Abstract

We present the theoretical foundations for a new quantum circuit optimisation technique based on an equational theory called the ZX-calculus. We first interpret quantum circuits as ZX-diagrams, which provide a flexible, lower-level language for describing quantum computations graphically. Then, using the rules of the ZX-calculus, we derive a terminating simplification procedure for ZX-diagrams based on the two graph transformations of local complementation and pivoting. While little is known about extracting an equivalent quantum circuit from an arbitrary ZX-diagram, we show that our simplification procedure preserves a graph property called focused gFlow, and use this property to derive a deterministic strategy for circuit re-extraction. Hence, this approach enables us to temporarily break the rigid quantum circuit structure and ‘see around’ obstructions (namely non-Clifford quantum gates) to produce significant reductions, while still maintaining enough data to extract a reduced quantum circuit at the end.

1 Introduction

Quantum circuits provide a de facto ‘assembly language’ for quantum computation, in which computations are described as the composition of many simple unitary linear maps called quantum gates. An important topic in the study of quantum circuits is quantum circuit optimisation, whereby quantum circuits which realise a given computation are transformed into new circuits involving fewer or simpler gates. While some strides have already been made in this area, the field is still in its infancy. Most approaches to quantum circuit optimisation are based on only a handful of techniques: gate substitutions, computation of small (pseudo-)normal forms for special families of circuits [1, 26, 24], optimisation of phase polynomials [3, 20], or some combination thereof [2, 28].

This paper lays a theoretical foundation for a completely new quantum circuit optimisation technique based on the ZX-calculus [9]. A key point of departure of our technique is that we break the rigid structure of quantum circuits and perform reductions on a lower-level, string-diagram based representation of a quantum circuit called a ZX-diagram. These diagrams are more flexible than circuits, in that they can be deformed arbitrarily, and are subject to a rich equational theory: the ZX-calculus. The core rules of the ZX-calculus give a sound and
Circuit Simplification with the ZX-calculus

complete [5] theory for Clifford circuits, a well-known class of circuits that can be efficiently classically simulated. More surprisingly, it was shown in 2018 that modest extensions to the ZX-calculus suffice to give completeness for families of circuits that are approximately universal [21] and exactly universal [18, 22, 30] for quantum computation.

Since ZX-diagrams provide a more flexible representation of a quantum computation than a circuit, we can derive many new simplifications from the rules of the ZX-calculus that have no analogue for quantum circuits. However, this added flexibility comes at a price: while any quantum circuit can be interpreted as a ZX-diagram by decomposing gates into smaller pieces, the converse is not true. For a generic ZX-diagram corresponding to a unitary, there is no known general-purpose procedure for efficiently recovering a quantum circuit. Hence, an important part of our optimisation procedure is keeping enough information about the quantum circuit structure to get a circuit back at the end. Schematically:

![Diagram](ZX-diagrams)

Quantum circuits

simplification

circuit extraction

Until recently, this extraction step was poorly understood, and the only techniques for doing circuit-to-circuit translation with the ZX-calculus did so without departing from the overall structure of the original quantum circuit [17], avoiding the extraction problem altogether. In this paper we adopt a more ambitious approach. First, building on prior work of two of the authors [14, 15, 16], we use the rules of the ZX-calculus to derive a sound, terminating simplification procedure for ZX-diagrams. The key simplification steps involve the graph-theoretic transformations of local complementation and pivoting [25, 7], which allow certain generators to be deleted from ZX-diagrams one-by-one or in pairs, respectively. When applied to Clifford circuits, the diagram resulting from the simplification procedure is represented in the form of a graph-state with local Cliffords (GS-LC) [19], a pseudo-normal form for representing Clifford circuits whose size is at most quadratic in the number of qubits. Hence, one side effect of our simplification procedure is that it gives a simple, graph-theoretic alternative to the normalisation of a ZX-diagram to GS-LC form proposed by Backens [5]. For non-Clifford circuits, the simplified ZX-diagram represents a ‘skeleton’ of the circuit we started with, consisting only of generators arising from non-Clifford phase gates and their nearest neighbours. Although this is no longer a canonical form, it can still be significantly smaller than the input circuit, especially when there is a relatively low proportion of non-Clifford gates. We then show that, even though this simplification breaks the circuit structure, it preserves a graph-theoretic invariant called focused $gFlow$ [8, 27], from which we can derive an efficient circuit extraction strategy. We demonstrate the full simplify-and-extract procedure by means of a running example, which is also available online as a Jupyter notebook [1].

In the case of Clifford circuits, this procedure will produce circuits comparable in size to those described by standard techniques [1, 29]. In the non-Clifford case, this can already find more simplifications than naively ‘cutting’ the circuit and simplifying purely-Clifford sections. Additionally, this simplify-and-extract setup provides a new framework for implementing

[1] nbviewer.jupyter.org/github/Quantomatic/pyzx/blob/906f6db3/demos/example-gtsimp.ipynb
further simplifications on the ‘skeleton’, e.g. for $T$-count reduction. We will comment on the latter point—and its implementation in a tool called PyZX [23]—as a topic of future work.

The paper is organised as follows. After giving a brief introduction to the ZX-calculus in Section 2 we will introduce graph-like ZX-diagrams in Section 3. This mild restriction to the family of all ZX-diagrams will enable us to speak more readily about graph-theoretic properties of the diagram, like the existence of a focused gFlow. In Section 4 we will show that both local complementation and pivoting preserve focused gFlow. Then in Section 5 we will derive a simplification routine using these graph-theoretic notions. We will show how the diagrams produced by this routine can be extracted into a circuit in Section 6, and finally, in Section 7 we conclude and discuss future work.

2 The ZX-calculus and quantum circuits

We will provide a brief overview of the ZX-calculus. For an in-depth reference see Ref. [10].

The ZX-calculus is a diagrammatic language similar to the familiar quantum circuit notation. A ZX-diagram consists of wires and spiders. Wires entering the diagram from the left are inputs; wires exiting to the right are outputs. Given two diagrams we can compose them by joining the outputs of the first to the inputs of the second, or form their tensor product by simply stacking the two diagrams.

Spiders are linear maps which can have any number of input or output wires. There are two varieties: $Z$ spiders depicted as green dots and $X$ spiders depicted as red dots. Written explicitly in Dirac ‘bra-ket’ notation, these linear maps are:

\[
\begin{align*}
Z\alpha & = |0\rangle + e^{i\alpha} |1\rangle \propto |+\rangle \\
X\alpha & = |0\rangle (|0\rangle + e^{i\alpha} |1\rangle) = Z\alpha
\end{align*}
\]

A ZX-diagram with $m$ input wires and $n$ output wires then represents a linear map $(\mathbb{C}^2)^\otimes m \to (\mathbb{C}^2)^\otimes n$ built from spiders (and permutations of qubits) by composition and tensor product of linear maps. As a special case, diagrams with no inputs and $n$ outputs represent vectors in $(\mathbb{C}^2)^\otimes n$, i.e. (unnormalised) $n$-qubit states.

Example 2.1. We can immediately write down some simple state preparations and unitaries in the ZX-calculus:

\[
\begin{align*}
\circ & = |0\rangle + |1\rangle \propto |+\rangle & \bullet & = |+\rangle + |-\rangle \propto |0\rangle \\
\circ & = |0\rangle (|0\rangle + e^{i\alpha} |1\rangle) = Z\alpha & \bullet & = |+\rangle (|+\rangle + e^{i\alpha} |-) = X\alpha
\end{align*}
\]

In particular we have the Pauli matrices:

\[
\begin{align*}
Z & = \begin{array}{c}
\circ \circ \circ \circ \circ \circ \\
\bullet \bullet \bullet \bullet \bullet \bullet
\end{array} & X & = \begin{array}{c}
\circ \circ \circ \\
\bullet \bullet \bullet
\end{array}
\end{align*}
\]

It will be convenient to introduce a symbol – a yellow square – for the Hadamard gate. This is defined by the equation:

\[
\begin{array}{c}
\circ \\
\bullet
\end{array} = \begin{array}{c}
\circ \circ \\
\bullet \bullet
\end{array}
\]

(1)

We will often use an alternative notation to simplify the diagrams, and replace a Hadamard between two spiders by a blue dashed edge, as illustrated below.

If you are reading this document in monochrome or otherwise have difficulty distinguishing green and red, $Z$ spiders will appear lightly-shaded and $X$ darkly-shaded.
Circuit Simplification with the ZX-calculus

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{ZX-calculus-diagrams.png}
\caption{A convenient presentation for the ZX-calculus. These rules hold for all \(\alpha, \beta \in [0, 2\pi]\), and due to \((h)\) and \((i2)\) all rules also hold with the colours interchanged.}
\end{figure}

Both the blue edge notation and the Hadamard box can always be translated back into
spiders when necessary. We will refer to the blue edge as a \textit{Hadamard edge}.

Two diagrams are considered equal when one can be deformed to the other by moving
the vertices around in the plane, bending, unbending, crossing, and uncrossing wires, so long
as the connectivity and the order of the inputs and outputs is maintained. Furthermore,
there is an additional set of equations that we call the rules of the ZX-calculus; these are
shown in Figure 1. Equal diagrams correspond to equal linear maps.\footnote{We will consider linear maps \(A\) and \(B\) to be “equal” if they satisfy \(A = zB\) for some non-zero \(z \in \mathbb{C}\). Backens \cite{Backens13} presents a version of the ZX-calculus where equality is “on the nose”.}

\begin{remark}
The ZX-calculus is universal in the sense that any linear map can be expressed
as a ZX-diagram. Furthermore, when restricted to Clifford ZX-diagrams, i.e. diagrams whose
phases are all multiples of \(\pi/2\), the version we present in Figure 1 is complete. That is,
for any two Clifford ZX-diagrams that describe the same linear map, there exists a series
of rewrites transforming one into the other. Recent extensions to the calculus have been
introduced which are complete for the larger Clifford+T family of ZX-diagrams \cite{Fal16},
where phases are multiples of \(\pi/4\), and for \textit{all} ZX-diagrams \cite{BP17,LP19,TV19}.

Quantum circuits can be translated into ZX-diagrams in a straightforward manner. We
will take as our starting point circuits constructed from the following universal set of gates:

\begin{align*}
\text{CNOT} & := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} & \text{Z}_\alpha & := \begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix} & H & := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}
\end{align*}

We fix this gate set because they admit a convenient representation in terms of spiders:

\begin{align*}
\text{CNOT} & = & \text{Z}_\alpha & = & H & =
\end{align*}

\begin{align*}
X_\alpha & = & \text{CZ} & =
\end{align*}

\begin{align*}
\text{CNOT} & := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} & \text{Z}_\alpha & := \begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix} & H & := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}
\end{align*}

Note that the green spider is the first (i.e. control) qubit of the CNOT and the red spider is the second (i.e. target) qubit. Other common gates can easily be expressed in terms of these

gates. In particular, \(S := Z_\pi\), \(T := Z_{\pi/2}\) and:

\begin{align*}
X_\alpha & = & \text{CZ} & =
\end{align*}
For our purposes, we will define quantum circuits as a special case of ZX-diagrams.

**Definition 2.3.** A circuit is a ZX-diagram generated by compositions and tensor products of the ZX-diagrams in equation (2).

Important subclasses of circuits are Clifford circuits, sometimes called stabilizer circuits, which are obtained from compositions of only CNOT, H, and S gates. They are efficiently classically simulable, thanks to the Gottesman-Knill theorem [1]. Local Clifford circuits are Clifford circuits on a single qubit, i.e. circuits consisting only of H and S gates.

The addition of T gates yields Clifford+T circuits, which are capable of approximating any n-qubit unitary to arbitrary precision, whereas the inclusion of Z_\alpha gates for all \alpha enables one to construct any unitary exactly.

### 3 Graph-like ZX-diagrams

Before starting our simplification procedure, we transform ZX-diagrams into the following, convenient form.

**Definition 3.1.** 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.

A ZX-diagram is called a graph state if it is graph-like, every spider is connected to an output, and there are no non-zero phases. This corresponds to the standard notion of graph state from the literature [19]. Hence, graph-like ZX-diagrams are a generalisation of graph states to maps where we allow arbitrary phases and some interior spiders, i.e. spiders not connected to an input or output.

**Lemma 3.2.** Every ZX-diagram is equal to a graph-like ZX-diagram.

**Proof.** Starting with an arbitrary ZX-diagram, we apply (h) to turn all red spiders into green spiders surrounded by Hadamard gates. We then remove excess Hadamards via (i2). Any non-Hadamard edge is then removed by fusing the adjacent spiders with (f). Any parallel Hadamard edges or self-loops can be removed via the following 3 rules:

\[
\begin{align*}
\cdots \quad \alpha \quad \beta \quad \cdots &= \cdots \quad \alpha \quad \beta \quad \cdots \quad \\
\cdots &= \cdots \quad \alpha \quad \cdots \quad \\
\alpha &= \alpha + \pi
\end{align*}
\]

which we prove in Appendix B.1. Finally, we can apply the rules (i1) and (i2) from right-to-left until the last condition is satisfied.

A useful feature of graph-like ZX-diagrams is that much of its structure is captured by its underlying open graph.

**Definition 3.3.** An open graph is a triple \((G, I, O)\) where \(G = (V, E)\) is an undirected graph, and \(I \subseteq V\) is a set of inputs and \(O \subseteq V\) a set of outputs. For a ZX-diagram \(D\), the underlying open graph \(G(D)\) is the open graph whose vertices are the spiders of \(D\), whose edges correspond to Hadamard edges, and whose sets \(I\) and \(O\) are the subsets of spiders connected to the inputs and outputs of \(D\), respectively.
Circuit Simplification with the ZX-calculus

\[ \alpha \rightarrow I \ni I \ni I \ni \in \in O \in O \]

Figure 2: A circuit, an equivalent graph-like ZX-diagram, and its underlying open graph.

Note that, because of Definition 3.1, the sets \( I \) and \( O \) in \( G(D) \) will always be disjoint. When discussing open graphs we will use the notation \( \overline{I} := V \setminus I \) and \( \overline{O} := V \setminus O \) for the non-inputs and non-outputs respectively. The set of neighbours of a vertex \( v \in V \) will be denoted by \( N(v) \).

Definition 3.4. [27] Given an open graph \( G \), a focused gFlow \((g, \prec)\) on \( G \) consists of a function \( g : O \rightarrow 2^I \) and a partial order \( \prec \) on the vertices \( V \) of \( G \) such that for all \( u \in O \),

1. \( \text{Odd}_G(g(u)) \cap \overline{O} = \{u\} \)
2. \( \forall v \in g(u), \ u \prec v \)

where \( \text{Odd}_G(A) := \{v \in V(G) \mid |N(v) \cap A| = 1 \mod 2\} \) is the odd neighbourhood of \( A \).

Remark 3.5. Generalised flow (gFlow) techniques were introduced in the context of measurement-based quantum computing [12, 13]. Focused gFlow is a special case of the standard notion of gFlow [8], where the latter allows other vertices \( v \) in the set \( \text{Odd}_G(g(u)) \cap \overline{O} \), provided they are in the future of \( u \) (i.e. \( u \prec v \)). However, as a property of graphs, the two notions are equivalent: an open graph has a gFlow if and only if it has a focused gFlow. We will rely on this equivalence in Appendix B.2 to prove the following Lemma.

Lemma 3.6. If \( D \) is a graph-like ZX-diagram obtained from a circuit by the procedure of Lemma 3.2, then \( G(D) \) admits a focused gFlow.

Proof. See Appendix B.2.

4 Local complementation and pivoting

Local complementation is a graph transformation introduced by Kotzig [25].

Definition 4.1. Let \( G \) be a graph and let \( u \) be a vertex of \( G \). The local complementation of \( G \) according to \( u \), written as \( G ^\ast u \), is a graph which has the same vertices as \( G \), but all the neighbours \( v, w \) of \( u \) are connected in \( G ^\ast u \) if and only if they are not connected in \( G \). All other edges are unchanged.

Example 4.2. Local complementations according to the vertices \( a \) and \( b \) are given by:

\[
\begin{align*}
G & \quad \xrightarrow{a} \quad G ^\ast a \\
& \quad \xrightarrow{b} \quad (G ^\ast a) ^\ast b
\end{align*}
\]

A related graph-theoretic operation is pivoting, which takes place at a pair of adjacent vertices.

Definition 4.3. Let \( G \) be a graph, and let \( u \) and \( v \) be a pair of connected vertices in \( G \). The pivot of \( G \) along the edge \( uv \) is the graph \( G \land uv := G ^\ast u \ast v \ast u \).
On a graph, pivoting consists in exchanging \( u \) and \( v \), and complementing the edges between three particular subsets of the vertices: the common neighbourhood of \( u \) and \( v \) (i.e. \( N_G(u) \cap N_G(v) \)), the exclusive neighbourhood of \( u \) (i.e. \( N_G(u) \setminus (N_G(v) \cup \{v\}) \)), and exclusive neighbourhood of \( v \) (i.e. \( N_G(v) \setminus (N_G(u) \cup \{u\}) \)):

\[
\begin{array}{c}
\text{G} \\
A \\
B \\
C \\
\end{array}
\begin{array}{c}
u \\
v \\
w \\
w \\
\end{array}
\]

\[
\begin{array}{c}
\text{G} \land uv \\
A \\
B \\
C \\
\end{array}
\begin{array}{c}
u \\
v \\
w \\
w \\
\end{array}
\]

Our primary interest in local complementation and pivoting is that each corresponds to a transformation of ZX-diagrams. In the special case where a ZX-diagram \( D \) is a graph-state, it is possible to obtain a diagram \( D' \) where \( G(D') = G(D) \star u \) by applying an \( X-\pi/2 \) gate on the spider corresponding to \( u \) and \( Z-\pi/2 \) gates on all of the spiders in \( N(v) \):

\[
\begin{array}{c}
\text{N}(u) \\
\end{array}
\begin{array}{c}
u \\
v \\
w \\
w \\
\end{array}
\]

This is a well-known property of graph states, and a derivation using the rules of the ZX-calculus was given in [14]. Similarly, it was shown in [16] that a pivot \( G(D') = G(D) \land uv \) can be introduced by applying Hadamard gates on \( u \) and \( v \) and \( Z_{\pi} \) gates on \( N(u) \setminus N(v) \):

\[
\begin{array}{c}
\text{N}(u) \setminus (N(v) \cup \{v\}) \\
\end{array}
\begin{array}{c}
u \\
v \\
w \\
w \\
\end{array}
\]

\[
\begin{array}{c}
\text{N}(v) \setminus (N(u) \cup \{u\}) \\
\end{array}
\begin{array}{c}
u \\
v \\
w \\
w \\
\end{array}
\]

Note that the swap on the RHS comes from the vertices \( u \) and \( v \) being interchanged by the pivot.

The following theorem will prove pivotal in our extraction routine. It shows that the existence of a focused gflow is preserved by local complementation (resp. pivoting) followed by the deletion of the vertex (resp. the pair of vertices) on which the transformation is applied:

\begin{itemize}
    \item \textbf{Theorem 4.4.} Let \((G, I, O)\) be an open graph that admits a focused gFlow,
    \begin{enumerate}
        \item for \( u \notin I \cup O \), setting \( G' := (G \star u) \setminus \{u\} \)
        \item for \( u, v \notin I \cup O \), setting \( G' := (G \land uv) \setminus \{u, v\} \)
    \end{enumerate}
    where \( G \setminus W \) is the graph obtained by deleting the vertices in \( W \) and any adjacent edges.
\end{itemize}

\textbf{Proof.} The two cases are proved in Appendix [B.3], Lemmas [B.9] and [B.12] respectively.

\section{A simplification strategy for circuits}

We now have all the necessary machinery to introduce our simplification routine. The general idea is to use local complementation and pivoting based rewrite rules to remove as many interior spiders as possible. A spider is called interior when it is not connected to an input or an output, otherwise it is called a boundary spider.
Definition 5.1. We call a spider Pauli if its phase is a multiple of $\pi$, and Clifford if its phase is a multiple of $\frac{\pi}{2}$. If the phase of a Clifford spider is an odd multiple of $\frac{\pi}{2}$ (and hence non-Pauli), we call this a proper Clifford spider.

The graph-like ZX-diagram resulting from the translation of a Clifford circuit contains only Clifford spiders, since the only time the phase changes on a spider is during a spider-fusion step, in which case the phases are added together.

We will show that our procedure is able to eliminate all interior proper Clifford spiders and all Pauli spiders adjacent either to a boundary spider or any interior Pauli spider. In particular, for Clifford circuits, we will obtain a pseudo-normal form which contains no interior spiders (cf. Section 6).

The main rewrite rules we use are based on local complementation and pivoting. The first one allows us to remove any interior proper Clifford spider while the second one removes any two connected interior Pauli spiders.

Lemma 5.2. The following rule holds in the ZX-calculus:

\[
\begin{align*}
\alpha_1 & \pm \frac{\pi}{2} \ldots \\
& \pm \left( \alpha_n \pm \frac{\pi}{2} \right) \\
\alpha_2 & \ldots \\
\end{align*}
\]

where the RHS is obtained from the LHS by performing a local complementation at the marked vertex, removing it, and updating the phases as shown.

Lemma 5.3. The following rule holds in the ZX-calculus:

\[
\begin{align*}
\alpha_1 & \mp j\pi \\
& \mp \left( \alpha_n \mp j\pi \right) \\
\alpha_2 & \ldots \\
\end{align*}
\]

where the RHS is obtained from the LHS by performing a pivot at the marked vertices, removing them, and updating the phases as shown.

The proof of these lemmas can be found in Appendix B.4.

We can additionally apply (8) to remove an interior Pauli spider that is adjacent to a boundary spider. To do this, we first turn the boundary spider into a (phaseless) interior spider via the following transformation, which follows from the rules \((f)\), \((i1)\), and \((i2)\):

\[
\begin{align*}
\alpha_1 & \mp j\pi \\
& \mp \left( \alpha_n \mp j\pi \right) \\
\alpha_2 & \ldots \\
\end{align*}
\]

After this transformation, we can perform (8) on the two spiders labelled \(\{u, v\}\) to remove them. For our purposes, we can then treat \(w\) as a boundary spider, and save the single-qubit unitaries to the right of \(w\) separately.

Theorem 5.4. There exists a terminating procedure which turns any graph-like ZX-diagram \(D\) into a graph-like ZX-diagram \(D'\) (up to single-qubit unitaries on inputs/outputs) which does not contain:
1. interior proper Clifford spiders
2. adjacent pairs of interior Pauli spiders
3. interior Pauli spiders adjacent to a boundary spider

In particular, if $D$ only contains Clifford spiders, then $D'$ contains no interior spiders.

**Proof.** Starting with a graph-like ZX-diagram, mark every interior Clifford spider for simplification. Then repeat the steps below (followed by (4) to remove parallel edges), until no rule matches:

- apply (7) to remove a marked proper Clifford spider
- apply (8) to remove an adjacent pair of marked Pauli spiders
- apply (9) to a boundary adjacent to a marked Pauli spider and immediately apply (8)

All 3 rules remove at least 1 marked node, so the procedure terminates. Since none of these rules will introduce non-Clifford spiders, if $D$ contains only Clifford spiders so too does $D'$. Since the only possibilities for interior spiders in $D'$ are covered by cases 1-3, $D'$ contains no interior spiders.

**Corollary 5.5.** For $D$ and $D'$ the ZX-diagrams from Theorem 5.4, if $G(D)$ admits a focused $gFlow$, then so too does $G(D')$.

**Proof.** It was shown in Theorem 4.4 that the transformations (7) and (8) preserve focused $gFlow$. The transformation (9) acts as an input/output extension on $G(D)$. That is, for $X \in \{I, O\}$, $v \in X$, it adds a new vertex $w$ and an edge $vw$, and sets $X' := (X \setminus \{v\}) \cup \{w\}$. It was shown in [27] that these transformations preserve focused $gflow$.

This simplification procedure is very effective, especially for circuits with few non-Clifford gates. We can see this in the following example:

The circuit above has 195 gates, with 4 non-Clifford gates. If we apply the simplification procedure, we obtain a small 'skeleton' of the circuit, containing 12 spiders:
Note the non-Clifford phases now clearly appear on the 4 interior spiders. In general, the interior spiders will either be non-Clifford or be Pauli with all non-Clifford neighbours. Any other possibilities will be removed.

6 Circuit extraction

As described in the previous section, when a Clifford circuit is fed into our simplification routine, no interior spiders will be left, and hence the diagram is in GS-LC form (graph-state with local Cliffords). It is straightforward to extract a Clifford circuit from a GS-LC form. First, we unfuse spiders to create CZ gates and local Cliffords on the inputs and outputs, then apply the \( (h) \) rule to the spiders on the right side of the resulting bipartite graph:

\[
\begin{align*}
\text{LC} & \quad \text{LC} \\
\text{LC} & \quad \text{LC} \\
\cdots & \quad \cdots \\
\text{LC} & \quad \text{LC}
\end{align*}
\]

The part of the diagram marked \( P \) now has the form of a classical parity circuit. That is, it is a permutation of computational basis states where the output basis is given in terms of parities of the inputs, e.g. \(|x_1, x_2, x_3, x_4\rangle \mapsto |x_1 \oplus x_2, x_1 \oplus x_3, x_4, x_3\rangle\). Such a unitary can always be realised via CNOT gates, using standard techniques based on Gaussian elimination over \( \mathbb{F}_2 \) (see e.g. \[26\]). Hence, we obtain a Clifford circuit with 6 layers of gates:

Local Cliffords + CZs + CNOTs + Hadamards + CZs + Local Cliffords \quad (11)

▶ Remark 6.1. This is similar to the 11-layer pseudo-normal form for Clifford circuits given in \[11\] and the slightly more compact version in \[29\]. The reason we obtain fewer layers is we allow CZ gates in the decomposition, rather than expanding them in terms of CNOT and local Cliffords.

If a Clifford+T circuit – or a more general circuit containing gates \( Z_\alpha \) or \( X_\alpha \) for \( \alpha \neq k\frac{\pi}{2} \) – is fed into our simplification routine, there might still be interior spiders left, and hence the diagram will not be in GS-LC form.

Our general strategy for extracting a circuit from a diagram produced by our simplification routine is to progress through the diagram from right-to-left, consuming vertices to the left and introducing quantum gates on the right. For an \( n \)-qubit circuit, we maintain a layer of \( n \) vertices called the frontier between these two parts. The goal is therefore to progress the frontier from the outputs all the way to the inputs, at which point we are left with an extracted circuit. We will see that the existence of a focused gFlow guarantees we can always progress the frontier leftward in this manner.

Much like in the Clifford case, our extraction procedure relies on applying (a variant of) Gaussian elimination on the adjacency matrix of a graph. Hence, we first establish a correspondence between primitive row operations and CNOT gates.

▶ Proposition 6.2. For any ZX-diagram \( D \), the following equation holds:

\[
\begin{align*}
\text{\ldots} & \quad D \quad \text{\ldots} \\
M & \quad M'
\end{align*}
\]
where $M$ describes the biadjacency matrix of the relevant vertices, and $M'$ is the matrix produced by starting with $M$ and then adding row 2 to row 1, taking sums modulo 2. Furthermore, if the diagram on the LHS has a focused $g$Flow, then so does the RHS.

Proof. See Appendix B.4.

We can now start describing our extraction procedure. To start, let the frontier $F$ consist of all the output vertices. If a vertex of the frontier has a non-zero phase, unfuse this phase in the direction of the output. Whenever two vertices of the frontier are connected, unfuse this connection into a CZ gate. If the diagram started out having a $g$Flow, then the resulting diagram still has a $g$Flow since we only affected output vertices. Let $g$ be the $g$Flow of the diagram. We push the frontier onto new vertices in the following way:

1. Find an unextracted vertex $v$ such that $g(v) \subseteq F$. Such a vertex always exists: take for instance any non-frontier vertex that is maximal in the $g$Flow order.

2. We know that $\text{Odd}(g(v)) = \{v\}$ hence we can pick a designated vertex $w \in F$ such that $w$ is connected to $v$.

3. Write the biadjency matrix of $g(v) \subseteq F$ and their neighbours to the left of the frontier (‘the past’). Because $\text{Odd}(g(v)) = \{v\}$ we know that the sum of all the rows in this matrix modulo 2 is an all-zero vector except for a 1 corresponding to $v$. Hence, if we apply row operations with $w$ as the target from every other vertex in $g(v)$ we know that $w$ will have only a single vertex connected to it in its past, namely $v$. Do these row operations using Proposition 6.2. As a result, CNOT gates will be pushed onto the extracted circuit. By that same proposition, the diagram will still have a $g$Flow.

4. At this point it might be that $v$ is still connected to vertices of $F$ that are not $w$. For each of these vertices $w' \in F$, do a row operation from $w$ to $w'$ by adding the corresponding CNOT to the extracted diagram in order to disconnect $w'$ from $v$. At this point $v$’s only connection to $F$ is via $w$. We can therefore conclude that the $g$Flow of the resulting diagram is such that the only vertex $u$ that has $w \in g(u)$ is $v$.

5. The vertex $w$ has arity 2 and since it is phaseless we can remove it from the diagram by (i1). Remove $w$ from the frontier, and replace it by $v$. By the remarks in the previous point the resulting diagram still has a $g$Flow. Unfuse the phase of $v$ as a phase gate to the extracted part.

6. If there is any vertex in $F$ that is not connected to an input, go back to step 1, otherwise we are done.

After all vertices have been extracted by the above procedure, we will have some permutation of the frontier vertices to the inputs. The extraction is finished by realising this permutation as a series of SWAP gates.

For a pseudocode description of the extraction algorithm, see Appendix C in the Appendix. For a fully worked-out example we refer to Appendix A.

\begin{remark} An interesting thing to note is we do not actually need a pre-calculated focused $g$Flow for this algorithm: knowing that one exists is enough. In step 3, it suffices to apply primitive row operations until a row with a single 1 appears in the biadjacency matrix. If this is possible by some set of row operations (i.e. if there exists a focused $g$Flow), such a row will always appear in the reduced echelon form, so we can simply do Gauss-Jordan elimination. This is how we describe this step in the pseudocode presented in Appendix C.

Picking up our example from the previous section, we can apply this procedure to the skeleton of the circuit in equation (10) and we obtain a new, extracted circuit:
The circuit has been reduced from 195 gates to 41 gates, going via an intermediate representation as a ZX-diagram with 12 spiders. As mentioned in the introduction, this example is available as a Jupyter notebook `demos/example-gtsimp.ipynb` in the PyZX repository on GitHub.

7 Conclusions and Future Work

We have introduced a terminating rewrite strategy for ZX-diagrams that simplifies Clifford circuits to GS-LC form using the graph-theoretic notions of local complementation and pivoting. We have shown how the diagrams produced by the rewrite strategy applied to non-Clifford circuits can be turned back into a circuit using the fact that our rewrites preserve focused gFlow.

The question of how a circuit can be extracted from a general ZX-diagram is still open. We speculate that this problem is not tractable in the general case, as it seems to be related to the problem of finding an ancilla-free unitary from a circuit containing ancillae. This observation does however not preclude finding larger classes of rewrites and ZX-diagrams such that we can efficiently extract a circuit.

The next step for optimising quantum circuits using ZX-diagrams is to find additional rewrite rules that allow one to produce smaller diagrams, while still being able to extract the diagrams into a circuit. An approach based on replacing non-Clifford phase gates with certain ZX-diagrams called phase gadgets during the simplification procedure has shown some promise. These phase gadgets enable phases to propagate non-locally through the circuit, much like they do in phase-polynomial approaches to quantum circuit optimisation [20, 4, 3]. However, unlike those approaches, the framework we described in this paper (with the addition of phase gadgets) applies to arbitrary circuits, rather than only \{CNOT, Z_\alpha\} circuits. Various heuristics involving phase gadgets have been implemented in the circuit optimisation tool PyZX, however a detailed description of these techniques and analysis of their performance is a topic of future work.

A second avenue of future work is to adapt the circuit extraction procedure to work in constrained qubit topologies. Many near-term quantum devices only allow interactions between certain pairs of qubits (e.g., nearest neighbours in some fixed planar graph). A notable feature of the circuit extraction procedure described in Section 6 is that we have some freedom in choosing which CNOTs to apply in reducing the bi-adjacency matrix of the ZX-diagram. We are currently exploring techniques by which this extraction can be constrained to only apply CNOTs in ‘allowed’ locations to produce circuits suitable for hardware with topological constraints.

Acknowledgements. SP acknowledges support from the projects ANR-17-CE25-0009 SoftQPro, ANR-17-CE24-0035 VanQuTe, PIA-GDN/Quantex, and LUE / UOQ. AK and JvdW are supported in part by AFOSR grant FA2386-18-1-4028.
References

1. Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5):052328, 2004.
2. Nabila Abdessaied, Mathias Soeken, and Rolf Drechsler. Quantum circuit optimization by hadamard gate reduction. In *International Conference on Reversible Computation*, pages 149–162. Springer, 2014.
3. Matthew Amy, Dmitri Maslov, and Michele Mosca. Polynomial-time T-depth optimization of Clifford+ T circuits via matroid partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(10):1476–1489, 2014.
4. Matthew Amy and Michele Mosca. T-count optimization and Reed-Muller codes. 2016. arXiv:1601.07363.
5. Miriam Backens. The zx-calculus is complete for stabilizer quantum mechanics. *New Journal of Physics*, 16(9):093021, 2014.
6. Miriam Backens. Making the stabilizer zx-calculus complete for scalars. In Chris Heunen, Peter Selinger, and Jamie Vicary, editors, *Proceedings of the 12th International Workshop on Quantum Physics and Logic (QPL 2015)*, volume 195 of *Electronic Proceedings in Theoretical Computer Science*, pages 17–32, 2015. arXiv:1507.03854 doi:10.4204/EPTCS.195.2.
7. André Bouchet. Graphic presentations of isotropic systems. *J. Comb. Theory Ser. A*, 45:58–76, July 1987. URL: [http://dl.acm.org/citation.cfm?id=51355.51360](http://dl.acm.org/citation.cfm?id=51355.51360).
8. Daniel E Browne, Elham Kashefi, Mehdi Mhalla, and Simon Perdrix. Generalized flow and determinism in measurement-based quantum computation. *New Journal of Physics*, 9(8):250, 2007. URL: [http://stacks.iop.org/1367-2630/9/i=8/a=250](http://stacks.iop.org/1367-2630/9/i=8/a=250).
9. V. Danos and E. Kashefi. Determinism in the one-way model. *Phys. Rev. A*, 74(052310), 2006. arXiv:0704.1263.
10. V. Danos, E. Kashefi, and P. Panangaden. The Measurement Calculus. April 2007. arXiv:0704.1263.
11. Vincent Danos, Elham Kashefi, Prakash Panangaden, and Simon Perdrix. Extended Measurement Calculus. *Semantic Techniques in Quantum Computation*, 2010.
12. R. Duncan and S. Perdrix. Graph states and the necessity of Euler decomposition. *Mathematical Theory and Computational Practice*, pages 167–177, 2009.
13. R. Duncan and S. Perdrix. Rewriting measurement-based quantum computations with generalised flow. In *Proceedings of ICALP*, Lecture Notes in Computer Science, pages 285–296. Springer, 2010.
14. Ross Duncan and Simon Perdrix. Pivoting makes the ZX-calculus complete for real stabilizers. In *Proceedings of the 10th International Workshop on Quantum Physics and Logic (QPL)*, volume 171 of *Electronic Proceedings in Theoretical Computer Science*, pages 50–62. Open Publishing Association, 2014. doi:10.4204/EPTCS.171.5.
15. Andrew Fagan and Ross Duncan. Optimising Clifford Circuits with Quantomatic. In *Proceedings of the 15th International Conference on Quantum Physics and Logic (QPL)*, volume 287 of *Electronic Proceedings in Theoretical Computer Science*, pages 85–105. Open Publishing Association, 2019. doi:10.4204/EPTCS.287.5.
16. Amar Hadžihalilović, Kang Feng Ng, and Quanlong Wang. Two complete axiomatisations of pure-state qubit quantum computing. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS ’18, pages 502–511, New York, NY, USA, 2018. ACM. doi:10.1145/3209108.3209128.
17. Marc Hein, Wolfgang Dür, Jens Eisert, Robert Raussendorf, M Nest, and H-J Briegel. Entanglement in graph states and its applications. 2006. arXiv:quant-ph/0602096.
18. Luke E Heyfron and Earl T Campbell. An efficient quantum compiler that reduces T count. *Quantum Science and Technology*, 4(015004), 2018. doi:10.1088/2058-9565/aad604.
Circuit Simplification with the ZX-calculus

21 Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. A Complete Axiomatisation of the ZX-Calculus for Clifford+T Quantum Mechanics. In Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '18, pages 559–568, New York, NY, USA, 2018. ACM. doi:10.1145/3209108.3209131

22 Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. Diagrammatic Reasoning Beyond Clifford+T Quantum Mechanics. In Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '18, pages 569–578, New York, NY, USA, 2018. ACM. doi:10.1145/3209108.3209139

23 Aleks Kissinger and John van de Wetering. PyZX: A circuit optimisation tool based on the ZX-calculus. http://github.com/Quantomatic/pyzx

24 Vadym Kliuchnikov and Dmitri Maslov. Optimization of Clifford circuits. Phys. Rev. A, 88:052307, Nov 2013. doi:10.1103/PhysRevA.88.052307

25 Anton Kotzig. Eulerian lines in finite 4-valent graphs and their transformations. In Colloquium on Graph Theory Tihany 1966, pages 219–230. Academic Press, 1968.

26 Ketan Markov, Igor Patel, and John Hayes. Optimal synthesis of linear reversible circuits. Quantum Information and Computation, 8(3&4):0282–0294, 2008.

27 Mehdi Mhalla, Mio Murao, Simon Perdrix, Masato Someya, and Peter S Turner. Which graph states are useful for quantum information processing? In Conference on Quantum Computation, Communication, and Cryptography, pages 174–187. Springer, 2011.

28 Yunseong Nam, Neil J Ross, Yuan Su, Andrew M Childs, and Dmitri Maslov. Automated optimization of large quantum circuits with continuous parameters. npj Quantum Information, 4(1):23, 2018.

29 Maarten Van Den Nest. Classical simulation of quantum computation, the gottesman-knill theorem, and slightly beyond. Quantum Information & Computation, 10(3):258–271, 2010.

30 Renaud Vilmart. A Near-Optimal Axiomatisation of ZX-Calculus for Pure Qubit Quantum Mechanics. 2018. arXiv:1812.09114
In this section we will work through a complete example of our simplification algorithm. Let us start with the following circuit:

This has 5 two-qubit gates and 19 single-qubit gates. Using the procedure of Lemma 3.2 to get it in graph-like form we get:

We see that we have multiple interior Pauli and proper Clifford phases and hence we must do local complementations and pivots. We start by doing a sequence of local complementations. The locations of these operations are marked by a red star.

We now have two internal Pauli vertices left, but they are not connected. Hence we must use equation (9), to be able to get rid of them:

Applying the pivots to the designated vertices gives us:
Note that in the bottom qubit we end up where we started before doing the pivots, but now we have marked the $\pi/2$ vertex as a local Clifford, and hence the $\pi$ vertex now counts as a boundary vertex. This means the simplification procedure has ended. The final step is extracting a circuit:

Here the frontier is marked by the box. We first unfuse the phases onto gates. Then we want to extract new vertices. These are marked by a red star. Since the three marked vertices have no other connections to the frontier, and the frontier vertices have only these single neighbours, we can extract these by simply putting them onto the frontier. The next step is to extract the bottomleft vertex, but in this case there is a connection too many which we must extract as a CNOT:

We now simply repeat this procedure until the entire circuit is extracted.
The extracted circuit has 4 two-qubit gates and 18 single-qubit gates. Applying some final cancellations between Hadamards on the inputs and outputs gets the number of single-qubit gates down to 14:

### B Proofs

#### B.1 Derived Rules in the ZX-calculus

**Lemma B.1.** The antipode rule,

\[
\begin{array}{c}
\begin{array}{c}
\bigcirc \\
\end{array}
\end{array}
\]

is derivable in the ZX-calculus.

**Proof.** To derive the \((a)\) rule we take advantage of the freedom to deform the diagram as shown below.

\[
\begin{array}{c}
\begin{array}{c}
\bigcirc \\
\end{array}
\end{array} = \begin{array}{c}
\begin{array}{c}
\bigcirc \\
\end{array}
\end{array} = \begin{array}{c}
\begin{array}{c}
\bigcirc \\
\end{array}
\end{array}
\]

The final equation is simply dropping the scalar.

**Lemma B.2.** The \(\pi\)-copy rule,

\[
\begin{array}{c}
\begin{array}{c}
\bigcirc \\
\end{array}
\end{array}
\]
is derivable in the ZX-calculus.

**Proof.** To derive the \((\pi c)\) rule we use the \((f)\) rule first to split the \(\pi\)-labelled vertex, and the \((\pi)\) and \((c)\) rules do the bulk of the work as shown below:

\[
\begin{align*}
\pi & \Rightarrow \alpha \\
\pi & \Rightarrow \alpha + \pi
\end{align*}
\]

**Lemma B.3.** Self-loops can be removed in the ZX-calculus:

\[
\begin{align*}
\alpha & \Rightarrow \alpha \\
\alpha & \Rightarrow \alpha + \pi
\end{align*}
\]

**Proof.** The first one follows by applying \((i1)\) from right to left and then using \((f)\). For the second one we do:

\[
\begin{align*}
\alpha & \Rightarrow \alpha + \pi
\end{align*}
\]

The first step is simply the definition of the Hadamard box, and in the last step we implicitly drop the scalar X spider that we are left with.

**Lemma B.4.** Parallel Hadamard edges can be removed in the ZX-calculus:

\[
\begin{align*}
\alpha & \Rightarrow \alpha
\end{align*}
\]

**Proof.**

\[
\begin{align*}
\alpha & \Rightarrow \alpha + \alpha
\end{align*}
\]

**B.2 Circuits and focused gFlow**

To show that graph-like ZX-diagrams arising from circuits admit a focused gFlow, we introduce a simpler sufficient condition.

**Definition B.5.** \([11]\) Given an open graph \(G\), a causal flow \((f, \prec)\) on \(G\) consists of a function \(f : O \rightarrow T\) and a partial order \(\prec\) on the set \(V\) satisfying the following properties:

1. \(f(v) \sim v\)
2. \(v \prec f(v)\)
3. if \(u \sim f(v)\) then \(v \prec u\)

where \(v \sim u\) means that \(u\) and \(v\) are adjacent in the graph.
Like focused gFlow, causal flow is a special case of a more general property called gFlow:

**Definition B.6.** Given an open graph $G$, a generalised flow, or gFlow consists of a pair $(g : \overline{O} \to 2^I, \prec)$ such that for any $u \in \overline{O}$:

1. $u \in Odd_G(g(u))$
2. if $v \in g(u)$, then $u \prec v$
3. if $v \in Odd_G(g(u))$ and $v \neq u$, then $u \prec v$

In the case where all of the sets $g(u)$ are singletons, the conditions above are equivalent to the causal flow conditions. Hence, if a graph admits a causal flow, it also admits a gFlow by letting $g(u) := \{f(u)\}$. It was furthermore shown in [27] that any gFlow can be systematically transformed into a focused gFlow. Combining these two facts, we obtain the following theorem.

**Theorem B.7.** If an open graph admits a causal flow $(f, \prec)$, then it also admits a focused gFlow.

The following proof of Lemma 3.6 shows that the diagrams produced by making a circuit graph-like according to the procedure outlined in Lemma 3.2 have causal flow and hence a focused gFlow.

**Proof of Lemma 3.6** Let $D$ denote the circuit, and let $D'$ denote the diagram produced by applying the procedure of Lemma 3.2 to $D$.

In order to prove that $D'$ has a focused gFlow, it suffices to show that it admits a causal flow, by Theorem B.7.

With every spider $v$ of the circuit $D$ we associate a number $q_v$ specifying on which ‘qubit-line’ it appears. We also associate a ‘row-number’ $r_v \geq 0$ specifying how ‘deep’ in the circuit it appears. Suppose now that $v \sim w$ in $D$. If they are on the same qubit, so $q_v = q_w$, then necessarily $r_v \neq r_w$. Conversely, if they are on different qubits, $q_v \neq q_w$, then they must be part of a CZ or CNOT gate, and hence $r_v = r_w$.

In the diagram resulting from Lemma 3.2, every spider arises from fusing together adjacent spiders on the same qubit line from the original diagram. For a spider $v$ in $D'$ we can thus associate two numbers $s_v$ and $t_v$, where $s_v$ is the lowest row-number of a spider fused into $v$, and $t_v$ is the highest. Spider fusion from $D$ only happens for spiders on the same qubit-line, and hence $v$ also inherits a $q_v$ from all the spiders that got fused into it. Any two spiders $v$ and $w$ in $D'$ with $q_v = q_w$ must have been produced by fusing together adjacent spiders on the same qubit-line, and hence we must have $t_v < s_w$ or $t_w < s_v$, depending on which of the spiders is most to the left. If instead $v \sim w$ and $q_v \neq q_w$, then their connection must have arisen from some CNOT or CZ gate in $D$, and hence the intervals $[s_v, t_v]$ and $[s_w, t_w]$ must have overlap, so that necessarily $s_v \leq t_v$ and $s_v \leq t_w$.

Now we let $O$ be the set of spiders connected to an output, and $I$ the set of spiders connected to an input in $D'$. For all $v \in O$ we let $f(v)$ be the unique connected spider to the right of $v$ on the same qubit-line. We define the partial order as follows: $v \prec w$ if and only if $v = w$ or $t_v < t_w$. It is straightforward to check that this is indeed a partial order.

By construction $f(v) \sim v$ and the property $v \prec f(v)$, follows from $t_v < s_{f(v)}$ discussed above, so let us look at the third property of causal flow.

Suppose $w \sim f(v)$. We need to show that $v \prec w$. If $v = w$ this is trivial so suppose $v \neq w$. First suppose that $q_w = q_{f(v)}$ (which is also equal to $q_v$). $f(v)$ has a maximum of two neighbours on the same qubit-line, and since one of them is $v$, this can only be if $w = f(f(v))$ and hence $v \prec f(v) \prec w$, and we are done. So suppose that $q_v \neq q_{f(v)}$. By the discussion above, we must then have $s_v \leq t_{f(v)}$ and $s_{f(v)} \leq t_w$. Since we also have $t_v < s_{f(v)}$ we get $t_v < s_{f(v)} \leq t_w$ so that indeed $v \prec w$. □
B.3 Preservation of focused gFlow

Throughout this section, we will rely extensively on the symmetric difference of sets: \( A \Delta B := (A \cup B) \setminus (A \cap B) \). This extends to an n-ary operation \( \Delta_i A_i \) in the obvious way. By convention, we assume these binds are far to the right as possible, i.e.

\[
\left( \bigwedge_{i \in I} A_i \right) \Delta B := \left( \bigwedge_{i \in I} (A_i \Delta B) \right)
\]

**B.3.1 Local complementation**

The following lemma will be needed in the other proofs, it shows how the odd neighbourhood of a set evolves under local complementation:

**Lemma B.8.** Given a graph \( G = (V,E) \), \( A \subseteq V \) and \( u \in V \),

\[
\text{Odd}_{G\uplus u}(A) = \begin{cases} 
\text{Odd}_G(A) \Delta (N_G(u) \cap A) \quad &\text{if } u \notin \text{Odd}_G(A) \\
\text{Odd}_G(A) \Delta (N_G(u) \setminus A) \quad &\text{if } u \in \text{Odd}_G(A)
\end{cases}
\]

**Proof.** First notice that \( \text{Odd}_G(.) \) is linear: \( \forall A,B, \, \text{Odd}_G(A \Delta B) = \text{Odd}_G(A) \Delta \text{Odd}_G(B) \). Moreover \( \forall v, \, \text{Odd}_G(\{v\}) = N_G(v) \), the neighbourhood of \( v \) in \( G \). As a consequence, \( \forall A, \text{Odd}_G(A) = \bigwedge_{v \in A} N_G(v) \).

The local complementation is acting as follows on the neighbourhoods:

\[
\forall v, N_{G\uplus u}(v) = \begin{cases} 
N_G(v) \Delta N_G(u) \Delta \{v\} \quad &\text{if } v \in N_G(u) \\
N_G(v) \quad &\text{otherwise}
\end{cases}
\]

As a consequence,

\[
\text{Odd}_{G\uplus u}(A) = \bigwedge_{v \in A} N_{G\uplus u}(v)
\]

\[
= \left( \bigwedge_{v \in A \cap N_G(u)} N_G(v) \Delta N_G(u) \Delta \{v\} \right) \Delta \left( \bigwedge_{v \in A \setminus N_G(u)} N_G(v) \right)
\]

\[
= \left( \bigwedge_{v \in A} N_G(v) \right) \Delta \left( \bigwedge_{v \in A \cap N_G(u)} N_G(u) \right) \Delta \left( \bigwedge_{v \in A \setminus N_G(u)} \{v\} \right)
\]

\[
= \text{Odd}_G(A) \Delta \left( \bigwedge_{v \in A \cap N_G(u)} N_G(u) \right) \Delta (A \cap N_G(u))
\]

Notice that \( |A \cap N_G(u)| = 1 \mod 2 \) iff \( u \in \text{Odd}_G(A) \). Hence, if \( u \notin \text{Odd}_G(A), \, \text{Odd}_{G\uplus u}(A) = \text{Odd}_G(A) \Delta (A \cap N_G(u)) \). Otherwise, if \( u \in \text{Odd}_G(A), \, \text{Odd}_{G\uplus u}(A) = \text{Odd}_G(A) \Delta N_G(u) \Delta (A \cap N_G(u)) = \text{Odd}_G(A) \Delta (N_G(u) \setminus A).

**Lemma B.9.** If \((g, \prec)\) is a focused gFlow for \((G,I,O)\) then \((g', \prec')\) is a focused gFlow for \((G \uplus u \setminus \{u\}, I, O)\) where \(g' : \overline{A} \setminus \{u\} \rightarrow 2^{\overline{\Omega}}\) is inductively defined as

\[
g'(w) := \begin{cases} 
g(w) \Delta \left( \bigwedge_{t \in R_w} g'(t) \right) \quad &\text{if } u \notin g(w) \\
g(w) \Delta \{u\} \Delta g(u) \Delta \left( \bigwedge_{t \in R_w \setminus \Delta R_w} g'(t) \right) \quad &\text{otherwise}
\end{cases}
\]

where \( R_w := N_G(u) \cap g(w) \cap \overline{\Omega} \).
Proof. If \((g, \prec)\) is a focused gFlow for \((G, I, O)\) then \((g', \prec)\) is a focused gFlow for \((G \star u \setminus u, I, O)\) where \(g' : \mathcal{O} \setminus \{u\} \to 2^\mathcal{O}\) is inductively defined as

\[
g'(w) := \begin{cases} g(w) \Delta \left( \bigcup_{t \in R_w} g'(t) \right) & \text{if } u \not\in g(w) \\ g(w) \Delta \{u\} \Delta g(u) \Delta \left( \bigcup_{t \in R_w \Delta R_w} g'(t) \right) & \text{otherwise} \end{cases}
\]

where \(R_w := N_G(u) \cap g(w) \cap \mathcal{O}\).

First of all, \(g'\) can indeed be defined inductively by starting at the maximal elements in the partial order, since for any \(t \in R_w, t \in g(w)\) so that \(w \prec t\), and similarly when \(u \in g(w), w \prec u\) so \(t \in R_u\) implies \(u \prec t\) hence \(w \prec t\). That \(g'\) preserves the order, i.e., \(v \in g'(w) \implies w \prec v\), follows similarly. Moreover, it is easy to show that for any \(w \in \mathcal{O} \setminus \{u\}\), \(g'(w) \subseteq \mathcal{I} \setminus \{u\}\). It remains to show that \(\forall w \in \mathcal{O} \setminus \{u\}, Odd_{G*\setminus u} (g'(w)) \cap \mathcal{O} = \{w\}\) or equivalently, \(\forall w \in \mathcal{O}, Odd_{G*\setminus u} (g'(w)) \cap \mathcal{O} = \{w\}\) where \(O' := O \cup \{u\}\).

First notice that for any \(w \in \mathcal{O}, u \not\in Odd_G (g(w))\), so, according to Lemma B.8 and using the linearity of Odd(\(.)\) and the distributivity of \(\Delta\) over \(\cap\),

\[
Odd_{G*\setminus u} (g(w)) \cap \mathcal{O} = Odd_G (g(w)) \cap \mathcal{O} \Delta (N_G(u) \cap g(w)) \Delta \mathcal{O} = \{w\} \Delta R_w
\]

Moreover, \(u \in Odd_G (g(u))\), so

\[
Odd_{G*\setminus u} \{u\} \Delta g(u) \cap \mathcal{O} = (N_G(u) \Delta Odd_{G*\setminus u} (g(u))) \cap \mathcal{O}
\]

(Lemma B.8)

\[
= (N_G(u) \Delta Odd_G (g(u)) \Delta (N_G(u) \setminus g(u))) \cap \mathcal{O}
\]

\[
= (Odd_G (g(u)) \cap \mathcal{O}) \Delta (N_G(u) \cap g(u) \cap \mathcal{O})
\]

\[
= N_G(u) \cap g(u) \cap \mathcal{O}
\]

\[
= R_u
\]

As a consequence,

\[
Odd_{G*\setminus u} (g'(w)) \cap \mathcal{O} = \begin{cases} \{w\} \Delta R_w \Delta \left( \bigcup_{t \in R_w} Odd_{G*\setminus u} (g'(t)) \cap \mathcal{O} \right) & \text{if } u \not\in g(w) \\ \{w\} \Delta R_w \Delta R_u \Delta \left( \bigcup_{t \in R_u \Delta R_w} Odd_{G*\setminus u} (g'(t)) \cap \mathcal{O} \right) & \text{otherwise} \end{cases}
\]

Now, we prove that \(\forall w \in \mathcal{O}, Odd_{G*\setminus u} (g'(w)) \cap \mathcal{O} = \{w\}\) by induction on \(w\) starting at maximal elements in the order.

Let \(w_0 \in \mathcal{O}\) be maximal for \(\prec\) i.e. \(\forall w \in \mathcal{O}, (w_0 \prec w)\). By maximality we must have \(g(w_0) \subseteq O'\) and hence \(R_{w_0} = \emptyset\). If \(u \not\in g(w_0)\), it implies that \(Odd_{G*\setminus u} (g'(w_0)) \cap \mathcal{O} = \{w_0\}\). If \(u \in g(w_0)\), then \(w_0 \prec u\), so \(u\) is maximal, so \(g(u) \subset O\) and \(R_u = \emptyset\), as a consequence \(Odd_{G*\setminus u} (g'(w)) \cap \mathcal{O}\) is also equal to \(\{w_0\}\) in this case.

By induction, for any \(w \in \mathcal{O}\),

\[
Odd_{G*\setminus u} (g'(w)) \cap \mathcal{O} = \{w\} \Delta R_w \Delta \left( \bigcup_{t \in R_w} Odd_{G*\setminus u} (g'(t)) \cap \mathcal{O} \right)
\]

\[
(IH) = \{w\} \Delta R_w \Delta \left( \bigcup_{t \in R_w} \{t\} \right)
\]

\[
= \{w\} \Delta R_w \Delta R_w
\]

\[
= \{w\}
\]
If \( u \in g(w) \),
\[
\text{Odd}_{G^w} (g'(w)) \cap \Omega \ = \ \{ w \} \ {\Delta} R_w \ {\Delta} R_u \ {\Delta} \left( \ {\Delta}_{t \in R_u \Delta R_w} \ \text{Odd}_{G^w} (g'(t)) \cap \Omega \right).
\]
\[
(\text{IH}) \ = \ \{ w \} \ {\Delta} R_w \ {\Delta} R_u \ {\Delta} \left( \ {\Delta}_{t \in R_u \Delta R_w} \ \{ t \} \right).
\]
\[
\ = \ \{ w \} \ {\Delta} R_w \ {\Delta} R_u \ {\Delta} R_u \ {\Delta} R_w.
\]
\[
\ = \ \{ w \}
\]
\[
\Rightarrow \text{Remark B.10.} \text{ If } (g, \prec) \text{ is a focused gFlow for } (G, I, O) \text{ then } (g', \prec) \text{ is a gFlow for } (G \ast u \setminus u, I, O) \text{ where } g' : \Omega \setminus \{ u \} \to 2^T \{ u \} \text{ is defined as }
\]
\[
g'(w) := \begin{cases} \ g(w) & \text{if } u \notin g(w) \\ \ g(w) \ {\Delta} \{ u \} \ {\Delta} g(u) & \text{otherwise} \end{cases}
\]
Notice that \((g', \prec)\) is not necessarily a focused gFlow.

### B.3.2 Pivoting

The action of the pivoting on the odd-neighbourhood can be described as follows:

**Lemma B.11.** Given a graph \( G = (V, E), A \subseteq V, u \in V, \) and \( v \in N_G(u) \),
\[
\text{Odd}_{G \wedge uv} (A) = \begin{cases} \text{Odd}_G (A) & \text{if } u, v \notin \text{Odd}_G [A] \\ \text{Odd}_G (A) \ {\Delta} N_G [v] & \text{if } u \in \text{Odd}_G [A], v \notin \text{Odd}_G [A] \\ \text{Odd}_G (A) \ {\Delta} N_G [u] & \text{if } u \notin \text{Odd}_G [A], v \in \text{Odd}_G [A] \\ \text{Odd}_G (A) \ {\Delta} N_G [u] \ {\Delta} N_G [v] & \text{if } u, v \in \text{Odd}_G [A] \end{cases} (13)
\]

where \( N_G[u] := N_G(u) \ {\Delta} \{ u \} \) and \( \text{Odd}_G [A] := \text{Odd}_G (A) \ {\Delta} A \) are the closed neighbourhood and the closed odd neighbourhood respectively.

**Proof.** The pivoting acts as follows on neighbourhoods:

\[
\forall w, N_{G \wedge uv} (w) = \begin{cases} N_G(w) \ {\Delta} N_G[u] \ {\Delta} N_G[v] & \text{if } w \in N_G[u] \cap N_G[v] \\ N_G(w) \ {\Delta} N_G[v] & \text{if } w \in N_G[u] \ \setminus \ N_G[v] \\ N_G(w) \ {\Delta} N_G[u] & \text{if } w \in N_G[v] \ \setminus \ N_G[u] \\ N_G(w) & \text{otherwise} \end{cases}
\]

As a consequence,
\[
\text{Odd}_{G \wedge uv} (A) = \ {\Delta}_{w \in A} \ N_{G \wedge uv} (w)
\]
\[
\ = \ \left( \ {\Delta}_{w \in A \setminus N_G[u] \ \setminus N_G[v]} \ N_{G \wedge uv} (w) \right) \ {\Delta} \left( \ {\Delta}_{w \in A \setminus N_G[u] \ \setminus N_G[v]} \ N_{G \wedge uv} (w) \right) \ {\Delta}
\]
\[
\ = \ \left( \ {\Delta}_{w \in A \setminus N_G[u] \ \setminus N_G[v]} \ N_G(w) \ {\Delta} N_G[u] \ {\Delta} N_G[v] \right) \ {\Delta} \left( \ {\Delta}_{w \in A \setminus N_G[u] \ \setminus N_G[v]} \ N_G(w) \ {\Delta} N_G[u] \ {\Delta} N_G[v] \right)
\]
\[
\ = \ \text{Odd}_G (A) \ {\Delta} \left( \ {\Delta}_{w \in A \setminus N_G[u]} \ N_G[u] \right) \ {\Delta} \left( \ {\Delta}_{w \in A \setminus N_G[v]} \ N_G[v] \right)
\]

Circuit Simplification with the ZX-calculus
Notice that $|A \cap N_G[u]| = 1 \mod 2$ iff $u \in \text{Odd}_G[A]$, and similarly for $v$. Hence we obtain (13) by case distinction.

Lemma B.12. If $(g, \prec)$ is a focused gFlow for $(G, I, O)$ then $(g', \prec)$ is a focused gFlow for $(G \land uv \setminus \{u, v\}, I, O)$ where $\forall w \in \mathcal{O} \setminus \{u, v\}$, $g'(w) := g(w) \setminus \{u, v\}$.

Proof. If $(g, \prec)$ is a focused gFlow for $(G, I, O)$ then $(g', \prec)$ is a focused gFlow for $(G \land uv \setminus uv, I, O)$ where $g'$ is defined as follows: $\forall w \in \mathcal{O} \setminus \{u, v\}$

$$g'(w) := g(w) \setminus \{u, v\}$$

It remains to show that $\forall w \in \mathcal{O} \setminus \{u, v\}$, Odd$_{G \land uv \setminus uv}(g'(w)) \cap \mathcal{O} = \{w\}$, or equivalently, $\forall w \in \mathcal{O} \setminus \{u, v\}$, Odd$_{G \land uv \setminus uv}(g'(w)) \cap \mathcal{O} = \{w\}$ where $O' := O \cup \{u, v\}$.

Let $w \in \mathcal{O}$. Note that Odd$_G[g(w)] = $ Odd$_G(g(w)) \Delta g(w) = \{w\} \Delta g(w) = g(w) \cup \{w\}$, and thus $u \in $ Odd$_G[g(w)] \iff u \in g(w)$ and similarly for $v$. Hence:

- If $u, v \in g(w)$,
  Odd$_{G \land uv \setminus uv}(g'(w)) \cap \mathcal{O}' = $ Odd$_{G \land uv \setminus uv}(g(w) \Delta \{u, v\}) \cap \mathcal{O}'$
  $= $ Odd$_{G \land uv \setminus uv}(g(w)) \cap \mathcal{O}' \Delta $ Odd$_{G \land uv \setminus uv}((\{u, v\}) \cap \mathcal{O}'$
  $= $ Odd$_G(g(w)) \cap \mathcal{O}' \Delta $ Odd$_G((\{u, v\}) \cap \mathcal{O}'$
  $= $ Odd$_G(g(w)) \cap \mathcal{O}' \Delta $ Odd$_G((\{u, v\}) \cap \mathcal{O}'$
  $= \{w\}$

- If $u \in g(w)$ and $v \notin g(w)$,
  Odd$_{G \land uv \setminus uv}(g'(w)) \cap \mathcal{O}' = $ Odd$_{G \land uv \setminus uv}(g(w) \Delta \{u\}) \cap \mathcal{O}'$
  $= $ Odd$_{G \land uv \setminus uv}(g(w)) \cap \mathcal{O}' \Delta $ Odd$_{G \land uv \setminus uv}((\{u\}) \cap \mathcal{O}'$
  $= $ Odd$_G(g(w)) \cap \mathcal{O}' \Delta $ Odd$_G((\{u\}) \cap \mathcal{O}'$
  $= \{w\}$

Here we have used that Odd$_{G \land uv \setminus uv}((\{u\}) = N_G(v)$ and that $N_G[v] \cap \mathcal{O}' = N_G(v) \cap \mathcal{O}'$.

- If $u \notin g(w)$ and $v \in g(w)$ we prove it similarly to the previous case.

- If $u, v \notin g(w)$,
  Odd$_{G \land uv \setminus uv}(g'(w)) \cap \mathcal{O}' = $ Odd$_{G \land uv \setminus uv}(g(w)) \cap \mathcal{O}'$
  $= $ Odd$_G(g(w)) \cap \mathcal{O}'$
  $= \{w\}$
B.4 ZX reduction rules

▷ Lemma 5.2.

\[
\begin{align*}
\alpha_1 & - \pi & \alpha_2 & \cdots & \alpha_{n-1} & - \pi & \alpha_n \\
\alpha_1 & - \pi & \alpha_2 & \cdots & \alpha_{n-1} & - \pi & \alpha_n
\end{align*}
\]

Proof. First of all, we will need the following equation:

\[
\begin{align*}
\begin{array}{ccc}
\frac{\pi}{2} & = & \frac{\pi}{2} \\
\frac{\pi}{2} & = & -\frac{\pi}{2} \\
\frac{\pi}{2} & = & -\frac{\pi}{2} \\
\frac{\pi}{2} & = & -\frac{\pi}{2} \\
\frac{\pi}{2} & = & -\frac{\pi}{2}
\end{array}
\end{align*}
\]  
(14)

We pull out all of the phases via \((f)\) then apply the local complementation rule \((5)\):

Thanks to equation \((14)\), the topmost spider in the RHS above becomes an X-spider, with phase \(\mp \pi/2\), which is canceled out by \(X_{\pm \pi/2}\) gate directly below it. The resulting (phaseless) X-spider copies and fuses with the neighbours:

▷ Lemma 5.3.

\[
\begin{align*}
\begin{array}{ccc}
\alpha_1 & + \pi & \alpha_2 & \cdots & \alpha_{n-1} & + \pi & \alpha_n \\
\alpha_1 & + \pi & \alpha_2 & \cdots & \alpha_{n-1} & + \pi & \alpha_n
\end{array}
\end{align*}
\]

Proof. We pull out all of the phases via \((f)\) then apply the pivoting rule \((6)\):
We then apply the colour-change rule to turn the Z-spiders with phases \(j\pi\) and \(k\pi\) into X-spiders, which copy and fuse with the neighbours of the original two vertices:

\[
\begin{align*}
\alpha_1 & \ldots & \alpha_n \\
\beta_1 & \ldots & \beta_n \\
\gamma_1 & \ldots & \gamma_n
\end{align*}
\]

\[
\begin{align*}
\pi & \ldots & \pi \\
\beta_1 & \ldots & \beta_n \\
\gamma_1 & \ldots & \gamma_n
\end{align*}
\]

\[
\begin{align*}
\alpha_1 & \ldots & \alpha_n \\
\beta_1 & \ldots & \beta_n \\
\gamma_1 & \ldots & \gamma_n
\end{align*}
\]

\[
\begin{align*}
\pi & \ldots & \pi \\
\beta_1 & \ldots & \beta_n \\
\gamma_1 & \ldots & \gamma_n
\end{align*}
\]

\[\vdash \text{Proposition (6.2).} \quad \text{The following equation holds.}\]

\[
\begin{align*}
M & = \\
M' & = \end{align*}
\]

\[\vdash \text{Proposition (6.2).} \quad \text{The following equation holds.}\]

\[
\begin{align*}
M & = \\
M' & = \end{align*}
\]

\[\vdash \text{Proposition (6.2).} \quad \text{The following equation holds.}\]

\[
\begin{align*}
M & = \\
M' & = \end{align*}
\]

\[\vdash \text{Proposition (6.2).} \quad \text{The following equation holds.}\]

\[
\begin{align*}
M & = \\
M' & = \end{align*}
\]

\[\vdash \text{Proposition (6.2).} \quad \text{The following equation holds.}\]
As we are only adding vertices at the outputs, it should be clear how the gFlow can be extended to incorporate these new vertices.

Now let $A$ denote the set of vertices connected to the top vertex, but not to the vertex beneath it, $B$ the set of vertices connected to both, and $C$ the vertices connected only to the bottom one. Further restricting our view of the diagram to just these two lines, we see that we can apply a pivot rewrite as in [S]:

By Corollary 5.5 this rewrite preserves focused gFlow. Looking at the connectivity matrix, it is straightforward to see that the matrix $M$ has now been changed in exactly the way described.
Algorithm 1 Extraction algorithm

1: procedure Extract(D) > input is graph-like diagram D
2: Init empty circuit C
3: G, I, O ← Graph(D) > get the underlying graph of D
4: F ← O > initialise the frontier
5: for v ∈ F do
6: if v connected to output by Hadamard then
7: C ← Hadamard(Qubit(v))
8: Remove Hadamard from v to output
9: if v has non-zero phase then
10: C ← Phase-gate(Phase(v), Qubit(v))
11: for edge between v and w in F do
12: C ← CZ(Qubit(v), Qubit(w))
13: Remove edge between v and w
14: while ∃ v ∈ D \ F do > there are still vertices to be processed
15: D, F, C ← UpdateFrontier(D, F, C) > the only vertices still in D are in F
16: for v ∈ F do
17: if v connected to input via Hadamard then
18: C ← Hadamard(Qubit(v))
19: Perm ← Permutation of Qubits of F to qubits of inputs
20: for swap(q1, q2) in PermutationAsSwaps(Perm) do
21: C ← swap(q1, q2)
22: return C
23: procedure UpdateFrontier(D, F, C)
24: N ← Neighbours(F)
25: M ← Biadjacency(F, N)
26: M′ ← GaussReduce(M)
27: Init ws > initialise empty set ws
28: for row r in M′ do > there is a single 1 on row r
29: if sum(r) == 1 then > w will be part of the new frontier
30: Set w to vertex corresponding to nonzero column of r
31: Add w to ws > smaller biadjacency matrix
32: M ← Biadjacency(F, ws)
33: for (r1, r2) ∈ GaussRowOperations(M) do
34: C ← CNOT(Qubit(r1), Qubit(r2))
35: Update D based on row operation
36: for w ∈ ws do > all w now have a unique neighbour in F
37: v ← Unique neighbour of w in F
38: C ← Hadamard(Qubit(v))
39: C ← Phase-gate(Phase(w), Qubit(v))
40: Remove phase from w
41: Remove v from F
42: Add w to F
43: for edge between w1 and w2 of ws do
44: C ← CZ(Qubit(w1), Qubit(w2))
45: Remove edge between w1 and w2
46: return D, F, C