Visualizing Graphical Proofs in Coq

Bakti Shah
University of Chicago

The ZX-calculus

The ZX-calculus comprises of a set of rewrite rules for manipulation of ZX diagrams, a diagrammatic representation of quantum operations. The ZX-calculus is an example of a *symmetric monoidal category*.

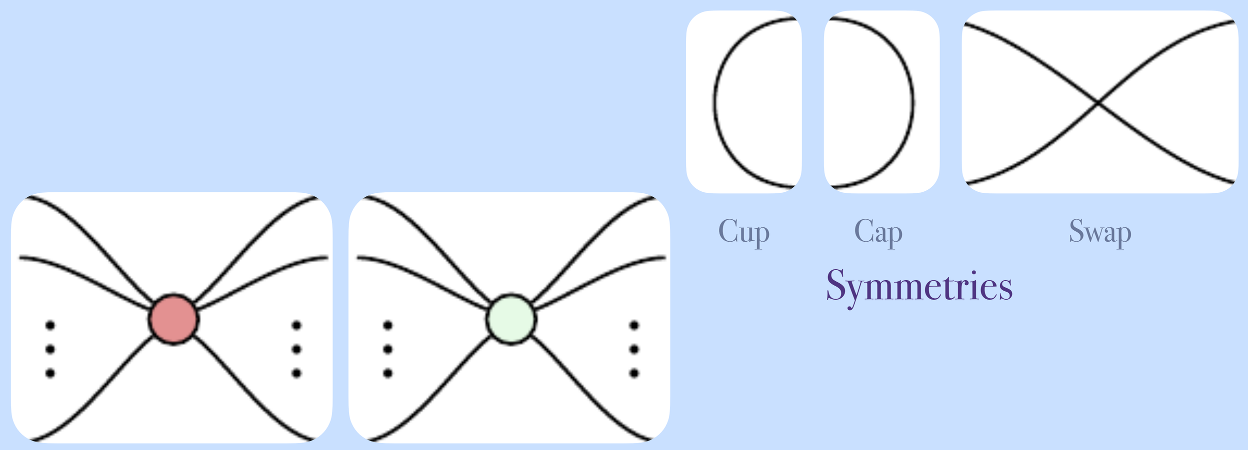
ZX diagrams are graphs that consist of red and green nodes, called Z and X spiders respectively. Each spider has a number of inputs and outputs (dimensions), as well as a rotational angle. Spiders can be connected via edges.

The ZX-calculus has two important rules:
> only *connectivity* matters. Wires can be arbitrarily deformed as long as the input and output order to the overall diagram is maintained.
> swapping red and green *everywhere* preserves the truth of a rule.

Inductive Definition

To verify the ZX-calculus formally, the language had to be fit to a format easy to reason about in a proof assistant. Inductively defined types and induction tactics were the de-facto choice. Hence, the core language is is an inductive structure representing string diagrams & contributing, amongst other things, a set of base morphisms and the ability to compose diagrams sequentially (horizontally) and in parallel (vertically).This representation does have its drawbacks, and is less direct than the standard graphical representation: a large amount of graphical information is shoe-horned into a single inductive structure.

Building Blocks of ZX-Diagrams



X spider   Z spider

Cup   Cap   Swap

Symmetries

Inductive Constructors

These are the constructors we use to form our inductively defined ZX diagrams. A diagram is parameterized over its inputs and outputs, so any valid diagram has type `ZX a b`, where `a,b` ∈ ℕ. The constructors include the Z and X spiders, operations for horizontal composition and vertical stacking, symmetries (swaps, caps, and cups), the hadamard box, the identity wire, and the empty diagram. Additionally, we provide a function to explicitly cast diagrams to have a different number of inputs and outputs, when given proofs of equivalence of current and desired dimensions. This set of constructors ensures readability and simplicity of proofs; we must explicitly consider symmetries as constructors as proofs often reason about them.

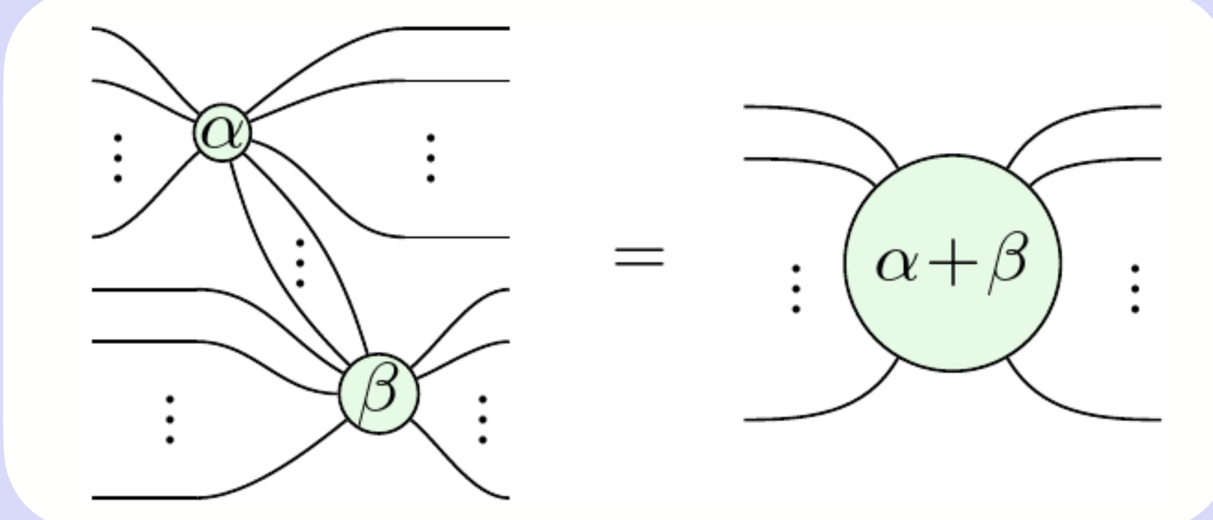
$$\frac{\text{in out} : \mathbb{N} \qquad \alpha : \mathbb{R}}{\texttt{Z}\ \text{in out}\ \alpha : \texttt{ZX}\ \text{in out}} \qquad \frac{}{\texttt{Cap} : \texttt{ZX 0 2}} \qquad \frac{}{\texttt{Cup} : \texttt{ZX 2 0}} \qquad \frac{\text{in out} : \mathbb{N} \qquad \alpha : \mathbb{R}}{\texttt{X}\ \text{in out}\ \alpha : \texttt{ZX}\ \text{in out}}$$

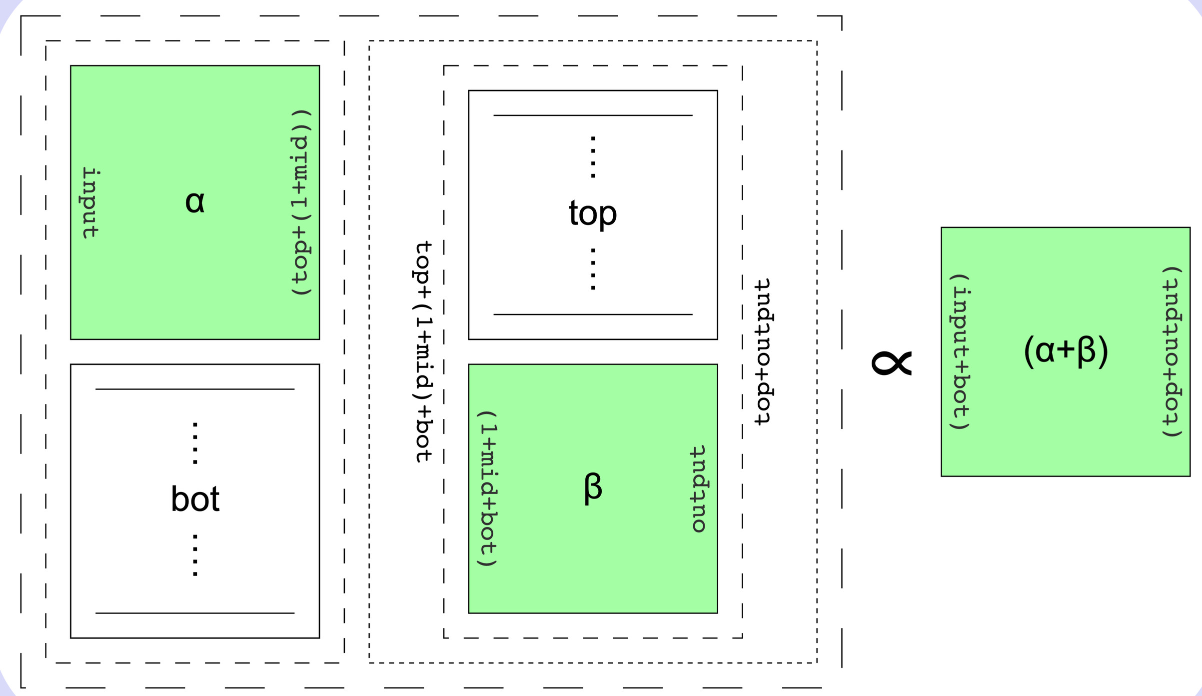
$$\frac{}{\texttt{Wire} : \texttt{ZX 1 1}} \qquad \frac{}{\texttt{Box} : \texttt{ZX 1 1}} \qquad \frac{}{\texttt{Swap} : \texttt{ZX 2 2}} \qquad \frac{}{\texttt{Empty} : \texttt{ZX 0 0}}$$

$$\frac{zx_0 : \texttt{ZX}\ \text{in mid} \qquad zx_1 : \texttt{ZX}\ \text{mid out}}{\texttt{Compose}\ zx_0\ zx_1 : \texttt{ZX}\ \text{in out}} \qquad \frac{zx_0 : \texttt{ZX}\ \text{in}_0\ \text{out}_0 \qquad zx_1 : \texttt{ZX}\ \text{in}_1\ \text{out}_1}{\texttt{Stack}\ zx_0\ zx_1 : \texttt{ZX}\ (\text{in}_0 + \text{in}_1)\ (\text{out}_0 + \text{out}_1)}$$

```
cast (n m : ℕ) {n' m' : ℕ} (prfn : n = n') (prfm : m = m') (zx : ZX n' m') : ZX n m.
```

Spider Fusion Example






```
Z input (top + S mid) α ↕ n_wire bot
⟷ $ top + S mid + bot, top + output
::: n_wire top ↕ Z (S mid + bot) output β $
∝ Z (input + bot) (top + output) (α + β)
```

In the example above, we see the standard ZX diagram representation of the ZX-calculus rewrite rule *spider fusion*, followed by the visualization of the same rule under the inductive definition and semantics. Spider fusion is a simple rule: if two or more spiders of the same color are connected via one or more wires, we can *fuse* them into a single spider, which has a rotation equal to the sum of the original rotations. To verify this rule, we must account for the exact number of inputs and outputs to each spider: something that the original visualization does not actually make explicit. In this case, we are left with this slightly more complex visualization of the term, carrying more information: but the structure roughly matches the standard visualization, making it clear to the proof engineer what this is an implication of.

Syntactically:

The concrete textual syntax for a Z spider is `Z in out rotation`, where `in`, `out` ∈ ℕ, `rotation` ∈ ℝ. Visually, it is represented as a green box with `in`, `out` labeling the edges, and `rotation` in the center. Vertical composition is represented textually by `term ↕ term`, while horizontal composition is `term ↔ term`. Visually, vertical composition is the placement of two terms in the same column, while horizontal composition is two terms in the same row. Equivalence of terms is represented both visually and textually by `term ∝ term`, where equivalent terms evaluate to the same semantics (up to a constant factor). `n_wire` is a function from a number `n` to a ZX diagram, that constructs a ZX diagram consisting of `n` wires composed vertically. Visually, `n_wire` is represented by the input `n` in a quadrilateral, with ellipses above and below it. We also have **casts**, that allow us to explicitly change the dimensions of a ZX-diagram `diagram` to `m'`,`n'` via the syntax `$ m', n' ::: diagram $`. Visually, this is represented by m' and n' annotations on the left and right sides of `diagram`, encapsulated in a dashed box.

References

John van de Wetering. 2020. *ZX-calculus for the working computer scientist*. arXiv.

inQWIRE. *VyZX: Verification of the ZX-calculus*. GitHub.

Adrian Lehmann et al. 2021. *VyZX: A Vision for verifying the ZX-calculus*. GitHub.

inQWIRE. *ViZX: Visalization of the ZX-calculus*. GitHub.

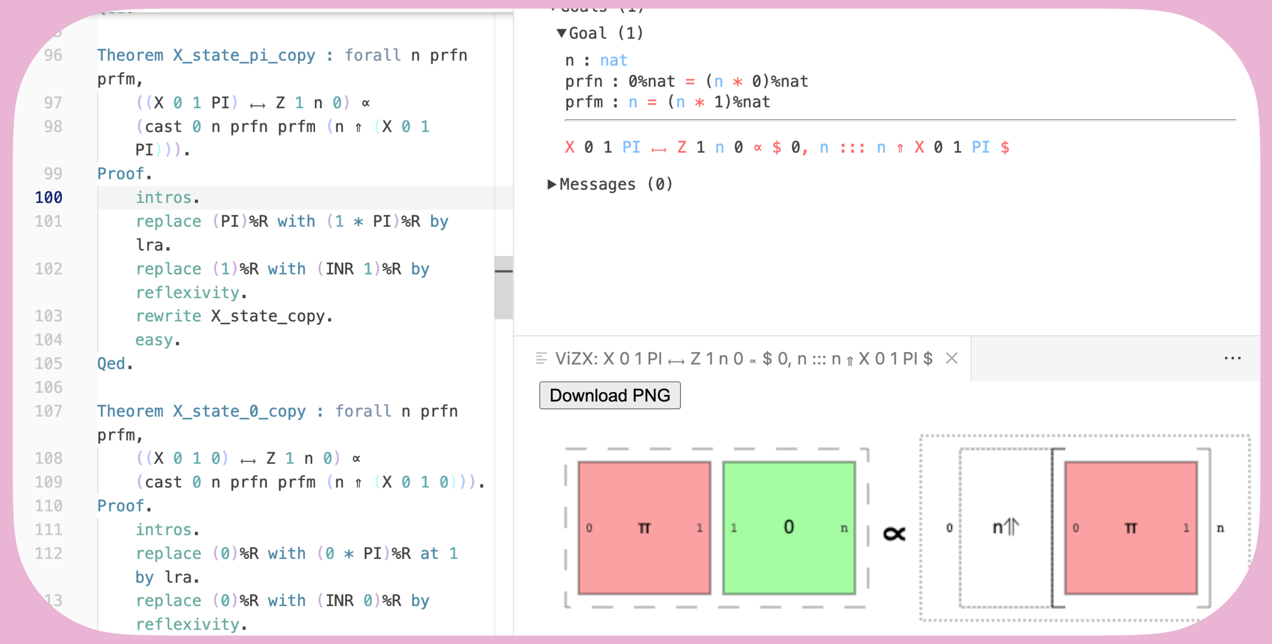
What's the problem?

Because the inductive representation carries a lot of structural information, textual representations of diagrams can be deeply nested, and hard to parse. This makes it difficult to identify sub-structures that can potentially be rewritten.

What's the solution?

The canonical representation of the ZX-calculus is primarily graphical. Thus, it seems natural that a visualization would make terms clearer. Though our inductive structure conveys the same semantics as the graphical structure, we want to focus on the diagram's *structure* rather than its *connectivity information*. Using the canonical visual syntax thus would not be helpful: we must design a visualization that emphasizes structural information over connectivity information.

Editor State



Proof Engineering Experience

To make this tool optimally efficient, integrating it actively into the proof engineering workflow was integral. A visualizer for the inductive ZX-diagram definition alone would be useful, but manual input would diminish the ease of use, and hence we wanted to look into ways to interleave it into the Coq ecosystem. We integrated the visualizer with the coq-lsp VSCode client, such that a visualization of any active term in the goal would be generated. On a change in the goal state, the visualization would automatically be updated.

Visualizer & IDE Integrated