UNIVERSITÄT BONN

Master's Thesis

Linear combinations of ZX-diagrams for parameterized quantum circuits

Gina Muuss

November 21, 2023

Semester WS22/23

Advisor:
Dr. Tobias Stollenwerk

Second Advisor:
Prof. Dr. Petra Mutzel

Institute for Computer Science
University of Bonn

Abstract

We formally define linear combinations of ZX-diagrams and show the practical impact of this definition by proposing an prototypical algorithm for calculating expectation values of parameterized quantum circuits. ZX-calculus is a modern approach to quantum computing circuits, that has also been applied to reasoning about solutions of combinatorial optimization problems on noisy intermediate-scale quantum (NISQ) devices. Arising in the analysis of the QAOA is the problem of simplifying symbolic ZX-diagrams to analytical expressions. We formalize the notion of previously proposed linear combinations of ZX-diagrams and harness their power towards a simplification algorithm. Prior research does not formally define sums of arbitrary ZX-diagrams. Further no simplification algorithm known to us is capable of simplifying symbolic ZX-diagrams to an analytical expression. This twofold gap is tackled first by means of assessing category theoretic properties of the category **ZX** and then by developing a simplification algorithm using these properties. We formalize the notion of linear combinations using the structure of a category enriched over commutative monoids. As a prototype for analysing arbitrary QAOA circuits, we present an algorithm for simplifying symbolic ZX-diagrams that come up when applying QAOA to the Maximum Cut Problem. This lays the foundation for fully automated tools to analyse parameterized quantum circuits (PQC).

Contents

1. Introduction

Quantum computing promises exponential speed-up over classical computers for certain applications [Sho94]. However, scientific consensus has not yet been formed on the question whether quantum computing with noise actually grants a speed-up [Bha+22].

Recently several Noisy intermediate-scale quantum (NISQ) algorithms have been proposed [Bha+22]. These algorithms can run on quantum devices that already exist but the devices are currently to small for to also run quantum error correction [Bha+22]. An example are variational quantum algorithms which use a classical optimizer and a quantum computer as an oracle for solving the original problem [Cer+21]. To this end, parameterized quantum algorithms are used and the parameters are trained with the classical optimizer [Cer+21]. A prominent example of variational quantum algorithms is the Quantum Approximate Optimization Algorithm (QAOA) [FGG14]. QAOA can be used to approximately solve a plethora of combinatorial optimization problems, such as MaxCut [SH22; Cer+21]. It is an open question which performance we can expect from QAOA in these scenarios [SH22]. Answering this question is a topic of active research [SH22; Bha+22; Cer+21]. In answering this question, it is often valuable to be able to convert a parameterized circuit to a simplified analytical expression [BK21].

Recent years have also given rise to a diagrammatic language for reasoning about quantum circuits [CK17; Wet20]. This language called ZX-calculus naturally expresses composition and parallel execution of processes [CK17].

In [SH22], Stollenwerk et al. tackle the problem of extracting simplified analytical expressions from parameterized quantum circuits using ZX-calculus. To do so they introduced linear combinations of ZX-diagrams. Omitting formal category theoretic considerations in an application driven approach [SH22]. The first goal of this thesis is to close this gap. Stollenwerk et al. used their linear combinations to simplify some parameterized quantum circuits by hand to reproduce results from literature as a demonstration of their method [SH22]. However, an extension towards automatic calculation would facilitate new insights into the performance of variational quantum algorithms [SH22]. Here we focus on automating simplification of diagrams used in QAOA for solving MaxCut instances. First, we implement linear combinations, including rewrite rules in a library for manipulating ZX-diagrams. Candidates include: PyZX [KW20], DisCoPy [FTC20] and Quantomatic [pro18]. Second we develop an algorithm which uses linear combinations to simplify symbolic ZX-diagrams stemming from QAOA.

In summary, the main contributions of this thesis are:

- Formal definition of linear combinations of ZX-diagrams in category theoretic terms
- An implementation of new definitions and rewrite-rules in a combination of DisCoPy [FTC20] and PyZX [KW20].
- An algorithm which uses linear combinations to simplify symbolic ZX-diagrams of parameterized quantum circuits expectation value to analytical expressions stemming from QAOA for MaxCut.

The remainder of this thesis is structured as follows: In Chapter 2 we lay necessary foundations from the areas of Quantum Mechanics, Category Theory and Combinatorial Optimization. We establish the scientific context of our work in Chapter 3. Building on these foundations we formally introduce linear combinations of ZX-diagrams in Chapter 4. Furthermore in Chapter 5 we give a prototypical algorithm for simplifying symbolic ZX-diagrams of a certain shape. In Chapter 6 we summarize, discuss our results and give an outlook for future work.

2. Basics

2.1. Quantum Computing

Analogously to a basic understanding of electronics being required to reason about classical computers, a basic understanding of quantum mechanics is required to reason about quantum computers. Therefore we will state the basics of how we describe closed quantum mechanical systems. In the future it may turn out that the universe does not actually work as currently described. For now, we will assume it does as we define below to ease considerations. We will omit any derivations and refer the reader to [NC02] for a more complete introduction suitable for computer scientists.

2.1.1. Mathematical prerequisites

In this section, we introduce some mathematical prerequisites. Since we only concern ourself with finite dimensional systems we exclude infinite dimensional cases.

Definition 2.1: Hilbert space

A *Hilbert Space* is a vector space V over a field with an inner product. Note: Such an inner product is a map $(\cdot,\cdot) : V \times V \to \mathbb{C}$, such that for $v, w \in V$:

1. $(\cdot,\cdot)$ is linear in its second argument
2. $(v, w) = (w, v)^*$ (where * denotes the complex conjugation)
3. $(v, w) \geq 0$, with equality iff $v = 0$

[NC02]

Furthermore we introduce what is known as *Dirac* notation [NC02]. Given some Hilbert space V over a field K, we have the following conventions:

Definition 2.2: Dirac notation

Given a matrix A over K

- $|v\rangle$ represents a vector in V, called *ket*, with label v
- $\langle v|$ dual to $|v\rangle$, called *bra*
- $\langle v|w\rangle$ inner product of (v, w)
- A^* complex conjugate of A
- A^T transpose of A
- $A^\dagger := (A^T)^*$ adjoint of A
- I identity matrix

[NC02]

We make note of some further conventions. In this entire thesis we always choose the canonical basis for any Hilbert Space we encounter. Given some basis we number the basis vectors and label kets representing the basis vectors with the binary representation of the number. For example in $\mathbb{C}^4$ the canonical basis is given as $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ and in $\mathbb{C}^2$ as $\{|0\rangle, |1\rangle\}$. This convention is extended further :

$$|+\rangle := \frac{|0\rangle + |1\rangle}{\sqrt{2}} \quad \text{and} \quad |-\rangle := \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

[HV19]

Additionally we define a very prominent subset of matrices in a Hilbert space.

Definition 2.3: Unitary matrix

Given a matrix A over K it is called *unitary* if and only if its adjoint is its inverse. So in Dirac notation $A^\dagger = A^{-1}$. Linear transformations given by unitary matrices preserve inner products.
Note: We sometimes use the word *unitary* as a noun and take it to mean unitary matrix. [GS18]

2.1.2. A very short tour of why quantum mechanics is surprising

Several experiments [Dem16] have shown that the physical universe on a quantum scale does not behave as we would expect from studying behaviours on a larger scale. This has lead to large disturbance in the world of physics [GS18].

There are several philosophical interpretations of quantum mechanics [GS18]. For this work the philosophical point is rather irrelevant, since we only exploit the way the universe works to our advantage. Here we do not need the reasons *why* the universe works in this way, we are only interested in a practical description of *how*, in order to use it. In fact, we are in good company doing so, since Richard Feynman remarked *I think I can safely say that nobody understands quantum mechanics.* [GS18].

One experiment demonstrating weirdness of quantum mechanics is the double-slit experiment [FLS15]. In this experiment we shot electrons at a wall with two slits and detect where on a second wall behind the electrons land [FLS15]. See Figure 2.1 for a schematic view of the experimental setup and two theoretical possible measurement outcomes [FLS15]. On the left we see the distribution we expect, if we regard electrons as particles [FLS15]. If instead we view electrons as wave we expect the right pattern [FLS15]. Running the experiment as described will yield the right pattern, seemingly confirming that electrons behave like waves [FLS15]. Interestingly if we now slightly modify the experiment by measuring at one of the slits whether an electron has passed it, we get the left pattern [FLS15]. This rather surprising result leads to several important implications [FLS15]. Explaining them in detail is well beyond the scope of this work. Therefore, we focus on the following two facts we notice: First, that measurements can change the outcome of an experiment [FLS15]. Second, that if not observed the electron travels like a wave, seemingly through both slits at the same time [FLS15].

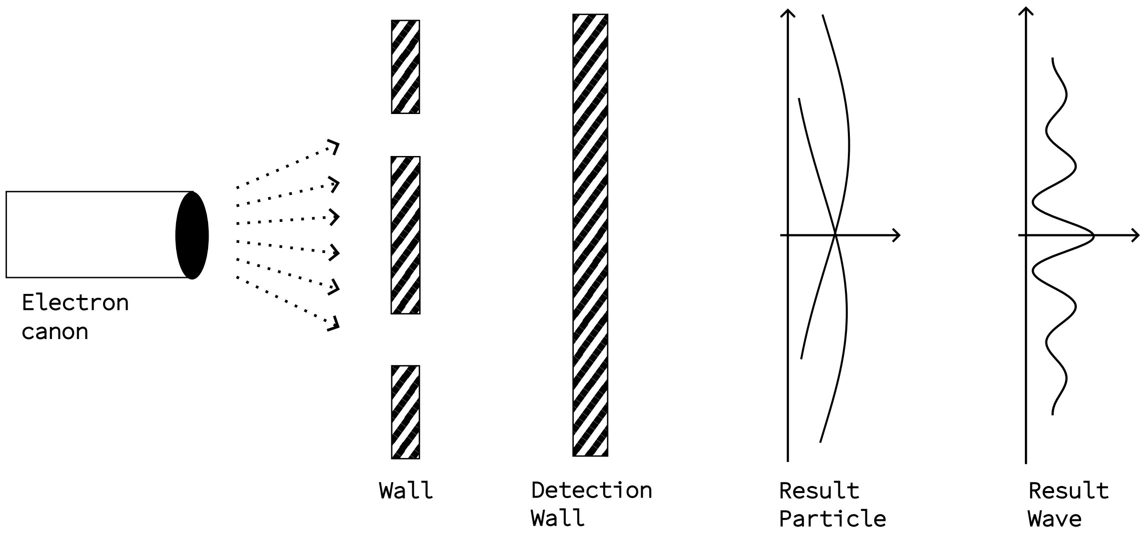


Figure 2.1.: Schematic setup of double slit experiment, with two theoretical possible outcomes

The second is akin to our central idea, that the state of a system does not have to be fully determined if it is not observed [GS18]. This is often described as systems being in several states at once [GS18].

2.1.3. Mathematical description of quantum mechanics

In order to formalize the outlined weirdness of quantum mechanics, a mathematical formulation has been chosen. It can be summarised in five postulates which we will state without a formal justification. We utilize this as layer of abstraction above the physical implementation of such experiments.

Definition 2.4: Postulate 1: Closed quantum mechanical system

A closed quantum mechanical system can be described by a Hilbert space. This is called *state space*.
Each state of the system is completely described as a vector in this space. This is called *state vector*. [NC02]

For a full description we also need to describe how time evolution works in these systems:

Definition 2.5: Postulate 2: Time evolution

Time evolution of a closed quantum system is always a unitary transformation of the state space. [NC02]

Gaining information about the state of a system from outside the system is typically done via measurements. The third postulate explains how and what we may measure:

Definition 2.6: Postulate 3: Measurements

Measurements of a quantum mechanical system are described by a set of measurement operators.
These operators M_i act on the state space and each corresponds to one possible measurement output.
The probability of a system in state $|\varphi\rangle$ yielding result i is given by $\langle\varphi| M_i^\dagger M_i |\varphi\rangle$.
After measuring result i the system is in state

$$\frac{M_i |\varphi\rangle}{\sqrt{\langle\varphi| M_i^\dagger M_i |\varphi\rangle}}$$

The measurements operators have to satisfy the completeness equation which ensures that the probability for the different results is normalized.

$$\sum_i M_i^\dagger M_i = I$$

[NC02]

Observables Observables are hermitian operators on the state space of a system [NC02]. These may be regarded a special case of Postulate 3 [NC02]. They will project the state to one of the eigenvectors of the operator [NC02]. These hermitian operators allow for a measurement to be defined in one matrix and not via a set of operators.

The final postulate allows us to combine systems and describes how they behave then.

Definition 2.7: Postulate 4: Composite systems

Physical systems can be composed using the tensor product of the state spaces. The resulting space is the state space of the composed system. [NC02]

Using these postulates we can describe quantum mechanical processes mathematically and predict their outcomes.

Expectation values One example of predicting the outcome of experiments is calculating the expectation value of an operator. Consider an observable U for a state space $\mathbb{C}^d$ with a matrix representation $U \in Mat_{\mathbb{C}}$. Given a state $|\varphi\rangle \in \mathbb{C}^d$ we can calculate the expectation value of the operator with respect to the state as: $\langle\varphi| U |\varphi\rangle$. This is the value we expect to measure if we observe U in a system in state $|\varphi\rangle$.

2.2. Quantum circuits

When working with a gate based quantum computer programs can be expressed as circuits [NC02]. In order to understand quantum circuits we first have to understand how information is stored in a quantum computer. To this end, we introduce the concept of qubits:

Definition 2.8: Qubit

A qubit is an abstract representation of a single bit of information.
The state space of a single qubit is $\mathbb{C}^2$. Two linear independent vectors, commonly $|0\rangle$ and $|1\rangle$, are interpreted as zero and one respectively.

Qubits are analogous to classical bits in classical computers as they can store a value 0 or 1. They differ since a qubit can also store, in some sense, values in between the two.

Qubits can have different physical representations [NC02]. We will not concern ourself with how qubits are represented in our applications but will just assume there to be some sensible representation.

Qubits can be combined as defined in Postulate 4, in order to represent larger chunks of information. This combination is called *qubit register*.

Definition 2.9: Qubit register

A qubit register of size n stores the composite state of n qubits.
Given n qubits each with state $|\phi_i\rangle \in \mathbb{C}^2$ the composite state is given by

$$|\phi\rangle = |\phi_1\rangle \otimes \cdots \otimes |\phi_n\rangle \in \mathbb{C}^{2n}$$

Further if we have two qubits in a a computational basis state, for example $|0\rangle$ and $|1\rangle$, we write the state of the register, consisting of both qubits, as $|01\rangle$. [NC02]

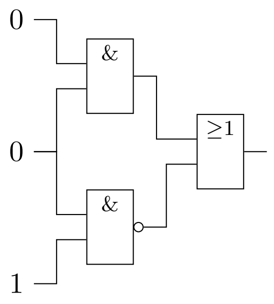


Figure 2.2.: Example for a classical circuit, with inputs 0, 0, 1 [Tan21]

Rather informally a classical computer can be described as an experiment involving lots of complex electronics. These electronics receive inputs and the system reacts to these inputs in ways predefined by whoever built the computer. We could just always plug different electronic components into each other and perform computations this way. These systems get complex rather fast and a consistent description of the circuits has been developed. Typically classical computers are described by combinations of gates from a specific set [Hof20]. This allows for easier reasoning by abstracting away the actual implementation in hardware [Hof20]. These gates can be described by their logic tables; examples can be found in Table 2.1. They may be combined to form a circuit (for an example see Figure 2.2).

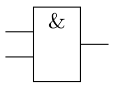
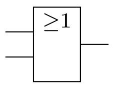
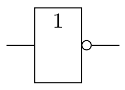
Logic operation	Notation according to DIN 40900
AND	
OR	
NOT	

Table 2.1.: Overview of classical circuit symbols [Hof20]

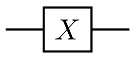
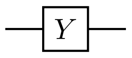
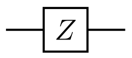
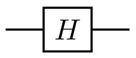
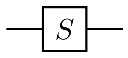
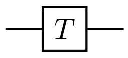
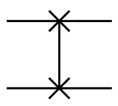
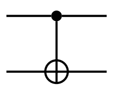
Canonical name	Symbol in circuit	Matrix representation, in the canonical basis
Pauli-X		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli-Y		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli-Z		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard		$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
Phase		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$\pi/8$		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$
SWAP		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
C-NOT		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$

Table 2.2.: Overview of typical quantum logic gates, definitions from [NC02]

For quantum computing, a similar step has been taken in recent years [NC02]. These gates typically have a small number of qubits as its input and output. They are all valid time operators and can therefore be represented by unitary matrices [NC02]. See Table 2.2 for an overview of gates typically used, although we may use arbitrary unitary matrices as gates. As in classical computing these gates may be combined in any sensible way to form what we call a circuit. For an example circuit see Figure 2.3.

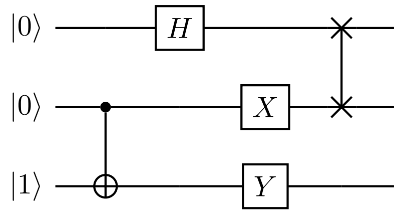


Figure 2.3.: Example for a quantum circuit, with inputs $|0\rangle, |0\rangle, |1\rangle$

In classical computing, the concept of universal operator sets has been convenient in asserting the usefulness of physical implementations [Hof20]. Once a computer can represent all gates in a universal operator set, it can represent any circuit [Hof20].

In order to reason about quantum computing, several sets of universal operator sets have been defined [NC02]. These consist of several one or two qubit gates which can (up to arbitrary precision) represent any possible operator for any system [NC02]. A typical universal operator set is given in [NC02] containing the Hadamard, Phase, C-NOT and $\pi/8$ gates.

2.2.1. Parameterized quantum circuits

A parameterized quantum circuit is a family of quantum circuits depending on a set of parameters [Cer+21]. Each assignment of values to the parameters gives a quantum circuit [Cer+21]. One can only evaluate the quantum circuit with all parameters set to some value. Informally, from a computer scientists view, this is akin to template programming in C++; before running the program, values for all template parameters need to be defined. Consider the following matrix:

$$U(x) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/x} \end{bmatrix}$$

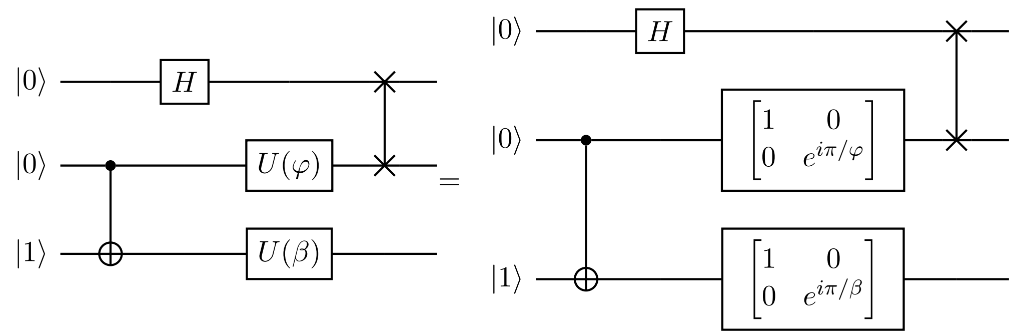
It is unitary and can therefore be used as a gate in a quantum circuit. That circuit then depends on the parameter which is given to U. An example depending on $\{\varphi, \beta\}$ can be seen in Figure 2.4.

2.2.2. Variational Quantum Algorithms

Variational Quantum Algorithms (VQAs) leverage the power of classical as well as quantum computers [Cer+21]. Mostly they are used for solving combinatorial optimization problems [Cer+21]. Using a classical optimizer, they train parameters of a parameterized quantum circuit [Cer+21]. The quantum circuit is used as an oracle for cost function (or its gradient), which is classically hard to compute [Cer+21].

2.3. Category Theory

In the following we will review some basics of category theory which are needed to understand the construction of linear combinations in ZX-calculus. Only the concepts required for the construction will be presented. It is not intended as an introduction to category theory for a new reader, but only serves to avoid ambiguity later on. This is therefore an incomplete view of the basics of category theory and the interested reader should consider reading an introductory book like [Mac13].



$|0\rangle$
$|0\rangle$
$|1\rangle$
H
$U(\varphi)$
$U(\beta)$
$=$
$|0\rangle$
$|0\rangle$
$|1\rangle$
H
$\begin{bmatrix}1 & 0\\ 0 & e^{i\pi/\varphi}\end{bmatrix}$
$\begin{bmatrix}1 & 0\\ 0 & e^{i\pi/\beta}\end{bmatrix}$

Figure 2.4.: A quantum circuit, with inputs $|0\rangle, |0\rangle, |1\rangle$, and parameters φ and β

The first definition we will concern ourself with is the one of a basic category:

Definition 2.10: Category

A *category* $\mathcal{C}$ consists of a collection of objects $Ob(\mathcal{C})$ and a collection of arrows (sometimes called morphisms), fulfilling these axioms:

1. For each arrow f there are objects *domain* $\mathrm{dom}(f)$ and *codomain* $\mathrm{cod}(f)$. We write $f : \mathrm{dom}(f) \to \mathrm{cod}(f)$.
2. Given $f : A \to B$ and $g : B \to C$ there is an arrow $g \circ f : A \to C$ called the *composite* of f and g.
3. For each object A there is an arrow $1_A : A \to A$ called the *identity arrow* of A.
4. Associativity: $h \circ (g \circ f) = (h \circ g) \circ f$ for all $f : A \to B$, $g : B \to C$ and $h : C \to D$
5. Unit: $f \circ 1_A = f = 1_B \circ f$ for all $f : A \to B$

[Awo06]

One example of a category that is interesting in the case considered here is the category $Mat_{\mathbb{C}}$.

Example 2.11: $Mat_{\mathbb{C}}$ is a category

We define a category $Mat_{\mathbb{C}}$ with the following data:

- the class of objects as the natural numbers $\mathbb{N}$ representing the complex vector spaces $\mathbb{C}^n$
- the class of morphisms as the linear maps between two vector spaces.

Note: The name $Mat_{\mathbb{C}}$ makes sense here, since linear maps between complex vector spaces are exactly the matrices over $\mathbb{C}$ [HV19]. Composition is defined as matrix multiplication.

Proof We check the axioms one by one, let $A, B, C, D \in Ob(Mat_{\mathbb{C}})$ and $f : A \to B$ and $g : B \to C$.

1. f has domain $dom(f)$ the number of columns of the matrix and co-domain $codom(f)$ the number of rows of the matrix
2. the composite of f and g is given by their matrix product, which fulfils the $g \circ f : A \to C$ property
3. the identity arrow on A is given by the identity matrix of size A
4. since matrix multiplication is associative, the morphisms are too
5. by definition the identity matrix is the neutral element of matrix multiplication, so the unit property is also fulfilled

We notice that a category alone does not have much structure. In order to describe quantum processes a specific type of category called *monoidal category* has prevailed [HV19]. Our scenarios are in fact nicely described using *strict monoidal categories*. Also each *monoidal category* is monoidally equivalent to a *strict monoidal category* so for reasons of conciseness we restrict our presentation to strict monoidal categories [HV19]. So we define the latter and refer the reader to [Bor94] for a definition of general *monoidal categories*.

Definition 2.12: Strict Monoidal Category

A *strict monoidal category* $\mathcal{C}$ is given by the following:

1. a category $\mathcal{C}$
2. a *bifunctor* $\boxdot : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$
3. an object $I \in \mathcal{C}$, called the unit
4. for every triple A, B, C of objects, an *associativity* isomorphism $a_{ABC} : (A \boxdot B) \boxdot C \to A \boxdot (B \boxdot C)$ is the identity, so $(A \boxdot B) \boxdot C = A \boxdot (B \boxdot C)$
5. for every object A, a *left unit* isomorphism $l_A : I \boxdot A \to A$ is the identity, so $I \boxdot A = A$
6. for every object A, a *right unit* isomorphism $l_A : A \boxdot I \to A$ is the identity, so $A \boxdot I = A$

[HV19]

Intuition for this structure can be gained by considering an example.

Example 2.13: $Mat_{\mathbb{C}}$ as a strict monoidal category

The category $Mat_{\mathbb{C}}$ is a *strict monoidal category* with the monoidal product on objects as the canonical sum on $\mathbb{N}$ and on morphisms as the tensor product over spaces. The unit is then $1 \in Ob(\mathbb{C})$ and the left and right unit morphisms are given by the identity matrix.

Proof : The tensor product of complex vector spaces is of the form $\mathbb{C}^n \otimes \mathbb{C}^m = \mathbb{C}^{m \cdot n}$. So in our category we define $n \otimes m = n \cdot m$.
Now we check the properties:

1. $Mat_{\mathbb{C}}$ is a category
2. $\otimes$ is our bifunctor
3. 1 is our unit
4. for $A, B, C \in Ob(Mat_{\mathbb{C}})$ we have $(A \otimes B) \otimes C = (A \cdot B) \cdot C = A \cdot (B \cdot C) = A \otimes (B \otimes C)$
5. $A \otimes I = A \cdot I = A \cdot 1 = A$
6. $I \otimes A = I \cdot A = 1 \cdot A = A$

So this is a monoidal category. $\square$

Note: One can proof that the tensor product for matrices is exactly what is called *kronecker product* [Mer92]. We define the kronecker product of the morphisms as in [Mer92]: Let $f = (a_{ij})_{i<m,j<n}, g = (b_{ij})_{i<l,j<k}$ be arbitrary matrices over $\mathbb{C}$. Then

$$f \otimes g := \begin{bmatrix} a_{11}g & \dots & a_{1n}g \\ \vdots & \ddots & \vdots \\ a_{m1}g & \dots & a_{mn}g \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & \dots & a_{1n}b_{1k} \\ a_{11}b_{21} & a_{11}b_{22} & \dots & a_{1n}b_{2k} \\ \vdots & \vdots & \ddots & \\ a_{m1}b_{l1} & a_{m1}b_{l2} & \dots & a_{mn}b_{lk} \end{bmatrix}$$

This is how the functor from our example acts on the morphims.

One can understand a monoidal category as a generalisation of a tensor product. There can be multiple different products to make a category into a (strict) monoidal category. It has been proven, that every monoidal category is equivalent to a strict one [HV19]. Therefore, without loss of generality we only consider strict monoidal categories here.

Another extension of a general category is a superposition rule. Literature sometimes also calls this enrichment in commutative monoids [HV19].

Definition 2.14: Superposition rule

Given a category $\mathcal{C}$, a *superposition rule* is an operation $(f, g) \mapsto f + g$ for all morphisms $A \xrightarrow{f,g} B$ such that:

- $f + g = g + f$
- $(f + g) + h = f + (g + h)$
- for all $A, B \in Ob(\mathcal{C})$ there is a $A \xrightarrow{u_{A,B}} B$, such that for all $A \xrightarrow{f} B$: $f + u_{A,B} = f$
- $(g + h) \circ f = (g \circ f) + (h \circ f)$
- $f \circ (g + h) = (f \circ g) + (f \circ h)$
- for all $f : A \to B$ and $C, D \in Ob(C)$: $u_{C,B} = f \circ u_{C,A}$ and $u_{A,D} = u_{B,D} \circ f$

[HV19]

Intuition for this structure can be gained by considering an example.

Example 2.15: $Mat_{\mathbb{C}}$ has a superposition rule

The category $Mat_{\mathbb{C}}$ has a *superposition rule* defined as canonical matrix addition: For morphisms $f = (a_{ij})_{i<m,j<n}, g = (b_{ij})_{i<l,j<k}$ we define their sum as:

$$f + g = (a_{ij} + b_{ij})_{i<m,j<n}$$

With $u_{A,B}$ the zero map it is fairly obvious that the axioms are fulfilled. Hence, addition is a superposition rule in $Mat_{\mathbb{C}}$

One may define a *superposition rule* for any category, so in particular also for (strict) monoidal ones. As we in fact have done in the example above. One question that arises is whether the monoidal structure and the superposition rule behave nicely with one another. It is not clear that any sort of interaction between the two is guaranteed. In a special case we can give some statements about their relationship, but in order to describe this case we need two more preliminary definitions.

Definition 2.16: Biproduct

Let $\mathcal{C}$ be a category with a zero object and a superposition rule. The biproduct of $A_1, A_2 \in Ob(\mathcal{C})$ is an object $A_1 \bigoplus A_2$ with *injection* morphisms $A_n \xrightarrow{i_n} A_1 \bigoplus A_2$ and *projection* morphisms $A_1 \bigoplus A_2 \xrightarrow{p_n} A_n$ for $n \in \{1, 2\}$, such that:

- $id_{A_n} = p_n \circ i_n$
- $0_{A_n,A_m} = p_m \circ i_n$
- $id_{A_1 \oplus A_2} = i_1 \circ p_1 + i_2 \circ p_2$

[HV19]

Example 2.17: $Mat_{\mathbb{C}}$ admits arbitrary biproducts

For our category $Mat_{\mathbb{C}}$ the biproduct of two objects given by the direct sum of vector spaces $\mathbb{C}^{n_1} \oplus \mathbb{C}^{n_2} \cong \mathbb{C}^{n_1+n_2}$ [HV19]. The injection and projection morphisms are the canonical ones.

Proof Without loss of generality we assume the vector space to be equipped with the standard basis. The zero object is given by the empty vector space $\mathbb{C}^0$, with the zero maps. Then the injection morphism $i_1 : \mathbb{C}^{n_1} \to \mathbb{C}^{n_1+n_2}$ maps $a \mapsto (a, 0)$ and $i_2 : \mathbb{C}^{n_2} \to \mathbb{C}^{n_1+n_2}$ maps $a \mapsto (0, a)$. The projection morphisms map:

$$\begin{aligned} p_1 :&\mathbb{C}^{n_1+n_2} \to \mathbb{C}^{n_1} \\ &(a, b) \mapsto a \end{aligned} \qquad \begin{aligned} p_2 :&\mathbb{C}^{n_1+n_2} \to \mathbb{C}^{n_2} \\ &(a, b) \mapsto b \end{aligned}$$

As these are linear maps they can be expressed as matrices, so they are morphisms in $Mat_{\mathbb{C}}$.

Now we check the desired properties: Let $a \in A_1, b \in A_2$ be arbitrary but fixed.

$$p_1 \circ i_1(a) = p_1((a, 0)) = a \Rightarrow id_{A_1} = p_1 \circ i_1$$

since a is arbitrary.

$$p_2 \circ i_2(b) = p_2((0, b)) = b \Rightarrow id_{A_2} = p_2 \circ i_2$$

since b is arbitrary.

$$p_1 \circ i_2(b) = p_1((0, b)) = 0 \Rightarrow 0_{A_1,A_2} = p_1 \circ i_2$$

since b is arbitrary.

$$p_2 \circ i_1(a) = p_2((a, 0)) = 0 \Rightarrow 0_{A_2,A_1} = p_2 \circ i_1$$

since a is arbitrary.

Note that matrix multiplication and addition have a distribution law, so this law also holds for our morphisms, when expressed as matrices.

$$
\begin{aligned}
(i_1 \circ p_1 + i_2 \circ p_2)(a,b) &= (i_1 \circ p_1)(a,b) + (i_2 \circ p_2)(a,b) \\
&= (i_1)(a) + (i_2)(b) \\
&= (a,0) + (0,b) \\
&= (a,b) \\
\Rightarrow (i_1 \circ p_1 + i_2 \circ p_2) &= id_{A_1 \oplus A_2}
\end{aligned}
$$

□

Definition 2.18: Dual

Let $\mathcal{C}$ be a monoidal category with operation $\boxdot$ and unit objects I, with objects L and R. L is the *left-dual* to R and R is the *right-dual* to L, if there exists a unit morphism $I \xrightarrow{\eta} (R \boxdot L)$ and a counit morphism $L \boxdot R \xrightarrow{\epsilon} I$ making the following diagrams commute:

$$\begin{tikzcd}
L \arrow[r, "\rho_L^{-1}"] \arrow[d, "id_L"'] & L \boxdot I \arrow[r, "id_L \boxdot \eta"] & L \boxdot (R \boxdot L) \arrow[d, "\alpha^{-1}_{L,R,L}"'] \\
L & I \boxdot L \arrow[l, "\lambda_L"] & (L \boxdot R) \boxdot L \arrow[l, "\varepsilon \boxdot id_L"]
\end{tikzcd}$$

$$\begin{tikzcd}
R \arrow[r, "\lambda_R^{-1}"] \arrow[d, "id_R"'] & I \boxdot R \arrow[r, "\eta \boxdot id_R"] & R \boxdot (L \boxdot R) \arrow[d, "\alpha_{R,L,R}"'] \\
R & R \boxdot I \arrow[l, "\rho_R \boxdot id_L"] & (R \boxdot L) \boxdot R \arrow[l, "id_R \boxdot \varepsilon"]
\end{tikzcd}$$

[HV19]

Example 2.19: $Mat_{\mathbb{C}}$ admits arbitrary duals

Each $A \in Ob(Mat_{\mathbb{C}})$ is its own left and right dual. In particular $Mat_{\mathbb{C}}$ admits arbitrary duals.

Proof Consider the maps

$$\rho : 1 \mapsto \sum_i |i\rangle \otimes |i\rangle \qquad \varepsilon : |i\rangle \otimes |j\rangle \mapsto \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

These clearly make the diagrams in Definition 2.18 commute. □
[HV19]

With these definitions we get the following nice lemma:

Lemma 2.20: Tensors distribute over superposition

Let $(\mathcal{C}, \boxdot)$ be a monoidal category with biproducts and objects A, B, C, D with morphisms $A \xrightarrow{f} B$ and $C \xrightarrow{g,h} D$. If A has either a left or a right dual, we have:

$$(f \boxdot g) + (f \boxdot h) = f \boxdot (g + h)$$

$$(g \boxdot f) + (h \boxdot f) = (g + h) \boxdot f$$

[HV19]

The proof can be found in [HV19] as Lemma 3.22. Moreover, we are not only interested how categories behave but also how they interact with one another. For this purpose we introduce the concept of a *functor*, which describes how one can change from category to category.

Definition 2.21: Functor

Given two categories $\mathcal{C}$ and $\widetilde{\mathcal{C}}$, a *functor* $F : \mathcal{C} \to \widetilde{\mathcal{C}}$ is given by

- $\forall A \in Ob(\mathcal{C}) \; : \; F(A) \in Ob(\widetilde{\mathcal{C}})$
- $\forall$ morphisms $A \xrightarrow{f} B$ in $\mathcal{C}$: $F(A) \xrightarrow{F(f)} F(B)$ is a morphism in $\widetilde{\mathcal{C}}$

These maps satisfy:

- $\forall$ morphisms $A \xrightarrow{f} B$, $B \xrightarrow{g} C$ in $\mathcal{C}$: $F(g \circ f) = F(g) \circ F(f)$
- $\forall A \in Ob(\mathcal{C}) : F(id_A) = id_{F(a)}$

[HV19]

We will need some additional properties functors may have, which are defined as follows:

Definition 2.22: Properties of functors

- A functor F is called *full* iff all functions $\mathcal{C}(A, B) \to \widetilde{\mathcal{C}}(F(A), F(B))$ given by $f \mapsto F(f)$ are surjective for all $A, B \in Ob(\mathcal{C})$.
- A functor F is called *faithful* iff all functions $\mathcal{C}(A, B) \to \widetilde{\mathcal{C}}(F(A), F(B))$ given by $f \mapsto F(f)$ are injective for all $A, B \in Ob(\mathcal{C})$.
- A functor is *essentially surjective on objects* iff for all $B \in Ob(\widetilde{\mathcal{C}})$ there is an object $A \in Ob(\mathcal{C})$ such that $B \equiv F(A)$.
- A functor is an *equivalence* iff it is *full*, *faithful* and *essentially surjective on objects*.

[HV19]

Definition 2.23: Equivalence of categories

Two categories $\mathcal{C}, \widetilde{\mathcal{C}}$ are called *equivalent* if there exists a functor between the two which is an equivalence.
Note: If two categories are equivalent then from a category theoretic point it does not matter with which one we work with. [HV19]

2.4. ZX-calculus

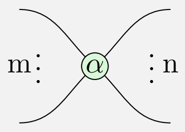
ZX-calculus is a graphical language for reasoning about quantum processes. It can represent arbitrary quantum operators and systems. The basic idea is represent quantum operators by composition and tensor product of certain linear operations. This representation in form of diagrams can be transformed to diagrams representing equivalent operators via rewrite rules which only make local changes.

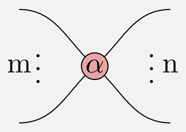
In the next section we will first explore how ZX-diagrams are built, which rewrite rules exist and how they can be interpreted as quantum systems and operators. Afterwards we will see that the interpretation for quantum systems is sound, universal and complete. Then a category theoretic view will be presented which will aid us in reaching the goal of this thesis, to formally define linear combinations of ZX-diagrams.

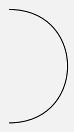
2.4.1. How to build diagrams

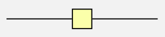
ZX-diagrams only have a very limited number of building blocks, called generators, and these may be combined arbitrarily. The building blocks we have are:

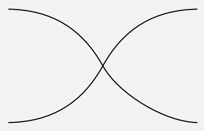
Definition 2.24: ZX-Generators

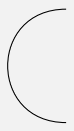
Z-spiders:


X-spiders:


Cup:


Hadamard:


Swap:


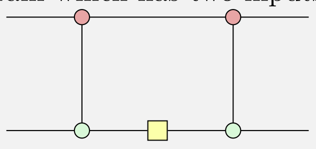
Cap:


[Wet20]

The spiders may contain a phase, if none is given we default to 0 [Wet20]. All diagrams in this thesis are to be read from left to right. Bare wires pointing to the left are regarded as inputs and bare wires pointing the the right are regarded as outputs. Bare wires not pointing in one of these directions constitute undefined behaviour.

Example 2.25: An arbitrary ZX-diagram

This is an arbitrary ZX-diagram which has two inputs and two outputs.

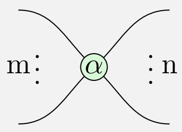


2.4.2. ZX-diagram as quantum operators

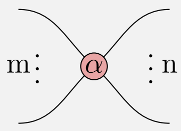
ZX-diagrams may be interpreted as a quantum operator, effect or state depending on the number of inputs and outputs. Operators without inputs are called state and operators without outputs are called effects. So for each ZX-diagram and a given basis, we can assign a matrix to it called its interpretation. This interpretation is how the diagram evolves an element of the state space. [Wet20]

To obtain an interpretation for any ZX-diagram, we start by defining the interpretation for the generators and then defining the interpretation for their interaction. So if we had a diagram that just consisted of a single generator it would have the interpretation stated below.

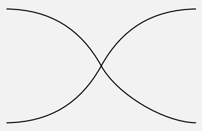
Definition 2.26: Generator Interpretations Overview

Z-spiders: 

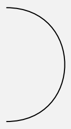
$$\widehat{=} \underbrace{|0\dots0\rangle}_{\text{n times}}\underbrace{\langle 0\dots 0|}_{\text{m times}} + e^{i\alpha}\underbrace{|1\dots1\rangle}_{\text{n times}}\underbrace{\langle 1\dots 1|}_{\text{m times}}$$

X-spiders: 

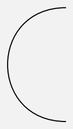
$$\widehat{=} \underbrace{|+\cdots+\rangle}_{\text{n times}}\underbrace{\langle +\cdots +|}_{\text{m times}} + e^{i\alpha}\underbrace{|-\cdots-\rangle}_{\text{n times}}\underbrace{\langle -\cdots -|}_{\text{m times}}$$

Swap: 

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

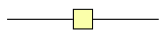
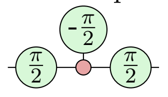
Cup: 

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

Cap: 

$$\widehat{=} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

[Wet20]

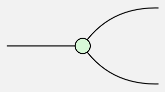
The hadamard generator can be decomposed into Z and X spiders in several different ways [Wet20]. As an example we give  $\widehat{=}$ 

[Wet20].

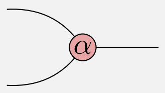
Interpretation of composition of these generators is represented by normal matrix multiplication [Wet20].

Example 2.27: Composition Interpretation

Consider these two simple diagrams and their interpretations, which compose nicely:

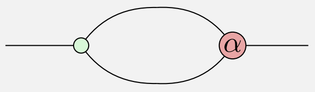


$$|00\rangle\langle 0| + e^{i\cdot 0}|11\rangle\langle 1|$$



$$|+\rangle\langle ++| + e^{i\alpha}|--\rangle\langle -|$$

The interpretation of the composed diagram is:

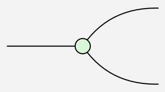


$$(|+\rangle\langle ++| + e^{i\alpha}|--\rangle\langle -|)\cdot(|00\rangle\langle 0| + e^{i\cdot 0}|11\rangle\langle 1|)$$

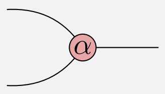
Parallel combination is represented via the tensor product of vector spaces [Wet20].

Example 2.28: Parallel combination interpretation

We again consider these two simple diagrams and their interpretations:

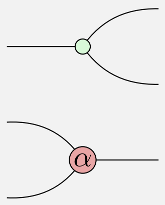


$$|00\rangle\langle 0| + e^{i\cdot 0}|11\rangle\langle 1|$$



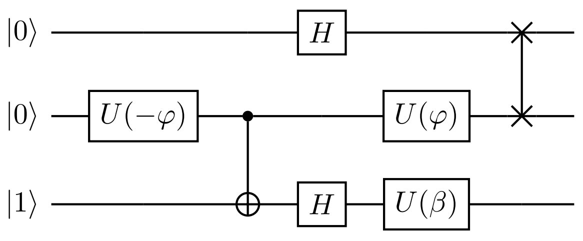
$$|+\rangle\langle ++| + e^{i\alpha}|--\rangle\langle -|$$

The interpretation of the composed diagram is:

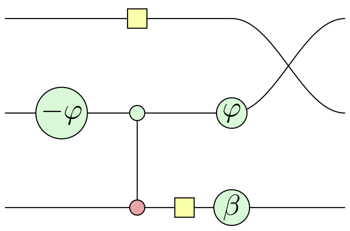


$$(|00\rangle\langle 0| + e^{i\cdot 0}|11\rangle\langle 1|)\otimes(|+\rangle\langle ++| + e^{i\alpha}|--\rangle\langle -|)$$

We consider a circuit example similar to before and its representation as a ZX-diagram in Figure 2.5. The conversion is done by keeping wires the same and replacing gates with the corresponding ZX-generators. Notice that we choose U exactly such that it corresponds to a Z-spider. This is not in general the case, for more complicated unitaries we might have to first construct a ZX-subdiagram representing said unitary.



(a) Quantum circuit



(b) Equivalent ZX-diagram

Figure 2.5.: Example conversion from quantum circuit to ZX-diagram

Note that in our diagrams we will sometimes use transparent squares labelled with a matrix. These squares are not to be understood as generators of ZX-calculus. They are merely a shorthand to represent an arbitrary diagram with the labelling matrix as interpretation. This is following the representation of gates in quantum circuits.

2.4.3. How to modify diagrams

Diagrams can be changed in a consistent way in two fundamentally different ways. The first way is less intrusive: ZX-diagrams follow the rule that *Only connectivity matters* [Wet20]. This nice slogan means that elements of ZX-diagrams may be arbitrarily moved around on the plane, as long as the order of the inputs and outputs does not change [Wet20].

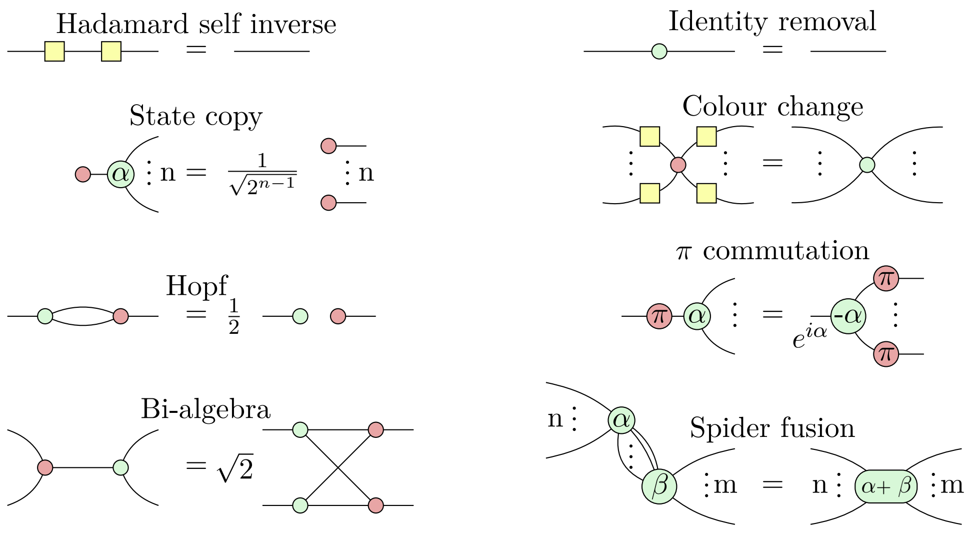
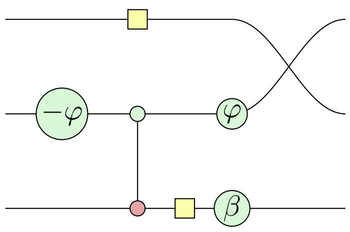
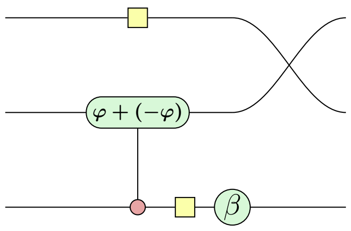


Figure 2.6.: ZX-calculus rewrite rules, including global phases

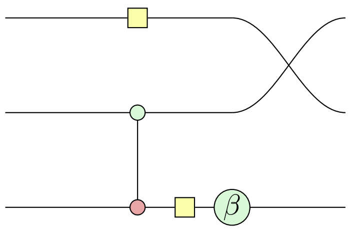
The second one are a number of rewrite rules which actually change generators present in the diagram. There are various rewrite rules and different authors use different sets which all basically accomplish the same [Wet20]. We present a subset of rewrite rules given in [SH22] in Figure 2.6. These have some nice properties, which we discuss in the next section. Included in the formulation are global phases[1] rules. We introduce these since global phases are relevant for our application. Typically global phases are dropped, since they are irrelevant for most applications [Wet20]. All these rules also hold with the colours exchanged, also α and β may take any value in $\mathbb{R}$ [Wet20].



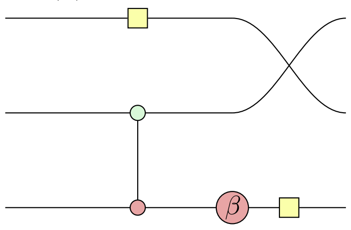
(a) Example ZX-diagram representing circuit from before



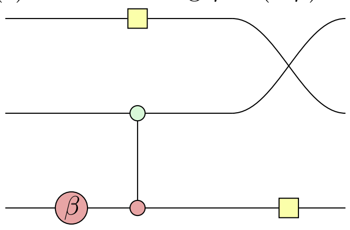
(b) After spider-fusion



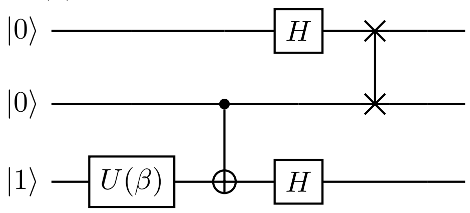
(c) After realizing $\varphi + (-\varphi) = 0$



(d) After Hadamard colour swap



(e) After spider-fusion and de-fusion in the other direction



(f) Quantum circuit

Figure 2.7.: Example derivation using ZX-calculus

As an example for how to apply these rules consider Figure 2.7. Here we first use the spider-fusion rule to see that we can drop both $U(\pm\varphi)$ gates. We then flip the colour on the $U(\beta)$ spider by introducing some Hadamard spiders. This gives us that the circuit in Figure 2.5a is equivalent to the quantum circuit pictured in Figure 2.7f. That both circuits are equal was not clear from their circuit representation, but in ZX-calculus the identity was fairly clear.

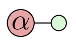
2.4.4. Soundness, universality and completeness

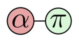
After these definitions we might rightfully ask whether representing quantum systems in this way is sensible. This question has three different facets which have answers of varying complexity. Once all these questions have been answered, we see that ZX-calculus is a useful and coherent tool to reason about quantum systems.

[1]These are represented as complex numbers without a spider in the diagrams.

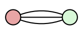
Is ZX-calculus universal? The first question we ask can also be asked as *Can we represent any operator as a ZX-diagram?* The answer to this question is indeed 'Yes' but it goes even further. We can actually represent any linear map between complex vector spaces as a ZX-diagram. This has been shown as the property that ZX-calculus is *universal* [Wet20]. We present a sketch of the proof taken from [Wet20]:

First we show that we can represent arbitrary scalars as ZX-diagrams. For this observe these identities:

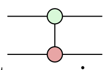
$= 2$  $= \sqrt{2}$

$= 0$ 

$= \sqrt{2}e^{i\alpha}$

$= 1 + e^{i\alpha}$  $= \frac{1}{\sqrt{2}}$

It suffices to show, that we can represent any $x \in \mathbb{C}$ with $|x| < 2$, since for larger x we choose y such that $x = y \cdot \sqrt{2}^k$ for $k \in \mathbb{N}$ and just represent y. Now we show we can represent each such y. Then choose α such that $|y| = |1 + e^{i\alpha}|$, so we get $\frac{y}{1+e^{i\alpha}} = e^{i\beta}$ for some β. This yields $y = (1 + e^{i\alpha})(\sqrt{2}e^{i\beta})\frac{1}{\sqrt{2}}$. So we can represent arbitrary scalars.

In order to now show universality, we need to see that we can represent some universal operator set. The set we choose before in Section 2.2 contains Hadamard, phase, CNOT and $\pi/8$ gates. By checking interpretations, notice that we can easily represent the one qubit gates. The CNOT gate can be represented as  , that can be verified by checking the interpretation as well. Once we represent a universal operator set, we can represent arbitrary unitary operators. □

Is ZX-calculus sound? This question can be informally asked as: *Do the rewrite rules keep the interpretation of a diagram invariant?* If the rewrite rules would not keep the interpretations invariant, the calculus would not be very useful as modifying the diagram would modify the underlying map. This concept is called Soundness of the ZX-calculus. It has been proven, that ZX-calculus is sound [Wet20]. Showing separately for each rewrite rule that it does not change the interpretation of the diagram is a rather tedious process, so the proof is out of scope.

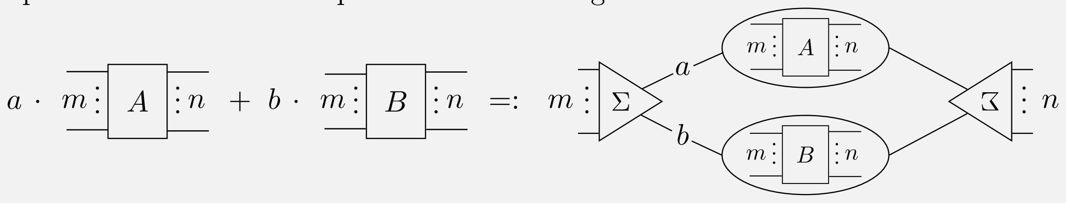
Is ZX-calculus complete? The last question we need to answer is whether using ZX-calculus with a given set of rewrite rules can proof any true equation (in our context). This property is known as *completeness*. Of the three we currently talk about, historically *completeness* is the on that took the longest to proof [Wet20]. There are several different sets of rewrite rules for which *completeness* has been proven [JPV18; HNW18; Wet20]. A historic overview is out of scope but can be found in [Wet20]. The rule set we defined is (to our knowledge) not complete, since we only present rules we will use in the following. It is a subset of the rules used by [HNW18], so by extending our rules set we could make the calculus we use here complete.

2.4.5. Linear combinations in ZX-calculus

Linear combinations of ZX-diagrams have been introduced by Stollenwerk et al. in [SH22]. They added a new symbol to ZX-calculus used to encode sums of diagrams [SH22].

Definition 2.29: Sum notation

In this figure a and b are complex scalars for the diagrams and A and B are unitary operators which can be represented as ZX-diagrams.

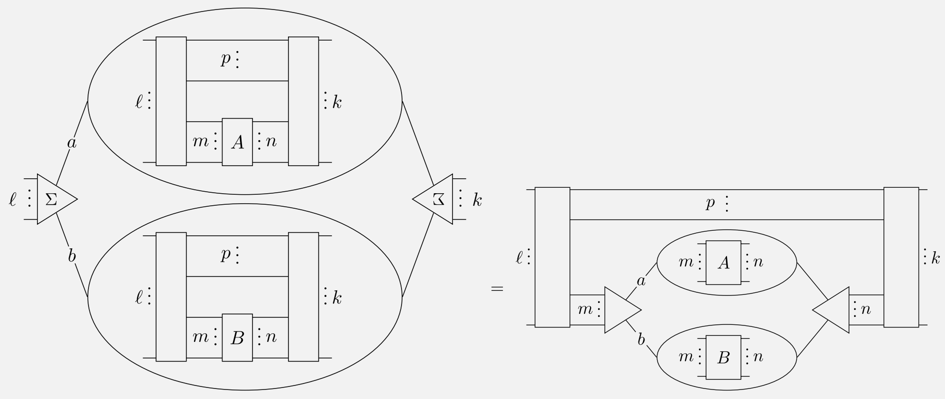


[SH22]

In addition to the diagrammatic definition we also define how the sums are to be interpreted. Each term of the sum is a ZX-diagram and therefore has an interpretation. The interpretation of the sum is given by the interpretations of the terms as their sum. In addition to this definition they also introduced two rewrite rules. [SH22]

The first rule basically means that if all terms of the sum have wires connecting both ends, those wires may be pulled out of the sum. This is a special case of our sum distributing over the tensor product.

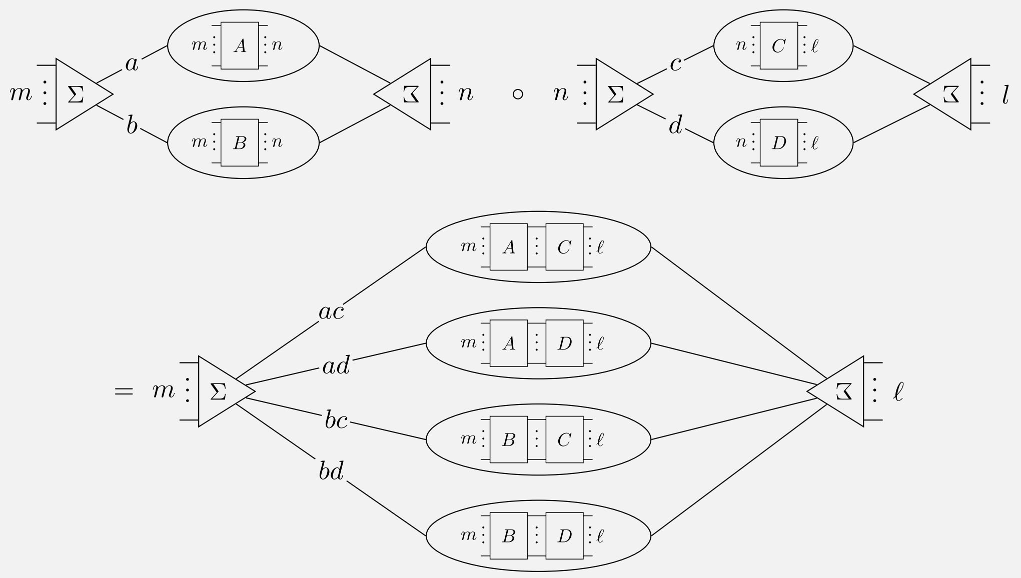
Definition 2.30: Diagram Pull Rule



[SH22]

Additionally Stollenwerk et al. introduced the Product (Composition) Rule. In matrix terms we can write this as $(A + B) \cdot (C + D) = AC + AD + BC + BD$.

Definition 2.31: Product (Composition) Rule



[SH22]

Both of these rules are then used in [SH22] to simplify symbolic ZX-diagrams. Though formal verification that these rules actually hold is omitted in [SH22].

2.4.6. Category theoretic view

In the previous section, ZX-calculus was presented with a focus on how it can be used. This section will introduce the formal foundations this usefulness is built upon.

First we need to define which category we are working with:

Definition 2.32: Category ZX

- **objects** Ob(**ZX**) $= \mathbb{N}$
- **morphisms** from n to m are ZX-diagrams with n inputs and m outputs

[Wet20]

We now need to check that this is actually a category. This fact is known in literature, for example in [Wet20], but we still check the axioms here:

1. Each arrow has domain and codomain as the number of inputs and outputs respectively
2. We compose arrows by composition of diagrams. The interpretation is matrix multiplication, as defined above, since ZX-diagrams are freely generated this is still a valid ZX-diagram
3. The identity arrow for $n \in Ob(\mathbb{N})$ is given by the diagram that has just n wires. These obviously always exist.
4. Since matrix multiplication is associative our composition is as well.

5. Empty wires can be contracted on the left and on the right, so our identity arrows fulfil the unit axiom.

An observant reader may have noticed that this definition does encode composition, but not tensoring diagrams. We can formalize tensoring diagrams using the structure of a monoidal category [Wet20]. **ZX** together with the kronecker product fulfil the required axioms since it is the tensor product of vector spaces of $\mathbb{C}$.

Lemma 2.33: Category ZX is strict monoidal

We show that **ZX** is a strict monoidal category with following data:

1. as category we use **ZX**
2. the bifunctor $\otimes : \mathbf{ZX} \times \mathbf{ZX} \to \mathbf{ZX}$, defined by
   - on objects: $a \otimes b = a + b$
   - on arrows: by writing them above one another, taking as interpretation the tensor product on complex vector spaces
3. the unit object given by $0 \in \mathbf{ZX}$
4. the associativity isomorphism is the identity, since addition in $\mathbb{N} = Ob(\mathbf{ZX})$ is associative
5. since 0 is the neutral element of addition the left and right unit isomorphisms are the identity

[Wet20] □

It is fairly natural to consider two diagrams as, in some sense, equal if they have the same interpretation as matrices. This is exactly what the next definition does:

Definition 2.34: Interpretation equivalence relation

We define an equivalence relation on the morphisms of **ZX**. Two morphisms f and g are equivalent if and only if their interpretation is equal, more formally $[f] = [g]$. We then write $f \approx g$
[Wet20]

With this definition we go on to define another useful category.

Definition 2.35: Category $^{\mathrm{ZX}}/_{\sim}$

- **objects** $\mathrm{Ob}(\mathbf{ZX}) = \mathbb{N}$
- **morphisms** equivalence classes of ZX-diagrams from n to m

Two ZX-diagrams from **ZX** are equivalent if they are related by the equivalence relation given in Definition 2.34. In a complete calculus this is equivalent there existing a sequence of application of rewrite rules that transforms one into the other.
[Wet20]

We make this into a monoidal category by using the same structure as we did for **ZX**.

Lemma 2.36: Category $\mathbf{ZX}/\sim$ is strict monoidal

We show that $\mathbf{ZX}/\sim$ is a strict monoidal category with following data:

1. a category we use $\mathbf{ZX}/\sim$
2. the bifunctor $\otimes : \mathbf{ZX}/\sim \times \mathbf{ZX}/\sim \to \mathbf{ZX}/\sim$, defined by
   - on objects: $a \otimes b = a + b$
   - on arrows: by writing them above one another, taking as interpretation the tensor product on complex vector spaces
3. the unit object given by $0 \in \mathbf{ZX}/\sim$
4. the associativity isomorphism is the identity, since addition in $\mathbb{N} = Ob(\mathbf{ZX}/\sim)$ is associative
5. since 0 is the neutral element of addition the left and right unit isomorphisms are the identity

The thing that remains to be checked is that this is well-defined. Since we only changed the arrows to be equivalence classes without changing the interpretation this is rather clear. $\square$

An important fact concerning these definitions and our example $Mat_{\mathbb{C}}$ from before is the following:

Lemma 2.37: Categories $\mathbf{ZX}/\sim$ and $Mat_{\mathbb{C}}$ are equivalent

We show the statement by giving a functor between the categories, which is faithful and full. Let $F : \mathbf{ZX}/\sim \to Mat_{\mathbb{C}}$ such that:

- Since $Ob(\mathbf{ZX}/\sim) = \mathbb{N} = Ob(Mat_{\mathbb{C}})$ on objects we take the identity on $\mathbb{N}$
- F maps a ZX-diagram f to its interpretation $[f] \in Mat_{\mathbb{C}}$, which is a morphism in $Mat_{\mathbb{C}}$

Proof: First we show that F is faithful. Let f, g be morphisms in $\mathbf{ZX}/\sim$, with $F(f) = F(g)$. So we have $[f] = F(f) = F(g) = [g]$, so the interpretations are equal. Then by definition of $\sim$ f and g are in the same equivalence class, so they are equal in $\mathbf{ZX}/\sim$. Therefore F is faithful.
Second we show that F is full. Let $[f]$ be an arbitrary morphism in $Mat_{\mathbb{C}}$. Since ZX-calculus is complete, the operator f, is the matrix representation for, can be represented by a ZX-diagram, which we call f. In $\mathbf{ZX}/\sim$, f is representative of an equivalence class, which F maps to $[f]$. So we constructed a preimage of $[f]$, which was arbitrary, so F is full. $\square$

Note that the functor used to show equivalence of the categories respects the monoidal products just as one would suspect.

Defining how the sum notation introduced by Stollenwerk et al. fits into this formal context is a main goal of this thesis and as such is discussed in Chapter 4.

2.5. Quantum Approximate Optimization Algorithm (QAOA)

QAOA is a variational quantum algorithm which has been developed to solve combinatorial problems [FGG14]. It is our primary application, but not directly relevant for our constructions so we will shortly describe the premise of the algorithm and refer the reader to [FGG14] for a more complete introduction.

A combinatorial problem is typically described by an objective function C and some constraints which may not be violated by a valid solution [FGG14]. Out of the valid solutions, we are searching for the one with the highest objective value $C(x)$ [Cer+21]. In QAOA we try to find values for parameters β and φ such that $|\beta, \varphi\rangle := U(\beta, \varphi) |+\rangle^{\otimes n}$ encodes a valid solution of maximal cost [FGG14]. This is done by mapping some input state to a lowest energy state of the so called *problem Hamiltonian* H_P [Cer+21]. Mapping is achieved by repeatedly applying the *problem unitary* $e^{i\varphi_l H_P}$ and the *mixing unitary* $e^{i\beta_l H_M}$ [Cer+21]. This gives total unitary $U(\varphi, \beta) = \prod_{l=1}^{p} e^{i\beta_l H_M} e^{i\varphi_l H_P}$ [Cer+21]. A classical optimizer optimizes β and φ such that applying the circuit and measuring yields a (close to) optimal solution [Cer+21].

A prominent question is what performance we can expect from this algorithm [SH22]. To answer this question, it can be helpful to compute an analytical form for the expectation value of some of the corresponding circuits [SH22]. In this work we only consider cases with one iteration, so $p = 1$. One example of a combinatorial problem solvable with QAOA is Max Cut [SH22]. We will consider circuits stemming from MaxCut problems in this work.

Definition 2.38: Maximum Cut problem (MaxCut)

Instance: Undirected Graph $G = (V, E)$ Problem: Find a cut in G with maximal number of edges.
A cut is an edge set which is generated by $\emptyset \neq X \subsetneq V$ as all edges, that have one end in X and one end in $V\ X$. [Kor+11]

This problem can be solved using integer linear programming using the following formulation:

Definition 2.39: Integer linear programm for MaxCut

Let $G = (V, E)$ be an instance of the MaxCut problem, with $n = |V|$. Optimal solutions are encoded as the optima of this linear program:

$$\max \frac{1}{2} \sum_{1 \le i < j \le n} c_{ij}(1 - y_i y_j)$$

$$s.t.\quad y_i \in \{-1, 1\} \quad \forall 1 \le i \le n$$

[Kor+11]

So we want to use QAOA to solve for optima of this integer linear program. To do so we define our cost function expectation value as follows

$$\langle C \rangle = \frac{|E|}{2} - \frac{1}{2} \sum_{u,v \in E} \langle Z_u Z_v \rangle$$

[SH22].

The involved diagrams for expectation value always have a similar structure in ZX-diagram form [SH22]. Each vertex is represented as one qubit [SH22]. We want to compute an assignment of the vertices to two disjoint sets that will generate a cut. The value of a Z measurement is either -1 or +1, so the measured value on qubit i determines set membership for vertex i.

For technical reasons that are out of scope, we can consider the contribution of each edge separately [SH22].

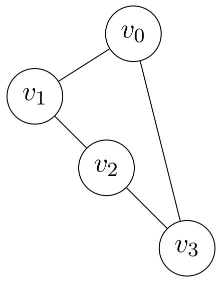


Figure 2.8.: Example Graph

The structure of the diagrams for expectation value for one contribution is easiest explained when considering an example, so observe the graph in Figure 2.8 and one corresponding diagram in Figure 2.9. The diagram represents the contribution of the edge from v_2 to v_3 to the expectation value of the QAOA state $\langle Z_{v_2} Z_{v_3} \rangle$. We see that first all edges of the graph are added to the diagram using so called *phase gadgets* [SH22]. Further, we add single X-spiders to each qubit to represent the mixing Hamiltonian [SH22]. Then two Z-spiders for v_2 and v_3 are appended, these represent the edge itself. Afterwards, we again add the X-spiders for the mixing Hamiltonian and all edges using *phase gadgets*, but this time with negated phases, representing the adjoint unitaries. For further details see [FGG14; SH22; Cer+21].

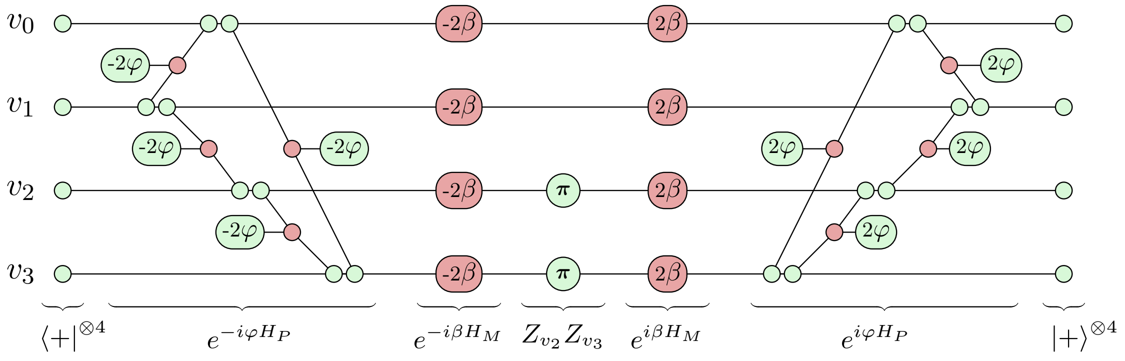


Figure 2.9.: Diagram for the QAOA expectation value $\langle Z_{v_2} Z_{v_3} \rangle$

Analytical expressions of these expectation value diagrams are helpful in analysing the performance of QAOA MaxCut [BK21].

3. Related Work

Linear combinations in diagrammatic reasoning for quantum applications have only recently been studied. Therefore, there is only some related work which can be clustered into three main topics. The first is the category theoretic approach dealing with monoidal categories without considering an interpretation. The second are works also introducing sums into ZX-calculus using theoretical approaches different from the one chosen here. The third and last one are works which try to achieve the same goal of circuit simplification as we do also using ZX-calculus but with other methods. We will shortly present them in the following and also note features which are similar to the approach chosen here or are distinctly different.

Monoidal categories with sums In [CDH20] Comfort et al. consider rig categories and diagrammatic representations of arrows in these categories. In short a rig category is a category with two monoidal structure which distribute over another up to unique isomorphisms [CDH20]. [CDH20] presents different approaches from literature to visualize these diagrams in the plane and in three dimensional space. They also develop their own representation in three dimensions called sheet diagrams [CDH20]. The additional structure is encoded by the diagram not living on the euclidean plane [CDH20]. Sheet diagrams live on topological manifolds with boundaries in $\mathbb{R}^3$ [CDH20]. An example taken from [CDH20] can be see in Figure 3.1. In contrast to the categories considered in this work, rig categories not only admit combining *compatible* morphisms using the monoidal structures. This makes the representation not directly feasible to be used in the application considered here, since arbitrary restrictions would have to be introduced. Also for our applications reasoning about the diagrams with sums by hand on paper is feasible and an expected use case. This also makes the three dimensional representation less suitable for solving the problems considered in this thesis.

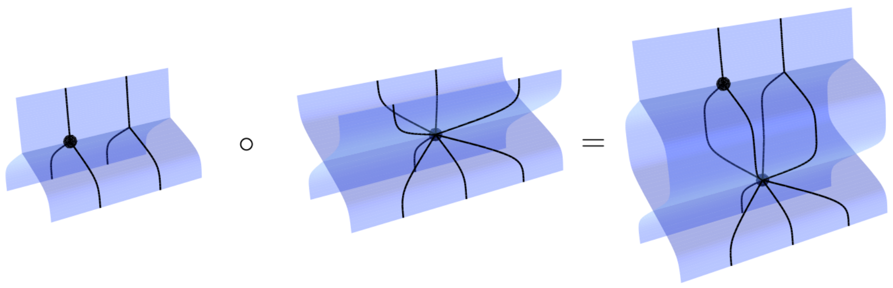


Figure 3.1.: Example of sheet diagram composition; taken from [CDH20]

Sums in ZX Several works published recently also introduce notions of sums in ZX-calculus [JPV22; WY22; SWY22].

Jeandel et al. proposed a notion of addition and differentiation of ZX-diagrams in [JPV18]. The primary use case they describe is akin to the one described here, as they also consider variational algorithms, such as QAOA [JPV22]. They also try to solve the problem of making ZX-calculus useable for applications with parameterized quantum circuits [JPV22]. Their construction relies on controlled diagrams[1], which are build from controlled states [JPV22]. The construction has the upside of being entirely contained in *vanilla* ZX-calculus which allows for manipulating the diagrams using the rewrite rules of normal ZX-calculus [JPV22]. One downside of this construction using controlled diagrams is that the resulting diagrams are fairly large, already for small input diagrams [JPV22]. Our approach does not have this downside, but our sums are not part of *vanilla* ZX-calculus, so we have to define new rewrite rules to manipulate the diagrams.

In [SWY22] Shaikh et al. introduced a calculus, called ZXW calculus, analogously to ZX-calculus. This calculus is similar to ZX, but uses a different basis [SWY22]. Namely they use the basis of GHZ and W states, but also define their diagrams as all diagrams freely generated over a set of generator [SWY22]. The GHZ and W states are different to the Z and X states in the regard that Z and X are basis elements of the same basis, while GHZ and W are not connected in this way [SWY22]. In this calculus they first show how to use controlled gates and controlled states to build controlled diagrams [SWY22]. These definitions are then used to define formal sums of diagrams of a specific type [SWY22]. In contrast to the work presented here it does not allow for sums of arbitrary diagrams. They also do not explicitly allow for linear combinations. The type of diagram they consider for summation represent Hamiltonians [SWY22]. They also define how to calculate e^{At} for an arbitrary matrix A as a diagram [SWY22]. Both formal sums and exponentiation are useful for improving Hamiltonian simulation [SWY22].

Another approach is presented in [WY22] by Wang et al. which also uses W spiders to represent sums. Their focus is on enabling differentiation of ZX-diagrams and they also introduce notions for integrating certain types of diagrams. In order to define the product rule for differentiation they use W-spiders to avoid having a sum of diagrams in the result. Sums of diagrams are not the focus of their work. Instead they directly replace the sums arising due to the product rule by a diagram where the sum structure is represented using W-spiders. Their solution does therefore not solve the problem posed in this work, where sums of diagrams are partially being introduced as a tool for manual rewriting. [WY22]

Circuit Simplification using ZX-calculus There exists some previous work on simplifying circuits using ZX-calculus [Dun+20; BPV21]. One example is [Dun+20], the authors introduce a graph theoretic simplification algorithm for quantum circuits. To this end they convert the circuit to a diagram and simplify this while preserving a property called *generalized flow* [Dun+20]. They also present a procedure for extracting circuits from ZX-diagrams which have the *generalized flow* property [Dun+20]. Their work is focused on simplifying the diagram and then getting circuits back out [Dun+20]. This is a stark contrast to the work done here, since our goal is to simplify the diagram entirely to

[1]Which are diagrams with an extra input, where a $|1\rangle$ gives a diagram with some value and a $|0\rangle$ gives a neutral diagram.

an analytical expression. Second we consider symbolic phases, which Duncan et al. do not explicitly do. Also both new rewrite-rules they introduce, *local complementation* and *pivoting*, depend on the existence of Clifford spiders[2] [Dun+20]. Since we consider symbolic phases these rewrite rules may not be as effective as in their cases.

Another work based on [Dun+20] is [BPV21] by Borgna et al. They extend Duncan's work by considering hybrid circuits [BPV21]. Those are circuits which also include classical components and measurements [BPV21]. To this end they use $ZX_{\underline{\perp}}$, an extension of ZX-calculus which allows easier description of interactions with the environment [BPV21]. They define a conversion from hybrid circuits to $ZX_{\underline{\perp}}$-diagrams, which also have the property of *generalized flow* [BPV21]. As well as [Dun+20] their goal is to simplify in (a dialect of) ZX-calculus and then extract simplified circuits back out [BPV21]. Since their approach builds on [Dun+20] we differentiate our work in a similar manner, since we consider symbolic phases and simplification to an analytical expression.

[2]Clifford spiders are spiders where the phase is a multiple of $\frac{\pi}{2}$

4. Linear Combinations of ZX-diagrams

In this chapter we will first cover in depth the motivation for introducing linear combinations into ZX-calculus. For this motivation we will derive desired properties our construction should aim to fulfil. Then we will introduce the actual formal construction building on the theory presented in Chapter 2.

4.1. Motivation

As we have seen in the introduction of ZX-calculus it is proven to be complete. Meaning that any two equivalent diagrams can be transformed into another using the rules presented above. This directly leads to the question what allowing linear combinations brings to the table. To illustrate the usefulness of the extension consider the example in Figure 4.1. Since it has no input and no output wires it represents a complex phase. This phase is dependent on the parameters φ and β. An interesting question for applications in variational algorithms is now to compute a term depending on these parameters representing the complex number the diagram represents [SH22].

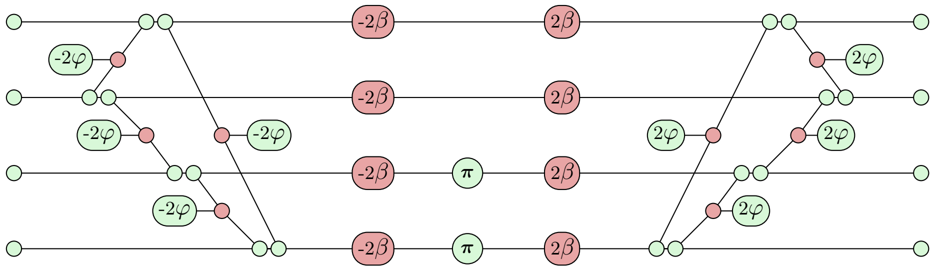


Figure 4.1.: Simple QAOA circuit as ZX-diagram [SH22]

As we have noticed in Chapter 3 state of the art simplification algorithms for ZX-diagrams all have at least one of two limitations. They can either not simplify with symbolic spiders or the they cannot track the global phase of a diagram whilst simplifying. Note that due to ZX being complete this is not a limitation of the calculus itself, but of the simplification algorithms. Regardless it can be beneficial to introduce further tools for rewriting diagrams which may simplify the procedure. Exactly that has been done in [SH22] where the authors simplified said example by hand using the notion of linear combinations of ZX-diagrams. As demonstrated there it can be a useful tool for simplifying ZX-diagrams whilst tracking the global phase. In this chapter we try to formalize this notion of linear combinations to later make use of it when proposing a general simplification algorithm for diagrams of similar shape to the one in Figure 4.1.

4.2. Formal Construction

We split the problem of formally constructing linear combinations of ZX-diagrams in two parts. First we will describe how we can realise scalar multiplication. The second part will then be defining sums of diagrams. In combination these two parts allow us to represent arbitrary linear combinations.

4.2.1. Scalar multiplication

As a first building block we clear up how to realise scalar multiplication. To do so we first observe that in our particular category **ZX** the tensor product is the kronecker product. With this we see the following lemma:

Lemma 4.1: Scalar multiplication

Let f be a ZX-diagram with interpretation $(a_{i,j})_{i<n,j<m}$ and let x be a complex scalar. By universality of ZX the 1×1-matrix $[x]$ can be represented as a ZX-diagram, say g. Scalar multiplication of f with x can be realised as $f \otimes g$.

Proof The interpretation of $f \otimes g$ is:

$$(a_{n,m})_{n,m} \otimes [x] = (x \cdot a_{n,m})_{n,m} = (a_{n,m})_{n,m} \cdot x$$

which is the interpretation of $f \cdot x$. $\square$

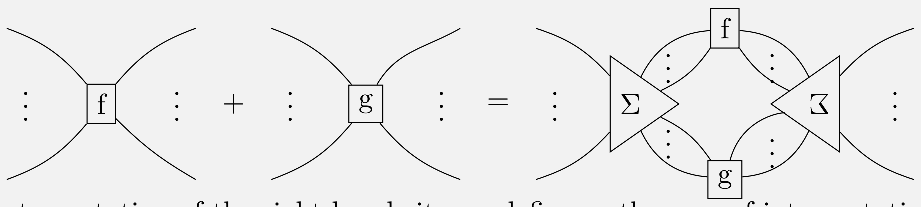
Note: Since everything is defined over $\mathbb{C}$ multiplication is commutative and scalar multiplication from the right follows directly.

4.2.2. Sums of diagrams

The second building block we need is addition of two ZX-diagrams. To do so we introduce a superposition rule $+$ on ${}^{\mathbf{ZX}}/_{\sim}$. So for this subsection let f and g be representatives for arbitrary ZX-diagrams each with n inputs and m outputs.

Definition 4.2: Sums of ZX-diagrams

We define addition of diagram equivalence classes by taking representatives f and g as

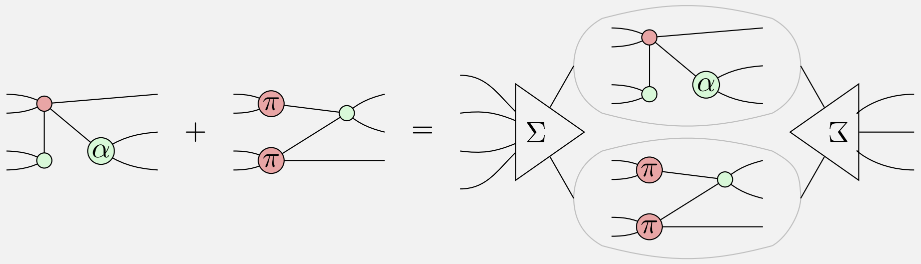


The interpretation of the right hand site we define as the sum of interpretations so $[f] + [g]$.

Note: We sometimes use a slightly adapted notation, where each term of the sum is contained in a grey bubble. The bubbles are then only connected to the Σ spider with one wire, since the number of connections is clear from context. The following example illustrates the utility of this more concise notation.

Example 4.3: Sum notation

We present an example of our sum notation, just to present its usefulness without considering the diagrams interpretation.

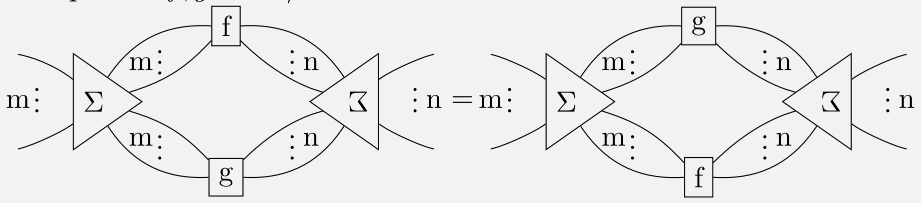


Definition 4.2 is well-defined since we are working in $\mathbf{ZX}/\sim$: Consider taking different representatives f' and g', with $[f] = [f']$ and $[g] = [g']$. Then $[f] + [g] = [f'] + [g] = [f'] + [g']$, so the interpretations of the sums are also only different representatives of the same equivalence class.

We will now see some important properties of this definition.

Lemma 4.4: Sums are commutative

Given morphisms f, g in $\mathbf{ZX}/\sim$ we have:



Proof We only need to check that the interpretation stays equivalent. The interpretation for the left hand side is $[f] + [g]$ and for the right hand side $[g] + [f]$. By commutativity of matrix addition these are equal. □

Lemma 4.5: Sums are associative

Given morphisms f, g in $\mathbf{ZX}/\sim$ we have:

$$(f + g) + h = f + (g + h)$$

Proof We only need to check that the interpretation stays equivalent. The interpretation for the left hand side is $([f] + [g]) + [h]$ and for the right hand side $[f] + ([g] + [h])$. By associativity of matrix addition these are equal. □

With these properties we can now go on to one of our main results:

Theorem 4.6: Sums are a superposition rule

The addition from definition 4.2 is a superposition rule, with $u_{A,B}$ a diagram with the zero matrix as interpretation.

Proof First we observe that we can always construct $u_{A,B}$ by taking an arbitrary diagram from A to B and multiply via scalar multiplication as defined above with 0. We now check the properties independently.

- $f + g = g + f$,
  - ▶ is the statement of Lemma 4.4
- $(f + g) + h = f + (g + h)$
  - ▶ is the statement of Lemma 4.5
- for all $A, B \in Ob(\mathbf{ZX}/\sim)$ there is a $A \xrightarrow{u_{A,B}} B$, such that for all $A \xrightarrow{f} B$: $f + u_{A,B} = f$
  - ▶ interpretation of $u_{A,B}$ is the zero matrix by construction which is the neutral element of matrix addition
- $(g + h) \circ f = (g \circ f) + (h \circ f)$
  - ▶ follows from distributivity of matrix addition and multiplication
- $f \circ (g + h) = (f \circ g) + (f \circ h)$
  - ▶ follows from distributivity of matrix addition and multiplication
- for all $f : A \to B$ and $C, D \in Ob(C)$: $u_{C,B} = f \circ u_{C,A}$ and $u_{A,D} = u_{B,D} \circ f$
  - ▶ since $u_{A,B}$ and $u_{C,D}$ both have the zero matrix as interpreation by construction they are equivalent, so represent the same morphism in $\mathbf{ZX}/\sim$

All properties are fulfilled so this addition is a superposition rule in $\mathbf{ZX}/\sim$. □

Corollary 4.7: $Mat_{\mathbb{C}}$ has a superposition rule

$Mat_{\mathbb{C}}$ also has a superposition rule given by matrix addition.
We have seen, that we can define sums of diagrams in such a way that it's interpretation coincides with the definition of matrix addition. Together with the fact, that we can scale diagrams arbitrarily this gives us arbitrary linear combinations of ZX-diagrams.

To formally make use of the above construction we first proof some properties the category fulfils. This will allow us to later use the category in a powerful way.

Lemma 4.8: $\mathbf{ZX}/\sim$ admits arbitrary duals

Each $A \in Ob(\mathbf{ZX}/\sim)$ is its own left and right dual. In particular $\mathbf{ZX}/\sim$ admits arbitrary duals.

Proof This follows directly from Example 2.19, by using the equivalence defined in Lemma 2.37. □

Theorem 4.9: $^{\mathbf{ZX}}/_{\sim}$ admits arbitrary biproducts

In $^{\mathbf{ZX}}/_{\sim}$ for each two objects $A_1, A_2 \in Ob(^{\mathbf{ZX}}/_{\sim}) = \mathbb{N}$ exists an biproduct $A_1 \otimes A_2 = A_1 + A_2$. Where addition is to be understood as the standard addition in $\mathbb{N}$. The injection and projection morphisms are constructed as follows: Identify A_1 and A_2 with $\mathbb{C}^{A_1}$ and $\mathbb{C}^{A_2}$ in $Mat_{\mathbb{C}}$. And take the matrices, which represent the morphisms constructed in Lemma 2.37. Then the morphisms in $^{\mathbf{ZX}}/_{\sim}$ are by universality of ZX given by arbitrary representatives of the equivalence classes related to the matrices.

Proof The above construction is well-defined since the functor from $^{\mathbf{ZX}}/_{\sim}$ to $Mat_{\mathbb{C}}$ is an equivalence by Lemma 2.37. It remains to be shown, that the construction above actually is a biproduct. All the properties follow directly as in the proof of Example 2.17. □

With these two properties we see the following theorem:

Theorem 4.10: Addition in $^{\mathbf{ZX}}/_{\sim}$ distributes over kronecker product

Sums of equivalence classes of ZX-diagrams in $^{\mathbf{ZX}}/_{\sim}$ as defined in Definition 4.2 distribute over kronecker product. So for f and g representatives of arbitrary compatible equivalence classes of morphisms we have:

$$(f+g) \otimes h = (f \otimes h) + (g \otimes h)$$

$$h \otimes (f+g) = (h \otimes f) + (h \otimes g)$$

Proof: By Theorem 4.9 we have arbitrary biproducts. By Lemma 4.8 $^{\mathbf{ZX}}/_{\sim}$ also admits arbitrary duals. Then by Lemma 3.22 in [HV19] we have that the superposition rule distributes over the monoidal action of the category. □

A diagrammatic view of this distributivity law can be found in Figure 4.2.

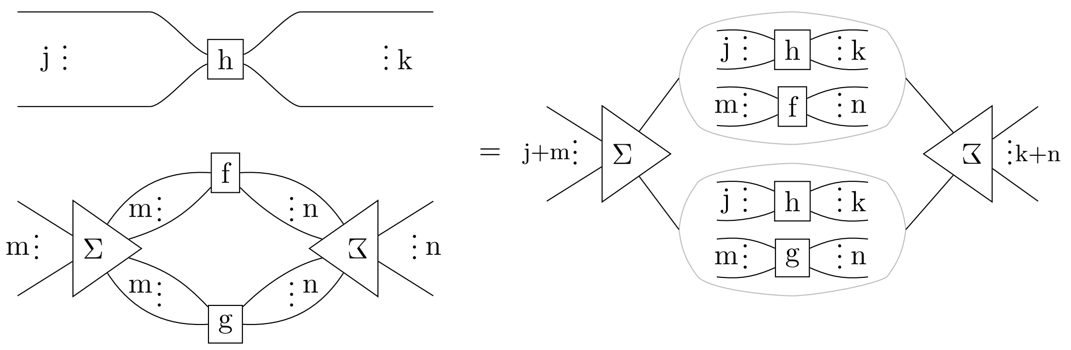


Figure 4.2.: Diagrammatic statement of the second statement of Theorem 4.10

4.3. Proofs of equivalence

After observing the above properties we will first see that this construction has exactly the properties introduced in [SH22]. Second some properties which are true for *vanilla* ZX-calculus are verified to still be true for our extended calculus.

4.3.1. Rules from Stollenwerk et al.

In Section 2.4.5 rules to manipulate sums of diagrams postulated by Stollenwerk et al. were presented. We need to check that these rules also work in our extended calculus. In [SH22] the rules are formulated with explicit scalars for each term of a sum. As we represent these scalars in the diagrams we drop them from our description.

Consider first the diagram pull rule: It states that the kronecker product of diagrams distributes over sums of diagrams, with respect to the matrices the diagrams represent. More formally:

Theorem 4.11: Diagram pull rule

For A, B morphisms in $\mathbf{ZX}/\sim$ with the same domain and co-domain and Id_p the identity morphism of $p \in Ob(\mathbf{ZX}/\sim)$, we have:

$$[(Id_p \otimes A) + (Id_p \otimes B)] = [Id_p \otimes (A + B)]$$

Proof Since the kronecker product is the monoidal operation of the category $\mathbf{ZX}/\sim$ and Id_p is in particular a morphism in $\mathbf{ZX}/\sim$ this identity was shown as Theorem 4.10. □

So we see that the diagram pull rule is true in our extended calculus. Furthermore, we notice that the formulation in Theorem 4.10 is strictly stronger than the one from Stollenwerk et al.

Now consider the product rule: It states that composition distributes over addition. So more formally we get:

Theorem 4.12: Product rule

For morphisms $A : m \to n$ $B : m \to n$ $C : n \to l$ we have:

$$(A + B) \circ C = (A \circ C) + (B \circ C)$$

$$A \circ (B + C) = (A \circ B) + (A \circ C)$$

Proof Distributivity is an axiom of addition being a superposition rule and was therefore checked in Theorem 4.6. □
For a diagrammatic view of the first rule see Figure 4.3.

Note that in the original formulation by Stollenwerk et al. also included a morphism $D : n \to l$. We now show that that formulation is equivalent, to our more concise variation.

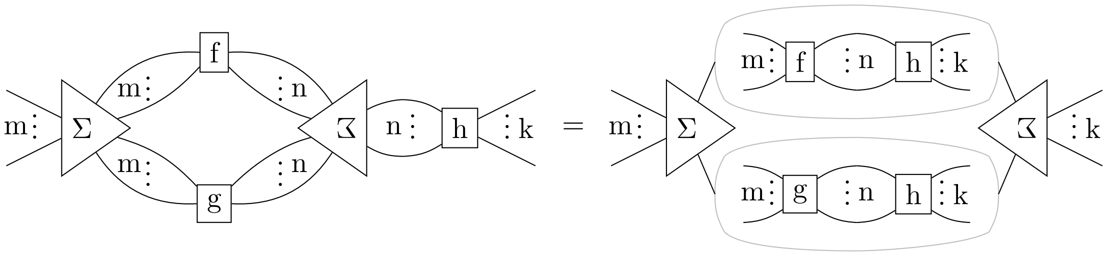


Figure 4.3.: Diagrammatic statement of the product rule Theorem 4.12

Lemma 4.13: Product rule and distributivity over composition are equivalent

Let f, g, h, k be morphisms in some strict monoidal category, where $+$ is a superposition rule and $\circ$ denotes composition. Then the Product (composition) rule given by Stollenwerk et al. as in Definition 2.31 holds if and only if addition distributes over composition, as proven in Theorem 4.12.

Proof We show the directions separately.

$\Leftarrow$ Suppose distribution of composition over addition is given. Then consider the following equations, where we apply distributivity twice and uses that composition of morphisms gives again a morphism.

$$\begin{aligned}&(f+g)\circ(h+k)\\=&(f\circ(h+k))+(g\circ(h+k))\\=&(f\circ h)+(f\circ k)+(g\circ h)+(g\circ k)\end{aligned}$$

$\Rightarrow$ Suppose the product pull rule holds. We want to see that the distributivity law holds in both directions. The neutral element of addition is any morphism that has as interpretation the zero map. By universality of ZX-calculus we can always choose 0 as such in a compatible manner. For any compatible morphisms f, g, h, we see by first inserting a zero, then using the product rule and then by using the neutral element of addition (and its properties in multiplication) again:

$$\begin{aligned}&(f+g)\circ(h)\\=&(f+g)\circ(h+0)\\=&(f\circ h)+(g\circ h)+(f\circ 0)+(g\circ 0)\\=&(f\circ h)+(g\circ h)+(0)+(0)\\=&(f\circ k)+(g\circ k)\end{aligned}$$

Similarly for the other direction we get:

$$\begin{aligned}
&(f) \circ (h + k) \\
=&(f + 0) \circ (h + k) \\
=&(f \circ h) + (0 \circ h) + (f \circ k) + (0 \circ k) \\
=&(f \circ h) + (0) + (f \circ k) + (0) \\
=&(f \circ k) + (f \circ k)
\end{aligned}$$

This gives both distribution laws. □

These two rules are the ones stated by Stollenwerk et al. and our extended calculus also fulfils them. So our extension is compatible to all further considerations done in [SH22]. Furthermore, our description is more concise and in some regards even stronger than the original formulation.

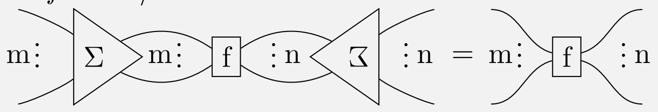
4.3.2. Further useful rules

Some rules we deem useful and reasonable are not explicitly stated by Stollenwerk et al. Therefore, we extend the rule set by some rules we find convenient; some of which we introduced previously.

Recall Lemma 4.4, we have shown that sums of diagrams are commutative. A fact not explicitly stated in [SH22]. This rule is practical since it continues the slogan *Only connectivity matters*, that is prominent with *vanilla* ZX-calculus. Since this rule shows that the order of the bubbles in a sum does not matter.

Lemma 4.14: Sums with only one term can be dropped

For any morphism f in $\mathbf{ZX}/\sim$ we have:



Proof It again suffices to check the equality of the interpretations. Since for matrices $\sum_{i=1}^{1}[f] = [f]$ this also holds for diagrams. □

Another interesting property is that for any diagram f we can find an diagram representing the additive inverse of f.

Lemma 4.15: Additive inverses exist

For any morphism f in $\mathbf{ZX}/\sim$ we can find a morphism g such that their sum can be dropped.

Proof Consider the interpretation $[f]$ and the matrix $-1 \cdot [f]$. For this second matrix we can find a diagram representing it by universality. By definition of matrix addition we see that $[f] - [f] = 0$. □

4.3.3. Properties of the extended calculus

Three properties, which we also checked for *vanilla* ZX-calculus are interesting to have for a calculus in general.

Universality Since ZX-calculus is universal, our extension is as well. For any linear map A the matrix representation $[A]$ can be implemented as a ZX-diagram. So in order for our extended calculus to be universal, the extension is not even needed. Therefore our extended calculus is also universal.

Soundness ZX-calculus with the rewrite rules defined in Figure 2.6 is sound. So those rewrite rules do not change the interpretation of the diagram. Since they can only be used in terms of the sum and not over the terms of a sum, they stay sound in the extended calculus. What we need to check is that the rewrite rules we introduce for linear combinations are also sound. For all rules we introduce we have checked this directly after introducing them in the previous section. These rules do hold on a category theoretic level and therefore do not change the interpretation of the diagram. So the extended calculus is still sound.

Completeness The last property left to consider is completeness. As it was with *vanilla* ZX-calculus it turns out to be the hardest to consider. First we notice that the rules from Stollenwerk et al. on their own are certainly not enough since they are missing commutativity of sums. Furthermore, we notice that our current rule set is still not complete. Consider the following rather trivial example: Let $\alpha, \beta \in \mathbb{C}$ and consider diagrams A and B which represent the 1×1 matrices $[\alpha]$ and $[\beta]$. Their sum is a diagram representing $[\alpha + \beta]$, which exists by universality of ZX-calculus. So if our rule set was complete we would have to be able to transform the diagram representing the sum of A and B to this diagram, which has the same interpretation. But with the current rule set we have no way of achieving this. Therefore, we leave introducing the needed rewrite rules and a concrete proof of completeness as future work.

5. Implementation

This chapter will detail our how we implement linear combinations and outline the developed algorithm for simplifying symbolic ZX-diagrams to an analytical expression. These are our solutions to the second and third goal of this thesis. We first outline the steps taken in order to implement linear combinations in software. For this purpose we will first present some frequently used frameworks that work with ZX-calculus and present reasoning as to why one of them was chosen above the others to extend. Then we discuss details of the introduction of linear combinations into the implementation; demonstrating our solution to the second goal. Lastly we will present an algorithm designed to simplify ZX-diagrams representing circuits from QAOA problems using said implementation. This is proposed as a solution to the third goal.

5.1. Framework selection

Three frameworks were considered as candidates for the implementation of the rewriting algorithm: PyZX [KW20], DisCoPy [FTC20] and Quantomatic [pro18]. We now shortly introduce each one and then present our reasoning for the choice we make.

PyZX The framework by Kissinger et al. is written in python built exclusively for ZX-calculus [KW20]. It can represent arbitrary ZX-diagrams and has several rewrite rules and simplification strategies built in [KW20].

At the start of our work it did not have support for symbolic computations. Said support was added later by integrating the python library *sympy* [Yeu22]. At the time of writing not all features work correctly with symbols [Muu22]. Some of these issues are solved during the implementation [Muu22]. Others are not resolved, this includes the simplification strategies.

It has a simplification strategy specifically tailored towards reducing diagrams where the global scalar is to be tracked called *reduce_scalar*.

DisCoPy DisCoPy is a framework developed for the more general setting, of reasoning about diagrams from arbitrary monoidal categories [FTC20]. It has custom support for ZX-diagrams built in, including conversion from and to PyZX diagrams. However rewriting ZX-diagrams directly according to rewrite rules is not supported. Therefore it also does not provided simplification strategies.

Since DisCoPy is built for working in arbitrary monoidal categories the boxes can contain any data. This includes native support for symbolic phases in ZX-diagrams.

DisCoPy also has support for formal sums of diagrams. These formal sums always distribute outward completely meaning that the sum structure is always the most outward operation.

Quantomatic Quantomatic has mostly been developed as a diagramatic proof assistant for ZX-calculus [Kis12]. It focuses on supporting a user in rewriting diagrams manually rather than rewriting for the user. As such it has built in rewrite rules for ZX-diagrams but includes only a limited number of automated rewriting strategies. Additionally Quantomatic does not have support for symbolic phases. Project activity levels for Quantomatic suggest that it is not being actively developed at the time of writing [Git22].

Since we need support for symbolic computation we choose DisCoPy as a basis for our work. Also it allows for flexibility since we can convert DisCoPy diagrams without sums to PyZX diagrams. Quantomatic was not considered further since it cannot handle symbolic phases. Furthermore the chances of our extension being accepted into the framework where thought to be higher with an actively maintained project. Therefore we opted to combine the strengths of DisCoPy and PyZX. So we decided to implement the sum structure in DisCoPy, but convert the terms of the sums to PyZX for simplification purposes not involving sum structures.

5.2. Implementation details

In order to understand the implementation of the sum structure we first discuss a simplified version of the data structure DisCoPy uses to represent diagrams. Afterwards we present the construction for local sums in this structure.

DisCoPy uses an object oriented approach for saving diagrams. Each object of a category is represented as an object of the class Ob. The arrows are also represented as a class, which in our simplified version can just be regarded as saving an arbitrary morphism. A diagram is a sorted list of *layers*, where adjacency in the list represents composition. Each layer contains a sorted list of arrows. Here adjacency represents the monoidal product. An visual representation, depicting an example can be seen in Figure 5.1. DisCoPy does support representing formal sums of diagrams. When using this construction sums are always and automatically distributed outward. Sums are represented as objects containing a list of terms.

In order to implement sums into DisCoPy we extended their notion of sums by a new class called LocalSum. This class also contains a list of terms but can be used like any other arrow and will not distribute outward. LocalSum and DisCoPy's Sum class have a common super class called AbstractSum. This abstraction makes it possible for Sums and LocalSums to interact as one would expect and reduces duplicate code. Since the LocalSum instances are just special cases of Arrows they fit in nicely with the diagram structure DisCoPy uses. The quantum specific diagrams DisCoPy supports are special cases of more general categories it can represent. As such it was sensible to implement the LocalSums not only for the quantum categories, but also for all other categories DisCoPy can represent. This allows for more flexibility in representing enrichment over monoids in any use case for DisCoPy.

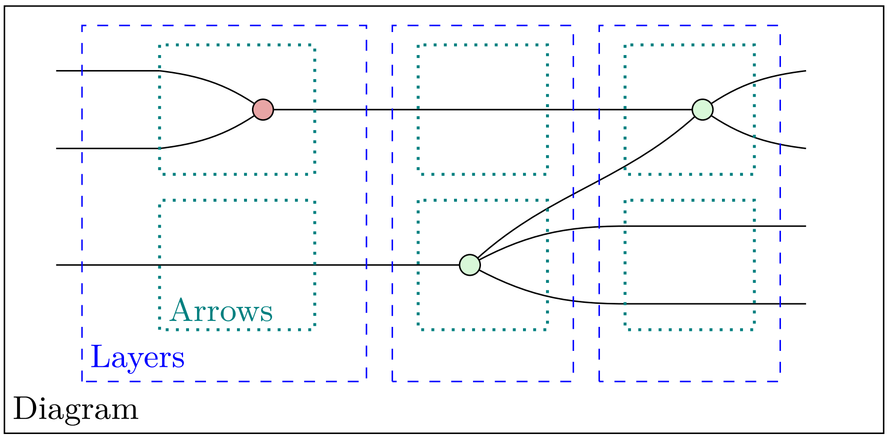


Figure 5.1.: Diagram with parts annotated with the classes DisCoPy uses to represent them

Distribution over composition was implemented analogously to the existing Sum class. In monoidal diagrams the input and output of a sum are possible not only connected *within* the sum, but maybe also on the outside. One has to take specific care to handle these cases. These cases were not considered in DisCoPy previously since they do not arise when we distribute the sum to the input and output layer. Reversing distribution over composition was not implemented prior but has been implemented for this work. Distribution over monoidal product was also not implemented prior to this work. We implement it for the case where the LocalSum and the box to be pulled in are on the same layer. If this is not given but mathematically distribution is possible it is necessary to first shift boxes between layers. This shifting is also implemented.

In summary we have implemented sums in DisCoPy, which by the construction in Chapter 4 directly gives us linear combinations. We have also provided implementations for several important rewrite rules, in particular the ones given in [SH22].

5.3. Simplification Algorithm

We now turn to the third goal of this thesis; implementing a simplification algorithm. In the following we present a prototype for a algorithm for simplifying ZX-diagrams for QAOA MaxCut problems using linear combinations. First we introduce the general idea of our simplification algorithm. Second we describe some subroutines we use frequently throughout the algorithm. We then present the algorithm in detail and highlight some properties. Last we discuss limitations and possible solutions.

5.3.1. Main Idea

As we discussed in Section 2.5 operators for QAOA MaxCut typically contain two parameters φ and β. The first is used in the problem Hamiltonian and the latter in the mixing Hamiltonian. Typical circuits are therefore of the form described in Section 2.5.

So as a ZX-diagram we have β spiders in the *middle* of the diagram and φ spiders in phase gadgets on the *outside* of the diagram.

The algorithm is divided into two stages, each dealing with one of the symbols. The output of the first stage are several diagrams, which only contain one symbol. These diagrams are then run through the second stage separately. In the following we introduce the stages in order of execution; in text form illustrating the steps on an example. Presentations in pseudo-code can be found in Appendix A.1. Note that between all steps we run simple simplification procedures, until these trivial measures fail to simplify further. We omit mentioning these steps for clarity, but they are defined below.

Pull symbol to scalar stage

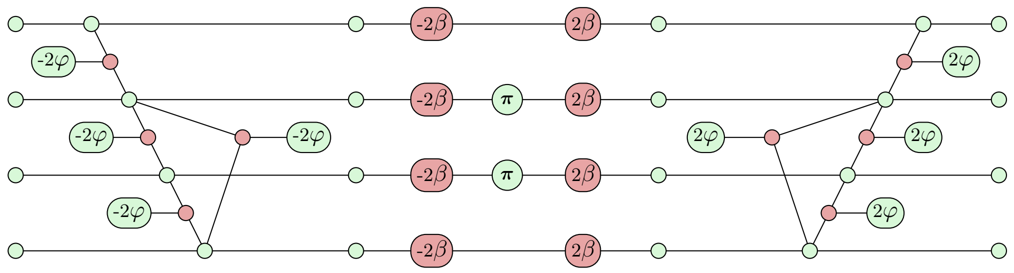


Figure 5.2.: Diagram to be simplified

In Figure 5.2 the initial diagram for our example can be seen. First we use spider fusion to simplify the diagram (Figure 5.3). Further, we pull the β spiders into scalars by introducing linear combinations.

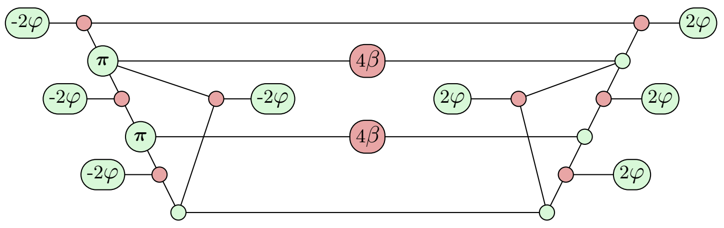


Figure 5.3.: Diagram after π-spiders commuted through left β-spiders

This gives us a diagram where the only spiders depending on β are isolated (Figure 5.4).

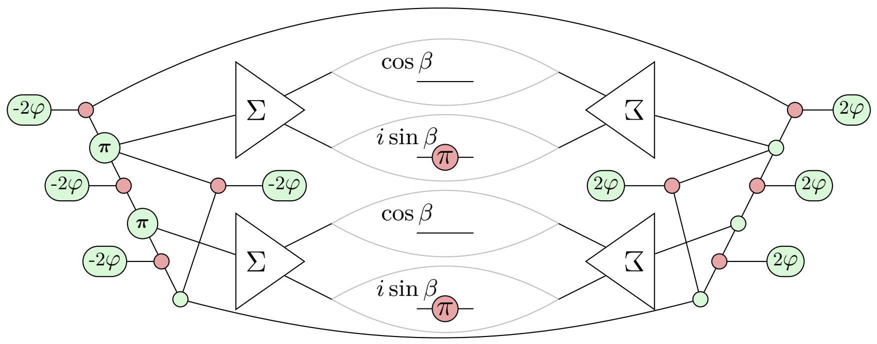


Figure 5.4.: Diagram with β-spiders pulled to scalar by introducing linear combinations

Now we uses distributivity over composition to pull everything into the sum terms. The sum is now the outmost part of the diagram. This gives multiple diagrams only depending on φ, which remain to be simplified. The second term of the sum is shown in Figure 5.5

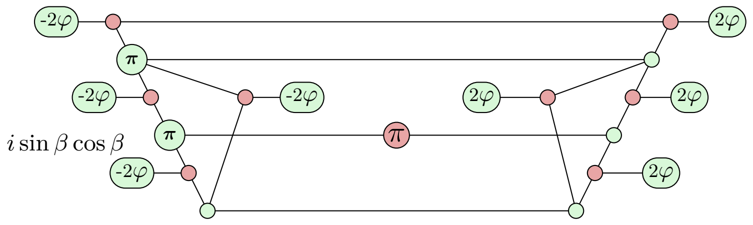


Figure 5.5.: Second term of diagram with β-spiders

Simplify terms with only one symbol stage

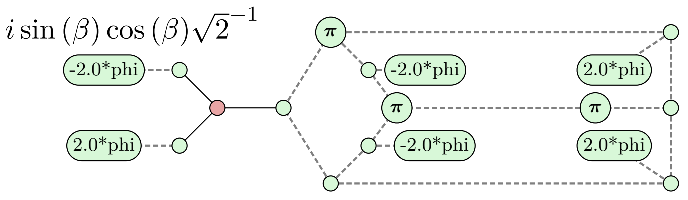
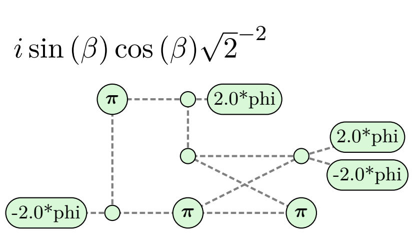


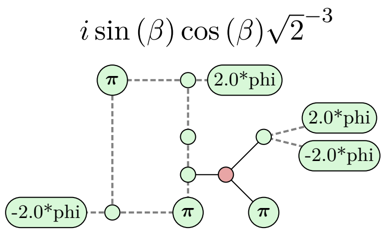
Figure 5.6.: Second term of the sum at the start of simplification

The dependency on φ in these diagrams is rather different in shape to the one β (Figure 5.6). Since the unitaries on the left and right are conjugates of one another we can always match up phase gadgets of different sign. So we iterate the following procedure until we have contracted the diagram to a scalar.

If we can we match up phase gadgets and simplified using the bi-algebra rule of ZX-calculus (Figure 5.7).



(a) Before application



(b) After application

Figure 5.7.: Bi-algebra rule application example in stage two

In any given iteration apply the bi-algebra rule does not produce a result we apply the state copy rule if possible. Again if none of the above resulted in change we try copying and commuting π-spiders through others.

The loop ends once none of these result in any change in the diagram. The remaining diagrams are then returned to the main algorithm. It then computes the scalar of each term separately and returns their sum.

Specific subroutines

Subroutines used within the simplification algorithm are described in the following. Used rules which are not described here are used from PyZX implementation. In a graph theoretic all these rewrite operations are algorithms replacing one subgraph with another. To do so the main task is to identify the subgraph to be replaced. This problem is called subgraph isomorphism problem and is NP-complete in general [Coo71]. So these subroutines all try to use properties of the desired matches to speed up the process.

Basic simplification routine The basic simplification routine is run between all steps of the simplification algorithm. It combines two rules PyZX provides; namely spider fusion and identity removal. Those two rules are run in a loop until none of them simplifies the diagram further.

Permute centre π-spiders This function searches for Z-spiders with a π phase which have an X-spider with a symbolic phase to the right. Rather simply this is done by iterating over all vertices and checking for the condition.

Pull symbolic phases to scalar Finding candidates for pulling phases into scalars is done in a similar fashion to above. We iterate over all vertices and check whether the phase contains the desired symbol. Once found we introduce a sum and pull the symbol to scalars.

Combine sums The third subroutine is used to combine sums. More concretely we use the distributing property of sums to perform the following operation:

$$(A + B) \otimes (C + D) = A \otimes C + A \otimes D + B \otimes C + B \otimes D$$

The operation is performed until no more candidates are found.

Bi-algebra replace We implement the bi-algebra replace rule only for a special case. The implementation finds four cycles of the form that bi-algebra may be applied and replaces it with two spiders which are connected. Finding these four cycles is done by first finding a candidate sets of three vertices which form a line but not a triangle. Then the neighbours of the outer two are compared and if they share a neighbour (which is not the one in the middle) we found a cycle. We then check that all spiders in the cycle have colours compatible with the bi-algebra rule. If so the cycle is replaced by two vertices and the neighbours are connected according to the rule.

π-commutation rule The π-commutation rule is implemented by finding all spiders that have a π phase and exactly two neighbours. This is done by exhaustion and all such spiders are considered as candidates. The algorithm arbitrarily picks one and applies the commutation rule to an arbitrary side.

5.3.2. Analysis

We first make some observations regarding the two stages of the algorithm. Stage one is deterministic and certainly terminates, if the input is of the shape we expect. On the other hand stage two has different properties. We make the observation that termination of this algorithm is not guaranteed. The π-commutation rule can introduce new spiders to the graph and so a cycle of rewrite rules could form.

#Vertices	#Instances	#Not completely simplified	Max. vertex count in result
3	6	0	0
4	33	7	5
5	170	48	10
6	1170	369	13

Table 5.1.: Overview of experimental results

Additionally, we analysed our algorithm experimentally. For all graphs with 3, 4, 5 or 6 vertices ran the algorithm once for each edge. Running for each edge is reasonable since we want to compute the contribution for each edge in the MaxCut problem. We observe that the algorithm terminates in reasonable time (seconds on a commodity laptop computer) for any of these instances. Furthermore, the algorithm always simplifies the diagrams up to a tree shape, an example can be seen in Figure 5.8. Unfortunately it does not produce a scalar as output in all cases. In 424 of the 1379 cases it outputs a tree shape. This tree shape has a maximal size of 13 nodes. A detailed overview can be found in Table 5.1.

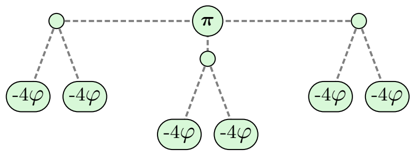


Figure 5.8.: Example of a tree residual the algorithm creates

5.3.3. Limitations

The algorithm has several limitations which we discuss in the following. We also introduce possible mitigations for the these shortcomings.

The first and most obvious shortcoming is that it only works for a very specific set of diagrams. It is only designed to handle circuits that arise from solving MaxCut using QAOA. Circuits arising from other combinatorial optimization problems might have similar shape but can also have completely different shapes. One may hope that the approach from our prototype is also true in more general settings, but currently we have no strong indication either way.

Our prototypes second limitation is that we cannot guarantee that the second stage terminates. Since the π-commutation rule may increase the number of vertices cycles in the rewrite procedure are conceivable. This limitation can be mitigated by including a stopping condition if the diagram reaches a state it had already reached in a previous iteration. Mitigating in this way has the drawback of having to track all previous states which is memory intensive. Since in the tested examples including all graphs with 3, 4, 5 or 6 vertices the algorithm terminated we did not include this mitigation.

A further limitation is the fact that the second stage not always simplifies the circuit completely. In 424 out of the 1373 tested cases we were left with a tree like diagram that could not be simplified further. The maximum node count of these trees was 13, the current mitigation is to compute the analytical expression of these trees via the matrix interpretation. This might result in problems with larger examples. A different conceivable solution would be to introduce further rewrite rules that allow for simplifying those trees.

6. Conclusion

In this thesis we investigate linear combinations of ZX-diagrams and how they can be applied to the analysis of variational quantum algorithms. In particular our goal was to formally define linear combinations of ZX-diagrams. We utilize the results in building a prototypical simplification algorithm for ZX-diagrams arising from NISQ-applications. More concretely, we consider diagrams which represent circuits from applying the Quantum Approximate Optimization Algorithm (QAOA) to the MaxCut problem. Achieving this goal is divided into three subproblems, which we consider separately.

Formal definition The first subproblem is to formally define what linear combinations are. In Chapter 4 we show that in $\mathbf{ZX}/\sim$ matrix addition as interpretation, defines a superposition rule. Furthermore, it is shown that this definition and the one given by Stollenwerk et al. coincide. Stronger and more concise rewrite-rules than were presented in literature are also proven to work in this construction. Namely distribution of addition over composition and over the monoidal product. We also show some additional rewrite-rules. These include commutativity and dropping of sums with only one term. Soundness and universality of the construction are proven. This fulfils the first goal of this thesis.

Implementation The second subproblem is to implement linear combinations in a library for ZX-diagrams. DisCoPy is chosen as a basis for the implementation. The previous notion of formal sums in DisCoPy is extended to represent local sums. Distribution of composition and monoidal product as proven are implemented. This is not only done for diagrams used for quantum applications but since DisCoPy can be used with arbitrary monoidal categories the implementation may be reused, if one has a category that obeys the rules. Implementing sums in this way was the second goal of this thesis.

Simplification algorithm Lastly, we propose a prototype of an algorithm for simplifying ZX-diagrams which arise from applications of QAOA to MaxCut. The algorithm uses a greedy approach, only introducing spiders, if all other operations do not yield simplification. Testing the algorithm on small examples yielded satisfactory results, which support its feasibility. Limitations are discussed and suggestions for their possible resolution are made. This prototype serves as a significant step towards general solutions for simplifying parameterized ZX-diagrams.

Future work

There are several topics for future work from this point forward. The first could be to find which additional rewrite-rules are needed for our extension of ZX-calculus to be complete. Tangentially it would be interesting to consider what other categories have similar structure and under which assumptions completeness holds.

Moreover, the presented prototype of an algorithm should be extended. This extensions can come in several forms. It would be interesting to analyse the asymptotic runtime and which conditions an input diagram needs to fulfil for the algorithm to terminate. Another fascinating path could be to extend the algorithm to work for either applications of QAOA to other combinatorial optimization problems or to different classes of ZX-diagrams alltogether. On a more practical level, we note that to improve practical relevance of the algorithm a user-friendly interface for automatic simplification is needed. Setting aside the proposed prototype we also wonder whether the addition of linear combinations can help in developing a general algorithm for simplifying parameterized ZX-diagrams.

Summary

In conclusion we have presented a formal extension of ZX-calculus, introducing linear combinations and proved necessary rewrite rules. Where our formulations are strictly stronger than the ones known in literature before. Furthermore, we presented a prototypical rewrite algorithm that simplifies a specific class of diagrams to an analytical form needed in applications. We have therefore reached the goals set at the start of this thesis, and have also illuminated several paths for future work.

Bibliography

[Awo06] Steve Awodey. *Category theory*. Oxford university press, 2006.

[Bha+22] Kishor Bharti et al. “Noisy intermediate-scale quantum algorithms”. In: *Rev. Mod. Phys.* 94 (1 Feb. 2022), p. 015004. DOI: 10.1103/RevModPhys.94.015004. URL: https://link.aps.org/doi/10.1103/RevModPhys.94.015004.

[BK21] Lennart Bittel and Martin Kliesch. “Training variational quantum algorithms is np-hard”. In: *Physical Review Letters* 127.12 (2021), p. 120502.

[Bor94] Francis Borceux. *Handbook of Categorical Algebra: Volume 2, Categories and Structures*. Vol. 2. Cambridge University Press, 1994.

[BPV21] Agustín Borgna, Simon Perdrix, and Benot Valiron. “Hybrid quantum-classical circuit simplification with the ZX-calculus”. In: *Asian Symposium on Programming Languages and Systems*. Springer. 2021, pp. 121–139.

[CDH20] Cole Comfort, Antonin Delpeuch, and Jules Hedges. “Sheet diagrams for bimonoidal categories”. In: *arXiv preprint arXiv:2010.13361* (2020).

[Cer+21] Marco Cerezo et al. “Variational quantum algorithms”. In: *Nature Reviews Physics* 3.9 (2021), pp. 625–644.

[CK17] Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, 2017. DOI: 10.1017/9781316219317.

[Coo71] Stephen A Cook. “The complexity of theorem-proving procedures”. In: *Proceedings of the third annual ACM symposium on Theory of computing*. 1971, pp. 151–158.

[Dem16] Wolfgang Demtröder. *Experimentalphysik 3: Atome, Moleküle und Festkörper*. Springer-Verlag, 2016.

[Dun+20] Ross Duncan et al. “Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus”. In: *Quantum* 4 (2020), p. 279.

[FGG14] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. “A quantum approximate optimization algorithm”. In: *arXiv preprint arXiv:1411.4028* (2014). URL: https://arxiv.org/abs/1411.4028.

[FLS15] Richard P Feynman, Robert B Leighton, and Matthew Sands. *Quantenmechanik*. Walter de Gruyter GmbH & Co KG, 2015.

[FTC20] Giovanni de Felice, Alexis Toumi, and Bob Coecke. “Discopy: monoidal categories in Python”. In: *arXiv preprint arXiv:2005.02975* (2020).

[Git22] Github. *Code frequency analysis for Quantomatic*. 2022. URL: https://github.com/Quantomatic/quantomatic/graphs/code-frequency (visited on 11/21/2022).

[GS18] David J Griffiths and Darrell F Schroeter. *Introduction to quantum mechanics.* Cambridge university press, 2018.

[HNW18] Amar Hadzihasanovic, Kang Feng Ng, and Quanlong Wang. "Two complete axiomatisations of pure-state qubit quantum computing". In: *Proceedings of the 33rd annual ACM/IEEE symposium on logic in computer science.* 2018, pp. 502–511.

[Hof20] Dirk W Hoffmann. *Grundlagen der technischen Informatik.* Carl Hanser Verlag GmbH Co KG, 2020.

[HV19] Chris Heunen and Jamie Vicary. *Categories for Quantum Theory: an introduction.* Oxford University Press, 2019.

[JPV18] Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. "A complete axiomatisation of the ZX-calculus for Clifford+ T quantum mechanics". In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science.* 2018, pp. 559–568.

[JPV22] Emmanuel Jeandel, Simon Perdrix, and Margarita Veshchezerova. "Addition and Differentiation of ZX-diagrams". In: *arXiv preprint arXiv:2202.11386* (2022).

[Kis12] Aleks Kissinger. "Pictures of processes: automated graph rewriting for monoidal categories and applications to quantum computing". In: *arXiv preprint arXiv:1203.0202* (2012).

[Kor+11] Bernhard H Korte et al. *Combinatorial optimization.* Vol. 1. Springer, 2011.

[KW20] Aleks Kissinger and John van de Wetering. "PyZX: Large Scale Automated Diagrammatic Reasoning". In: Proceedings 16th International Conference on *Quantum Physics and Logic,* Chapman University, Orange, CA, USA., 10-14 June 2019. Ed. by Bob Coecke and Matthew Leifer. Vol. 318. Electronic Proceedings in Theoretical Computer Science. Open Publishing Association, 2020, pp. 229–241. DOI: `10.4204/EPTCS.318.14`.

[Mac13] Saunders Mac Lane. *Categories for the working mathematician.* Vol. 5. Springer Science & Business Media, 2013.

[Mer92] Dennis Iligan Merino. *Topics in matrix analysis.* The Johns Hopkins University, 1992.

[Muu22] Gina Muuss. *Fix-branch for some bugs involving symbols in PyZX.* 2022. URL: `https://github.com/GinaMuuss/pyzx/tree/sympy-fixes-wip` (visited on 11/21/2022).

[NC02] Michael A Nielsen and Isaac Chuang. *Quantum computation and quantum information.* 2002.

[pro18] Quantomatic project. *Quantomatic.* 2018. URL: `https://quantomatic.github.io/index.html` (visited on 05/02/2022).

[SH22] Tobias Stollenwerk and Stuart Hadfield. "Diagrammatic Analysis for Parameterized Quantum Circuits". In: *arXiv preprint arXiv:2204.01307* (2022).

[Sho94] P.W. Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science.* 1994, pp. 124–134. DOI: `10.1109/SFCS.1994.365700`.

[SWY22] Razin Shaikh, Quanlong Wang, and Richie Yeung. “How to sum and exponentiate Hamiltonians in ZXW calculus”. In: *Quantum Physics and Logic 2022* (2022).

[Tan21] Till Tantau. *PGF/TikZ Manual.* "Accessed: 2022-10-22". Institut für Theoretische Informatik Universität zu Lübeck. Lübeck, Germany, 2021.

[Wet20] John van de Wetering. “ZX-calculus for the working quantum computer scientist”. In: *arXiv preprint arXiv:2012.13966* (2020).

[WY22] Quanlong Wang and Richie Yeung. “Differentiating and Integrating ZX Diagrams”. In: *arXiv preprint arXiv:2201.13250* (2022).

[Yeu22] Richie Yeung. *Pull-Request: Support representing symbolic graphs.* 2022. URL: `https://github.com/Quantomatic/pyzx/pull/97` (visited on 11/21/2022).

A. Appendix

A.1. Pseudo-code simplification algorithm

```python
def stage_one(diagram, inner_symbol, outer_symbol):
    # Look for the center pi spiders
    candidates = match_center_pi(diagram, inner_symbol)
    diagram = permutate_pi_through(diagram, candidates)

    # find some spiders with symbols where we pull the symbol into a scalar
    candidates = find_candidates_for_pull_symbol_to_scalar(disco_diag, inner_symbol)
    new_boxes = disco_diag.boxes
    for candidate in candidates:
        new_boxes[candidate] = sum_from_Xspider(disco_diag.boxes[candidate])
    new_diag = discoZxDiag(
        disco_diag.dom, disco_diag.cod, new_boxes, disco_diag.offsets
    )

    # we combine adjacent sums
    while len(candidates := find_combinable_sums(new_diag)) > 0:
        new_diag = combine_sums(new_diag, candidates[0])

    # distribute the sum outward and simplify independetly
    candidates = []
    for i, box in enumerate(new_diag.boxes[:-1]):
        if isinstance(box, LocalSum):
            candidates.append(i)

    candidate = candidates[0]
    results = []
    for term in new_diag.boxes[candidate].terms:
        dist_diag = discoZxDiag(
            new_diag.dom,
            new_diag.cod,
            new_diag.boxes[:candidate] + [term] + new_diag.boxes[candidate + 1 :],
            new_diag.offsets,
        )
        dist_diag = dist_diag.upgrade(
            discopy.quantum.zx.Functor(
                lambda x: x, lambda f: f, ob_factory=PRO, ar_factory=discoZxDiag
            )(dist_diag)
        )
        results.append(d)

    # return multiple diagrams, all only containing inner_symbol in global phases
    return results
```

```python
def stage_two(diagram, inner_symbol, outer_symbol):
    smth_changed = True
    while smth_changed:
        smth_changed = False

        if cycle := find_bialg_reverse(diagram):
            diagram = replace_bialg_reverse(diagram, cycle)
            smth_changed = True

        if smth_changed:
            continue
        smth_changed = smth_changed or zx.simplify.copy_simp(diagram) > 0

        if smth_changed:
            continue
        pi_comm_cand = find_pi_commute(diagram)
        if len(pi_comm_cand) > 0:
            diagram = apply_pi_commute(diagram, *(pi_comm_cand[0]))
            smth_changed = True

    return diagram
```

A.2. Open source contributions

As described in section Chapter 5 during this work we implemented several things in different open source projects. The two we actively contributed to are DisCoPy (`https://github.com/oxford-quantum-group/discopy`) and PyZX (`https://github.com/Quantomatic/pyzx`). To allow for repeatability we give the branch names and git-commit hashes the experiments in Section 5.3.2 were run on in Table A.1. At the time of writing, the current version of sympy was not working correctly with respect to modulo computation of non integer variables. Therefore, for our testing we used the current version of the master branch, the commit hash is also listed in the table below.

Project name	Branch	Commit Hash
DisCoPy	feat-localsum-wip1	49ef50c6b6ecc5c402d1a931a8630fdd2b31c103
PyZX	sympy-fixes-wip	da75f067546a621b216157f299b9303cd0baa1a0
Sympy	master	69c654b27d939718cd060172ad0fba95ada5a699

Table A.1.: Overview of open source versions for experiements