Equivalence Checking of Quantum Circuits with the ZX-Calculus

Tom Peham* *Graduate Student Member, IEEE,*
Lukas Burgholzer† *Graduate Student Member, IEEE,* Robert Wille*‡ *Senior Member, IEEE*

*Chair for Design Automation, Technical University of Munich, Germany
†Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
‡Software Competence Center Hagenberg GmbH (SCCH), Hagenberg, Austria

tom.peham@tum.de lukas.burgholzer@jku.at robert.wille@tum.de

https://www.cda.cit.tum.de/research/quantum/

***Abstract*—As state-of-the-art quantum computers are capable of running increasingly complex algorithms, the need for automated methods to design and test potential applications rises. Equivalence checking of quantum circuits is an important, yet hardly automated, task in the development of the quantum software stack. Recently, new methods have been proposed that tackle this problem from widely different perspectives. One of them is based on the ZX-calculus, a graphical rewriting system for quantum computing. However, the power and capability of this equivalence checking method has barely been explored. The aim of this work is to evaluate the ZX-calculus as a tool for equivalence checking of quantum circuits. To this end, it is demonstrated how the ZX-calculus based approach for equivalence checking can be expanded in order to verify the results of compilation flows and optimizations on quantum circuits. It is also shown that the ZX-calculus based method is not complete—especially for quantum circuits with ancillary qubits. In order to properly evaluate the proposed method, we conduct a detailed case study by comparing it to two other state-of-the-art methods for equivalence checking: one based on path-sums and another based on decision diagrams. The proposed methods have been integrated into the publicly available QCEC tool (https://github.com/cda-tum/qcec) which is part of the Munich Quantum Toolkit (MQT).**

I. Introduction

Quantum computing [1] has had a surge in research endeavors by academia and industry in recent years. While quantum computers have not reached a stage of widespread practical usability yet, they promise to outperform classical computers in various important tasks, such as unstructured search, integer factorization, optimization problems, the simulation of molecules, and more [2]–[7]. To keep pace with the rapid developments in quantum hardware, various tools have been developed that help in designing corresponding applications.

Initially, a quantum computation is described as a sequence of (high-level) quantum gates—somewhat similar to a classical C program. However, just like assembly for a classical processor, the actual machine instructions that may be performed on a given quantum processor are generally restricted to a small (low-level) gate-set and might only allow interactions between specific pairs of qubits. Therefore, in order to execute a given circuit on quantum hardware, it needs to be *compiled* to a representation that adheres to all constraints imposed by the targeted device [8]–[11]. Since quantum computers are heavily affected by noise and decoherence, it is paramount to optimize circuits as much as possible in order to maximize the expected fidelity when running the circuit [12]–[16].

Since the compiled quantum circuit might be altered drastically from its original high-level description, it is of utmost importance that the circuit to be executed on the hardware still implements the same functionality as originally intended. Verification of compilation results or, more generally, *equivalence checking of quantum circuits*, turns out to be an extremely complex, even QMA-complete[1] [17], task and is in dire need of automation. Although various methods have been proposed [18]–[25] to tackle the equivalence checking problem from completely different perspectives, a baseline indicating which paradigm is suited best for which use case is yet to be established.

One method for equivalence checking of quantum circuits is based on the *ZX*-calculus [25]–[28], a graphical calculus used for reasoning about quantum computing. While some results on this equivalence checking method exist [28], it was introduced as more of a side-note in the original work than a fully-fledged method. At the time of writing, the only publicly available implementation of this algorithm is written in Python [29]. Therefore it is difficult to assess the performance of this method due to the inherently slower runtime of the Python interpreter when compared to a compiled language like C++. This makes it somewhat problematic to compare the method with other established equivalence checking algorithms. Furthermore, issues unique to design automation in quantum computing, like inaccurate representations of complex numbers, different logical-to-physical qubit mappings, and ancillary qubits, have not been addressed in the ZX-calculus framework. Apart from practical considerations, theoretical aspects have also hardly been investigated so far. It is not known for what class of circuits the equivalence checking method based on the ZX-calculus is complete, i.e., whether it can actually prove the equivalence of any two equivalent quantum circuits.

Motivated by that, the aim of this work is twofold. Firstly, to establish whether the ZX-calculus provides a solid equivalence checking methodology, we review the current state of the art in equivalence checking with the ZX-calculus. To expand on this, we discuss how this method can be augmented to

[1]QMA is the quantum computing analogue to NP. Indeed, NP is a subset of QMA

handle inaccurate representations of complex numbers arising from compilation and optimization processes, deal with alterations of the input and output layout of a circuit that happen during compilation, and integrate ancillary qubits into the equivalence checking procedure. We provide first results on the completeness of this equivalence checking algorithm. Secondly, in order to empirically show that the ZX-calculus is a practically relevant method in equivalence checking, we conduct a detailed case study[2] to establish a baseline for the current state of the art in equivalence checking of quantum circuits considering a large range of benchmarks. To this end, we re-implemented the ZX-calculus based equivalence checking algorithm in C++ and expanded it with the capabilities mentioned above. We compare this implementation—which has been integrated into the publicly available equivalence checking tool *QCEC* (https://github.com/cda-tum/qcec)—with two other state-of-the-art equivalence checking methods: one based on path-sums [22] and another based on quantum decision diagrams [24], [31]–[34].

Overall, we show that the ZX-calculus can be adapted to verify the results of compilation flows effectively, that it outperforms the path-sum approach by a constant factor, and that it performs on par with the decision diagram based method in many cases. However, both the ZX-calculus and the decision diagram based methods have domains where they clearly outperform the other. Since we show that the ZX-calculus based approach may fail to prove the equivalence of two equivalent quantum circuits, it cannot be used to prove non-equivalence of quantum circuits—only to give an indication of non-equivalence. All in all, we conclude that neither the ZX-calculus nor the decision diagram based method is clearly better than the other but that they rather serve as complementary methods that are best used in conjunction.

The remainder of this work is structured as follows: Section II provides the necessary background and motivates the necessity of equivalence checking routines in quantum computing. Then, Section III describes the equivalence checking problem in detail. Based on that, Section IV recapitulates the theory of the ZX-calculus, explains the state-of-the art equivalence checking algorithm based on the ZX-calculus in detail, and shows how the method can be expanded to handle more relevant equivalence checking problems in quantum circuit compilation and optimization. There we also prove that the ZX-calculus based equivalence checking method is complete for Clifford circuits and illustrate that it is not complete in general—failing to prove the equivalence of a simple example. In Section VII we compare ZX-calculus based equivalence checking with methods based on paths-sums and decision diagrams. Finally, Section VIII concludes this work.

II. Background

To keep this work as self-contained as possible, this section provides a brief introduction to the concepts of quantum computing and quantum circuits relevant for this work.

A. Quantum Computing

In classical computing, information is encoded in classical bits that can be either 0 or 1. Analogously, in quantum computing, *quantum bits* (or *qubits* in short) are used which can be either in the $|0\rangle$ or $|1\rangle$ state (in Dirac notation). Contrary to the classical domain, qubits can also be in *superposition* of multiple states. Formally, the state $|\phi\rangle$ of a qubit is written as

$$|\phi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle = \alpha_0 \begin{bmatrix}1\\0\end{bmatrix} + \alpha_1 \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}\alpha_0\\\alpha_1\end{bmatrix}$$

with *amplitudes* $\alpha_0, \alpha_1 \in \mathbb{C}, |\alpha_0|^2 + |\alpha_1|^2 = 1$.

The basis states of multi-qubit systems are obtained as the *tensor product* of single qubit states. So a basis state of a 3-qubit system would for example be written as $|1\rangle \otimes |1\rangle \otimes |0\rangle = |110\rangle =: |6\rangle$. In general, an n-qubit state $|\phi\rangle$ is described by a linear combination of basis vectors, i.e.,

$$\sum_{i=0}^{2^n-1} \alpha_i |i\rangle \text{ with } \sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1 \text{ and } \alpha_i \in \mathbb{C}.$$

Any operation manipulating the state of a quantum system must again yield a valid quantum state. As a consequence, any such operation U must be *unitary*, i.e., it must obey the equation $UU^\dagger = U^\dagger U = I$ where $U^\dagger$ is the *conjugate transpose* of U and I is the identity transformation.

Example 1. *Consider the* Hadamard transform $H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$. *It can be easily checked by matrix multiplication that H is a unitary transformation. The Hadamard transform maps* Z-basis *states to* X-basis *states, i.e.,*

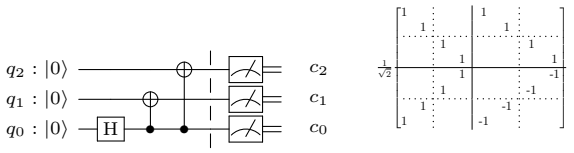
$$H|0\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle =: |+\rangle$$
$$H|1\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle =: |-\rangle.$$

An important unitary acting on two qubits is the controlled not *or CNOT gate. It is defined by the matrix* $\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}$ *and flips the second qubit (the* target*) when the first qubit (the* control*) is in state* $|1\rangle$.

A quantum computation is a unitary transformation acting on some initial state (usually the qubits are all prepared to be $|0\rangle$). Instead of writing the *system matrix* (i.e., the unitary describing the behavior of the whole circuit) explicitly, a common way to describe the *unitary evolution* of a quantum system is through *quantum circuit* notation [1]. There, qubits are represented by wires and operations (called *gates*) are annotated as boxes and circles on the wires. The evolution of the initial state is read from left to right. Thus, a quantum circuit G is described as a sequence of gates $g_0 \dots g_{m-1}$. Due to their unitary nature, quantum circuits are inherently reversible. More specifically, the inverse of a quantum circuit $G = g_0 \dots g_{m-1}$ is obtained by inverting each gate and reversing the order of operations, i.e., $G^\dagger = g_{m-1}^\dagger \dots g_0^\dagger$.

Example 2. *The circuit G in Fig. 1a represents a 3-qubit system. The box annotated with H is a Hadamard transform on qubit q_0 and the connected circles and dots are CNOT gates with q_0 as control and q_1 and q_2 as target qubit, respectively. The circuit maps $|000\rangle$ to $\frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|111\rangle$, the well-known GHZ state [35]. The system matrix describing the unitary this circuit realizes is given in Fig. 1b.*

[2] A preliminary version of this case study has been published in [30].



(a) GHZ state preparation circuit G (b) System matrix U of G

Fig. 1: GHZ state preparation

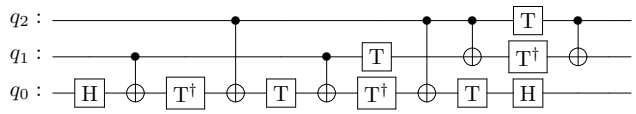


Fig. 2: Decomposition of the Toffoli gate in Clifford+T

B. Quantum Circuit Compilation

Quantum algorithms are typically designed at a rather high abstraction level without considering specific hardware restrictions. In order to execute a conceptual quantum algorithm on an actual device, it has to be *compiled* to a representation that conforms to all restrictions imposed by the targeted device. Since quantum computers typically only support a limited gate-set, every high-level operation has to be *decomposed* into that gate-set [36]–[38]. This can sometimes significantly increase the size of the circuit. Fig. 2 shows an exemplary decomposition of the Toffoli gate in the Clifford+T gate-set.

In addition, many architectures (such as those based on superconducting qubits) restrict the pairs of qubits that operations may be applied to. Hence, it is necessary to *map* the decomposed circuit to the device such that it adheres to the device's coupling constraints [39]–[41]. In general, this is accomplished by establishing a mapping between the circuit's logical qubits and the device's physical qubits. Since it is generally not possible to determine a conforming mapping in a static fashion, SWAP gates are inserted into the circuit that allow to dynamically change the logical-to-physical qubit mapping over the course of the compilation.

Example 3. *Consider again the GHZ preparation circuit shown in Fig. 1a and assume it shall be mapped to the* 5*-qubit, linear architecture shown on the left-hand side of Fig. 3. Assume that, initially, logical qubit q_i is mapped to physical qubit Q_i for $0 \leq i \leq 2$. Then, the first two operations can be directly applied, while the last operation cannot—due to the fact that Q_0 and Q_2 are not directly connected on the architecture. Hence, a SWAP operation between Q_2 and Q_1 is introduced, which allows to execute the final gate. At the end of the circuit q_0 is measured on Q_0, q_1 on Q_2 and q_2 on Q_1.*

C. Quantum Circuit Optimization

Through decomposition and mapping, even small quantum circuits can significantly increase in size. In classical computing, circuits are usually optimized in order to require less space, time, or energy. While time is still an important factor in quantum computing, this is because of a different reason. The *coherence time* of a quantum mechanical system is the time for which the system remains quantum-mechanically coherent [1].

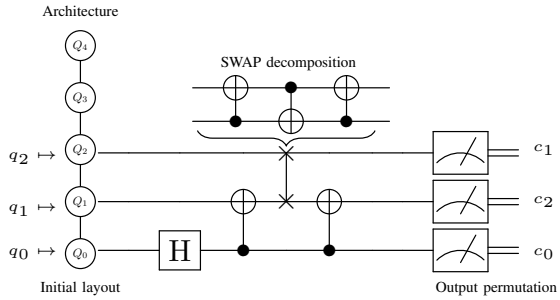
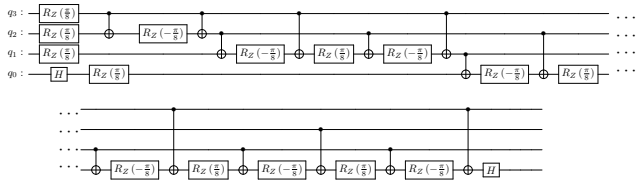
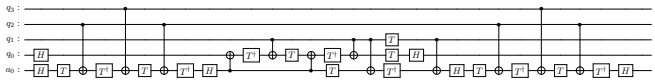


Fig. 3: Compilation of GHZ state preparation circuit



(a) Multi-controlled Toffoli without ancillary qubits



(b) Multi-controlled Toffoli gate with ancillary qubits

Fig. 4: Decompositions of the multi-controlled Toffoli gate

When this time is exceeded, the system collapses into some basis state and is therefore no longer in superposition and qubits are not entangled anymore. The decoherence time essentially puts a limit on the maximum number of operations that can be performed on a quantum system before it collapses.

Another factor unique to quantum computing is the *gate error rate*. Gates are difficult to realize precisely in practice. Every operation performed on qubits potentially introduces some error. This is often quantified using the *fidelity* $\mathcal{F}$, which measures the distance between two quantum states. The *gate fidelity* [1] is then a measure of the fidelity of the quantum state after applying a noisy gate compared to the quantum state if an ideal (noiseless) gate was applied. The higher the gate fidelity, the better the implementation of the gate. Some gates are easier to realize than others. On superconducting architectures, CNOT error is the dominating error factor. At the time of writing, for example, the single-qubit Pauli X error on the IBMQ Montreal quantum computer was 2.003×10^{-4} on the physical qubit $Q0$ compared to the CNOT error between qubit $Q0$ and $Q1$ of 2.276×10^{-3}.

Because of the coherence time and gate errors, it is important to optimize the number of elementary gate operations used in a quantum algorithm. It is not just a matter of execution time but of whether a meaningful result can be obtained at all from the execution of a quantum algorithm on a specific architecture. Many optimization schemes exist and optimization of quantum circuits is still an active area of research [14], [15], [42].

The simplest of these optimization schemes is the *single-qubit gate fusion*. Since any single qubit unitary represents a rotation of the Bloch sphere, any sequence of such gates also represents a rotation. Instead of performing each rotation separately, the complete rotation can be performed at once.

Another kind of optimization involves using *ancillary qubits*. Ancillary qubits are additional qubits apart from those required for the quantum algorithm. They can act as a sort of "working memory" to allow for a more compact representation of certain quantum gates or algorithms. Because the value of ancillary qubits is only important during the computation, they are not measured and do not factor into the final result.

Example 4. *Consider the quantum circuit in Fig. 4a which represents a multi-controlled Toffoli gate with three control qubits. This gate performs a Pauli X gate on the last qubit if and only if all the previous qubits are in the $|1\rangle$ state. Otherwise it acts as the identity on all qubits. Fig. 4b shows an implementation of this gate with the addition of one ancillary qubit. This is now a quantum circuit acting on five qubits with a higher total gate count, using 33 compared to the 31 of the original circuit. This circuit uses less CNOT gates however, needing only 12 CNOT operations compared to the 14 of the previous circuit. This trade-off is desirable because of the higher gate error of CNOT gates.*

Of course, optimizations are also employed in conjunction with compilation methods to optimize a compiled circuit while still obeying all restrictions enforced by the hardware. But sometimes optimizations are done before mapping. In fact, some available optimization methods are not designed to handle hardware-specific restrictions [28]. Therefore, equivalence checking methods are not only relevant when verifying the result of a compilation from a high-level description but also when verifying the results of optimizations of an uncompiled circuit or even just verifying the equivalence of a high-level description with an uncompiled optimized version of the high-level description.

Eventually, compilation and optimization yields a new circuit that might look quite different from the original high-level description. It is essential for the successful execution of a quantum computation to verify that the compiled circuit still implements the same functionality as the original one. To this end, methods to check the equivalence of quantum circuits are necessary.

III. Equivalence Checking

In order to discuss equivalence checking methods for quantum circuits we first need to precisely define the equivalence checking problem and aspects unique to equivalence checking in the quantum realm—namely permutations of the input and output layout of a quantum circuit, inaccuracies stemming from working with complex numbers, and ancillary qubits.

In general, given two quantum circuits

$$G = g_0 \dots g_{m-1} \text{ and } G' = g'_0 \dots g'_{m'-1}$$

with corresponding system matrices

$$U = U_{m-1} \cdots U_0 \text{ and } U' = U'_{m'-1} \cdots U'_0,$$

the *equivalence checking problem for quantum circuits* asks whether

$$U = e^{i\theta} U' \text{ or, equivalently, } U^\dagger U' = e^{i\theta} I,$$

where $\theta \in (-\pi, \pi]$ denotes a physically unobservable global phase.

So, in principle, checking the equivalence of two quantum circuits reduces to the construction and the comparison of the respective system matrices. While this is straightforward conceptually, it quickly becomes an increasingly difficult task due to the size of the involved matrices scaling exponentially with the number of qubits. Equivalence checking of quantum circuits has even been shown to be QMA-complete [17].

Even this definition is lacking when talking about the results of compilation flows. Compilation and optimization can alter a circuit in such a way that two circuits can be considered equal even if they have different system matrices.

Firstly, there are numerical inaccuracies. This is one of the biggest, yet hardly talked about, practical issues when actually conducting equivalence checking. Because quantum gates are described by matrices over $\mathbb{C}$, they are hard to accurately represent in memory. Usually, these matrices are stored using floating point numbers which leads to imprecisions and rounding errors. Therefore, comparing two matrices for exact equality becomes pointless in many practical cases. Instead, the Hilbert-Schmidt inner product can be used to quantify the similarity between two matrices. Let tr denote the trace of a matrix, i.e., the sum of its diagonal elements. Then, because $\mathrm{tr}(I) = 2^n$ for the identity transformation on n qubits, one can check whether $|\mathrm{tr}(U^\dagger U')| \approx 2^n$ in order to conclude the equivalence of both circuits up to a given tolerance.

Secondly, compilation flows introduce SWAP operations into a circuit such that the resulting circuit conforms to the hardware topology. These techniques use a circuit's initial layout and output permutation as an additional degree of freedom for saving SWAP operations, as, e.g., illustrated in Example 3. Because of these SWAPs, two circuits might only be equivalent up to a reordering of the qubits. Hence, in order to verify the equivalence of compilation flow results, any equivalence checking routine must be able to handle these kinds of permutations and accurately track which gate is performed on which qubit.

Lastly, as discussed in Section II-C, sometimes it is necessary to check the equivalence of two circuits that might not even operate on the same number of qubits due to the use of ancillary qubits. Ideally, an equivalence checking routine can also handle those cases. The difficulty comes from the fact that quantum circuits with differing numbers of qubits cannot represent the same unitary. Indeed, their unitaries do not even have the same dimensions. Ancillary qubits have a constant initial state, being either in the $|0\rangle$ or $|1\rangle$ state.

Given two quantum circuits

$$G = g_0 \dots g_{m-1} \text{ and } G' = g'_0 \dots g'_{m'-1}$$

with ancillaries qubits with corresponding system matrices

$$U = U_{m-1} \cdots U_0 \text{ and } U' = U'_{m'-1} \cdots U'_0,$$

the *equivalence checking problem for quantum circuits involving ancillary qubits* asks whether

$$U_{\text{anc}} = e^{i\theta} U'_{\text{anc}} \text{ or, equivalently, } U^\dagger_{\text{anc}} U'_{\text{anc}} = e^{i\theta} I,$$

where $\theta \in (-\pi, \pi]$ denotes a physically unobservable global phase and U_{anc} is the unitary obtained from U by fixing the ancillary qubits in G to their constant state.

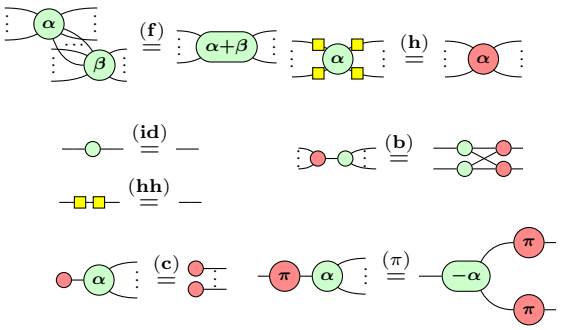


Fig. 5: Axioms of the scalar-free ZX-calculus

Example 5. *A CNOT gate can be trivially used as a Pauli X gate by treating the control qubit as an ancillary with constant state* $|1\rangle$*. To conform with our construction that the ancilla is the last qubit, we consider the matrix of a CNOT where the target is on qubit* 0*.*

$$U = \begin{bmatrix} 1&0&0&0\\0&0&0&1\\0&0&1&0\\0&1&0&0 \end{bmatrix} = |00\rangle\langle 00| + |01\rangle\langle 11| + |10\rangle\langle 10| + |11\rangle\langle 01|.$$

Fixing the last qubit gives:

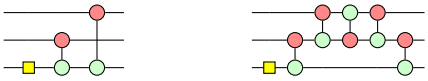
$$\begin{aligned} U_{anc} &= \langle 10| \begin{bmatrix} 1&0&0&0\\0&0&0&1\\0&0&1&0\\0&1&0&0 \end{bmatrix} |01\rangle|0\rangle\langle 0| + \langle 11| \begin{bmatrix} 1&0&0&0\\0&0&0&1\\0&0&1&0\\0&1&0&0 \end{bmatrix} |01\rangle|1\rangle\langle 0| \\ &+ \langle 10| \begin{bmatrix} 1&0&0&0\\0&0&0&1\\0&0&1&0\\0&1&0&0 \end{bmatrix} |11\rangle|0\rangle\langle 1| + \langle 11| \begin{bmatrix} 1&0&0&0\\0&0&0&1\\0&0&1&0\\0&1&0&0 \end{bmatrix} |11\rangle|1\rangle\langle 1| \\ &= |1\rangle\langle 0| + |0\rangle\langle 1| = \begin{bmatrix} 0&1\\1&0 \end{bmatrix} = X. \end{aligned}$$

This shows that U_{anc} *of a CNOT gate with constant control of* $|1\rangle$ *is equivalent to a Pauli* X *gate.*

In order to avoid the emergence of a verification gap as for classical systems, automated software solutions for equivalence checking of quantum circuits have to be developed. To this end, various methods have been proposed [18]–[25]. However, most of them either only work on small circuits, lack publicly available implementations or are based on paradigms established in classical computing that do not take the full picture of quantum computing into account. Few methods exist that approach equivalence checking entirely from the perspective of quantum computing [22], [23], [25]. Even these existing approaches view the equivalence checking problem from completely different perspectives and a baseline indicating which paradigm is suited best for which use case is yet to be established.

IV. ZX-Calculus

The ZX-calculus [26], [43] is a graphical notation for quantum circuits equipped with a powerful set of rewrite rules that enable diagrammatic reasoning about quantum computing. It has been successfully applied to quantum circuit compilation and optimization [25], [27], [28] and to some extent also to equivalence checking of quantum circuits [28]. The algorithm for equivalence checking using the ZX-calculus was only mentioned briefly as an alternative application of the optimization proposed in [28] and has not been adapted to handle numerical inaccuracies, permutations of input and output layout of a compiled circuit, and ancillary qubits as mentioned in Section III. In short, more work is needed to handle equivalence checking of compilation results with the ZX-calculus. In this section, we are going to discuss how this can be done as well as provide first results on the (in-)completeness of the ZX-calculus based equivalence checking algorithm.



(a) Uncompiled GHZ circuit (b) Compiled GHZ circuit

Fig. 6: ZX-diagrams of GHZ state preparation circuits

In order to do this, we are first going to introduce the necessary background on the ZX-calculus and the current state-of-the-art algorithm in equivalence checking with the ZX-calculus.

A. Basics of ZX-Calculus

A ZX-diagram is made up of colored nodes (called *spiders*) that are connected by wires (representing qubits, similar to quantum circuit notation). Each spider can either be green (Z-spider ○) or red (X-spider ●) and is optionally attributed a scalar phase.

ZX-diagrams can be composed just like quantum circuits. Horizontal composition or *concatenation* (denoted $\circ$) is achieved by connecting the outputs of one diagram to the input of another. The bare wire "—" acts as the identity for concatenation. Vertical composition (denoted $\otimes$) is achieved by simply "stacking" two diagrams on top of each other. The *empty diagram* acts as the identity for vertical composition. Additionally, a ZX-diagram can carry a global phase that is annotated along the diagram. Since global phases are negligible in most cases, they are frequently omitted from ZX-diagrams and equations in the ZX-calculus usually hold up to a global phase. A spider with a phase of $\pm\pi$ is called a *Pauli spider*. A spider with a phase $\alpha \in \{k\ \frac{\pi}{2} \mid k \in \mathbb{Z}\}$ is called a *Clifford spider*. A Clifford spider that is not a Pauli spider is called a *proper Clifford spider*. A ZX-diagram consisting entirely of Clifford (Pauli) spiders is called a *Clifford (Pauli) ZX-diagram*.

Any quantum circuit can be interpreted as a ZX-diagram. The reverse of this statement is not true, i.e., not every ZX-diagram can be interpreted as a quantum circuit because the ZX-diagram does not necessarily encode a unitary transformation. Every ZX-diagram does, however, have an interpretation as a linear map.

$$[\![\alpha]\!] = |0\ldots0\rangle\langle 0\ldots0| + e^{i\alpha}|1\ldots1\rangle\langle 1\ldots1|$$
$$[\![\alpha]\!] = |+\cdots+\rangle\langle+\cdots+| + e^{i\alpha}|-\cdots-\rangle\langle-\cdots-|$$

Here $[\![\cdot]\!]$ denotes the *interpretation function* which maps a ZX-diagram to its corresponding linear map. Any linear map on qubits can then be built up from Z- and X-spiders by connecting and stacking diagrams.

Spiders without inputs are called *states*, whereas spiders with no outputs are called *effects*. Interpreted as linear maps, states represent column vectors whereas effects represent row vectors. A ZX-diagram without inputs or outputs represents a number.

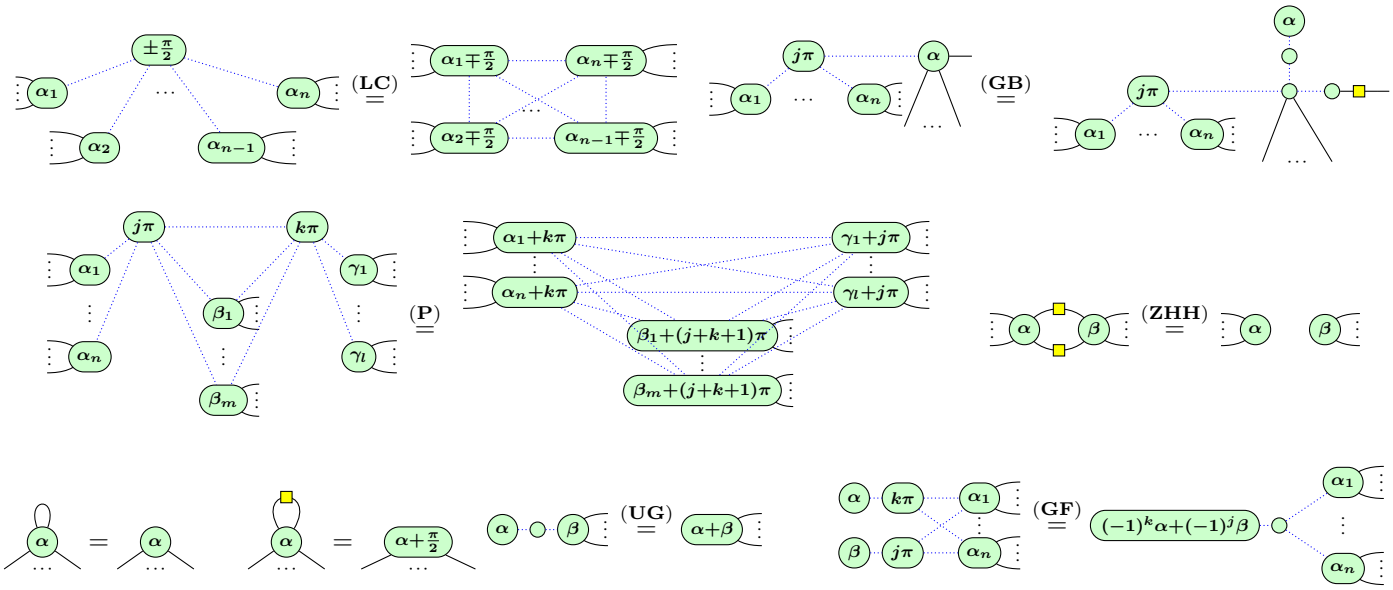
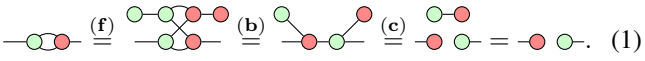


Fig. 7: Rewrite system for graph-like ZX-diagrams

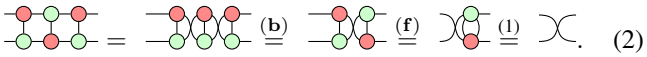
The real power of ZX-diagrams becomes evident when adding rewrite rules to the language. The axioms of the scalar-free ZX-calculus are given in Fig. 5.

Example 6. *To give a feel for how to work with ZX-diagrams, we are going to prove the well-known equivalence of a SWAP with 3 CNOT operations (as shown in Fig. 3). For this, we first need to prove the following rule, which is sometimes listed explicitly among the axioms for the ZX-calculus but can also be derived from the axioms as follows*



(1)

With this we can proceed with

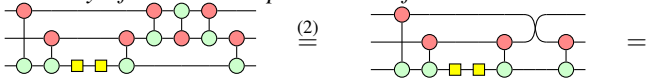


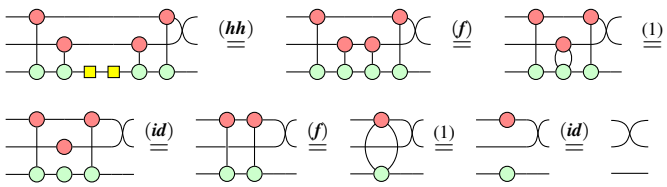
(2)

B. Equivalence Checking Graph-like ZX-diagrams

Equivalence checking with the ZX-calculus can be done in one of two ways, by either rewriting the diagram of both circuits into one another (as in Ex. 6) or by inverting one diagram, composing the diagrams, and simplifying as much as possible. This is sometimes called an *equivalence checking miter*. If the composed diagram simplifies to a diagram composed only of bare wires, it is either the identity or contains swaps, i.e., resembles a permutation.

Example 7. *Consider again the circuits G from Fig. 1a and G' from Fig. 3. Their respective ZX-diagrams are shown in Fig. 6a and Fig. 6b. Since all phases in all spiders are 0, the inverse of each diagram is obtained by just reversing the diagram. Using the rewrite rules of the ZX-calculus to prove the identity of the circuits proceeds as follows:*





The diagram contains a SWAP which permutes qubit Q_1 and Q_2. Since this is what we expect from the output permutation shown in Fig. 3 it can be concluded that the circuits are equivalent.

This example shows that the ZX-calculus cannot only show the equivalence of circuits but that it can also provide a proof certificate in the form of the order of rewrite rules that are applied to derive the identity.

The set of equations in Fig. 5 is not well suited for automated equivalence checking. In Eq. (1) we needed to use the spider fusion rule in both directions. This is undesirable in automated rewriting where *terminating* rewriting systems are preferable. In [27], the authors introduce an alternative structure for ZX-diagrams coupled with additional rewrite rules.

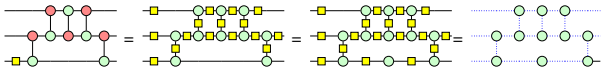
A ZX-diagram is *graph-like* when:

1) All spiders are Z-spiders.
2) Z-spiders are only connected via Hadamard edges.
3) There are no parallel Hadamard edges or self-loops.
4) Every input or output is connected to a Z-spider and every Z-spider is connected to at most one input or output.

In graph-like ZX-diagrams, a spider connected to an input or output is called a *boundary* spider. Otherwise, it is called an *interior* spider.

Most importantly, every ZX-diagram is equal to a graph-like ZX-diagram [27]. Every ZX-diagram can be rewritten to its equivalent graph-like form using the basic rules given in Fig. 5. Instead of rigorously defining this rewriting procedure we give an intuition with the following example.

Example 8. *The ZX-diagram of the GHZ state preparation circuit from Fig. 6a can easily be transformed to its equivalent graph-like form by applying rules* (**id**) *and* (**h**).



Graph-like diagrams allow for the formulation of a normalizing rewrite system. The rules of this system are given in Fig. 7. Adding rules (**f**), (**id**) and (**hh**) (applied from left to right) to this system allows for the definition of a simplification algorithm that reduces every ZX-diagram into a *reduced gadget form* [28].

This automated rewriting of diagrams into reduced gadget form allows for the definition of an equivalence checking algorithm. Given two quantum circuits G and G' we can check them for equivalence by taking their respective representations as ZX-diagrams D and D', combining them to $D^\dagger D'$ and simplifying the combined diagram to reduced gadget form. If the reduced gadget form is the identity diagram—the ZX-diagram consisting only of bare wires—then G and G' are equivalent. Otherwise, nothing can be concluded about the relation of G and $G^\dagger$ because there are generally multiple reduced gadget forms for a ZX-diagram.

V. Equivalence Checking Compilation Flows Using the ZX-Calculus

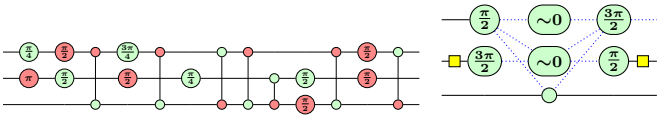
The original ZX-calculus equivalence checking algorithm proposed in [26] has been introduced as a byproduct of the optimization algorithm proposed in that work. It has, therefore, not been expanded to handle the more technical aspects of quantum circuit equivalence checking necessary to check the results of compilation flows. In the following, we will remedy this by showing how inaccuracies, permutations, and ancillae can be handled in ZX-calculus equivalence checking.

A. Handling Inaccuracies

The equivalence checking routine based on the ZX-calculus is an exact method. Hence, when considering two quantum circuits G and G', where G' is equivalent to G up to some small error, the ZX-calculus is unable to conclude equivalence. But can anything be concluded about the reduced gadget form of $D^\dagger D'$ where $[\![D]\!] \approx [\![D']\!]$?

To give an intuition, consider the Clifford+T circuit given in Fig. 8a. Introducing an error of 10^{-15} in the phase of two spiders and checking the equivalence of the original and the erroneous circuit yields the ZX-diagram shown in Fig. 8b. The phases indicated with $\sim \alpha$ means that the phase is $\alpha \pm 10^{-15}$. It is not at all obvious that this diagram is close to the identity. However, if we were to round the phases and simplified further, we would indeed be able to derive the identity.

One strategy is to interleave simplification to normal form with detection and rounding of phases close to $k\frac{\pi}{2}$ for some $k \in \mathbb{Z}$. This allows equivalence checking of quantum circuits that differ by small numerical inaccuracies in some continuous parameters. The corresponding algorithm is obviously not correct in a formal sense, i.e., it can attest two non-equivalent circuits to be equivalent, but that is the whole point. The threshold for rounding ϵ can be used to tune the degree to which errors are allowed. However, it does not give any indication about the absolute error.



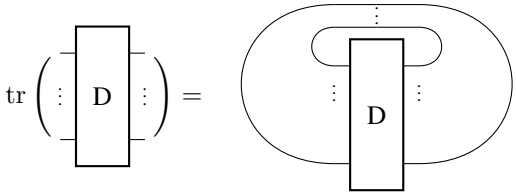
(a) ZX-diagram of Clifford+T circuit (b) Reduced diagram

Fig. 8: Equivalence checking with the ZX-calculus in the presence of few small errors

To clarify this point, consider a ZX-diagram M after simplification and the corresponding ZX-diagram M' obtained after rounding and simplifying again. The tolerance ϵ can not be used to assess $\mathrm{tr}([\![D^\dagger]\!][\![D']\!])$—the Hilbert-Schmidt inner product discussed in Section III.

A question one might ask is why the rounding doesn't already occur on the diagrams D and D'. The reason is that even phases that are not nice fractions of π (or very small fractions) might cancel during simplification due to the rules **UG** and **GF**. Thus rounding before simplifying would increase the total error made during the equivalence check.

This way of handling inaccuracies is still lacking. As discussed above it is hard to gauge the tolerance required in order to ensure the absolute error allowed is within some bound. Given the ZX-diagram M after full simplification, how can we determine whether $|\,\mathrm{tr}([\![M]\!])| \approx 2^n$? The diagrammatic trace is defined as follows:



Unfortunately, this definition is hardly helpful if we want to actually compute the trace. In order to compute the trace, further simplifications have to be made after the inputs and outputs have been connected. This doesn't necessarily enable the ZX-diagram to be simplified to a point where calculations are practical. A possible solution to this problem is to leave the ZX-calculus framework entirely. ZX-diagrams are, in essence, tensor networks [43], [44]. Therefore methods from the tensor network domain can be used to compute the trace of a ZX-diagram.

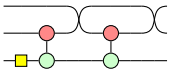
B. Handling Permutations

Handling SWAPs in ZX-diagrams is a trivial matter. Since SWAPs are nothing but edges connecting spiders acting on different qubits, they do not add much complexity to a ZX-diagram. To correct permutations of the initial layout, it has to be ensured that wires are connected accordingly when constructing $D^\dagger \circ D'$. But since SWAPs incur such little overhead in ZX-diagrams, the initial layout can also just be encoded into the original diagrams themselves before performing the equivalence check. The wires can then be connected in the usual fashion, i.e. by connecting the i-th output wire of $D^\dagger$ with the i-th input wire of D'.

Output permutations can be handled in a similar fashion as with decision diagrams, by comparing the permutation of wires after fully simplifying $D^\dagger \circ D'$ with the expected permutation. Once again the permutation can also just be handled by encoding the SWAPs directly into the diagrams.

As discussed in Section II-B, permutations of input and output layouts are performed in order to save CNOT gates that need to be executed on the quantum hardware. When converting a compiled circuit to a ZX-diagram, Eq. (2) can be used to reconstruct compiled SWAP gates if they are not optimized away. Since a SWAP in the ZX-calculus is only a crossing of the wires, this reconstruction can greatly improve the performance of the equivalence check, decreasing runtime by up to two orders of magnitude.

Example 9. *The output permutation of the qubits in Fig. 6b can be directly encoded back into the circuit via a SWAP at the end of the circuit. Additionally, the* 3 *CNOTs can be converted back into a SWAP, yielding the following ZX-diagram:*

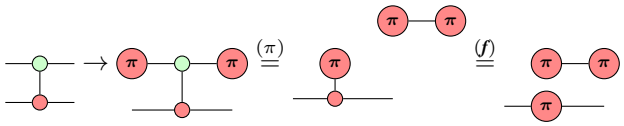


This ZX-diagram is equivalent to the one in Fig. 6a up to an untangling of the wires.

C. Handling Ancillaries

In the ZX-calculus the equivalence checking problem using ancillaries is reducible to the ancilla-free case in a straightforward fashion. Remember that ancillaries are qubits that have a constant initial state and end in the same state. This is easily translated into the diagrammatic language of the ZX-calculus, by replacing each input and output belonging to an ancilla qubit by the respective state and effect, which are just X-spiders with a phase of either 0 or π.

Example 10. *In Example 5 it was shown how fixing the control qubit of a CNOT gate to the $|1\rangle$ state, transforms it into a Pauli X gate. This can also be shown in the ZX-calculus with only basic applications of the axioms. We start by applying the $|1\rangle$ state and effect to the ancillary line and proceed to simplify.*



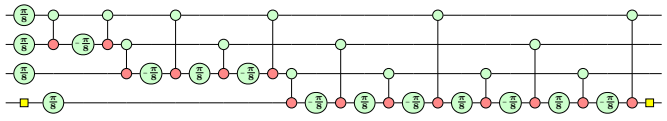
Since we ignore scalars, the right-hand side indeed implements a Pauli X gate (*).*

VI. Completeness

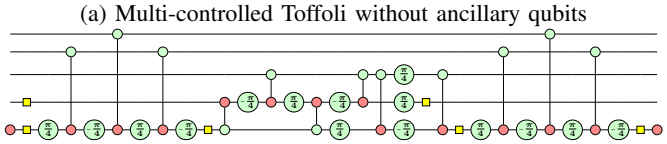
A natural question to ask is whether the ZX-calculus is powerful enough to derive the identity for any pair of functionally equivalent circuits. The good news is that the ruleset provided in this paper is complete for circuits solely composed of Clifford gates [45]. The bad news is that, in order to achieve completeness for universal quantum computing, the ruleset has to be extended with a rule involving complicated iterated trigonometric functions [46], which makes it difficult to apply in automated reasoning.

The question of completeness arises naturally in the context of rewriting. Given two circuits G and G', can we prove their (non-)equivalence using the ZX-calculus rewriting strategy?

As the ZX-calculus is complete for Clifford ZX-diagrams, it is not surprising that automated equivalence checking with the ZX-calculus is also complete for Clifford ZX-diagrams.

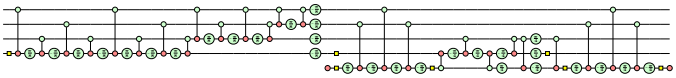


(a) Multi-controlled Toffoli without ancillary qubits

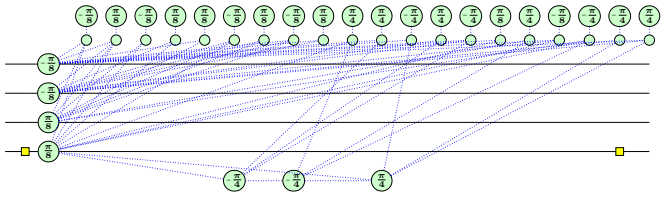


(b) Multi-controlled Toffoli with ancillary qubits

Fig. 9: ZX-diagrams of the multi-controlled Toffoli gate



(a) Miter of multi-controlled Toffoli ZX-diagrams

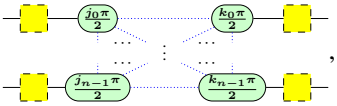


(b) Reduced miter

Fig. 10: Counterexample to completeness

Theorem 1. *Given two quantum Circuits G and G' consisting only of Clifford gates with corresponding ZX-diagrams D and D' the only reduced gadget form of $D^\dagger D'$ is the identity diagram.*

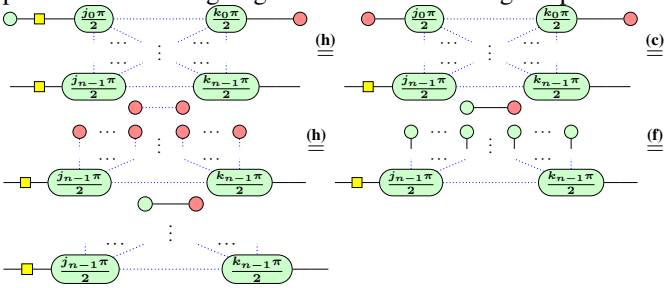
Proof. The rules **LC** and **P** remove every interior Clifford spider from a ZX-diagram. Since all spiders in $D^\dagger D'$ are Clifford, there are no more interior spiders left after simplifying. After simplification $D^\dagger D'$ must therefore be of the form



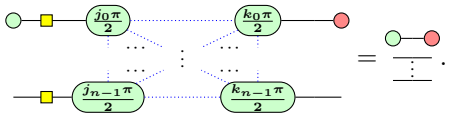
,

i.e. a Clifford ZX-diagram with only boundary spiders, where all boundary spiders (that are in fact connected) are connected via Hadamard edges and where there must be exactly one Hadamard box on each line. In fact this structure is independent of the fact that $[\![D^\dagger D']\!] = I_n$—every Clifford ZX-Diagram has this reduced gadget form. W.l.o.g. assume that the Hadamard boxes are all on the left-hand side and call this diagram D_{simp}

We need to show $j_i = k_i = 0$ for all $0 \leq i < n$ and that there are no connections between spiders on different lines. To break this problem into simpler subproblems we are going to use the following helpful trick:

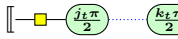
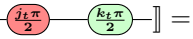


The spider fusion in the last equality is due to the fact that all spiders in the second row have a phase of 0 and all the spiders they are connected to are Z-spiders (all spiders in the diagram are Z-spiders). With this trick, we can effectively eliminate a line from the diagram because the diagram manipulations only effect the connections from the first row and not the connections between other rows. The rest of the diagram still has to act as the identity on the rest of the qubits. This is due to the fact that if $[\![D_{\text{simp}}]\!] = [\![\overline{\vdots}]\!]$ then



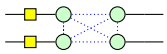
If two ZX-diagrams have the same interpretation they can be replaced with each other in every context. This is a simple consequence of the soundness of the ZX-calculus.

With the introduced trick, we can effectively reduce D_{simp} until only one line remains. In this case the remaining line  still has to represent the identity. We prove that $j_t = k_t = 0$ by concrete calculation of the matrix of the diagram.

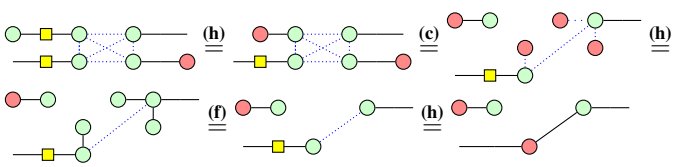
$$[\![\ \text{  }\ ]\!] \overset{\textbf{(h)}}{=} [\![\ \text{  }\ ]\!] =$$
$$(|+\rangle\langle+| + e^{ij_t\frac{\pi}{2}}|-\rangle\langle-|)(|0\rangle\langle 0| + e^{ik_t\frac{\pi}{2}}|1\rangle\langle 1|) =$$
$$\frac{1}{2}\begin{bmatrix} 1+e^{ij_t\frac{\pi}{2}} & (1-e^{ij_t\frac{\pi}{2}})e^{ik_t\frac{\pi}{2}} \\ 1-e^{ij_t\frac{\pi}{2}} & (1+e^{ij_t\frac{\pi}{2}})e^{ik_t\frac{\pi}{2}} \end{bmatrix}$$

Since this matrix has to equal $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ the constraints force $e^{ij_t\frac{\pi}{2}} = e^{ik_t\frac{\pi}{2}} = 1$ which can only be true if $j_t = k_t = 0$. Thus we can conclude that all spiders in D_{simp} have a phase of 0.

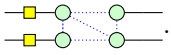
To show that no spiders belonging to different lines can be connected, we use our trick again. But this time we reduce down to two lines.



The Hadamard edges between spiders on different lines may or may not exist. We are going to see that for the purpose of this proof we do not need to make a case distinction on all possible combinations of connections. Using a similar strategy as for the line removal trick we obtain

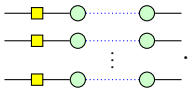


Since this diagram has to be the identity on each line and we input  in the first line and  on the second line, they also have to be the output on their respective lines. But this can only be the case if the diagonal connection between the first and second line does not exist. Therefore there is also no connection in the original diagram, i.e. it has to look like

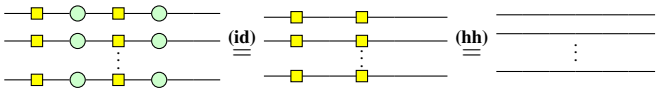


Similar reasoning can be applied to conclude that the remaining inter-line connections cannot exist if the diagram is equal to the identity diagram.

We have proven that all phases in D_{simp} are 0 and that there are no connections between spiders on different lines. Therefore D_{simp} looks like



Rules $(\mathbf{hh})$, $(\mathbf{f})$ and $(\mathbf{id})$ remove identity spiders, fuse spiders and cancel adjacent Hadamard boxes as much as possible. Thus the diagram is further simplified. Then,



Due to symmetry the proof still works if some of the Hadamard boxes are at the outputs. □

Theorem 1 establishes a baseline for what equivalences can be proven via automated reasoning with the ZX-calculus. Next, we want to look at completeness from a different perspective, by showing that rewriting to reduced gadget form is not sufficient for proving the equivalence of arbitrary equivalent circuits. In particular, we are going to show that this algorithm is not even sufficient for proving equivalence of reversible circuits.

Theorem 2. *There exist two equivalent quantum Circuits G and G'—using ancillary qubits—with corresponding ZX-diagrams D and D' where $D^\dagger D'$ possesses a reduced gadget form that is not the identity diagram.*

Proof. Unfortunately the proof of this theorem is not achieved through cunning manipulation of diagrams, kets and bras but by brute-force calculation. Consider the ZX-diagrams in Fig. 9 which are the ZX-diagrams of the circuits in Fig. 4 where the ancillary qubit's input and output has been set to $|0\rangle =$  . Taking the adjoint of the diagram in Fig. 9a, and concatenating the two diagrams, yields the diagram in Fig. 10a. A reduced gadget form of this diagrams is shown in Fig. 10b. No further simplifications can be made but this diagram is clearly not the identity. It can be checked by (tedious) computation of the corresponding matrices that the ZX-diagram in Fig. 10a does actually implement the identity transformation. □

It is not surprising at all that equivalence checking via simplification to reduced gadget form is not complete in general. Because the equivalence checking problem is QMA-complete and since NP is a subset of QMA, it would be entirely unexpected that the ZX-calculus based equivalence checking algorithm solves the equivalence checking problem, given that a reduced gadget form can be derived in polynomial time with respect to the number of spiders of the original diagram [28]. But the proof by counterexample shows that it cannot even show the equivalence of two circuits (involving ancillaries) even when they are fairly simple.

VII. Case Study

The basic equivalence checking routine based on the ZX-calculus is publicly available via the Python library pyzx [29]. Since pyzx does not support layout permutations, inaccuracies, or ancillary qubits, and because Python is inherently slower than a compiled programming language, the ZX-calculus based equivalence checking algorithm has been re-implemented in C++ and integrated into the publicly available QCEC tool (https://github.com/cda-tum/qcec) which is part of the Munich Quantum Toolkit (MQT, formerly known as JKQ [47]). This re-implementation has additional features that allow for handling of the mentioned problems.

To properly evaluate the resulting implementation of the ZX-calculus based equivalence checking algorithm, two state-of-the-art equivalence checking tools have been considered as a comparison: First, the proposed ZX-calculus equivalence checker is compared against an approach based on path-sums [22] on a large set of random Clifford circuits. This is done in order to assess how the proposed checker—which we proved to be complete for Clifford circuits—performs in relation to another Clifford-complete method.

Second, an extensive comparison is performed against the complete equivalence checking approach based on decision diagrams proposed in [23] to see how the incomplete ZX-calculus checker compares on a broad range of quantum circuits.

The path-sum equivalence checker is publicly available via the *Feynver* tool which is part of the Feynman toolset. The decision diagram based equivalence checker is also publicly available via *QCEC*. For the remainder of this section "QCEC" explicitly refers to the decision diagram based equivalence checker proposed in [23] and implemented in QCEC.

Before the experimental setup and the results are discussed, we will briefly introduce the basics of the two other equivalence checking methods considered for the evaluation.

A. Equivalence Checking Using Decision Diagrams

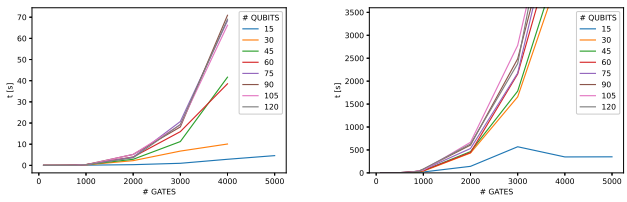
Decision Diagrams [24], [31]–[34] are a data structure used for efficiently representing complex matrices. Using *redundancies* in the representation of a matrix, decision diagrams can often represent an exponentially large matrix using only polynomial resources. This makes them great candidates for use in equivalence checking of quantum circuits, as a quantum circuit is just another way of writing a unitary matrix.

Recall that two quantum circuits $G = g_0 \cdots g_m$ and $G' = g'_0 \cdots g'_{m'}$ are equivalent if $G^\dagger G' = g_m^\dagger \cdots g_0^\dagger g'_0 \cdots g'_{m'} = I$. Similar to equivalence checking using ZX-diagrams, decision diagrams can be used to efficiently carry out the matrix multiplication of $G^\dagger$ and G'. The idea is to start constructing the functionality of the combined circuit from the "middle" and alternating between applications of $G^\dagger$ and G', such that the decision diagram being constructed remains as close to the identity as possible [23]. This is desirable because the n-qubit identity matrix only requires linear space when represented as a decision diagram instead of $2^n \times 2^n$ complex numbers for the entire matrix.

B. Equivalence Checking Using Path-Sums

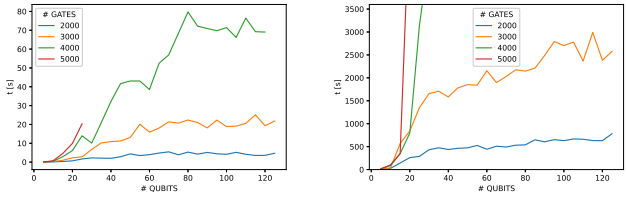
Path-sums [22] are an abstract representation of quantum circuits in the form of multivariate polynomials over Boolean variables. Similar to Feynman path integrals, the idea of path-sums is to encode the action of a unitary as a sum over all possible input-output basis states of a quantum computation. This symbolic representation of a quantum circuit allows for handling Clifford + $R_Z(\frac{\pi}{2^k})$ $k \in \mathbb{Z}$, circuits on a high abstraction level.

The symbolic treatment of quantum circuits as path-sums allows for the formulation of a set of rewrite rules which—similar to ZX-calculus rewriting—can be used to successively reduce a path-sum into a normal form in polynomial time. While this approach is not complete in general, it is complete for Clifford circuits [22]. Thus path-sum rewriting serves as an alternative complete approach to proving equivalence of Clifford circuits.



ZX-calculus checker Feynver

(a) Runtimes for differing gate count



ZX-calculus checker Feynver

(b) Runtimes for differing qubit count

Fig. 11: Equivalence checking random Clifford benchmarks

C. Experimental Setup

While there is no explicit configuration for the ZX-calculus and path-sum equivalence checker, QCEC has different methods with their respective parameters based on [23], [48], [49]. For the evaluations involving decision diagrams, we compare the ZX-calculus based equivalence checking routine with the combined approach as presented in [23]. For QCEC, we run the equivalence checking routine in parallel with a sequence of 16 simulation runs. If the simulations manage to prove non-equivalence of the circuits, the equivalence checking routine is terminated early.

In order to compare the methods, various benchmarks have been considered. All benchmarks are provided in the form of QASM [50] files, which serves as a common language for the ZX-calculus tool and QCEC. All circuits have been compiled using *qiskit-terra* 0.18.3, either with the optimization level $O1$ or $O2$ depending on the benchmark set. Before checking the Clifford Circuits with Feynver, the circuits had to be translated to a format supported by Feynver, but since only Clifford circuits were considered, this was a trivial matter.

To compare the scaling of the ZX-checker and Feynver, a large set of random Clifford circuits with a varying number of

TABLE I: Slightly optimized reversible circuits

Benchmark				Equivalent		1 Gate Missing		Flipped CNOT	
Name	n	$\|G\|$	$\|G'\|$	t_{zx}[s]	t_{qcec}[s]	t_{zx}[s]	t_{qcec}[s]	t_{zx}[s]	t_{qcec}[s]
9symml-195	20	15701	13662	8.04	7.37	2.04	0.12	8.25	0.11
dist-223	20	22778	19458	11.82	11.88	24.65	0.16	9.13	0.17
clip-206	53	23782	20872	12.53	20.49	2.18	0.29	8.92	0.31
hwb7-61	20	14680	12557	13.45	1.56	0.56	0.11	13.68	0.10
life-238	20	12114	10441	13.55	3.78	1.57	0.10	9.92	0.09
hwb7-60	20	14096	12167	19.24	0.83	1.14	0.10	51.74	0.11
alu2-199	53	21474	19007	19.76	28.05	2.04	0.26	19.16	0.29
sym9-193	20	15944	13327	21.70	6.17	1.89	0.11	24.64	0.11
example2-231	53	22017	19411	23.63	18.36	5.12	0.27	25.64	0.25
hwb7-59	20	18435	15616	40.81	2.53	0.75	0.18	20.55	0.15
sym9-148	20	17061	14044	75.53	0.60	6.59	0.11	34.26	0.11
urf2-277	20	33348	30940	76.59	3.36	2.03	0.26	77.75	0.26
add6-196	53	30296	25460	82.48	14.94	3.13	0.38	95.46	0.34
urf2-153	20	55243	47155	214.21	210.77	6.54	0.35	162.52	0.39
hwb8-117	20	23596	20697	259.78	1.54	43.95	0.22	27.28	0.17
hwb8-116	20	23353	20996	274.82	2.68	18.58	0.15	272.41	0.16
urf2-161	20	100189	92597	403.99	5.08	5.48	0.69	25.83	1.32
hwb8-114	20	45079	38733	453.13	38.64	16.01	0.31	40.13	0.36
hwb8-118	20	52692	45784	521.59	63.61	85.34	0.43	617.56	0.39
urf2-154	20	52432	44615	587.01	145.44	7.97	0.54	16.57	0.49
hwb8-115	20	45247	39172	734.52	30.47	9.78	0.28	467.74	0.28
hwb8-113	20	52905	46230	1211.62	68.62	22.03	0.46	1476.73	0.37
urf5-159	20	65250	57099	1325.16	45.17	25.78	0.39	659.13	0.46
plus63mod4096-163	53	94520	82195	>3600	87.94	48.61	1.28	>3600	0.94
plus63mod8192-164	53	125612	111720	>3600	412.22	>3600	1.22	78.41	1.60
urf3-279	20	172651	157462	>3600	416.96	158.13	1.38	>3600	1.50
urf1-150	20	156575	134216	>3600	>3600	51.37	1.06	3084.44	1.34
urf6-281	20	98462	94097	>3600	>3600	604.79	1.30	>3600	1.28

TABLE II: Highly optimized reversible circuits

Benchmark				Equivalent		1 Gate Missing		Flipped CNOT	
Name	n	$\|G\|$	$\|G'\|$	t_{zx}[s]	t_{qcec}[s]	t_{zx}[s]	t_{qcec}[s]	t_{zx}[s]	t_{qcec}[s]
9symml-195	20	15701	13159	42.82	5.41	1.96	0.13	2.39	0.13
dist-223	20	22778	18721	16.44	11.96	3.45	0.16	2.49	0.17
clip-206	53	23782	21407	11.14	15.65	2.00	0.43	1.37	0.39
hwb7-61	20	14680	12741	34.99	0.52	0.47	0.18	0.46	0.17
life-238	20	12114	9869	8.73	2.80	1.12	0.10	0.69	0.10
alu2-199	53	21474	19080	26.23	4.27	7.00	0.31	6.23	0.36
sym9-193	20	15944	13908	7.76	11.21	1.41	0.14	1.07	0.13
example2-231	53	22017	19207	38.71	51.12	3.13	0.31	3.40	0.31
hwb7-59	20	18435	16027	15.68	1.24	0.46	0.15	0.50	0.16
sym9-148	20	17061	14002	284.87	0.96	1.07	0.11	1.08	0.11
urf2-277	20	33348	29711	68.28	16.10	2.95	0.32	2.68	0.31
add6-196	53	30296	26201	154.88	30.21	10.77	0.45	11.97	0.43
urf2-153	20	55243	46103	296.04	42.08	3.12	0.48	3.18	0.43
hwb8-117	20	23596	20413	123.04	17.21	2.56	0.20	3.03	0.20
hwb8-116	20	23353	20519	323.72	13.97	11.78	0.17	10.28	0.18
urf2-161	20	100189	91018	405.14	122.92	3.11	1.15	3.16	1.33
hwb8-114	20	45079	38398	1263.37	9.11	10.15	0.33	10.63	0.35
hwb8-118	20	52692	44901	712.38	13.65	8.40	0.68	5.93	0.41
urf2-154	20	52432	44306	1150.70	20.02	5.53	0.57	5.54	0.52
hwb8-115	20	45247	38205	269.99	13.15	6.26	0.40	8.26	0.32
hwb8-113	20	52905	45833	1036.07	11.59	15.25	0.41	16.68	0.48
urf5-159	20	65250	56089	2369.23	18.12	21.70	0.52	30.52	0.47
plus63mod4096-163	53	94520	83835	>3600	1249.90	78.66	1.27	73.99	1.12
plus63mod8192-164	53	125612	111258	>3600	>3600	313.11	1.49	300.36	1.61
urf3-279	20	172651	153474	>3600	>3600	40.35	1.40	40.53	1.75
urf1-150	20	156575	132898	>3600	>3600	103.27	1.17	105.97	1.16
urf6-281	20	98462	88750	>3600	>3600	198.70	1.36	187.00	1.32

qubits and gate counts has been generated and verified using both tools. Each of these circuits has been optimized with $O2$. The resulting runtimes can be seen in Fig. 11.

The remainder of the comparison was done against QCEC. QCEC has been previously evaluated on a benchmark set of reversible circuits (from [51]) which are mapped to suitable quantum architectures. We also use these in our evaluation as well as a selection of common quantum circuits that are available as part of the *MQT Bench* benchmark set [52]. For each benchmark, we consider three configurations. First, two circuits that are indeed equivalent are used as input. Then, two instances are created where errors are injected into one of the circuits—one with a random gate removed and one where the control and target of one CNOT gate have been swapped.

The benchmark set of reversible circuits is compiled to the to the 65-qubit IBM Manhattan architecture using optimization level $O0$ (no optimizations), $O1$ (slight optimizations) and $O2$ (advanced optimizations). The circuits compiled to $O1$ and $O2$ were checked against the unoptimized compiled circuit.

For the quantum circuits, we distinguish two use cases: The first is concerned with verifying the compilation result of a high-level circuit. To this end, the circuits are compiled to the 65-qubit IBM Manhattan architecture with a gate-set comprised of arbitrary single qubit rotations and the CNOT gate. The second use case is about verifying the equivalence of two different implementations of the same functionality—an original circuit and an optimized version ($O2$).

In the following, we summarize the results of our evaluations by means of a representative subset of benchmarks. The results for the reversible benchmarks are shown in Table I and Table II. The results for the quantum benchmarks are shown in Table III.

For further analysis of the influence of the size of the circuits on the runtimes of the equivalence checking routines, we consider a set of random quantum circuits comprised of CNOT, Hadamard and T gates with specific numbers of qubits and gates. Every gate in this set of benchmarks has a 20% chance of being a Hadamard gate and a 20% chance of being a T gate. For this benchmark set only equivalent instances have been considered. Every circuit has been checked twice, once against a slightly optimized version ($O1$) and once against a highly optimized version ($O2$).The resulting runtimes can be seen in Fig. 12a and Fig. 12b.

All computations were conducted on a 4.2 GHz Intel i7-7700K machine running Ubuntu 18.04 and 32 GiB main memory. Each benchmark was run with a hard timeout of 1 h for each method.

D. Discussion

Fig. 11 shows runtimes for the proposed ZX-calculus checker and Feynver for the set of random Clifford benchmarks. Fig. 11a shows runtimes with respect to the number of gates in the original circuit for fixed numbers of qubits and Fig. 11a shows runtimes with respect to the number of qubits for fixed numbers of gates. The similarity between the plots for both methods suggests that both methods scale somewhat similarly with respect to the size of the circuits, however. Indeed, the two methods exhibit the same asymptotic behavior. However, considering the scaling of the vertical axis in the plots, one can clearly see that the proposed implementation outperforms Feynver by orders of magnitude on all benchmarks.

In the comparison with QCEC, both methods managed to prove the correct result for all considered circuits where a result is obtained within the given time frame. As discussed before, this is not guaranteed by the theory of the ZX-calculus. On the other hand, the question of completeness for the decision diagram based approach is trivial. Decision diagrams are a canonical representation of a matrix. Thus, if the combined circuit $G^\dagger G'$ has the identity system matrix, the decision diagram for $G^\dagger G'$ has to be the identity decision diagram as well.

For the set of reversible benchmarks (Table I and Table II), the two methods finished within 10 s of each other for 92 % of benchmark instances in the case of equivalent instances for both optimization levels. The remaining reversible benchmarks and circuits containing large reversible parts in their high-level description (such as Grover's algorithm and the Quantum Random Walk) favor the decision diagram based approach. These circuits can be *exactly* compiled to polynomially-sized quantum circuits comprised only of Clifford+T gates, i.e., circuits only using Hadamard (H), Phase (S), CNOT (CX), and T gates. As a consequence, the respective functionalities

TABLE III: Common quantum algorithms

Benchmark				Equivalent		1 Gate Missing		Flipped CNOT	
Name	n	$\|G\|$	$\|G'\|$	t_{zx}[s]	t_{qcec}[s]	t_{zx}[s]	t_{qcec}[s]	t_{zx}[s]	t_{qcec}[s]
Compiled Circuits									
Grover	6	1606	2803	0.39	3.40	0.31	0.04	0.47	0.04
Grover	7	4732	8476	1.24	0.30	3.03	0.14	4.16	0.14
Grover	8	12482	22860	12.15	0.91	5.11	0.42	189.61	0.39
QFT	23	1311	3741	0.06	2.00	0.05	>3600	0.05	902.99
QFT	38	3591	10449	0.32	>3600	0.20	>3600	0.21	>3600
Random-Walk	7	6523	8955	150.36	0.24	9.35	0.14	55.44	0.16
Random-Walk	8	14084	19755	1289.13	0.57	455.58	0.33	687.95	0.31
Random-Walk	9	29325	41942	>3600	1.31	1001.93	0.59	2477.31	0.50
QPE-Exact	22	1217	3006	0.83	0.10	0.78	>3600	0.79	0.82
QPE-Exact	39	3823	11552	3.19	>3600	2.89	>3600	2.92	>3600
GHZ	65	130	493	0.06	<0.01	0.06	<0.01	0.59	0.01
Graph State	62	403	2041	0.36	0.17	0.43	0.17	0.35	0.17
Optimized Circuits									
Grover	8	12479	12287	8.00	0.04	2.11	0.24	9.837	0.04
Grover	9	37193	36881	82.9446	0.14	3.02	129.56	145.574	0.17
Grover	10	104977	104501	779.291	0.42	72.05	> 3600	90.68	41.24
Grover	11	308074	307322	2178.13	0.07	2316.00	588.62	2264.67	281.j04
QFT	32	2544	2482	1.43	0.04	1.49	3.57	1.89	14.53
QFT	43	4601	4502	2.86	10.837	2.86	17.78	2.81	1.02
QFT	44	4818	4702	3.01	>3600	3.05	1.27	2.90	1.21
QFT	75	14136	11013	1.23	>3600	1.30	>3600	1.26	>3600
Random-Walk	7	2351	1906	140.90	0.02	11.86	0.10	212.527	0.02
Random-Walk	8	4648	3925	2175.39	0.04	78.91	62.15	149.49	0.11
Random-Walk	9	9249	7987	>3600	0.09	245.39	0.11	>3600	0.11

(i.e., the system matrices) possess lots of structure that can be exploited by decision diagrams and, additionally, only feature a very limited set of complex numbers which limits the effect of numerical instabilities. In contrast, the ZX-calculus based approach does not benefit from this structure very much.

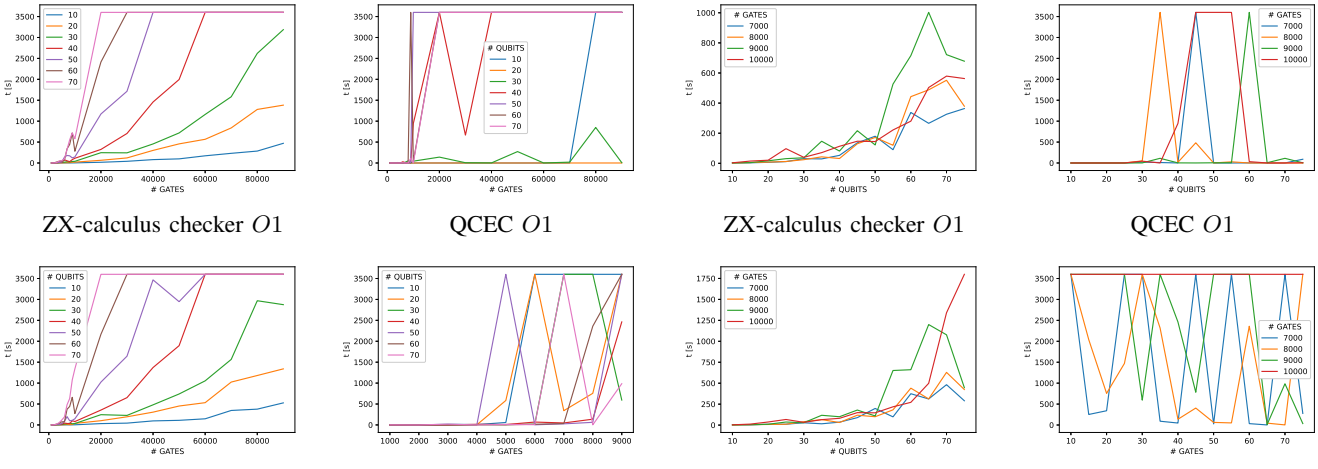
In the case of proper quantum circuits (Table III) the story looks a bit different. For circuits containing no or smaller reversible parts (such as the QFT or Quantum Phase Estimation), the ZX-calculus approach fairs much better in comparison to decision diagrams. The main obstacle in these cases is that the considered algorithms feature many rotation gates with arbitrarily small rotation angles. Due to numerical instabilities and rounding errors, it might happen that two decision diagram nodes that should be identical in theory, differ by a small margin in practice. As a consequence, inherent redundancies in the underlying representations cannot be captured accurately anymore. Thus, while the resulting decision diagram is very close to the identity with respect to the Hilbert-Schmidt norm, it might grow exponentially large in the worst case. In contrast, ZX-diagrams are not susceptible to such exponential growth under numerical errors.

The above observations are similar in the case of non-equivalent instances. Although runtimes for both methods are generally lower, the relative performances are still similar. Since the resulting decision diagram is almost guaranteed to not be very close to the identity during the equivalence check, the alternating scheme discussed cannot be as efficient as in the equivalent case. Due to this, QCEC resorts to simulations of the circuit with random inputs which, as shown in [23], are expected to show the non-equivalence within a few simulations. Yet, the complexity of decision diagram based simulation is still exponential in the worst case. The rewriting approach of the ZX-calculus is less volatile to errors in the circuit. During the equivalence check, the combined circuit diagram is simplified as much as possible until no more rules can be applied. Depending on the severity and kind of error, the procedure stops sooner or later. Of course, the ZX-calculus checker cannot prove non-equivalence of circuits, but the experiments show that inability to show equivalence of circuits with the ZX-calculus at least gives a strong indication that two circuits are indeed non-equivalent since the ZX-calculus checker managed to prove equivalence in all equivalent benchmarks.

What Table III also shows is the volatility of the decision diagram based approach. Runtimes can increase dramatically for the same type of circuit with a differing number of qubits. As soon as the intermediate decision diagram does not admit a compact representation, applying gates to it is a costly operation. This volatility is shown in more detail in the case of random Clifford+T benchmarks. Fig. 12a and Fig. 12b show the runtimes of equivalence checking random Clifford+T circuits with increasing gate count and number of qubits.

In Fig. 12a the different lines correspond to benchmarks with differing qubit counts. If a data point is at 3600s in the graph this indicates that the equivalence check took longer than the timeout limit of 3600s or—in the case of decision diagrams— it means that the memory limit has been exceeded. In Fig. 12b different lines correspond to benchmarks with differing gate counts in the original circuit. With $O1$ an average of 0.9% of gates were optimized away. With $O2$ an average of 6.1% of gates were optimized away.

Both Fig. 12a and Fig. 12b clearly show that the equivalence checking routine based on the ZX-calculus has a clear correlation between runtime and the size of the circuits, whether that size is due to the number of gates or qubits. Although there are some simpler instances where the runtime decreases,



(a) Runtimes for differing gate count (b) Runtimes for differing qubit count

Fig. 12: Equivalence checking random Clifford+T benchmarks

the general trend can be clearly seen. Fig. 12a also further supports the claim in [28] that the complexity of reducing a ZX-diagram to reduced gadget form is between $O(n)$ and $O(n^2)$ where n is the number of gates in the circuit.

On the other hand, the plots for QCEC show no such correlation. Whether QCEC manages to prove equivalence for a circuit only depends on the specific circuit in question after a certain circuit size and complexity has been reached. This is not too surprising—decision diagrams can blow up to exponential size with respect to the number of qubits. This volatility can hardly be held against decision diagrams though. After all, Fig. 12a and Fig. 12b show runtimes for *random* benchmarks, i.e. circuits that do not exhibit much structure. This volatility is actually a positive feature of the decision diagram based approach. If this method yields a result at all it usually does so using significantly less time than the ZX-calculus based method and is, therefore, able to prove equivalence of some very large circuits where the worst case complexity of $O(n^3)$ of the ZX-calculus based approach leads to long runtimes.

This shows that the two methods are complementary and are best used in tandem, especially for more optimized circuits. For circuits with many qubits but a smaller number of gates, the ZX-calculus based approach performs more favorably. For even larger circuits the problem itself is too complex to be solved even in polynomial time for the ZX-calculus based approach. In this case, the decision diagram based method might still be able to show equivalence by keeping the intermediate decision diagrams small. Because the size of the ZX-diagram during the equivalence check is bounded by the size of the original circuit (the number of spiders is strictly decreasing) the ZX-calculus based equivalence checking method has a low memory footprint. It can therefore easily be used in parallel with the decision diagram based method without using too many resources.

VIII. Conclusion

In this work, we examined the viability and effectiveness of the ZX-calculus for equivalence checking of quantum circuits. By improving the state of the art to be able to handle inaccurate representations of complex numbers, permutations of the input, and output layout of a circuit and ancillary qubits, we can now verify the results of compilation flows with the ZX-calculus. We have also discussed the limitations of the ZX-calculus based approach which prevents it from being a general equivalence checking method.

To give empirical results on the practicality of the ZX-calculus in equivalence checking, we conducted a case study comparing the ZX-calculus equivalence checker with one based on path-sums and one based on decision diagrams.

Empirical results show that path-sums and ZX-calculus exhibit similar scaling when checking the equivalence of Clifford circuits but the ZX-calculus based approach is still orders of magnitude faster on average. Also, the ZX-calculus and decision diagram show similar performance in many cases: but they differ in key aspects. Decision diagrams show significant benefits for circuits containing large reversible parts, such as oracles or adders. The sensibility of decision diagrams to numerical imprecision makes them hard to use on quantum algorithms that cannot be exactly represented using floating points, such as algorithms relying on arbitrary rotation angles, due to the potential blow-up of the intermediate representation. The ZX-calculus based equivalence checking procedure is less sensitive to this and is useful in showing equivalence in these cases. However, the ZX-calculus tends to be more suitable for verifying smaller building blocks than whole quantum algorithms due to the large number of involved gates. In conclusion, we can see that decision diagrams and the ZX-calculus can serve as complementary approaches for the equivalence checking problem.

Acknowledgements

This work received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 101001318), was part of the Munich Quantum Valley, which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus, and has been supported by the BMWK on the basis of a decision by the German Bundestag through project QuaST.

REFERENCES

[1] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.

[2] L. K. Grover, "A fast quantum mechanical algorithm for database search," *Proc. of the ACM*, pp. 212–219, 1996.

[3] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, 1997.

[4] M. Cerezo *et al.*, *Variational quantum algorithms*, 2020. arXiv: 2012.09265.

[5] E. Farhi *et al.*, *A quantum approximate optimization algorithm*, 2014. arXiv: 1411.4028.

[6] D. Herman *et al.*, *A survey of quantum computing for finance*, 2022. arXiv: 2201.02773.

[7] J. Biamonte *et al.*, "Quantum machine learning," *Nature*, vol. 549, no. 7671, pp. 195–202, 2017. arXiv: 1611.09347.

[8] S. Sivarajah *et al.*, "T—ket>: A retargetable compiler for NISQ devices," *Quantum Sci. Technol.*, 2020.

[9] M. Amy and V. Gheorghiu, "Staq—A full-stack quantum processing toolkit," *Quantum Sci. Technol.*, vol. 5, no. 3, p. 034 016, 2020.

[10] K. N. Smith and M. A. Thornton, "A quantum computational compiler and design tool for technology-specific targets," in *Int'l Symp. on Computer Architecture*, 2019, pp. 579–588.

[11] T. Häner *et al.*, "A software methodology for compiling quantum programs," *Quantum Sci. Technol.*, vol. 3, no. 2, p. 020 501, 2018.

[12] W. Hattori and S. Yamashita, "Quantum circuit optimization by changing the gate order for 2D nearest neighbor architectures," in *Int'l Conf. of Reversible Computation*, 2018, pp. 228–243.

[13] Z. Sasanian and D. M. Miller, "Reversible and quantum circuit optimization: A functional approach," in *Int'l Conf. of Reversible Computation*, R. Glück and T. Yokoyama, Eds., 2013, pp. 112–124.

[14] T. Itoko *et al.*, "Optimization of quantum circuit mapping using gate transformation and commutation," *Integration*, vol. 70, pp. 43–50, 2020.

[15] Y. Nam *et al.*, "Automated optimization of large quantum circuits with continuous parameters," *npj Quantum Inf*, 2018.

[16] B. A. Cordier *et al.*, *Biology and medicine in the landscape of quantum advantages*, 2021. arXiv: 2112.00760.

[17] D. Janzing *et al.*, ""Non-identity check" is QMA-complete," *Int. J. Quantum Inform.*, vol. 03, no. 03, pp. 463–473, 2005.

[18] S. Yamashita and I. L. Markov, "Fast equivalence-checking for quantum circuits," in *Int'l Symp. on Nanoscale Architectures*, 2010.

[19] L. Berent *et al.*, *Towards a SAT encoding for quantum circuits: A journey from classical circuits to Clifford circuits and beyond*, 2022. arXiv: 2203.00698.

[20] X. Hong *et al.*, *Approximate equivalence checking of noisy quantum circuits*, 2021. arXiv: 2103.11595.

[21] G. F. Viamontes *et al.*, "Checking equivalence of quantum circuits and states," in *Int'l Conf. on CAD*, 2007.

[22] M. Amy, "Towards large-scale functional verification of universal quantum circuits," in *International Conference on Quantum Physics and Logic*, 2019.

[23] L. Burgholzer and R. Wille, "Advanced equivalence checking for quantum circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2021.

[24] S.-A. Wang *et al.*, "An XQDD-based verification method for quantum circuits," in *IEICE Trans. Fundamentals*, 2008, pp. 584–594.

[25] A. Cowtan *et al.*, *A generic compilation strategy for the unitary coupled cluster ansatz*, 2020. arXiv: 2007.10515.

[26] J. van de Wetering, *ZX-calculus for the working quantum computer scientist*, 2020. arXiv: 2012.13966.

[27] R. Duncan *et al.*, *Graph-theoretic simplification of quantum circuits with the ZX-calculus*, 2019. arXiv: 1902.03178.

[28] A. Kissinger and J. van de Wetering, "Reducing T-count with the ZX-calculus," *Phys. Rev. A*, 2020.

[29] A. Kissinger and J. van de Wetering, "PyZX: Large scale automated diagrammatic reasoning," presented at the Quantum Physics and Logic, vol. 318, 2019, pp. 229–241.

[30] T. Peham *et al.*, "Equivalence checking paradigms in quantum circuit design: A case study," in *Design Automation Conf.*, 2022.

[31] G. F. Viamontes *et al.*, "Gate-level simulation of quantum circuits," in *Asia and South Pacific Design Automation Conf.*, 2003, pp. 295–301.

[32] P. Niemann *et al.*, "QMDDs: Efficient quantum function representation and manipulation," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2016.

[33] D. Miller and M. Thornton, "QMDD: A decision diagram structure for reversible and quantum circuits," in *Int'l Symp. on Multi-Valued Logic*, 2006.

[34] A. Zulehner *et al.*, "How to efficiently handle complex values? Implementing decision diagrams for quantum computing," in *Int'l Conf. on CAD*, 2019.

[35] D. M. Greenberger *et al.*, *Going beyond Bell's theorem*, 2007. arXiv: 0712.0921.

[36] G. Vidal and C. M. Dawson, "Universal quantum circuit for two-qubit transformations with three controlled-NOT gates," *Phys. Rev. A*, vol. 69, no. 1, p. 010 301, 2004.

[37] A. Barenco *et al.*, "Elementary gates for quantum computation," *Phys. Rev. A*, 1995.

[38] D. Maslov, "On the advantages of using relative phase Toffolis with an application to multiple control Toffoli optimization," *Phys. Rev. A*, vol. 93, no. 2, p. 022 311, 2016.

[39] R. Wille *et al.*, "Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations," in *Design Automation Conf.*, 2019.

[40] P. Murali *et al.*, "Noise-adaptive compiler mappings for Noisy Intermediate-Scale Quantum computers," in *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, 2019.

[41] G. Li *et al.*, "Tackling the qubit mapping problem for NISQ-era quantum devices," in *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, 2019.

[42] K. Hietala *et al.*, *A verified optimizer for quantum circuits*, 2019. arXiv: 1912.02250.

[43] B. Coecke and A. Kissinger, "Picturing quantum processes," in *Diagrammatic Representation and Inference*, P. Chapman *et al.*, Eds., 2018.

[44] J. D. Biamonte and V. Bergholm, *Tensor networks in a nutshell*, 2017. arXiv: 1708.00006.

[45] M. Backens, *The ZX-calculus is complete for stabilizer quantum mechanics*, 2013. arXiv: 1307.7025.

[46] R. Vilmart, *A near-optimal axiomatisation of ZX-calculus for pure qubit quantum mechanics*, 2018. arXiv: 1812.09114.

[47] R. Wille *et al.*, "JKQ: JKU tools for quantum computing," in *Int'l Conf. on CAD*, 2020.

[48] L. Burgholzer *et al.*, "Random stimuli generation for the verification of quantum circuits," in *Asia and South Pacific Design Automation Conf.*, 2021.

[49] L. Burgholzer *et al.*, "Verifying results of the IBM Qiskit quantum circuit compilation flow," in *Int'l Conf. on Quantum Computing and Engineering*, 2020.

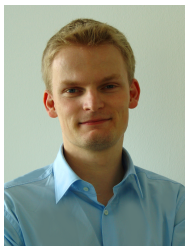
[50] A. W. Cross *et al.*, *OpenQASM 3: A broader and deeper quantum assembly language*, 2021. arXiv: 2104.14722.

[51] R. Wille *et al.*, "RevLib: An online resource for reversible functions and reversible circuits," in *Int'l Symp. on Multi-Valued Logic*, 2008, pp. 220–225.

[52] N. Quetschlich *et al.*, "MQT Bench: Benchmarking software and design automation tools for quantum computing," 2022. arXiv: 2204.13719.

Tom Peham Tom Peham received his Master's degree in computer science (2022) from the Johannes Kepler University Linz, Austria. He is currently a Ph.D. student at the Chair for Design Automation at the Technical University of Munich, Germany. His research interests include design automation for quantum computing—currently focusing on applications of the ZX-calculus in this domain.

Lukas Burgholzer Lukas Burgholzer (S'19) received his Master's degree in industrial mathematics (2018) and Bachelor's degree in computer science (2019) from the Johannes Kepler University Linz, Austria. He is currently a Ph.D. student at the Institute for Integrated Circuits at the Johannes Kepler University Linz, Austria. His research focuses on design automation and software for quantum computing. In these areas, he has published several papers on international conferences such as ASP-DAC, DAC, ICCAD, DATE, and QCE.



Robert Wille Robert Wille is a Full and Distinguished Professor at the Technical University of Munich, Germany, and Chief Scientific Officer at the Software Competence Center Hagenberg, Austria. He received the Diploma and Dr.-Ing. degrees in Computer Science from the University of Bremen, Germany, in 2006 and 2009, respectively. Since then, he worked at the University of Bremen, the German Research Center for Artificial Intelligence (DFKI), the University of Applied Science of Bremen, the University of Potsdam, and the Technical University Dresden. From 2015 until 2022, he was Full Professor at the Johannes Kepler University Linz, Austria, until he moved to Munich. His research interests are in the design of circuits and systems for both conventional and emerging technologies. In these areas, he published more than 400 papers and served in editorial boards as well as program committees of numerous journals/conferences such as TCAD, ASP-DAC, DAC, DATE, and ICCAD. For his research, he was awarded, e.g., with Best Paper Awards, e.g., at TCAD and ICCAD, an ERC Consolidator Grant, a Distinguished and a Lighthouse Professor appointment, a Google Research Award, and more.