

Canonical Forms and Equivalence Classes of QECC's in ZX Calculus

Andrey Boris Khesin¹ and Alexander M. Li²

¹*Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA*

²*C. Leon King High School*

(Dated: February 4, 2024)

Quantum error-correcting codes (QECC's) are needed to combat the inherent noise affecting quantum processes. Using ZX calculus, we represent QECC's in a form called a ZX diagram, consisting of a graph made up of nodes and edges. In this paper, we present canonical forms for the ZX diagrams of the toric codes and certain surface codes. We derive these forms by rewriting them using the bialgebra rule, which removes extra internal nodes and was implemented through *Quantomatic*, and edge local complementation rule, which exchanges the colors of two nodes. Next, we tabulate the equivalence classes, including properties such as their size and the presence (or lack) of bipartite forms, of generic ZX diagrams of QECC's. This work expands on previous works in exploring the canonical forms of QECC's in their ZX diagram representations.

Keywords: canonical form, Clifford codes, error-correcting codes, graph states, quantum compilers, quantum computing, quantum mechanical system, surface code, toric code, ZX calculus, ZX normal form

1. INTRODUCTION

The work done in the past half century on quantum computing have brought large-scale quantum computers closer to reality. Today, quantum computers can employ a low number (up to a few hundred) of qubits, in the form of photons and nuclear spins [1], but they have been used mainly for experiments. These quantum computers differ from classical computers and classical supercomputers by employing the use of qubits rather than bits. The properties of quantum mechanics inherent in qubits, including superposition and entanglement, allow quantum computers to simulate quantum systems, which can make certain calculations much more efficient than classical computers [2].

However, as with classical information processing systems, quantum information processing systems also face noise that can disrupt information transmission between a sender and receiver. One of the principal challenges in quantum computing is to account for this noise, due to the fragility of quantum bits [3]. To this end, quantum error-correcting codes are used to transmit quantum information successfully in the presence of noise [2]. While classical computers can copy bits, quantum mechanics does not allow the cloning of unknown qubits, and the measurement of a qubit eliminates the information available in the qubit. As such, the construction of suitable quantum error-correcting codes presents new challenges when compared to the construction of classical error-correcting codes. For decades, a quantum error-correcting code against general errors seemed impossible, until the Shor code [4] was first

published in 1995 and Steane code [5] in 1996. Other examples of quantum error-correcting codes are the five qubit code [6] and the toric code [7].

A number of approaches have been created to represent the components of quantum error-correcting codes. The *stabilizer* formalism is a method that expresses quantum error-correcting codes in terms of stabilizers, operators that, when applied to certain stabilizer states, preserve the state [8]. This approach borrows ideas from group theory to represent the whole class of stabilizers with a finite number of generators. To make the idea of quantum error-correcting codes visual, recent advances have made progress on the topic of presenting *graph states* [9, 10].

Following the work on graph states, work has been done on representing Clifford codes using ZX calculus [11?–13]. The properties of ZX calculus that allow it to replace the stabilizer tableau formalism (a tabulated form of the generators of the stabilizers) are its universality (it can express every quantum operation), soundness (tableaus can derive equivalence of ZX calculus diagrams), and completeness (ZX calculus diagrams can derive equivalence of tableaus) [11, 13]. This graphical language has had various applications in quantum information [14–17] and quantum computation problems [18, 19].

Expressing quantum Clifford circuits as graphs and finding canonical forms for equivalent graphs has had recent advances in the past few years. The Hu-Khesin (HK) form from [10] provides a canonical form for quantum Clifford states. In the context of quantum encoders, this is equivalent to having no inputs and only outputs. Then, the Khesin-Lu-Shor

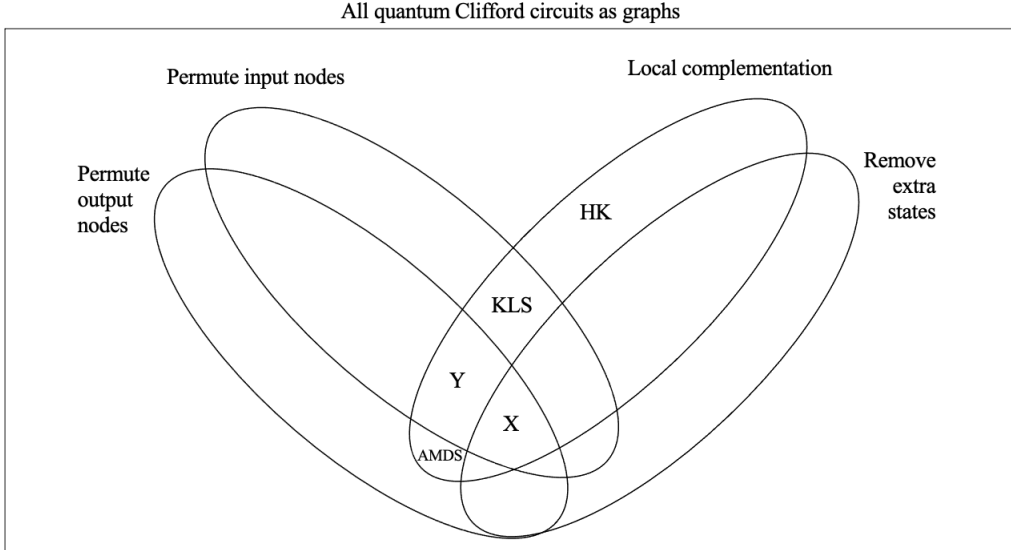


FIG. 1: Shown is a Venn diagram classifying quantum Clifford circuits as graphs, with the universal set of graphs surrounding the diagram. Khesin-Lu-Shor (KLS) forms are determined based on equivalence through permuting inputs and local complementation. The Hu-Khesin (HK) form is in the category of only local complementation because the HK states do not have inputs to permute. Adcock, Morley-Short, Dahlberg, and Silverstone (AMDS) considered equivalence of graph states under local complementations and the effects of relabelling the nodes. The graphical forms in X have the most general definition of equivalence, allowing all four operations. On the other hand, the forms in Y do not consider the removal of extra states in the output nodes, and they will be explored in Sections 4 to 8.

(KLS) form from [20] built on the HK form, providing a canonical form for Clifford encoders. The KLS paper gives a detailed process of transforming stabilizer tableaux into ZX calculus, then performing operations that preserve equivalence to find a canonical form. Section 3 of this work expands on the results from KLS and focuses more topologically on the general shape the vertices and edges that the ZX diagram forms. Specifically, we analyze selected surface and toric codes to see how we can simplify the diagram into an intuitive canonical form. The second part similarly focuses on the general properties of the ZX graph, taking into account the bipartite portion of the graph between the input and output nodes while also allowing the permutation of output nodes to stay in the same equivalence class.

See Figure 1 for a summary of the work done in quantum Clifford encoders’ graphical representations. The set of all quantum Clifford circuits as graphs is split into sectors depending on whether we consider the operation as giving equivalent Clifford codes. For example, the sector labeled KLS is positioned at the intersection of “permute inputs” and “local complementation.” Therefore, the KLS canonical form relied on the encoders staying in

the same equivalence class upon these two operations while the encoders changed equivalence classes when the output nodes are permuted or extra states (nodes unconnected to input nodes) are removed from the output nodes. The second part of this paper considers the sector labeled Y, as output nodes will be allowed to permute.

In this paper, Section 2 contains key definitions and background on ZX calculus and Clifford encoders. Section 3 contains our work on Calderbank-Shor-Steane (CSS) codes, specifically surface and toric codes, in making an intuitive canonical form for select encoders. This builds on recent work from Kissinger [21, 22] that introduced the normal form of CSS codes, which are “explainable” in that it is efficient to determine the stabilizers from the ZX normal form. For an $(n - 2k)$ -to- n encoder with k X-checks and k Z-checks, the resulting normal form will consist of $(n - 2k) + n + k = 2n - k$ nodes, which are the input nodes, output nodes, and internal nodes representing the X-checks (for the ZX normal form) or the Z-checks (for the XZ normal form). Section 3 focuses on presenting a canonical form of the codes that eliminates these internal nodes while keeping

the diagram for the encoder as intuitive and elegant as possible.

Section 4 provides another definition of equivalence, permitting outputs to be permuted as a valid operation among equivalent graphs. The reason this definition of equivalence is also considered is that changing the order of the outputs does not change the “amount” of entanglement that the encoder puts the input qubits through. Section 5 expands on this definition by omitting parts of the set of quantum encoders that will not be necessary for the remainder of the paper. Sections 6 and 7 provide the work we have done towards identifying equivalence classes and finding representative forms. Section 6 explains a method of representing encoders as integers so as to sort them into equivalence classes. Then, it goes on to revealing more information about these equivalence classes, including sizes and the presence or lack of bipartite forms. Section 7 expands on the equivalence classes containing bipartite forms.

2. BACKGROUND

In this section, we define key terms and background on error-correcting codes and ZX calculus.

First, we define the following matrices.

Definition 2.1. The *Pauli matrices* are

$$I \equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad X \equiv \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$Y \equiv \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z \equiv \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

The Pauli matrices numerically represent quantum gates that can act on qubits and alter their state. Then, the *Pauli operators on n qubits* are n -fold tensor products of Pauli matrices, multiplied by a factor of the form i^k where $k \in \{0, 1, 2, 3\}$ and $i = \sqrt{-1}$.

The Pauli operators are all equal to their conjugate transposes, making them Hermitian and unitary. The Pauli operators form a group, called the Pauli group.

Pauli operators can act on states in multi-qubit systems. For example, in a three qubit system, the tensor product $Z \otimes Z \otimes I$ will make Z act on the first qubit, Z act on the second qubit, and I act on the third qubit. The notation for the tensor product can be simplified to $Z_1 Z_2$, with the subscripts showing which qubit the operators are acting on. We

can similarly denote other tensor products of Pauli operators.

Other quantum gates that are useful are the Hadamard, controlled-NOT, phase, and $\pi/8$ gates, denoted as H , CNOT, S , and T , respectively:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad \text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix},$$

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}, \quad T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}.$$

These operations have the property of universality, the ability to approximate any operator to arbitrary accuracy [2].

Clifford codes can be represented as a group of generators of stabilizers. These stabilizers are Pauli operators on n qubits and listing out the generators determines the whole encoder. In this paper, the encoders take $n - k$ inputs, which are logical qubits, and give n outputs, which are physical qubits. The generators are chosen to be independent, so each degree of freedom going from inputs to outputs requires an additional generator for the stabilizers, implying there are k generators for an $(n - k)$ -to- n code.

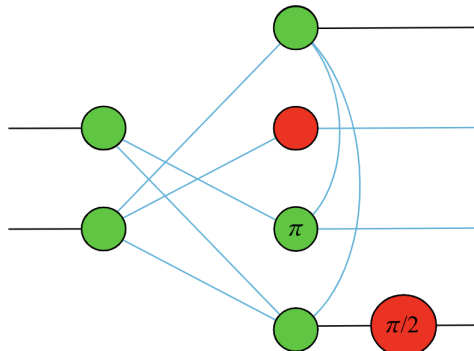


FIG. 2: Example of an encoder in ZX calculus. The incoming edges from the left side are input edges (sending quantum information in) and the outgoing edges on the right side are output edges (sending encoded quantum information out). Note the local operations applied on the output qubits, with blue edges representing Hadamarded edges.

ZX calculus makes the representation of Clifford codes graphical, and we use the conventions as described in [13, 20]. The green \circ and red nodes \bullet represent quantum processes, which could be qubits, gates, or measurements. Each node has a phase, with empty nodes representing a phase of 0. By connecting edges between nodes, the quantum processes can be linked together, as in circuits. It is possible

to convert quickly between quantum circuits and ZX diagrams, with a specific example given in Appendix C. Hadamard gates, \square , can be placed on edges between nodes or free edges. In this work, Hadamarded edges will be represented with blue edges while un-Hadamarded edges will be represented with black edges. Two Hadamard gates on one edge can be reduced to the identity, since $HH = I$.

An example of a Clifford code expressed in ZX calculus is shown in Figure 2. For an $(n - k)$ -to- n encoder, the $n - k$ input nodes are connected with edges to the n output nodes, and the output nodes share edges amongst each other. There are local operations on the output edges, as shown by the blue free edges and red $\pi/2$ gate. There are internal edges in the graph between the input and output nodes and amongst the output nodes. Two qubits of information are encoded into four qubits by applying this encoder.

The canonical forms described in Section 3 are based on the KLS canonical form [20], which consists of four rules that can be efficiently checked on a given ZX diagram.

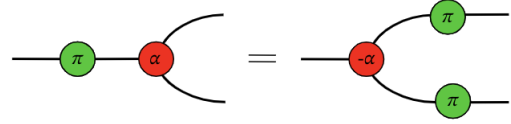
Theorem 2.1. (*Khesin-Lu-Shor*) *There is a canonical form for the ZX diagrams of an encoder, where the ZX diagrams in the same equivalence class have identical stabilizers. Suppose the output nodes are numbered using the integers 1 to n , inclusive. The canonical form satisfies the following four rules:*

1. *Edge rule: All internal edges have Hadamards, and there is exactly one Z node per free edge.*
2. *Hadamard rule: Output nodes with Hadamards on their free edge cannot share an edge with a lower-numbered output node or with an input node.*
3. *RREF rule: The adjacency matrix representing the edges between input nodes and output nodes is in row-reduced echelon form.*
4. *Clifford rule: In the RREF form, the pivot columns of the input to output adjacency matrix correspond to output nodes. There are no local Clifford operations on the pivot or input nodes, or their free edges. There are also no input-input edges or pivot-pivot edges.*

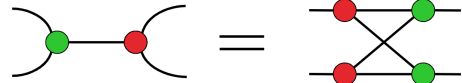
A ZX diagram satisfying these four rules is the unique KLS canonical form for the equivalence class where all the ZX diagrams have the same stabilizers. Also, a given ZX diagram can be efficiently transformed to its KLS canonical form using a series of operations. In our derivation of the toric code and

surface code ZX diagrams, we will be using the following rules in ZX calculus:

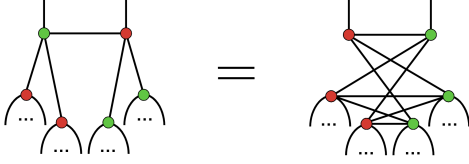
1. **Merging/un-merging rule:** Two green (or red) nodes with phases α and β and connected by edges with no Hadamards may be combined into a node with phase $\alpha + \beta$. The resulting node has all external edges of the two original nodes. A node may also be un-merged, and the external edges connected to each of the two resulting nodes may be chosen at will.
2. **π -copy rule:** A green (or red) node of phase π slides through and copies onto all the other edges of a red (or green) node. The red (or green) node has its phase negated. Shown below is an example with a green π node.



3. **Loop rule:** Self-loops on a node can be removed. If the self-loop has a Hadamard, then removing the loop adds a phase of π to the node.
4. **Hopf rule:** If a red node and green node share two non-Hadamarded edges, both edges can be removed.
5. **Bi-algebra rule:** By acting on an edge between a red and a green node, each external edge gets one node, and a complete bipartite graph is formed between these new nodes. An example is shown below. There may be one or more (rather than two) edges coming in from the left side of the graph, and there may be one or more edges exiting on the right side of the graph.



6. **Hadamard-sliding rule:** This rule allows the colors of two adjacent vertices to be swapped while switching neighbors and toggling the edges between the neighbors of the two vertices. See Appendix B for more about this rule.



The KLS canonical forms may be transformed into quantum circuits by the following steps. Suppose we are considering an $(n - k)$ -to- n encoder.

1. Start with $n - k$ open wires representing the inputs of the circuit.
2. Add a $|0\rangle$ state for each of the k non-pivot output nodes.
3. Apply an H gate to all n wires.
4. Apply a CX gate between the wires corresponding to the edges between inputs and non-pivot outputs. The input node is the target qubit, and the output node is the controlled qubit.
5. Apply a CZ gate between the wires corresponding to the edges between only outputs.
6. Apply the local operations attached to the outputs.

This procedure works by building up the encoder's quantum circuit representation in layers, starting from the input qubits, which correspond to the ZX diagram input nodes, adding auxiliary qubits that encode the information from the input qubits, and connecting the wires using the appropriate gates. More details on the proof of this procedure can be found in Appendix C.

The neighborhood $N(v)$ of a vertex v in a graph $G = (V, E)$ is the set of all vertices in V adjacent to v . An operation commonly used to transform equivalent ZX diagrams between each other by transforming the neighbors of a vertex is defined below.

Definition 2.2. Consider a simple graph $G = (V, E)$, where V is the set of vertices and E is the set of undirected edges between vertices. Take a vertex $v \in V$. By performing a *local complementation* about vertex v , all edges connecting two vertices in $N(v)$ are toggled. That is, if the edge existed before the local complementation, it is removed; if it did not exist before, it is added.

Suppose a graph G has vertices v_1 and v_2 connected by an edge. Local complementation about a vertex from Definition 2.2 can be applied on vertices v_1, v_2 , then v_1 . This new operation is defined below.

Definition 2.3. Consider a simple graph $G = (V, E)$, where V is the set of vertices and E is the set of undirected edges between vertices. Take an edge $v_1v_2 \in E$, where v_1 and v_2 are distinct vertices in V . By performing an *edge local complementation* about edge v_1v_2 , the neighbors of v_1 and v_2 are swapped, and any connections from $N(v_1)$ to $N(v_2)$ are toggled.

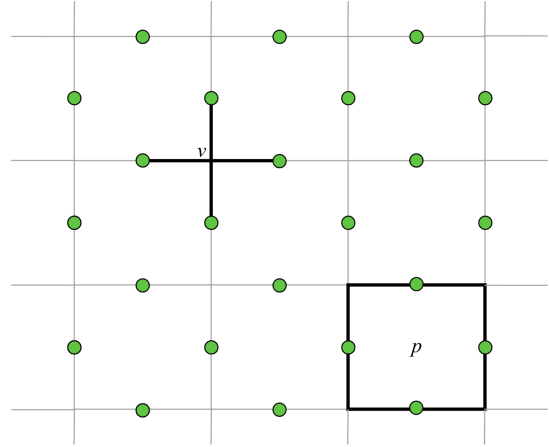


FIG. 3: A section of the torus after placing it onto a 2-dimensional plane. The stabilizers corresponding to the vertices (v is an example) have X gates on the nodes immediately surrounding the vertex. The stabilizers corresponding to the plaquettes (p is an example) have Z gates on the nodes immediately surrounding the plaquette. All nodes have a default green color.

3. SURFACE AND TORIC CODES

Calderbank-Shor-Steane (CSS) codes are an important class of Clifford codes which are constructed from two classical linear codes [2]. These quantum error-correcting codes used classical computing ideas and applied them to quantum computing.

To find the canonical forms of CSS codes based on the ZX normal forms, we first consider toric codes, a specific class of surface codes as introduced in [23], and certain surface codes, as explained in [22]. The canonical forms presented here have 0 internal nodes, so that each node corresponds to an input or output edge.

We begin with a definition of toric codes.

Definition 3.1. A *toric code* is a quantum error-correcting code that can be represented on a three-

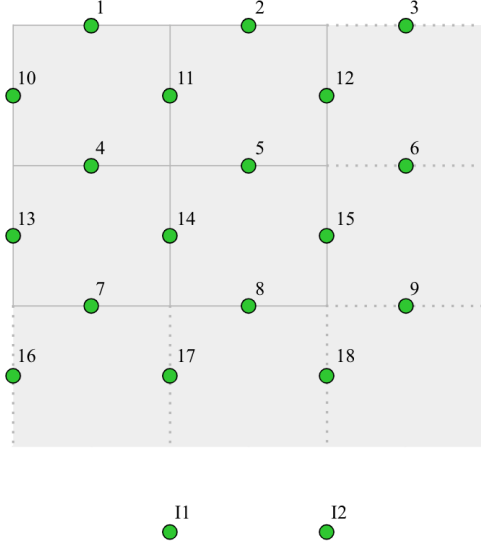


FIG. 4: The nodes in the 3-by-3 toric code. The two input nodes are labelled I1 and I2, and the output nodes are labelled with integers 1 through 18. Note that these labels are not the phases of the nodes. In the ZX diagram, each node is a qubit.

dimensional torus \mathcal{T} . For an $m \times n$ toric code, \mathcal{T} is wrapped by $m - 1$ circles parallel to the plane of the major circle and $n - 1$ circles perpendicular to the plane of the major circle.

A node is placed at the midpoint of each of the $2mn$ edges on \mathcal{T} . The stabilizers are defined as follows.

The four nodes surrounding each of the mn four-sided faces form a Z -check (stabilizer with only Z 's and I 's) consisting of Z 's on these four nodes and I 's on all other nodes.

The four nodes surrounding each of the mn intersections form an X -check (stabilizer with only X 's and I 's) consisting of X 's on these four nodes and I 's on all other nodes.

We now turn to the 3-by-3 toric code. The 18 nodes corresponding to the output edges and the 2 nodes corresponding to the input nodes are shown in Figure 4. The stabilizers of the 3-by-3 toric code are analogous to those shown in Figure 3. Note that, in Figure 4, the Z -check corresponding to the plaquette with nodes 3, 6, and 12 would also have node 10, which wraps around the torus. Similarly, nodes 7, 16, 17, and 1 are part of the same Z -check.

We now determine the structure of the 3-by-3 toric code's ZX diagram by deducing the placements of Hadamards on output edges and using the stabilizers to force the connections between output nodes.

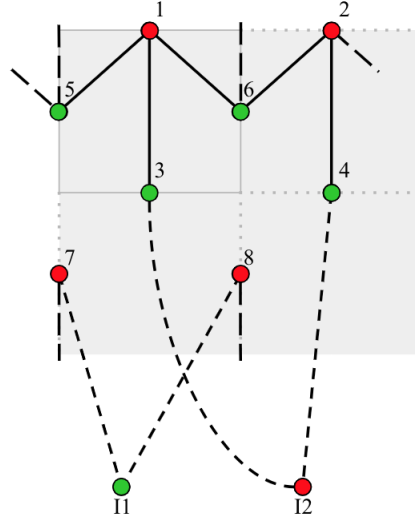


FIG. 5: The 2-by-2 toric code in ZX calculus. Longer-dashed edges represent edges that wrap around the torus. For example, the vertical dashed edge coming from node 5 meets node 7 and the dashed edge from node 2 meets node 5. The shorter-dashed edges are input-output edges. The free input and output edges are not shown.

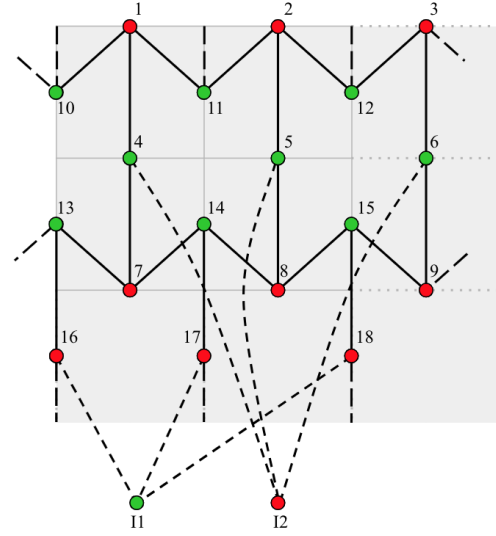
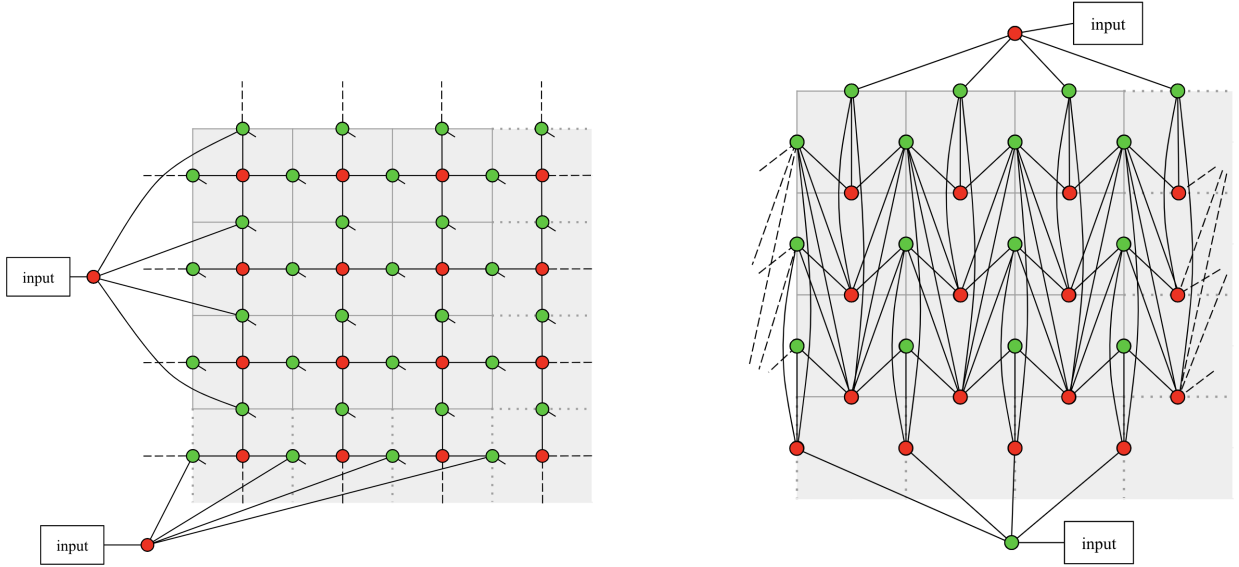


FIG. 6: The 3-by-3 toric code in ZX calculus. Longer-dashed edges represent edges that wrap around the torus. For example, the vertical dashed edge coming from node 10 meets node 16 and the dashed edge from node 3 meets node 10. The shorter-dashed edges are input-output edges. The free input and output edges are not shown.



(a) The ZX normal form [22]. The output nodes, which are all the green nodes, are shown with free edges protruding from them. The red nodes in the toric grid are internal nodes.

(b) The canonical form of the 4-by-4 toric code. There is a symmetry between the red and green nodes. Though this is not in KLS form, it is easily converted into the KLS form.

FIG. 7: The 4-by-4 toric code, shown in two equivalent ZX diagrams. Note that, while the ZX normal form is local in both the vertical and horizontal directions, it has $4^2 = 16$ more nodes than the canonical form in (b).

By similar methods, we can also present the ZX calculus form of the 2-by-2 toric code. The diagram is shown in Figure 5. This has an arrow-like structure among the output-output edges, as seen by nodes 1, 3, 5, and 6, as well as nodes 2, 4, 5, and 6.

The final 3-by-3 toric code is shown in Figure 6, with its full derivation given in Appendix A. Note that, by wrapping this pattern around a torus, it would be horizontally periodic. The edges among vertices 1, 4, 10, and 11 form an upward-arrow-like figure. Similarly, nodes 2, 5, 11, and 12 form this figure and, on a torus, nodes 3, 6, 10, and 12 for this figure as well. By the simplicity of this diagram, it is relatively easy to read off the stabilizers by noting how the π -copy rule causes certain stabilizers to exist. Not much more work has to be done to convert this to the KLS form of the encoder.

The KLS form specifies labeling the vertices, which would bias the resulting canonical form into a strict ordering of vertices. On the other hand, the symmetric forms shown in Figure 5 and Figure 6 ignore a strict labeling of vertices; the numbers inside the nodes may be removed at will. Instead, these highly symmetrized versions of the KLS form clearly show the structures formed by the nodes and edges

to satisfy the stabilizers.

For larger toric codes and the surface codes, we use the ZX calculus software *Quantomatic* [24] to simplify the known ZX normal form [22] of a code into its canonical form. In our algorithm, the main focus is on performing the bialgebra rule on internal nodes, so that, after running the first part of the algorithm, all internal nodes will be removed from the diagram, leaving only input and output nodes. Then, the Hadamard sliding rule, Eq. (10) from [10], will provide the operation that can repeatedly move Hadamards until the encoder diagram is symmetric.

The procedure we follow may be written as the following algorithm.

1. Use basic simplifications, by merging red nodes, merging green nodes, applying the red-copy rule, applying the green-copy rule, applying the Hopf rule, removing scalars, removing loops, or combining two Hadamards into the identity [13].
2. Apply one iteration of the bialgebra rule (any variation) that removes an internal node. Then, apply step 1 again.

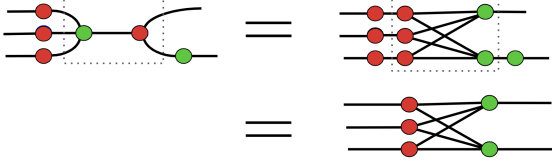


FIG. 8: An example of the bialgebra rule removing an internal node. The bialgebra rule is applied to the edge within the dotted box.

3. Apply step 2 until all internal nodes are removed.
4. Apply the Hadamard-sliding rule until the colors of the nodes are (mostly) alternating. (Note: In the toric code, it turns out that it is impossible for the colors to alternate every row, but the main section of nodes have alternating colors every row.)

The reason step 2 works is that internal nodes are absorbed into neighboring nodes in the bialgebra rule. An example is shown in Figure 8.

In the diagrams for the 4-by-4 toric code (Figure 7) and the 5-by-5 surface code (Figure 9), instead of depicting all nodes as green and drawing dashes within the nodes to indicate Hadamards on free edges, we will instead use the standard red and green nodes in ZX calculus.

To this end, we use the above algorithm to derive the general ZX diagram for the m -by- n toric code.

First, we present the canonical form of the 4-by-4 toric code, which was derived from the ZX normal form. These are both shown in Figure 7. As can be seen in the diagrams, the number of output nodes is reduced by a factor of 2, and the diagram in Figure 7(b) retains a high degree of symmetry. When moving horizontally, it can be seen that there is a periodic pattern of nodes, with one column having 3 green nodes and 1 red node and the next having 3 red nodes and 1 green node. Also, the edges between the columns of nodes are local in one direction, as their length does not scale with the horizontal dimension of the toric code.

Using the algorithm on larger dimension m -by- n toric codes shows that they have the same general structure as that of Figure 7(b). To construct it geometrically, first place a red input node at the top of the diagram and a green input node at the bottom of the diagram.

Then, on the unfolded toric grid, the first and second layers (out of $2n$ layers) of m nodes each are designated as green. Then, the colors alternate between red and green until the very bottom two layers of the

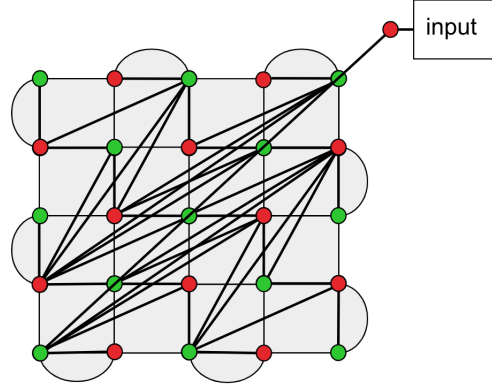


FIG. 9: The 5-by-5 surface code in its ZX canonical form. Note its resemblance with the toric code diagram when tilted by 45° . The red input node connects to all 5 green nodes along the bottom-left to top-right diagonal.

output nodes, which are both red. This is reflected in the 4-by-4 example.

To draw out the edges, each of the top layer's green nodes has one edge to each of the red nodes within its column. Furthermore, the second layer's green nodes have edges connecting them to each of the red nodes in the neighboring columns, as well as one edge connecting it to the bottommost red node in the same column. The next layer of green nodes (the fourth layer of m nodes from the top) have edges to all the red nodes in the neighboring columns that are in rows strictly below it, as well as one edge to the bottommost red node in the same column. This pattern follows for the other green layers.

Since there is a symmetry between the green and red nodes, the same arrow-shaped patterns can be seen extending upwards from layers of red nodes.

We can also extend our results to the rotated surface codes, shown in Eq. (12) from [22].

In Figure 9, we show the result of simplifying the 5-by-5 surface code. If it is rotated by 45° counterclockwise, it heavily resembles the patterns of edges and colors seen in the general toric code. The neighboring diagonals (from bottom-left to top-right) of nodes of different colors connect in arrow-shaped patterns, just as in the toric code.

In general, the $(2k + 1)$ -by- $(2k + 1)$ surface code can be made to have a similar structure as shown in Figure 9.

More work done on CSS states (CSS codes with 0 input nodes) can be found in Appendix D.

Transforming the adjacency matrix:

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

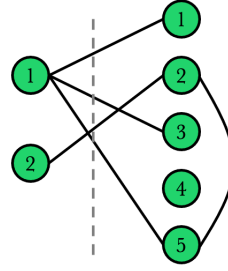


FIG. 10: In the adjacency matrix, the first row corresponds to the first input, the second row corresponds to the second input, and so on. After the inputs, the following row corresponds to the first output, and the other outputs follow. In the ZX diagram on the right, all edges shown are internal edges. For clarity, the Hadamard gates are not shown.

4. ANOTHER DEFINITION OF EQUIVALENCE

Previous works have examined the equivalence classes of graphs under local complementation [25–27]. In Clifford codes, the presence of designated input and output vertices makes the definition of equivalence more exotic.

In the following sections, we consider only the ZX diagrams for Clifford codes that have no local operations on the free output edges.

Definition 4.1. Two ZX diagrams are *locally equivalent* if and only if one can be converted to the other through a sequence of local complementations and local operations on the free output edges.

Definition 4.2. Two Clifford codes \mathcal{C}_1 and \mathcal{C}_2 are *equivalent* if and only if the ZX diagrams are locally equivalent or locally equivalent after some permutation of the output nodes and/or applications of any unitary operators on the inputs.

We now list five different operations which keep encoder graphs equivalent.

Conjecture 4.1. The ZX diagrams for two Clifford codes \mathcal{C}_1 and \mathcal{C}_2 are *equivalent* if and only if the diagram for one of the codes can be reached by the other after a sequence of operations consisting of the following operations:

1. Local complementing about any vertex of the graph.
2. Permuting the output vertices.
3. Permuting the input vertices.
4. Performing linear operations on the adjacency matrix of input to output edges.
5. Removing an input-input edge.

6. Applying local operations on the output edges.

All of the operations in Conjecture 4.1 are reversible, so, if code \mathcal{C}_1 can be made equivalent to \mathcal{C}_2 , the reverse is also true.

Operation 1, local complementation as in Definition 2.2, is included to account for equivalence of encoder graphs based on their entanglement [25].

Two encoder diagrams should also be equivalent if the information they produce can be ordered differently to become the same. In this way, operations 2 and 3 reflect this, since connections among the vertices of the graph remain the same and these operations only change the order in which the information is inputted or outputted.

Operation 4 consists of adding rows of the adjacency matrix between input and output vertices in modulo 2. By [20], after linear operations on the adjacency matrix, the following property remains preserved across all stabilizers and input nodes: there is always an even number of connections between a given input node and output nodes that allow the π -copy rule for an X or Y gate of a given stabilizer. Since the parity of these connections remains the same, all the stabilizers will continue to stabilize the encoder.

Operation 5 takes away a unitary operation from the input vertices, which is allowed by Definition 4.2. Lastly, operation 6 preserves equivalence since all local operations can be removed by multiplying by their corresponding conjugate, which is allowed by Definition 4.1.

Note that this definition of equivalence does not allow two encoders to be in the same equivalence class if they only differ by an extra output (which is not connected to anything else). That is, if the two encoders differ by a quantum state, this definition of

equivalence marks them as different. Therefore, this implies we focus on section Y of Figure 1, instead of section X.

As an example of these extra outputs/states, see Figure 10. The output vertex labeled 4 is not connected to any input or output. It does not provide any more encoding of information from the inputs than if it was not present. For this reason, section X of Figure 1 is more useful for practical purposes.

5. SIMPLIFICATIONS

There are some simplifications that can be made using the above operations so we consider only encoder graphs that could possibly be non-equivalent.

Operations 3 and 4 of Conjecture 4.1 allow the input-to-output portion of the encoder diagram to be expressed in row-reduced echelon form (RREF). All encoder diagrams considered from here on are expressed in RREF form, as in the RREF rule from [20]. In the RREF form, each non-zero row has a leftmost non-zero entry of 1. Each column with these 1's is then a pivot column. Since the columns of the input-output edge adjacency matrix correspond to output nodes, each pivot column corresponds to a pivot node. Therefore, the input nodes have corresponding output nodes that represent the pivots of the RREF. These particular output nodes are called *pivot* nodes.

Continuing from the RREF form of the encoder graph from the above paragraph, operation 2 from Conjecture 4.1 can be used to move the pivot nodes to be at the front of the line of the sequence of output nodes. In this way, the first output node can be made into the pivot node corresponding to the first input node, the second output node can be made into the pivot node corresponding to the second input node, and so on. Thus, these $n - k$ pivot nodes are fixed among the top of the output nodes. For brevity, the other k non-pivot output nodes are called *free* output nodes.

In Conjecture 4.1, no operation was included that affected local Clifford gates at the nodes. Therefore, this definition of equivalence neglects the presence of phase changing gates at vertices of the encoder's graph. This is because local Clifford gates change the qubits using a unitary operation but does not contribute to changes in entanglement of the qubits in any way. Therefore, for our purposes, we remove all local Clifford gates present at the nodes of the ZX diagram for the encoder.

Furthermore, to simplify the diagrams we draw, the incoming and outgoing edges (i.e. free edges from

[20]) will be omitted. They are implied, since the diagrams for $(n - k)$ -to- n encoders will be drawn with $n - k$ input nodes on the left side and n output nodes on the right side. Furthermore, for the internal edges that are included in the diagram, all edges have Hadamard gates as detailed in the Edge Rule from [20], but these gates will be omitted in our diagrams.

A further simplification, carried over from [20], is that graphs with pivot-pivot edges are omitted, since they can always be transformed into a graph without pivot-pivot edges using a sequence of local complementations.

As an example of the simplifications on the diagrams, as well as how the adjacency matrices transform into encoders, see Figure 10.

6. TABULATIONS FROM CODE

After considering the simplifications to the encoders from Section 5, the encoders were sorted into equivalence classes. To do this, we used the *disjoint set algorithm* to split encoders into equivalence classes based on whether an operation from Conjecture 4.1 caused one encoder to change into another.

We will call the non-pivot output nodes *free* nodes. Each encoder graph is converted into an integer based on the *variable* edges present in the graph, which are the input-free edges, pivot-free edges, and free-free edges. Note that the input-pivot, input-input, and pivot-pivot edges are fixed, so these are not included among the variable edges.

The variable edges' values in the adjacency matrix are made into a single integer using a binary representation. Note that this adjacency matrix includes all vertices, so it is a $(2n - k) \times (2n - k)$ matrix.

For example, in the 2-to-5 codes, the 7×7 adjacency matrix would look like the following:

$$\begin{pmatrix} 0 & 0 & 1 & 0 & a_{14} & a_{13} & a_{12} \\ 0 & 0 & 0 & 1 & a_{11} & a_{10} & a_9 \\ 1 & 0 & 0 & 0 & a_8 & a_7 & a_6 \\ 0 & 1 & 0 & 0 & a_5 & a_4 & a_3 \\ a_{14} & a_{11} & a_8 & a_5 & 0 & a_2 & a_1 \\ a_{13} & a_{10} & a_7 & a_4 & a_2 & 0 & a_0 \\ a_{12} & a_9 & a_6 & a_3 & a_1 & a_0 & 0 \end{pmatrix}$$

The top-left 4×4 submatrix reflects the fixed input-pivot edges, as well as the lack of input-input edges and pivot-pivot edges. We place a_0 near the bottom-right corner and fill in the rows above from right to left.

After converting the ZX diagrams into integers,

$n = 2$	$n = 3$	$n = 4$	$n = 5$	$n = 6$	
1	3	11	40	185	

Rep:			
Size:	1	6	9

(a) Number of equivalence classes for 2-to- n codes. (b) 2-to-3 codes equivalence classes and sizes.

Rep:					
Size:	1	1	6	12	18

18	36	42	45	99	234

(c) 2-to-4 codes equivalence classes and sizes.

1	3	4	6	18	18	18	27

27	54	54	60	63	90	108	108

108	126	135	144	168	297	378	396

414	459	486	540	702	972	1080	1080

1152	1188	1620	2268	2484	4896	5184	5832

(d) 2-to-5 codes equivalence classes showing the size of the class underneath a representative.

FIG. 11: (a) shows the number of equivalence classes for 2-to- n encoder graphs. (b-d) show an element of the equivalence classes to denote the representative of the class and gives the size of the class.

we use the *disjoint set algorithm*, which is useful for separating the whole set of possible encoder graphs

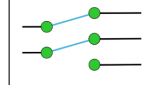
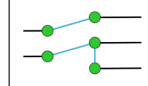
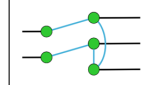
Rep.	#
	1
	2
	1

FIG. 12: The 2-to-3 encoders classes with bipartite forms. A representative of the class and the number of such bipartite forms in the class is shown.

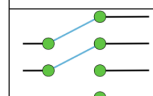
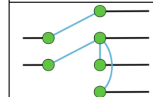
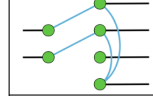
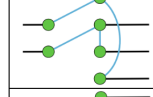
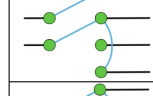

Rep.	#
	1
	2
	2
	2
	4
	5

FIG. 13: The 2-to-4 encoders classes with bipartite forms. A representative of the class and the number of such bipartite forms in the class is shown.

into equivalence classes.

Our code takes an integer representation, say n , of an encoder graph, performs one operation from Conjecture 4.1 on the encoder graph, then merges the disjoint sets of n and the integer representing the resulting encoder graph. All possible operations are applied, and the resulting values are merged with n 's disjoint set.

When a local complementation is performed on a free output, it is possible that an input-pivot edge is removed. Furthermore, some input-input edges could be added. To fix this, we first employ op-

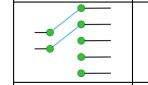
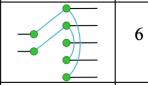
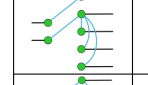

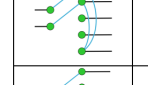

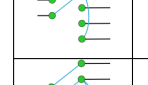

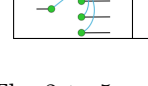
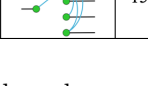
Rep.	#	Rep.	#
	1		6
	2		6
	3		7
	6		12
	6		15

FIG. 14: The 2-to-5 encoders classes with bipartite forms. A representative of the class and the number of such bipartite forms in the class is shown.

eration 5 from Conjecture 4.1 to set all input-input edges to 0. Then, operations 3 and 4 are used to turn the submatrix representing the input-to-output adjacency matrix into RREF form. Operation 2 is used to put the pivots back into their fixed positions, so they once again correspond to their input vertices.

By looping the code to run through all integers in the set of representatives of the possible encoder graphs, all the encoder graphs are grouped into an equivalence class.

The results of the code are shown in Figure 11. As shown in Figure 11(a), the number of equivalence classes increases quickly as the number of output vertices increases. Furthermore, in Figure 11(b-d), the equivalence classes show a variety of sizes, with many of the sizes in (c-d) divisible by 9.

Due to the nature of the weights of the edges in the adjacency matrix, it is also possible to find out which equivalence classes have a bipartite form. For example, in the 2-to-5 codes, since a bipartite form would need a_0, a_1, \dots, a_8 to all equal 0, we look for the integer representations that are $0 \pmod{512}$.

By running a conditional statement in this way, we find the equivalence classes that have bipartite forms and write down the number of bipartite forms in these classes.

We tabulate the equivalence classes with bipartite forms and the number of bipartite forms in these classes for the 2-to-3, 2-to-4, and 2-to-5 encoders in Figure 12, Figure 13, and Figure 14.

Note that, for 2-to- n encoders, the total number of bipartite encoders is $(2^{n-2})^2 = 2^{2n-4}$ since the

two input nodes have 2 choices for each of the $n - 2$ non-pivot output nodes.

The reason we separated out the equivalence classes that have bipartite forms from those that do not is that bipartite forms are simpler from their non-bipartite counterparts. Bipartite forms avoid including output-output edges. For the equivalence classes that have bipartite forms, it is possible that the simplest form of the ZX diagrams is a bipartite form.

7. EQUIVALENCE CLASSES WITH BIPARTITE FORM(S)

Building off of Section 6, we now turn to choosing a specific bipartite form in an equivalence class that can serve as a simple representative.

First, consider the small case of 2-to-4 codes. The adjacency matrix of a bipartite form, taking into account the RREF and pivot simplifications from section IV, would look like:

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

By changing the bolded entries between 0s and 1s, there are 16 possible bipartite forms among all 2-to-4 codes.

In some encoder diagrams, the graph can be split into separate parts; that is, some vertices are not entangled in any way with some other vertices.

Definition 7.1. Suppose a graph $G = (V, E)$ has two distinct vertices a and b such that there is no path between them. Then, graph G is a *disjoint* graph.

In the form of an adjacency matrix for 2-to- n , the graph is not disjoint if and only if there exists a column in which both entries are 1 and there exists no column in which both entries are 0. This is clear, since this means the two parts of the graph (one containing all input-output edges from the first input, the other containing all input-output edges from the second input) are entangled at some vertex, and there is no output vertex not connected to anything.

Claim 7.1. Among all non-disjoint 2-to-4 bipartite graphs, there is only 1 distinct graph up to equivalence through operations 2 through 4 from Conjecture 4.1.

Proof. There are only 5 possible input-to-output ad-

jacency matrices in this case:

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

Consider the first matrix above. Switching the third and fourth output vertices (corresponding to the third and fourth columns of the matrix) results in

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

which is the second matrix. From here, use operation 4 to replace the first row with the sum of the first and second rows modulo 2:

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

Permuting the outputs achieves the third and fourth matrices. Lastly, starting from the above matrix, use operation 4 to replace the second row with the sum of the current first and second rows modulo 2 to find

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

This can be permuted to give the fifth matrix. Analogous sequences of operations can bring any of the other matrices to another, so all 5 of the graphs are equivalent, showing the claim. \square

Using Claim 7.1, the canonical form for the equivalence class that contains these 5 adjacency matrices can be chosen to be

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

First, this graph shares the property of having the least number of edges. Next, to distinguish between the four matrices with the least number of edges, we take the one that has more edges reserved for the first input and first free output, which would be the one shown above.

By writing one of these input-to-output adjacency matrices into the full 6×6 matrix, we can deduce the integer representation (as described in Section 6) of the chosen matrix, and display its ZX diagram. In Figure 15, the ZX diagram is shown in the center and the representative of the equivalence class of this

diagram is shown at left. The rightmost ZX diagram is another bipartite form in the same equivalence class.

Now, we present a general method of simplifying a bipartite $2 \times n$ input-to-output adjacency matrix.

Claim 7.2. Consider a 2-to- n encoder graph equivalent to some bipartite form. It is also equivalent to a bipartite form where the two inputs are both connected to at most $\lfloor \frac{n}{2} - 1 \rfloor$ of the same free outputs.

Proof. In this proof, we only consider encoders that are equivalent to a bipartite form.

For the sake of contradiction, suppose all bipartite forms of this equivalence class have at least $\lfloor \frac{n}{2} \rfloor$ shared free outputs. In an input-to-output adjacency matrix, this would look like

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

The first two columns are fixed to be input-pivot edges, as usual. If there are at least $\lfloor \frac{n}{2} \rfloor$ shared free outputs, the other $n - 2$ columns must have a majority of columns containing two 1's. In this example, 3 out of 5 columns contain two 1's.

However, using operation 4 from Conjecture 4.1, the top row can be replaced with the sum of the top and bottom row modulo 2.

Note that this means all the free outputs that were shared by both inputs have their edges with the first input disconnected, so at least $\lfloor \frac{n}{2} \rfloor$ columns do not have two 1's.

Furthermore, after the operation, the first column cannot possibly have two 1's, so one additional column does not have two 1's. The example matrix above turns into

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

We can rearrange output vertices to bring back the pivots. The following is thus equivalent

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

Thus, the maximum number of columns with two 1's is now $n - 1 - \lfloor \frac{n}{2} \rfloor$. However,

$$n - 1 - \lfloor \frac{n}{2} \rfloor < \lfloor \frac{n}{2} \rfloor,$$

so we reach a contradiction, since there are now less than $\lfloor \frac{n}{2} \rfloor$ shared free outputs in an equivalent bipartite form.

Therefore, the claim holds. \square

Claim 7.2 demonstrates that we can choose a bipartite form that has a relatively small number of shared free outputs. In fact, if the top row is the horizontal vector \mathbf{a} and the bottom row is the horizontal vector \mathbf{b} , by linear operations, there are only 3 possibilities of unordered combinations of two vectors in the rows. It could be (\mathbf{a}, \mathbf{b}) , $(\mathbf{a} + \mathbf{b}, \mathbf{b})$, $(\mathbf{a} + \mathbf{b}, \mathbf{a})$. Then, we can choose which of these bipartite forms has the least number of shared free outputs and thus minimize this number.

Now, we classify operations 2, 3, and 4 from Conjecture 4.1 as *easy* operations, while operations 1 and 5 are *hard* operations.

In an attempt to show the uniqueness of the bipartite forms in an equivalence class, we conjecture the following:

Conjecture 7.1. In an equivalence class with bipartite forms, all such bipartite forms can be transformed from one to another using only easy operations.

One straightforward approach starts by assuming for the sake of contradiction that two bipartite graphs, G_1 and G_2 , are equivalent even though they cannot be transformed from one to another using only easy operations. Then, we can write down partial traces between all pairs of vertices of one bipartite graph. This quantifies the entanglement of the whole graph. If the partial traces between all pairs of vertices of the other bipartite graph are different, then the entanglement is evidently different and we would reach a contradiction.

8. OTHER EQUIVALENCE CLASSES

In addition to equivalence classes that have bipartite forms, there are also equivalence classes with no bipartite graphs, and it is less intuitive to determine which graphs we call canonical forms.

Definition 8.1. A *lacking* equivalence class has zero bipartite forms among its ZX diagrams.

From Definition 7.1, disjoint encoder graphs cannot be transformed into a non-disjoint encoder graph using operations from Conjecture 4.1. Therefore, if there exists a disjoint encoder graph in an equivalence class, all graphs in the equivalence class are disjoint.

In the 2-to-4 codes, using the tables from Section 5, there are 5 *lacking* equivalence classes.

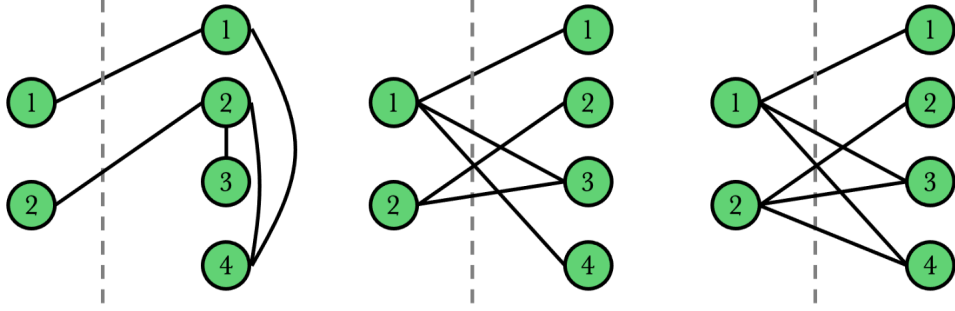


FIG. 15: In the 2-to-4 equivalence class with the representative shown at left, the possible bipartite forms are shown above.

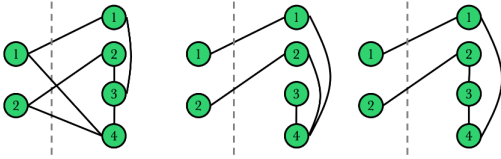


FIG. 16: The representatives of the three different lacking equivalence classes among the classes for 2-to-4 ZX diagrams.

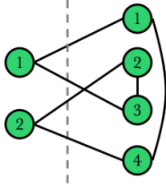


FIG. 17: A ZX diagram with the minimum number of output-output edges in its equivalence class. The equivalence class is lacking and does not contain any ZX diagrams with 0 output-output edges or 1 output-output edge.

The diagrams for these classes reveal that 3 of these classes have only non-disjoint ZX diagrams, which are shown in Figure 16.

Using the code from Section 5, we were able to determine that the equivalence class of the leftmost diagram in Figure 16 does not contain any elements with 0 output-output edges or 1 output-output edge. Instead, the smallest number of output-output edges is 2. An example of a ZX diagram in this equivalence class with 2 output-output edges is shown in Figure 17.

The symmetry of the graph in Figure 17 suggests that the graph for the representative of the class in Figure 16 is also symmetrical. Indeed, the inputs

in the leftmost graph of Figure 16 share symmetrical connections, and their pivots share symmetrical connections. Both pivots, outputs 1 and 2, have an edge with output 3. Both inputs 1 and 2 share an edge with output 4, and they also both have an input-pivot edge. Note that the graph in Figure 17 makes this symmetry more clear.

9. CONCLUSION

This paper presented our work on detailing the canonical forms and structures of select toric and surface codes and tabulated results from Java code that determined the representative sizes of equivalence classes for ZX diagrams within sector Y of Figure 1, which allowed equivalence up to permuting output nodes, permuting input nodes, and local complementation, but not removing extra states. Among these equivalence classes, we also analyzed those that contained bipartite forms among the 2-to-4 codes.

The work done on toric and surface codes ties in with the larger goal of finding explainable canonical forms given the stabilizers of an error-correcting code. The advantage of the toric and surface codes presented in this paper are their structural simplicity. As seen by the general toric codes and the selected surface codes derived, the number of internal nodes in the ZX normal form is reduced to 0 using algorithm we provided. Future works can extend these simplifications (i.e. of removing internal nodes) to other CSS codes and find patterns among the structures of the codes to see if newer definitions of canonicity could yield efficient transformations from the stabilizer formalism to the ZX calculus, as well as from the ZX calculus to the stabilizer formalism.

Furthermore, the tabulations resulting from consideration of the permutation of outputs and removal

of local operations will provide future works a basis to determining patterns among equivalence class sizes and representatives. Extending the definition of equivalence to allow for permutation of outputs is physically significant as these permutations do not affect the amount of entanglement the input qubits go through, and thus the ordering of output qubits could be considered as a non-fundamental aspect of a quantum error-correcting code.

10. ACKNOWLEDGMENTS

We would like to thank the MIT PRIMES-USA program for the opportunity to conduct this research. Thank you to Jonathan Lu for support in creating the diagrams. ABK was supported by the National Science Foundation (NSF) under Grant No. CCF-1729369.

-
- [1] H. Xu, C. Li, G. Wang, H. Wang, H. Tang, A. R. Barr, P. Cappellaro, and J. Li, Two-photon interface of nuclear spins based on the optonuclear quadrupolar effect, *Physical Review X* **13**, 011017 (2023).
- [2] M. A. Nielsen and I. Chuang, *Quantum computation and quantum information* (2002).
- [3] Y. Kim, A. Eddins, S. Anand, K. X. Wei, E. Van Den Berg, S. Rosenblatt, H. Nayfeh, Y. Wu, M. Zaletel, K. Temme, *et al.*, Evidence for the utility of quantum computing before fault tolerance, *Nature* **618**, 500 (2023).
- [4] P. W. Shor, Scheme for reducing decoherence in quantum computer memory, *Physical review A* **52**, R2493 (1995).
- [5] A. Steane, Multiple-particle interference and quantum error correction, *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* **452**, 2551 (1996).
- [6] E. Knill, R. Laflamme, R. Martinez, and C. Negrevergne, Benchmarking quantum computers: The five-qubit error correcting code, *Physical Review Letters* **86**, 5811 (2001).
- [7] A. Y. Kitaev, Quantum error correction with imperfect gates, in *Quantum communication, computing, and measurement* (Springer, 1997) pp. 181–188.
- [8] D. Gottesman, *Stabilizer codes and quantum error correction* (California Institute of Technology, 1997).
- [9] M. Van den Nest, J. Dehaene, and B. De Moor, Graphical description of the action of local clifford transformations on graph states, *Physical Review A* **69**, 022316 (2004).
- [10] A. T. Hu and A. B. Khesin, Improved graph formalism for quantum circuit simulation, *Physical Review A* **105**, 022432 (2022).
- [11] B. Coecke and R. Duncan, Interacting quantum observables, in *International Colloquium on Automata, Languages, and Programming* (Springer, 2008) pp. 298–310.
- [12] B. Coecke and R. Duncan, Interacting quantum observables: categorical algebra and diagrammatics, *New Journal of Physics* **13**(4), 043016 (2011).
- [13] M. Backens, The zx-calculus is complete for stabilizer quantum mechanics, *New Journal of Physics* **16**, 093021 (2014).
- [14] T. Peham, L. Burgholzer, and R. Wille, Equivalence checking of quantum circuits with the zx-calculus, *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* **12**, 662 (2022).
- [15] A. Cowtan and S. Majid, Quantum double aspects of surface code models, *Journal of Mathematical Physics* **63**, 042202 (2022).
- [16] J. van de Wetering, Constructing quantum circuits with global gates, *New Journal of Physics* **23**, 043015 (2021).
- [17] R. D. East, J. van de Wetering, N. Chancellor, and A. G. Grushin, Aklt-states as zx-diagrams: diagrammatic reasoning for quantum states, *PRX Quantum* **3**, 010302 (2022).
- [18] N. de Beaudrap and D. Horsman, The zx calculus is a language for surface code lattice surgery, *Quantum* **4**, 218 (2020).
- [19] A. Kissinger and J. van de Wetering, Reducing the number of non-clifford gates in quantum circuits, *Physical Review A* **102**, 022406 (2020).
- [20] A. B. Khesin, J. Z. Lu, and P. W. Shor, Graphical quantum clifford-encoder compilers from the zx calculus (2023), [arXiv:2301.02356 \[quant-ph\]](https://arxiv.org/abs/2301.02356).
- [21] J. Huang, S. M. Li, L. Yeh, A. Kissinger, M. Mosca, and M. Vasmer, Graphical css code transformation using zx calculus, *arXiv preprint arXiv:2307.02437* (2023).
- [22] A. Kissinger, Phase-free zx diagrams are css codes (... or how to graphically grok the surface code), *arXiv preprint arXiv:2204.14038* (2022).
- [23] A. Y. Kitaev, Fault-tolerant quantum computation by anyons, *Annals of physics* **303**, 2 (2003).
- [24] Quantomatic, <https://quantomatic.github.io/>, accessed: August 5, 2023.
- [25] J. C. Adcock, S. Morley-Short, A. Dahlberg, and J. W. Silverstone, Mapping graph state orbits under local complementation, *Quantum* **4**, 305 (2020).
- [26] M. Bahramgiri and S. Beigi, Enumerating the classes of local equivalency in graphs, *arXiv preprint math/0702267* (2007).
- [27] A. Bouchet, Recognizing locally equivalent graphs, *Discrete Mathematics* **114**, 75 (1993).

Appendix A: Constructing the 3-by-3 toric code

In Section 3, we provided the 2-by-2 and 3-by-3 toric codes in ZX calculus. Here, we provide a more detailed description of the methodology used to determine the structure of the 3-by-3 toric code.

After placing down the output nodes as in Figure 4, we expect some of the output edges to contain Hadamard gates. Suppose the output edge onto vertex 1 has a Hadamard. This implies that applying the stabilizer $Z_1Z_4Z_{10}Z_{11}$ would result in sliding a Z gate from the end of the output edge, through the Hadamard (which converts the Z gate to an X gate), then through vertex 1 itself. By the π -copy rule [13], the X gate, which is an X node with phase π , copies itself onto the edges (excluding the output edge) connected to vertex 1. Diagrammatically, applying Z_1 onto the vertex 1 is shown in Figure 18a and Figure 18b since, after passing through the Hadamard gate, the green π node (representing the Z_1) changes into a red π node. Vertex 1 is shown as the green node.

Similarly, while continuing the assumption that output node 1 has a Hadamard, if we instead applied the stabilizer $X_1X_3X_{10}X_{16}$, we slide an X gate from the end of the output edge, through the Hadamard (which converts the X gate to a Z gate), then onto vertex 1. By the spider rule [13], the Z gate, which is a phase π green node, merges with vertex 1, a phase 0 green node. This results in vertex 1 gaining a phase of π .

By the preceding paragraphs, the behavior of the Z and X gates on a Hadamarded output node is understood. The analogous behavior occurs on a non-Hadamarded output node by switching all the colors used in the processes above.

To determine all of the edges in the 3-by-3 toric code in Figure 4, we consider these processes of applying the stabilizers onto the output nodes. Note that all of the internal edges among nodes in the diagram must be Hadamarded edges so that the spider rule cannot be applied to merge multiple nodes into one. To simplify our work, we set the Hadamarded output nodes to be 1, 2, 3, 7, 8, 9, and 16, 17, 18. Then, by stabilizer $Z_1Z_4Z_{10}Z_{11}$, the nodes 4, 10, and 11 gain phase π from their Z gates while node 1 will cause a π -copy rule to move X gates onto the internal edges connected node 1. Since all internal edges are Hadamarded, moving the X gates through the Hadamards will result in Z gates. If these Z gates went to any nodes other than nodes 4, 10, and 11, the stabilizer would not have kept the configuration the same. Therefore, the Z gates must

arrive at only nodes 4, 10, and 11. This works because the π 's from these Z gates cancel with the π s already at the nodes. Thus, the only internal edges to node 1 are from nodes 4, 10, and 11.

Using similar reasoning, we can deduce the rest of the internal edges among the output nodes. Furthermore, to determine the logical operators (to connect the input nodes to), we look for sets of nodes that, when any stabilizer is applied, keep the input node at phase 0.

Appendix B: Hadamard-sliding rule

In the algorithm given in Section 3, we invoked Eq. (10) from [10], which we call the Hadamard sliding rule. The name comes from its ability to move a Hadamard gate from one output node's free edge to a connected output node's free edge, after some operation on the nodes connected to these two.

Diagrammatically, it looks like Figure 19. Note that the colors of the output nodes are swapped. Also, the connections between the neighbors of the output nodes are toggled, and the sets of neighbors are swapped between output nodes. This is equivalent to *local complementation about an edge*, defined in Definition 2.3.

Figure 19 is a simplified version of the Hadamard-slide rule, and it assumes that the graph is bipartite in red and green nodes, which is appropriate for the toric and surface codes that we considered.

Appendix C: Converting a KLS canonical form into a quantum circuit

In Section 2, we described a procedure for converting a KLS canonical form into a quantum circuit, which is repeated here.

1. Start with $n - k$ open wires representing the inputs of the circuit.
2. Add a $|0\rangle$ state for each of the k non-pivot output nodes.
3. Apply an H gate to all n wires.
4. Apply a CX gate between the wires corresponding to the edges between inputs and non-pivot outputs. The input node is the target qubit, and the output node is the controlled qubit.
5. Apply a CZ gate between the wires corresponding to the edges between only outputs.

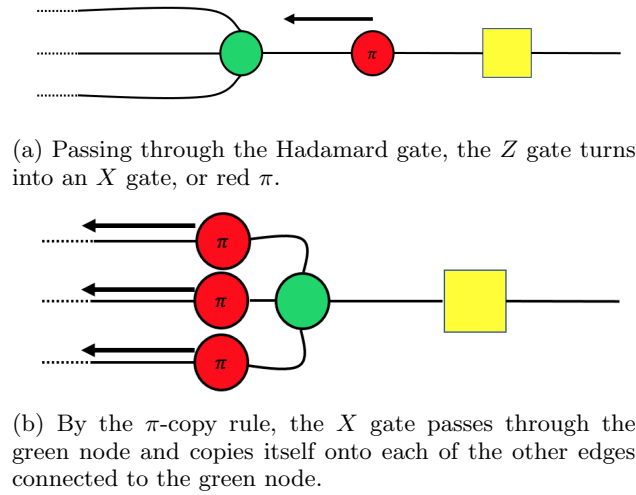


FIG. 18: π -copy rule on a green π node after passing through a Hadamard gate.

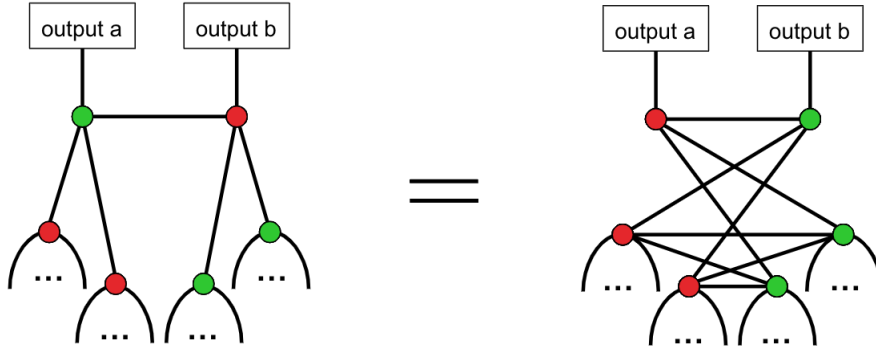


FIG. 19: The Hadamard-sliding rewrite rule for switching Hadamard gates from the free edges of connected output nodes. Note that a red node is equivalent to a green node with a Hadamard on the output edge. Furthermore, the equality shown above assumes that the graph is bipartite in red and green nodes, which is the case for the codes considered in Section 3.

6. Apply the local operations attached to the outputs.

We will now show why this works.

Consider the example given in Figure 20a. We will convert this KLS form into a circuit. We can first move the input nodes to be along the same horizontal wire as their pivot nodes. Then, we split the non-pivot nodes by un-merging two zero-phase green nodes. The resulting diagram is in Figure 20b. From here, the edges from inputs to non-pivot outputs can be separated by un-merging nodes and expressing each edge separately, as in Figure 20c. In Figure 20d, the Hadamards between the inputs and pivots are shown explicitly. In Figure 20e, we do a similar un-merging of nodes to separately express the edges between nodes.

The steps used in these diagrams hold in general. We can un-merge each node until all the edges are expressed separately (and the non-pivot nodes have an initial state), and, to keep things organized, we can keep the input-output edges on the left side and the output-output edges on the right side.

From Figure 20e, note that the Hadamarded edges between the nodes of a ZX diagram are equivalent to the CZ gates between the corresponding wires in a quantum circuit. Also, the green nodes at the start are equivalent to $|+\rangle$.

Now, consider sliding the two Hadamards in the middle towards the left of the diagram. Because $ZH = HX$, this means each of the CZ 's that the H 's pass through turns into a CX with the target qubit on the input's wire. Also, we may exchange the green nodes at the start for a red node and an H ,

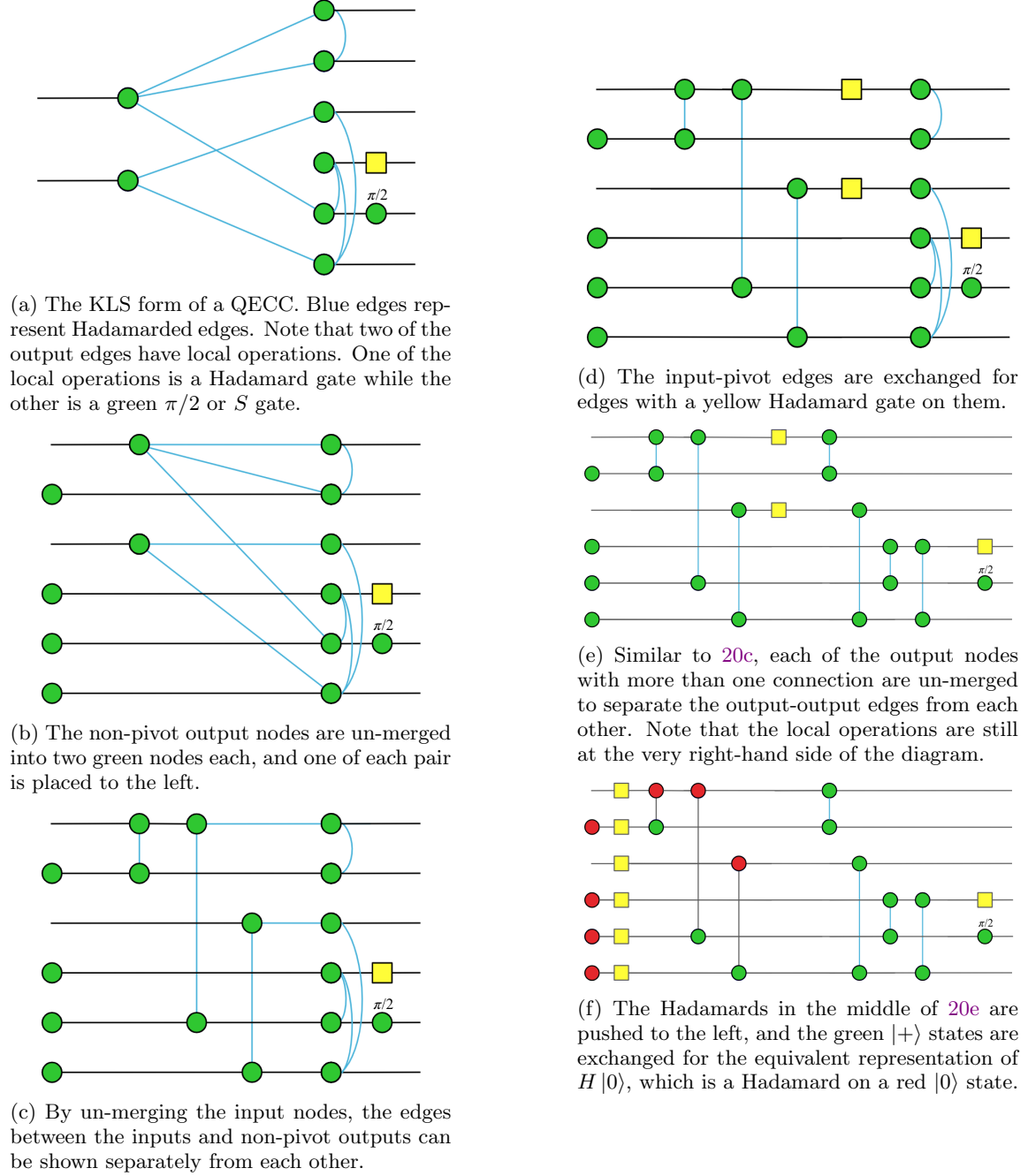


FIG. 20: Conversion of a KLS form ZX diagram into the equivalent quantum circuit diagram.

since $H|0\rangle = |+\rangle$. This gives Figure 20f. From here, we can see why the procedure for creating the circuit from KLS form works, since each of the ZX calculus components in Figure 20f can quickly be converted to a circuit diagram component.

Appendix D: Necessary conditions on the canonical forms of CSS states

In this section of the appendix, we present partial work on attempting to find a canonical form for CSS states, which we define as CSS codes with exactly 0 input nodes.

Definition D.1. *CSS states* are CSS encoders that

have 0 input nodes. Their stabilizer tableaux can be expressed in terms of only X , Z , and I gates, as is the case for CSS codes.

This is based on the conditions of equivalence established in [20] with the additional constraint that any local operations may be added to the output nodes. That is, here we will find necessary conditions on canonical forms that are locally equivalent, as defined in Definition 4.1.

Claim D.1. The following are necessary (but not sufficient) conditions on the canonical forms of locally equivalent CSS states in ZX calculus.

1. *Edge rule:* There is exactly one node per free edge (i.e. output edge), and every internal edge has a Hadamard gate on it [20].
2. *Bipartite rule:* The graph must be bipartite. If the state's nodes are labeled from 1 to n , node 1 is type A , and all nodes are split into type A or B nodes, where the only edges in the graph are between type A and B nodes.
3. *Ordered connections rule:* Type A nodes only connect to nodes that have a higher label.
4. *Local operations rule:* The graph cannot have any local operations on any output nodes. Since all nodes are output nodes, there are no local operations on any nodes.

Before we begin the proof, we refer the reader to [25] for the more general discussion of the orbits of equivalence classes of graph states equivalent under local complementation. Our work here narrows the focus to CSS states and presents rules that could be used to find canonical forms of CSS graph states in future works.

Proof. Because the definition of equivalence was based on those used for the KLS form, the edge rule

is satisfied by considering the CSS state as an encoder with 0 input nodes.

To show the bipartite rule, consider the ZX normal form from [22]. In the normal form based on X -stabilizers, the stabilizers are represented by green nodes connected to red output nodes that the X -check acts on. None of the green nodes are connected to each other, and none of the red nodes are connected to each other. Therefore, the ZX diagram of a CSS state in ZX normal form is initially bipartite. As outlined in Section 3, we can convert the ZX diagram into a form with exactly n nodes, with one node per output edge, using the bialgebra and Hadamard-sliding rules. It can be easily verified that both of these rules cannot change a bipartite graph (with nodes separated by color) into a non-bipartite graph. Therefore, the final graph can always be bipartite.

Now, label the n nodes from 1 to n , and WLOG suppose that node 1 is an A -type node. Then, the bipartite ZX diagram can be split into separate groups of nodes of A and B -type nodes. Now, we multiply the graph state by the necessary local operations such that all A -type nodes have no Hadamard gates on their output edge while all B -type nodes have one Hadamard gate on their output edge. By the Hadamard rule in [20], this means we can necessarily transform the graph using local complementations so that all B -type nodes are only connected to A -type nodes that have lower-numbered nodes. This is equivalent to ordered connections rule listed in Claim D.1.

After converting the ZX normal form into a bipartite ZX diagram with all of the above rules, we can remove all local operations on the output nodes. This is because we can multiply the state by any local operations (by our definition of local equivalence), so we can multiply the state by the conjugate local operations needed to leave behind only green Z nodes connected to the output edges. □