**Veriﬁcation and the ZX-Calculus**

The Coq Proof Assistant

The Coq proof assistant is a proof assistant and programming language for formal veriﬁcation. The methods of formal veriﬁcation have been used to verify optimizing compilers to guarantee they are bug free. The veriﬁed quantum circuit optimizer VQOC [1] uses the Coq proof assistant to verify that the optimizations it performs do not change the circuit semantics.

The ZX Calculus

The ZX calculus is an alternative approach to representing quantum programs. The objects of the ZX calculus are the Z spiders (green) and X spiders (red) to the right, which generalize the quantum mechanical Z and X rotations on qubits. They are called “spiders” as they can have any number of input or output wires which gives them a spider-like appearance. The ZX calculus comes paired with rewrite rules which are known to preserve semantics. Tools like PyZX[2] take advantage of these rules in order to create quantum circuit optimizers.

Verifying ZX

Inspired by both VQOC and PyZX, we set out to create VyZX: a veriﬁed library for working with ZX diagrams. VyZX is the ﬁrst step towards creating a fully veriﬁed PyZX-style ZX diagram optimizer. We aim to make VyZX general enough that it can apply to other domains related to the ZX calculus, including lattice surgery and quantum circuit simulation.

**Drawing Inspiration**

By looking at how many ways we can construct string diagrams, we have reduced them to a small number of inductive constructors:

1. The unit object, which is the empty diagram,
2. The single wire,
3. Morphisms, which take n inputs to m outputs,
4. Braids, which swap two wires,
5. Sequential composition, which composes two diagrams in sequence, and
6. Tensor products, which arrange two diagrams in parallel.

![Diagram](image)

Figure 2. Composition, tensor product, braid, cap, and cup for symmetric monoidal string diagrams.

To obtain the full ZX calculus we will have two morphisms, the Z and X spider.

**Inductively Drawing Diagrams**

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty</td>
<td>⊥</td>
</tr>
<tr>
<td>Cap</td>
<td>⊖</td>
</tr>
<tr>
<td>Swap</td>
<td>⊗</td>
</tr>
<tr>
<td>Compose</td>
<td>⊗</td>
</tr>
<tr>
<td>Stack</td>
<td>⊕</td>
</tr>
</tbody>
</table>

**Diagram Semantics**

To give our inductive diagrams a semantics we use the veriﬁed library QuantumLib and transform our diagrams into QuantumLib matrices.

![Diagram](image)

Figure 3. The inductive constructors for block representation ZX diagrams.

From these constructors we then deﬁne Wire as a notation for Z_spider 1 1 0.

**Example: Untangling Nots**

Definition CNOT_R :=

$$\begin{array}{c}
\left(\text{Z} \text{ \_spider} 1 2 0 \downarrow \downarrow \downarrow\right) \\
\left(\text{X} \text{ \_spider} 2 1 0\right)
\end{array}$$

Definition CNOT_L :=

$$\begin{array}{c}
\left(\text{X} \text{ \_spider} 2 1 0\downarrow \downarrow \downarrow\right) \\
\left(\text{Z} \text{ \_spider} 1 2 0\downarrow \downarrow \downarrow\right)
\end{array}$$

Lemma CNOT_R_PROP_L :

$$\text{sem}(\text{CNOT}_R) = \text{sem}(\text{CNOT}_L)$$

Performing the ﬁnal reduction step and turning the diagrams into matrices we get for the left hand side

$$\begin{array}{c}
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}
\otimes
\begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix}
\otimes
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}
\otimes
\begin{bmatrix}
0 & 0 \\
0 & 1
\end{bmatrix}
\end{array}$$

and for the right hand side

$$\begin{array}{c}
\begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}
\otimes
\begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix}
\otimes
\begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0
\end{bmatrix}
\end{array}$$

Using the proof automation available to QuantumLib, this equality can be solved automatically, and will look like this:

**References**
