FRAUNHOFER INSTITUT (AISEC)

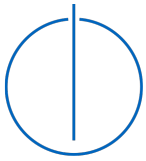
TECHNISCHE UNIVERSITÄT MÜNCHEN

Seminararbeit

ZX-Calculus

Author: Manuel Lerchner

Submission Date: July 9, 2023



CONTENTS

ZX-Calculus

Manuel Lerchner
Technical University of Munich
Munich, Germany

***Abstract*—ZX-Calculus is a graphical language which 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. Using a set of rewrite rules, ZX-diagrams can be transformed into each other. This allows for a more intuitive way of reasoning about the optimization of quantum circuits, as there are fewer rewrite rules to remember than in the classical logic gate model. In this paper, I will introduce the ZX-Calculus and its rewrite rules, as well as some of its applications. I will give a special focus on the optimization of 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. The main reason for this is the fact that quantum computers promise to outperform classical computers in certain tasks. The most famous example of this is Shor's algorithm [13] which can factorize large numbers in polynomial time. This is a problem that is believed to be intractable for classical computers. Another example is Grover's algorithm [6] which can search an unsorted database in $\mathcal{O}(\sqrt{N})$ time. This is a quadratic speedup compared to the classical $\mathcal{O}(N)$ time.

Usually those algorithms described in a very high-level "language", the so called (quantum-) circuit model. This allows for a very intuitive way reasoning about quantum algorithms, because every unitary operation applied to the quantum state can be compactly represented by a single gate. However, this model does not take the restrictions of real quantum computers into account. [14]

A. Quantum Circuit Compilation

Real quantum computers come with a set of restrictions. For example, the set of gates is typically limited to a small set of universal gates (e.g. the Clifford+T gate set). Furthermore, the connectivity of the qubits is limited. This means that not every operation can be applied to every pair of qubits. As a consequence, the original circuits needs to be *compiled* into a circuit that can be executed on the specific quantum computer. This process is called *quantum circuit compilation*.

In circuit 1 the Toffoli gate is shown. However it is not represented in the Clifford+T gate set. In order to execute this circuit on a real quantum computer, we need to transform it into the Clifford+T gate set first, as this allows an efficient, and fault-tolerant implementation using surface code error correction. [9]

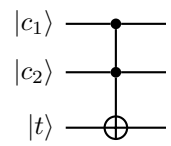


Fig. 1: The Toffoli gate.

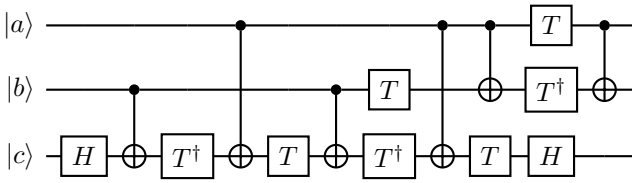


Fig. 2: Decomposition of the Toffoli gate in the Clifford+T gate set.

B. Quantum Circuit Optimization

Such a decomposition is shown in figure 2. Notice how the amount of gates increases significantly. This is a common problem in quantum circuit compilation. It means that the compiled circuit will be slower and therefore the execution time can exceed the coherence time of the qubits, and thus making the computation useless. [11]

This is where Circuit Optimizers come into play. They try to reduce the amount of gates in a circuit. This can be done by applying a set of rewrite rules to the circuit. The goal is to find a circuit that is equivalent to the original circuit, but has less gates. An important metric for simplifying quantum circuits is the so called *T-Count*. The T-Count is the amount of T gates in a circuit. Since the T-Gate is a non-Clifford gate it is very expensive to simulate and requires order of magnitudes more resources than the other clifford gates. [9]

C. 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]

However, this approach is typically very inefficient, as there exists a huge amount of possible rewrite rules (Some of them are shown in figure 3). Furthermore, rewrite rules are typically not independent of each other. This means that applying a rewrite rule can introduce new opportunities for other rewrite rules. This makes it very hard to find an optimal solution. [8]

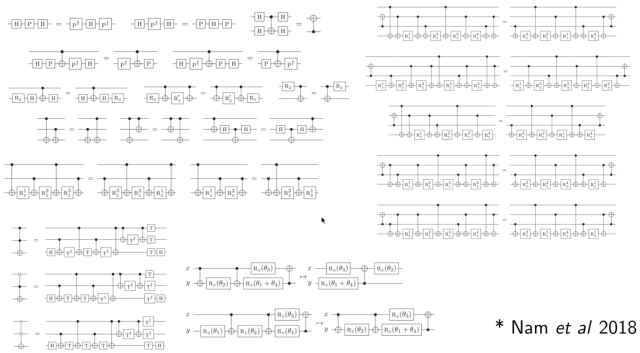


Fig. 3: Small subset of rewrite rules for classical circuit optimization [8]

D. 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. Since this new representation utilizes way fewer *gate-types* than the classical representation, there exist way fewer rewrite rules.

ZX-Calculus has been kickstarted by Coecke and Duncan in 2008 [2]. Since then, it has been used to prove a lot of interesting 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 ZX-Calculus is based on the mathematical model of Category-theory. A category is an abstract mathematical model which consists of objects and morphisms. Objects are the building blocks of the category, whereas morphisms act like mappings between objects. In particular a ZX-diagram is a strict compact closed symmetric monoidal categor [7].

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 . This means that we can compose morphisms to get new morphisms. In our example, we can compose f and g to get a new morphism $g \circ f : A \to C$. This composition is associative, meaning that $(h \circ g) \circ f = h \circ (g \circ f)$. Additionally, there exists an identity morphism $id_A : A \to A$ for each object A. This identity morphism is neutral with respect to 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 associative, meaning that $(f \otimes g) \otimes h = f \otimes (g \otimes h)$. Additionally, there exists a unit object I which acts as a neutral element with respect to the Bifunctor, meaning that $f \otimes I = f = I \otimes f$ for each morphism f.

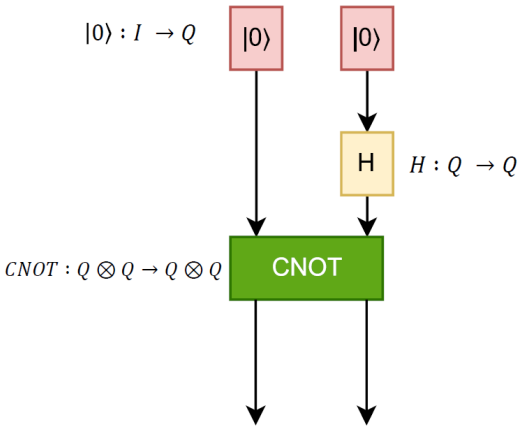


Fig. 4: Circuit for creating Bell pairs

Furthermore, there exists a natural isomorphism $\sigma_{A,B} : A \otimes B \to B \otimes A$ for each pair of objects A and B. This means that we can always swap the order of parallel morphisms without changing their meaning. This is also called the *swap* rule.

Finally, there also exists a notion of dual objects. Given an object A, there exists a dual object A^*. 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. This special morphisms are called *CAP* and *CUP* respectively.

A. Quantum Circuits based on Category Theory

Using the mathematical model of category theory, we can now define a quantum circuit. For this we are going to restrict our category to the category of finite dimensional Hilbert spaces and linear maps. This means that the objects of our category are finite dimensional Hilbert spaces and the morphisms are linear maps between those Hilbert spaces. We are going to denote this category as **FDHilb**. Furthermore we define the composition of morphisms as the matrix multiplication of linear maps. The Bifunctor $\otimes$ is defined as the tensor/kronecker product respectively.

We can now define an actual circuit, using this model: For this we are going to look at an entanglement circuit, which creates Bell pairs. This circuit is shown in figure 4. Note that it looks very similar to the normal gate-based circuit. The only difference is that we are translating it in such a way, that we are using the morphisms of our category instead of the gates.

It uses the following morphisms:

- $|0\rangle$ is 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 from one qubit to one qubit.
- $CNOT$ is the controlled-not gate, which is 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. We can not yet define the actual circuit, because we have not yet defined specifically what the objects and

morphisms of our category are. We are going to do this in the next section, where we are going to define the basic building Blocks of ZX-Calculus.

III. Introduction to ZX-Calculus

As stated in the previous section, the ZX-Calculus represents the **FDHilb** category. This means that the objects of our category are finite dimensional Hilbert spaces and the morphisms are linear maps between those Hilbert spaces. We are going to represent the morphisims as *Spiders*. So in total the whole classical quantum circuit is represented as a graph of morphism/spiders. In this representation it is very easy to to apply simplification rules to the circuit.

A. Spiders

Spiders are the *atoms* of a ZX-Diagram. They represent the decomposition of quantum gates into even smaller and more fundamental operations. Some important decompositions are shown in figure 7.

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 color respectively when drawing the spiders. Additionally spiders may carry a phase value $\alpha \in [0, 2\pi)$. Which can be ommited if it is zero.

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

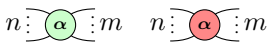


Fig. 5: Fundamental spiders with n incoming and m outgoing edges

It is important to remember that each spider represents a morphism in the **FDHilb** category. This means that each spider represents a linear map from the incoming n to the outgoing m qubits. Those linear maps are shown in figure 6.

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

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

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

Fig. 6: Linear maps represented by individual spiders

Note that the linear maps in figure 6 do not have to be unitary, nor do they have to 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 $\mathbb{C}^{2^m \times 2^n}$.

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]

B. Classical Quantum Gates as Spiders

By just 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. Note that we are going to ignore the global scalar values of ZX-Diagrams in the following sections, since they are negligible for most cases [14]. Furthermore, when working with unitary circuits, the scalar value can be restored at the end of the calculation as the resulting circuit-matrix is proportional to the unitary matrix [15].

It is easy to verify that the gates from the figure 7 can indeed be represented using just the basic spiders. This boils down just inserting the corresponding spiders into the formulas from figure 6 and to compare the resulting matrices with the matrices of the Pauli gates. These proofs are shown 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 bigger circuits we need to combine multiple spiders into a single diagram. This is done by connecting the spiders leg to leg. This means that the outgoing legs of one spider are connected to the incoming legs of another spider.

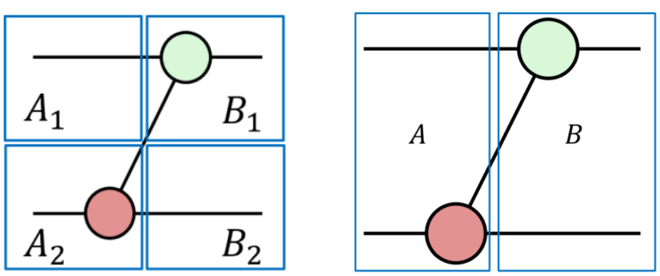
But before we can work with such circuits we need to define some more rules to calculate the matrix representation of a circuit.

1) **Only Topology Matters**
The first rule is that we can move spiders around freely, as long as we maintain the correct order of the incoming andoutgoing legs. This rule corresponds to the *Only Topology Matters* mantra of the ZX-Calculus.
2) **Parallel Composition** The second rule allows to calculate the matrix representation of spiders acting in parallel. We have seen this rule bevore, when we looked at the category representation of the ZX-Calculus (II). In **FDHilb** this rule corresponds to the kroncker product of matrices for the individual spiders.
3) **Sequential Composition** The third rule allows 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 allows to calculate the matrix representation such a circuit by multiplying the matrices of the individual spiders.

D. Example: CNOT Gate

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

In order to calculate the matrix representation of this circuit we first need to split the circuit into the individual spiders. The result this process is already shown in figure 8a.

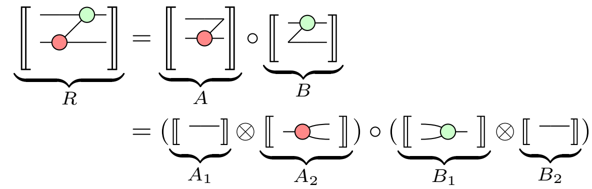


(a) CNOT split into 4 Regions (b) CNOT split into 2 Regions

Fig. 8: Splitting the CNOT gate into regions

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

We can also write this reduction as a 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.



(1)

The concrete calculation looks as follows:

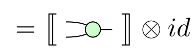
$$A = A_1 \otimes A_2$$

$$= id \otimes |+\rangle\langle + + | + |-\rangle\langle - - |$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$B = B_1 \otimes B_2$$

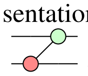


$$= \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

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

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

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

The resulting matrix R is proportional to the matrix representation of the CNOT gate. This means that the ZX-Diagram  is indeed a valid representation of the CNOT gate.

IV. Simplifcation Rules

In the previous section we saw how we can calculate the matrix-representation of a ZX-diagram. 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 tensor products. This is exactly 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 ZX-diagram. In this section we will look at a set of rewrite rules which allow us to simplify ZX-diagrams by using rewrite rules.

Note that all the rules can be applied in both directions. This means that we can use the rules to simplify a diagram, but we can also use them to make a diagram more complex. Additionally, we can apply the rules to any subgraph of the diagram, provided that the shape of the subgraph matches the shape of the rule. It is also important to know that all the rules also hold if we replace all the spiders with their adjoint (I.e.we swap the color of **all** spiders).

We also don't have to keep track which leg of the spider is the input and which one 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. It isn't actually a simplification rule, but it makes it way easier to remove clutter from the diagram. 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. This rule is shown in figure 9.

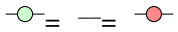


Fig. 9: Identity Removal Rule

B. Spider Fusion

The spider fusion rule allows us to fuse two spiders together if they are connected by an edge. The inputs and outputs of the two spiders are merged together. Additionally the phase of the two spiders is added together.

This rule is shown in figure 10.

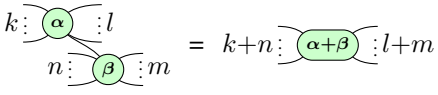


Fig. 10: Spider Fusion Rule

C. Hopf Rule

The Hopf rule allows us to split apart two spiders of opposite color, given that they are connected by two edges.

This rule is shown in figure 11.

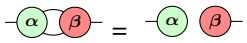


Fig. 11: Hopf Rule

D. Copy Rule

The copy rule allows to copy 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.

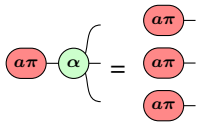


Fig. 12: Copy Rule

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, where it states that the copied XOR result, corresponds to first copying the inputs and then performing an XOR on the swapped, and copied inputs. This rule is shown in figure 13. It works by using the copy-rule from figure 12 and a XOR spider.

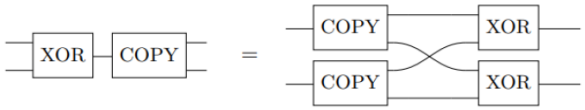


Fig. 13: Bi-Algebra Rule in Digital Logic

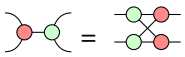


Fig. 14: Bi-Algebra Rule

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.

G. Color Change Rule

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

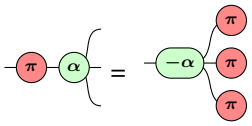


Fig. 15: π-Commutation Rule

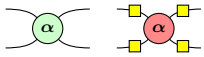


Fig. 16: Color Change Rule

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.

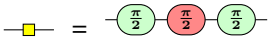


Fig. 17: Hadamard Gate

V. Simplifcation Example

A. GHZ Circuit

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

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

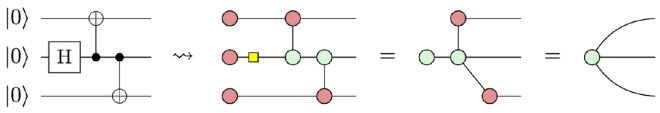


Fig. 18: GHZ Circuit

In order to achieve a simplification we will perform these steps:

1) In the first step we translated the gates from the circuit into their corresponding ZX-diagrams. For this we used the **CNOT** and **Hadamard** rules from the previous section.
2) Now we can start simplifying the diagram. We will start by applying the **Spider Fusion** rule to merge all free spiders together.
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 we perform the Spider Fusion rule again, and remove the identity spiders, we end up with the simplified diagram shown in rightmost diagram of figure 18. Using the formula for calculating the matrix-representation of single spider we calculate $|GHZ\rangle = 1\cdot|000\rangle + 1\cdot|111\rangle = |000\rangle + |111\rangle$. Which is proportional to the real GHZ state. Remember that in order to get the real GHZ state we have to normalize the state first, which we did not do here.

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.

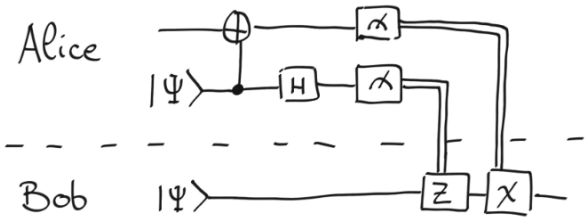


Fig. 19: Quantum Teleportation Circuit Img: Pennylane [12]

In order to understand the circuit we will first look at the representation of measurements in the ZX-Calculus. We will use the following rules to represent measurements in the ZX-Calculus. The measurements in the given basis are represented by parameterized spiders of the same basis [15]. The parameter of the spider are boolean values, which represent the measurement outcome. Based on the outcome of the measurement we can influence other parts of the circuit, by including the parameter of the measurement spider in the target spiders of the other gates.

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 (which has a matrix representation of $\begin{pmatrix}1 & 0 & 0 & 1\end{pmatrix}^T$, and therefore is proportional to the bell state $|\phi^+\rangle$.

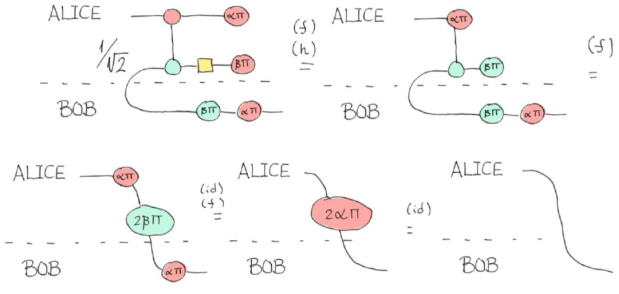


Fig. 20: Teleportation in the ZX-Calculus Img: Pennylane [12]

By applying the rewrite rules from above we can deduce that the whole circuit *cancels* itself out. This means that the circuit

is equivalent to the identity circuit between Alice and Bob. Therefore we showed that the teleportation works as expected.

VI. Universality & Completeness

As statet in [4] the ZX-Calculus is universal, because any linear map can be represented as a ZX-diagram. This means that any valid quantum circuit can be represented as a ZX-diagram. Additionally it can be shown that ZX-Calculus is complete for circuits which 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. Such a simplification process is shown in figure 21.

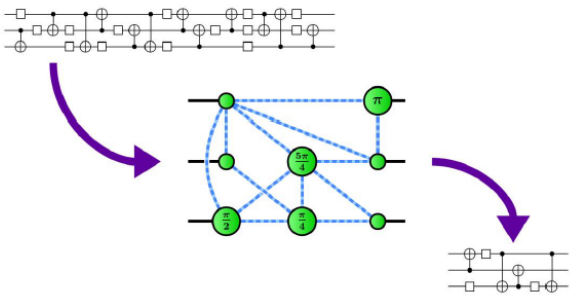


Fig. 21: Simplification process [5]

VII. Future Work

An important problem 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 want to be able to extract a classical gate representation of the circuit in order to simulate it on real hardware. This generally is not a trivial task, since ZX-Diagrams are a superset of the classical quantum circuits [4] as seen in figure 22.

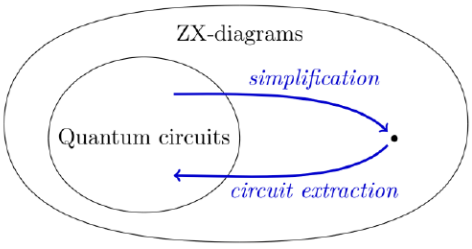


Fig. 22: Simplifcation process [4]

VIII. Conclusion

We have seen that ZX-calculus is a powerful tool for reasoning about quantum circuits. We have seen that it offers new ways to simplify and calculate quantum circuits by relying on a graphical language, which is very intuitive and easy to use. ZX-calculus is a very active field of research and that there are still many open questions left to answer.

One of the ongoing areas of research in ZX-calculus is the development of efficient extraction algorithm which are needed in order to gain back a classical gate representation of the circuit. Since ZX-Calculus is a very new field of research, there is still huge potential for new discoveries and applications.

References

[1] Dave Bacon. Cse 599d - quantum computing the no-cloning theorem, classical teleportation and quantum teleportation, superdense 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.

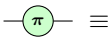
[13] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring, 1994.

[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.

APPENDIX

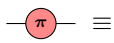
A. ZX-Representation of the Pauli Z-Gate



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

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

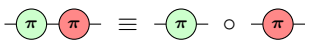
B. ZX-Representation of the Pauli X-Gate



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

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