Radboud University

Master's Thesis

ZX-Calculus and Quantum Stabilizer Theory

Coen Borghans

supervised by
Dr. Aleks Kissinger

November 20, 2019

1 Introduction

Quantum computing is the theoretical study of computation based on the rules of quantum mechanics. Where standard models of computation have access to a binary memory consisting of bits, quantum computing uses so called qubits, which are quantum mechanical superpositions of bits. Exploiting quantum phenomena to manipulate these qubits in a useful way could yield enormous speedups when compared to classical computing. As of November 2019 this research area is still mostly theoretical as any scalable, reliable, fault tolerant quantum computers have yet to be constructed. The largest quantum computer that exists today was created by Google and consists of 72 programmable qubits [14]. This limit in size comes from the great difficulties that arise when dealing with quantum effects, which make qubits hard to control and very unstable. Therefore the area of quantum computing is mainly concerned with researching the possibilities of quantum computers, should they exist.

There are a number of ways to look at quantum computing, using different notations or varying in the approach. One such system is a relatively new formalism known as the ZX-Calculus. This is a graphical language for reasoning about quantum circuits based on category theory. Using a set of rewrite rules to transform between diagrams one can intuitively examine the behaviour of quantum circuits. The ZX-Calculus makes complicated quantum circuits more easily readable and allows both humans and machines to reason about them more efficiently.

Quantum stabilizer theory is a part of quantum mechanics that does not describe a quantum state directly, but rather in terms of specific operators for which that state is a fixed point. These operators are called stabilizers. Writing a state in terms of its stabilizers can be more efficient and gives a lot of information about the state that can be used in areas like quantum cryptography.

We will look at how we can convert between ZX-Calculus and stabilizer theory. This means that we develop a procedure to find stabilizers for a given state in the ZX-Calculus as well as a method to create a ZX-state for a given set of stabilizers. These are then presented as algorithms that can be efficiently implemented.

In section 2 we lay out all the required background information. First we give a short introduction to quantum mechanics and quantum computing. Then we explain the basics of ZX-Calculus and stabilizer theory, followed by a short description of how we can transition between the two. The next two sections focus on either side of the conversion process. Section 3 builds up to an algorithm for finding the stabilizers for a given ZX-state and section 4 goes the other way, helping us find a ZX-diagram for a state when given its stabilizers.

2 Background

This section contains a short description of the background that is needed. We discuss some of the basic properties of quantum mechanics, how these are used in quantum computing and we describe our two main areas of interest: Stabilizer Theory and ZX-Calculus. For a more complete introduction to quantum computing we refer to [3]. We assume knowledge of basic linear algebra as taught in a bachelor level course.

2.1 Dirac notation

Quantum computing is based on quantum mechanics, which is mainly studied by physicists. Therefore, we adopt the physicist's convention of writing our linear algebra in Dirac notation. We quickly summarise this notation here.

Given a vector v we write the vector itself as $|v\rangle = v$ and its conjugate transpose as $\langle v| = v^\dagger$. We call the first a *ket* and the second a *bra*. Multiplying a bra and a ket gives a *bra-ket*, or bracket, $\langle v|w\rangle = \langle v||w\rangle$. This is the inner product of v and w. The outer product of two vectors v and w is obtained by multiplying the bra and ket the other way; $|v\rangle\langle w|$.

Looking at tensor products of outer products we have $|v\rangle\langle v| \otimes |w\rangle\langle w| = |v\rangle \otimes |w\rangle\langle v| \otimes \langle w|$. Taking the conjugate transpose of a tensor product gives $(|v\rangle \otimes |w\rangle)^\dagger = \langle v| \otimes \langle w|$. For any linear map A we have $(A|v\rangle)^\dagger = \langle v| A^\dagger$.

2.2 Quantum mechanics

The central idea in quantum computing is to use quantum mechanical phenomena to our advantage when preforming computations. In order to understand what happens in quantum computing, we obviously need a little knowledge of the underlying quantum mechanics. We give a short description of the notions of superposition, entanglement, measurement and unitary evolution.

In classical physics a certain physical system can be in one state at a time. The spin of an electron is either facing up, or facing down. A photon that is shot at a wall with two slits either passes through the upper slit, or it passes through the lower slit. A cat in a box with unstable poisonous gas is either dead or still alive. However, physicists have discovered through numerous experiments that things are not as simple as that. The world we live in shows quantum mechanical behaviour, which surpasses our intuition. It turns out that the spin of an electron can be a superposition of both up and down: it is in both states at once and only when we measure, it collapses to either the one or the other.

More formally speaking, we start with a physical system that can be in one of N classical states. Here classical state means a state in which the system can be when we observe it. To represent these states, we use an orthonormal basis of an

N-dimensional Hilbert space $\mathcal{H}$. For simplicity we will always use $\mathcal{H} = \mathbb{C}^N$. We usually write these basis states as $|0\rangle, |1\rangle, ..., |N-1\rangle$, following the computer science convention. A quantum state $|\psi\rangle$ is a superposition of these classical states, which amounts to a vector of norm 1 in $\mathcal{H}$. That is

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle + ... + \alpha_{N-1} |N-1\rangle$$

Where the value $\alpha_i \in \mathbb{C}$ is called the amplitude of $|i\rangle$. We have that the norm of $|\psi\rangle$ is 1, so $\sum_{i=0}^{N-1} |\alpha_i|^2 = 1$.

We can get make bigger spaces from smaller ones by taking tensor products. Given two spaces, $\mathcal{H}_1$ with orthonormal basis $\{|0\rangle, |1\rangle, ..., |N-1\rangle\}$ and $\mathcal{H}_2$ with orthonormal basis $\{|0\rangle, |1\rangle, ..., |M-1\rangle\}$, we can make a new space $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2$. This space is NM-dimensional and has basis $\{|n\rangle \otimes |m\rangle \, | n \in \{0, ..., N-1\}, m \in \{0, ..., M-1\}\}$. A state $|\psi\rangle \in \mathcal{H}$ is called a bipartite state. We can create a bipartite state by taking the tensor product of two states $|\psi\rangle_1 \in \mathcal{H}_1$ and $|\psi\rangle_2 \in \mathcal{H}_2$: $|\psi\rangle = |\psi\rangle_1 \otimes |\psi\rangle_2 \in \mathcal{H}$. However, not all states in $\mathcal{H}$ can be made like this. A bipartite state that cannot be written as a tensor product of states from the original spaces is called an entangled state. An example of this is the state $\frac{1}{\sqrt{2}} |00\rangle + |11\rangle \in \mathbb{C}^2 \otimes \mathbb{C}^2$. This state is called an EPR-pair after Einstein, Podolski and Rosen and shows some interesting, counter-intuitive behaviour. At first the state is not classical in either space. However, if we measure just one of the parts we get $|0\rangle$ or $|1\rangle$ and have the other part collapse to the same. This means that measuring in one space can have effect on the state in another space if we have entanglement. This phenomenon is known as quantum non-locality.

There are two things that we can do to a quantum state. The first is that we can measure it. When observing a quantum state, we cannot see the full superposition. The only thing we can do is measure it, that is to say, observe it, at which moment the state collapses to only one of the possible classical states, which we can see. So measuring a quantum state destroys all the quantum mechanical information and we are left with a classical state. However, it is not predetermined which classical state will be found when measuring a quantum state. In fact, the probability of getting outcome $|i\rangle$ is $|\alpha_i|^2$, the squared norm of the amplitude. This is the main reason why we require a quantum state $|\psi\rangle$ to have Euclidean norm 1.

The second thing we can do is to apply some linear function to a quantum state. This amounts to multiplying the state by some operator U. The new state $U|\psi\rangle$ must again be a quantum state, so we require

$$|U|\psi\rangle|^2 = 1$$

Or in other words, we need U to be unitary. As every unitary has an inverse ($U^{-1} = U^\dagger$), we know that every operation on quantum states is reversible. This is in contrast to the previously described measuring, which is not reversible as

you cannot construct the full superposition from just one classical state.

Some of the most important unitaries in quantum computing include the Pauli operators X, Y and Z, the Hadamard gate H and the controlled not CNOT:

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

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

2.3 Quantum Computing

Classical computing is based around the bit, a unit of information that can take the values 0 and 1. When computing we start with a number of bits and manipulate them with certain operations in order to get a result. In quantum computing we have the quantum bit, or qubit, which is a superposition of the classical 0 and 1. Mathematically we represent this as follows. Let $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ be orthonormal basis states of $\mathbb{C}^2$. Then a qubit is a unit vector in $\mathbb{C}^2$, or in other words, a superposition of $|0\rangle$ and $|1\rangle$. So a qubit looks like

$$\alpha_0 |0\rangle + \alpha_1 |1\rangle$$

where $|\alpha_0|^2 + |\alpha_1|^2 = 1$.

Of course we can also use different bases for the same space. Next to $\{|0\rangle, |1\rangle\}$, which is known as the computational basis, we also often use $|+\rangle = H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ as an orthonormal basis.

As in classical computing, we often look at systems of many qubits at the same time. Such a system is just the tensor product of the one qubit case. For example, a system on two qubits exists in $\mathbb{C}^2 \otimes \mathbb{C}^2 \cong \mathbb{C}^4$. The basis vectors of this space are

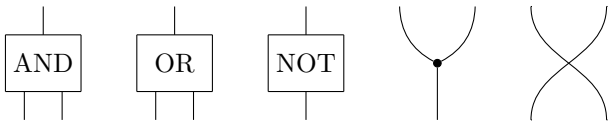
$$|0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \ |0\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \ |1\rangle \otimes |0\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \text{ and } |1\rangle \otimes |1\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix},$$

which is often abbreviated to $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$.

When looking at n qubits together we call this a register of qubits. Such a register of n qubits can be in a superposition of 2^n basis states of $\mathbb{C}^{2^n}$. This makes

it seem as if qubits can store exponentially more information than regular bits, but we must not forget that we cannot access all this information. When we measure the register of qubits, it collapses to a classical state, removing all the other information. In quantum computing the challenge is to make use of the hidden quantum information in such a way that after measuring, we still get a useful answer. Many algorithms exist that do this successfully. Famous examples include Shor's factoring algorithm and Grover's search algorithm, which provide great speedups compared to all known classical counterparts. See e.g. [3] for more details.

In classical computing the bits are manipulated by applying a series of elementary gates in a specific order. These gates include AND, OR, NOT, COPY, SWAP, etc. Formally the way to manipulate bits is by applying a map $f : \mathbb{B}^n \to \mathbb{B}^m$. Making such a map from elementary gates amounts to taking tensor products and composing. Writing something out like that can get complicated and difficult to read or interpret. Instead, it is often much more intuitive and useful to draw such a map as a diagram or circuit where the elementary gates are represented as boxes, dots and wires:



These can then be composed by plugging the output wires (to the top) into the input (at the bottom) of the next , or tensored by simply placing them next to each other.

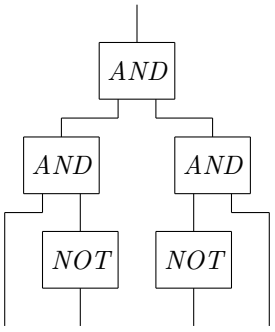
Example 2.3.1. *The map*

$$\begin{aligned} f : \mathbb{B}^4 &\to \mathbb{B} \\ x &\mapsto 1 \ \textit{if}\ x = 1001 \\ x &\mapsto 0 \ \textit{otherwise} \end{aligned}$$

can be expressed in terms of elementary gates as follows:

$$f = AND \circ AND \otimes AND \circ I \otimes NOT \otimes NOT \otimes I$$

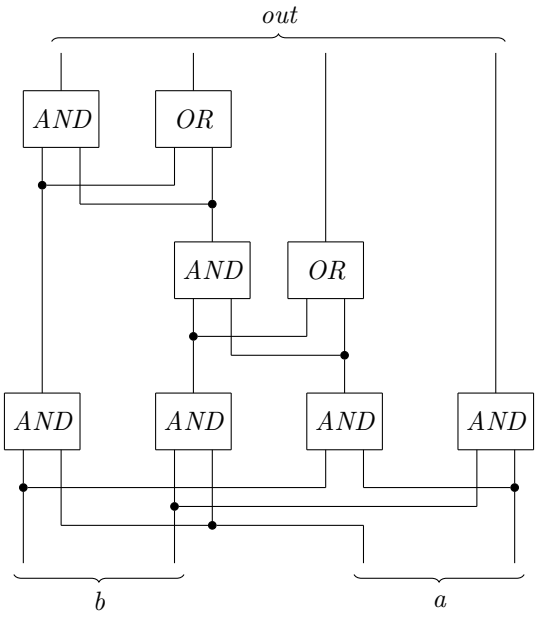
Alternatively, we can represent the same map with a circuit:



Example 2.3.2. *The map $f : \mathbb{B}^4 \to \mathbb{B}^4$ that multiplies two binary numbers a and b can be expressed in terms of elementary gates as follows:*

$$\begin{aligned} f = {} & AND \otimes OR \otimes I \otimes I \circ SWAP_{4,5} \circ COPY \otimes COPY \otimes I \otimes I \circ I \otimes AND \otimes OR \otimes I \\ & \circ SWAP_{3,4} \circ I \otimes COPY \otimes COPY \otimes I \circ AND \otimes AND \otimes AND \otimes AND \\ & \circ SWAP_{4,7} \circ SWAP_{2,5} \circ SWAP_{2,3} \circ COPY \otimes COPY \otimes COPY \otimes COPY \end{aligned}$$

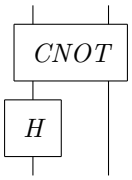
Alternatively, it can be represented as a circuit, which is significantly easier for humans to read and reason about:



In quantum computing we do something quite similar to classical computing. The qubits are prepared in some state, usually $|00...0\rangle$, then we apply a series of unitaries to these qubits, followed by a measurement to give a final outcome. As n qubits live in a 2^n-dimensional space, calculating the matrices that correspond to some larger unitaries can get hard quite fast. To simplify things, often some small set of unitaries, including $X, Z, CNOT$ etc., is chosen to be regarded as elemental gates. These can then be combined by taking tensor products and composing, which leads to bigger maps. However, just like in classical computing it can be difficult to read bigger combinations of elementary gates. Therefore it can be more useful to look at quantum computing through circuits. This is called the quantum circuit model.

Example 2.3.3. *When we plug the state $|00\rangle$ into the following quantum circuit, we first apply H to the first qubit, yielding $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|0\rangle$. Then CNOT gives*

$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. *So this circuit turns* $|00\rangle$ *into an entangled state.*

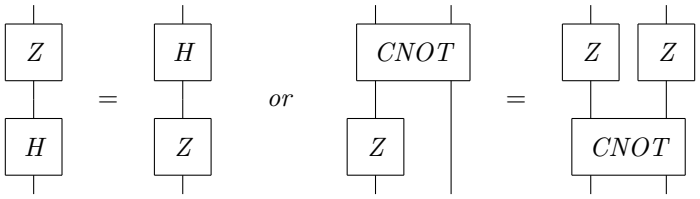


The analogue of the classical NOT is the Pauli X as $X|0\rangle = |1\rangle$ and $X|1\rangle = |0\rangle$. There are no analogues for AND and OR because these are not reversible. Also, we cannot copy qubits, by the quantum no-cloning theorem (see e.g. [1]).

2.4 ZX-Calculus

ZX-Calculus is a graphical language that is similar to the quantum circuit notation that we described before. Instead of reasoning about big tensor products of linear maps in the form of matrices, the calculation of which gets complicated rather quickly, ZX-Calculus offers a way to analyse quantum processes in a simpler, more intuitive diagrammatic way. In addition to that, it provides a set of rewrite rules that make reasoning about diagrams easier. This makes it so that one can graphically derive equalities between diagrams and get a better understanding of what a diagram does.

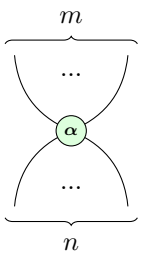
Example 2.4.1. *Equalities such as*



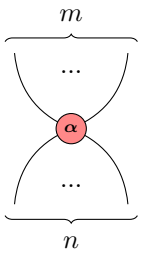
might not be evident in the quantum circuit notation without writing out the associated matrices or checking the possible inputs. However, using the rewrite rules of the ZX-Calculus such equalities are immediately clear.

Quantum states and maps are represented by ZX-diagrams. These diagrams consist of so called spiders that are connected by wires. Some wires are not connected at the bottom, in which case they are an input, and other wires are not connected at the top, making them outputs. There are two types of spiders; green and red. Next to that, spiders also have a phase α, a number that is taken

modulo 2π. These red and green spiders represent the following maps:



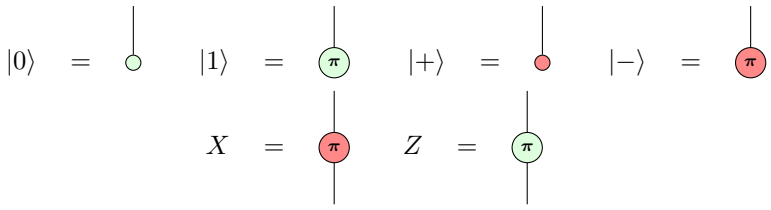
$$= \quad |0^m\rangle\langle 0^n| + e^{i\alpha}\,|1^m\rangle\langle 1^n|$$



$$= \quad |+^m\rangle\langle +^n| + e^{i\alpha}\,|-^m\rangle\langle -^n|$$

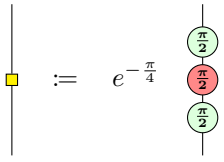
If the phase of a spider is 0, we will omit the number and draw a smaller empty spider.

In particular, if a diagram has no input it represents a state. We recognize the following simple ZX-diagrams (up to normalization):



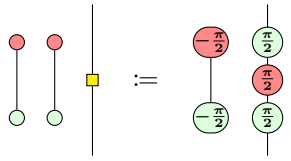
Two spiders can be connected by plugging any output wires of one spider into some input wires of the other. This can be repeated to create larger diagrams.

We will use the following shorthand notation for the Hadamard gate H in the ZX-Calculus:

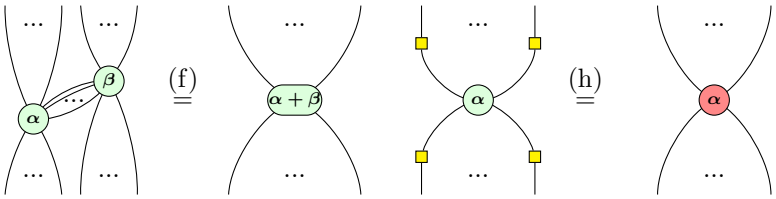


Remark. Diagrams with no input and no output represent scalars. When studying ZX-Calculus these are often omitted, as they have little physical relevance and are not interesting for the structure of a diagram. However, in what we are going to do, keeping track of scalars is necessary. We have chosen to write

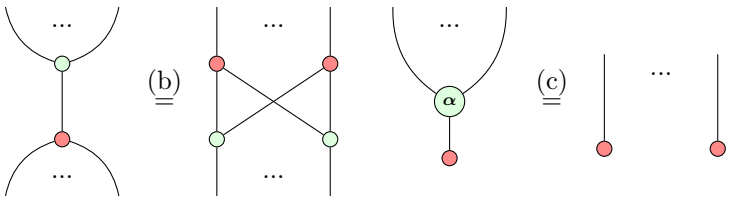
scalars out as the number they represent, instead of leaving them as a diagram. This is why we have the above equation and not the purely ZX-variant:



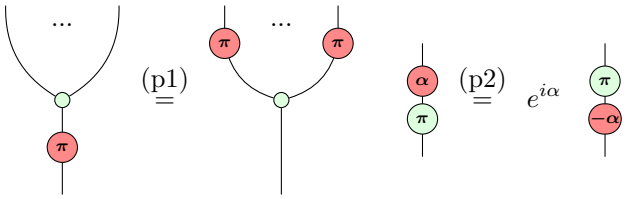
We now describe the rules of the ZX-Calculus as presented in [4]. In all these rules ellipses (...) mean zero or more wires. First of all, we can fuse spiders of the same colour together and change the colour of a spider as follows:



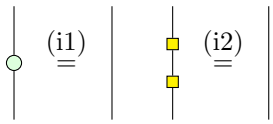
Next, the bialgebra rule can be used to replace a specific combination of red and green spiders by one green and red spider. The copy rule allows us to copy phase zero spiders through opposite coloured spiders:



Phase π-spiders can be copied though using the pi-copy rule and two spiders can be rearranged:

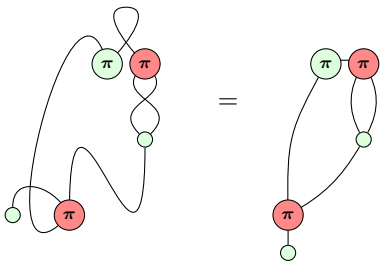


Lastly, we have two identity rules:



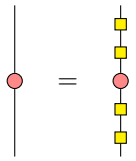
Next to these explicit rules, there is one more overarching rule: only connectivity matters. This means that two diagrams that are topologically the same,

represent the same map or matrix. Moving spiders around without changing connections or bending wires does not change the meaning of a diagram. We have for instance:

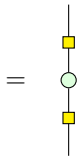


Using rules (h), (f) and (i2) we see that all rules are also true when the colours are flipped.

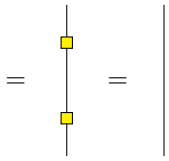
Example 2.4.2. *To see that rule (i1) also holds for a red spider, we apply rule (i2) to both wires:*



Next, we change colour with rule (h):

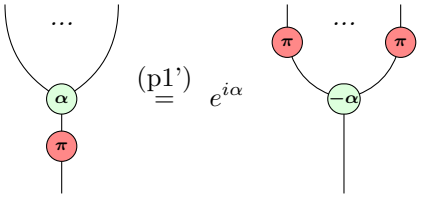


What is left is to use (i1) and (i2) to get the desired result:

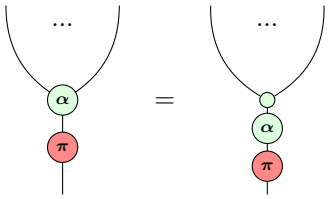


From the given rules of the ZX-Calculus we can derive many others. Some of these will prove to be useful later on and are therefore ideal to serve as an example of reasoning in the ZX-Calculus.

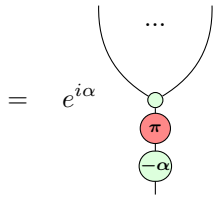
Proposition 2.4.3. *The following stronger version of rule (p1) is derivable in the ZX-Calculus:*



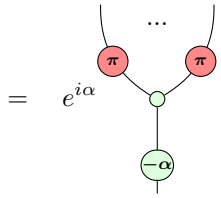
Proof. We un-fuse the green α-spider to get:



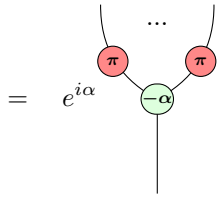
Then we swap using rule (p2)



Now we can copy by applying (p1)

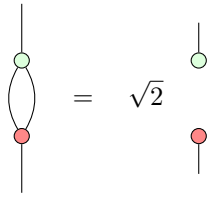


Which just leaves fusing the green spiders with rule (f) to get the desired result

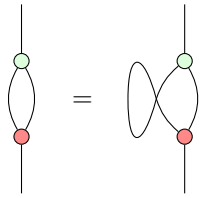


□

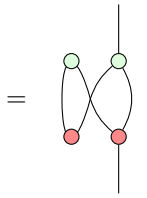
Proposition 2.4.4 (Lemma 2.2 in [2]). *The following rule is derivable in the ZX-Calculus*



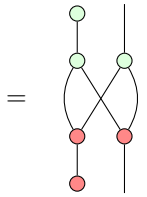
Proof. Starting with the left hand side, we can bend one of the two parallel wires to get



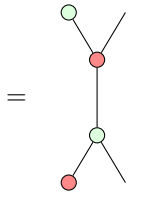
Using the both colour versions of the identity rule (i1), we add two spiders to the newly created loop



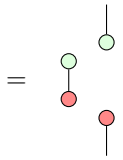
From these spiders we can extract two new spiders by the spider fusion rule (f)



By rule (b) we have



Now, by applying the copy rule (c), we get



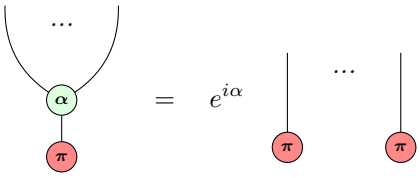
Which leaves us to verify that the value of the scalar is indeed $\sqrt{2}$. We know that  $= |+\rangle + |-\rangle$ and  $= \langle 0| + \langle 1|$. Composing them gives us

$$\begin{aligned} &= (\langle 0| + \langle 1|)(|+\rangle + |-\rangle) \\ &= (\langle 0| + \langle 1|)(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) + \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)) \\ &= (\langle 0| + \langle 1|)\frac{2}{\sqrt{2}}|0\rangle \\ &= \sqrt{2}(\langle 0|0\rangle + \langle 0|1\rangle) \\ &= \sqrt{2} \end{aligned}$$

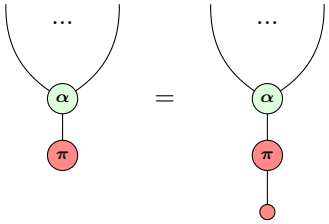
as desired

□

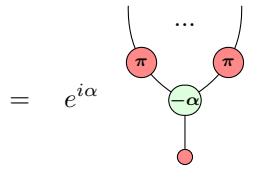
Proposition 2.4.5 (Lemma 2.3 in [2])**.** *The following is rule is derivable in the ZX-Calculus*



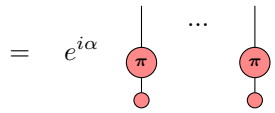
Proof. We can pull a red spider out the bottom by using the spider fusion rule (f) from right to left:



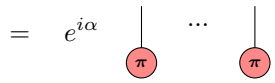
Then we copy the red π using rule (p1'), which we derived in Proposition 2.4.3



Now we can copy the red spider with rule (c)



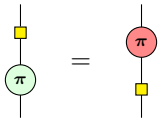
and finish the proof by fusing the spiders with rule (f)



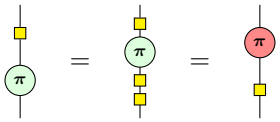
□

Another useful thing to note is that spiders on a wire can pass by Hadamard gates by changing colour.

Lemma 2.4.6. *The following holds in the ZX-Calculus:*



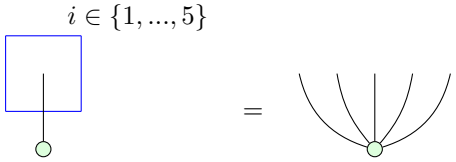
Proof. We apply rule (i2) backwards and then change colour with rule (h):

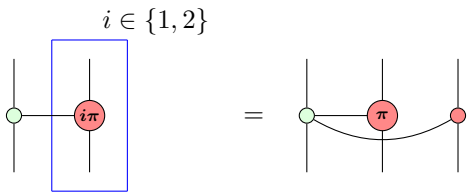


□

To simplify large ZX-diagrams we make use of the !-box (pronounced as *bang-box*) notation introduced in [12]. A !-box is drawn as a blue rectangle in a diagram and is used to indicate that the sub-graph it surrounds is copied as many times as indicated, potentially depending on some parameter.

Example 2.4.7. *The !-box notation is used as follows:*





This notation can make complicated ZX-diagrams more structured and makes it easier to prove more general features of diagrams later on.

ZX-Calculus is a new way of representing quantum states and maps. Of course this is only useful if it agrees with the old theory. Luckily it is not hard to check that all the rules in the ZX-Calculus are sound, meaning that they are also true when translated to the linear maps they represent. Furthermore, when restricted to a small part of quantum mechanics, namely stabilizer quantum mechanics, the ZX-Calculus is known to be complete. This means that any equality that is valid in stabilizer quantum mechanics can also be derived in the ZX-Calculus.

We only covered the bare basics of ZX-Calculus in this section. A full description of the ZX-Calculus and graphical reasoning can be found in [1].

2.5 Stabilizer theory

Quantum error correction plays a significant role in realizing quantum computers in practice. As qubits are hugely susceptible to noise due to their unstable nature, it is essential that there exists a way to detect and correct errors that are induced during computation. This can be achieved by creating so-called quantum error correcting codes (see e.g. [8]), which are analogous to their classical counterparts. One powerful way of creating such quantum error correcting codes is by using stabilizer theory. This area of quantum mechanics only allows a certain number of operators to be used. These are known as Clifford operators. After this restriction, stabilizer quantum mechanics can be efficiently simulated classically (as seen in [11]), so without using quantum mechanics. However, stabilizer quantum mechanics still shows quantum behaviour and is interesting to study.

At the core of stabilizer theory stand the Pauli operators that we have encountered before:

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

Note that $X^2 = Y^2 = Z^2 = I$.

Definition 2.5.1. *The Pauli group, $\mathcal{P}_1$, is the group generated by $\{X, Y, Z\}$ under composition.*

It can be easily verified that the Pauli group consists of the sixteen operators of the form $\alpha \cdot P$, where $\alpha \in \{1, -1, i, -i\}$ and $P \in \{I, X, Y, Z\}$. Furthermore, we see that two elements of the Pauli group always either commute or anti-commute.

We can expand this definition to operators on multiple qubits.

Definition 2.5.2. *The n-qubit Pauli group, $\mathcal{P}_n$, is the n-fold tensor product of $\mathcal{P}_1$:*

$$\mathcal{P}_n = \mathcal{P}_1 \otimes \mathcal{P}_1 \otimes ... \otimes \mathcal{P}_1 = \{\alpha \cdot P_1 \otimes P_2 \otimes ... \otimes P_n | \alpha \in \{1, -1, i, -i\}, P \in \{I, X, Y, Z\}\}$$

The operators in the Pauli group have some interesting, easy to prove properties. We discuss these in the next proposition.

Proposition 2.5.3. *An operator $P = \alpha \cdot P_1 \otimes P_2 \otimes ... \otimes P_n \in \mathcal{P}_n$ has the following three properties:*

1. *P is unitary.*
2. *$P^2 = \alpha^2 \cdot I = \pm I$.*
3. *Given $Q = \beta \cdot Q_1 \otimes Q_2 \otimes ... \otimes Q_n \in \mathcal{P}_n$ we have $PQ = \pm QP$. Where P and Q commute iff an even number of P_i and Q_i anti-commute and P and Q anti-commute iff an odd number of P_i and Q_i anti-commute.*

Proof. First of all, we have

$$\begin{aligned} P^\dagger = (\alpha \cdot P_1 \otimes P_2 \otimes ... \otimes P_n)^\dagger &= \bar{\alpha} \cdot P_1^\dagger \otimes P_2^\dagger \otimes ... \otimes P_n^\dagger \\ &= \bar{\alpha} \cdot P_1^{-1} \otimes P_2^{-1} \otimes ... \otimes P_n^{-1} \\ &= (\alpha \cdot P_1 \otimes P_2 \otimes ... \otimes P_n)^{-1} = P^{-1}. \end{aligned}$$

Next, the second property holds as

$$\begin{aligned} P^2 = (\alpha \cdot P_1 \otimes P_2 \otimes ... \otimes P_n)^2 = \alpha^2 \cdot P_1^2 \otimes P_2^2 \otimes ... \otimes P_n^2 &= \alpha^2 \cdot I \otimes I \otimes ... \otimes I \\ &= \alpha^2 I. \end{aligned}$$

Lastly,

$$\begin{aligned} PQ &= (\alpha \cdot P_1 \otimes P_2 \otimes ... \otimes P_n)(\beta \cdot Q_1 \otimes Q_2 \otimes ... \otimes Q_n) \\ &= \alpha\beta \cdot P_1 Q_1 \otimes P_2 Q_2 \otimes ... \otimes P_n Q_n \\ &= \alpha\beta \cdot (-1)^{s_1} Q_1 P_1 \otimes (-1)^{s_2} Q_2 P_2 \otimes ... \otimes (-1)^{s_n} Q_n P_n = \alpha\beta(-1)^{\sum s_i} \cdot QP. \end{aligned}$$

Here we take $s_i = 0$ if P_i and Q_i commute and $s_i = 1$ if they do not. Then indeed P and Q commute iff an even number of P_i and Q_i anti-commute and they anti-commute otherwise. □

If we ignore the scalar, every member of the n-qubit Pauli group can be expressed as $X^{x_1}Z^{z_1} \otimes X^{x_2}Z^{z_2} \otimes ... \otimes X^{x_n}Z^{z_n}$ with $x_i, z_i \in \{0, 1\}$, as $XZ = iY$. This gives us a nice correspondence between $\mathcal{P}_n$ and $\mathbb{B}^{2n}$. More formally, if we factor out the center $Z(\mathcal{P}_n) = \{\alpha I | \alpha = 1, -1, i, -i\}$ we end up with a group $\overline{\mathcal{P}_n} = \mathcal{P}_n / Z(\mathcal{P}_n)$ of order 4^n. This can be identified with the binary vector space $\mathbb{B}^{2n}$ by a useful isomorphism.

Proposition 2.5.4. *The function*

$$f : \mathbb{B}^{2n} \to \overline{\mathcal{P}_n}$$
$$(x_1, x_2, ..., x_n, z_1, z_2, ..., z_n)^T \mapsto \overline{X^{x_1}Z^{z_1} \otimes X^{x_2}Z^{z_2} \otimes ... \otimes X^{x_n}Z^{z_n}}$$

is an isomorphism.

Proof. Given two vectors

$$v = (x_1, x_2, ..., x_n, z_1, z_2, ..., z_n)^T, w = (p_1, p_2, ..., p_n, q_1, q_2, ..., q_n)^T \in \mathbb{B}^{2n}$$

suppose that $f(v) = f(w)$. Then

$$\overline{X^{x_1}Z^{z_1} \otimes X^{x_2}Z^{z_2} \otimes ... \otimes X^{x_n}Z^{z_n}} = \overline{X^{p_1}Z^{q_1} \otimes X^{p_2}Z^{q_2} \otimes ... \otimes X^{p_n}Z^{q_n}},$$

so

$$X^{x_1}Z^{z_1} \otimes X^{x_2}Z^{z_2} \otimes ... \otimes X^{x_n}Z^{z_n} = \alpha X^{p_1}Z^{q_1} \otimes X^{p_2}Z^{q_2} \otimes ... \otimes X^{p_n}Z^{q_n}$$

for some $\alpha \in \{1, -1, i, -i\}$. This can only be true if $X^{x_i}Z^{z_i} = X^{p_i}Z^{q_i}$ for all i, which in turn means that $v = w$. Therefore f is injective. Furthermore, as the order of both groups is equal, we have a bijection.

Now

$$\begin{aligned} f(v + w) &= f((x_1 + p_1, x_2 + p_2, ..., x_n + p_n, z_1 + q_1, z_2 + q_2, ..., z_n + q_n)^T) \\ &= \overline{X^{x_1+p_1}Z^{z_1+q_1} \otimes X^{x_2+p_2}Z^{z_2+q_2} \otimes ... \otimes X^{x_n+p_n}Z^{z_n+q_n}} \\ &= \overline{X^{x_1}Z^{z_1} \otimes X^{x_2}Z^{z_2} \otimes ... \otimes X^{x_n}Z^{z_n}} \circ \overline{X^{p_1}Z^{q_1} \otimes X^{p_2}Z^{q_2} \otimes ... \otimes X^{p_n}Z^{q_n}} \\ &= f(v) \circ f(w). \end{aligned}$$

So indeed f is an isomorphism. □

This isomorphism can be used to treat n-qubit Pauli operators as vectors (up to a scalar), which allows us to speak of linear independence and use linear algebra to analyse Pauli operators.

Lemma 2.5.5. *Given two operators $S, T \in \mathcal{P}_n$, let $S = \alpha X^{x_1}Z^{z_1} \otimes X^{x_2}Z^{z_2} \otimes ... \otimes X^{x_n}Z^{z_n}$ and $T = \beta X^{p_1}Z^{q_1} \otimes X^{p_2}Z^{q_2} \otimes ... \otimes X^{p_n}Z^{q_n}$. S and T commute iff $\sum_{i=1}^{n} x_i q_i + \sum_{i=1}^{n} z_i p_i = 0 \mod 2$.*

Proof. By Proposition 2.5.3 we know that S and T commute iff an even number of $X^{x_i}Z^{z_i}$ and $X^{p_i}Z^{q_i}$ anti-commute.

We want to use the fact that for any $a, b \in \{0,1\}$ we have $X^aZ^b = (-1)^{ab}Z^bX^a$. This is true because if either a or b is equal to 0, then the corresponding Pauli is the identity, which commutes with everything. If both are equal to 1, then we have $XY = -YX$.

For a given $i \in \{1, ..., n\}$ $X^{x_i}Z^{z_i}$ and $X^{p_i}Z^{q_i}$ anti-commute iff

$$X^{x_i}Z^{z_i}X^{p_i}Z^{q_i} = -X^{p_i}Z^{q_i}X^{x_i}Z^{z_i}.$$

By our previously stated fact we have

$$\begin{aligned} X^{x_i}Z^{z_i}X^{p_i}Z^{q_i} = (-1)^{p_iz_i}X^{x_i}X^{p_i}Z^{z_i}Z^{q_i} &= (-1)^{p_iz_i}X^{p_i}X^{x_i}Z^{q_i}Z^{z_i} \\ &= (-1)^{p_iz_i+x_iq_i}X^{p_i}Z^{q_i}X^{x_i}Z^{z_i}. \end{aligned}$$

Therefore we know that $X^{x_i}Z^{z_i}$ and $X^{p_i}Z^{q_i}$ anti-commute iff $p_iz_i + x_iq_i = 1 \mod 2$. This is true for an even number of i iff $\sum_{i=1}^{n} x_iq_i + \sum_{i=1}^{n} z_ip_i = 0 \mod 2$, proving the lemma. □

Definition 2.5.6. *Given a unitary U and a state $|\psi\rangle$ we say that U stabilizes $|\psi\rangle$ or that U is a stabilizer of $|\psi\rangle$ if $|\psi\rangle$ is a $+1$-eigenvector of U:*

$$U|\psi\rangle = |\psi\rangle$$

Definition 2.5.7. *A stabilizer group $\mathcal{S}$ is an abelian subgroup of $\mathcal{P}_n$ that does not contain $-I$.*

Such a stabilizer group can be identified by a set of independent generators. We call a set of stabilizers independent if none can be written as a product of the others. This corresponds to linear independence of the binary vectors that are associated with the stabilizers by the isomorphism of Proposition 2.5.4.

Definition 2.5.8. *The stabilizer space of a stabilizer group $\mathcal{S} \subseteq \mathcal{P}_n$ is the set of vectors that are stabilized by all $S \in \mathcal{S}$.*

Example 2.5.9. *We take a look at the stabilizer group $S = \{II, IX, XI, XX\} \subseteq \mathcal{P}_2$. Clearly II stabilizes all vectors. Basis vectors for the space that is stabilized by IX are $|0\rangle|+\rangle$ and $|1\rangle|+\rangle$. For XI these are $|+\rangle|0\rangle$ and $|+\rangle|1\rangle$ and for XX we have $|+\rangle|+\rangle$ and $|-\rangle|-\rangle$. The only states that are in all of these spaces at the same time are of the form $\alpha|+\rangle|+\rangle$, so the stabilizer space of $\mathcal{S}$ is generated by $|+\rangle|+\rangle$.*

Lemma 2.5.10. *Given a stabilizer group $\mathcal{S} \subseteq \mathcal{P}_N$ with independent generators $S_1, S_2, ..., S_n$, where $n \leq N$, and a number $m < n$ we can find an operator $T \in \mathcal{P}_N$ that commutes with $S_1, ..., S_m$ and that anti-commutes with $S_{m+1}, ..., S_n$.*

Proof. Let $S_i = \alpha_i X^{x_{i,1}} Z^{z_{i,1}} \otimes X^{x_{i,2}} Z^{z_{i,2}} \otimes ... \otimes X^{x_{i,N}} Z^{z_{i,N}}$ for all $i \in \{1, ..., n\}$. By Lemma 2.5.5 we have to find an operator $T = \alpha X^{x_1} Z^{z_1} \otimes X^{x_2} Z^{z_2} \otimes ... \otimes X^{x_N} Z^{z_N} \in \mathcal{P}_N$ such that $\sum_{j=1}^{N} x_i z_{i,j} + \sum_{i=1}^{N} z_i x_{i,j} = 0 \mod 2$ for all $i \in \{1, ..., m\}$ and $\sum_{j=1}^{N} x_i z_{i,j} + \sum_{i=1}^{N} z_i x_{i,j} = 1 \mod 2$ for all $i \in \{m+1, ..., n\}$. As the S_i are independent, we know the vectors $(x_{i,1}, x_{i,2}, ..., x_{i,N}, z_{i,1}, z_{i,2}, ..., z_{i,N})^T, i \in \{1, ..., n\}$ are linearly independent. This means that we can find a solution $(x_1, x_2, ..., x_N, z_1, z_2, ..., z_N)^T$ for the system described above. This means that the operator $T = X^{x_1} Z^{z_1} \otimes X^{x_2} Z^{z_2} \otimes ... \otimes X^{x_N} Z^{z_N}$ satisfies the conditions. □

Proposition 2.5.11. *If stabilizer group $\mathcal{S} \subseteq \mathcal{P}_n$ is generated by m independent generators $S_1, ..., S_m$, then its stabilizer space has dimension 2^{n-m}.*

Proof. We prove this by induction on m. If $m = 0$, then $\mathcal{S}$ has no generators, meaning that $\mathcal{S} = \{I\}$. Therefore, every vector gets stabilized by $\mathcal{S}$, which means that the entire n-qubit space is the stabilizer space of $\mathcal{S}$. This has dimension 2^n, so we are done.

Now suppose $m > 0$ and that the stabilizer space of the group $\langle S_1, ..., S_{m-1} \rangle$ has dimension 2^{n-m+1}. For each $S \in \mathcal{S}$ we have $S^2 = I$ as otherwise $-I$ would be in $\mathcal{S}$. This means that the only possible eigenvalues for $I \neq S \in \mathcal{S}$ are ± 1. By Lemma 2.5.10 we can find an operator $T \in \mathcal{P}_n$ that commutes with $S_1, ..., S_{m-1}$, but anti-commutes with S_m. For every eigenvector $|\psi\rangle$ of S_m in the stabilizer space of $\langle S_1, ..., S_{m-1} \rangle$ that has eigenvalue $+1$ we can now find one with eigenvalue -1:

$$S_m T |\psi\rangle = -T S_m |\psi\rangle = -T |\psi\rangle .$$

This eigenvector $T |\psi\rangle$ is then still in the stabilizer space of $\langle S_1, ..., S_{m-1} \rangle$ as for $i < m$:

$$S_i T |\psi\rangle = T S_i |\psi\rangle = T |\psi\rangle .$$

Therefore, in the 2^{n-m+1}-dimensional stabilizer space of $\langle S_1, ..., S_{m-1} \rangle$ we can find exactly 2^{n-m} orthogonal $+1$-eigenvectors of S_m. This means that the stabilizer space of $\mathcal{S}$ has dimension 2^{n-m}. □

A special case of this proposition shows us that n independent stabilizers stabilize exactly one state (up to a scalar).

Definition 2.5.12. *A stabilizer state is a quantum state that is stabilized by some non-trivial stabilizer group.*

Stabilizer quantum mechanics is limited to only using stabilizer states. This means that we need to restrict the unitaries that we allow as well, as most unitaries do not preserve stabilizer states.

Definition 2.5.13. *The n-qubit Clifford group $\mathcal{C}_n$ is the normalizer of the Pauli group $\mathcal{P}_n$. That is, $\mathcal{C}_n = \{C | CPC^\dagger \in \mathcal{P}_n \; \forall P \in \mathcal{P}_n\}$.*

So in stabilizer quantum mechanics we only allow stabilizer states, Clifford unitaries and measurements in the computational basis.

In the ZX-Calculus we can apply the same restriction. This amounts to limiting the phases that we allow in our diagrams.

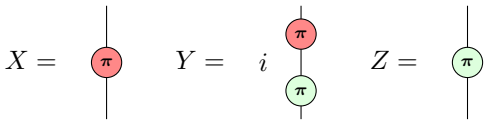
Definition 2.5.14. *A Clifford diagram is a ZX-diagram where every spider has a phase* $\alpha \in \{0, \frac{\pi}{2}, \pi, -\frac{\pi}{2}\}$.

This restriction of the ZX-Calculus corresponds to stabilizer quantum mechanics. Moreover, as mentioned before, this part of the ZX-Calculus is complete for stabilizer quantum mechanics [5].

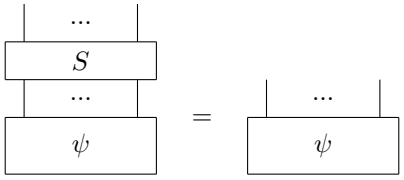
2.6 Transitioning between ZX-Calculus and Stabilizer theory

The goal of this thesis is two-fold. First, we want to have a direct way to find n generating stabilizers for a given n-qubit Clifford ZX-state. Second, given a stabilizer group $G \subseteq \mathcal{P}_n$ with n independent generators, we want to be able to find an explicit ZX-diagram that represents a state that is stabilized by G. Before we discuss these two problems themselves, we first look at how we can translate stabilizer theory into the ZX-Calculus.

To speak of stabilizers in the ZX-Calculus, we first need the equivalents of the Pauli operators. We have the following equalities:



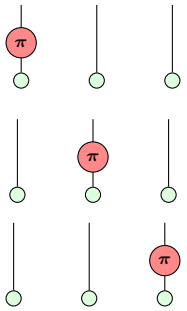
The fact that some operator S stabilizes a state $|\psi\rangle$ can simply be expressed as follows:



Example 2.6.1. *The state*

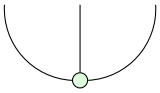
$|\psi\rangle =$

has generating set of independent stabilizers $\{X \otimes I \otimes I, I \otimes X \otimes I, I \otimes I \otimes X\}$ *as the following three diagrams*

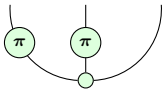


are all equal to $|\psi\rangle$ *by rule (c).*

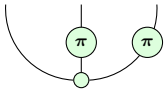
The state

$|\phi\rangle =$ 

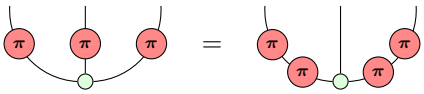
is stabilized by the group generated by $\{Z \otimes Z \otimes I, I \otimes Z \otimes Z, X \otimes X \otimes X\}$ *because both*



and



can be shown to be equal to $|\phi\rangle$ *by fusing the spiders with rule (f) and we have*

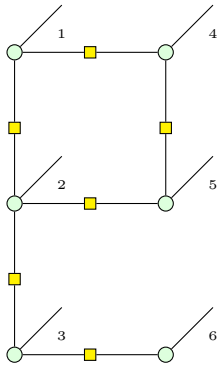


by copying with rule (p1), which is equal to $|\phi\rangle$ *because we can fuse with (f) and then apply (i1).*

We already know of a way to find stabilizers for a given ZX-state. This method uses so-called graph states and local Clifford operations to get a ZX-diagram in a form from which it is easy to see what the stabilizers are. We take a brief look at how this works.

Definition 2.6.2. *A graph state is a ZX-diagram that consists of Z-spiders with phase* 0 *that are connected by edges with a Hadamard gate on it. There are no parallel edges or self-loops and every Z-spider is connected to exactly one output and no inputs.*

Example 2.6.3. *The following ZX-diagram is a graph state:*

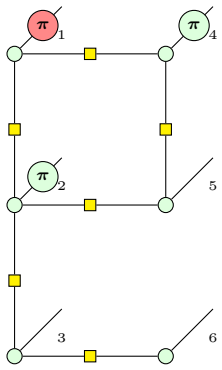


Here we have labeled the outputs $1, ..., 6$.

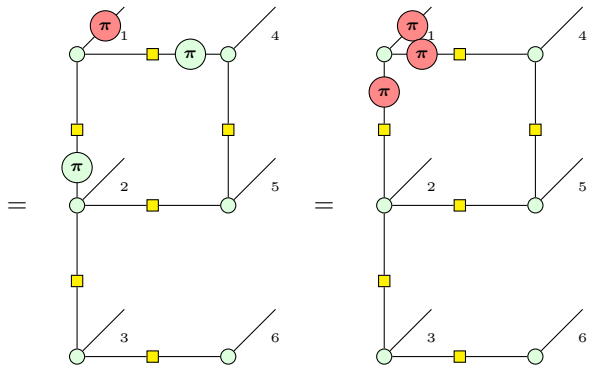
Definition 2.6.4. *A local Clifford operation on n qubits is an operation in* $\mathcal{C}_1^{\otimes n}$.

It has been shown in [9] that we can transform a ZX-diagram into a graph state up to a local Clifford operation. To see how we find stabilizers for such a state, we first illustrate how to find them for a graph state.

For each of the Z-spiders we can put an X on its output wire and a Z on all of its neighbours' and obtain a stabilizer. To see why this works we look at the graph state of Example 2.6.3. If we pick the first Z-spider to put an X on the output and a Z on its neighbours'. We get the following:



Here we can push the two Z's towards the first Z-spider and change their colour:



By copying one of the X's and fusing them we get back to our original state, showing that $X \otimes Z \otimes I \otimes Z \otimes I \otimes I$ is indeed a stabilizer. Repeating the process for all Z-spiders gives us n independent stabilizers, which generate the group that only stabilizes the given state.

To find stabilizers for a graph state with a local Clifford operation C applied to it, we can simply replace any stabilizer S for the graph state with $CSC^\dagger$, which will be a stabilizer for the whole state. Moreover, as Cliffords preserve Pauli's, we still have a generating set for a stabilizer group.

Our goal here is to completely bypass graph states and give a direct way to find stabilizers for any Clifford state. We want this method to be constructive and to lead to an easy algorithm. In order to achieve this we will only transform a Clifford state to a minimal degree before using its properties to discover what operators stabilize it.

Going the other way, from stabilizers to a ZX-diagram, we can also use graph states. There are ways to construct graph states from a given set of stabilizers, as [7] indirectly shows, however, it does not provide an easy constructive algorithm. Our goal is to again pass by graph states and produce a regular ZX-diagram that is stabilized by a given set of stabilizers. Our method is somewhat similar to and in the same spirit as [10] and [6], where maps to encode stabilizer codes are constructed.

3 ZX to Stabilizers

The aim of this section is to create an algorithm that can take in the ZX-diagram of any Clifford state and derive a generating set of stabilizers for it. We start off by discussing graph-like diagrams, which are the standard form we will want a diagram to be in before looking for stabilizers. Then we produce an algorithm to find stabilizers in the simplified case where all spiders in the diagram have phase 0. Lastly we generalize the solution to work for all Clifford states.

3.1 Graph-like diagrams

In order to retrieve the stabilizers from a given Clifford state, we will first put it in a standard form that is useful for us. This form is similar to the one used in [2], but differs slightly from it to suit our needs.

Definition 3.1.1. *A ZX-diagram is called graph-like if*

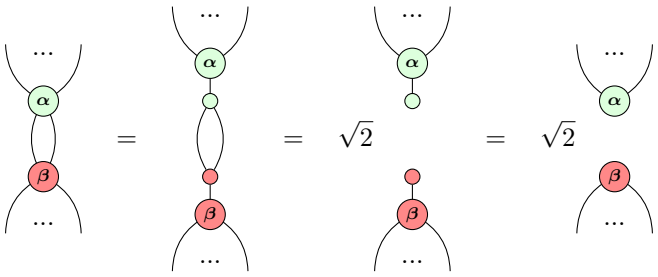
1. *No two connected spiders have the same colour.*
2. *There are no parallel edges or self loops.*
3. *Every input or output is connected to a phaseless Z-spider and every phaseless Z-spider is connected to at most one input or output.*

Transforming a ZX-diagram into this form enables us to think of it as a graph. This gives us some tools which will simplify the process of finding the stabilizers of a given diagram. We can transform every ZX-diagram to be graph-like by the following lemma.

Lemma 3.1.2. *Every ZX-diagram is equal to a graph-like ZX-diagram.*

Proof. Given a ZX-diagram, we first fuse all adjoining same-coloured spiders together using rule (f). This gives us a diagram where no two connected spiders have the same colour.

Next we can remove any occurrence of a parallel edge by applying Lemma 2.4.4 as follows:

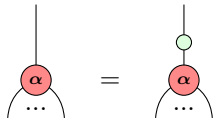


Self loops can be eliminated as we have:

This also holds for red spiders, so we end up with a diagram that has no self loops.

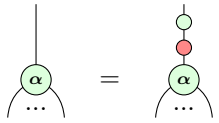
Now we ensure that property 3 is met whilst preserving properties 1 and 2. We do this by adding a green spider to every input or output connected to a red

spider using the identity rule (i1):



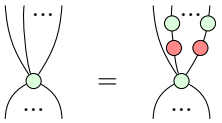
This does not introduce any same coloured connected spiders, nor do we obtain any parallel edges or self loops, so the first two properties are still valid.

For inputs or outputs connected to a green spider with a nonzero phase we do something similar:



Again, we preserve properties 1 and 2.

All that is left is to ensure that every spider is connected to at most one input or output. As we have already enforced that every input or output is connected to a green phaseless spider, we only have to look at the case where multiple inputs or outputs are connected to a single green phaseless spider. In that case we can do the same as we did before to all but one of the input or output wires, giving us:



Which ensures that property 3 is met, whilst also preserving properties 1 and 2. Therefore we have obtained a graph-like ZX-diagram that is equal to the original diagram.

□

Remark. Note that when finding stabilizers for a given ZX-state, scalars are unimportant because if $S|\psi\rangle = |\psi\rangle$ then also $S\alpha|\psi\rangle = \alpha|\psi\rangle$. This means that we will often drop the scalar that is induced by removing parallel edges, as this will not have any effect on the stabilizers we will find.

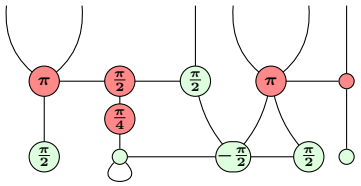
As the proof of Lemma 3.1.2 is constructive, we can use it to create an algorithm to put a ZX-diagram in the desired standard form. The method works for any ZX-diagram, but we are only interested in Clifford ZX-states, so this algorithm only regards those cases.

Algorithm 1: Finding an equal graph-like diagram

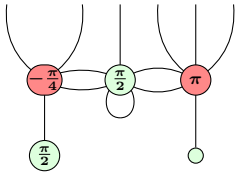
input : Clifford ZX-state D
output: Graph-like Clifford ZX-state that is equal to D up to a scalar
Fuse all the adjoining same-coloured spiders in D;
Remove all parallel edges in D;
Remove all self-loops in D;
Add a green spider to every output wire in D that is connected to a red spider;
Add a green and a red spider to every output wire in D that is connected to a green spider with a nonzero phase;
Add a green and a red spider to every output wire in D that is connected to a green spider with more than one output wire;
return D;

The correctness of this algorithm follows directly from the proof of Lemma 3.1.2.

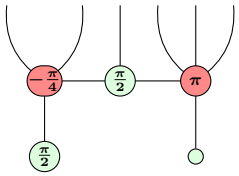
Example 3.1.3. *Consider the following ZX-diagram:*



This diagram is not graph-like, as it has a self loop and there are green spiders connected to green spiders as well as red spiders to red spiders. When we merge all the same coloured spiders together, we get:

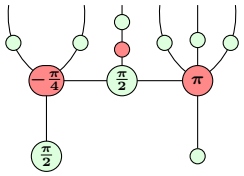


This has created some parallel edges which we still have to get rid of. As we have seen, we can simply remove parallel edges and self loops in the ZX-Calculus, leaving us with:



What remains is to ensure that the outputs are connected to green phaseless

spiders:

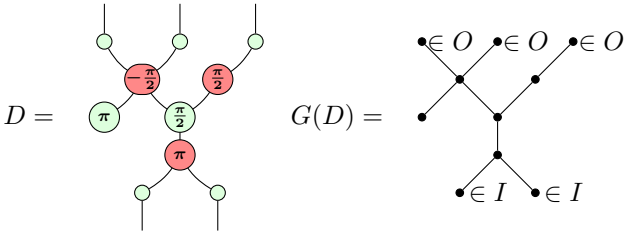


Resulting in a graph-like diagram that is equal to the original diagram (up to a scalar).

Definition 3.1.4. *An open graph is a triple (G, I, O), where $G = (V, E)$ is an undirected graph and I and O are subsets of V. We call I the set of inputs and O the set of outputs of G.*

Definition 3.1.5. *Given a graph-like ZX-diagram D, the underlying open graph $G(D)$ is an open graph with vertices that correspond to the spiders and edges that correspond to the wires of D. The sets I and O are exactly those vertices that correspond to spiders that are connected to the inputs or outputs respectively.*

Example 3.1.6. *Below we see an example of a graph-like ZX-diagram together with its underlying graph:*



With the help of the underlying graph we can now apply graph theory to graph-like ZX-diagrams. This will help us analyse the problem of finding stabilizers.

Remark. When discussing a certain graph-like ZX-diagram D we might implicitly use some of its underlying graphs properties. For example $x : (x, y) \in E$ will be understood to mean the spiders x that are connected to the spider y in diagram D.

3.2 Phaseless States

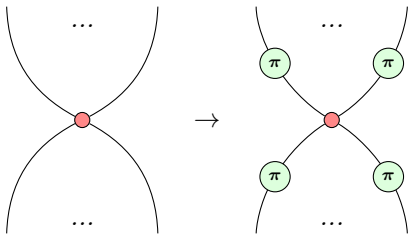
We want to find stabilizers for any given Clifford state. However, to simplify things, we start off by only allowing spiders with phase zero in the diagrams.

Definition 3.2.1. *A ZX-diagram is called phaseless if all of its spiders have phase* 0.

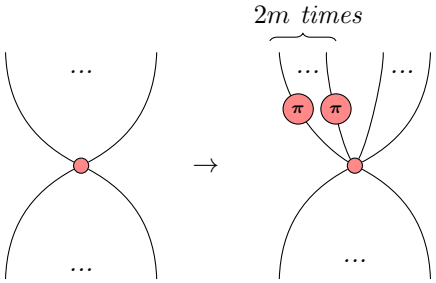
We will show how to find the stabilizers of a phaseless ZX-state and then later expand the method to work for any Clifford state.

The central property in this method for constructing stabilizers is that we can *fire* a spider in either colour as follows.

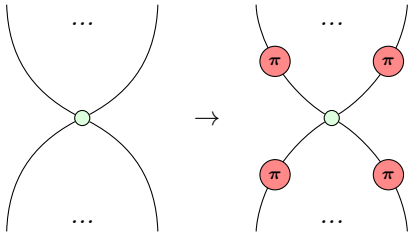
Definition 3.2.2. *Given a red node v, we say that we fire v in green when we put a green π on all of its legs:*



We say that we fire v in red when we put a red π on an even number of its legs:

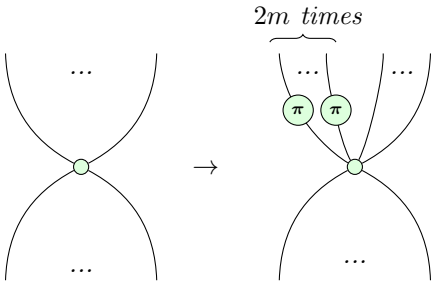


Analogously, given a green node w, we that that we fire w in red when we put a red π on all of its legs:



Lastly, we say that we fire w in green when we put a green π on an even number

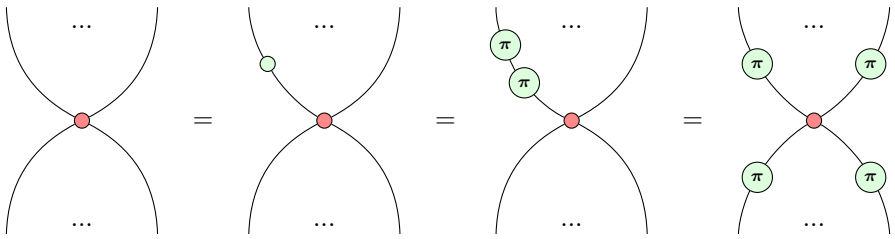
of its legs:



This definition of firing will prove to be very useful for us when we look for stabilizers for a Clifford diagram. The first reason for this is that firing nodes in a diagram does not change the underlying state, as seen in the next proposition.

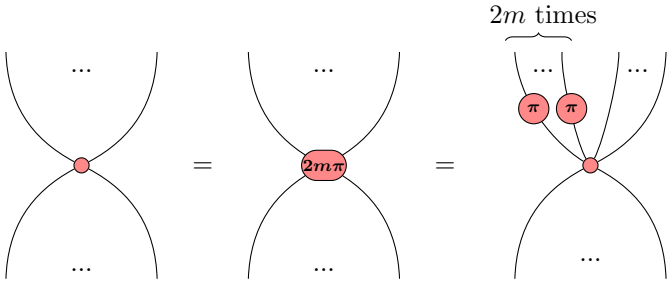
Proposition 3.2.3. *Firing any spider in a phaseless ZX-diagram is allowed in the ZX-Calculus.*

Proof. For any X-spider we have



showing that firing a red spider green is allowed in the ZX-Calculus. Flipping all the colours provides us with a proof that firing a green node red also preserves the diagram.

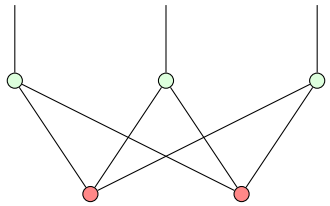
Next, firing a red spider red doesn't change anything by spider fusion



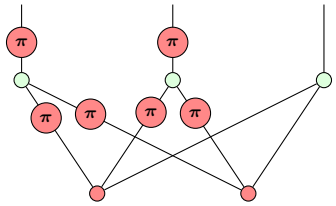
Again, flipping the colours also proves the case where we fire a green node green. □

Firing a spider, either in its own or in the opposite colour, does not change the state itself, but it does introduce Pauli's, which we will use to find the stabilizers of the state.

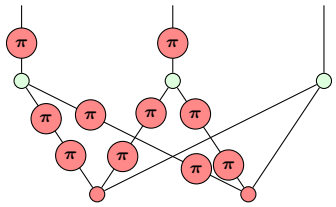
Example 3.2.4. *When we have the state*



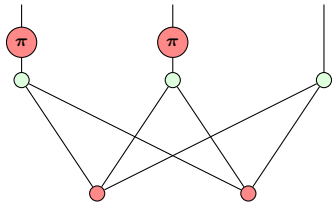
We can fire the leftmost and the middle Z-spider in red to obtain



Now we fire the bottom two X-spiders in red, by putting a red π on their two leftmost legs.



These new π's cancel with the ones that were already there from the previous step by fusing them with rule (f) and then applying (i1), so we get



By Proposition 3.2.3 we now see that the original state is equal to itself with $X \otimes X \otimes I$ applied to it. In other words, we found $X \otimes X \otimes I$ as a stabilizer for this state by firing certain spiders.

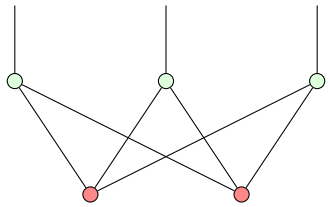
This example shows us that firing spiders can help us discover stabilizers, but some questions still remain. How do we decide which spiders to fire? Can all stabilizers be found with the help of firing? To answer the these questions we take a look at what needs to happen if we are to find a stabilizer by firing some spiders and leaving others be. To end up with a stabilizer, we need to find the original state back, plus some Pauli's on the output wires. This means that we

need all the internal wires to not have any left over π-gates on them. We are certain to get rid of all the gates on these wires if every two neighbours fire in the same colour. In that case the π's they put on their wires get cancelled out, as seen in the above example. So if a certain node x fires in the opposite colour, then its neighbour y must fire in that colour too, putting a π on the wire to x. As we assume the diagram to be graph-like, we know that this neighbour must fire in its own colour. Because of how firing in the same colour works, we see that y must have an even number of neighbours that fire in their opposite colour. Only then can y counteract this by putting an even number of π's on its wires. This holds for any non-output node y: we need an even number of its neighbours to fire.

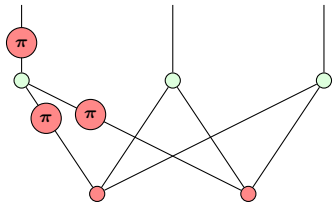
For output nodes something similar holds. The output wires are the only wires where we allow left over π's, as these spiders will give us the stabilizers we are looking for. Therefore if an odd number of neighbours of a output node fire in their opposite colour (which is necessarily green, as we are dealing with graph-like diagrams), this can still be counteracted by the output node. It simply fires green in all the directions of those firing neighbours and additionally puts a green π on the output wire, making the total number of green π's it put on the wires even again. If an even numbers of its neighbours fire green, it can fire in those directions and not put a green π on the output wire.

In the next example we will see what goes wrong when an odd number of neighbours of some node fire in the opposite colour.

Example 3.2.5. *Starting again from the state*

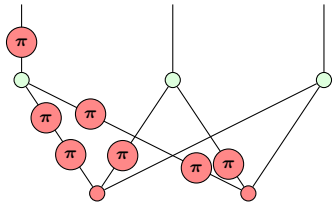


When we fire only the left green node, we get

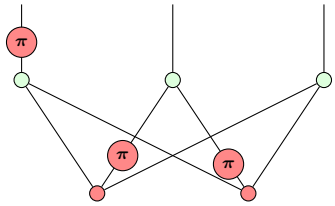


To compensate for the π's on the internal wires, we need to fire the red spiders

in red, but we cannot do that on just one leg, so we would get, for instance



Which is the same as



Where we see that we are left with red π's on the wires. The two red nodes had only one neighbour fire in red, which makes them unable to clear the wires again.

We want to keep track of all the possible combinations of green and red π's that can be put on the output wires without changing the rest of the diagram, as this will directly give us stabilizers. To do this we introduce binary variables $\mathtt{r}_i$ and $\mathtt{g}_i$ representing whether or not a red or a green π has been put on the i^{th} output wire respectively. So in Example 3.2.4, we would have $\mathtt{r}_1 = 1$, $\mathtt{r}_2 = 1$ and $\mathtt{r}_3 = 0$, because the first and second output wires get a red π. As there are no green π's, we have $\mathtt{g}_1 = \mathtt{g}_2 = \mathtt{g}_3 = 0$.

Next, we also have to keep track of which spiders fired in the opposite colour and which did not. As we are dealing with a graph-like ZX-diagram we know that the nodes that are connected to the output are all green spiders. The only way to get a red π on an output wire by firing spiders is to have the corresponding output spider fire red. Therefore, we already have a variable that tells us whether or not the i^{th} output spider fired, namely $\mathtt{r}_i$. For all other nodes $q_j \in V \setminus O$ we introduce a new binary variable $\mathtt{q}_j$ that models if the corresponding node has fired in its opposite colour.

Remark. There is a natural way to order the output nodes of a graph-like ZX-diagram, simply looking at which of the n output wires they are connected to. In assigning variables $\mathtt{q}_1, ..., \mathtt{q}_m$ to the m remaining internal nodes, we are implicitly using an ordering of these nodes. We will not distinguish between any such orderings and always choose an arbitrary one as everything we do will be completely independent of which ordering is used. To simplify things, we slightly abuse notation and write $\mathtt{v}$ for the binary variable associated to a node v and vice versa. So r_i is the node that corresponds to binary variable $\mathtt{r}_i$, which

is the i^{th} output node, and q_j is the node that corresponds to the variable $\mathtt{q}_j$ by the implicit ordering we picked.

Definition 3.2.6. *Given an n-qubit phaseless graph-like ZX-state D with $m+n$ spiders, where $r_1, ..., r_n$ are output spiders and $q_1, ..., q_m$ are internal spiders, we call $\vec{v} = (\mathtt{g}_1 \, ... \, \mathtt{g}_n \, \mathtt{r}_1 \, ... \, \mathtt{r}_n \, \mathtt{q}_1 \, ... \, \mathtt{q}_m)^T \in \mathbb{F}_2^{2n+m}$ a firing assignment vector for D.*

What we want to do when we are given a firing assignment vector for a certain diagram D is first to fire all green spiders $x \in D$ in red if we have $\mathtt{x} = 1$ in $\vec{v}$. Then we want to fire red spiders $y \in D$ in red by putting a red π on every edge where one of its neighbours already put one; so on every wire leading to a green node x with $\mathtt{x} = 1$. Next, we want to fire all red nodes $y \in D$ in green if $\mathtt{y} = 1$ in $\vec{v}$ and then fire the green spiders by putting a green π on every wire leading to a red node y with $\mathtt{y} = 1$ as well as on the i^{th} output wire if $\mathtt{g}_i = 1$. This series of firings is only allowed if it complies with Definition 3.2.2. Therefore we need same-colour firings to occur on an even number of legs. This means that we need some conditions to hold in order for $\vec{v}$ to lead to something useful.

Definition 3.2.7. *A firing assignment vector $\vec{v}$ for a diagram D is called valid if for every internal node $q \in V \setminus O$ we have*

$$\sum_{x:(q,x)\in E} \mathtt{x} = 0 \mod 2$$

and for every output node $r \in O$

$$\mathtt{r} + \sum_{x:(r,x)\in E} \mathtt{x} = 0 \mod 2$$

We can write these conditions more concisely in matrix form.

Definition 3.2.8. *Given an n-qubit phaseless graph-like ZX-state D we define the firing verification matrix M_D of D to be:*

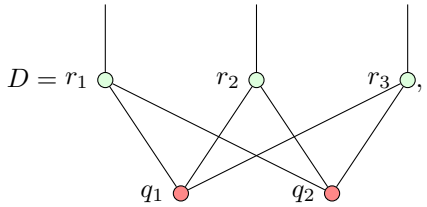
$$M_D \coloneqq \left(\begin{array}{c|c} I_n & \\ \emptyset & N \end{array}\right)$$

where N is the adjacency matrix of the underlying graph of D, using the implicit ordering of the nodes of D.

Lemma 3.2.9. *A firing assignment vector $\vec{v}$ for D is valid if $M_D\vec{v} = \vec{0}$.*

Proof. Writing out the system of linear equations we see that we get exactly the constraints described in Definition 3.2.7, as the adjacency matrix N of D contains a 1 for every neighbour of a node. □

Example 3.2.10. *We take a look at the state from the previous example and for once we explicitly give it some labeling.*



Now we have $\vec{v} = (\mathtt{g}_1\, \mathtt{g}_2\, \mathtt{g}_3\, \mathtt{r}_1\, \mathtt{r}_2\, \mathtt{r}_3\, \mathtt{q}_1\, \mathtt{q}_2)^T$ *and we get the following adjacency matrix for* D

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

Adding three columns for the output wires we get

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

Setting $M_D\vec{v} = \vec{0}$ *ensures that* $\vec{v}$ *is a valid firing assignment.*

Theorem 3.2.11. *If a vector* $\vec{v} = (\mathtt{g}_1\, \mathtt{g}_2 \ldots \mathtt{g}_n\, \mathtt{r}_1\, \mathtt{r}_2 \ldots \mathtt{r}_n\, \mathtt{q}_1\, \mathtt{q}_2 \ldots \mathtt{q}_m)^T$ *is a valid firing assignment for a phaseless graph-like ZX-state* D*, then* $X^{\mathtt{r}_1}Z^{\mathtt{g}_1} \otimes X^{\mathtt{r}_2}Z^{\mathtt{g}_2} \otimes \ldots \otimes X^{\mathtt{r}_n}Z^{\mathtt{g}_n}$ *is a stabilizer for* D*.*

Proof. Given fining assignment vector $\vec{v}$, we fire all green nodes $x \in V$ with $\mathtt{x} = 1$ in red. Next we fire the red nodes in red in every direction where its neighbour has corresponding binary value 1. This doesn't change the state and we are ensured by $M_D\vec{v} = \vec{0}$ that every red node has an even number of neighbours with value 1, so this firing is allowed. This leaves us with an X-spider on the output wires where $\mathtt{r}_i = 1$, as r_i fired red in that direction. Nothing else changed in the diagram of D, as every other wire has either no red π's on it, in case the neighbouring green node didn't fire, or two red π's, in case the green node did fire. Now we do the same for firing in green. We fire all red nodes $y \in V$ with $\mathtt{y} = 1$ green and fire the green nodes green in every direction where its neighbour has value 1 and if $\mathtt{g}_i = 1$ also on the i^{th} output wire. The same argument shows that we end up with the original state, with a Z-spider on any output wire where $\mathtt{g}_i = 1$, followed by an X-spider whenever $\mathtt{r}_i = 1$. □

We now know that any solution to $M_D\vec{v} = \vec{0}$ gives us a stabilizer for our state. What we would like is that every stabilizer can be found this way. That would

mean that finding all the stabilizers for a given state can be reduced to solving a system of linear equations. This turns out to be the case, as seen in the following theorem.

Theorem 3.2.12. *Every stabilizer S of a phaseless graph-like ZX-state D corresponds to at least one valid firing assignment $\vec{v}$ of D.*

Proof. Let n be the number of outputs of D. We know that linearly independent vectors $(z_1\, z_2 \dots z_n\, x_1\, x_2 \dots x_n)^T$ correspond to independent operators $X^{x_1}Z^{z_1} \otimes X^{x_2}Z^{z_2} \otimes \dots \otimes X^{x_n}Z^{z_n}$. So if we find n valid firing assignment vectors for $|\psi\rangle$ that are independent of each other when restricted to the first $2n$ entries $(\mathtt{g}_1\, \mathtt{g}_2 \dots \mathtt{g}_n\, \mathtt{r}_1\, \mathtt{r}_2 \dots \mathtt{r}_n)^T$, we know that we found n independent stabilizers and therefore we found them all by Proposition 2.5.11. So we will look at how many different stabilizers we can find using Theorem 3.2.11.

As M_D contains the adjacency matrix of the underlying graph of D we know something about its structure. We can use the fact yhat output nodes are not connected to each other, as D is graph-like. We get the following general form

$$M_D = \begin{pmatrix} I & \emptyset & A^T \\ \emptyset & A & B \end{pmatrix}$$

where $B = B^T$. The dimension of the solution space to $M_D\vec{v} = \vec{0}$ is the dimension of the kernel of M_D. We have

$$\begin{aligned} \dim(\ker(M_D)) &= 2n + m - \operatorname{rank}(M_D) \\ &= 2n + m - (n + \operatorname{rank}(\begin{pmatrix} A & B \end{pmatrix})) \\ &= n + m - \operatorname{rank}(\begin{pmatrix} A^T \\ B \end{pmatrix}) \\ &= n + \dim(\ker(\begin{pmatrix} A^T \\ B \end{pmatrix})) \end{aligned}$$

The solutions we are interested in are the solutions where the first $2n$ entries are not all zero, or in other words, solutions that lead to a non-trivial stabilizer. The ones we are not interested in, solutions starting with $2n$ zeroes, are of the form $\vec{v} = (0^n\, 0^n\, \vec{q})^T$. We then have

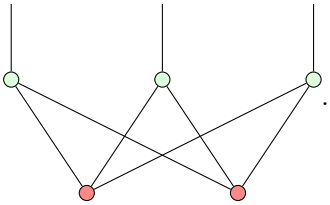
$$\vec{0} = M_D\vec{v} = \begin{pmatrix} A^T \\ B \end{pmatrix} \vec{q}.$$

So $\vec{q} \in \ker(\begin{pmatrix} A^T \\ B \end{pmatrix})$, so if we pick a basis $\{\vec{q_1}, \vec{q_2}, ..., \vec{q_d}\}$ for $\ker(\begin{pmatrix} A^T \\ B \end{pmatrix})$, then the vectors $\vec{b_i} = (0^{2n}\, \vec{q_i})^T \in \ker(M_D)$ are linearly independent and we can expand this to a basis of $\ker(M_D)$ by adding n more independent vectors, which cannot have zeroes as their $2n$ first entries, as otherwise they would be dependent of the $\vec{b_i}$'s. This means that we have found n vectors of which the first $2n$ entries are linearly independent. Therefore, with the previous result, we have found n

independent stabilizers for D, showing that all stabilizers are found using this method. □

In the proof of this theorem we encountered solutions that we do not care about: solutions starting with $2n$ zeroes. In the next example we see what these solutions are and why we get them.

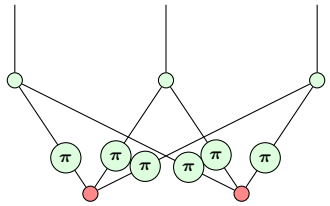
Example 3.2.13. *We start again with the state*

$D =$  .

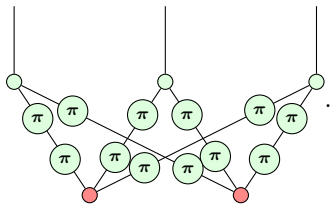
We already saw what M_D looks like in Example 3.2.10. From this we see that

$$\begin{pmatrix} A^T \\ B \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

Now we have $\vec{q} = (1\,1)^T \in \ker \begin{pmatrix} A^T \\ B \end{pmatrix}$), so we get a trivial solution $\vec{b_1} = (0\,0\,0\,1\,1)^T$. This corresponds to firing the bottom two red spiders in green



followed by firing the green nodes on their two bottom wires in green too

 .

All the π's cancel out and we are left with D. So we see that $\vec{b_1}$ is indeed a solution that doesn't give us a stabilizer; it has only internal firings and does not leave anything on the output wires as a result.

These last two theorems directly lead to a relatively simple algorithm to find stabilizers for a given phaseless Clifford state D.

Algorithm 2: Finding stabilizers for a phaseless ZX-state

input : Phaseless ZX-state D
output: Generating set of stabilizers for the group consisting of all the stabilizers of D
Let D' be the outcome of Algorithm 1 when applied to D;
Let n be the number of output wires of D;
Find a basis $\mathcal{B} = \{\vec{b_1}, ..., \vec{b_{n+d}}\}$ for the solution space of $M_{D'}\vec{v} = \vec{0}$;
Pick a set $\mathcal{B}'$ of n vectors from $\mathcal{B}$ such that they are independent when restricted to the first $2n$ entries;
return *The set of stabilizers that correspond to the vectors in $\mathcal{B}'$ as in Theorem 3.2.11*;

Correctness of this algorithm can be shown by first noting that $\mathcal{B}$ consists of valid firing assignment vectors of D', which is a graph-like phaseless ZX-state because Algorithm 1 does not introduce any phases. Next we know from Theorem **??** that $\mathcal{B}$ must include vectors that correspond to a generating set of stabilizers for the group of all the stabilizers of D'. This means that we can indeed find n vectors that are independent in the first $2n$ entries. These directly correspond to such a generating set by Theorem 3.2.11 and as D' and D are equal up to a scalar, we have found generating stabilizers for D as well.

Example 3.2.14. *For the state D of the last examples we found the matrix M_D in example 3.2.10. Solving $M_D\vec{v} = \vec{0}$ gives us*

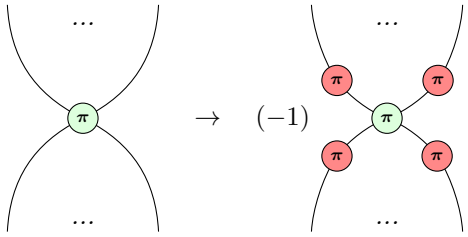
$$\mathcal{B} = \{ \begin{pmatrix} 1\\1\\1\\0\\0\\0\\1\\0 \end{pmatrix}, \begin{pmatrix} 0\\0\\0\\1\\1\\0\\0\\0 \end{pmatrix}, \begin{pmatrix} 0\\0\\0\\0\\1\\1\\0\\0 \end{pmatrix}, \begin{pmatrix} 0\\0\\0\\0\\0\\0\\1\\1 \end{pmatrix} \}$$

Where the last vector is the uninteresting internal solution we found in Example 3.2.13. Reducing to the first six entries we see that the other three are independent and therefore give us a generating set of stabilizers $Z \otimes Z \otimes Z$, $X \otimes X \otimes I$ and $I \otimes X \otimes X$.

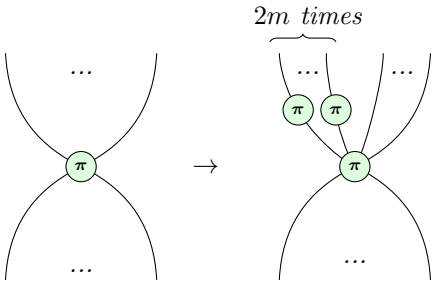
3.3 Clifford States

Now that we have seen how we can find the stabilizers for a phaseless ZX-state, it is time to generalize out solution to all Clifford states. We will still work with the basic concept of firing nodes in a certain colour. However, firing nodes with a phase has a little more going on than we have seen thus far.

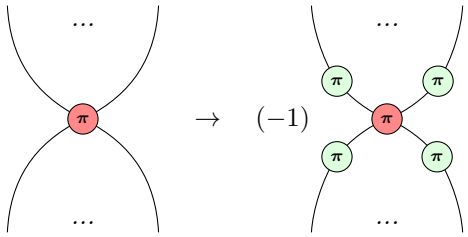
Definition 3.3.1 (Firing π-spiders)**.** *Given a green π-spider x we say that we fire x in red when we put red π's on all of its wires, while multiplying by a phase of -1:*



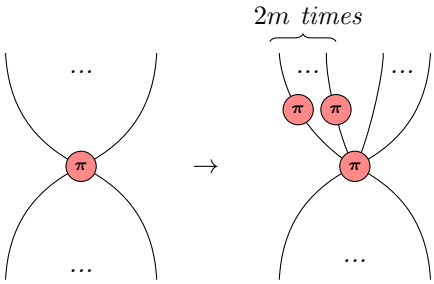
We say that we fire x in green when we put a green π on an even number of its wires.



Analogously, for a red π-spider y we say that we fire y in green when we put green π's on all of its wires, while multiplying by a phase of -1.



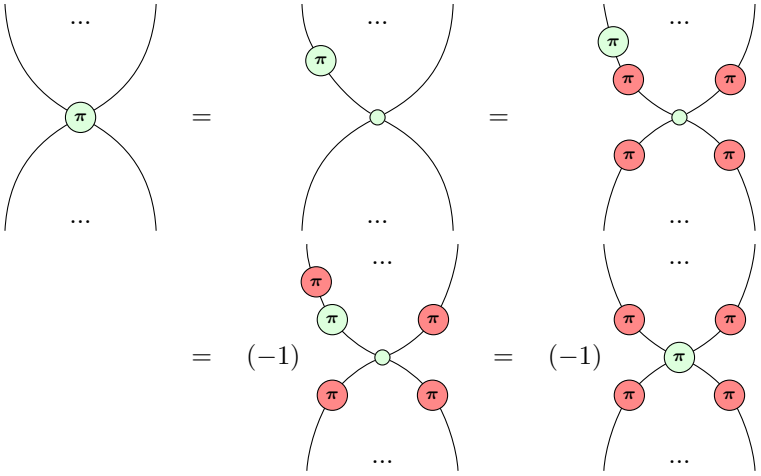
Last, we say that we fire y in red when we put an even number of red π's on its wires.



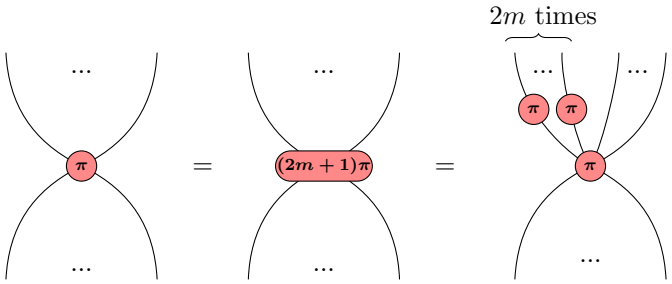
Just as in the phaseless case these definitions make sense in the way that we can fire π-spiders without changing the meaning of the diagram. This follows from the next proposition.

Proposition 3.3.2. *Firing any π-spider in a Clifford diagram is allowed in the ZX-Calculus.*

Proof. For any green π-spider we have



Which shows that firing a green π-spider red is allowed in the ZX-Calculus. Again, the proof for firing a red π-spider green is obtained by flipping the colours. Next, similar to the phaseless case, for any red π-spider we have

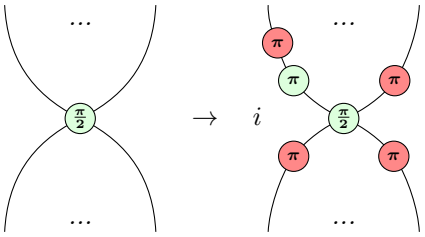


Where again, flipping the colours proves the green case. □

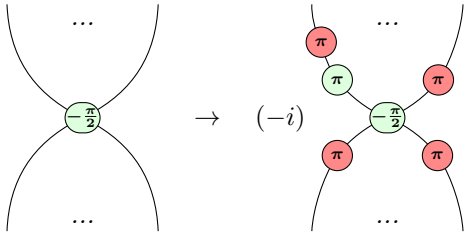
Now we know how to fire red and green nodes with phase 0 or π. What remains is to define firing for the $\pm\frac{\pi}{2}$-phase case.

Definition 3.3.3 (Firing $\pm\frac{\pi}{2}$-spiders)**.** *Given a green $\frac{\pi}{2}$-spider x, we say that we fire x in red when we put red π's on all of its legs, followed by one green π*

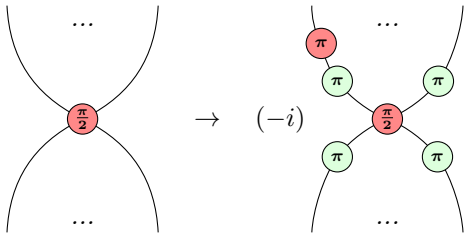
on one of its legs and we multiply by a factor i.



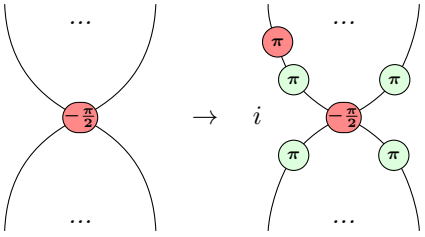
For a green $-\frac{\pi}{2}$-spider x, we say that we fire x in red when we put red π's on all of its legs, followed by one green π on one of its legs and we multiply by a factor $-i$.



In a similar way, but slightly different, for a red $\frac{\pi}{2}$-spider y we say that we fire y in green when we put a red π on one of its legs, followed by green π's on all of its legs and we multiply by a factor of $-i$.

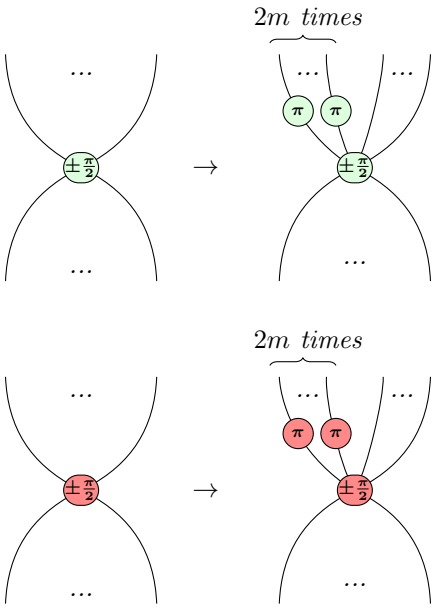


Firing a red $-\frac{\pi}{2}$-spider is the same, but multiplying by a factor i instead.



Lastly firing red and green $\pm\frac{\pi}{2}$-spiders in their own colour is the same as in the

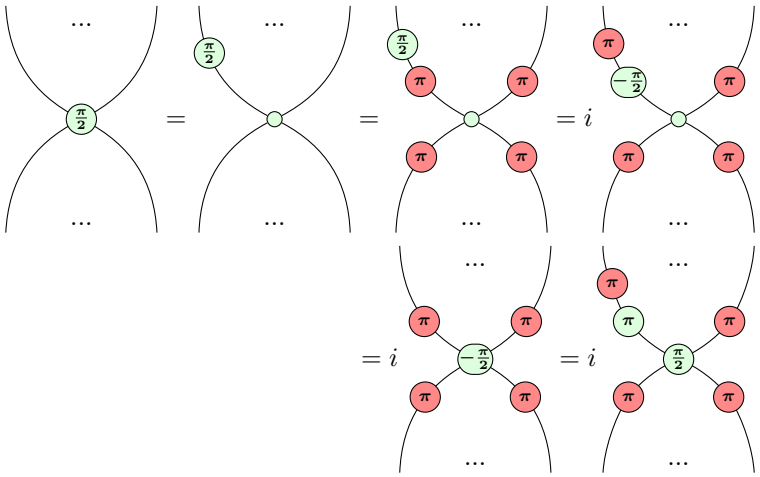
other phase cases; we put an even number of same coloured π's on its wires.



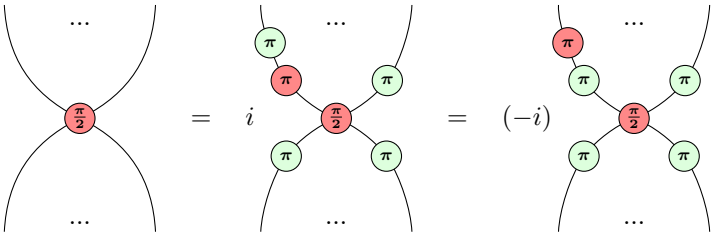
Of course, the reason that firing $\pm\frac{\pi}{2}$-spiders gets a little more complicated is to make sure that we can still fire all nodes without changing the meaning of the diagram we are working on. We show that this is indeed the case in the next proposition. There is a slight difference in firing red and green nodes in the opposite colour to make keeping track of the phases easier later on.

Proposition 3.3.4. *Firing any $\pm\frac{\pi}{2}$-spider in a Clifford diagram is allowed in the ZX-Calculus.*

Proof. For any green $\frac{\pi}{2}$-phase spider we have



Swapping the colours almost completes the proof for the red case, we only need to switch the red and green π around.



The proof for $-\frac{\pi}{2}$-spiders is very similar. In the green case we get a factor $-i$ instead of i because switching a green $-\frac{\pi}{2}$ and a red π in the third step introduces $-i$. In the red case, we have the same phase difference occuring for that reason.

Lastly, firing in the same colour has the same proof as in Propositions 3.2.3 and 3.3.2 as adding an even number of π's does not change the phase of the spider. □

We will approach finding stabilizers for a given state the same way we did in the phaseless case. Because different kinds of spiders show different behaviour when fired, we need to keep track of the types of spiders we have. Therefore, for a graph-like diagram D we split its set of nodes V up even further. We already had the set $O \subseteq V$ of output nodes and introduce $P, H \subseteq V \setminus O$, where P is the set of spiders with phase π and H the set of spiders with phase $\pm\frac{\pi}{2}$. The set H can then be split in two, according to whether firing a spider introduces $+i$ or $-i$. We take $H^+ \subseteq H$ to be the set of green phase $\frac{\pi}{2}$ and red phase $-\frac{\pi}{2}$ spiders and $H^- \subseteq H$ to consist of the red phase $\frac{\pi}{2}$ and green phase $-\frac{\pi}{2}$ spiders.

Again, we want to use binary variables to keep track of which nodes to fire and which to leave alone. To this end we expand the definition of firing assignment vector to the general Clifford case.

Definition 3.3.5. *Given an n-qubit graph-like Clifford ZX-state D with $n+m+r$ spiders, where $r_1, ..., r_n$ are output spiders, $q_1, ..., q_m$ are internal spiders with phases 0 or π and $h_1, ..., h_r$ are internal spiders with phase $\pm\frac{\pi}{2}$, we call $\vec{v} = (\mathtt{g}_1 \dots \mathtt{g}_n \, \mathtt{r}_1 \dots \mathtt{r}_n \, \mathtt{q}_1 \dots \mathtt{q}_m \, \mathtt{h}_1 \dots \mathtt{h}_r)^T \in \mathbb{F}_2^{2n+m+r}$ a firing assignment vector for D.*

In order for such a firing assignment to be useful, we need some conditions to hold. Again, when a neighbour of a node x fires, we want x to fire in the same colour in that direction in order to counteract the π on that wire, leaving us with the original state plus some Pauli's on the output wires. If we want to achieve this, we again need to put restrictions on our system, as phaseless and π-spiders can only fire in their own colour on an even number of wires. The $\pm\frac{\pi}{2}$-spiders are a little more complicated. They can fire their own colour on an even number of wires in case they didn't fire the opposite colour and can fire

on an odd number of wires if they did. So in order for all the internal wires to be clean again after all the nodes have fired, we need the following constraints to hold.

Definition 3.3.6. *A firing assignment vector $\vec{v}$ for a Clifford diagram D is called valid if for every phase 0 or phase π internal node $q \in (V \setminus O) \setminus H$ we have*

$$\sum_{x:(q,x)\in E} \mathtt{x} = 0 \mod 2,$$

for every output node $r \in O$

$$\mathtt{r} + \sum_{x:(r,x)\in E} \mathtt{x} = 0 \mod 2$$

and for every phase $\pm\frac{\pi}{2}$ internal node $h \in H$

$$\sum_{x:(h,x)\in E} \mathtt{x} = \mathtt{h} \mod 2.$$

Once again, these conditions can be written down more concisely in matrix form using the extended definition of the firing verification matrix.

Definition 3.3.7. *Given an n-qubit graph-like Clifford ZX-state D we define the firing verification matrix M_D of D to be:*

$$M_D \coloneqq \left(\begin{array}{c|c} I_n & \\ \emptyset & N \end{array}\right) - \begin{pmatrix} \emptyset & \emptyset \\ \emptyset & I_r \end{pmatrix}$$

where N is the adjacency matrix of the underlying graph of D, using the implicit ordering of the nodes of D and r is the number of phase $\pm\frac{\pi}{2}$ internal nodes.

This matrix can be used to verify if a firing assignment vector is valid.

Lemma 3.3.8. *A firing assignment vector $\vec{v}$ for D is valid if $M_D\vec{v} = \vec{0}$.*

Proof. Writing out the system of equations immediately yields the desired result. □

Just like in the phaseless case, we now want to show that a basis of valid firing assignment vectors corresponds to a generating set of stabilizers for the given state. Again, we spit this into two parts. The first shows that valid firing assignment vectors do indeed give stabilizers.

Theorem 3.3.9. *If a vector $\vec{v} = (\mathtt{g}_1 ... \mathtt{g}_n \mathtt{r}_1 ... \mathtt{r}_n \mathtt{q}_1 ... \mathtt{q}_m \mathtt{h}_1 ... \mathtt{h}_r)^T$ is a valid firing assignment for a graph-like Clifford-state D, then $i^{S(\vec{v})} \cdot X^{\mathtt{r}_1} Z^{\mathtt{g}_1} \otimes X^{\mathtt{r}_2} Z^{\mathtt{g}_2} \otimes ... \otimes X^{\mathtt{r}_n} Z^{\mathtt{g}_n}$ is a stabilizer for D, where $S(\vec{v}) = 2\sum_{q\in P} \mathtt{q} + \sum_{h\in H^+} \mathtt{h} - \sum_{h\in H^-} \mathtt{h}$*

Proof. The proof is similar to the phaseless case. We first fire all green nodes with $\mathtt{x} = 1$ red, then compensating on all its neighbours by firing red in that direction, then doing the same for firing green. $M_D\vec{v} = \vec{0}$ ensures that all firings are allowed. What we need to keep track of is the sign of the resulting state. As we have seen, firing a π-spider introduces a factor -1, so we need to multiply by i^2 for every fired π-spider. Firing a green $\frac{\pi}{2}$-spider or a red $-\frac{\pi}{2}$-spider introduces a factor i. Lastly, firing a green $-\frac{\pi}{2}$-spider or a red $\frac{\pi}{2}$-spider introduces a factor $-i$. So we multiply by i^{-1}. Because we always fire red first, we ensure that there are no places where we need to switch red and green π's around to let them cancel out. So we introduce no more sign changes. This shows that the sign of the resulting state is indeed $i^{S(\vec{v})}$ and when we put this phase in the stabilizer, we end up with our original state with a stabilizer applied to it. □

The second part shows that we get enough stabilizers this way to get a generating set.

Theorem 3.3.10. *Every stabilizer S of a graph-like Clifford state D corresponds to at least one valid firing assignment $\vec{v}$ of D.*

Proof. If the first $2n$ entries of valid firing assignment $\vec{v}_i$ of D are independent, then the corresponding stabilizers are independent as well. We see that this is true by first noting that independent vectors $(z_1\, z_2 \ldots z_n\, x_1\, x_2 \ldots x_n)^T$ correspond to independent operators $X^{x_1}Z^{z_1} \otimes X^{x_2}Z^{z_2} \otimes \ldots \otimes X^{x_n}Z^{z_n}$. This means that the only chance to have any dependencies between the stabilizers is to have them differ only by a factor i^s. However, this is impossible, as two operators that stabilize the same state cannot differ by a scalar.

This means we can continue to use the same argument as in Theorem 3.2.12. We have

$$M_D = \begin{pmatrix} I & \emptyset & A^T \\ \emptyset & A & B \end{pmatrix} - \begin{pmatrix} \emptyset & \emptyset \\ \emptyset & I_r \end{pmatrix},$$

which is still of the form

$$\begin{pmatrix} I & \emptyset & A^T \\ \emptyset & A & B' \end{pmatrix}$$

with $B' = B'^{\dagger}$, so the argument still holds. □

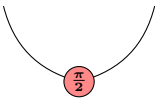
These last two theorems lead us to an algorithm for finding stabilizers from a given Clifford diagram. This algorithm is very similar to Algorithm 2.

Algorithm 3: Finding stabilizers for a Clifford ZX-state

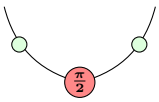
input : Clifford ZX-state D
output: Generating set of stabilizers for the group consisting of all the stabilizers of D
Let D' be the outcome of Algorithm 1 when applied to D;
Let n be the number of output wires of D;
Find a basis $\mathcal{B} = \{\vec{b_1}, ..., \vec{b_{n+d}}\}$ for the solution space of $M_{D'}\vec{v} = \vec{0}$;
Pick a set $\mathcal{B}'$ of n vectors from $\mathcal{B}$ such that they are independent when restricted to the first $2n$ entries;
return *The set of stabilizers that correspond to the vectors in $\mathcal{B}'$ as in Theorem 3.3.9*;

Correctness of this algorithm can be shown the same way we showed it for Algorithm 2 using the results of this section.

Example 3.3.11. *Suppose we want to find stabilizers for the state*

$D =$ 

We first find an equal diagram that is graph-like:

$D' =$ 

Then we construct its firing verification matrix

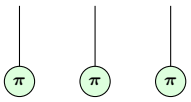
$$M_{D'} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

and solve $M_{D'}\vec{v} = \vec{0}$. This gives us

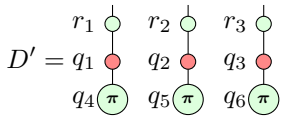
$$\mathcal{B} = \{ \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \}$$

These solutions correspond to the stabilizers $X \otimes X$ and $-i \cdot XZ \otimes Z = -Y \otimes Z$

Example 3.3.12. *Consider the state*

$D =$ 

If we want to find stabilizers for this state, we first find a graph like diagram that represents the same state, which we give some labeling:



Next, we create the firing verification matrix

$$M_{D'} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Solving $M_{D'}\vec{v} = \vec{0}$ gives

$$\mathcal{B} = \{ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \}$$

These correspond to the stabilizers $-X \otimes I \otimes I$, $-I \otimes X \otimes I$ and $-I \otimes I \otimes X$.

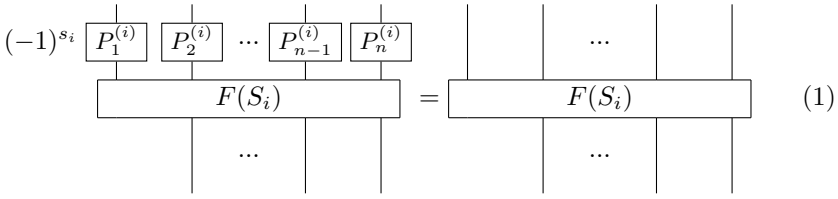
4 Stabilizers to ZX

In this section we aim to find a way of constructing a ZX-diagram for a state of which we know the stabilizer group. This group will be given in terms of a generating set of stabilizers and we will use them to build up the desired diagram. We first describe the general idea of the solution, then we discuss the normal form we want the stabilizers to be in and we finish the section by looking at how filter diagrams can help us reach the desired algorithm.

4.1 Introduction

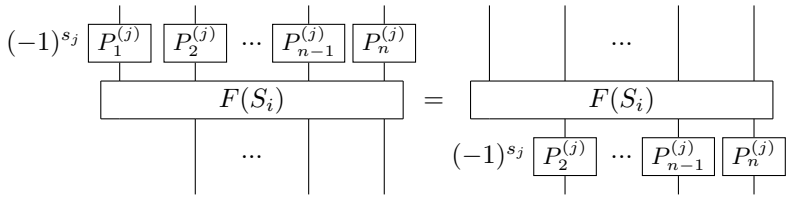
When we are given a stabilizer group on n qubits in terms of n independent generators $\mathcal{S} = \{S_1, ..., S_n\}$, we know that the group stabilizes exactly one state (up to a scalar). Our goal is to be able to construct an explicit ZX-diagram for a state $|\psi\rangle$ from a generating set of its stabilizers.

To achieve this we use filter diagrams to make sure our state is stabilized by $\mathcal{S}$ one generator at a time. Such a diagram is a map from $n-1$ to n qubits and uses the first qubit to absorb one of the generators. Given a generating set of stabilizers $\mathcal{S}$ we choose one $S_i = (-1)^{s_i} \cdot P_1^{(i)} \otimes ... \otimes P_n^{(i)}$ where $P_k^{(i)} \in \{I, X, Y, Z\}$. Ideally the corresponding filter diagram will filter out S_i as follows.



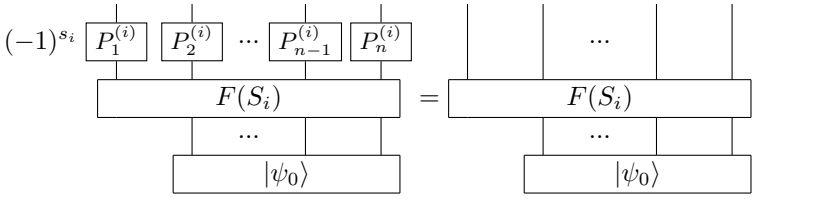
(1)

Whilst leaving the other generators $S_j = (-1)^{s_j} \cdot P_1^{(j)} \otimes ... \otimes P_n^{(j)} \in \mathcal{S} \setminus S_i$ alone on the last $n-1$ qubits.



(2)

This means that if we apply $F(S_i)$ to any $n-1$-qubit state $|\psi_0\rangle$, we obtain a state $|\psi_1\rangle = F(S_i)|\psi_0\rangle$ that is guaranteed to be stabilized by S_i. As for any such $|\psi_0\rangle$ we can apply (1) directly to obtain



So we used up one of the n available qubits to ensure that one of the n given generators is indeed a stabilizer for the state we are constructing. What remains is to find a state $|\psi_0\rangle$ which is in turn stabilized by the remaining, mostly unaffected operators $\{(-1)^{s_j} \cdot P_2^{(j)} \otimes ... \otimes P_n^{(j)} | j \neq i\}$. If we can always find such a state $|\psi_0\rangle$, we can use the same filtering strategy all the way down until there are no qubits left and we have found a state that is stabilized by all of our original n stabilizers. It turns out that this strategy works, but we need to be a bit careful as the next example points out.

Example 4.1.1. *Say we are given the following generating set of stabilizers on 3 qubits:* $\{Z \otimes I \otimes Z,\ Z \otimes Z \otimes I,\ X \otimes X \otimes X\}$. *When we choose to filter out* $Z \otimes I \otimes Z$ *first, we are left with the task of finding a state* $|\psi_0\rangle$ *which is stabilized by the remaining two generators* $Z \otimes I$ *and* $X \otimes X$. *The problem is that these two operators do not commute and therefore cannot have a simultaneous eigenstate. This means that there is no such* $|\psi_0\rangle$*, leaving us empty handed.*

4.2 Normal Form

Example 4.1.1 shows us that the filtering technique will not immediately work on any set of generators. But it turns out that choosing different generators of the same set of stabilizers will solve our problem. Our goal is to get the generators in such a form that the $n-1$ operators that remain after filtering, form a generating set of stabilizers again. To do this we need the following lemma.

Lemma 4.2.1. *Given a generating set of stabilizers* $\mathcal{S}$ *on* n *qubits, for every* $i \in \{1, ..., n\}$ *there is at least one* $S \in \mathcal{S}$ *that has support on qubit* i*, i.e.* $S = (-1)^s \cdot P_1 \otimes ... \otimes P_n$ *with* $P_i \neq I$.

Proof. Suppose there is a qubit $i \in \{1, ..., n\}$ with no support in $\mathcal{S}$. If $n = 1$ we have $S = \{I\}$, which is a contradiction as this set is not independent. Now suppose $n > 1$. We can make a new set $\mathcal{S}'$ by taking every $S \in \mathcal{S}$ and removing the i^{th} Pauli. So we let

$$\mathcal{S}' = \{(-1)^s \cdot P_1 \otimes ... \otimes P_{i-1} \otimes P_{i+1} \otimes ... \otimes P_n | (-1)^s \cdot P_1 \otimes ... \otimes P_n \in \mathcal{S}\}.$$

We see that $\mathcal{S}'$ consists of operators from $n-1$ to $n-1$ qubits. However, these operators still commute and are independent, as $\mathcal{S}$ is a generating set of stabilizers and we only removed a column of I's. So we have found n independent, commuting stabilizers for a state on $n-1$ qubits. This is a contradiction. Therefore we cannot have one qubit with no support. □

We will use Lemma 4.2.1 to define the normal form for a generating set of stabilizers. This form will ensure that the remainder after filtering out the first stabilizer will be a generating set of stabilizers again. To make this possible we need to pick a compatible Pauli for each of X, Y and Z.

Definition 4.2.2. *Given* $P \in \{X, Y, Z\}$ *we call* $Q(P) \in \{X, Y, Z\}$ *the compatible Pauli of* P. *We have* $Q(X) = Z$, $Q(Y) = X$ *and* $Q(Z) = X$.

Remark. The compatible Pauli of P has to differ from P, as we will see later. However, the choice between the other two possibilities has been made purely for simplicity of the resulting diagrams. Other choices would have worked just as well. The only difference is that we would have needed more spiders to create our filter diagrams later on.

Definition 4.2.3 (Stabilizer Normal Form)**.** *Given a generating set of stabilizers* $\mathcal{S} = \{S_1, ..., S_n\}$ *where* $S_i = (-1)^{s_i} \cdot P_1^{(i)} \otimes ... \otimes P_n^{(i)}$*, we say* $\mathcal{S}$ *has a normal*

form if there is an ordering $(S_1, ..., S_n)$ *of* $\mathcal{S}$ *such that the following properties hold:*

1. $P_i^{(i)} \in \{X, Y, Z\}$ *for all* $i \in \{1, ..., n\}$.
2. $P_i^{(j)} \in \{I, Q(P_i^{(i)})\}$ *for all* $i \in \{1, ..., n\}$ *and* $i < j \leq n$.

We say that such an ordering $(S_1, ..., S_n)$ *is in normal form.*

When we order the stabilizers of a generating set and put them in a table, we see that it is in normal form if it has support on the diagonal and in every column, below the diagonal, we only find I and one other Pauli, depending on the one on the diagonal.

Example 4.2.4. *The generating set of stabilizers* $\mathcal{S} = \{X \otimes X \otimes X \otimes X, Z \otimes Z \otimes I \otimes I, I \otimes I \otimes Y \otimes Z, Z \otimes I \otimes X \otimes X\}$ *is in normal form when ordered from left to right. This can be most easily checked by writing* $\mathcal{S}$ *in a table.*

$$\begin{array}{ccccccc} X & \otimes & X & \otimes & X & \otimes & X \\ Z & \otimes & Z & \otimes & I & \otimes & I \\ I & \otimes & I & \otimes & Y & \otimes & Z \\ Z & \otimes & I & \otimes & X & \otimes & X \end{array}$$

Now we see that indeed the diagonal does not have an I *on it. Furthermore, in the first column we only have* I *and* Z *under the diagonal, in the second we only have* I *and in the third only* X*. Ensuring that this ordering of* $\mathcal{S}$ *is in normal form.*

Rearranging the stabilizers in $\mathcal{S}$ *as follows*

$$\begin{array}{ccccccc} I & \otimes & I & \otimes & Y & \otimes & Z \\ Z & \otimes & Z & \otimes & I & \otimes & I \\ X & \otimes & X & \otimes & X & \otimes & X \\ Z & \otimes & I & \otimes & X & \otimes & X \end{array}$$

makes it so that this ordering is not in normal form. This is because the first diagonal entry is an I*, the first column has both* X *and* Y *below the diagonal and the third column has an* X *below the diagonal whilst there is already an* X *on the diagonal as well.*

Getting a generating set of stabilizers in normal form is useful, because it will ensure that we can filter out the first stabilizer and be left with another generating set of stabilizers on $n-1$ qubits, as we see from the next proposition.

Proposition 4.2.5. *Given a generating set of stabilizers* $\mathcal{S}$ *with normal form* $(S_1, ..., S_n)$*, the resulting set of operators after filtering out* S_1*,*

$$\mathcal{S}' = \{S_i' = (-1)^{s_i} \cdot P_2^{(i)} \otimes ... \otimes P_n^{(i)} | 1 < i \leq n\},$$

is a generating set of stabilizers on $n-1$ *qubits that has normal form* $(S_2', ..., S_n')$.

Proof. Suppose we have two operators $S'_i, S'_j \in \mathcal{S}'$, $S'_i = (-1)^{s_i} \cdot P_2^{(i)} \otimes ... \otimes P_n^{(i)}$ and $S'_j = (-1)^{s_j} \cdot P_2^{(j)} \otimes ... \otimes P_n^{(j)}$. As the corresponding $S_i, S_j \in \mathcal{S}$ commute we know that there is an even number of indices $k \in \{1, ..., n\}$ such that $P_k^{(i)}$ and $P_k^{(j)}$ anti-commute. As $i, j > 1$ we know that $P_1^{(i)}, P_1^{(j)} \in \{I, Q(P_1^{(1)})\}$, because $\mathcal{S}$ is in normal form. Therefore $P_1^{(i)}$ and $P_1^{(j)}$ must commute. This means that we have an even number of indices $k \in \{2, ..., n\}$ such that $P_k^{(i)}$ and $P_k^{(j)}$ anti-commute. Which shows that S'_i and S'_j commute.

Next suppose that $S'_2, ..., S'_n$ are dependent. We have

$$\prod_{2 \leq i \leq n} a_i S'_i = I \otimes ... \otimes I \text{ for some } a_i \in \{0, 1\}, \text{ not all 0.} \tag{3}$$

Because $S_1, ..., S_n$ are independent we must have $S = \prod_{2 \leq i \leq n} a_i S_i \neq I \otimes ... \otimes I$. As the last $n - 1$ Pauli's of S are equal to I by (3), the first Pauli of S cannot be I. The only other possibility for this first Pauli is $Q(P_1^{(1)})$, because $\mathcal{S}$ is in normal form. So we have $S = (-1)^s Q(P_1^{(1)}) \otimes I \otimes ... \otimes I$ for some s. However, as $Q(P_1^{(1)}) \neq P_1^{(1)}$ we have that S and S_1 do not commute, which is a contradiction. So $S'_2, ..., S'_n$ must be independent of each other.

Lastly, the fact that $\mathcal{S}'$ is in normal form follows directly from $\mathcal{S}$ being in normal form. □

Having a generating set of stabilizers in normal form solves the problem we encountered in Example 4.1.1. As after filtering out the first stabilizer we end up with another generating set in normal form. So we can repeat the process all the way down. What we would like, is that we can use the filtering method for finding the underlying state for every generating set of stabilizers $\mathcal{S}$. It turns out that this is possible by transforming $\mathcal{S}$ into an equivalent generating set that is in normal form.

Proposition 4.2.6. *For any generating set of stabilizers $\mathcal{S}$ there is a generating set $\mathcal{S}'$ with a normal form that stabilizes the same state.*

Proof. We prove this by induction on n. Suppose $n = 1$. Then $\mathcal{S}$ consists of one Pauli. This Pauli cannot be I, because $\mathcal{S}$ is an independent set. Therefore the diagonal property is satisfied. The sub-diagonal property is automatically satisfied too. So we have that $\mathcal{S}$ is in normal form already.

Now suppose $n > 1$. By Lemma 4.2.1 we can find at least one $S \in \mathcal{S}$ that has support on the first qubit. We set this to be our first stabilizer S'_1. We now have a Pauli other than I in the first diagonal entry.

Given two stabilizers $S, T \in \mathcal{S}$, we can create another generating set of stabilizers, $\mathcal{S}'$ that stabilizes the same state, by replacing T by ST. So we can

ensure that the second property is met in the first column by replacing every $T \in \mathcal{S} \setminus \{S_1'\}$ that has an unwanted first Pauli by $S_1'T$. As S_1' has support on the first entry we can see from the basic properties of Pauli matrices that we do indeed get that the first column now obeys the second property for being in normal form.

Next, we can follow the proof of Proposition 4.2.5 to see that the $n-1$ by $n-1$ block in the bottom right is again a generating set of stabilizers, however, this time not necessarily in normal form. We can apply the induction hypothesis and find a way to reorder and replace the rows of this block to get it in normal form. Applying these same reorderings and replacements on the last $n-1$ rows of the whole table results in a table for $\mathcal{S}'$. Clearly $\mathcal{S}'$ stabilizes the same state as $\mathcal{S}$ does, as we only rearranged the rows and added some rows to others. Also the bottom right $n-1$ by $n-1$ block is in normal form by the induction hypothesis. This means that the 2^{nd} to the n^{th} diagonal entries are not I and per column we have only one Pauli other than I under the diagonal. We didn't touch the first row, so also the first diagonal entry is not an I. Lastly, in the last step we rearranged and replaced within the last $n-1$ rows. This cannot introduce a new Pauli to the first column, so this still satisfies the normal form property. Therefore the entire table for $\mathcal{S}'$ is in normal form. □

The proof of this last proposition shows us how to construct a new generating set of stabilizers that is in normal form and stabilizes the same state. This leads to the following algorithm.

Algorithm 4: Putting stabilizers in normal form

input : A set $\mathcal{S}$ of n generating stabilizers on n qubits
output: A normal form $(S_1', ..., S_n')$ of a generating set of stabilizers that stabilizes the same state as $\mathcal{S}$

```
Choose an ordering (S_1, ..., S_n) of S and create the array
 S = [S_1, ..., S_n];
for i = 1, ..., n do
    Pick j ≥ i such that S[j] has support on the i^th qubit;
    Switch S[i] and S[j];
    Let P be the i^th Pauli of S[i];
    for j = i + 1, ...n do
        if the i^th Pauli of S[j] is not I or Q(P) then
            Replace S[j] by S[i]S[j];
        end
    end
end
return The ordering (S[1], ..., S[n]);
```

The correctness of this algorithm follows from the proof of Proposition 4.2.6.

Example 4.2.7. *We saw that the generating set from Example 4.1.1 caused some problems. We can solve these by putting $\mathcal{S}$ in normal form. We start with*

the original stabilizers

$$\begin{array}{ccccc} Z & \otimes & I & \otimes & Z \\ Z & \otimes & Z & \otimes & I \\ X & \otimes & X & \otimes & X \end{array}$$

We already have support on the first diagonal entry, so we can immediately start using the first row to clear the others of Z or Y in the first column. So we add the first row to the second to obtain

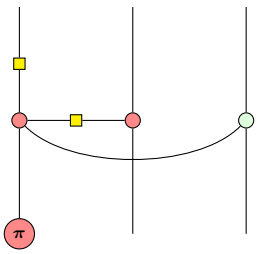
$$\begin{array}{ccccc} Z & \otimes & I & \otimes & Z \\ I & \otimes & Z & \otimes & Z \\ X & \otimes & X & \otimes & X \end{array}$$

which is in normal form.

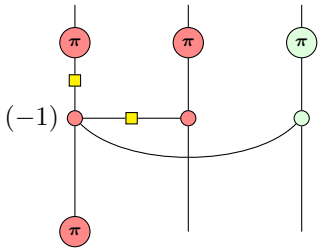
4.3 Filter Diagrams

What we have seen so far is that if we can make filter diagrams that satisfy (1) and (2), we can use them to construct a ZX-diagram for a state $|\psi\rangle$, given a generating set $\mathcal{S}$ of its stabilizers. What remains is to actually create such filter diagrams.
We will start by putting $\mathcal{S}$ in normal form $(S_1, ..., S_n)$. The idea is to turn the first Pauli operator $P_1^{(1)} = X, Y, Z$ into a Z, then make n copies of this Z, use the first copy to absorb the phase $(-1)^{s_i}$ and change the others so that they absorb the rest of the Pauli's of that stabilizer. This process is best viewed through an example.

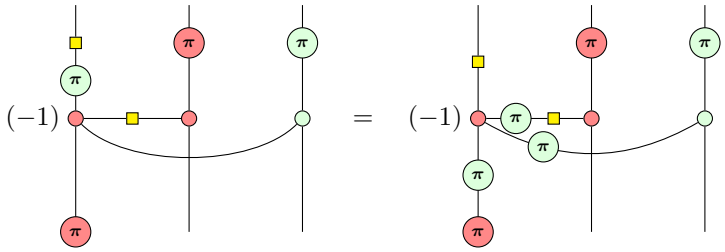
Example 4.3.1. *If we want to make a filter diagram for $-X \otimes X \otimes Z$ it would look like this.*



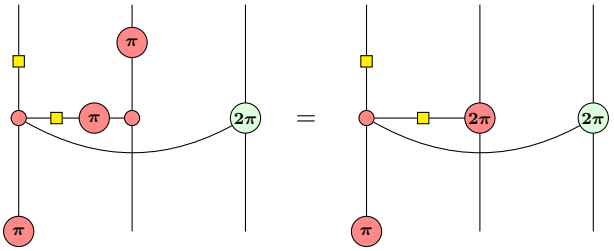
Plugging in our stabilizer $-X \otimes X \otimes Z$ we get



We see that we can now change the first X to a Z and create 3 copies.

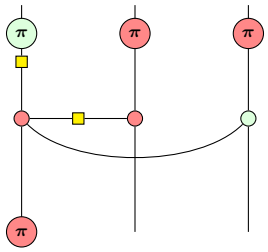


One copy goes down and by getting absorbed by the bottom red π it changes the phase to $+1$. Of the other two, one is transformed into an X to absorb the second Pauli, whilst the other remains a Z and absorbs the third.

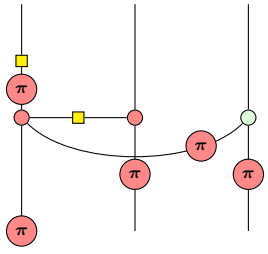


This last diagram of course being equal to our original filter diagram, as desired.

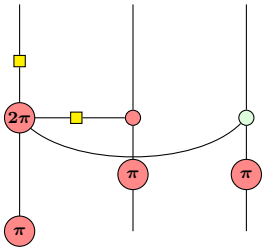
On the other hand, if we plug in $Z \otimes X \otimes X$, a stabilizer that commutes with $X \otimes X \otimes Z$ and could appear after it in normal form, we get



The Z gets transformed to an X, the first X just passes through and the last X gets copied.



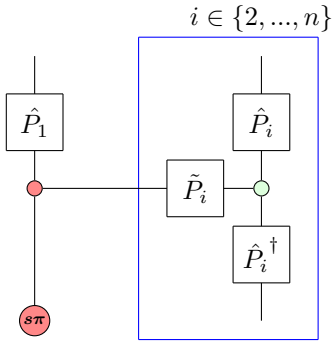
We see that we end up with the desired result, as the two left over X's merge and disappear.



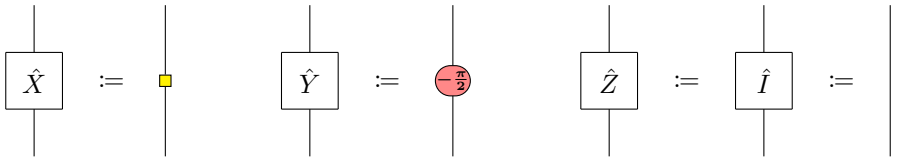
So indeed we have that this stabilizer is left mostly intact. We saw that we can end up with some left over π's on the wires to the leftmost node. It turns out that these appear whenever the i^{th} Pauli of the filtered stabilizer doesn't commute with the stabilizer that is plugged in. As these stabilizers will always commute we end up with an even number of these left over π's, making sure that they always disappear. We will see that this is the case in general.

Filter diagrams will consist of two parts. The first turns the incoming Pauli into n copies of a Z spider, of which it uses one to get rid of the phase. The second gets one of the remaining $n-1$ copies of Z and uses this to absorb the incoming Pauli on that wire. Using these parts we define a filter diagram as follows.

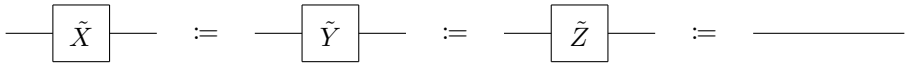
Definition 4.3.2 (Filter diagram)**.** *Given a stabilizer $S = (-1)^s \cdot P_1 \otimes P_2 \otimes ... \otimes P_n$ the filter diagram $F(S)$ of S is*

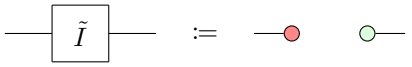


Where

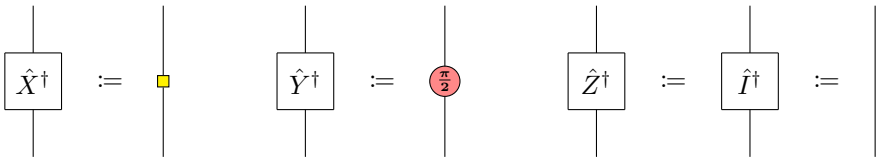


and



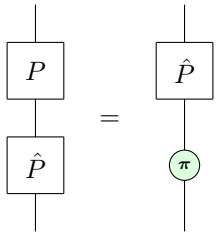


Remark. Note that as all the $\hat{P}$ are unitary, we have $\hat{P}^\dagger = \hat{P}^{-1}$. This means that

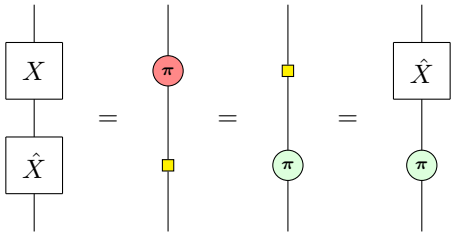


To prove that these filter diagrams do indeed show the behaviour that we want, we need a couple of lemmas. These are about the properties of smaller parts of filter diagrams and will make it easier to see the big picture later on. However, before we take a look at these, we first need another small lemma about the $\hat{P}$-spiders that we just defined.

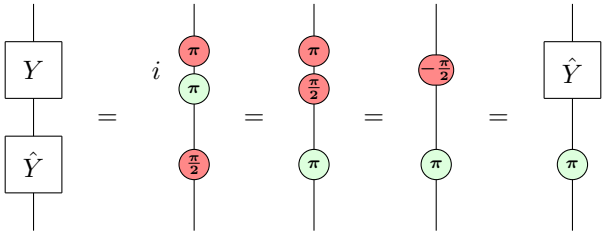
Lemma 4.3.3. *For any Pauli $P \neq I$ we have:*



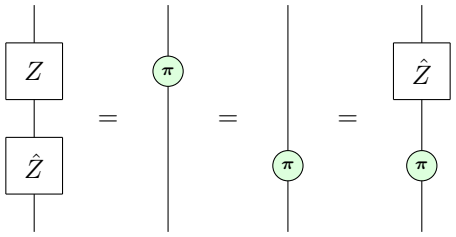
Proof. We distinguish between the three possible cases $P = X, Y, Z$. First, when plugging in $P = X$ we get:



Next, taking $P = Y$ gives us



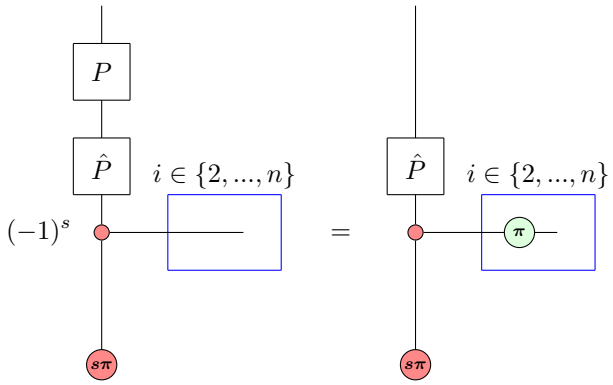
Finally, the case where $P = Z$ is completely trivial as:



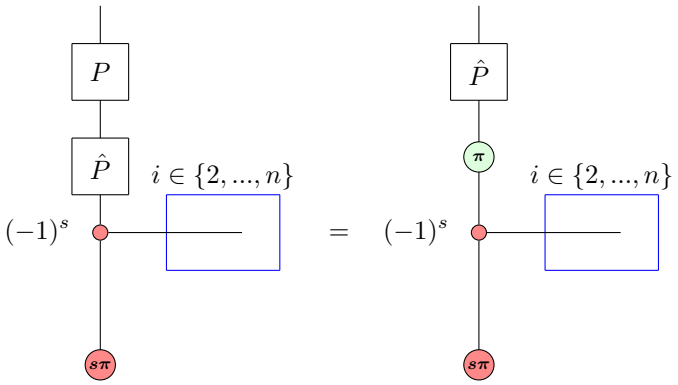
□

Using Lemma 4.3.3 we can now examine the behaviour of the left side of a filter diagram when it encounters the Pauli it was meant for.

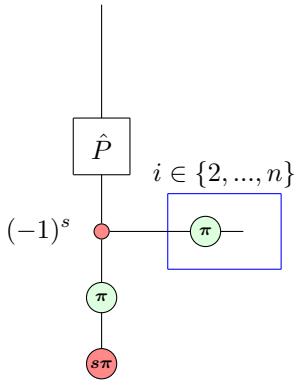
Lemma 4.3.4. *For any Pauli $P \neq I$ and phase $(-1)^s$ we have*



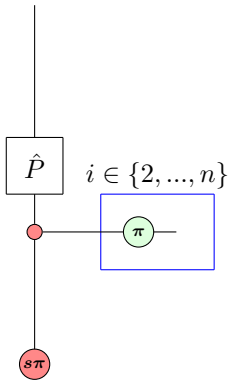
Proof. By applying Lemma 4.3.3 we have:



Copying the Z-spider with rule (c) we get:



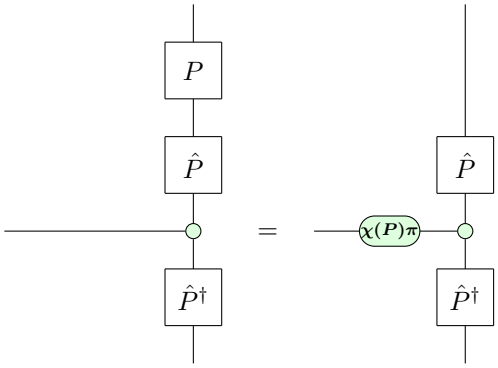
Next we absorb the bottom Z-spider by applying (c) or Proposition 2.4.5 to obtain



□

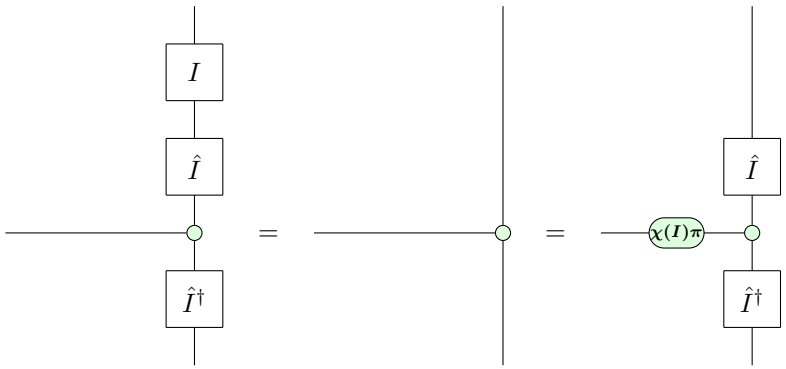
Next we take a look at what happens on the right side of a filter diagram.

Lemma 4.3.5. *For any Pauli P we have*

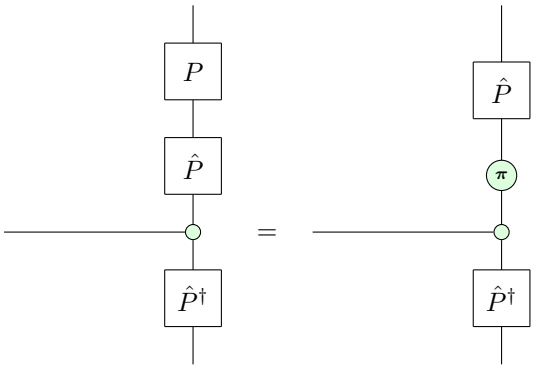


where $\chi(P) = 1$ if $P \neq I$ and $\chi(P) = 0$ if $P = I$.

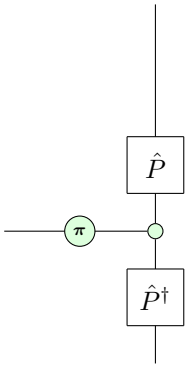
Proof. We split the proof into two cases: $P = I$ and $P \neq I$. We first consider the easiest case $P = I$, the proof of which is just writing out the definition.



Next, taking $P \neq I$, we apply Lemma 4.3.3 to the top and obtain



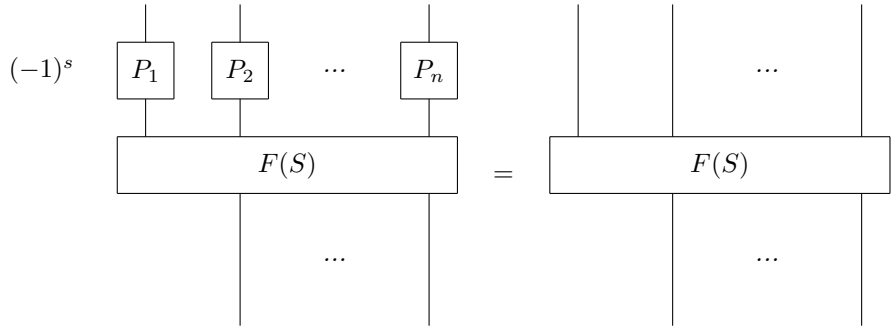
So we can just push the Z-spider through by using (f) twice and get



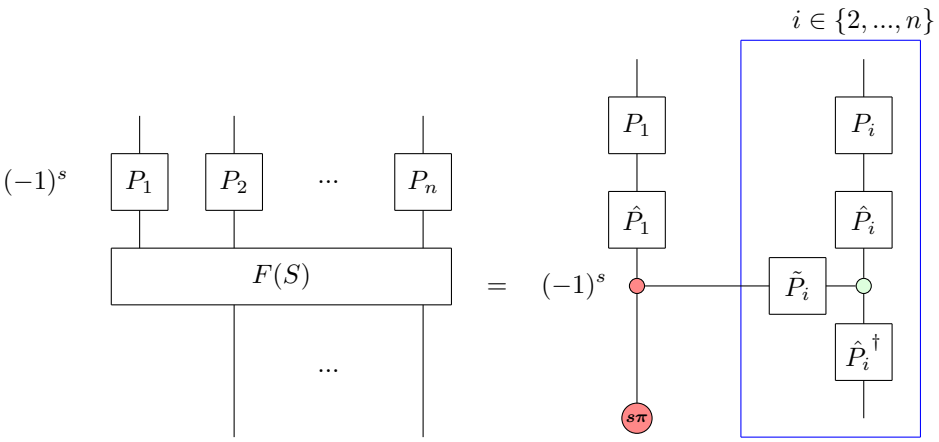
as desired. □

This gives us all that we need to show that the filter diagram defined in 4.3.2 satisfies (1). In other words, our filter diagrams do actually behave the way we want them to.

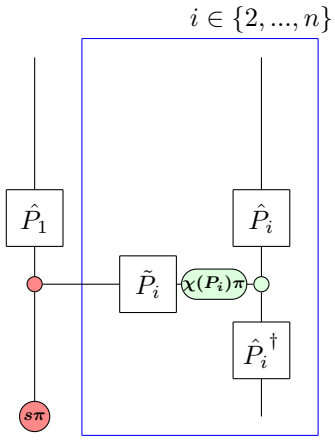
Theorem 4.3.6. *Given a stabilizer $S = (-1)^s \cdot P_1 \otimes P_2 \otimes ... \otimes P_n$ the filter diagram $F(S)$ of S, as defined in 4.3.2, satisfies*



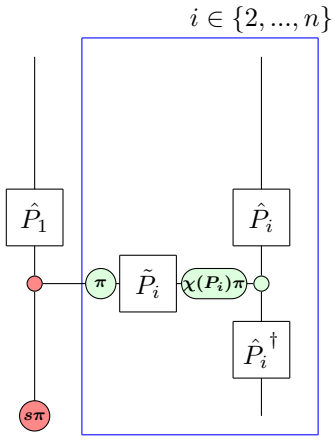
Proof. Writing out the definition we have



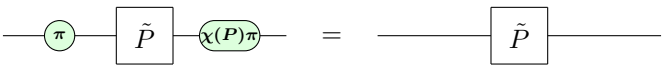
Now we use Lemma 4.3.4 to obtain



Next by applying Lemma 4.3.5 we get

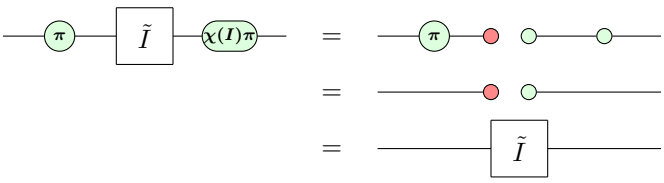


What remains is to show that

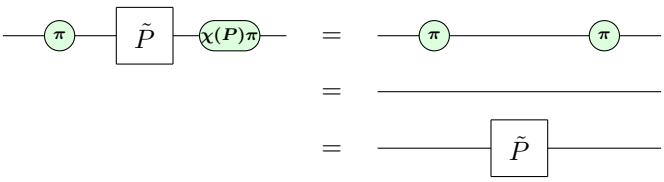


(4)

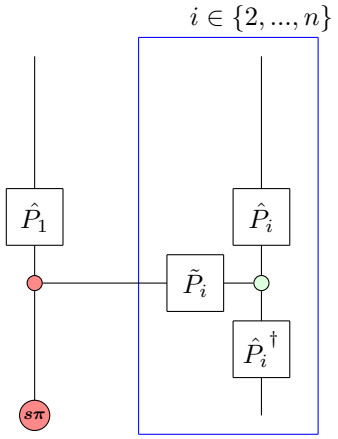
First, for $P = I$ we have



With $P \neq I$ we get



proving (4). Returning to the filter diagram, we apply (4) and obtain

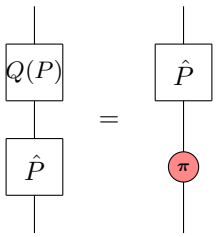


which is $F(S)$, as we desired. □

Now that we have seen that our filter diagram satisfies (1) for the stabilizer that it is supposed to filter out, it is time to look at what happens when stabilizers that are lower in the table get encountered. We will assume that the table we are stabilizing is in normal form.

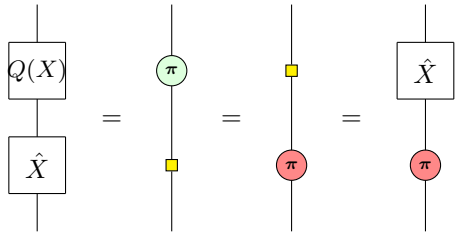
If we are constructing a diagram for the state that is stabilized by a given stabilizer table in normal form, we have some knowledge of what stabilizers can come after the one we are filtering out. When we are filtering a stabilizer of the form $S_i = X \otimes ...$ we know that we can only have $S_j = I \otimes ...$ or $S_j = Z \otimes ....$ Similarly when filtering $S_i = Y \otimes ...$ or $S_i = Z \otimes ...$ we can only get $S_j = I \otimes ...$ or $S_j = X \otimes ....$ The next lemma tells us something about the behaviour of the left part of a filter diagram when we encounter the nontrivial stabilizers lower in the table.

Lemma 4.3.7. *For any Pauli $P \neq I$ we have*

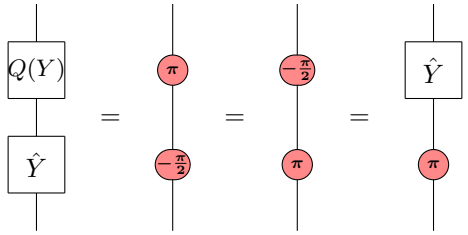


Proof. We prove this separately for all three cases $P = X, Y, Z$. First taking $P = X$ we write out the definitions and change the colour using rules (i2) and

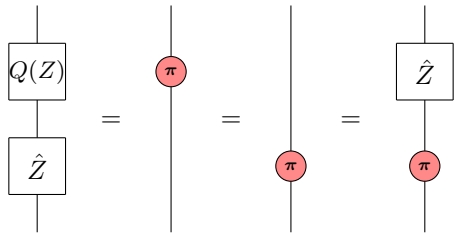
(h):



Next, $P = Y$ is even easier as we can just fuse and un-fuse using (f) twice:



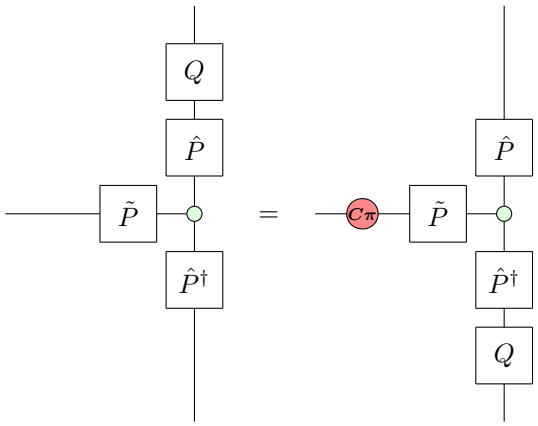
Finally, the case $P = Z$ is trivial:



□

Our goal for the right part of a filter diagram is to let every Pauli go through. That way we get the behaviour of property (2). As we've seen in Example 4.3.1, it is possible that some red π's leak to the side. This lemma captures what happens to the right side of filter diagrams.

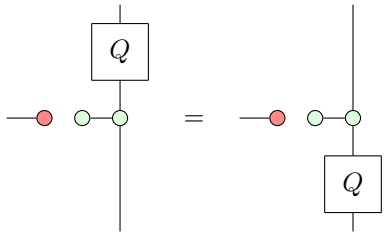
Lemma 4.3.8. *Given two Pauli's P and Q the following holds*



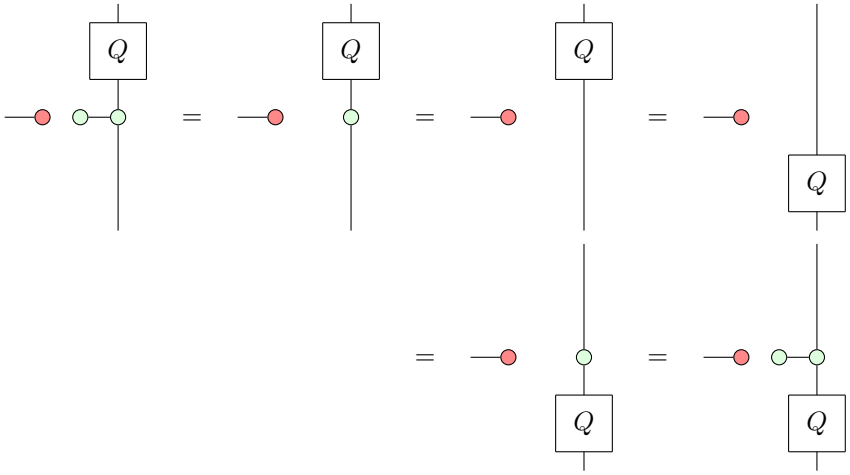
where $C = 0$ if P and Q commute and $C = 1$ if they do not.

Proof. We will prove this by looking at all the cases for P and Q. First, note that if $Q = I$ then P and Q always commute. Therefore $C = 0$ and we are done.

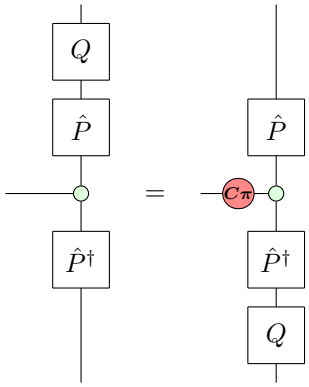
Now, if $P = I$ we also have $C = 0$, so what we want to show is:



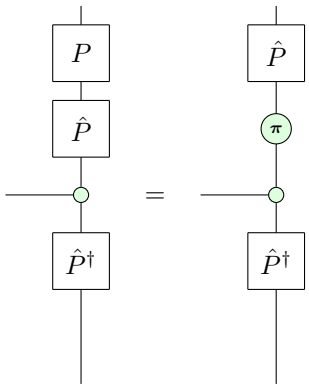
This can be easily proven as for any Pauli Q:



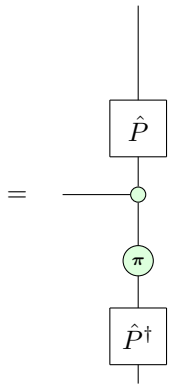
What remains are the cases $P, Q \in \{X, Y, Z\}$. We need to show that:



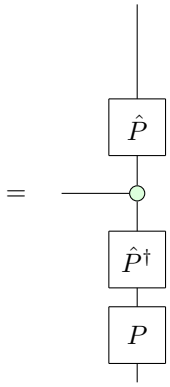
If $P = Q$ we know that $C = 0$, as they definitely commute. In that case, by Lemma 4.3.3 we have



We push the green π through and get:

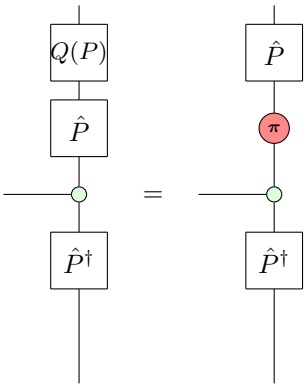


As we are dealing with unitaries we know $\hat{P}^\dagger = \hat{P}^{-1}$. Therefore we can reverse Lemma 4.3.3:

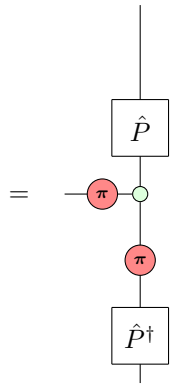


which is the desired result as we know $C = 0$.

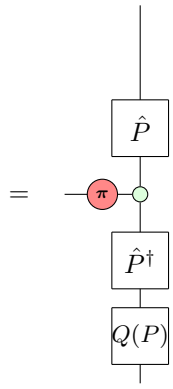
Six cases remain. Three of these are of the form Q=Q(P) as in Definition 4.2.2. We start with those, applying Lemma 4.3.7 to get:



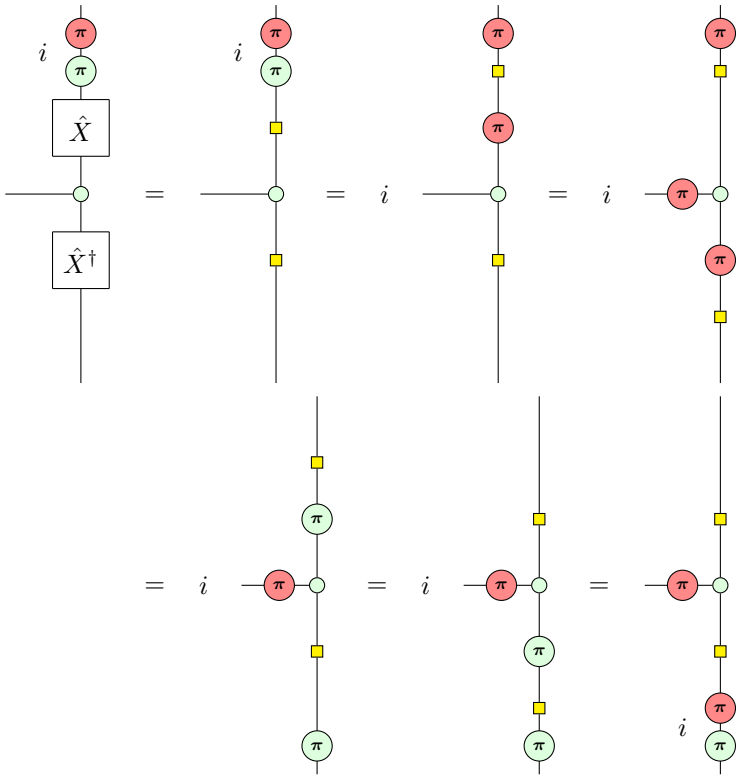
Then we copy the red π:



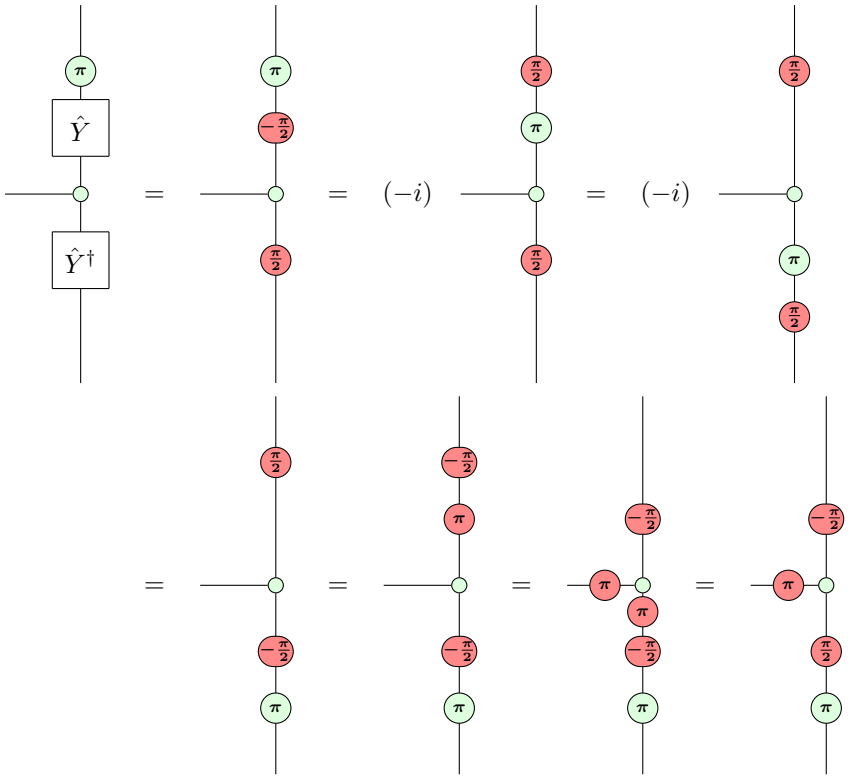
Just like we did previously, we can reverse Lemma 4.3.7 because $\hat{P}$ is unitary. This gives the desired result:



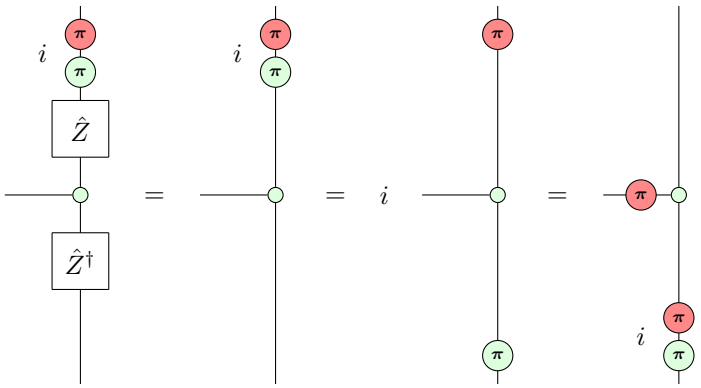
The last three cases we do separately, starting with $P = X, Q = Y$:



We continue with $P = Y, Q = Z$:

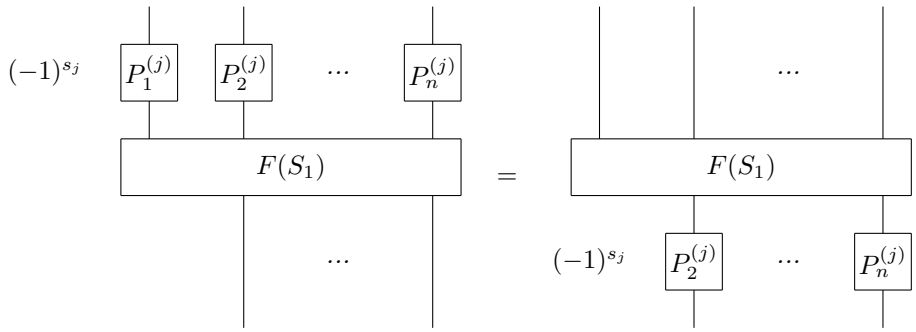


Lastly, we look at $P = Z, Q = Y$, which is relatively simple:



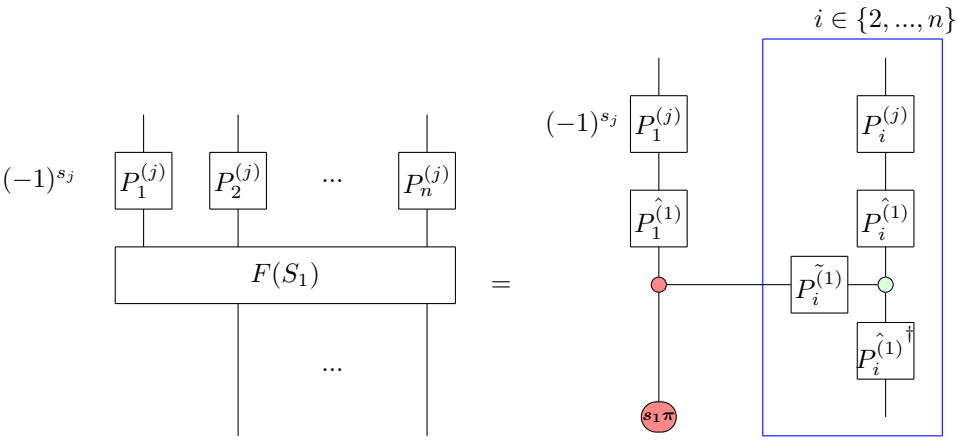
As we have covered the seven cases where $P = I$ or $Q = I$, then looked at the three remaining possibilities for $P = Q$, followed by the six cases $I \neq P \neq Q \neq I$, we have proven the lemma for all sixteen combinations of Pauli's P and Q. $\square$

Theorem 4.3.9. *Given a generating set of stabilizers $\mathcal{S}$ in normal form $(S_1, ..., S_n)$, where $S_j = (-1)^{s_j} \cdot P_1^{(j)} \otimes ... \otimes P_n^{(j)}$ we have*

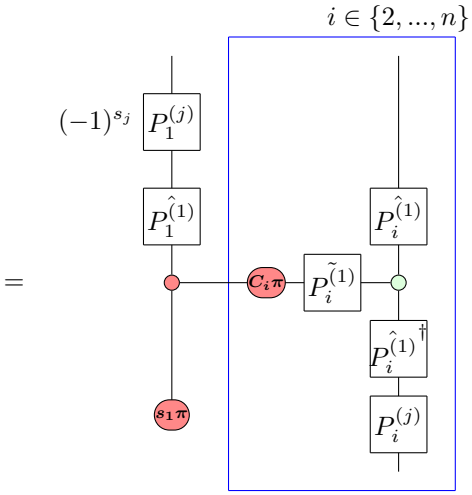


for $j > 1$.

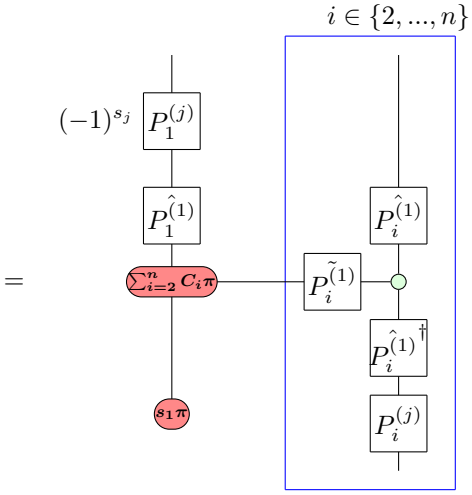
Proof. We start by writhing out the definition of $F(S_1)$:



By Lemma 4.3.8 we have, with $C_i = 0$ if $P_i^{(1)}$ and $P_i^{(j)}$ commute and $C_i = 1$ if they do not:

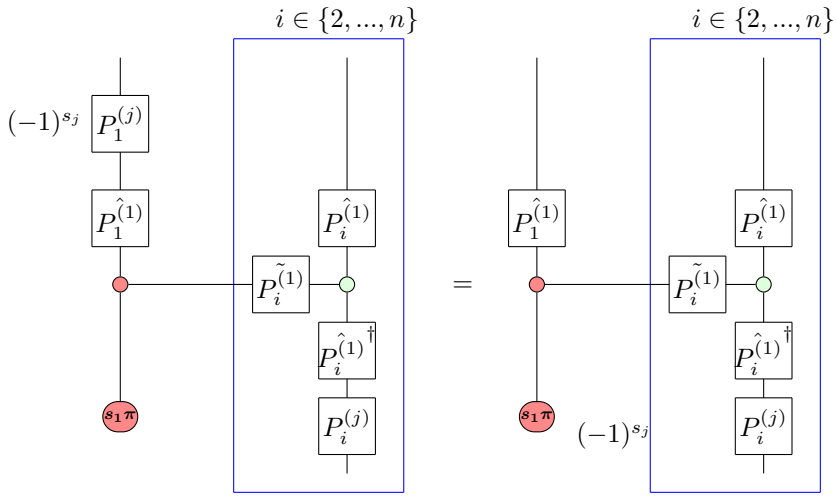


Fusing the red spiders we get:



In the case that $\sum_{i=2}^n C_i\pi = 0 \mod 2\pi$ we know that S_1 and S_j commute in an even number of the entries $2, ...n$. As S_1 and S_j stabilize the same state, we know that they commute as a whole. This means that $P_1^{(1)}$ and $P_1^{(j)}$ must commute as well. Because $\mathcal{S}$ is in normal form we have that $P_1^{(j)} \in \{I, Q(P_1^{(1)})\}$. But as $P_1^{(1)}$ and $Q(P_1^{(1)})$ don't commute, we know that $P_1^{(j)} = I$. So in that

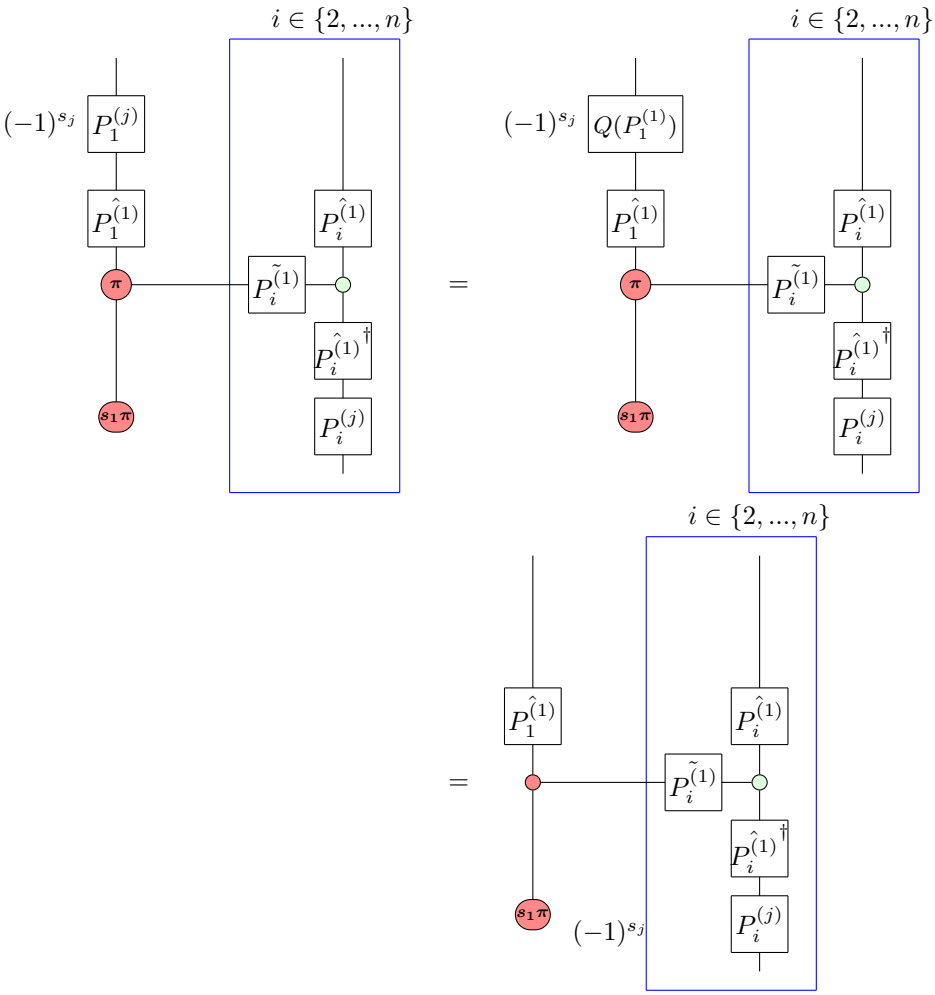
case:



which is the desired result.

Similarly, in the other case, in which $\sum_{i=2}^{n} C_i\pi = \pi \mod 2\pi$, we know that

$P_1^{(j)} = Q(P_1^{(1)})$. Therefore we can apply Lemma 4.3.7 and obtain:

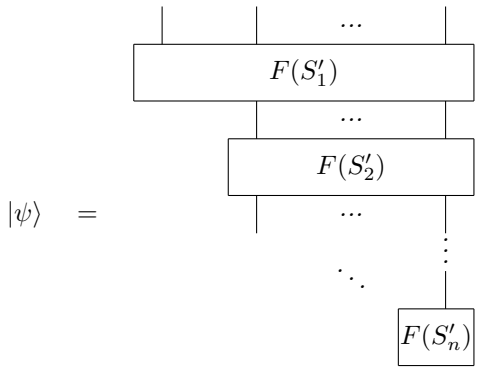


proving the theorem. □

With this theorem we have shown that property (2) is indeed satisfied by the filter diagram that we constructed in Definition 4.3.2. This means that the filter diagrams behave as desired. We can therefore use them to create a ZX-diagram for any given generating set of stabilizers.

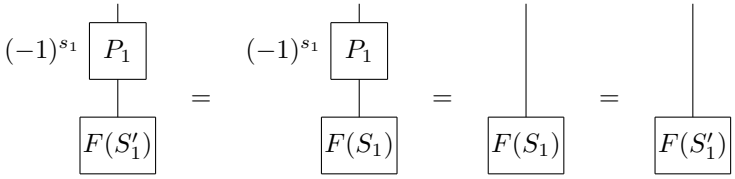
Theorem 4.3.10. *Given a generating set of stabilizers $\mathcal{S}$ for a state $|\psi\rangle$ with normal form $(S_1, ..., S_n)$, where $S_i = (-1)^{s_i} \cdot P_1^{(i)} \otimes ... \otimes P_n^{(i)}$. If we write*

$S'_i = (-1)^{s_i} \cdot P_i^{(i)} \otimes ... \otimes P_n^{(i)}$, *then up to normalization*

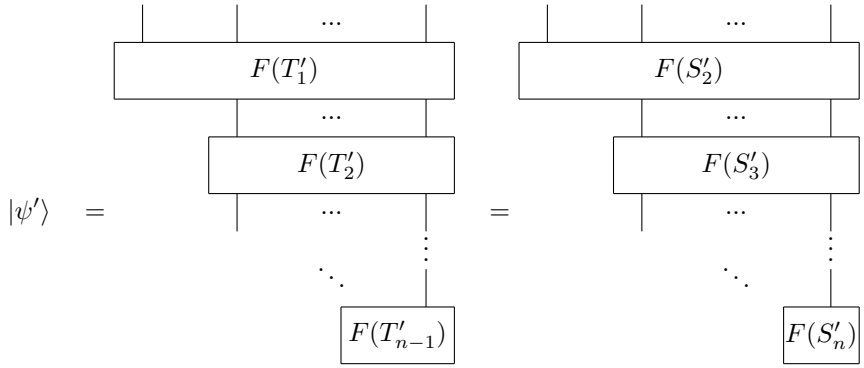


Proof. We use induction on n.

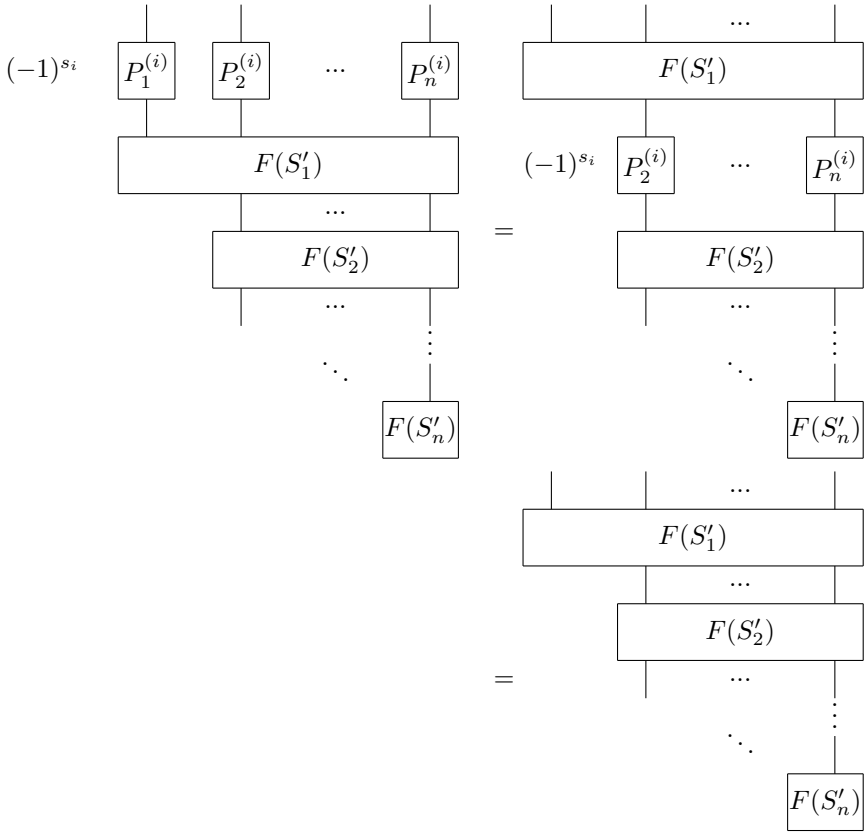
When $n = 1$ we have normal form (S_1), where $S_1 = (-1)^{s_1} \cdot P_1$. By Theorem 4.3.6 we see that



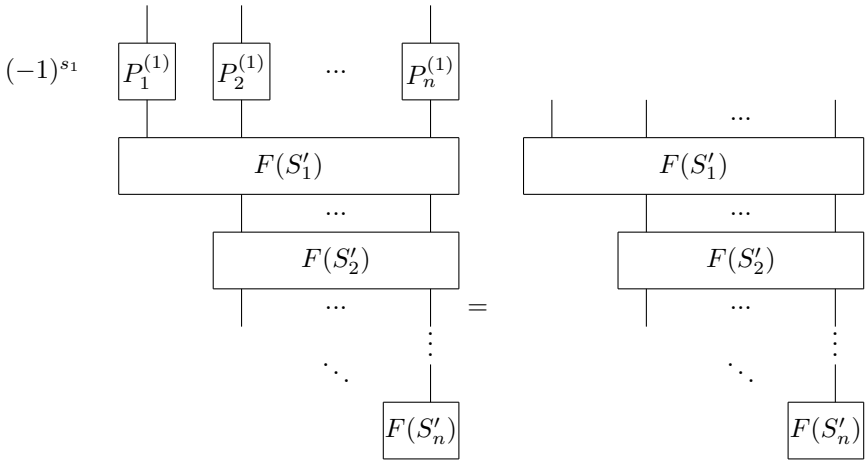
Let $T_i = (-1)^{s_{i+1}} \cdot P_2^{(i+1)} \otimes ... \otimes P_n^{(i+1)}$. By Proposition 4.2.5 we see that $(T_1, ..., T_{n-1})$ is a stabilizer normal form for some state $|\psi'\rangle$. By the induction hypothesis we have



So by using Theorem 4.3.9 we see that for all $i \in \{2, ..., n\}$ we have



by Theorem 4.3.6 we also have that with $i = 1$:



So the diagram is indeed stabilized by $S_1, ..., S_n$ and as up to normalization

there is only one n-qubit state that is stabilized by n independent stabilizers, the diagram must equal $|\psi\rangle$. □

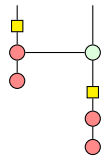
Using this last theorem we can directly formulate our final algorithm.

Algorithm 5: Finding a ZX-state for given stabilizers

input : A set $\mathcal{S}$ of n generating stabilizers on n qubits
output: A ZX-diagram D that represents a state that is stabilized by $\mathcal{S}$
Let $(S_1, ..., S_n)$ be the ordering that we get from Algorithm 4 when applied to $\mathcal{S}$, where $S_i = (-1)^{s_i} \cdot P_1^{(i)} \otimes ... \otimes P_n^{(i)}$ for every $i = 1, ..., n$;
Let $S_i' = (-1)^{s_i} \cdot P_i^{(i)} \otimes ... \otimes P_n^{(i)}$ for every $i = 1, ..., n$;
Let D be the filter diagram of S_n';
for $i = n-1, ..., 1$ **do**
  Replace D by D plugged into the filter diagram of S_i';
end
return D;

The correctness of this algorithm follows immediately when noting that $(S_1, ..., S_n)$ stabilizes the same state as $\mathcal{S}$ by Proposition 4.2.6 and then applying Theorem 4.3.10.

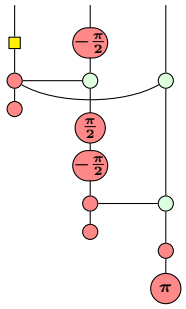
Example 4.3.11. *Given the following generating set of stabilizers $\mathcal{S} = \{X \otimes Z, Z \otimes X\}$. The first step in finding a state stat is stabilized by these operators is to find a set of stabilizers that generate the same group that has a normal form. It so happens that $(X \otimes Z, Z \otimes X)$ is a normal form, so we pick that. Now we create the state by plugging the respective filter diagrams together:*



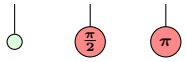
This can then be simplified to become:

Example 4.3.12. *Consider the following generating set of stabilizers $\mathcal{S} = \{X \otimes Y \otimes Z, -I \otimes I \otimes Z, X \otimes I \otimes I\}$. We first find a normal form that stabilizes the same state. We can simply take $S_1 = X \otimes Y \otimes Z$, but then we need to change $X \otimes I \otimes I$ out for $I \otimes Y \otimes Z$ by multiplying it by S_1 in order for our table to be in normal form. We then take this to be S_2. Lastly we can just take $S_3 = -I \otimes I \otimes Z$.*

Now that we have a normal form, we can plug in the filter diagrams and obtain:



Which we can simplify to:



5 Discussion and Future Work

In this section we will briefly discuss the results that we derived in the last two sections. We look at the similarities between the problems and describe related problems that arose on the way.

First of all, the two problems that we looked into, finding stabilizers for a given ZX-diagram and finding a ZX-diagram for a given set of stabilizers, seem to be each others inverse, or dual. The solutions that we presented here however lack any direct link. Where we find stabilizers by solving systems of equations using linear algebra, we do not apply any of this when going back. The main reason for this is that the resulting stabilizers are only part of the solution to the system of equations. We are only interested in the output nodes and whether or not they fired. The internal firings are disregarded, as they do not add anything to the actual stabilizers, apart from contributing to the sign. It is much harder to retrieve these internal workings when presented only with the resulting stabilizer. We cannot reconstruct the whole matrix M_D from a basis of stabilizers, as these are only the first $2n$ entries of the actual solutions. Next to that, we would not even know the dimension of the solutions we are looking for, as we are ignorant of the amount of internal nodes that a certain state should have. Instead of working around these problems, we opted for another solution, taking for granted that the connection between the two methods would be less explicit.

Going the other way, from stabilizers to diagrams, we used a layer-by-layer construction that filtered out every stabilizer, making sure that each of them indeed stabilizes the constructed state. Any Clifford state can be transformed into the general form of a state constructed by filter diagrams as in Theorem 4.3.10, as the ZX-Calculus is complete for stabilizer quantum mechanics. From this it would be easy to read off a generating set of stabilizers for that state. We have not looked into how plausible this method is for finding stabilizers from a given

state, but it might be interesting to research that in the future. This would give us a nice two way algorithm for going back and forth between ZX-Calculus and stabilizer theory.

The algorithms that are presented, for going from ZX-Calculus to stabilizers and to go back, have not yet been implemented. All of them seem very suitable for implementation in PyZX. This is a Python tool for reasoning in ZX-Calculus, enabling the user to visualize and rewrite large-scale quantum circuits [13].

References

[1] Coecke, B. Kissinger, A. *Picturing Quantum Processes*, Oxford University Press (2007).

[2] Duncan, R. Kissinger, A. Perfrix, S. Van de Wetering, J. *Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus* Quantum Physics and Logic (2019).

[3] De Wolf, R.*Quantum Computing: Lecture Notes* (2019)

[4] Backens, M. Perdrix, S. Wang, Q.*A Simplified Stabilizer ZX-calculus*, Quantum Physics and Logic (2017).

[5] Backens, M.*The ZX-calculus is complete for stabilizer quantum mechanics*, New Journal of Physics, Volume 16, Issue 9 (2014).

[6] Duncan, R. Lucas, M. *Verifying the Steane code with Quantomatic*, Quantum Physics and Logic (2014).

[7] Schlingemann, D. *Stabilizer codes can be realized as graph codes* (2001).

[8] Preskill, J. *Quantum Computation: Lecture Notes, Chapter 7: Quantum Error Correction.*

[9] Van den Nest, M. Dehaene, J. De Moor, B. *Graphical description of the action of local Clifford transformations on graph states*, Physical Review A, vol. 69, Issue 2 (2004).

[10] Grassl, M. *Variations on Encoding Circuits for Stabilizer Quantum Codes* Chee Y.M. et al. (eds) Coding and Cryptology. IWCC 2011. Lecture Notes in Computer Science, vol 6639. Springer, Berlin, Heidelberg (2011)

[11] Aaronson, S. Gottesman, D. *Improved Simulation of Stabilizer Circuits*, Physical Review A, vol. 70, Issue 5 (2004).

[12] Dixon, L. Duncan, R. *Graphical Reasoning in Compact Closed Categories for Quantum Computation*, R. Ann Math Artif Intell (2009).

[13] Kissinger, A. Van de Wetering, J. *PyZX: Large Scale Automated Diagrammatic Reasoning*, arXiv:1904.04735v1 (2019).

[14] Kelly, J. *A Preview of Bristlecone, Google's New Quantum Processor*, Google AI blog (2018).