FRAUNHOFER INSTITUT (AISEC)

TECHNISCHE UNIVERSITÄT MÜNCHEN

Seminararbeit

ZX-Calculus

Author: Manuel Lerchner
Submission Date: July 10, 2023

CONTENTS

ZX-Calculus

Manuel Lerchner
Technical University of Munich
Munich, Germany

Abstract—**ZX-Calculus is a graphical language that extends classical quantum circuits by splitting up logic gates into even smaller building blocks. Those building blocks are called spiders and are represented by colored nodes in a graph. Together with edges connecting those nodes, they form a ZX diagram. A set of rewrite rules allows us to perform transformations on those ZX diagrams. ZX-Calculus promises a more intuitive way of reasoning about optimizing quantum circuits, as there are fewer rewrite rules to remember than in the classical logic gate model. This paper will introduce the ZX-Calculus, its rewrite rules, and some of its applications. I will give a particular focus on optimizing quantum circuits, as this is one of the main applications of the ZX-Calculus.**

Index Terms—**quantum computing, ZX-Calculus, quantum circuits, circuit optimization**

I. INTRODUCTION

In the last few decades, quantum computing has become a very active field of research. A significant reason for their success is their promise to outperform classical computers in specific tasks. The most famous example of such a speedup is Shor's algorithm [13], as it allows factorizing large numbers in polynomial time. The algorithm allows for a significant speedup compared to the best-known classical algorithm (GNFS), which takes sub-exponential time. Another example is Grover's algorithm [6], which allows for searching entries in an unsorted database in $\mathcal{O}(\sqrt{N})$ time. Grover's algorithm provides a quadratic speedup compared to the classical $\mathcal{O}(N)$ search time. [11]

Usually, those algorithms are described in a high-level language, the so-called (quantum-) circuit model. This model allows us to intuitively reason about quantum algorithms because every operation applied to the quantum state is represented by a single gate. However, this model does not consider the restrictions of real quantum computers [14] and needs to be compiled into a circuit that can be executed on a real quantum computer. This process is called *quantum circuit compilation*.

A. Quantum Circuit Compilation

Real quantum computers come with a set of restrictions. For example, the allowed gates are typically limited to a small set of universal gates (like the Clifford+T gate set). Furthermore, the connectivity of the qubits is limited. These restrictions imply that not every operation can be applied to every pair of qubits. Consequently, the original circuit needs to be *compiled* into an equivalent circuit that can be executed on the specific quantum computer while still respecting the restrictions. [14]
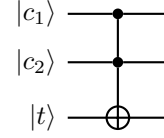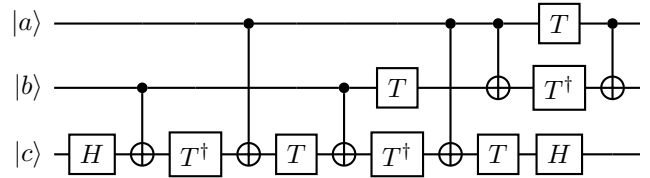

Fig. 1: The Toffoli gate.

Fig. 2: Decomposition of the Toffoli gate in the Clifford+T gate set.

B. Compilation of the Toffoli Gate

In this section, we will look at the compilation of the Toffoli gate using the Clifford+T gate set. We first look at the Toffoli gate, shown in figure 1. The Toffoli gate doesn't exist in the Clifford+T gate set, so to be able to execute it on a real quantum computer supporting only the Clifford+T gates, we need to compile it first. Many quantum computers only support the Clifford+T gate set because it is an approximately universal gate set, meaning that every circuit can be simulated to a certain degree of accuracy [11]. Furthermore, this set allows for an efficient, fault-tolerant implementation using surface code error correction [9].

C. Quantum Circuit Optimization

The compilation of the Toffoli gate is shown in figure 2. Notice how the amount of gates increases significantly. This increase in complexity is a common problem in quantum circuit compilation, as it means that the compiled circuit will be slower. This is a massive problem since the execution time of the new circuit can exceed the coherence time of the qubits, thus making the computation useless [11].

Circuit Optimizers can help to mitigate this problem, as they try to reduce the number of gates in a circuit. Optimizers often work by applying rewrite rules to the circuit, which keep the circuit equivalent to the original, but reduce the number of gates. An important metric for simplifying quantum circuits is the so-called *T-Count*. The T-Count is the number of T gates in a circuit. Since the T-Gate is a non-Clifford gate, it is costly

to simulate and requires orders of magnitudes more resources than the traditional Clifford gates. [9]

D. Drawbacks of Classical Circuit Optimization

There are many different approaches to quantum circuit optimization. The most basic approach is to apply a set of rewrite rules directly to the logic-gate representation of the circuit. This approach is called *gate-level optimization*. [10]

This trivial approach is typically very inefficient, as there exist many possible rewrite rules (Some of them are shown in figure 3). Furthermore, rewrite rules are typically not independent of each other, as applying a rewrite rule can introduce new opportunities for different rewrite rules, thus making it very hard to find an optimal solution [8].
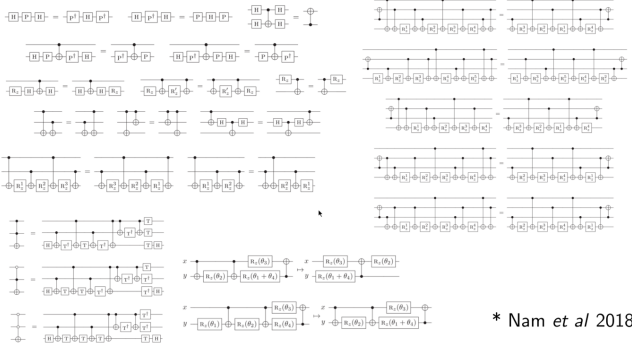
Fig. 3: Small subset of rewrite rules for classical circuit optimization [8]

E. Quantum Circuit Optimization using the ZX-Calculus

A more efficient approach for quantum circuit optimization is to use the ZX-Calculus. The ZX-Calculus is a graphical language for quantum computing, differing from the logic-gate representation by using connected nodes to represent operations on qubits. This new representation utilizes way fewer *gate-types* than the classical representation; consequently, there are fewer rewrite rules to consider in each step.

ZX-Calculus has been kickstarted by Coecke and Duncan in 2008 [2]. Since then, it has been used to prove many exciting results in quantum computing. It was initially used in the field of Measurement-Based Quantum Computing (MBQC) [3]. But recently, it has found wide application in quantum circuit optimization and verification [15].

II. MATHEMATICAL BACKGROUND: CATEGORY THEORY

The fundaments of ZX-Calculus are based on the mathematical model of Category-theory. A category is an abstract mathematical model which consists of a class of objects and a set of morphisms [16]. Objects represent the current *state* of the calculation, whereas morphisms act like mappings between those objects. In particular, a ZX-diagram is represented by strict compact closed symmetric monoidal category [7]. Which has (among other things) the following properties:

Given such a category $\mathcal{C}$ with objects $\{A, B, C, D\}$ and morphisms $\{f : A \to B, g : B \to C, h : C \to D\}$, there

exists a notion of composition . Therefore, we can compose morphisms that are applied sequentially to get new morphisms. We can, for example, combine f and g to get the morphism $g \circ f : A \to C$. The composition operator is associative. Thus the following statement holds $(h \circ g) \circ f = h \circ (g \circ f)$. An identity morphism $id_A : A \to A$ exists for each object A. This identity morphism is neutral concerning composition, meaning that $f \circ id_A = f = id_B \circ f$.

There also exists a Bifunctor $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ which is used to combine multiple *parallel* morphisms into a single morphism . In our example, we can combine f and g to get a new morphism $f \otimes g : A \otimes B \to C \otimes D$. This Bifunctor is also associative, meaning that $(f \otimes g) \otimes h = f \otimes (g \otimes h)$ always holds. Furthermore, there exists a unit object I which acts as a neutral element concerning the Bifunctor, meaning that $f \otimes I = f = I \otimes f$ for each morphism f.

The category also provides a natural isomorphism $\sigma_{A,B} : A \otimes B \to B \otimes A$ for each pair of objects A and B. This means that swapping the order of two parallel morphisms is always possible.

Finally, there also exists a notion of dual objects. Given an object A, there exists a dual object A^*. By using the special morphisms $\eta_A : I \to A \otimes A^*$ and $\epsilon_A : A^* \otimes A \to I$, we can introduce a notion of *curved* morphism. These unique morphisms are called *CAP* and *CUP*, respectively. We will see later that these morphisms are used to introduce a notion of *entanglement* between two objects.

A. Quantum Circuits based on Category Theory

Using the mathematical model of category theory, we can now define a quantum circuit based purely on objects and morphisms. We will restrict our category to the category of finite dimensional Hilbert spaces and linear maps to achieve a concrete model. Particularly, this means that the class of the objects in our category are finite-dimensional Hilbert spaces, and the morphisms are linear maps between those Hilbert spaces. This category is denoted as **FDHilb**. Furthermore, we define the sequential composition of morphisms as the matrix multiplication of the underlying linear maps. The Bifunctor $\otimes$ is defined as the Kronecker product respectively.
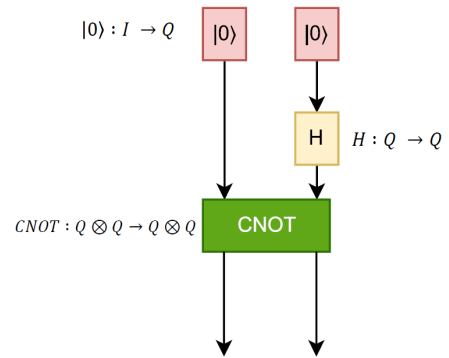

Fig. 4: Circuit for creating Bell pairs

Using this model, we can now define an actual circuit: We will look at the entanglement circuit shown in figure 4. This circuit produces entangled qubit pairs. Note that the visual representation looks very similar to the classical gate-based circuit. The only difference is that we translate it to use the morphisms of our category instead of the gates.

The circuit shown in figure 4 consists of three morphisms.

- $|0\rangle$ is defined as the morphism $I \to Q$ which initializes a qubit to the state $|0\rangle$.
- H is a morphism of the form $Q \to Q$. It is a linear map between two qubits, acting as a single qubit Hadamard gate.
- $CNOT$ is the controlled-not gate, analogously to the classical **CNOT** gate, it represents a linear map from two qubits to two qubits. It is a morphism of the form $Q \otimes Q \to Q \otimes Q$.

At the moment, we are only able to define the circuit abstractly. As we only represented the circuit using the abstract morphisms from above. In the next section, we will describe the objects and morphisms used in the category, thereby introducing the basic building blocks of ZX-Calculus.

III. INTRODUCTION TO ZX-CALCULUS

As stated in the previous section, the ZX-Calculus is represented by the **FDHilb** category, meaning that the objects of our category are finite-dimensional Hilbert spaces, and the morphisms are linear maps between those Hilbert spaces. The morphisms will be represented using the so-called *Spiders*. This definition means that the whole classical quantum circuit will be represented by a network of morphism/spiders. This representation is advantageous, as it allows for a straightforward application of the simplification rules we will define later.

A. Spiders

Spiders are the *atoms* of a ZX-Diagram. They represent the decomposition of quantum gates into even smaller and more fundamental operations. Spiders can have an arbitrary number of incoming and outgoing edges and appear in two flavors: *Z-Spiders* and *X-Spiders*. This distinction is shown visually using the green and red colors when drawing the spiders. Additionally, spiders may carry a phase value $\alpha \in [0, 2\pi)$, which can be omitted if the angle is zero.

The spiders are shown visually in figure 5. The first spider is a *Z-Spider*, and the second is an *X-Spider*. Both have n incoming and m outgoing edges, carrying a phase value α.

Fig. 5: Fundamental spiders with n incoming and m outgoing edges

It is important to remember that each spider represents a morphism in the **FDHilb** category. Therefore, each spider represents a linear map from the incoming n to the outgoing m qubits. Those linear maps can be calculated explicitly and

$$\left[\!\!\left[\ n \cdots \alpha \cdots m\ \right]\!\!\right] := \underbrace{|0\ldots0\rangle}_{m}\underbrace{\langle0\ldots0|}_{n} + e^{i\alpha}\underbrace{|1\ldots1\rangle}_{m}\underbrace{\langle1\ldots1|}_{n}$$

$$= \begin{pmatrix} 1 & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & e^{i\alpha} \end{pmatrix}$$

$$\left[\!\!\left[\ n \cdots \alpha \cdots m\ \right]\!\!\right] := \underbrace{|+\cdots+\rangle}_{m}\underbrace{\langle+\cdots+|}_{n} + e^{i\alpha}\underbrace{|-\cdots-\rangle}_{m}\underbrace{\langle-\cdots-|}_{n}$$

Fig. 6: Linear maps represented by individual spiders

are shown in figure 6. We will use the notation $[\![\cdot]\!]$ to denote the linear map represented by a spider.

Note that the linear maps in figure 6 do not have to be unitary, nor do they have to be square. The dimension of the linear map depends only on the number of incoming and outgoing edges. For example, a spider with n incoming and m outgoing edges represents a linear map in $\mathbb{C}^{2^m \times 2^n}$.

B. Classical Quantum Gates as Spiders

Name	Diagram	Matrix
identity	———	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
Pauli Z	—π—	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
Pauli X NOT gate	—π—	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Pauli Y	i —π—π—	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$
Hadamard gate	—□—	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
S gate	—$\frac{\pi}{2}$—	$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$
V gate	—$\frac{\pi}{2}$—	$\frac{1}{2}\begin{pmatrix} 1+i & 1-i \\ 1-i & 1+i \end{pmatrix}$
T gate	—$\frac{\pi}{4}$—	$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}$

Fig. 7: Spiders representing quantum gates [15][P.87]

By using the single spiders shown in figure 5 we can already represent a lot of quantum gates. The most important ones are shown in figure 7. We will ignore the global scalar values of ZX-Diagrams in the following sections since they

are negligible for most calculations [14]. Furthermore, when working with unitary circuits, the scalar value can be restored at the end of the analysis as the resulting circuit matrix can be related to the unitary matrix [15].

It is easy to verify that the gates from figure 7 can be represented using just the basic spiders. This proof boils down to just inserting the corresponding spiders into the formulas from figure 6 and comparing the resulting matrices with the matrices of the Pauli gates. These proofs are shown in the Appendix in figures A, B and C.

C. Bigger Circuits

Using the rules from figure 7 we can only represent circuits consisting of a single gate. For more extensive circuits, we need to combine multiple spiders into a single diagram. This is done by connecting the different spiders leg to leg. In particular, this means that one spider's outgoing leg connects to another spider's incoming legs.

But before working with such circuits, we need to define more rules to calculate the matrix representation of such a composed circuit.

1) **Only Topology Matters**
   The first rule is that we can move spiders around freely if we maintain the correct order of the incoming and outgoing legs. This rule corresponds to the *Only Topology Matters* mantra of the ZX-Calculus.
2) **Parallel Composition**
   The second rule allows us to calculate the matrix representation of spiders acting in parallel. We had seen this rule before when we looked at the category representation of parallel morphisms (II). In **FDHilb**, this rule corresponds to the Kroncker product of matrices for the individual spiders.
3) **Sequential Composition**
   The third rule allows us to calculate the matrix representation of spiders acting in sequence. This rule corresponds to the sequential composition (II) in the category representation of the ZX-Calculus. In particular, it states that the composed linear map can be calculated by multiplying the individual spiders' matrices.

D. Example: CNOT Gate

An example of a circuit consisting of multiple spiders is the classical **CNOT** gate, represented by the ZX-Diagram shown in figure 8.

To calculate this circuit's matrix representation, we must first split the ZX-Diagram into regions consisting of individual spiders. The result of this splitting is already shown in figure 8a.

Using the rules we just defined, we can now calculate the matrix representation of the circuit. We apply the *Parallel Composition* rule to combine sections A_1 and A_2 into a new section A. The same is done for sections B_1 and B_2, respectively. This reduction results in the ZX-Diagram shown in figure 8b.

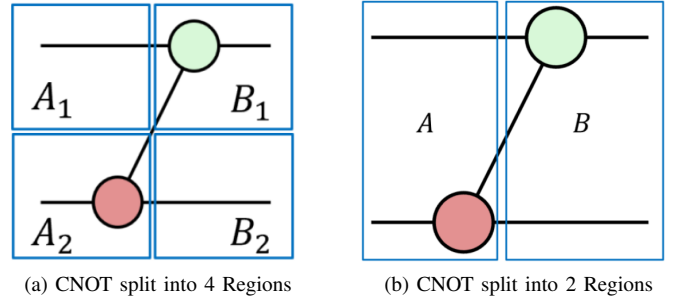(a) CNOT split into 4 Regions   (b) CNOT split into 2 Regions

Fig. 8: Splitting the CNOT gate into regions

We can also write this calculation using a decomposition formula: Let R be the matrix representation of the circuit shown in figure 8a. Then we can write the reduction as shown in equation 1. All elements of the decomposition are individual spiders, so we can easily calculate their matrix representation using the rules from figure 6.

$$\underbrace{\left[\!\!\left[\; \right]\!\!\right]}_{R} = \underbrace{\left[\!\!\left[\; \right]\!\!\right]}_{A} \circ \underbrace{\left[\!\!\left[\; \right]\!\!\right]}_{B}$$

$$= (\underbrace{\left[\!\!\left[\; \right]\!\!\right]}_{A_1} \otimes \underbrace{\left[\!\!\left[\; \right]\!\!\right]}_{A_2}) \circ (\underbrace{\left[\!\!\left[\; \right]\!\!\right]}_{B_1} \otimes \underbrace{\left[\!\!\left[\; \right]\!\!\right]}_{B_2}) \quad (1)$$

The concrete calculation looks as follows:

$$A = A_1 \otimes A_2$$
$$= id \otimes \left[\!\!\left[\; \right]\!\!\right]$$
$$= id \otimes |+\rangle\langle + + | + |-\rangle\langle - - |$$
$$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}$$
$$= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$B = B_1 \otimes B_2$$
$$= \left[\!\!\left[\; \right]\!\!\right] \otimes id$$
$$= \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$
$$= \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Using the the *Sequential Composition* rule we can combine sections A and B into the resulting matrix R:

$$R = A \circ B$$
$$= B \cdot A$$
$$= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$
$$\propto \text{CNOT}$$

The resulting matrix R is proportional to the matrix representation of the CNOT gate. Therefore we showed that the ZX-Diagram ⟋ is indeed a valid representation of the CNOT gate.

IV. Simplifcation Rules

In the previous section, we saw how to calculate a ZX-diagram's matrix representation. But the way we used to evaluate the circuit was very inefficient. We had to calculate the matrix representation of every single spider and had to perform heavy matrix multiplications and Kronecker products. This is the same process as if we had just calculated the circuit using the classical logic gate model. So we did not gain anything from using the optimization approach via the ZX-Calculus. In this section, we will look at a set of rewrite rules which allow us to simplify ZX-diagrams by using rewrite rules instead of performing heavy matrix operations.

Note that all the rules we will discuss can be applied in both directions. In particular, this means we can use the rules to simplify a diagram and make it more complex if it helps with further simplifications. Additionally, we can apply the rules to any subgraph of the diagram, provided that the shape matches the rule. It is also essential to know that all the rules also hold if we replace all the spiders with their adjoint. This means that we can swap the color of **all** spiders in a rule and can still expect the rule to hold.

We also don't have to track which leg of the spider is an input and which is the output. This rule corresponds to the *Only topology matters* mantra we introduced in the previous section.

A. Identity Removal

The rule we will look at is the identity removal rule. The rule states that we can remove any spider with exactly one input and one output if it has a phase of 0 and can replace it with a single edge. The application of this rule is shown in figure 9.

Fig. 9: Identity Removal Rule

B. Spider Fusion

The spider fusion rule allows us to fuse two spiders if an edge connects them. The inputs and outputs of the two spiders are merged together. Additionally, the phases of the two spiders must be added together.

This rule is shown in figure 10.

Fig. 10: Spider Fusion Rule

C. Hopf Rule

The Hopf rule allows us to split apart two spiders of opposite colors, given that two edges connect them.

This rule is shown in figure 11.

Fig. 11: Hopf Rule

D. Copy Rule

The copy rule allows copying the computational basis states through a spider of the opposite color. This rule corresponds to the *"Green copies Red"* / *"Red copies Green"* mantra. The copied state is then moved to **all** the outputs of the spider.

This rule is shown in figure 12.

Fig. 12: Copy Rule

Note that this rule only works when a is 0 or 1, as we can only copy computational basis states as they are orthogonal to each other [1]. This means only $|0\rangle$, $|1\rangle$, $|+\rangle$ and $|-\rangle$ can be copied using this rule.

E. Bi-Algebra Rule

The bi-algebra works as seen in figure 14. It has an analogous rule in digital logic, stating that the copied *XOR* result corresponds to first copying the inputs and then performing two *XOR* operations on the swapped inputs. This is shown in figure 13. The ZX-Calculus analogy uses the copy-rule from figure 12 and a new *XOR* spider.
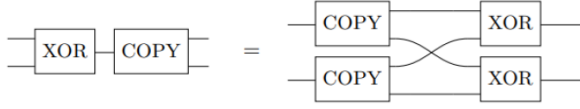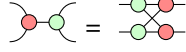
Fig. 13: Bi-Algebra Rule in Digital Logic

Fig. 14: Bi-Algebra Rule

F. π-Commutation Rule

The π-commutation rule allows us to move a spider with phase π through a spider of the opposite color. By moving the spider through the other spider, the phase of the spider is inverted. This rule is shown in figure 15.
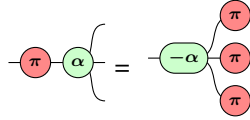

Fig. 15: π-Commutation Rule

G. Color Change Rule

The color change rule allows us to change the color of a spider. This is done by adding *H* gates to all the inputs and outputs of the spider. This rule is shown in figure 16.

Fig. 16: Color Change Rule

Where a Hadamard gate is defined in figure 17, note that we use the decomposition of the Hadamard gate into three sequential $\frac{\pi}{2}$ rotations to represent the Hadamard gate in the ZX-calculus.
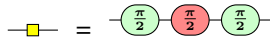

Fig. 17: Hadamard Gate

V. EXAMPLE: ZX-DIAGRAMS OPTIMIZATION

A. GHZ Circuit

We now have all the tools to simplify a ZX diagram. Notice that due to the small number of *atoms*, it's way easier to come up with a complete set of rewrite rules. In contrast to the classical logic gate model, where we have many different gates, we only have two different kinds of spiders. This makes finding an efficient simplification much easier.

We will now look at a simple example of using ZX-Calculus to simplify a circuit. We will use the circuit from figure 18 as an example.
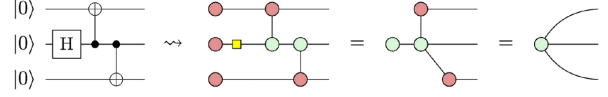

Fig. 18: GHZ Circuit in ZX-Calculus

To achieve a simplified circuit, we will perform these steps:
1) In the first step, we translated the gates from the circuit into their corresponding ZX diagrams. We used the **CNOT** and **Hadamard** circuits from before to perform a lookup and translate the gates into their corresponding ZX diagrams.
2) Now we can start simplifying the diagram. We will begin by applying the **Spider Fusion** rule to merge all free spiders.
3) Next, we will apply the **Color Change** rule to change the color of the spider left of the Hadamard gate. This operation cancels out the Hadamard gate.
4) After that, we perform the Spider Fusion rule again and remove the identity spiders; we end up with the simplified diagram shown in the rightmost diagram of figure 18. Using the formula for calculating the matrix representation of single spiders, we compute $|GHZ\rangle = 1 \cdot |000\rangle + 1 \cdot |111\rangle = |000\rangle + |111\rangle$, which is proportional to the actual GHZ state. Remember that to get the correct GHZ state, we need to normalize the state first.

By using the intermediate steps of translating the circuit into a ZX-diagram first, we were able to effortlessly calculate the matrix representation of the circuit, which would have been a lot harder if we had to calculate the matrix representation of the circuit using only its gates and the classical circuit model.

B. Quantum Teleportation

We will now look at a more complex example. We will look at the quantum teleportation circuit from figure 19.
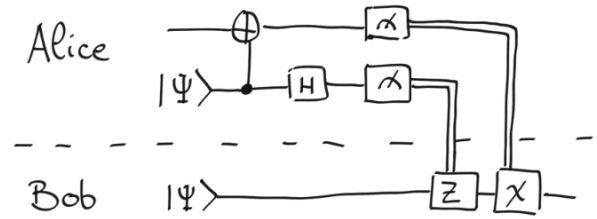

Fig. 19: Quantum Teleportation Circuit Img: Pennylane [12]

To understand the circuit, we will first look at the representation of measurements and entangled states in the ZX-Calculus. The measurement operation in a given basis is represented by parameterized spiders of the same basis [15]. The parameter of the measurement-spider is a boolean value representing the measurement outcome. Based on the measurement result, we can influence other parts of the circuit by including the

parameter of the measurement spider in other spiders. Another interesting thing to notice is that the shared bell state, which is the key to the quantum teleportation algorithm, can be represented by the spider $\subset$. This spider is the special η-morphisms we introduced earlier in the mathematical foundation of the ZX-Calculus. It has a matrix representation of $\left[\!\left[\subset \right]\!\right] = \begin{pmatrix} 1 & 0 & 0 & 1 \end{pmatrix}^T = |\phi^+\rangle$ and therefore represents a bell state.

In figure 20, we translated the circuit from figure 19 into a ZX diagram. Notice that it can be clearly seen from the diagram that the shared bell state works as the *bridge* between Alice and Bob.
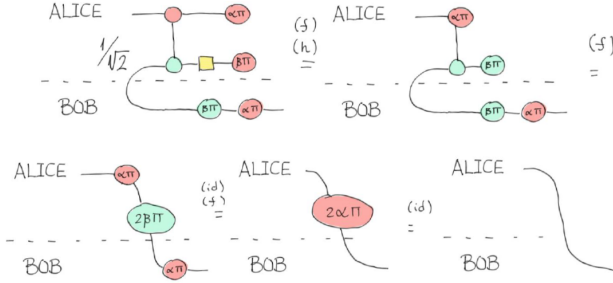

Fig. 20: Teleportation in the ZX-Calculus Img: Pennylane [12]

By applying the earlier rewrite rules, we can deduce that the whole circuit *cancels* itself out, as the classically measured values from Alice perfectly negate the effects of Allice. This means that the circuit is equivalent to a direct channel between Alice and Bob. Therefore we showed that the teleportation works as expected.

VI. Universality & Completeness

As stated in [4], the ZX-Calculus is (approximately) universal because any linear map can be represented as a ZX-diagram. Therefore, any valid quantum circuit can be described as a ZX diagram. Additionally, it can be shown that ZX-Calculus is complete for circuits that are expressed using the *Clifford+T* gates. This means that any two circuits in this family can be transformed into each other by a series of rewrite rules from above. In particular, this means that if a simplified version of the circuit exists, it can be found by the rules of the ZX-Calculus. Figure 21 shows an example of such a simplification process.
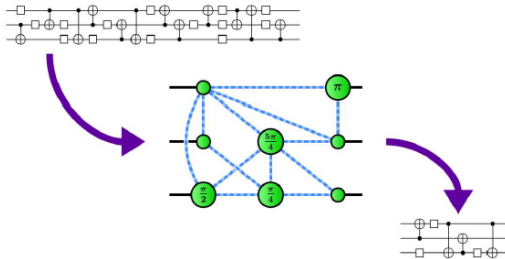

Fig. 21: Simplification process [5]

VII. Further Work

A significant problem concerning the optimization via ZX-Calculus, which is actively being researched, is the development of efficient extraction algorithms. Such algorithms are needed because when we are done simplifying a circuit, we are left with a compact ZX-Diagram. Since we want to be able to simulate the optimized circuit, we need to specify an algorithm that extracts a classical circuit out of ZX-Diagrams. This generally is not a trivial task since ZX-Diagrams are a superset of the classical quantum circuits [4] as seen in figure 22, which means that not every ZX-Diagram corresponds to a valid quantum circuit. An important optimization procedure is keeping enough information about the quantum circuit to allow for an extraction of a classical circuit after the simplification. [4].
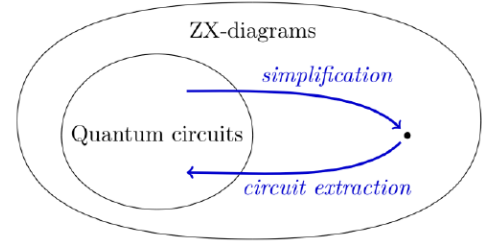

Fig. 22: Simplifcation process [4]

VIII. Conclusion

We have seen that ZX calculus is a powerful tool for reasoning about quantum circuits and offers new ways to simplify quantum circuits and reason about quantum algorithms. By relying on a graphical language, which is very intuitive and easy to use, ZX calculus is a handy tool for teaching quantum computing and quantum algorithms.

ZX calculus is a very new field of research and remains very active as there are still many open questions left to answer. ZX calculus promises a potential for new discoveries and applications in the field of quantum computing.

REFERENCES

[1] Dave Bacon. Cse 599d - quantum computing the no-cloning theorem, classical teleportation and quantum teleportation, super-dense coding. https://courses.cs.washington.edu/courses/cse599d/06wi/lecturenotes4.pdf, 2006. accessed: 08.07.2023.

[2] Bob Coecke and Ross Duncan. A graphical calculus for quantum observables. *Preprint*, 2007.

[3] Ross Duncan. A graphical approach to measurement-based quantum computing, 2012.

[4] Ross Duncan, Aleks Kissinger, and John van de Wetering. Graph-theoretic simplification of quantum circuits with the zx-calculus, 2020.

[5] Ross Duncan, Aleks Kissinger, and John van de Wetering. Graph-theoretic simplification of quantum circuits with the zx-calculus. https://quantum-journal.org/papers/q-2020-06-04-279/, 2020. accessed: 09.07.2023.

[6] Lov K. Grover. A fast quantum mechanical algorithm for database search, 1996.

[7] Emmanuel Jeandel. Completeness of the zx-calculus, 2020.

[8] Aleks Kissinger. An introduction to the zx-calculus. https://www.cs.ox.ac.uk/people/aleks.kissinger/slides/fullstack-25mins.pdf, 2020. accessed: 30.06.2023.

[9] Aleks Kissinger and John van de Wetering. Reducing t-count with the zx-calculus, 2020.

[10] Yunseong Nam, Neil J. Ross, Yuan Su, Andrew M. Childs, and Dmitri Maslov. Automated optimization of large quantum circuits with continuous parameters, may 2018.

[11] Michael A. Nielsen and Isaac L. Chuang. Quantum computation and quantum information, 2010.

[12] PennyLane. Introduction to the zx-calculus. https://pennylane.ai/qml/demos/tutorial_zx_calculus, 2023. accessed: 09.07.2023.

[13] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring, 1994.

[14] Robert Wille Tom Peham, Lukas Burgholzer. Equivalence checking of quantum circuits with the zx-calculus. https://www.cda.cit.tum.de/files/eda/2022_jetcas_equivalence_checking_of_quantum_circuits_with_the_zx_calculus.pdf, 2022. accessed: 30.06.2023.

[15] John van de Wetering. Zx-calculus for the working quantum computer scientist, 2020.

[16] Quanlong Wang. Completeness of the zx-calculus. https://arxiv.org/pdf/2209.14894.pdf, 2023. accessed: 09.07.2023.

APPENDIX

A. ZX-Representation of the Pauli Z-Gate

$$
\begin{aligned}
&\equiv |\underbrace{0\ldots0}_{1}\rangle\langle\underbrace{0\ldots0}_{1}| + e^{i\pi}|\underbrace{1\ldots1}_{1}\rangle\langle\underbrace{1\ldots1}_{1}| \\
&= |0\rangle\langle0| - |1\rangle\langle1| \\
&= \begin{pmatrix} 1 \\ 0 \end{pmatrix}\begin{pmatrix} 1 & 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 1 \end{pmatrix}\begin{pmatrix} 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \\
&= Z
\end{aligned}
$$

Fig. 23: Pauli-Z gate represented as a ZX-Diagram

B. ZX-Representation of the Pauli X-Gate

$$
\begin{aligned}
&\equiv |\underbrace{+\cdots+}_{1}\rangle\langle\underbrace{+\cdots+}_{1}| + e^{i\pi}|\underbrace{-\cdots-}_{1}\rangle\langle\underbrace{-\cdots-}_{1}| \\
&= |+\rangle\langle+| - |-\rangle\langle-| \\
&= \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \end{pmatrix} - \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix}\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \end{pmatrix} \\
&= \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} - \frac{1}{2}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \\
&= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\
&= X
\end{aligned}
$$

Fig. 24: Pauli-X gate represented as a ZX-Diagram

C. ZX-Representation of the Pauli Y-Gate

$$
\begin{aligned}
&\equiv \;\; \circ \\
&\equiv Z \circ X \\
&= XZ \\
&= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \\
&= \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \\
&= Y/i \\
&\propto Y
\end{aligned}
$$

Fig. 25: Pauli-Y gate represented as a ZX-Diagram