VyZX: Verifying the ZX Calculus

Adrian Lehmann **Ben Caldwell** Robert Rand

University of Chicago

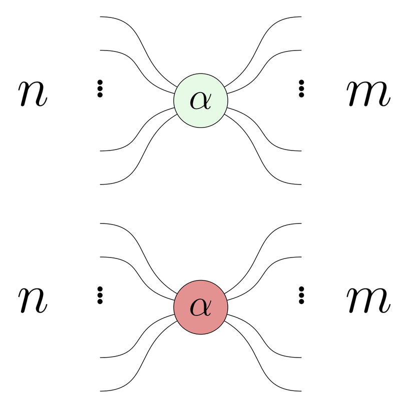
Verification and the ZX-Calculus

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 VOQC [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 a 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 appearence. 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 VOQC 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.

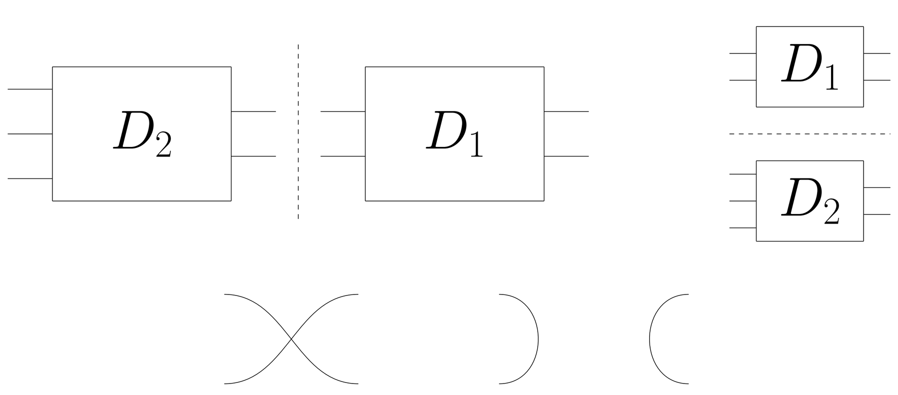


Figure 2. Composition, tensor product, braid, cap, and cup for symmetric monoidal string diagrams.

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

Inductively Drawing Diagrams

$$\frac{\text{in out} : \mathbb{N} \qquad \alpha : \mathbb{R}}{\texttt{Z Spider}\ \text{in out}\ \alpha : \texttt{ZX}\ \text{in out}}$$

$$\frac{\text{in out} : \mathbb{N} \qquad \alpha : \mathbb{R}}{\texttt{X Spider}\ \text{in out}\ \alpha : \texttt{ZX}\ \text{in out}}$$

$$\frac{}{\texttt{Cap} : \texttt{ZX}\ 0\ 2} \qquad \frac{}{\texttt{Cup} : \texttt{ZX}\ 2\ 0}$$

$$\frac{}{\texttt{Swap} : \texttt{ZX}\ 2\ 2} \qquad \frac{}{\texttt{Empty} : \texttt{ZX}\ 0\ 0}$$

$$\frac{\text{zx1} : \texttt{ZX}\ \text{in mid} \qquad \text{zx2} : \texttt{ZX}\ \text{mid out}}{\texttt{Compose}\ \text{zx1 zx2} : \texttt{ZX}\ \text{in out}}$$

$$\frac{\text{zx1} : \texttt{ZX}\ \text{in1 out1} \qquad \text{zx2} : \texttt{ZX}\ \text{in2 out2}}{\texttt{Stack}\ \text{zx1 zx2} : \texttt{ZX}\ (\text{in1} + \text{in2})\ (\text{out1} + \text{out2})}$$

Figure 3. The inductive constructors for block representation ZX diagrams

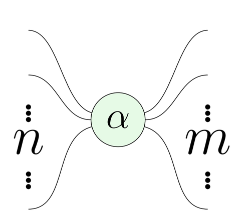
From these constructors we then define Wire as a notation for `Z_spider 1 1 0`.

Constructor	Notation
Empty	⦰
Cap	⊂
Cup	⊃
Swap	×
Compose	⟷
Stack	↕
Wire	—

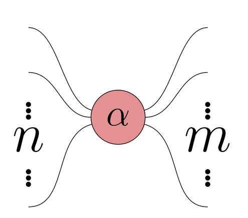
Figure 4. Coq notations for our diagrammatic constructors.

Diagram Semantics

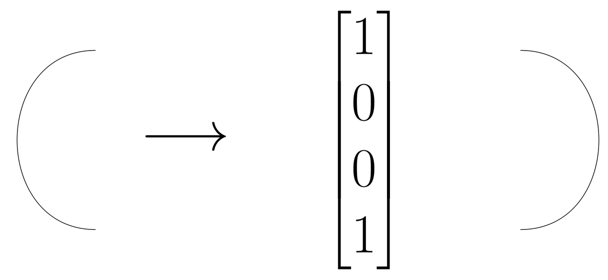
To give our inductive diagrams a semantics we use the verified library `QuantumLib` and transform our diagrams into `QuantumLib` matrices.



$$\longrightarrow \quad \begin{array}{cc} & 2^n \\ 2^m & \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & e^{i\alpha} \end{bmatrix} \end{array}$$



$$\longrightarrow \quad H^{\otimes m} \times \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & e^{i\alpha} \end{bmatrix} \times H^{\otimes n}$$



$$\longrightarrow \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad \longrightarrow \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}$$

$$\boxed{D_1} \longleftrightarrow \boxed{D_2} \longrightarrow \texttt{sem}(D_2) \times \texttt{sem}(D_1)$$

$$\boxed{D_1} \updownarrow \boxed{D_2} \longrightarrow \texttt{sem}(D_1) \otimes \texttt{sem}(D_2)$$

Figure 5. The `semantics` function for giving semantics to inductively defined ZX diagrams.

Using these semantics we define a relation $\propto$ and say that $zx_1 \propto zx_2$ if and only if

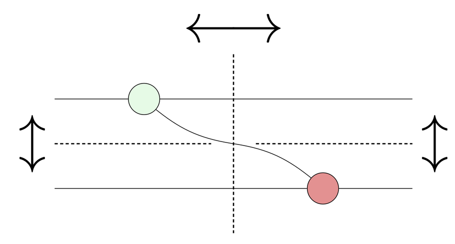
$$\exists c \in \mathbb{C} : c \neq 0 \land \texttt{sem}(zx_1) = c \cdot \texttt{sem}(zx_2)$$

this relation will allow us to define, prove, and use the ZX calculus rewrite rules.

Example: Untangling Nots

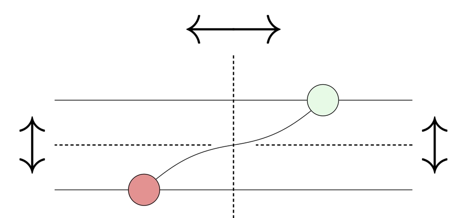
Definition `CNOT_R` :=

(`Z_spider 1 2 0` ↕ —) ⟷
(— ↕ `X_spider 2 1 0`)



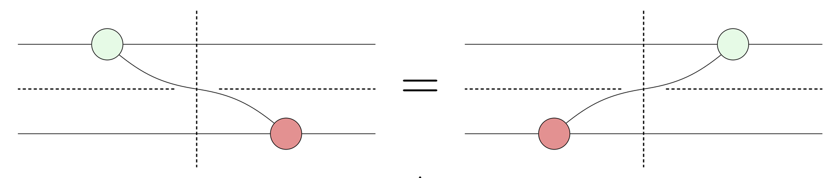
Definition `CNOT_L` :=

(— ↕ `X_spider 2 1 0`) ⟷
(`Z_spider 1 2 0` ↕ —)

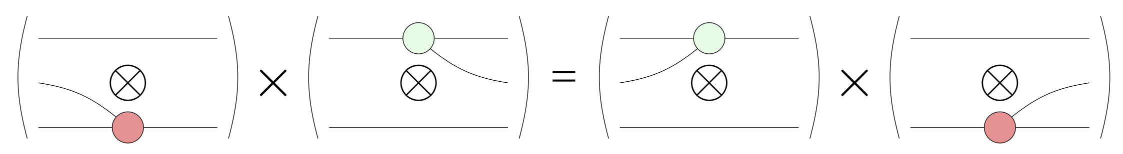


Lemma `CNOT_R_PROP_L` :

`sem(CNOT_R)` = `sem(CNOT_L)`



↓evaluation



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

$$\left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \left(H \times \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times (H \otimes H)\right)\right) \times \left(\begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right)$$

and for the right hand side

$$\left(\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right) \times \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \left((H \otimes H) \times \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \times H\right)\right)$$

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

```
Lemma CNOT_R_EQ_L :
    ZX_semantics ZX_CNOT_R =
    ZX_semantics ZX_CNOT_L.
Proof.
  simpl.
  rewrite wire_identity_semantics.
  unfold_spider.
  autorewrite with Cexp_db.
  solve_matrix.
Qed.
```

References

[1] Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. A verified optimizer for quantum circuits. *Principles of Programming Languages*, 2021.

[2] Aleks Kissinger and John van de Wetering. PyZX: Large scale automated diagrammatic reasoning. In *Quantum Physics and Logic*, 2019.

caldwellb@uchicago.edu