VyZX: Formal Verification of a Graphical Quantum Language

Lehmann, Caldwell, Shah, Rand (UChicago)

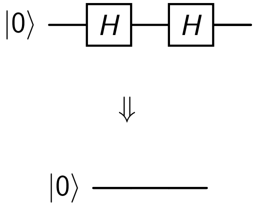
Presented by Wil Cram

November 5, 2025

Outline

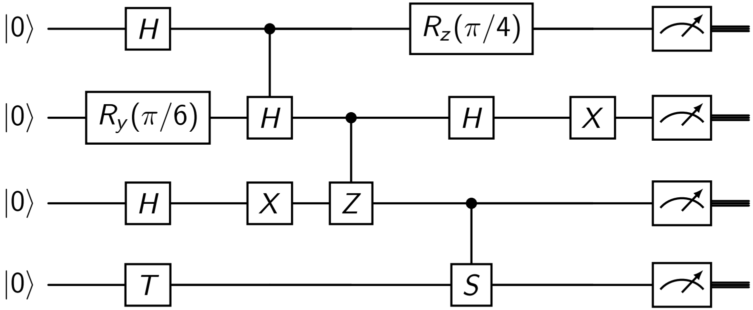
Motivation

This circuit is easy enough to optimize...



Motivation

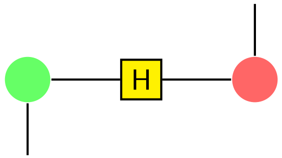
What about this one?



Motivation

We want a form for these circuits that is easier to reason about when doing compiler optimizations

The ZX-Calculus is such a form that has a minimalist structure



The main contribution of this paper is **VyZX**, a tool for doing formal proofs in this ZX-Calculus

Why Should You Actually Care?

The ZX-Calculus is a tool for reasoning about program semantics

This means we can use it for:

1. Circuit Optimization
2. Formal Verification
3. Equality Validation

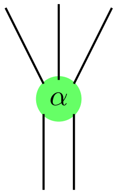
The PyZX[1] library can translate Quipper circuits to a ZX diagram, optimize in this domain, then translate back to a quantum circuit

[1] https://github.com/zxcalc/pyzx

Outline

ZX Calculus - Intro

ZX Diagrams are undirected graphs that contain two kinds of nodes, the first is a **Z-Spider**:

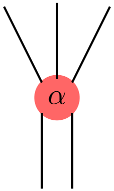


These have *n* input wires and *m* output wires, and represent a matrix

$$|0\rangle^{\otimes m}\langle 0|^{\otimes n} + e^{i\alpha}|1\rangle^{\otimes m}\langle 1|^{\otimes n}$$

ZX Calculus - Intro

X-Spiders do the same thing, but in the Hadamard basis:



$$|+\rangle^{\otimes m}\langle+|^{\otimes n} + e^{i\alpha}|-\rangle^{\otimes m}\langle-|^{\otimes n}$$

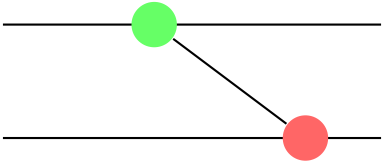
A blank node means there is zero phase.

ZX Calculus - Examples

The (1,1) Z and X spiders with phase π are just the Z and X gates

The (0,2) Z spider with phase 0 is (up to normalization) the bell state

The CNOT gate looks something like this:



We can draw it shifted either way due to an important principle:
Only Connectivity Matters

Soundness and Completeness

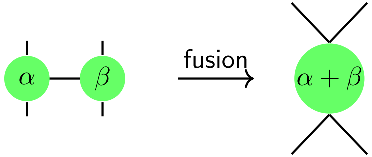
A deep result (which took a decade to prove!) says that ZX-Calculus is sound and complete with respect to the category **Qubit**, which has objects $\bigotimes_n \mathbb{C}^2$ for all n

This means that any theorem we can prove in **Qubit** holds in ZX-Calculus and vice versa

But how do we prove things in ZX-Calculus?

Rewriting Rules - Fusion

Given two adjacent spiders of the same color, we can fuse them together by adding their phases:

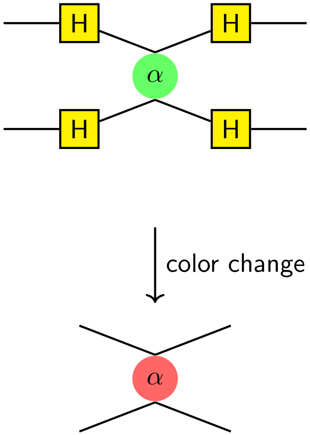


The reverse process also holds: we can break spiders apart into spiders of smaller angles

Rewriting Rules - Basis Change

Hadamards are not a primitive notion in ZX-Calculus; they can be constructed by composing Z and X spiders

We can do a change-of-basis procedure to switch the color of a spider:



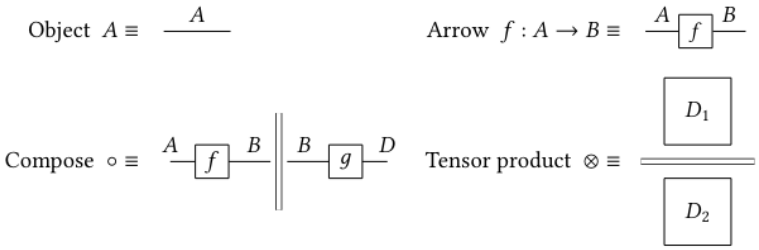
Example: 3 CNOTs $\cong$ *SWAP*

Thanks to Fabrizio Genovese!

ZX Calculus - Abstraction

After composing a bunch of spiders, we can encapsulate the ZX-Diagram as a morphism in **Qubit** (a matrix)

We have ways to combine these morphisms in serial and in parallel: notice the correspondence between the graphical manipulations and the related linear-algebraic operations



Outline

Aside - Theorem Provers

Theorem Provers are programming languages that allow one to prove theorems by writing programs

They can be used for both mathematical theorems (Lean[2]) and formal verification of programs (Rocq[3]).

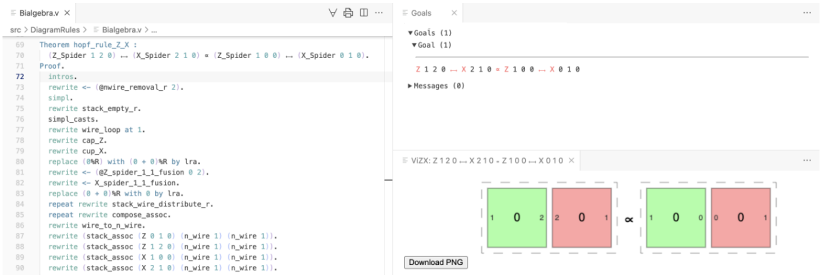
Rocq is *interactive* in the sense that you can step through your proof and watch the "proof state" change over time

[2] https://leanprover-community.github.io/mathlib-overview.html
[3] https://rocq-prover.org/

VyZX

VyZX is a library for Rocq that allows for proving theorems in the ZX-Calculus

The authors have also written an extension (ZXViz) which turns the (often complicated) textual proof state into a ZX-Diagram


Bialgebra.v
src > DiagramRules > Bialgebra.v > ...
Goals
ViZX: Z 1 2 0 ⟷ X 2 1 0 ∝ Z 1 0 0 ⟷ X 0 1 0
1 0 2
2 0 1
∝
1 0 0
0 0 1
Download PNG

```
Theorem hopf_rule_Z_X :
  (Z_Spider 1 2 0) ⟷ (X_Spider 2 1 0) ∝ (Z_Spider 1 0 0) ⟷ (X_Spider 0 1 0).
Proof.
  intros.
  rewrite <- (@nwire_removal_r 2).
  simpl.
  rewrite stack_empty_r.
  simpl_casts.
  rewrite wire_loop at 1.
  rewrite cap_Z.
  rewrite cup_X.
  replace (0%R) with (0 + 0)%R by lra.
  rewrite <- (@Z_spider_1_1_fusion 0 2).
  rewrite <- X_spider_1_1_fusion.
  replace (0 + 0)%R with 0 by lra.
  repeat rewrite stack_wire_distribute_r.
  repeat rewrite compose_assoc.
  rewrite wire_to_n_wire.
  rewrite (stack_assoc (Z 0 1 0) (n_wire 1) (n_wire 1)).
  rewrite (stack_assoc (Z 1 2 0) (n_wire 1) (n_wire 1)).
  rewrite (stack_assoc (X 1 0 0) (n_wire 1) (n_wire 1)).
  rewrite (stack_assoc (X 2 1 0) (n_wire 1) (n_wire 1)).
```

```
▼Goals (1)
 ▼Goal (1)
 ________________________________________
 Z 1 2 0 ⟷ X 2 1 0 ∝ Z 1 0 0 ⟷ X 0 1 0
▶Messages (0)
```

Internal Representation - Diagrams

In Rocq, we can inductively define a datatype for ZX-diagrams with `in` inputs and `out` outputs, called `ZX in out`

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

$$\frac{\emptyset}{\texttt{Wire} : \texttt{ZX}\ 1\ 1}$$

$$\frac{\texttt{f:}\quad \texttt{ZX in mid} \qquad \texttt{g:}\quad \texttt{ZX mid out}}{\texttt{Compose f g:}\quad \texttt{ZX in out}}$$

Internal Representation - Matrices

We can again define a **denotation** of our ZX-diagrams, with

$$[[.]] : \texttt{ZX n m} \to \mathbb{C}^{2^m \times 2^n}$$

$$[[\texttt{Z n m } \alpha]] = |0\rangle^{\otimes m} \langle 0|^{\otimes n} + e^{i\alpha} |1\rangle^{\otimes m} \langle 1|^{\otimes n}$$

$$[[\texttt{Wire}]] = I_2$$

$$[[\texttt{Compose f g}]] = [[g]][[f]]$$

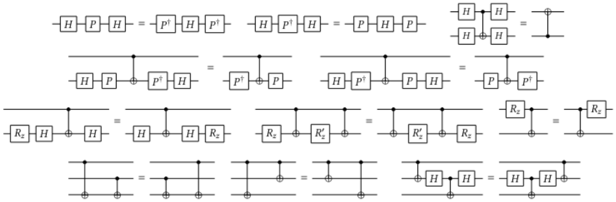
One corollary of soundness is that applying the rewriting rules never changes the denotation

Translation and Results

VyZX assumes that circuits are written using $H, X, R_z(\alpha), CNOT$, and translates these gates to corresponding spiders

The stacking and composition rules can then be used to put the whole circuit together as a ZX-Diagram

The authors formally verified the bell state preparation circuit, as well as the below set of peephole circuit optimizations



Using VyZX

The rewrite rules from before are encoded as Rocq lemmas which we can use to advance the proof

The syntactic constructions of ZX-Diagrams give an implicit associativity to the diagrams

Keeping with the ”only connectivity matters” principle, the authors provide convenience lemmas to re-associate the syntax trees

However, this is pretty time-consuming: authors find that 27% of all of their tactic applications are just re-association!

DC $\Updownarrow$ *AC*

DC $\Updownarrow$ *AC* is a tool that checks for equality of two syntactic ZX-Diagrams

It works by rewriting one side of the proposed equality using the re-association lemmas in a kind of tree search, while using another method (E-Graphs) to prevent the search space from blowing up

The authors are not yet capable of extracting a Rocq proof in all cases from the tool, but this is being actively worked on

Other Features

ZX Calculus has the property that any equality of diagrams also holds when you swap all of the colors

The authors provide a `colorswap` tactic which automatically proves the color-swapped lemma from the original one, and a similar tactic for transposed lemmas

VyZX has support for inductive proofs, both over the number of inputs and outputs

In this case, the inductive hypothesis becomes a rewriting rule on a subdiagram of the proof state

Outline

Contributions

VyZX, a library in Rocq for working with ZX-Calculus

ZXViz, a tool to visualize proof state as a ZX-Diagram

$DC \Updownarrow AC$, an automation to show equality of syntactic ZX-Diagrams

Formal verification example of some compiler optimizations

Future Work

Incorporation of ZX-Calculus extensions into VyZX

One example of this is support for postselection

Verification of surface codes and proving equivalence of different codes

Extension of VyZX to work with categories other than **Qubit**

Takeaways

The ZX Calculus is a useful tool for working with quantum programs, especially for optimization

There is a lot of work to be done in formally verifying quantum algorithms

Incorporating visual elements into proof assistants can make using them much easier

Further Reading

"ZX-Calculus for the working quantum computer scientist," van de Wetering, 2020, https://arxiv.org/pdf/2012.13966

"VyZX: A vision for verifying the ZX calculus", 2022, https://arxiv.org/pdf/2205.05781

"PyZX: Large Scale Diagrammatic Automated Reasoning", 2020, https://arxiv.org/pdf/1904.04735v2