolerant contexts.

Our goal is to remove these restrictions by developing an "on-the-nose" normal form that faithfully captures the full stabiliser tableau, rather than merely its equivalence class under local Cliffords. In doing so, we aim to provide a canonical graphical representation that retains the complete algebraic and circuit-level structure of the code.

Contributions In this work, we establish a canonical correspondence between stabiliser tableaus, encoder circuits, and ZX-diagrams. Building on the Affine-with-Phases (AP) normal form, we prove that every stabiliser code admits a unique ZX normal form in one-to-one correspondence with a canonical stabiliser tableau, extending previous results that were limited to CSS codes or to stabiliser states up to local Clifford equivalence. From this correspondence, we derive efficient algorithms to interconvert between graphical, circuit-based, and algebraic representations for arbitrary stabiliser codes. Figure 2 summarises the translation procedures between these representations. The technical manuscript is attached in this submission.

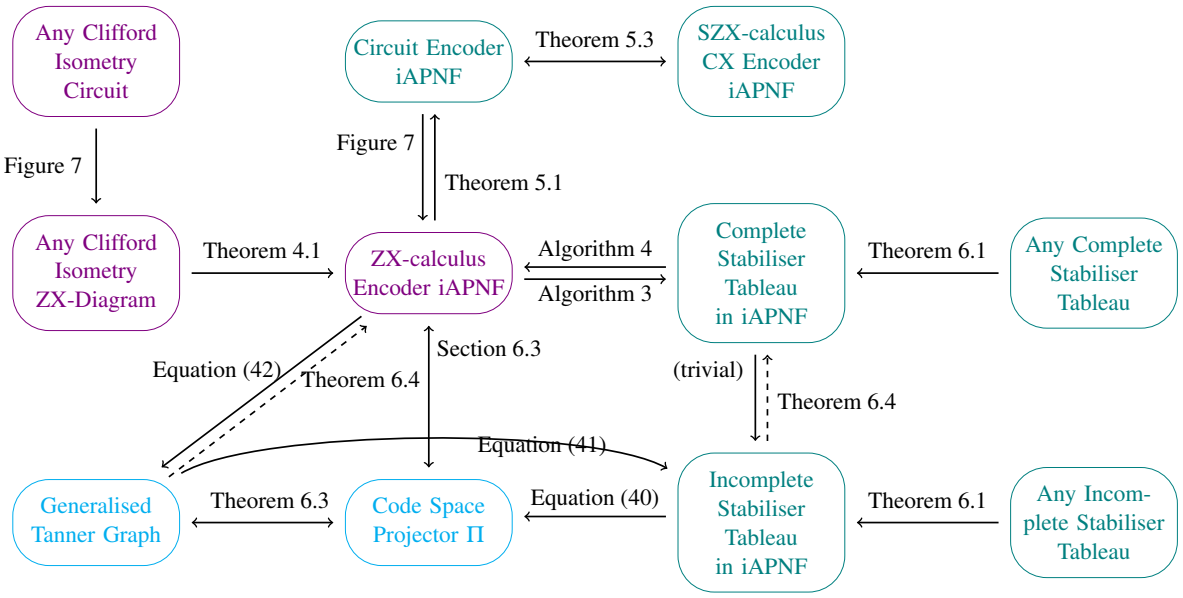


Figure 4: Through the Affine-with-Phases normal form, we reveal the mathematical connection between four prevalent representations of a stabiliser code: ZX-diagrams, quantum circuits, stabiliser tableaus, and what we call generalised Tanner graphs. Every arrow in the diagram corresponds to a procedure and its correctness proof in [61]. The dashed arrows indicate that a choice had to be made to fix some degrees of freedom; for instance, when going from incomplete to complete stabiliser tableaus, a choice of logical operators must be made.

In what follows, we systematically lift diagrammatic correspondences from restricted code families to a fully general, canonical treatment of stabiliser encoders.

From CSS Codes to Phase-Free Normal Forms We begin by revisiting CSS codes and their representation as phase-free ZX-diagrams. Using the Generalised Parity Form (GPF), we show that any CSS encoder can be rewritten into a canonical bipartite diagram whose connectivity directly encodes the X-type and Z-type stabiliser generators. In this setting, reading off the stabiliser tableau amounts to extracting biadjacency matrices from the diagram. Conversely, given the stabiliser tableau of a CSS code, one can reconstruct the corresponding GPF diagram constructively. This establishes a strict correspondence between CSS tableaus and phase-free ZX normal forms, clarifying and formalising earlier diagrammatic constructions.

Graph Codes and GSLC Form Next, we analyse graph codes and their representation in Graph State with Local Cliffords (GSLC) form. Any stabiliser code is equivalent to a graph code up to local Clifford transformations. In GSLC form, one can read off stabiliser generators by associating each vertex with an X operator on that vertex and Z operators on its neighbours. However, the presence of local Cliffords obscures the precise logical structure and prevents a direct canonical correspondence with tableaus. We demonstrate how to systematically rewrite GSLC diagrams into forms where linearly independent stabiliser generators can be extracted, thereby preparing the ground for a fully general construction.

Affine-with-Phases (AP) Normal Form for Isometries The central technical contribution of this work is the extension of the Affine-with-Phases (AP) normal form from stabiliser states to stabiliser isometries,

that is, encoder circuits. The AP normal form represents a stabiliser state as a uniform superposition over an affine subspace together with a diagonal Clifford component. We generalise this construction to the isometry setting, thereby capturing the full encoder of any stabiliser code in a canonical ZX-diagram.

We prove three key results. First, any stabiliser encoder can be efficiently rewritten—using ZX rewrite rules—to AP normal form in polynomial time. Second, the reduced AP normal form is unique. Third, the parameters of the AP form directly determine a canonical stabiliser tableau. Together, these results establish a precise and algorithmic correspondence between encoder circuits, ZX-diagrams, and stabiliser tableaus.

In particular, we show that any stabiliser tableau can be written in the normal form $T = \left(\begin{array}{c|c} H_X & H_X G \\ 0 & H_Z \end{array}\right)$, where H_X and H_Z encode the affine connectivity and G is a symmetric matrix encoding the diagonal Clifford component. Diagrammatically, H_X and H_Z correspond to the bipartite structure of the ZX-diagram, while G specifies a layer of S and CZ gates. This establishes a precise correspondence:

$$\text{ZX AP normal form} \longleftrightarrow \text{canonical stabiliser tableau} \longleftrightarrow \text{encoder circuit}.$$

Bidirectional Translation Algorithms Our framework yields explicit and efficient procedures for extracting a complete stabiliser tableau from a ZX-diagram in AP normal form, constructing a valid encoder circuit from any stabiliser tableau, and translating between tableau, circuit, and graphical representations without leaving the stabiliser fragment. All reductions are implemented via polynomial-time graphical Gaussian elimination procedures expressed as ZX rewrites. Moreover, equality of stabiliser diagrams reduces to comparison of their reduced AP parameters, providing an efficient graphical decision procedure.

Generalised Tanner Graphs and Projectors We further show that the projector onto the codespace admits a natural description as a generalised Tanner graph derived from the AP normal form. This clarifies the relationship between parity-check matrices, graphical encodings, and ZX structure, and provides a diagrammatic proof of the Fundamental Theorem of Stabiliser Theory within the ZX framework.

Our results unify three previously distinct viewpoints on stabiliser codes: the algebraic description in terms of symplectic stabiliser tableaus, the circuit-theoretic perspective given by Clifford encoders, and the graphical representation provided by ZX-diagrams. As emphasised in the introduction and illustrated in Figure 2 of the paper, these three representations are typically studied in isolation, each with its own strengths and technical tools. By establishing a canonical ZX normal form that is in one-to-one correspondence with a canonical stabiliser tableau and a Clifford encoder circuit, we show that they are in fact different manifestations of the same underlying structure. This unification makes precise the relationship between graph-based and tableau-based reasoning and places them within a single diagrammatic framework.

The resulting ZX normal form has several key structural properties. First, it is canonical: once reduced to the appropriate form, it is unique, mirroring the uniqueness of a stabiliser tableau in reduced row echelon form. Second, it is faithful: unlike GSLC-based descriptions that identify codes only up to local Cliffords, our normal form captures the exact Clifford isometry and therefore retains the full logical and stabiliser structure. Third, it is compositional, since it lives entirely within the rewrite theory of the ZX-calculus and is compatible with standard diagrammatic transformations such as Only Connectivity Matters (OCM). Finally, it is algorithmic: the conversion procedures between tableaus, circuits, and ZX-diagrams are efficient and constructive, relying on graphical Gaussian elimination and reductions to (reduced) Affine-with-Phases form as developed in the main body of the paper.

This unified and canonical perspective opens several avenues for application. Because logical and stabiliser operators can be read off diagrammatically, one can systematically study transversal gates by pushing Pauli operators through encoder diagrams, extending techniques already available for CSS codes to arbitrary stabiliser codes. Similarly, code switching and code deformation protocols can be analysed as graphical transformations within the ZX-calculus, providing a high-level and visually intuitive framework for comparing different code families. The normal form also enables the transfer of structural results from CSS codes to general stabiliser codes by identifying CSS structure as a special case of the Affine-with-Phases decomposition. Finally, since the normal form is compatible with automated ZX-rewriting tools, it supports the integration of stabiliser-code reasoning directly into ZX-based quantum compilers and verification pipelines, strengthening the bridge between quantum error correction and diagrammatic quantum reasoning.

Conclusion We provide the first fully canonical ZX normal form for arbitrary stabiliser codes and prove its one-to-one correspondence with a canonical stabiliser tableau. By extending Affine-with-Phases normal form to isometries, we unify stabiliser theory and the ZX-calculus at the level of encoders rather than merely states. This establishes a common diagrammatic foundation for QECC design, fault-tolerant compilation, and graphical reasoning in quantum computation, and in turn provides a foundation for new automated tools for code search, protocol verification, and transversal-gate analysis. More broadly, it reinforces the role of categorical and diagrammatic methods in fault-tolerant quantum computing.

2 Preliminaries

In this section, we introduce notions, conventions, and basics of ZX-diagrams that will be used in the remainder of this paper.

2.1 Stabiliser Formalism

Here, we give a succinct overview of the Pauli and Clifford groups, and how they are used to define stabiliser states.

For any $m \in \mathbb{N}$, $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$ and it is equipped with the standard addition modulo m. We fix $i = e^{2\pi/4}$ as the *primitive fourth root of unity*. For simplicity, RHS is short for *the righthand side* and LHS is short for *the lefthand side*. Let $g_1, \ldots, g_m$ be unitary operators such that $g_i g_j = g_j g_i$, for any $1 \leq i, j \leq m$. $\mathcal{W} = \langle g_1, \ldots, g_m \rangle$ denotes a collection of circuits composed of $g_1, \ldots, g_m$, via matrix multiplication and tensor product. In what follows, we use operators, unitaries, matrices, and quantum gates interchangeably.

Definition 2.1. The *single-qubit Pauli matrices* are defined as

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

Definition 2.2. For $n \in \mathbb{N}$, the *n-qubit Pauli group* $\mathcal{P}_n$ is defined as

$$\mathcal{P}_n = \{i^k P_1 \otimes \cdots \otimes P_n;\ k \in \mathbb{Z}_4,\ P_j \in \{I, X, Y, Z\}\}. \tag{1}$$

Definition 2.3. $\overrightarrow{P} = i^k P_1 \otimes \cdots \otimes P_n$ is called a *Pauli string* with *phase* i^k. It can be represented by a $2n$-bit string, with a phase p at the end, $p \in \{i^k;\, k \in \mathbb{Z}_4\}$.

$$\overrightarrow{P} = (\overrightarrow{x} \mid \overrightarrow{z})p,\ \overrightarrow{x}, \overrightarrow{z} \in \mathbb{Z}_2^n.$$

$\overrightarrow{x}$ and $\overrightarrow{z}$ indicate the participation of Paulis X and Z in $\overrightarrow{P}$ respectively. $\overrightarrow{P}$ acts as a X (Z) on the ith qubit when the ith entry of $\overrightarrow{x}$ ($\overrightarrow{z}$) is 1, and I otherwise.

In what follows, we use Pauli strings and Pauli operators interchangeably. For the expression in (1), we ignore $\otimes$ in the composition of $\overrightarrow{P}$ and simply write $\overrightarrow{P} = i^k P_1 \ldots P_n$. For $1 \leq j \leq n$, P_j denotes a single-qubit Pauli (I, X, Y, or Z) acting on qubit j. I_n denotes an identity operator acing on n qubits. When it is clear from the context, we drop the subscript and simply write I instead of I_n.

When k is even, $\overrightarrow{P}$ is *Hermitian* and $\overrightarrow{P}^2 = I$. For our purpose, we are interested in the Hermitian Pauli strings. When $k = 0$ (or $k = 1$), $\overrightarrow{P}$ defines a projector onto the $+1$ (or -1) eigenspace of $\overrightarrow{P}$.

Definition 2.4. For a Hermitian Pauli string $\overrightarrow{P}$, we define the associated *Pauli projectors* as

$$\Pi_{\overrightarrow{P}}^{(k)} = \frac{1}{2}(I + (-1)^k \overrightarrow{P}),\ k \in \mathbb{Z}_2.$$

Definition 2.5. For $n \in \mathbb{N}$, the *n-qubit Clifford group* is defined as the set of all n-qubit unitaries U satisfying

$$\forall P \in \mathcal{P}_n, \exists Q \in \mathcal{P}_n \text{ s.t. } U^\dagger P U = Q \tag{2}$$

The Pauli group necessarily is a subgroup of the Clifford group.

We refer to unitary circuits composed of all Clifford gates as *Clifford circuits*, and isometries composed of Clifford circuits acting on identity wire(s) tensored with any number of qubits in the $|0\rangle$ state as *Clifford isometries*.

Any Clifford circuit (up to an irrelevant global phase) admits decomposition into just the H, S, and CNOT gates. Their conventional definitions, along with that of the CZ gate, are as follows.

Definition 2.6. The single-qubit *H (Hadamard) and S gates*, the two-qubit *CNOT and CZ gates* are defined as

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \qquad S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}, \qquad \text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \qquad \text{CZ} = \text{diag}(1,1,1,-1).$$

Definition 2.7. A *CNOT* circuit is a circuit composed of only CNOT gates.

Definition 2.8. Conjugating a CZ gate by an H gate on both qubits gives us the *DX gate*. This stands for "diagonal in X", as DX is diagonal in the X basis.

Definition 2.9. A state ψ is *stabilised* by an operator S if

$$S|\psi\rangle = |\psi\rangle \tag{3}$$

Definition 2.10. Let $|\psi\rangle$ be a non-zero n-qubit state. $|\psi\rangle$ is a stabiliser state if and only if it is stabilised by n independent operators in $\mathcal{P}_n$:

$$\mathcal{S} = \langle S_1, \ldots, S_n \rangle \iff \text{Stab}(\mathcal{S}) = \{|\psi\rangle;\, S_j|\psi\rangle = |\psi\rangle, 1 \leq j \leq n\}.$$

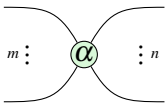
Lastly, we introduce a linear algebraic notion that will be used later for defining the normal form of an arbitrary stabiliser code.

Definition 2.11. An *affine subspace* $S' \subseteq \mathbb{Z}_2^n$ is either:

- the empty set, or
- a set of the form $\vec{w}+S := \{\vec{w}+\vec{v} \,|\, \vec{v} \in S\}$ for some fixed vector $\vec{w} \in \mathbb{Z}_2^n$ and a linear subspace $S \subseteq \mathbb{Z}_2^n$.

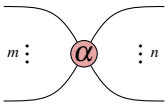
2.2 The ZX-Calculus

The ZX-calculus is a graphical calculus for diagrammatic reasoning of any qubit quantum computation [11, 12, 13, 60]. Every diagram in the calculus is composed of two types of generators: Z spiders, which sum over the eigenbasis of the Pauli Z operator:



$$:= |0\rangle^{\otimes n}\langle 0|^{\otimes m} + e^{i\alpha}|1\rangle^{\otimes n}\langle 1|^{\otimes m}, \tag{4}$$

and X spiders, which sum over the eigenbasis of the Pauli X operator:



$$:= |+\rangle^{\otimes n}\langle +|^{\otimes m} + e^{i\alpha}|-\rangle^{\otimes n}\langle -|^{\otimes m}. \tag{5}$$

The ZX-calculus is *universal* [12] in the sense that any linear map from m qubits to n qubits corresponds exactly to a ZX-diagram, using the generators in Equations (4) and (5) and the composition of linear maps. The ZX-calculus is *complete* [32, 35] in the sense that any equality of linear maps on any number of qubits is derivable using only a finite set of rules in the calculus. The smallest complete rule set to date [56] is shown in Figure 16.

In this work, we restrict the ZX-calculus to two families of ZX-diagrams that is closed under composition and tensor product: *the phase-free ZX-calculus* and *the Clifford ZX-calculus*. They are also known as *the ZX fragments*.

Definition 2.12. A *phase-free ZX-diagram* is a ZX-diagram where all of its spiders have zero phases. When a spider has phase zero, its phase is omitted.

The rules in Figure 5 is complete for the phase-free ZX fragments.

Definition 2.13. A *Clifford ZX-diagram* is a ZX-diagram where all the phases on the spiders are integer multiples of $\pi/2$, $k \in \mathbb{Z}_4$.

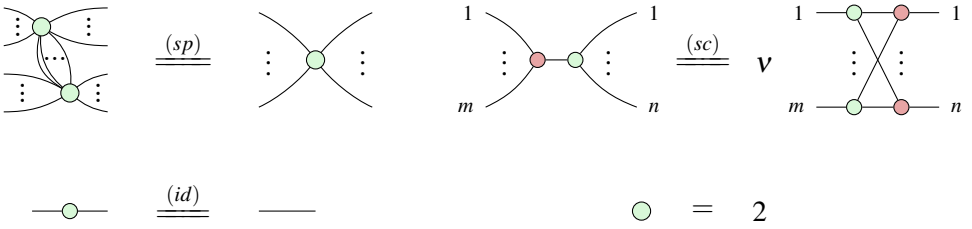


Figure 5: The complete set of rewrite rules of the phase-free ZX-calculus. "sp" is short for *spider fusion* and "sc" is short for *strong complementarity*. The RHS of the (sc) rule is a complete bipartite graph between m input Z spiders and n output X spiders. ν is a normalisation factor, $\nu = 2^{(m-1)(n-1)/2}$.

The reason these diagrams are called Clifford, is because they correspond to the same fragment of quantum computation generated by Clifford circuits and computational basis state preparation and post-selection. This fragment is called the *stabiliser fragment* of qubit quantum mechanics, because any linear maps in this fragment with zero input qubits and at least one output qubit is precisely a stabiliser state.

Up to a global phase, Figure 6 presents a complete set of rewrite rules for the Clifford ZX fragments. Based on this, Figure 7 presents the ZX-diagrams corresponding to operators given in Definitions 2.1, 2.6 and 2.8. The complete and scalar-accurate set of rewrite rules [39] can be found in Section A. Figure 9 summarises additional rules that will be convenient to use in later sections. For more about the phase-free and Clifford fragments of the ZX-calculus, we encourage readers to consult [37, 39]. In Sections 2.4 and 2.5, we discuss important properties of these two fragments and introduce various normal forms that will be used in the remainder of this paper.

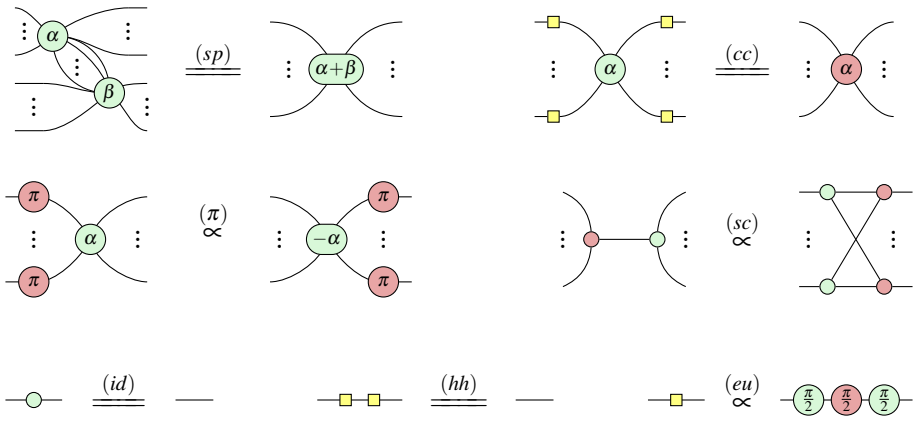


Figure 6: The complete set of rewrite rules for the Clifford ZX-calculus. $\alpha, \beta \in \{k\frac{\pi}{2};\ k \in \mathbb{Z}_4\}$. Each equation still holds when we interchange X and Z spiders. When there is a $\propto$ between two diagrams, it means the rule holds up to a scalar.

Next, we introduce a useful notation called *measurement gadgets*. They will be used in Section 4 to picture measurements of Pauli observables. Figure 8 defines the four basic Pauli boxes.

Then for a self-inverse Pauli string $\overrightarrow{P} = P_1 \otimes \ldots \otimes P_n$ with $P_j \in \{I, X, Y, Z\}$, the associated *Pauli box* is defined as follows.

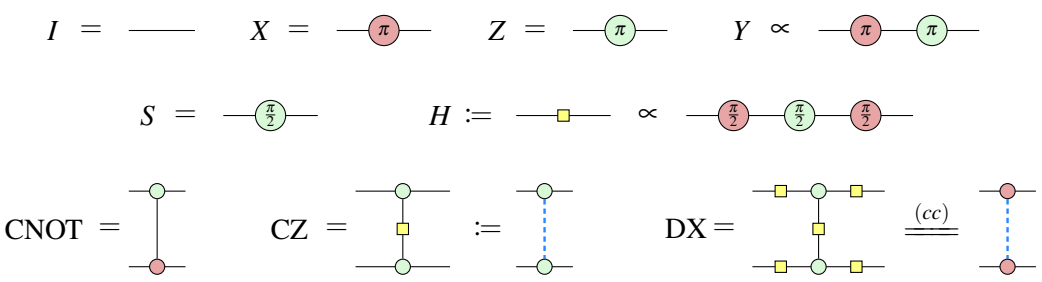


Figure 7: Some Clifford diagrams that are important for our purpose. Note that the blue dashed edge is a shorthand for the Hadamard edge.

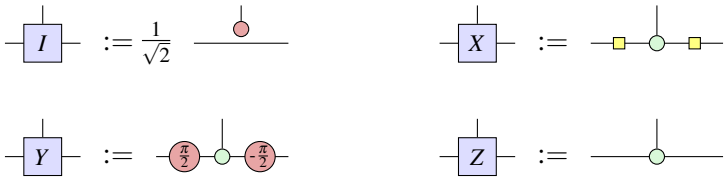
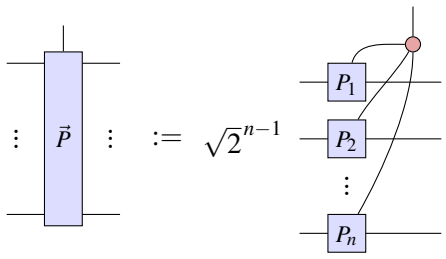
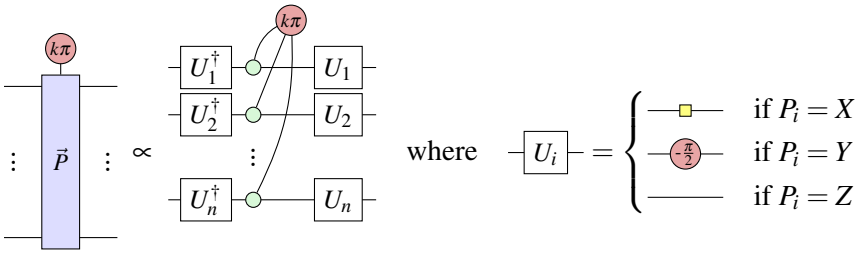


Figure 8: The four basic Pauli boxes.



(6)

Putting Figure 8 and equation (6) together, we can construct the Pauli box for any Pauli string as follows.



(7)

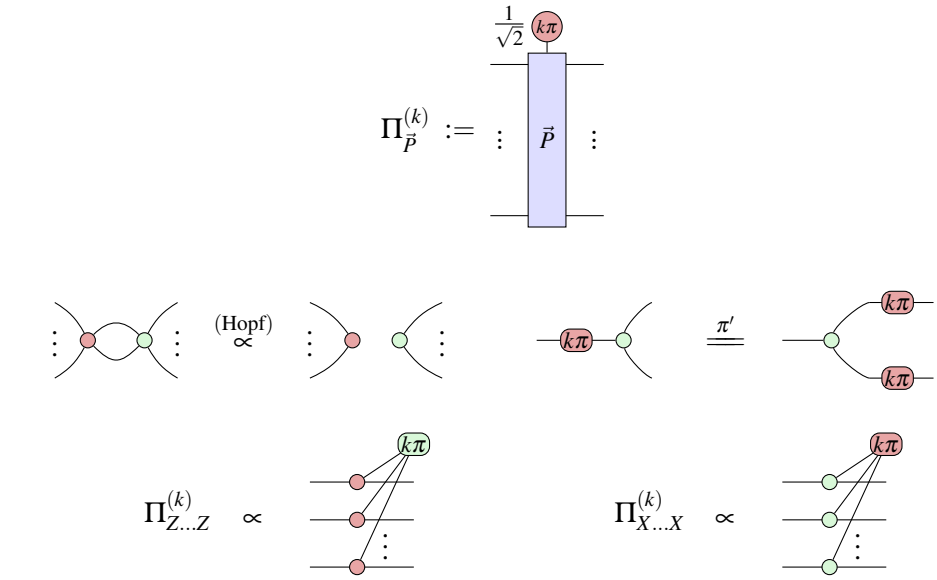
Definition 2.14. For a self-adjoint Pauli string, we define the associated *Pauli projectors* as follows.

$$\Pi_{\vec{P}}^{(0)} = \frac{1}{2}(I+\vec{P}) \qquad \Pi_{\vec{P}}^{(1)} = \frac{1}{2}(I-\vec{P}) \tag{8}$$

Proposition 2.1 ([39])**.** The maps $\Pi_{\vec{P}}^{(k)}$ for $k = 0,1$ are projectors and furthermore:

$$\vec{P}|\psi\rangle = (-1)^k|\psi\rangle \iff \Pi_{\vec{P}}^{(k)}|\psi\rangle = |\psi\rangle \tag{9}$$

It follows that the Pauli projectors given in Definition 2.14 corresponds to the following ZX diagrams [39]. In particular, the non-destructive $Z \otimes \ldots \otimes Z$ and $X \otimes \ldots \otimes X$ measurements are pictured by two measurement gadgets in Figure 9.



(10)

Figure 9: Some useful rewrite rules, each derivable from [39] and the rules in Figure 16. $k \in \mathbb{Z}_2$. Each equation still holds when we interchange X and Z spiders. These rules allow us to work with Pauli operators and non-destructive Pauli measurements in the phase-free ZX diagrams.

Lastly, we introduce a "meta-rule" in the ZX-calculus, *Only Connectivity Matter (OCM)* [60]. Because of this, we can treat ZX-diagrams based solely on their connectivity, regardless of their relative positions. In other words, we can rearrange ZX-diagrams without changing their interpretation, as long as the connectivity is preserved.

2.3 Stabiliser Codes and Their Tableau Representations

The stabiliser formalism provides the algebraic language for describing and analysing stabiliser codes [29, 31]. In this section, we extend the symplectic representation of Pauli strings to define tableau representations for general stabiliser codes. We will use three related notions of tableau: the complete stabiliser tableau, the incomplete stabiliser tableau, and the encoder tableau. Each captures different information about the code, and we will refer to them as needed throughout the paper.

To start, we review important notations of a stabiliser group.

Definition 2.15. For any $m, n \in \mathbb{N}$, $m \leq n$. Let $\{S_1, \ldots, S_m\}$ be a set of independent n-qubit Pauli strings that pairwise commute. Then $\mathcal{S} = \langle S_1, \ldots, S_m \rangle$ is called a *stabiliser group*, and S_j is called a *stabiliser generator*, for $1 \leq j \leq m$. The *stabiliser code* $\mathcal{C}$ is the *stabiliser subspace* of $\mathcal{S}$,

$$\mathrm{Stab}(\mathcal{S}) = \{|\psi\rangle \in \mathbb{C}^{2^n};\, M|\psi\rangle = |\psi\rangle, \forall M \in \mathcal{S}\}.$$

$\mathcal{C}$ is characterised by a tuple $[\![n,k,d]\!]$, where n is the number of *physical qubits*, k is the number of *logical qubits*, and d denotes the *code distance*.

Definition 2.16. $\overline{U}$ denotes an operator U acting on logical qubits. When $k > 0$, $\overline{X}_j, \overline{Z}_j \in \mathcal{P}_n$ denote the X and Z operators acting on the j-th logical qubit, for $1 \leq j \leq k$.

[29, 45] established the fundamental theorem of stabiliser theory (FTST) that $\mathrm{Stab}(\mathcal{S})$ is of dimension 2^k, $k = n - m$. A corollary of this states that a maximal stabiliser group on n qubits (in other words, $k = 0$) corresponds uniquely to an n-qubit *stabiliser state*.

Next, we introduce the tableau representation of a stabiliser state and call it an *encoder tableau*

Definition 2.17. Each n-qubit stabiliser state $|\psi\rangle$ admits a representation by an $n \times (2n+1)$ matrix T_{enc} over $\mathbb{Z}_2$, called the encoder tableau of $|\psi\rangle$. Its jth row, for $1 \leq j \leq n$, encodes a stabiliser generator S_j: the first n columns record the X components, the next n columns record the Z components, and the final column records the phase, where 0 denotes $+1$ and 1 denotes -1. For each qubit ℓ, the entries in columns ℓ and $n+\ell$ together determine whether S_j acts as I, X, Z, or Y on that qubit.

Note that the tableau representation of $|\psi\rangle$ is not unique, because the choice of generators for a maximal stabiliser group is not unique. By performing a sequence of elementary row operations, we can always reduce the tableau to its *reduced row echelon form (RREF)*, which is unique. We refer readers to consult [49] for more details on the RREF.

[1] generalises Definition 2.17 to *stabiliser mixed states*, i.e., uniform distributions over all states jointly stabilised by operators in $\mathcal{S}$. When there are fewer stabiliser generators than the number of qubits (i.e., $m < n$), the stabiliser tableau does not completely determine the state. By FTST, the code space given by $\mathcal{S}$ is of dimension 2^k, $k > 0$.

Next, we define a *complete stabiliser tableau* for any $k > 1$ stabiliser code $\mathcal{C}$. In addition to the m independent stabiliser generators, the tableau encodes $2k$ independent logical operators $\overline{X}_j$ and $\overline{Z}_j$, $1 \leq j \leq k$.

Definition 2.18. Each $[\![n,k,d]\!]$ stabiliser code $\mathcal{C}$ can be represented by an $(n+k) \times (2n+1)$ matrix T_{stab} over $\mathbb{Z}_2$, called a *complete stabiliser tableau* of $\mathcal{C}$. For $1 \leq j \leq m$, row j encodes the stabiliser generator S_j; for $m+1 \leq j \leq m+k$, row j encodes the logical operator $\overline{X}_{j-m}$; and for $m+k+1 \leq j \leq m+2k$, row j encodes the logical operator $\overline{Z}_{j-m-k}$. As in Definition 2.17, the first $2n$ columns record the Pauli support, and the final column records the phase, with 0 denoting $+1$ and 1 denoting -1.

Finally, we review the Calderbank-Shor-Steane (CSS) codes [7, 52, 53]. They are special types of stabiliser codes whose generators can be split into Pauli strings over either $\{X, I\}$ or $\{Z, I\}$. We call the former *X-type stabiliser generators*, and the latter *Z-type stabiliser generators*. Accordingly, Figure 10b depicts the block-diagonal form of the stabiliser tableau of a $k = 0$ CSS code. The X-type stabiliser generators are encoded in the top left corner and the Z-type stabiliser generators are encoded in the bottom right corner.

2.4 The Normal Forms of Phase-Free ZX-diagrams

Phase-free ZX-diagrams correspond to linear maps of vectors over $\mathbb{Z}_2$. As visualised in Figure 11, [37] establishes the correspondence between a CSS code encoder E and a normalised phase-free ZX-diagram. This correspondence will become useful for establishing the correspondence between any stabiliser codes and the normal forms for Clifford ZX-diagrams.

We begin by reviewing several normal forms of phase-free ZX-diagrams. These normal forms are not unique, but they have rich structures that are suitable for diagrammatic simplification and unification.

Definition 2.19. A ZX-diagram is *two-coloured* when

1. no spiders of the same colour are connected to each other,

$T =$

	$X_1 \cdots X_n$	$Z_1 \cdots Z_n$	p
S_1 $\vdots$ S_m	$\ddots$	$\ddots$	$\vdots$
$\overline{X}_1$ $\vdots$ $\overline{X}_k$	$\ddots$	$\ddots$	$\vdots$
$\overline{Z}_1$ $\vdots$ $\overline{Z}_k$	$\ddots$	$\ddots$	$\vdots$

(a) The encoder tableau of an $[\![n,k,d]\!]$ stabiliser code.

$T =$

	$X_1 \cdots X_n$	$Z_1 \cdots Z_n$	p
S_1^X $\vdots$ $S_{m_X}^X$	$\ddots$	$\mathbf{0}$	$\vdots$
S_1^Z $\vdots$ $S_{m_Z}^Z$	$\mathbf{0}$	$\ddots$	$\vdots$
L_X $\vdots$ L_Z	$\ddots$ $\mathbf{0}$	$\mathbf{0}$ $\ddots$	$\vdots$

(b) The encoder tableau of an $[\![n,k,d]\!]$ CSS code.

Figure 10: Compare the encoder tableau of a general stabiliser code and a CSS code. The latter has a block diagonal structure. For $1 \leq j \leq m_X$ and $1 \leq \ell \leq m_Z$, S_j^X and S_ℓ^Z denote the X- and Z-type stabiliser generators respectively. $m_X, m_Z \in \mathbb{N}$ and $m_X + m_Z = n$. L_X and L_Z denotes the collection of logical X and Z operators.

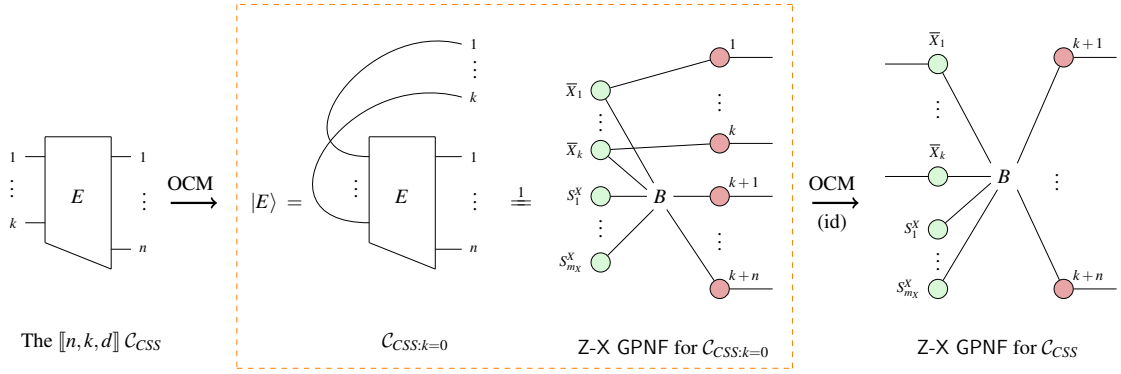
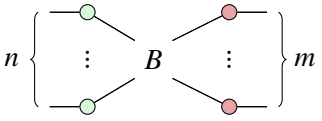


Figure 11: $\mathcal{C}_{\text{CSS}}$ denotes an $[\![n,k,d]\!]$ CSS code. Let $m_X, m_Z \in \mathbb{N}$ and $m_X + m_Z + k = n$. $\mathcal{S} = \langle S_1^X, \ldots, S_{m_X}^X, S_1^Z, \ldots, S_{m_Z}^Z \rangle$. $L = \langle \overline{X}_1, \ldots, \overline{X}_k, \overline{Z}_1, \ldots, \overline{Z}_k \rangle$. Specifically, $\mathcal{S}_X = \langle S_1^X, \ldots, S_{m_X}^X \rangle$ and $L_X = \langle \overline{X}_1, \ldots, \overline{X}_k \rangle$. For the $k = 0$ CSS code $\mathcal{C}_{\text{CSS:k=0}}$, $S_{MAX} = S \cup L$. B is the biadjacency matrix describing the connectivity between the input and output spiders.

2. there are no self-loops,
3. there is at most one wire between each pair of spiders,
4. there are no Hadamards.

Definition 2.20. A phase-free ZX-diagram is in *parity normal form* (PNF) when

- there is a one-to-one correspondence between an input and a Z spider.
- there is a one-to-one correspondence between an output and an X spider.
- spiders are only connected to spiders of the opposite colour and via at most one wire.



(11)

Definition 2.21. A spider is an *input spider* (*output spider*) when it is connected to at least one input (output). A spider is an *internal spider* when it is neither an input spider nor an output spider.

Note that there is no internal spider in PNF. The connectivity between the n input and m output spiders is described by an $m \times n$ matrix B over $\mathbb{Z}_2$. As shown in Example 2.1, one can verify that (11) indeed acts as a linear map described by B, by inputing computational basis states into the (11) and applying the (sc) rule from Figure 5 to simplify the diagram.

Example 2.1. Recall that for $a \in \mathbb{Z}_2$, $|a\rangle \propto$ 

. Figure 12.(a) shows a linear map described by a biadjacency matrix B. For $1 \leq j \leq 3$ and $a_j \in \mathbb{Z}_2$, its action on the computational basis states is below.

$$B = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad B \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} a_1 \oplus a_3 \\ a_1 \oplus a_2 \\ a_1 \oplus a_2 \\ a_3 \end{bmatrix}.$$

Figure 12.(b). shows that this agrees with the result obtained from the diagrammatic rewrites.

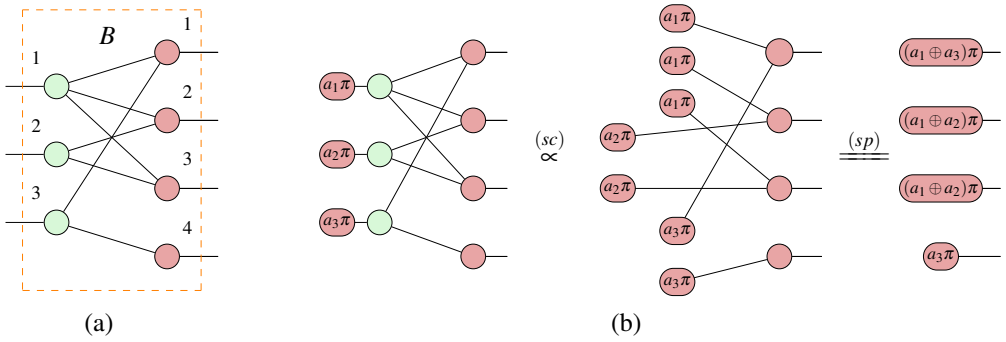


Figure 12: A phase-free ZX-diagram corresponds to a linear map of vectors over $\mathbb{Z}_2$.

Corollary 2.2. A PNF corresponds exactly to a CNOT circuit when its biadjacency matrix is invertible.

In Section B, we show how to efficiently convert between any phase-free ZX-diagram in PNF and its corresponding CNOT circuit by Gaussian elimination.

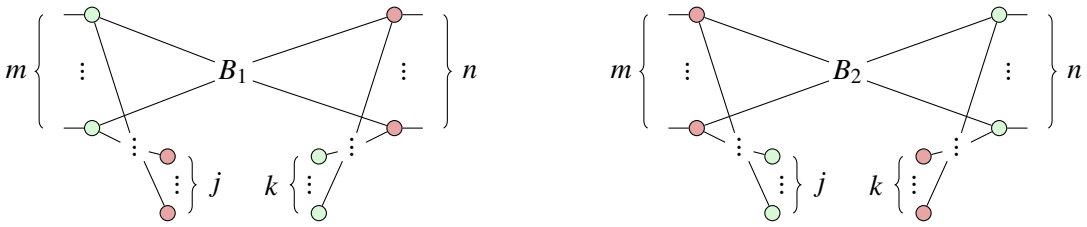
Definition 2.22 generalises Definition 2.20 to include internal Z and X spiders.

Definition 2.22. A phase-free ZX-diagram is in *Z-X generalised parity normal form (Z-X GPNF)* when it is two-coloured and

1. every input is connected to a Z spider and every output to an X spider,
2. every spider is connected to at most 1 input or output,
3. there are no zero-arity (i.e. scalar) spiders, and
4. no internal spiders are connected directly to each other.

The same phase-free ZX-diagram can be written into X-Z GPNF by exchanging the roles of Z and X spiders above. Figure 13 visualises these two normal forms.

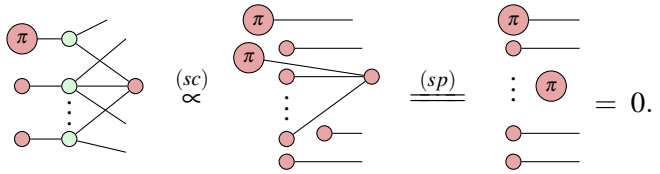
Lemma 2.1. If Z-X GPNF (or X-Z GPNF) is an isometry, $j = 0$.



(a) Z-X GPNF (b) X-Z GPNF

Figure 13: B_1 and B_2 are biadjacency matrices describing the connectivity between the input and output spiders.

Proof. Suppose towards contradiction that $j \neq 0$. That is, there is an X-spider that is not connected to any output. By assumption we don't have any floating scalar spiders, so it must be connected to at least some Z-spiders. Since the diagram is in Z-X GPNF, these must then all be input Z-spiders. Input a $|1\rangle$ to one of the input Z-spiders that the X-spider is connected to and $|0\rangle$ to all the others. After a sequence of diagrammatic rewrites, there is a π phase on the X-spider:



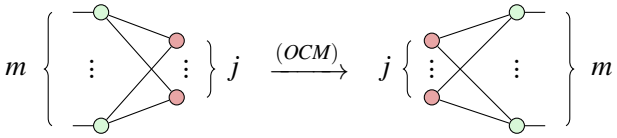
Hence, we have some input state that is mapped to zero, which contradicts the assumption that this Z-X GPNF is an isometry. Hence, all X-spiders must be connected to an output, and by Definition 2.22, to a unique one. □

Remark 2.1. Lemma 2.1 states a necessary but insufficient condition for a Z-X GPNF (X-Z GPNF) to be isometry. Examples when a Z-X GPNF (X-Z GPNF) with $j = 0$ is not isometry include when every input is connected to zero output, or when the number of inputs exceeds the number of outputs.

Corollary 2.3. If Z-X GPNF (or X-Z GPNF) is unitary, the biadjacency matrix B is invertible and $j = k = 0$.

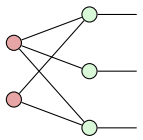
Finally, we discuss a special case of GPNF when there is no input spider. In [39], it was shown that if a quantum state can be written as a superposition of uniformly distributed computational basis states, it can be equivalently represented, up to a global phase, by a Z-X GPNF and an X-Z GPNF. Without loss of generality, below we consider this state in X-Z GPNF and we give it a name.

Definition 2.23. The *X-Z normal form* of a phase-free ZX-diagram consists of a row of internal X spiders connected to a row of Z spiders. Each Z spider is connected to precisely 1 input. It can be described as a set of vectors $\{w_1, \ldots, w_j\}$ over $\mathbb{F}_2$, where $(w_p)_q = 1$ if and only if the p-th internal X spider is connected to the q-th Z spider.



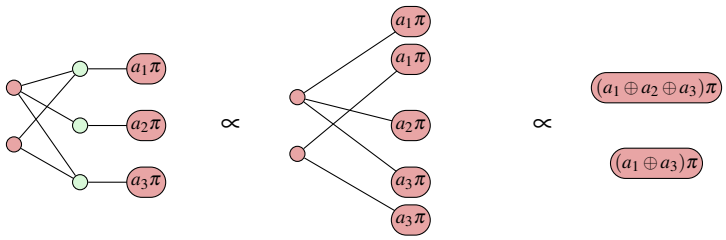
The X-Z normal form uniquely encodes a state defined by a system of linear equation. In Sections 3 and 4, the generalised notion of this normal form will be used to define a normal form for an arbitrary stabiliser code.

Example 2.2. S is defined by a *homogeneous system of linear equations*, meaning the RHS of every equation is 0. Equivalently, it is the set of bit vectors $\vec{a}$ satisfying $M\vec{a} = 0$ for some $\mathbb{Z}_2$-matrix M.



$$\propto \sum_{\vec{a}\in S} |\vec{a}\rangle \qquad \text{where} \qquad S = \left\{ \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \;\middle|\; \begin{aligned} a_1 \oplus a_2 \oplus a_3 &= 0 \\ a_1 \oplus a_3 &= 0 \end{aligned} \right\}$$

Similar to Example 2.1, we can verify this by post-selecting on the computation basis states $\langle a_1, a_2, a_3|$, for $1 \leq j \leq 3$ and $a_j \in \mathbb{Z}_2$.



2.5 The Normal Forms of Clifford ZX-diagrams

Let $k \in \mathbb{Z}_4$. Clifford ZX-diagrams are ZX-diagrams whose spiders have phases of the form $k\frac{\pi}{2}$. When these diagrams are unitary, they correspond to Clifford circuits, which are indispensable for qunatum error correction, gate teleportation, and quantum algorithms [29, 45]. By Gottesman-Knill theorem, Clifford circuits can be efficiently simulated on a classical computer [30]. Thanks to [44, 23, 3, 19, 21, 13, 22, 5, 39, 55], Clifford ZX-diagrams can be efficiently simplified by performing a sequence of diagrammatic rewrites. The resulting diagrams have well-described compact structures, so they are referred to as the *Clifford normal forms*. In what follows, we will introduce two normal forms, both of which are suitable for visualising Clifford encoders.

Definition 2.24. A ZX-diagram is *graph-like* when

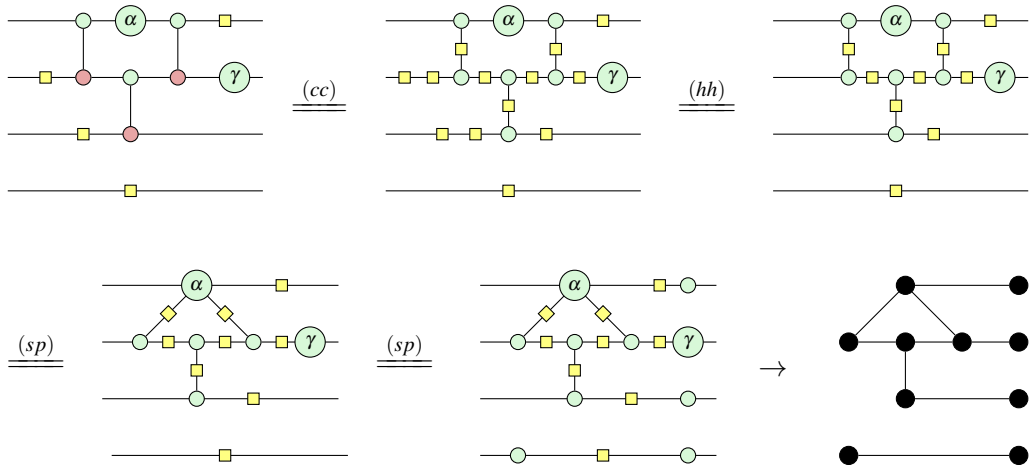
- every spider is a Z-spider.
- spiders are only connected via Hadamard edges.
- there are no self-loops or parallel edges.
- every Z-spider is connected to at most one input and at most one output.
- every input and output wire is connected to a Z-spider.

These diagrams are called graph-like because they can be described by an *open graph* [18], a graph with a specified list of inputs and outputs.

Proposition 2.4. Every ZX-diagram can be efficiently reduced to an equivalent graph-like diagram.

Example 2.3. Here we demonstrate how to transform a ZX-diagram into a graph-like diagram. In the

last step we write down its underlying graph.

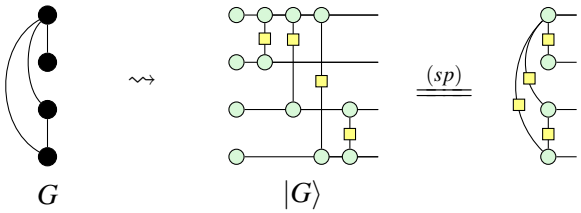


Definition 2.25. Consider a simple undirected graph $G = (V,E)$, V denotes the set of vertices and E denotes the set of edges. Then the graph state $|G\rangle$ is defined as

$$|G\rangle = \prod_{(v,w)\in E} \mathrm{CZ}_{v,w}|+\rangle^{\otimes|V|}.$$

Example 2.4. To go from a graph G to a graph state $|G\rangle$ represented in the ZX-diagram,

- each vertex becomes a phase-free Z-spider with an output attached to it.
- each edge becomes an Hadamard edge between the corresponding spiders.



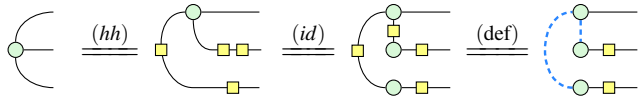
Definition 2.26. A graph-like diagram is a *graph state* when

- it has no inputs.
- every spider is connected to an output.
- all phases are zero.

Definition 2.27. A *local* unitary is single-qubit unitary acting on some output of a graph state.

Example 2.5. The GHZ-state is not a graph state, but we can view it as applying local H gates on a graph state, as shown below.

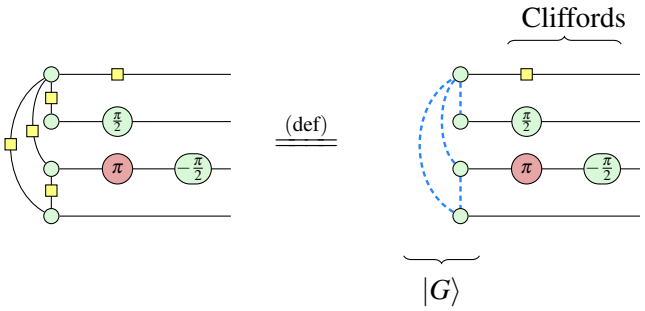
$$|GHZ\rangle = \tfrac{1}{\sqrt{2}}(|000\rangle + |111\rangle) \propto$$



2.5.1 The GSLC Normal Form

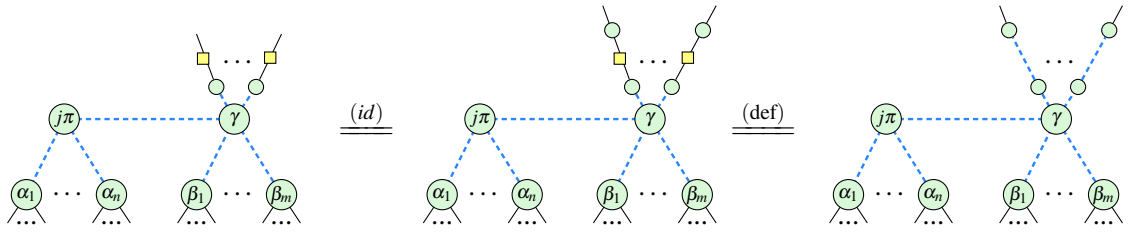
Definition 2.28. A *graph state with local Cliffords* (GSLC) is a graph state with some local Clifford unitaries applied to it.

Example 2.6. Every single-qubit Clifford unitary can be written as a composition of three phase gates where the phases are multiples of $\frac{\pi}{2}$ [39]. Here is an example of a GSLC state. It is a composition of the graph state of Example 2.4 with any $Z(\frac{\pi}{2})$ or $X(\frac{\pi}{2})$ phase gates.



Definition 2.29. We say a diagram is *graph-like with Hadamards* when it satisfies all the conditions for being graph-like (Definition 2.24), except that inputs and outputs are also allowed to be connected to Z-spiders via a Hadamard gate.

Example 2.7. A graph-like diagram with Hadamards, for instance (a), can be easily transformed into a graph-like diagram (c) by introducing some additional Z-spiders using the (id) rule.



Definition 2.30. We say a Clifford diagram is in *GSLC form* when it is graph-like with Hadamards and has no internal spiders.

As in Definition 2.28, GSLC here stands for *graph state with local Cliffords*. In fact, when a GSLC form has not input, it is a GSLC.

Proposition 2.5. Every Clifford ZX-diagram can be efficiently rewritten to an equivalent diagram in GSLC form.

The proofs of Theorems 2.4 and 2.5 are omitted here. For more details, we encourage readers to consult Chapter 5 of [39].

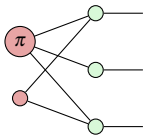
2.5.2 The Affine with Phases Normal Form

The *Affine with Phases Normal Form* (AP) normal form is based on the stabiliser state normal form first introduced by Dehaene and De Moor in [17] which represents a state as an equal weight superposition

over an affine subspace of computational basis states. Its utility and representation in the ZX-calculus was independently discovered by Backens and McElvanney [43] and Kissinger and van de Wetering [39]. Its adaptation to the qudit setting was used to give a completeness proof of stabiliser ZX-calculus over the odd-prime-dimensional qudits [46]. The algorithm for rewriting to AP normal form is detailed in Section B. Figure 14 visualizes AP normal form diagrams consistent with the notation in this text.

We first generalise Definition 2.23 by additionally allowing the X spiders to carry 0 or π phases. This gives us almost the same thing, except now S can be defined by an *inhomogeneous system of linear equations*. Each X spider with a 0 phase gives us an equation with a 0 on the RHS, whereas an X spider with a π phase gives us an equation with a 1 on the RHS.

Example 2.8.



$$\propto \sum_{\vec{a}\in S} |\vec{a}\rangle \qquad \text{where} \qquad S' = \left\{ \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \;\middle|\; \begin{aligned} a_1 \oplus a_2 \oplus a_3 &= 1 \\ a_1 \oplus a_3 &= 0 \end{aligned} \right\}$$

That is, we get the set of vectors $\vec{a}$ satisfying $M\vec{a} = \vec{b}$ for some fixed matrix M and vector $\vec{b}$. In the example above, its:

$$M = \begin{pmatrix} 1\,1\,1 \\ 1\,0\,1 \end{pmatrix} \qquad \vec{b} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

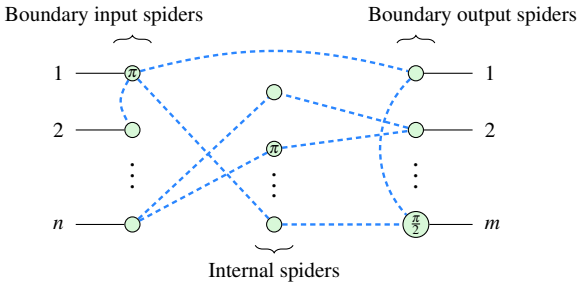
Remark 2.2. The set of all solutions to a system of homogeneous linear equations forms a *linear subspace* of $\mathbb{Z}_2^n$. The solutions to an inhomogeneous system, in general, forms an *affine subspace*, as given in Definition 2.11. Intuitively, an affine subspace is like a linear subspace that has been shifted away from the origin by some fixed amount.

Now we can define the other normal form of the Clifford ZX-diagrams.

Definition 2.31. A graph-like Clifford diagram is in *affine with phases form* (AP form) when:

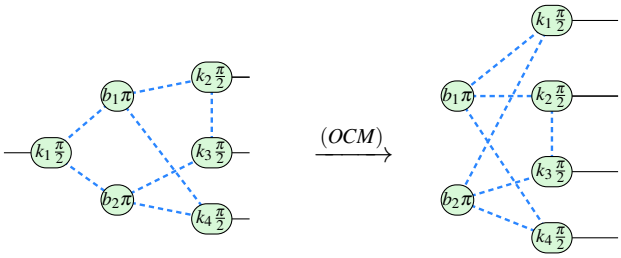
1. every boundary spider is connected to exactly one input or output.
2. every internal spider has a phase of 0 or π.
3. no two internal spiders are connected to each other.

Example 2.9. Here we show a simple example for the AP form. Let n, m be positive integers. There are $n+m$ boundary spiders: n of them are input spiders, and the remaining are output spiders. Note that there is no connectivity between internal spiders. There could be connectivity between any boundary spiders.



By Definition 2.31, AP form treats inputs and outputs equally. By OCM, an arbitrary linear map in AP form can be treated as a state by 'bending wires' to turn inputs into outputs. Thus, from this point onwards, we focus on the states in AP form.

Example 2.10. For $1 \leq j \leq 2$, $b_j \in \{0,1\}$. For $1 \leq \ell \leq 4$, $k_\ell \in \{0,1,2,3\}$. The linear map on LHS can can be treated as a state on RHS.



Proposition 2.6. Every Clifford ZX-diagram can be efficiently rewritten into an equivalent AP form.

Definition 2.32. A non-zero diagram in AP form defined by the triple $A, \overrightarrow{b}, \phi$ is in *reduced AP-form* when:

1. A is in reduced row echelon form (RREF) with no zero rows,
2. ϕ only contains free variables from the system $A\overrightarrow{x} = \overrightarrow{b}$.
3. all the coefficients of ϕ are in the interval $(-1,1]$.

Recall that the first non-zero element in a row of a matrix A in RREF is called a **pivot** (no relation to the pivot graph rewrite rule). The variable x_i is called a *free variable* if the ith column of A does *not* contain a pivot, otherwise it is called a *bound variable*. The utility of looking at the reduced AP-form is the following theorem.

Theorem 2.7. *For any non-zero state $|\psi\rangle$, there is at most one triple $(A, \overrightarrow{b}, \phi)$ satisfying the conditions of Definition 2.32 such that:*

$$|\psi\rangle \propto \sum_{\overrightarrow{x}, A\overrightarrow{x} = \overrightarrow{b}} e^{i\pi\cdot\phi} |\overrightarrow{x}\rangle$$

The proofs of Theorems 2.6 and 2.7 are omitted here. For more details, we encourage readers to consult Chapter 5 of [39].

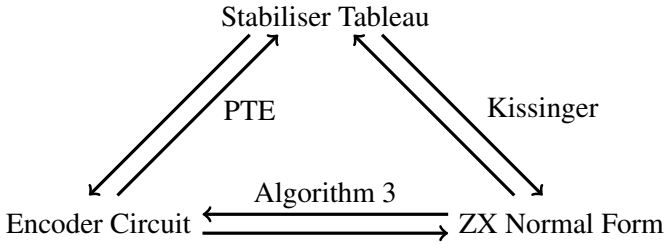
3 ZX Normal Forms for CSS Codes and Graph Codes

In this section, we review ZX normal forms for two important classes of stabiliser codes: CSS codes and graph codes. Any stabiliser code is equivalent to a CSS code up to a diagonal Clifford unitary and some X gates [57]. Any stabiliser code is also equivalent to a graph code up to local (i.e. single-qubit) Clifford unitaries [48]. How are these related? In this paper, we will show that there is more to the story.

3.1 CSS Codes

In [37], Kissinger drew the equivalence between any $k = 0$ CSS code encoder and phase-free ZX diagrams. For CSS codes with $k > 0$, he described how when the encoder is Clifford, its Choi state is a stabiliser state, exampified by lattice surgery of surface code patches.

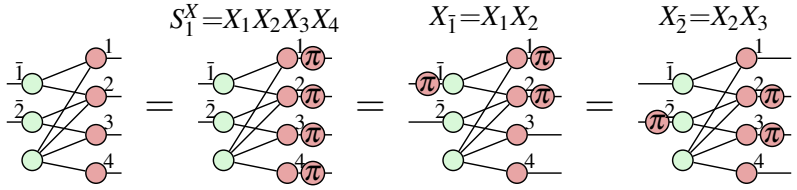
In this work, our first result is to formally spell out the connection between all CSS codes and phase-free ZX-diagrams.



We start by explaining for any CSS code, how to interconvert between stabiliser tableau and phase-free ZX-calculus representations through the Generalised Parity Form (GPF). By Algorithm 5, we proved that this form, utilised for a number of CSS codes in our prior works [34, 37], is in fact a normal form that applies to all CSS codes.

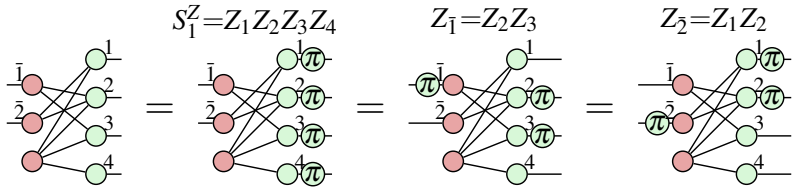
Example 3.1. As a minimalist example of a self-dual CSS code, we show how GPNF diagrams are in one-to-one correspondence with a complete stabiliser tableau for the $[\![4,2,2]\!]$ code.

We are able to read off one X-type logical operator for each qubit, and X-type stabiliser generators spanning the code space, directly from the connectivity of an encoder for the $[\![4,2,2]\!]$ code in Z-X GPNF.



(12)

With the same encoder in X-Z GPNF, we can read off one Z-type logical operator per qubit, and a spanning set of Z-type stabiliser generators.



(13)

In the other direction, from the X-type half or the Z-type half of the rows of this complete stabiliser tableau, we can reconstruct the corresponding Z-X GPNF and X-Z GPNF diagram respectively.

Ensuing this, we remark that while the more general case of any graph code is suited to Graph State with Local Cliffords (GSLC) normal forms, this is identifiable with a complete stabiliser tableau, but not with the usual (i.e. incomplete) stabiliser tableau.

To bridge this gap, we adapt the reduced Affine with Phases (AP) normal form to enable reading off both complete and incomplete stabiliser tableaus for the most general setting of any stabiliser code. In turn, given any stabiliser tableau, we give an entirely graphical procedure to produce a valid encoder diagram. We can further conceptualize our adapted AP normal forms as a generalisation of both the GPF and GSLC forms. This thereby sets up a unified framework to extend graphical results for CSS and graph codes to all stabiliser codes, explored in the next section.

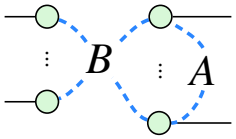
3.2 Graph Codes

As a prelude to the case of all stabiliser codes, we briefly discuss another well-studied class of stabiliser codes: graph codes [47]. A graph code is defined by the simple graph indicating the graph state that

spans the code space. Any stabiliser code is equivalent to a graph code, up to local (a.k.a., single-qubit) Clifford gates.

In the ZX-calculus, any stabiliser state can be reduced to Graph State with Local Cliffords (GSLC) [4] gave an efficient algorithm to convert any stabiliser state to GSLC form together with a way to compare graph states representing identical states, resulting in the original proof of completeness of the qubit ZX-calculus for stabiliser quantum mechanics. A more practical algorithm for reducing an arbitrary stabiliser ZX-diagram, including those with inputs, to GSLC form was given in [5].

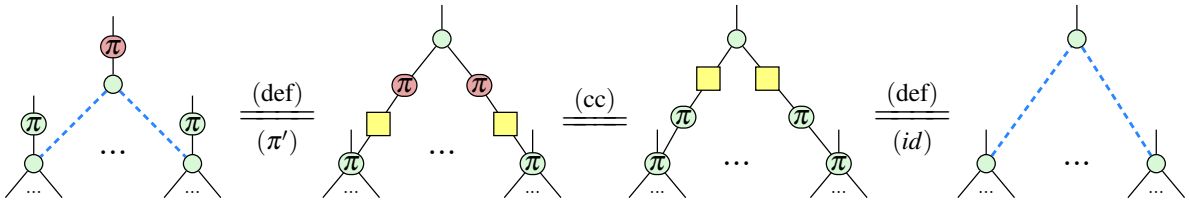
Up to local Cliffords on the output qubits[1], encoders of graph codes are always expressible in the form



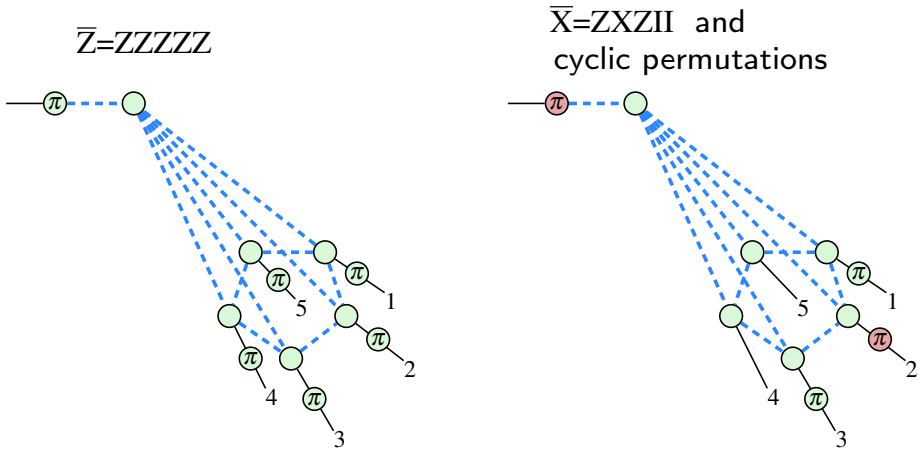
(14)

where B is a biadjacency matrix and A is an adjacency matrix.

The GSLC form has a desirable property: It is possible to directly read off enough linearly independent stabilisers from the graph state, to fix the diagram. This is due to the local complementation of graph states(Example 6.3.8 in [39]), resulting in a well-known property where one stabiliser can be written down for any vertex of the graph: X on that vertex, and Z on all its neighbours. Next to this, commuting through the local Cliffords is straightforward. In the ZX-calculus, the proof of this property on a subgraph consisting of a vertex and all its neighbors is as follows:



A toy example through which to illustrate this is the 5-qubit code [41, 36]. From examining its encoder in GSLC form, we can read off six linearly independent Pauli strings: one that is a logical Z operator, and five that are logical X operators.



(15)

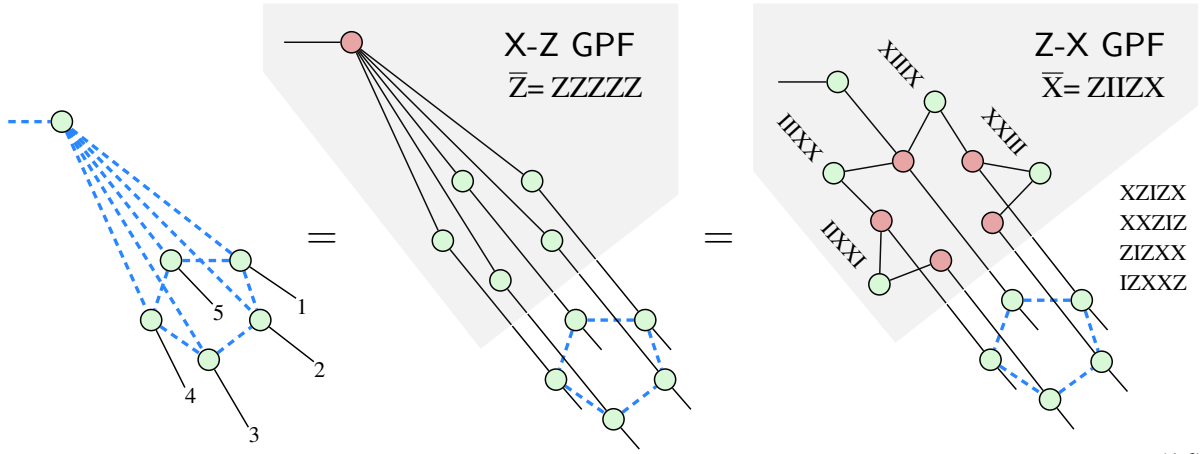
[1]Local Cliffords on the input qubits can always be disregarded for either of two reasons. One, any local Cliffords on the inputs can be relegated to the outputs, by operations on the graph such as local complementations. Two, stabiliser codes are defined by stabiliser groups; all encoder maps henceforth are equivalent up to unitaries on the logical qubits.

In other words, it is not the case that four ($= n - k = 5 - 1$) of these are stabiliser generators, and the other two fix logical X and Z operators respectively. To derive a stabiliser tableau from these Pauli strings, more steps are necessary in order to isolate stabiliser generators from any dependence on the logical qubits.

This corresponds to rewrites which bring us outside the simple graph formalism, but are still well captured in the ZX-calculus. The following example of the 5-qubit code is instructive for demonstrating the steps involved, before we proceed to the general case of all stabiliser codes in the next section.

Example 3.2. At the start of (16), we have the GSLC form of an encoder for the 5-qubit code, a graph code. By unfusion, this splits into an X-Z GPNF diagram identified with the biadjacency matrix B, followed by a CZ circuit identified with the adjacency matrix A. The connectivity of the X-Z GPNF diagram directly allows reading out the logical Z operator $\overline{Z} = ZZZZZ$. Note that Z-type logicals and stabilisers (i.e. Pauli strings over $\{Z, I\}$) always commute with CZ gates.

Next, we make use of the phase-free GPNF algorithm from the previous section to rewrite that X-Z GPNF diagram to Z-X GPNF. Reading off from it a logical X operator and four linearly independent X-type stabilisers, we can then commute these Paulis through the CZ circuit to obtain the rest of the complete stabiliser tableau.



(16)

In brief, although we cannot read off stabiliser generators directly from any graph code in GSLC form, we showed an example where we rewrote it to a form where we can. In the next section, we show this for any stabiliser code, how to ensure there are enough linearly independent rows for the complete stabiliser tableau.

4 Stabiliser ZX-calculus Isometry AP Normal Form

Up to now, we can read off stabiliser tableaus for any CSS code and any graph code. As any stabiliser code is equivalent to a graph code up to local Cliffords, for many intents and purposes, the earlier algorithms will suffice. For instance, in searching for stabiliser codes with a good distance, it is only necessary to search up to the equivalence class defined by each graph code, as local Cliffords do not alter the code distance.

With this said, there are circumstances where tracking the precise local Clifford is important. A notable example is in determining the threshold based on an error model, local Cliffords transform the type of error relative to what the code corrects. For quantum error correcting codes to be effective, they

must account for the dominant sources of physical noise, decoding and correcting based on the likelihood of certain errors occurring. Better yet, codes can be designed for biased noise models, in order to detect certain types of errors more effectively than other, less probable types of errors.

In this section, we introduce a normal form for the encoder of any stabiliser code in the ZX-calculus. It is on-the-dot, meaning that there is no longer the limitation of being up to local Cliffords. Moreover, it enables reading off rows of the stabiliser tableau directly from the diagram. It is derived from the Affine with Phases (AP) normal form, which we will first review. This will allow us to directly read off rows of the stabiliser tableau from it. After that, we use a *reduced* AP normal form to study the case of isometries, in order to capture the complete stabiliser tableau for any stabiliser code on-the-dot.

The AP normal form is based on the stabiliser state normal form first introduced by Dehaene and De Moor in [17] which represents a state as an equal weight superposition over an affine subspace of computational basis states. Its utility and representation in the ZX-calculus was independently discovered by Backens and McElvanney [43] and Kissinger and van de Wetering [39]. Its adaptation to the qudit setting was used to give a completeness proof of stabiliser ZX-calculus over the odd-prime-dimensional qudits [46]. The algorithm for rewriting to AP normal form is detailed in Appendix B. Figure 14 visualizes AP normal form diagrams consistent with the notation in this text.

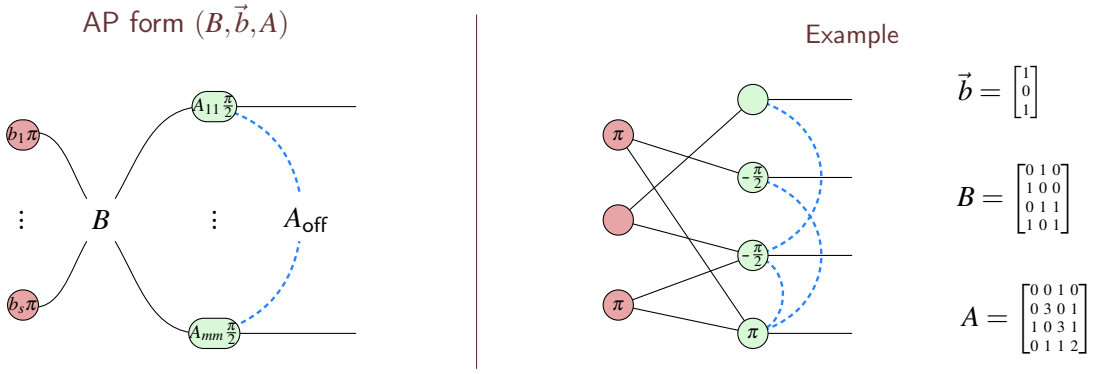


Figure 14: Left: General diagram for a stabiliser state in AP normal form. Right: An example stabiliser state in AP normal form, alongside its parameters $(B,\vec{b},A)$.

From any stabiliser state in AP normal form, we can further refine the diagram to *reduced* AP (rAP) normal form. For reduced AP normal form, the three binary matrices that describe it $(B,\vec{b},A)$ have the property of being *unique* for any stabiliser state.This is accomplished by putting $(B,\vec{b})$ into Reduced Row Echelon Form (RREF), and thereafter reparametrizing A to be dependent only on the free variables, i.e. qubits that are not the pivots of $(B,\vec{b})$. The algorithm for rewriting an AP normal form diagram to reduced AP normal form is given in Appendix B. These two algorithms run in polynomial time of $O(n^3)$, performing the graphical equivalent of Gaussian elimination. Therefore, applying them to any two stabiliser diagrams achieves efficient determination of whether they are equal. If equal, this thus finds a series of rewrites that transform one diagram into the other.

The AP normal form lets us establish a connection between ZX-calculus diagrams and tableaus, for any stabiliser states and hence any $k = 0$ stabiliser code. It is helpful to discuss this case first, before we present this result formally for the general case of isometries ($k > 0$).

4.1 Adapting AP Normal Form to Isometries

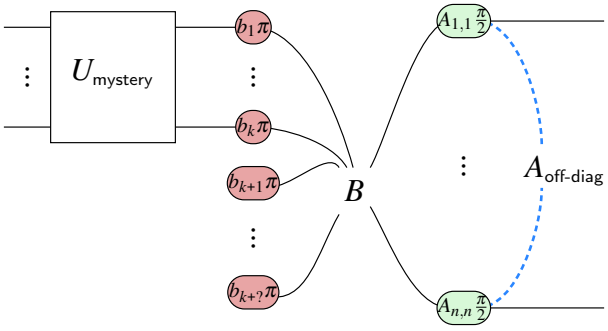
These algorithms were developed for states, but in the context of quantum error correction, we are interested in encoder maps and measurement circuits. By the Choi-Jamiolkowski isomorphism, for any $k > 1$ code, we can convert any encoder map (which is an isometry) to a state. This allows us to subsequently apply the above algorithms for any stabiliser code. At this point, to recover an isometry, what we can do is "un-Choi", i.e. bend the output qubits that were originally inputs back into inputs.

A similar observation was made in Ref. [59] that decomposes stabiliser code encoders into a CSS code followed by Paulis followed by a diagonal Clifford gate.

We extend rAPNF for any stabiliser state, to a normal form for any isometry in the stabiliser fragment of the ZX-calculus, i.e. any Clifford isometry.

Definition 4.1. We will use the notation $\{G_{11}, G_{12}, ...\}; \{G_{21}, G_{22}, ...\}; ...$ to denote layers in circuit decompositions. Here $\{G_{11}, G_{12}, ...\}$ is the leftmost layer, $\{G_{21}, G_{22}, ...\}$ is the second layer, and so on. Each layer consists of one or more gates, which mutually commute. The identity on any qubit(s) is always implicitly allowed in each layer. Each circuit decomposition means that for the class of circuits in question, there always exists a way to decompose into a circuit consisting of only gates which are allowed in the first layer, followed by a circuit of only gates allowed in the second layer, and so on.

Observation 4.1. Before we begin, we know that the decomposition we are looking for has to be of the form

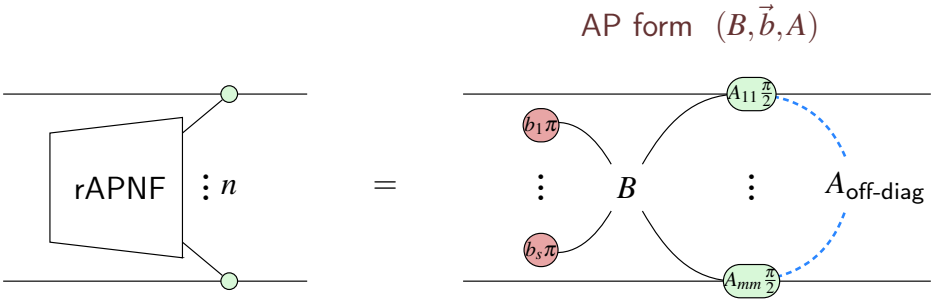


(17)

where we don't know yet what goes in the mystery box, other than that it's a Clifford unitary.

Proof. We know this must be the case as a consequence of three facts.

First, we know that we can always choose the rightmost layer of this decomposition to be a projector to an APNF or rAPNF diagram of the form:



(18)

This is possible because we know that the pure Z-type stabiliser generators must be stabilised by a projector defined by the biadjacency H_Z.

Second, we know from [59] that for any stabiliser code, its codespace its decomposable into three layers: the codespace for some CSS code; {X}; diagonal Clifford unitary. It follows from this that for

any stabiliser code, there exists a choice of encoder that decomposes as: X-Z GPNF; {X}; {S, CZ}. This is because for any CSS code, there exists a choice of encoder that is in X-Z GPNF. Making this choice fixes the logical operators of the encoder of the stabiliser code.

Third, we know that for any code, any two choices of encoders are equivalent up to a unitary on the logical qubits. Since we are comparing an arbitrary Clifford isometry with a specific choice of Clifford isometry for the same code, the unitary that relates them must be Clifford.

We conclude the proof by propagating all X gates to the left to merge with all the other X spiders, after which we can fuse the Z spiders in the X-Z GPNF with those in the diagonal Clifford unitary. □

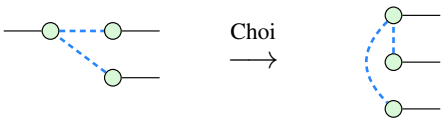
While this confirms the existence of such a normal form, we prove in the remainder of this section how to get to such a normal form. This also deduces what's in the mystery box.

This procedure starts by taking the Choi state of the isometry, then rewriting to rAPNF. We first discuss a few counterexamples to narrow down the cases that we need to consider.

It would be convenient if we had $k+m_z$ total X spiders in the rAPNF, where k is the number of logical qubits and m_z is the number of Z-type stabiliser generators, as we do in the CSS case. Then in the rAPNF algorithm, we could designate one pivot qubit for each X spider: For k of the X spiders we choose input qubits; for the other m_z we choose output qubits. This corresponds to Gaussian elimination of the biadjacency matrix. We would terminate the algorithm by 'un-Choi-ing' the map, i.e. transposing the k qubits to be inputs again.

However, we might not even have k X spiders in the rAPNF, much less $k+m_z$.

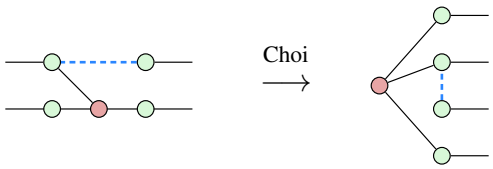
Example 4.1. Here is a toy counterexample where $k=1$, but the number of X spiders in rAPNF is zero.



(19)

A natural question is how much we can modify the original rAPNF algorithm for states. While we can save some time by choosing pivots of X spiders to be inputs (instead of outputs) when possible, we cannot guarantee that every input that is connected to one X spider can be made the pivot of that X spider due to the following.

Example 4.2. If an input is connected to a X spider, it is not necessarily the only input connected to that X spider. An example showing this is a CX gate followed by an H gate:



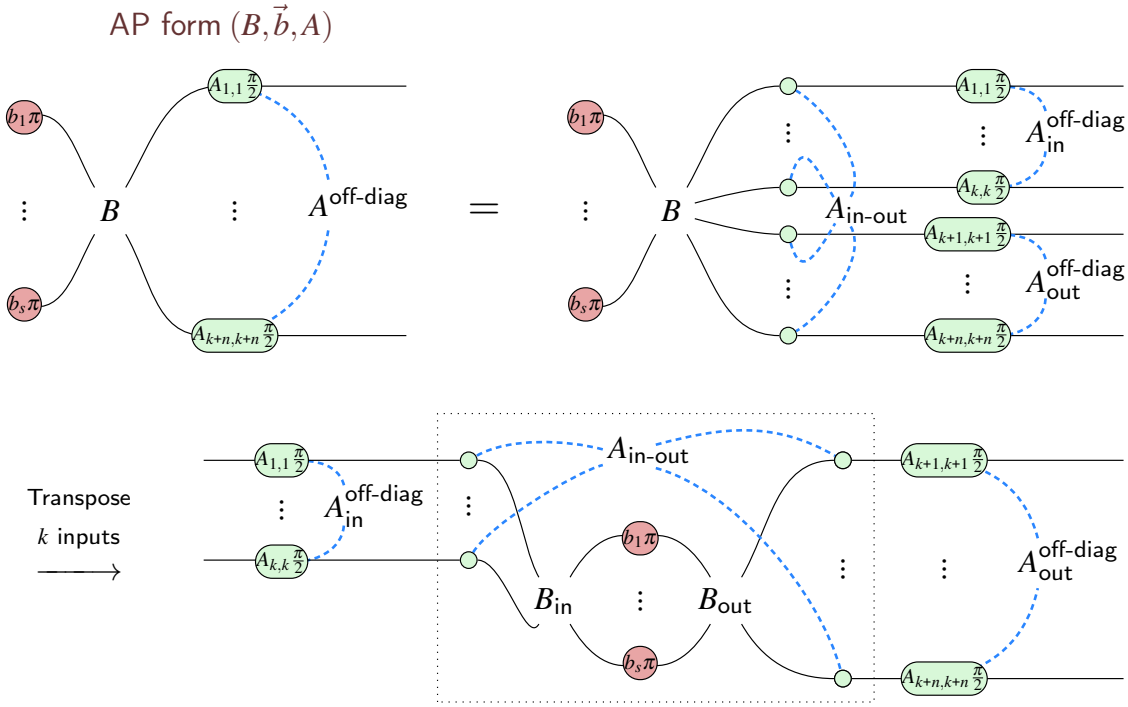
(20)

These examples tell us about what more we need to do to get to the normal form. After returning the rAPNF state back into the original isometry shape, we next unfuse two diagonal Clifford unitaries: all components of A that act only on inputs or only on outputs respectively.

Procedure 1: Choi-Jamiolkowski isomorphism on the encoder state in rAPNF

Input: A stabiliser state in rAPNF and a subset of its qubits to become inputs

Output: An intermediate form between rAPNF and iAPNF



From here, we work with the isometry subdiagram in the dotted box above. In addition to $(B,\vec{b})$, it has the remainder of A, consisting of CZ edges between an input and an output.

Compare our objective, a diagram satisfying the more general form of Observation 4.1, to the result of Algorithm 1 which we have thus far. To achieve the desired transformation, we will iterate through all possible case distinctions of inputs, to make them pivots of X spiders in the biadjacency matrix, hence without CZ edges to the outputs.

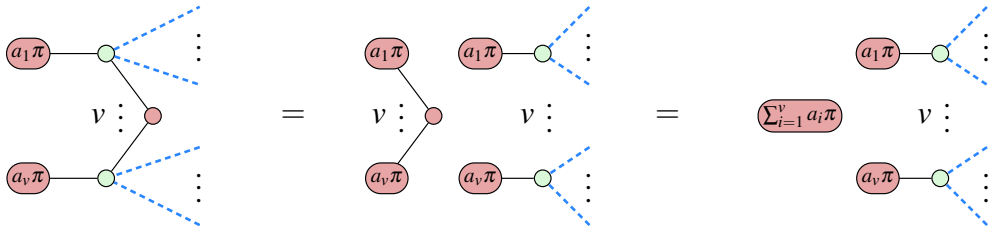
We just need all inputs to be addressed; the order affects the final diagram, but is arbitrary. Since we are starting from rAPNF, a unique normal form, any deterministic algorithm will preserve uniqueness of the final isometry normal form. To fix an order here, we address each input from 1 to k, drawn from top to bottom, according to which of the below cases it satisfies.

Before going through the cases, we introduce a definition and preliminary lemmas that we will require.

Lemma 4.1. Due to all encoders being isometries, it is not possible to have any X spider of the APNF be connected to only input qubits.

Proof. If this were the case, half of all possible input Z-basis states would be annihilated, hence contra-

dicting the isometry condition of the encoder.



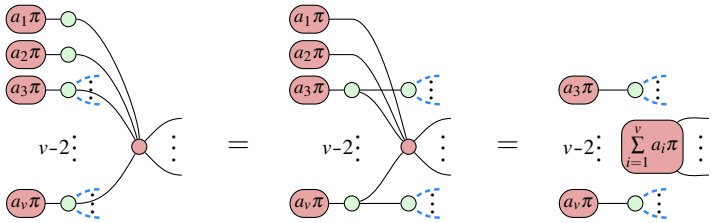
(21)

□

We introduce the next lemma to guarantee that CZ edges must exist in some of the subsequent cases.

Lemma 4.2. Whenever v inputs to an isometry in rAPNF are each connected to the same X spider and no other X spiders, at most one of the v inputs has zero CZ edges to the outputs.

Proof. We do this proof by contradiction. Assume that for v inputs connected to the same X spider and no other spiders, w inputs have zero CZ edges to the outputs for some $w \geq 2$. Consider the below subdiagram of these v inputs, prior to which the encoder is unitary thus far and therefore does not result in any redundant information of the input qubits. Inputting Z-basis states to this subdiagram denoted by v independent bits, we can see that the subdiagram takes this to $v-w+1$ linearly independent binary variables.



(22)

As $w \geq 2$, $v-w+1 < v$. As this makes it impossible to recover the input bits, this contradicts the fact that the encoder is an isometry. □

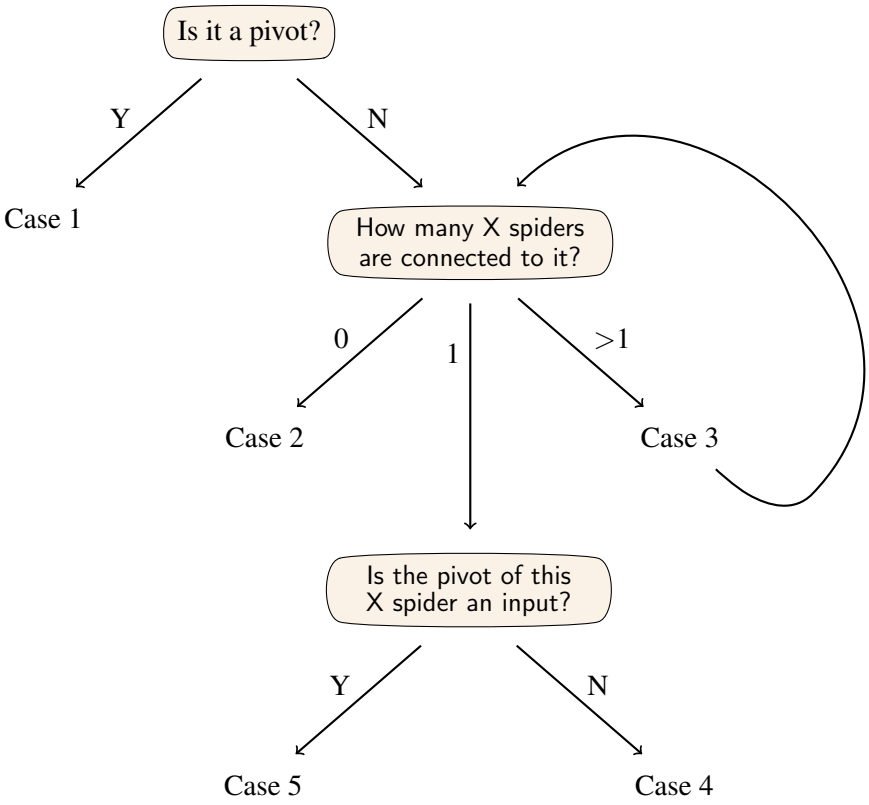
The procedure to derive a ZX diagram for the encoder in iAPNF proceeds with going through the below cases for each input. The control flow is specified in Algorithm 2.

Algorithm 2: Reducing any Clifford isometry to isometry AP normal form

Input: A Clifford isometry in rAPNF that has then undergone Algorithm 1

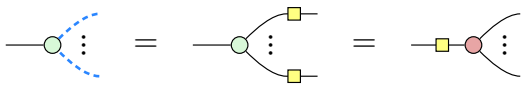
Output: The Clifford isometry in iAPNF

For each input of the dotted subdiagram from Algorithm 1, traverse the flowchart below and resolve the cases accordingly.



Case 1. The simplest case is when there is nothing to do: the input was already the pivot of an X spider in the rAPNF. This implies it is connected to no CZ edges to the outputs and to no other X spiders.

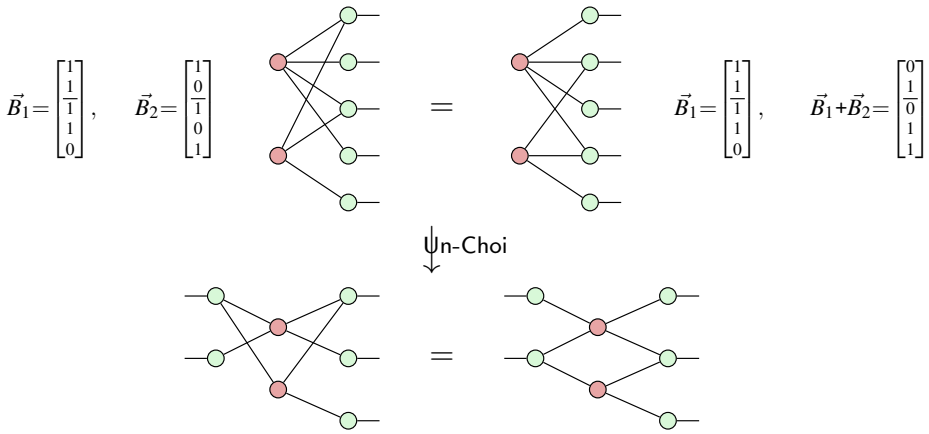
Case 2. The second simplest case is when the input is connected to no X spiders. By Lemma 4.2, there has to be at least one CZ edge to the outputs. Therefore, we can transform the Z spider to an X spider, with the same connectivity as all its CZ edges. This produces an H gate that is added to U_{in}.



(23)

Case 3. When the input is connected to two or more X spiders, we can add the connectivity of one (can be any; say, the topmost one) to all of the others [37]. This corresponds precisely to row addition in the

biadjacency matrix B. An example is depicted below.



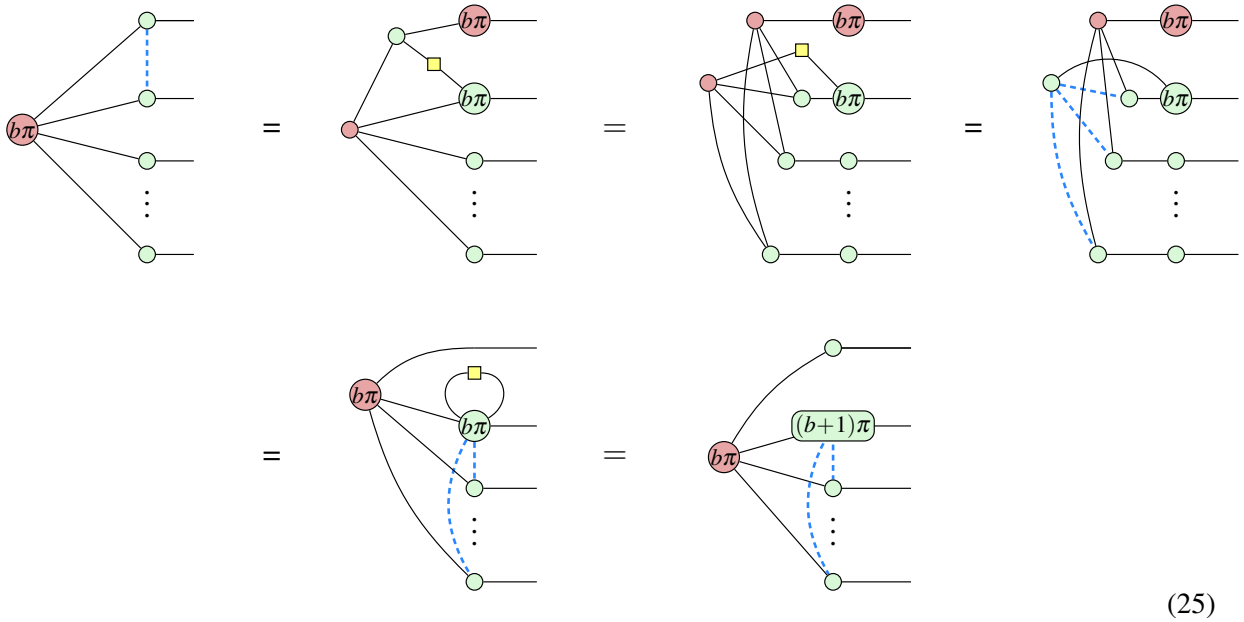
(24)

As a result, the input is connected to only one X spider, and will match one of the subsequent cases. The input is not necessarily the pivot of this X spider yet, because it could be connected by CZ edges to outputs—these will be removed by Case 4.

Remark 4.1. As for the other X spiders this input was connected, they are no longer connected to this input. Each of their existing pivots are preserved by this row addition in the biadjacency matrix B, as by definition of being a pivot it is guaranteed to be the only 1 in its column. This is exemplified in Equation (24): Substituting the bottom X spider (with connectivity $\vec{B}_2$) with its XOR with the top X spider (with connectivity $\vec{B}_1$), preserves the bottommost output as its pivot.

The remaining cases are of the input being connected to exactly one X spider.

Case 4. If that X spider does not already have another input as a pivot, make this input its pivot. The procedure for making any of its qubits its pivot is from the rAPNF algorithm for states. As we started by unfusing and subsequently disregarding any integer multiples of $\frac{\pi}{2}$ phases on this input as S gates, and any CZ edges to other inputs as CZ gates, we only need to handle the case of CZ edges from this input to any outputs. We achieve this with the following rewrite from [39]:

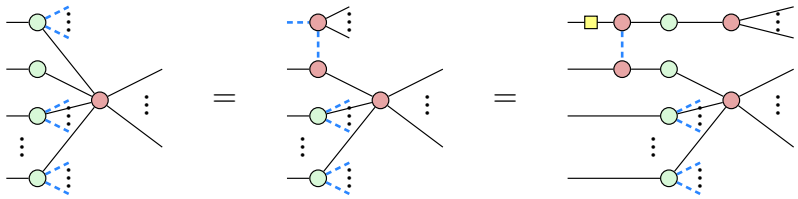


(25)

Remark 4.2. Any creation of CZ edges on other inputs or outputs of this X spider during Case 4, preserves all pivots of other X spiders. This is because they can only be connected to the X spider they are of the pivot of, and thus not connected to this X spider.

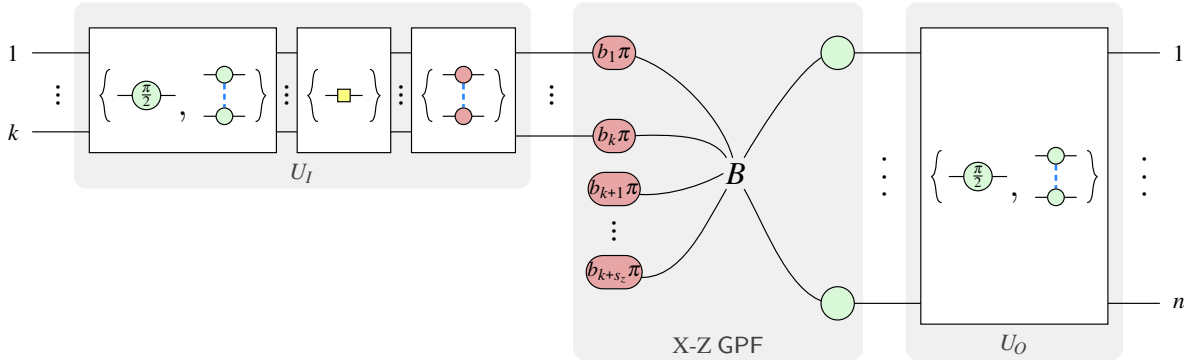
For subsequent cases, the input is connected to exactly one X spider, which already has a pivot which is another input. We have from Lemma 4.1 that the X spider must be connected to at least one output, and from Lemma 4.2 that this input must have at least one CZ gate to the outputs.

Case 5. Since the X spider already has a pivot which is another input, we can unfuse a CX gate with the control being the input in question and the target being the pivot. We can then transform the Z spider to an X spider, adding H; DX to U_{in}.



(26)

Theorem 4.1 (ZX-calculus Isometry AP Normal Form)**.** *Every Clifford isometry can be written to a* ZX-calculus normal form *composed of the following layers:*



We call this isometry AP normal form (iAPNF).

Proof. All internal X spiders (those labeled with phases $b_{k+1}\pi$ to $b_{k+s_z}\pi$) have an output pivot qubit. This necessarily follows as we started in rAPNF, where all X spiders have pivots; throughout the process of attaining iAPNF, we could change some of these X spiders to be input pivots or create new X spiders as input pivots. However, all cases preserved existing pivots of all other X spiders, as evidenced by Remarks 4.1 and 4.2 for Cases 3 and 4, and all other cases not affecting other X spiders. As a consequence, the output diagonal Clifford unitary D_2 acts only upon $n - s_z$ qubits—those that are not pivots of any X spiders.

By reordering the physical qubits to make all such X spiders ordered first from top to bottom, we can make the Z-type stabiliser generator pivots form an identity matrix for the leftmost columns of the stabiliser tableau. □

A similar observation was made in [59], where a stabiliser code encoder is decomposed into a CSS code followed by Paulis and a diagonal Clifford unitary.

Proof. □

It is interesting that we are able to recover iAPNF without needing more than three layers of unitary gates on the logical qubits, only two of which are entangling. This number of distinct circuits is certainly fewer than the number of all possible Clifford unitaries on k qubits. Indeed, there are unitaries that are missing. Notably, the SWAP gate is not decomposable into these three layers. An easy way to see this is to conjugate the CZ and DX gate by the H gate on one qubit each to change them to CX gates; after this, the maximum number of entangling gates between any pair of qubits is two CX gates. However, three CX gates are necessary to implement SWAP.

Another way to look at this, is that unitaries not generatable by these three layers, satisfy a symmetry property with respect to all possible iAPNF diagrams. For example, any SWAP of inputs has the same action as changing columns of the parameters $(B,\vec{b})$ of iAPNF. Another missing unitary, CX, has the same action as a change of B, specifically adding the column of the target to the column of the control. As for $Z(\frac{\pi}{2})$ and $X(\frac{\pi}{2})$, they both have the same action as a change of A—the former by pushing through the encoder, and the latter by local complementation.

In this section, we provide a normalization algorithm that maps T to its unique normal form N. Then we show how to recover T given N. For $1 \leq j \leq m$, we show how to read off stabiliser generators S_j from N. For $1 \leq j \leq k$, we show how to find all logical $\overline{X}_j$ and $\overline{Z}_k$ by pushing π spiders through N. This establishes a one-to-one correspondence between a $[\![n,k,d]\!]$ stabiliser code and its ZX normal form.

In quantum error correction, we are interested in encoder maps (i.e., Clifford isometries) and measurement circuits. By the Choi-Jamiolkowski isomorphism, for any $k > 1$ stabiliser code, we can convert its encoder map to a stabiliser state and rewrite it into the reduced AP form using the normalization algorithms in Section 4. To recover the isometry, we can "un-Choi" the normalized state. That is, bend the output qubit wires that were originally inputs back into the input wires.

In what follows, we show how to factor the un-Choi'ed reduced AP form into a three-layered circuit, as shown below.

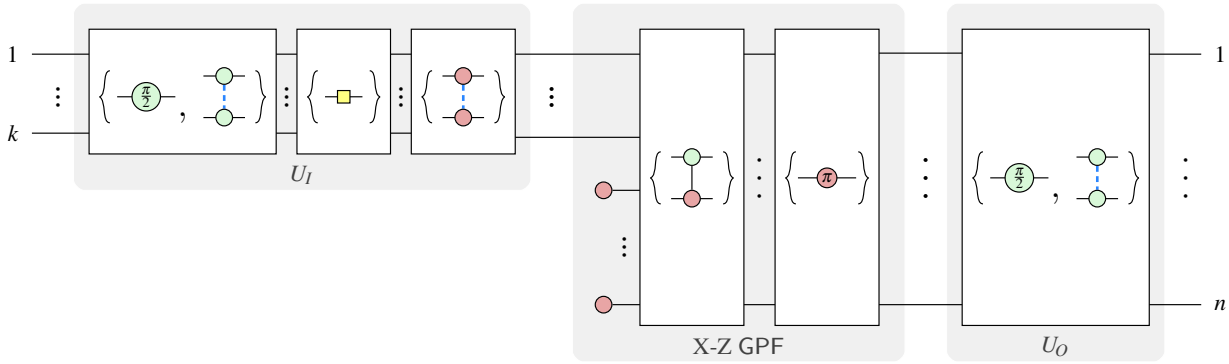
Recall that each $[\![n,k,d]\!]$ stabiliser code is unqiuely determined by a complete stabiliser tableau, $T \in \mathbb{F}_2^{(n+k)\times 2n}$, $m = n-k$, $n,k \in \mathbb{N}$, and $1 \leq k < n$.

$$T = \begin{array}{|c|c|c|c|}
\hline
 & X_1 \;\cdots\; X_n & Z_1 \;\cdots\; Z_n & p \\
\hline
\begin{matrix} S_1 \\ \vdots \\ S_m \end{matrix} & \ddots & \ddots & \vdots \\
\hline
\begin{matrix} \overline{X}_1 \\ \vdots \\ \overline{X}_k \end{matrix} & \ddots & \ddots & \vdots \\
\hline
\begin{matrix} \overline{Z}_1 \\ \vdots \\ \overline{Z}_k \end{matrix} & \ddots & \ddots & \vdots \\
\hline
\end{array}$$

5 Clifford Circuit Isometry AP Normal Form

In Section 4, we established the isometry AP normal form in the ZX-calculus. While it is apparent how to decompose the input and output unitaries U_I and U_O into layers of unitary gates, we require further steps to derive a quantum circuit decomposition for the filling of the sandwich, the X-Z GPF. In this section, we will prove the following theorem:

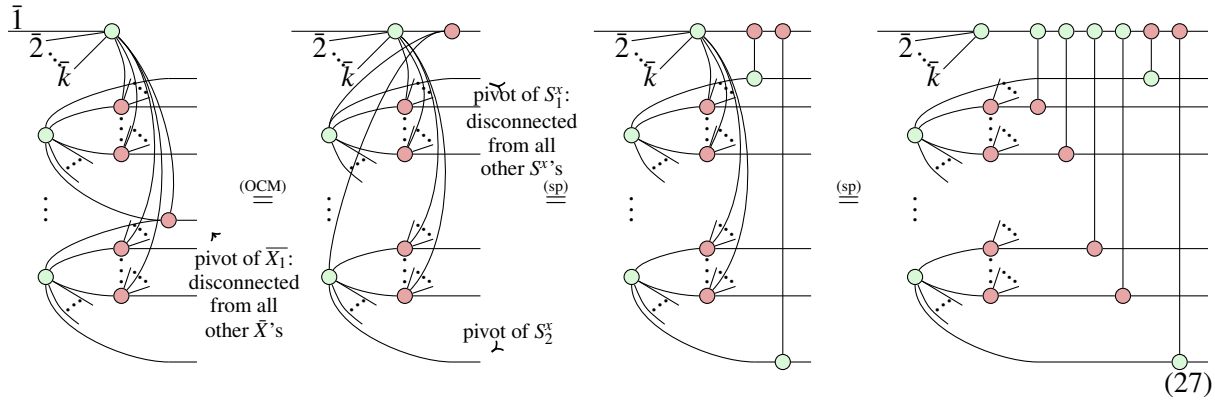
Theorem 5.1 (Circuit Isometry AP Normal Form)**.** *Every Clifford isometry can be written to a* quantum circuit normal form *composed of the following layers:*



5.1 Circuit Extraction

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

Proposition 5.2. For any CSS code encoder in phase-free Z-X (or X-Z) GPF, we can always extract a CNOT circuit from it. First, the *Z-X GPF* must be in RREF_X (or *X-Z GPF* in RREF_Z) form; if not, it can be efficiently rewritten into this form by Gaussian elimination as described in the previous section. The key step of the circuit extraction is given below.



(27)

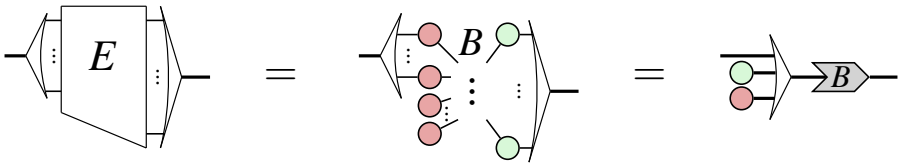
Here, the pivot of each X logical operator is moved as to be on the same horizontal qubit wire as the Z spider for that logical qubit, by permuting the order the outputs are drawn. Following that, CX gates can be unfused for all other X spiders connecting to that Z spider.

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

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

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

Corollary 5.3. As a result, we can leverage the Scalable ZX calculus to concisely represent phase-free isometries. Any CNOT circuit can be represented as a map in $\mathbb{F}_2$ by a reversible binary matrix. Therefore, any CSS code encoder can be drawn in the *scalable* ZX calculus as:

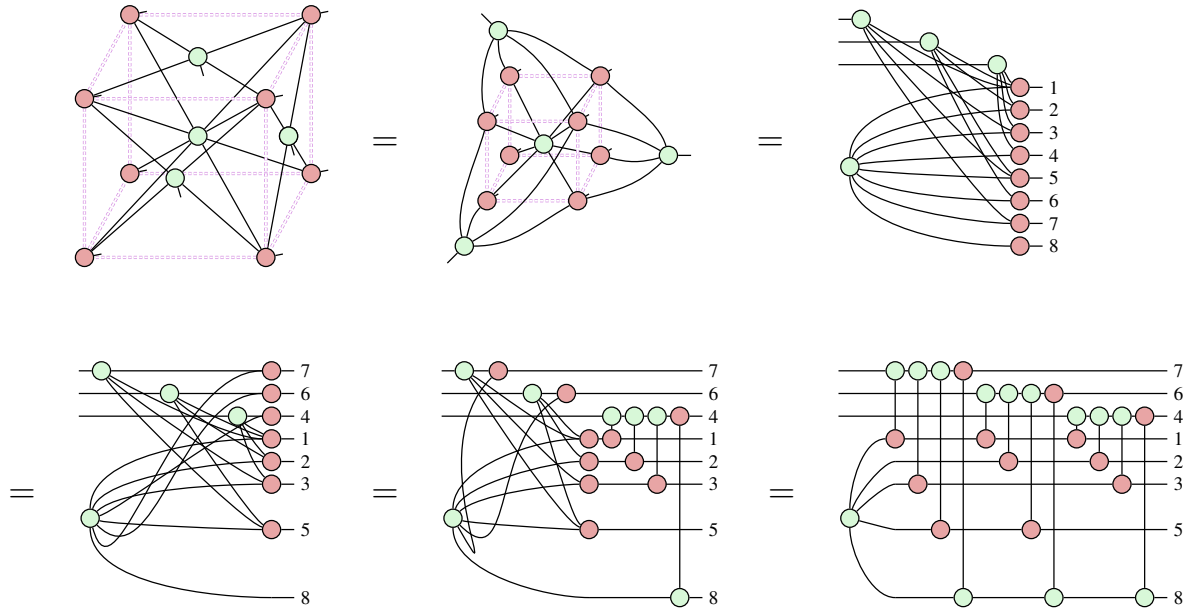


(28)

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

Next, we consider an example with more than one logical qubit.

Example 5.1. Let us apply the above procedure to the $[\![8,3,2]\!]$ code encoder. This is commonly drawn as a cube; to illustrate this cube shape, we have superimposed a cube in pink on top of the ZX diagram of its encoder map. Then the X logical operators are on three independent faces of the cube, while the weight 8 X-type stabiliser generator is in the volume of the cube.



(29)

Note that following the last step here, the 5-qubit GHZ state can be further decomposed into 5 CNOT gates acting on one $|+\rangle$ state and four $|0\rangle$ states.

Remark 5.1. Circuit extraction to a unitary embedding of the $[\![8,3,2]\!]$ entirely by ZX calculus rewrites was previously done in Ref. [25, Proposition 5.3], and moreover believed to be the first time a circuit for

this code was given. They achieve this using 56 rewrite rules in the ZX calculus proof assistant Quantomatic: The first half of this is automated, and the latter half required substantial human intervention.

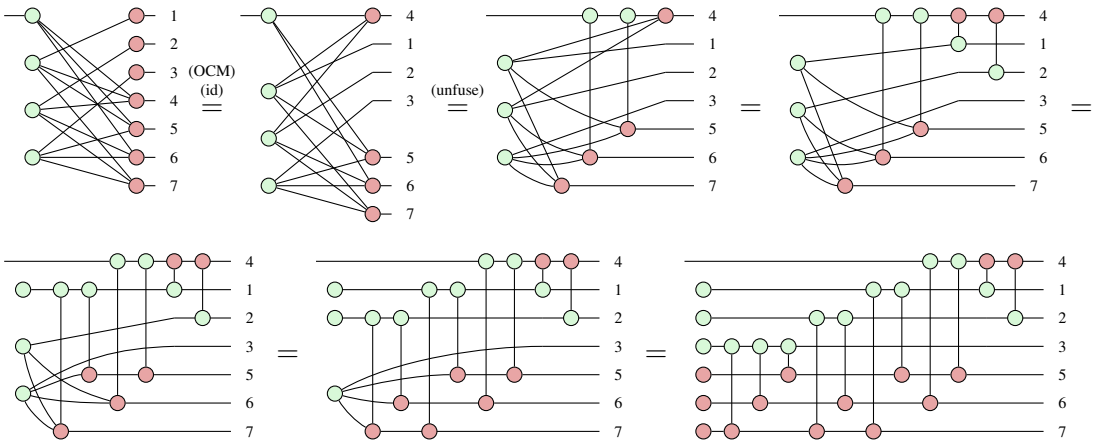
To explain how our procedure improves upon this, the primary contribution is that this is an algorithm applicable to all CSS codes, and as we will later explain, to all stabiliser codes with negligible modification. Additionally, this is both efficiently automated, and easy to do by hand so long as the code itself is small enough to work with.

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

Example 5.2. Let us apply the above ciruit extraction procedure to the $[\![7,1,3]\!]$ Steane code encoder. Its RREF_X matrix is

$$\left[\begin{array}{c|ccccccc} 1&2&3&4&5&6&7 \\ \hline 1&0&0&1&1&0&1 \\ 0&1&0&1&0&1&1 \\ 0&0&1&0&1&1&1 \\ 0&0&0&1&1&1&0 \end{array}\right]$$

where the the first row is a choice of indices of the physical qubits, the last row is for the X logical operator, and the middle 3 rows are the X-type stabiliser generators. Starting with the *Z-X GPF* of the above matrix, the circuit extracts to:



(30)

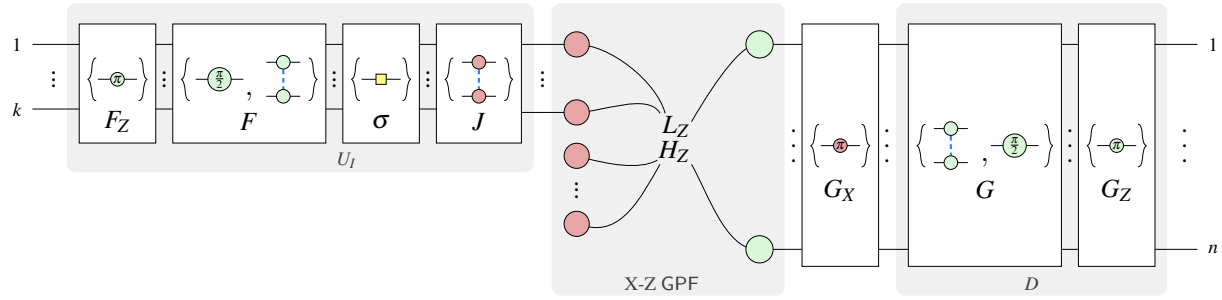
This allows representing CSS code encoders in a new normal form in the scalable ZX-calculus, which we call SZX-calculus CX Encoder Normal Form. We show this and how it enables a succinct ZX-calculus proof that all CSS codes admit transversal CX gates in the appendix.

6 Stabiliser Tableau Isometry AP Normal Form

In this section, we identify a new stabiliser tableau normal form, corresponding to the isometry AP normal form for ZX-diagrams established in the previous section. In the first half, we show how to efficiently read off a complete stabiliser tableau from any ZX-diagram in isometry AP normal form. In the second half, we describe the form of the corresponding stabiliser tableau and prove it is a *normal form*: that is, we prove we can always efficiently convert any stabiliser tableau into this form.

6.1 Stabiliser tableaus from isometry AP normal form

In order to read off a complete stabiliser tableau from any ZX-diagram in isometry AP normal form, we define the matrices F_Z, F, σ, J, G_X, G, and G_Z as to perform the maps in $\mathbb{F}_2$ linearly transforming the stabiliser tableau, for each layer between the inner phase-free encoder and the complete isometry AP normal form.



Procedure 3: Identifying a complete stabiliser tableau from isometry AP normal form

Input: A ZX-diagram in iAPNF

Output: A complete stabiliser tableau

1. We can directly read off the pure Z-type stabiliser generators H_Z: It is simply the biadjacency matrix of the internal X spiders to the output Z spiders, acquiring a sign of -1 whenever that internal X spider has a π phase.
2. We can read off the Z logical operators L_Z with one extra step: It is the biadjacency matrix of the input X spiders, pushed through U_I, acquiring a sign of -1 whenever that input X spider has a π phase.
3. Unfuse any π phases on the input X spiders to the left and unfuse any π phases on the internal X spiders to the right along their pivots, then apply Algorithm 5 to efficiently rewrite the inner CSS code phase-free encoder from X-Z to Z-X GPF, and finally copy those π phases through the input Z spiders to form a layer of X gates between phase-free Z-X GPF and the diagonal Clifford unitary of the isometry AP normal form.
4. We can read off X-type stabiliser generators H_X and the X logical operators L_X, by pushing H_X and L_X through the rightmost diagonal Clifford unitary, and pushing L_X through U_I, to obtain the mixed stabiliser generators $H_X G$. The sign of a stabiliser generator is flipped whenever a Pauli X commutes past a Z phase of either π or $\frac{3\pi}{2}$ in either diagonal Clifford unitary.

Remark 6.1. If taking the joint +1 eigenspace convention, as is common for stabiliser codes in the literature, there is no $\{X\}$ layer, and the only allowed powers of S in the final layer are 0 and 1.

Lemma 6.1.

$$\begin{bmatrix} H_x & 0 \\ 0 & H_z \end{bmatrix} \begin{bmatrix} I & G \\ 0 & I \end{bmatrix} = \begin{bmatrix} H_x & H_x G \\ 0 & H_z \end{bmatrix} \tag{31}$$

The complete stabiliser tableau of the inner CSS code is:

$$\tau = \begin{array}{c|c|c|c} k & k & n & n \\ \hline I & 0 & L_x & 0 \\ \hline 0 & I & 0 & L_z \\ \hline 0 & 0 & H_x & 0 \\ \hline 0 & 0 & 0 & H_z \end{array} \tag{32}$$

Let

$$L_k = \begin{bmatrix} L_k^A & L_k^B \\ L_k^C & L_k^D \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & 0 \\ J & I \end{bmatrix} \sigma \begin{bmatrix} I & F \\ I & 0 \end{bmatrix}. \tag{33}$$

The signed complete stabiliser tableau in bijection with isometry AP normal form, which we shall call *Stabiliser Tableau Isometry AP Normal Form*, is as follows:

$2k$	n	n	sign
L_k	L_x	$L_x G$	$L_k^A F_z^T + L_x G_z^T$
	0	L_z	$L_k^C F_z^T + L_z G_x^T$
0	H_x	$H_x G$	$H_x G_z^T$
	H_z	0	$H_z G_x^T$

(34)

6.2 Writing any stabiliser tableau into the normal form

We next present an efficient procedure to write down a ZX-diagram for any k-input, n-output Clifford isometry, given any encoder tableau of it. Note that this procedure identically applies to when $k = 0$, the special case of any stabiliser state, and $k = n$, the special case of any Clifford unitary.

Procedure 4: Synthesising a ZX-diagram from any encoder tableau

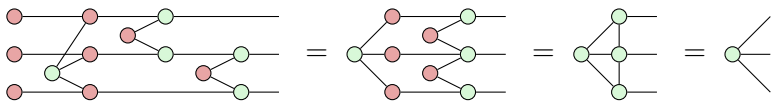
Input: An encoder tableau for any Clifford isometry

Output: A ZX-diagram for the linear map uniquely determined by the tableau

1. If this is an isometry, reorder the columns of the tableau so that the $2k$ columns for its k inputs and $2n$ columns for its n outputs match that of its Choi state: $n+k$ columns on the left for the Z action and $n+k$ columns on the right for the X action.
2. For every row of the tableau, write down the ZX-diagram for the projector of that Pauli operator, sequentially composed with all already initialised input qubits. For all not yet initialised input qubits, input to the measure box:
   - If I, leave the qubit not yet initialised.
   - If X, input $|0\rangle$.
   - If Y, input $|+\rangle$.
   - If Z, input $|+\rangle$.
3. If k was non-zero, un-Choi the resulting ZX-diagram by bending the first k output wires to become inputs.
4. Optional: Write the ZX-diagram to isometry AP normal form, by Algorithm 2.

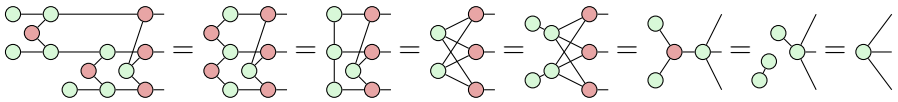
As an example, consider the GHZ state. With its rows ordered XXX, ZZI, IZZ, we write down the

leftmost ZX-diagram below.



(35)

With its rows ordered ZZI, IZZ, XXX, we write down the leftmost ZX-diagram below.



(36)

We see that both Equations 35 and 36 synthesise ZX-diagrams for the GHZ state, as expected.

The correctness of Procedure 4 is proven by showing that the resulting ZX-diagram is stabilised by all $n+k$ linearly independent Pauli operators in the encoder tableau, and therefore represents the unique linear map determined by the encoder tableau.

Proof. Consider the ZX-diagram of the Choi state of the encoder output from Procedure 4, acted on by Pauli operator defined by one row of the encoder tableau. Push the Pauli operator left, until it is just to the right of the projector identified with it; this is always possible because of the property that all rows of the encoder tableau must commute with each other. The Pauli operator can then be fused with the spiders on the Pauli boxes of the projector, then un-fused and commuted until it becomes "absorbed" into the X spider of the projector—equivalently, the projector is stabilised by its corresponding Pauli operator. □

Evidently, following the tableau-to-ZX Procedure 4 by the ZX-to-tableau Procedure 3 gives an efficient way to write any encoder stabiliser tableau into complete stabiliser tableau isometry AP normal form.

This raises the question of whether we can factorise stabiliser tableaus which do not have specified logical operators, into the form implied by the isometry AP normal form we derived in the ZX-calculus. We prove this in the affirmative, in the remainder of this subsection.

Theorem 6.1. *Given any stabiliser tableau for any stabiliser code, it can be efficiently written into the normal form*

$$T = \left(\begin{array}{c|c} H_X & H_X G \\ 0 & H_Z \end{array}\right) \tag{37}$$

where G is a symmetric matrix, and all rows are linearly independent.

This factorisation into H_X, H_Z, and G gives an explicit and efficient algorithm to isolate the inner CSS code and diagonal Clifford unitary D decomposition of any stabiliser code, an observation from [59, Appendix C] which linked a software implementation of an exponential-time algorithm to find D, based on [58, Algorithm 5.3.1] in the XP formalism [57].

While the normal form of Theorem 6.1 is simple, we were unable to find it in the literature or through discussions. To arrive at it, first, consider a stabiliser tableau of the following form:

$$T = \left(\begin{array}{c|c} H_X & N_X \\ 0 & H_Z \end{array}\right) \tag{38}$$

Note that we adhere to the convention of the joint +1 eigenspace and thereby do not need an additional column for the sign of each stabiliser generators.

Up to re-ordering rows, this is just an arbitrary stabiliser tableau, to which we have chosen to make the rows of $H_X \in \mathbb{F}_2^{s_x \times n}$ linearly independent. We can always do this. Suppose row r of H_X is a linear

combination of some other rows of H_X. Then we can simply add those rows to row r and replace row r with a pure-Z stabiliser, which we incorporate into H_Z. The number of rows s_z of $H_Z \in \mathbb{F}_2^{s_z \times n}$ is the maximum number of pure-Z stabiliser generators supported in the codespace. s_x and s_z are allowed to be zero.

Since the rows of H_X are linearly independent, it has a right inverse R, i.e. a matrix such that $H_X R = I$. This implies that we can write $N_Z = H_X G$ for some matrix G—namely, $G := RN_Z$.

We can in fact always choose G to be symmetric, i.e. $G = G^T$. One way to see this, is that this form of tableau corresponds to AP normal form, which any stabiliser state can be rewritten to. H_Z and H_X specify the connectivity of the affine part of the ZX diagram (in X-Z form and Z-X form, respectively). Accordingly, G defines the diagonal Clifford part consisting of S and CZ gates. If $G_{ij} = G_{ji} = 1$, introduce an S gate if $i = j$ and a CZ gate otherwise. Therefore, we can always find a G which is symmetric, namely the symplectic representation of the diagonal Clifford part of the AP normal form ZX-diagram.

Conversely, given any CSS code (which is the special case of $G = 0$), we can always "twist" it with an adjacency matrix G to get a new code:

Lemma 6.2. If the following matrix:

$$S = \left(\begin{array}{c|c} H_X & 0 \\ 0 & H_Z \end{array}\right)$$

is symplectic, then so too is

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

for any symmetric G.

Proof. We already know the H_Z part "commutes" with everything, so we only need to check the lower half of the matrix is symplectic, i.e. that $H_X(H_X G)^T + (H_X G)H_X^T = 0$. Because we are working in $\mathbb{F}_2$, this is equivalent to the following condition, thereby completing the proof.

$$H_X(H_X G)^T = H_X G^T {H_X}^T = (H_X G){H_X}^T \tag{39}$$

□

This gives us a recipe for directly writing down stabiliser tableaus of any $k = 0$ stabiliser code, through procedures entirely within the ZX-calculus. From AP normal form which is determined by $(B, \vec{b}, A)$, we can write down $H_Z = B$ directly. As the codespace of a stabiliser code is the joint $+1$ eigenspace of all stabiliser generators, it must be the case that $\vec{b} = 0$ and A is binary. This is because the presence of any X or Z gate changes at least one $+1$ eigenvalue of the constituent CSS code to -1. This lets us write down $G = A$. Finally, we can convert the X-Z GPF subdiagram to Z-X GPF, enabling us to read off H_X as its biadjacency matrix.

This also gives us a recipe in the other direction, for directly writing down an AP normal form ZX-diagram given H_Z and G. Note that H_X is not necessary in this direction, but only because the bottom left quadrant (the quadrants can be different sizes) of the tableau is known to be zero. This is because it takes less information to fix a state, than to reconstruct a basis for all its stabiliser generators.

One might hope that there is a general way to write down a ZX-diagram in normal form from an arbitrary stabiliser tableau without having to do this 'preprocessing' of the tableau. However, this does not seem likely considering the analogy to CSS codes. If we are given a CSS code tableau where the generators are neatly separate into the Z and X parts, then we can directly read off the phase-free ZX-diagram that implements it. However, if we are given a 'scrambled' version of the tableau where the Z

and X part are mixed, then it is no longer obvious that it is in fact a CSS code. The only way we can recover the knowledge that it was a CSS code (and hence should correspond to a phase-free diagram), is by unscrambling it, which corresponds to solving a set of linear equations similar to what we described above.

In fact, we find that we can give a constructive, tableau-only algorithm to put any stabiliser tableau into this tableau normal form, i.e. a triple (H_X, H_Z, G). The linear equations that suffice to be solved is precisely the above procedure to put an arbitrary tableau into the form of Equation (38), then finding a right inverse of H_X: any $R \in \mathbb{F}_2^{n\times s_x}$ satisfying $H_X R = I_{s_x}$. One common choice of right inverse is the Moore-Penrose inverse, ${H_X}^\dagger \left(H_X {H_X}^\dagger\right)^{-1}$.

Lemma 6.3. Define

$$G := RN_Z + N_Z^T R^T - RH_X N_Z^T R^T.$$

Then $G \in \mathbb{F}_2^{n\times n}$ satisfies

$$H_X G = N_Z.$$

Proof. Using $H_X R = I_m$, we compute

$$\begin{aligned} H_X G &= H_X RN_Z + H_X N_Z^T R^T - H_X RH_X N_Z^T R^T \\ &= N_Z + H_X N_Z^T R^T - H_X N_Z^T R^T \\ &= N_Z. \end{aligned}$$

□

Lemma 6.4. If G defined in Lemma 6.3 satisfies $G = G^T$, then

$$H_X N_Z^T = N_Z H_X^T.$$

Proof. Taking the transpose of the definition of G yields

$$G^T = N_Z^T R^T + RN_Z - RN_Z H_X^T R^T.$$

Thus $G = G^T$ holds if and only if

$$RH_X N_Z^T R^T = RN_Z H_X^T R^T.$$

Multiplying on the left by H_X and on the right by H_X^T, and using $H_X R = I_m$, gives

$$H_X N_Z^T = N_Z H_X^T.$$

□

Lemma 6.5. If

$$H_X N_Z^T = N_Z H_X^T,$$

then the matrix G defined in Lemma 6.3 is symmetric.

Proof. Assuming $H_X N_Z^T = N_Z H_X^T$, the equality

$$RH_X N_Z^T R^T = RN_Z H_X^T R^T$$

holds. Substituting this into the expressions for G and G^T shows that $G = G^T$. □

Proposition 6.2. There exists a symmetric matrix $G \in \mathbb{F}_2^{n\times n}$ such that $H_X G = N_Z$, if and only if $H_X N_Z^T = N_Z H_X^T$. When this condition holds, a symmetric solution is given by

$$G = RN_Z + N_Z^T R^T - RH_X N_Z^T R^T,$$

where R is any right inverse of H_X.

Proof. Assuming G is symmetric, Lemma 6.4 shows that $H_X N_Z^T = N_Z H_X^T$ must hold. In the other direction, assuming $H_X N_Z^T = N_Z H_X^T$, Lemma 6.5 shows that G must be symmetric. □

To finalise our proof of Theorem 6.1, we observe that $H_X N_Z^T = N_Z H_X^T$ is precisely the condition for the stabiliser tableau to be well-defined, i.e. symplectic. Therefore, given any stabiliser tableau for any stabiliser code, we can always find a symmetric $G \in \mathbb{F}_2^{n\times n}$ such that the tableau can be written in the form of Theorem 6.1. This gives an efficient factorisation of any stabiliser code into the inner CSS code defined by H_X and H_Z, together with this diagonal Clifford unitary defined by G.

6.3 Completing incomplete stabiliser tableaus

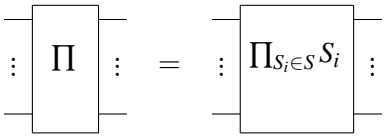
In this section, we consider the problem of starting with a stabiliser tableau for a stabiliser code, which we will refer to as *incomplete* because it has no information about the logical operators. We present an efficient and fully graphical algorithm to produce from this a *complete* stabiliser tableau, which fixes a choice of logical operators and therefore determines an encoder map for the code. This is done by focusing solely on the inner CSS code of the stabiliser code in the phase-free ZX-calculus, made possible because by the theorem proven in the previous subsection.

Remark 6.2. In this section, we impose a restriction to only X logical operators over $\{X, I\}$ and Z logical operators over $\{Z, I\}$. This is nevertheless applicable to any CSS code because a CSS code fixes the *image* of the encoder map, but not the encoder map itself. In fact, changing the choice of logical operators does not change the code distance.

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

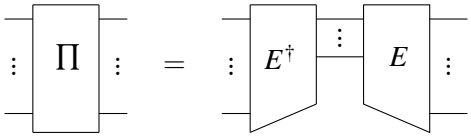
We can apply the reasoning of Remark 6.2 for phase-free encoders, to all encoders for CSS codes, by virtue of the fact that any choice of logical operator is always unitarily equal to a phase-free choice of logical operators of the form above.

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



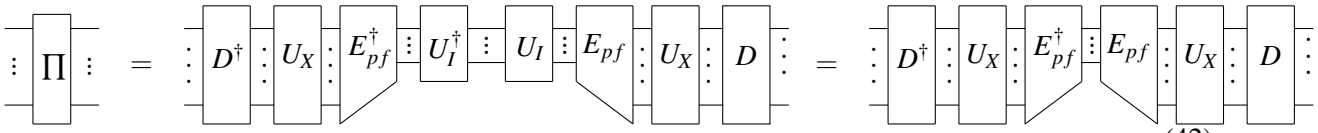
(40)

The same projector on the code space, Π, can also be written as $E^\dagger$ then E, i.e. the ideal decoder followed by the encoder.



(41)

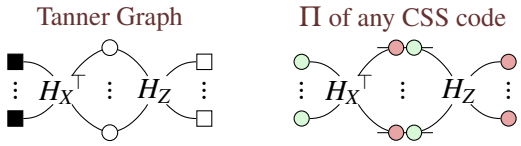
From our normal form for isometries, we have that this further factorises to the projector of the inner CSS code, conjugated by the layers generatored by $\{X,Z,S,andCZ\}$ gates.



(42)

The projector for any CSS code can be written as a ZX-diagram in what we shall call Tanner graph normal form.

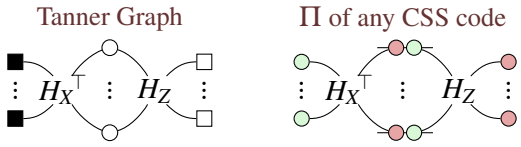
Definition 6.1 (Tanner graph normal form)**.** A phase-free ZX-diagram for the projector onto the code space of a CSS code is in *Tanner graph normal form* if it consists of two bipartite graphs, the left being the projector onto the code space for all X-type stabiliser generators fused together, and the right being the projector onto the code space for all Z-type stabiliser generators fused together:



(43)

This gives a natural definition for a normal form for the projector onto the code space of any stabiliser code.

Proposition 6.3 (Generalised Tanner graph normal form)**.** A stabiliser ZX-diagram for the projector onto the code space of any stabiliser code is in *Generalised Tanner graph normal form* if it consists of the projector onto the code space for the inner CSS code in Tanner graph normal form, conjugated by the layers generatored by $\{X,Z,S,andCZ\}$ gates. Pushing the $\{X,Z\}$ gates into the Tanner graph normal form, we have the following signed Tanner graph normal form conjugated by $\{S,CZ\}$ gates:

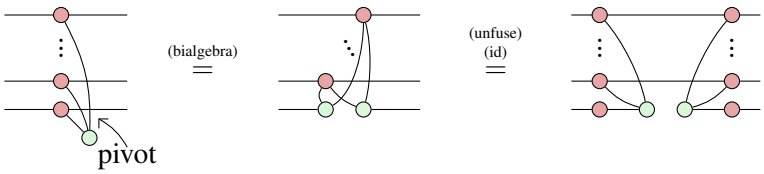


(44)

Therefore,

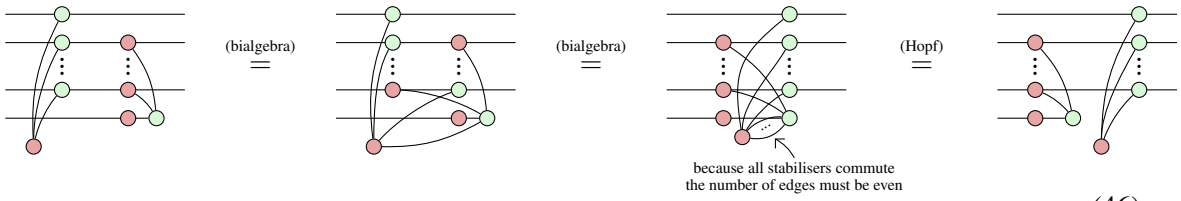
Proposition 6.4. Given a stabiliser tableau for any CSS code, i.e. a set of $n-k$ linearly independent stabiliser generators without specifying the logical operators, we can extract a phase-free encoder map in *Z-X GPF* . This implies that there always exists a choice of logical operators for any CSS code, such that all X logical operators are over $\{X,I\}$ and all Z logical operators are over $\{Z,I\}$.

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



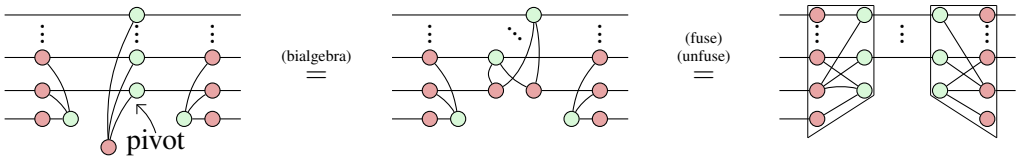
(45)

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



(46)

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



(47)

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

Corollary 6.5. In the diagrams above, it can be seen that the stabiliser generators fix the image of E, and moreover that there are $k=n-s$ wires in the middle, for k logical qubits. To preserve the equality of Equation (40), no modifications can be done to the input n wires nor the output n wires of $EE^\dagger$, because doing so would make it no longer equal to $\prod$. The only modification which preserves this equality and the number of logical qubits k, is prepending the phase-free E by a unitary U which defines a new encoder, whilst appending the phase-free $E^\dagger$ by $U^\dagger$ which equals the dagger of the new encoder.

7 Discussion

The results of this paper enable us to generalise to stabiliser codes, the recipe for gauge fixing CSS codes in the ZX-calculus of [34] summarised in Figure 15.

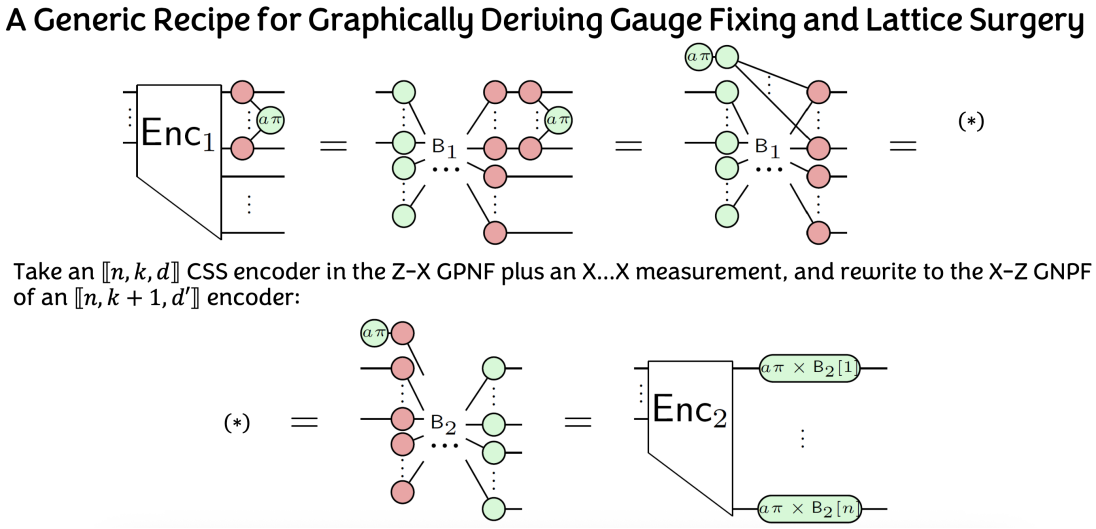


Figure 15: We can generalise any CSS code gauge fixing procedure, to any stabiliser code whose inner CSS code is that CSS code.

The claim of Figure 15 follows from replacing the above E by isometry AP normal form, measure Z...Z instead of X...X for the gauge-fixing measurement of the stabiliser being added, doing the same proof after changing the inner CSS X-Z GPF to Z-X GPF, and one final step of pushing the X's from copying through the diagonal Clifford unitary to get the single-qubit corrections conditional on the gauge fixing measurement outcome.

Acknowledgement We acknowledge the use of automated language tools for editorial assistance. The circuit diagrams in this paper were typeset using TikZiT [38]. LY is thankful for funding from Google PhD Fellowship, and for former funding by the Oxford - Basil Reeve Graduate Scholarship at Oriel College, University of Oxford with the Clarendon Fund.

References

[1] Scott Aaronson & Daniel Gottesman (2004): *Improved simulation of stabilizer circuits*. *Physical Review A* 70(5), p. 052328, doi:10.1103/PhysRevA.70.052328.

[2] Clarice Dias de Albuquerque, Eduardo Brandani da Silva & Waldir Silva Soares Jr (2022): *Quantum Codes for Topological Quantum Computation*. Springer Nature, doi:https://doi.org/10.1007/978-3-031-06833-1.

[3] Miriam Backens (2014): *The ZX-calculus is complete for stabilizer quantum mechanics*. *New Journal of Physics* 16(9), p. 093021.

[4] Miriam Backens (2014): *The ZX-calculus is complete for stabilizer quantum mechanics*. *New Journal of Physics* 16(9), p. 093021, doi:10.1088/1367-2630/16/9/093021.

[5] Miriam Backens, Hector Miller-Bakewell, Giovanni de Felice, Leo Lobski & John van de Wetering (2021): *There and back again: A circuit extraction tale*. *Quantum* 5, p. 421.

[6] Niel de Beaudrap & Dominic Horsman (2020): *The ZX calculus is a language for surface code lattice surgery*. *Quantum* 4, p. 218, doi:10.22331/q-2020-01-09-218.

[7] A. R. Calderbank & Peter W. Shor (1996): *Good quantum error-correcting codes exist*. *Physical Review A* 54, pp. 1098–1105, doi:10.1103/PhysRevA.54.1098.

[8] A Robert Calderbank, Eric M Rains, Peter W Shor & Neil JA Sloane (1997): *Quantum error correction and orthogonal geometry*. *Physical Review Letters* 78(3), p. 405, doi:10.1103/PhysRevLett.78.405.

[9] Titouan Carette, Dominic Horsman & Simon Perdrix (2019): *SZX-calculus: scalable graphical quantum reasoning*. In: *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 138, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 55:1–55:15, doi:10.4230/LIPIcs.MFCS.2019.55. Available at `http://drops.dagstuhl.de/opus/volltexte/2019/10999`.

[10] Nicholas Chancellor, Aleks Kissinger, Joschka Roffe, Stefan Zohren & Dominic Horsman (2023): *Graphical Structures for Design and Verification of Quantum Error Correction*, doi:10.48550/arXiv.1611.08012. arXiv:1611.08012.

[11] Bob Coecke & Ross Duncan (2008): *Interacting quantum observables*. In: *ICALP*, Lecture Notes in Computer Science, pp. 298–310, doi:10.1007/978-3-540-70583-3_25.

[12] Bob Coecke & Ross Duncan (2011): *Interacting quantum observables: categorical algebra and diagrammatics*. *New Journal of Physics* 13(4), p. 043016, doi:10.1088/1367-2630/13/4/043016.

[13] Bob Coecke & Aleks Kissinger (2017): *Picturing quantum processes*. Cambridge University Press, doi:10.1007/978-3-319-91376-6_6.

[14] Bob Coecke & Quanlong Wang (2018): *ZX-rules for 2-qubit Clifford+T quantum circuits*. In Jarkko Kari & Irek Ulidowski, editors: *Reversible Computation*, Springer International Publishing, Cham, pp. 144–161, doi:10.1007/978-3-319-99498-7_10.

[15] Alexander Cowtan (2022): *Qudit lattice surgery*, doi:10.48550/arXiv.2204.13228. arXiv:2204.13228.

[16] Alexander Cowtan & Simon Burton (2023): *CSS code surgery as a universal construction*. *arXiv preprint arXiv:2301.13738*, doi:10.48550/arXiv.2301.13738.

[17] Jeroen Dehaene & Bart De Moor (2003): *Clifford group, stabilizer states, and linear and quadratic operations over GF(2)*. *Phys. Rev. A* 68, p. 042318, doi:10.1103/PhysRevA.68.042318. Available at `https://link.aps.org/doi/10.1103/PhysRevA.68.042318`.

[18] Lucas Dixon, Ross Duncan & Aleks Kissinger (2010): *Open Graphs and Computational Reasoning*. *Electronic Proceedings in Theoretical Computer Science* 26, p. 169–180, doi:10.4204/eptcs.26.16. Available at `http://dx.doi.org/10.4204/EPTCS.26.16`.

[19] Ross Duncan, Aleks Kissinger, Simon Perdrix & John Van De Wetering (2020): *Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus*. *Quantum* 4, p. 279.

[20] Ross Duncan & Maxime Lucas (2014): *Verifying the Steane code with Quantomatic*. *Electronic Proceedings in Theoretical Computer Science* 171, pp. 33–49, doi:10.4204/eptcs.171.4. Available at `https://doi.org/10.4204%2Feptcs.171.4`.

[21] Ross Duncan & Simon Perdrix (2009): *Graph states and the necessity of Euler decomposition*. In: *Mathematical Theory and Computational Practice: 5th Conference on Computability in Europe, CiE 2009, Heidelberg, Germany, July 19-24, 2009. Proceedings 5*, Springer, pp. 167–177.

[22] Ross Duncan & Simon Perdrix (2014): *Pivoting makes the ZX-calculus complete for real stabilizers*. *Electronic Proceedings in Theoretical Computer Science* 171, p. 50–62, doi:10.4204/eptcs.171.5. Available at `http://dx.doi.org/10.4204/EPTCS.171.5`.

[23] Matthew B. Elliott, Bryan Eastin & Carlton M. Caves (2008): *Graphical description of the action of Clifford operators on stabilizer states*. *Phys. Rev. A* 77, p. 042307, doi:10.1103/PhysRevA.77.042307. Available at `https://link.aps.org/doi/10.1103/PhysRevA.77.042307`.

[24] Austin G Fowler, Matteo Mariantoni, John M Martinis & Andrew N Cleland (2012): *Surface codes: Towards practical large-scale quantum computation*. *Physical Review A* 86(3), p. 032324.

[25] Liam Garvie & Ross Duncan (2017): *Verifying the smallest interesting colour code with Quantomatic*. *arXiv preprint arXiv:1706.02717*, doi:10.4204/EPTCS.266.10.

[26] Craig Gidney (2022): *A pair measurement surface code on pentagons*, doi:10.48550/arXiv.2206.12780. arXiv:2206.12780.

[27] Craig Gidney & Austin G. Fowler (2019): *Efficient magic state factories with a catalyzed |CCZ> to 2|T> transformation*. Quantum 3, p. 135, doi:10.22331/q-2019-04-30-135.

[28] Craig Gidney & Austin G. Fowler (2019): *Flexible layout of surface code computations using AutoCCZ states*, doi:10.48550/arXiv.1905.08916. Available at `https://arxiv.org/abs/1905.08916`.

[29] Daniel Gottesman (1997): *Stabilizer codes and quantum error correction*. Ph.D. thesis, California Institute of Technology, doi:10.48550/arXiv.quant-ph/9705052.

[30] Daniel Gottesman (1998): *The Heisenberg Representation of Quantum Computers*. Available at `https://arxiv.org/abs/quant-ph/9807006`.

[31] Daniel Gottesman (1998): *Theory of fault-tolerant quantum computation*. Physical Review A 57, pp. 127–137, doi:10.1103/PhysRevA.57.127.

[32] Amar Hadzihasanovic, Kang Feng Ng & Quanlong Wang (2018): *Two complete axiomatisations of pure-state qubit quantum computing*. In: ACM/IEEE Symposium on Logic in Computer Science, LICS '18, Association for Computing Machinery, New York, NY, USA, p. 502–511, doi:10.1145/3209108.3209128.

[33] Alexander Tianlin Hu & Andrey Boris Khesin (2022): *Improved graph formalism for quantum circuit simulation*. Physical Review A 105(2), doi:10.1103/physreva.105.022432.

[34] Jiaxin Huang, Sarah Meng Li, Lia Yeh, Aleks Kissinger, Michele Mosca & Michael Vasmer (2023): *Graphical CSS Code Transformation Using ZX Calculus*. Electronic Proceedings in Theoretical Computer Science 384, p. 1–19, doi:10.4204/eptcs.384.1. Available at `http://dx.doi.org/10.4204/EPTCS.384.1`.

[35] Emmanuel Jeandel, Simon Perdrix & Renaud Vilmart (2020): *Completeness of the ZX-calculus*. Logical Methods in Computer Science Volume 16, Issue 2, doi:10.23638/LMCS-16(2:11)2020.

[36] Andrey Boris Khesin (2022): *Quantum Computing from Graphs*. Phd thesis, Massachusetts Institute of Technology. Available at `https://arxiv.org/pdf/2501.17959`.

[37] Aleks Kissinger (2022): *Phase-free ZX diagrams are CSS codes (... or how to graphically grok the surface code)*. arXiv preprint arXiv:2204.14038, doi:10.48550/arXiv.2204.14038.

[38] Aleks Kissinger, Alexander Merry, Chris Heunen & K. Johan Paulsson: *TikZiT*. `https://tikzit.github.io/index.html`. Accessed: 2024-12-08.

[39] Aleks Kissinger & John van de Wetering (2024): *Picturing Quantum Software: An Introduction to the ZX-Calculus and Quantum Compilation*. Preprint.

[40] Emanuel Knill & Raymond Laflamme (1997): *Theory of quantum error-correcting codes*. Physical Review A 55(2), p. 900, doi:10.1103/PhysRevLett.84.2525.

[41] Raymond Laflamme, Cesar Miquel, Juan Pablo Paz & Wojciech Hubert Zurek (1996): *Perfect Quantum Error Correction Code*. arXiv:quant-ph/9602019.

[42] Andrew J Landahl, Jonas T Anderson & Patrick R Rice (2011): *Fault-tolerant quantum computing with color codes*. arXiv preprint arXiv:1108.5738, doi:https://doi.org/10.48550/arXiv.1108.5738.

[43] Tommy McElvanney & Miriam Backens (2023): *Complete Flow-Preserving Rewrite Rules for MBQC Patterns with Pauli Measurements*. Electronic Proceedings in Theoretical Computer Science 394, p. 66–82, doi:10.4204/eptcs.394.5. Available at `http://dx.doi.org/10.4204/EPTCS.394.5`.

[44] Maarten Van den Nest, Jeroen Dehaene & Bart De Moor (2004): *Graphical description of the action of local Clifford transformations on graph states*. Phys. Rev. A 69, p. 022316, doi:10.1103/PhysRevA.69.022316. Available at `https://link.aps.org/doi/10.1103/PhysRevA.69.022316`.

[45] Michael A. Nielsen & Isaac L. Chuang (2010): *Quantum computation and quantum information*, 10th anniversary ed edition. Cambridge University Press, doi:10.1017/CBO9780511976667.

[46] Boldizsár Poór, Robert I. Booth, Titouan Carette, John van de Wetering & Lia Yeh (2023): *The Qupit Stabiliser ZX-travaganza: Simplified Axioms, Normal Forms and Graph-Theoretic Simplification*. Electronic Proceedings in Theoretical Computer Science 384, p. 220–264, doi:10.4204/eptcs.384.13. Available at `http://dx.doi.org/10.4204/EPTCS.384.13`.

[47] D. Schlingemann (2001): *Stabilizer codes can be realized as graph codes*. arXiv:quant-ph/0111080.

[48] Dirk Schlingemann & Reinhard F Werner (2001): *Quantum error-correcting codes associated with graphs*. *Physical Review A* 65(1), p. 012308.

[49] Peter Selinger (2018): *Matrix Theory and Linear Algebra*. Available at `https://www.mathstat.dal.ca/~selinger/linear-algebra/downloads/LinearAlgebra.pdf`.

[50] Alexis T. E. Shaw, Michael J. Bremner, Alexandru Paler, Daniel Herr & Simon J. Devitt (2022): *Quantum computation on a 19-qubit wide 2d nearest neighbour qubit array*, doi:10.48550/arXiv.2212.01550. Available at `https://arxiv.org/abs/2212.01550`.

[51] Peter W. Shor (1995): *Scheme for reducing decoherence in quantum computer memory*. *Physical Review A* 52, pp. R2493–R2496, doi:10.1103/PhysRevA.52.R2493.

[52] Andrew Steane (1996): *Multiple-particle interference and quantum error correction*. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 452(1954), pp. 2551–2577, doi:10.1098/rspa.1996.0136.

[53] Andrew M Steane (1996): *Error correcting codes in quantum theory*. *Physical Review Letters* 77(5), p. 793.

[54] Andrew M Steane (1996): *Simple quantum error-correcting codes*. *Physical Review A* 54(6), p. 4741, doi:10.1103/PhysRevA.54.4741.

[55] Maarten Van Den Nes (2010): *Classical simulation of quantum computation, the Gottesman-Knill theorem, and slightly beyond*. *Quantum Info. Comput.* 10(3), p. 258–271.

[56] Renaud Vilmart (2019): *A near-minimal axiomatisation of ZX-calculus for pure qubit quantum mechanics*. In: *LICS 2019*, pp. 1–10, doi:10.1109/LICS.2019.8785765.

[57] Mark Webster (2023): *The XP Stabilizer Formalism*. Ph.D. thesis, The University of Sydney.

[58] Mark A. Webster, Benjamin J. Brown & Stephen D. Bartlett (2022): *The XP Stabiliser Formalism: a Generalisation of the Pauli Stabiliser Formalism with Arbitrary Phases*. *Quantum* 6, p. 815, doi:10.22331/q-2022-09-22-815. Available at `http://dx.doi.org/10.22331/q-2022-09-22-815`.

[59] Mark A Webster, Armanda O Quintavalle & Stephen D Bartlett (2023): *Transversal diagonal logical operators for stabiliser codes*. *New Journal of Physics* 25(10), p. 103018, doi:10.1088/1367-2630/acfc5f. Available at `http://dx.doi.org/10.1088/1367-2630/acfc5f`.

[60] John van de Wetering (2020): *ZX-calculus for the working quantum computer scientist*. *arXiv preprint arXiv:2012.13966*, doi:10.48550/arXiv.2012.13966.

[61] Lia Yeh, Jiaxin Huang, Aleks Kissinger, Sarah Meng Li & John van de Wetering (2026): *ZX Normal Forms for Stabiliser Codes (... or How to Graphically Grok Stabiliser Tableaus)*.

A Basics of the ZX-Calculus

Here, we present a few more basics of the ZX-calculus.

B Normal Form Reduction Algorithms

For the normal forms in the ZX-calculus, here we give their reduction algorithms and proof that they terminate.

B.1 Z-X and X-Z Generalised Parity Form for CSS codes

Firstly, here is the algorithm to reduce any phase-free ZX-calculus diagram to Z-X Generalised Parity Form (GPF). The X-Z GPNF form reduction algorithm follows by exchanging the roles of Z and X.

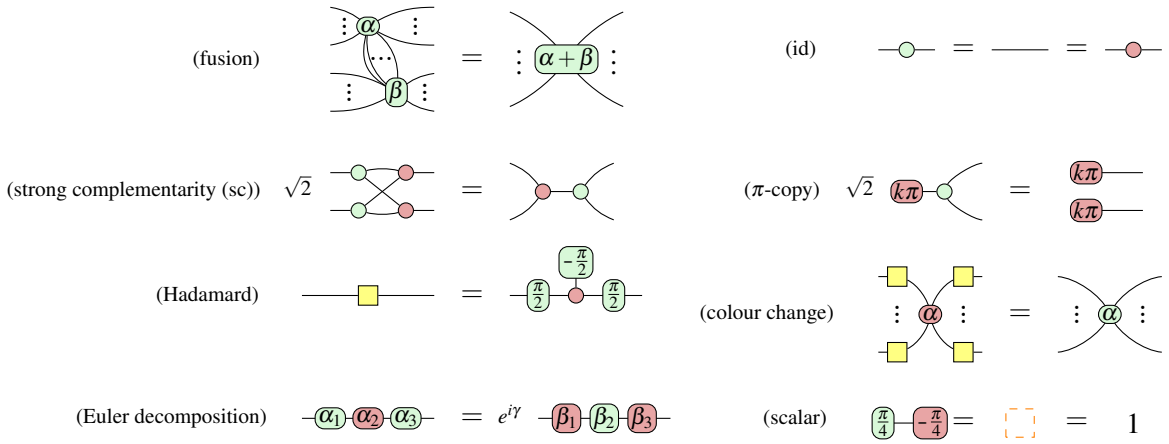


Figure 16: These eight equations suffice to derive all other equalities of linear maps on qubits [56]. $k \in \mathbb{Z}_2$. α_i, β_i and γ are real numbers satisfying the trigonometric relations derived in [14]. Each equation still holds when we interchange X and Z spiders. Whenever there are any two wires with ... between them, the rule holds when replacing this with any number of wires (i.e., 0 or greater).

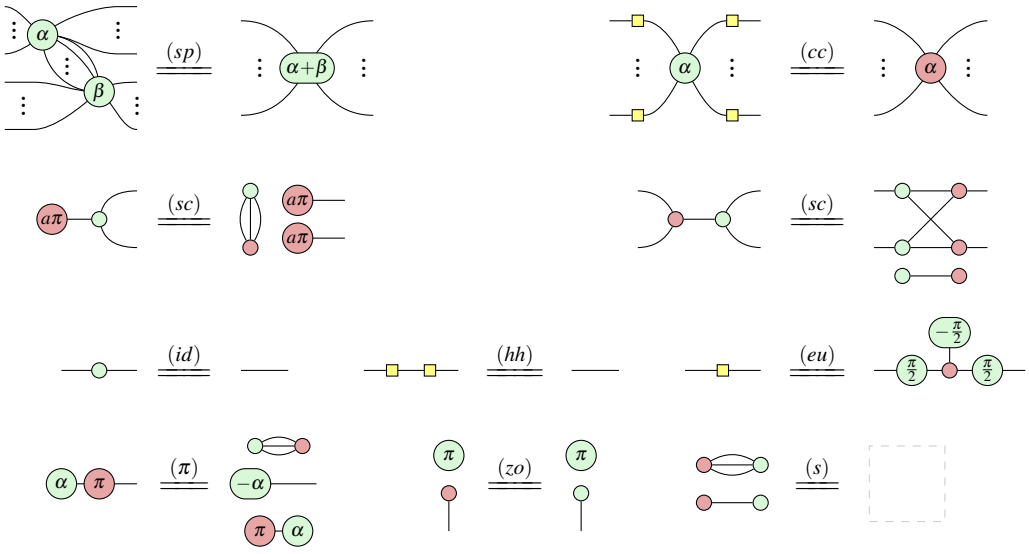


Figure 17: The complete and scalar-accurate set of rewrite rules for the Clifford ZX-calculus. $\alpha, \beta \in \{k\frac{\pi}{2}; k \in \mathbb{Z}_4\}$. Each equation still holds when we interchange X and Z spiders.

Algorithm 5: Reducing to GPF

Input: A phase-free ZX diagram

Output: A phase-free ZX diagram in GPF

1. Apply (fusion) as much as possible and remove 0-arity spiders by multiplying the overall scalar by 2.
2. Try to apply (sc) where
   - The Z spider is **not** an input and
   - The X spider is **not** an output.
3. If (sc) was applied at step 2, go to step 1, otherwise go to step 4.
4. Use (id) in reverse to ensure every input is connected to a Z-spider and every output to an X-spider.

Remark B.1. The above algorithm also applies to the scalar (zero input, zero output) case. Given any scalar phase-free ZX-calculus diagram, after performing all possible fusions, the resulting diagram must be bipartite. Put all the Z spiders on the left and all the X spiders on the right. Unfuse a one-legged Z spider from each Z spider, and a one-legged X spider from each X spider. Then excluding these new one-legged Z and X spiders, rewrite the remaining diagram to X-Z GPNF by applying algorithm 5. Finally, copy all the one-legged Z spiders through the X spider they are connected to, and all the one-legged X spiders through the Z spider they are connected to. The end result is scalars made of a one-legged Z spider connected to a one-legged X spider, one for each edge in the X-Z GPF.

Lemma B.1. Algorithm 5 terminates efficiently with a ZX-diagram in GPF.

Proof. Note that step 1 always removes spiders from the diagram. Step 2 will remove a pair of spiders, but it will introduce new Z-spiders on all the neighbours of X spiders and new X-spiders on all the neighbours of Z spiders. Since an X spider is not an output, the only possibility is that the new Z-spiders will be inputs or they will be adjacent to other Z-spiders. In the latter case, they will get removed by spider fusion when step 1 is repeated. Hence, step 2 removes a non-input Z-spider without introducing any new non-input Z-spiders.

This shows that the algorithm terminates after a number of iterations of steps 1–3 bounded by the number of non-output Z-spiders, where each of steps 1–4 takes polynomial time (in the number of spiders), so the whole algorithm terminates in polynomial time. The main loop in Algorithm 5 will only terminate once condition 3 of Definition 2.22 is satisfied, and since it applies step 1 just before exiting the loop, it will be in two-coloured form. Finally, step 4 ensures condition 1 is satisfied while preserving the other conditions. □

B.2 AP And Reduced AP Normal Form for stabiliser codes

Algorithm 6: Reducing stabiliser states to AP normal form

Input: A stabiliser state written as a Clifford diagram (PQS Prop 6.3.9)

Output: An AP normal form for the stabiliser state

1. Apply the graph-like-reduction algorithm to convert the Clifford diagram to a graph-like diagram.
2. Apply local complementation rule to remove all internal spiders with a phase of $\pm\frac{\pi}{2}$.
3. Apply pivot rule to remove all internal spiders that are connected to another internal spider.

Algorithm 7: Reducing Clifford ZX diagrams to graph-like diagrams (PQS prop 5.1.8)

Input: a Clifford ZX diagram

Output: a graph-like diagram

1. Convert all X-spiders to Z-spiders using the (cc) rule.
2. Cancel all pairs of adjacent Hadamards using the (hh) rule.
3. Fuse all spiders by applying the (sp) rule until it can no longer be applied.
4. Remove parallel Hadamard edges using Lemma 5.1.5.
5. Remove self-loops using Eq. (3.39), and remove Hadamard self-loops using Lemma 5.1.6.
6. Introduce Z-spiders and Hadamards using (id) and (hh) in reverse, in order to ensure every input and output is directly connected to a Z-spider and no Z-spiders are connected to multiple inputs/outputs.

Algorithm 8: Reducing graph-like diagrams to AP normal forms (PQS prop 5.3.2)

Input: A graph-like diagram

Output: A corresponding AP normal form

1. Apply local complementation rule to remove all internal spiders with a phase of $\pm\frac{\pi}{2}$.
2. Apply pivot rule to remove all internal spiders that are connected to another internal spider.

Note that the above two steps strictly reduce number of internal spiders, so the algorithm terminates with no more than m steps, where m is the number of internal spiders. After that, the resulting diagram is a graph-like diagram where all boundary spiders are connected to only one input or output, and all internal spiders carry only 0 and π phases and are connected only to boundary spiders. This is exactly an AP normal form.

Algorithm 9: Reducing AP normal forms to reduced AP normal forms (PQS prop 5.5.6)

Input: A AP normal form described by parity matrix A, bit string vector $\vec{b}$ and a phase function ϕ

Output: A reduced AP normal form

1. Bring A to RREF by Gaussian eliminating the pair $(A, \vec{b})$ and remove all zero rows.
2. For every internal spider i in the corresponding diagram, do the following:
   (a) Identify the boundary spider b_i that i is connected to, and does not connect to any other internal spider. Such b_i exists because it corresponds to a pivot in the RREF A.
   (b) If b_i has a phase of π, push such phase across i to other boundary spiders. If b_i has a phase of $\frac{\pi}{2}$, remove it using local complementation.
   (c) Remove Hadamard edges that connects b_j to .

Algorithm 10: Reducing stabiliser states to GSLC normal forms

Input: A stabiliser state written as a Clifford diagram

Output: A GSLC normal form for the stabiliser state

1. Apply Algorithm 6 to convert the Clifford diagram to AP normal form.
2. For each internal spider i_1, perform the following steps:
   (a) Identify one boundary spider that this internal spider is connected to.
   (b) Apply (hh) and (id) rule to the input/output wire of the boundary spide to convert this boundary spider to another internal spider i_2.
   (c) If i_2 carries a phase of 0 or π, remove i_1 and i_2 together using pivot rule. Othersie, remove i_1 and i_2 sequentially, with two applications of local complementation.
3. remove extra Hadamards appearing on the input/output wire.