*Vy*ZX: Formal Verification of a Graphical Quantum Language with automated structural rewrites

ADRIAN LEHMANN*, University of Chicago, USA
BEN CALDWELL*, University of Chicago, USA
BHAKTI SHAH, University of Chicago, USA
ROBERT RAND, University of Chicago, USA

ZX-diagrams are typically represented as adjacency-based graphs, reflecting the guiding principle that "only connectivity matters". In the context of formal theorem provers like Coq, however, such graphs are difficult to reason about, especially when we seek to give them semantics. To address this gap, we build *Vy*ZX, a verified library for reasoning about the ZX-calculus, using inductive constructs that arise naturally from category theoretic definitions. One of the issues arising from this representation is that we explicitly encode associativity information that can be an obstacle to reasoning about connectivity. To address this, we present DC⇕AC, a solver to automatically reason about associativity structures built on top of the e-graph solver *egg*. *egg* performs equivalence saturation with all possible structural rewrites in an efficient manner. We incorporate *egg* into *Vy*ZX to allow easy rewriting with the rules of the ZX-calculus.

1 INTRODUCTION

This work introduces *Vy*ZX, a formally verified implementation of the ZX-calculus in the Coq proof assistant. *Vy*ZX aims to bridge the gap between graphs, the natural representation of ZX-diagrams, and the inductive structures necessary to reason about the ZX-calculus in Coq. To do so, it draws upon the category theory that underlies ZX-calculus and represents graphs using $m \times n$ spiders connected by stack and compose constructors, where Coq's dependent types guarantee that the connections are valid. In order to aid reasoning about ZX-diagrams in practice, we also provide a visualization tool, ZXViz, which allows us to see associativity information in the Coq proof environment.

Reasoning about associativity is a key challenge for formal diagrammatic reasoning. In order to make *Vy*ZX usable in practice, we draw upon the study of E-Graphs [Nelson and Oppen 1980; Nieuwenhuis and Oliveras 2005], data structures that help us store and reason about equivalence relations. We can use equality saturation based on E-Graphs to efficiently find rewrite paths by considering equivalences in states reached using the tools *egg* and *egglog* [Willsey et al. 2021; Zhang et al. 2023].

We give ZX-diagrams their traditional semantics in terms of matrices, using the existing formalization of matrices from QuantumLib [INQWIRE Developers 2022]. This provides interoperability with a range of verified quantum computing projects, including the SQIR quantum IR [Hietala et al. 2021a] and the voqc verified compiler [Hietala et al. 2021b]. It allows us to read in quantum circuits and convert them into ZX-diagrams while guaranteeing that their semantics is preserved. We use the ZX semantics in order to prove the correctness of a complete equational theory over ZX-diagrams, specifically the theory presented by Jeandel et al. [2020]. This allows us to then prove further theorems purely using the ZX-calculus rewrite rules, without directly referring to the matrix semantics

Verifying the ZX-calculus allows us to build a verified software library based on ZX-diagrams. The desire to have verified software has inspired the CompCert verified C compiler [Leroy 2009]

* Equal contribution

Authors' addresses: Adrian Lehmann*, University of Chicago, USA; Ben Caldwell*, University of Chicago, USA; Bhakti Shah, University of Chicago, USA; Robert Rand, University of Chicago, USA.

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

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

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

Fig. 1. The inductive constructors for block representation ZX-diagrams

and also led to verified quantum optimizers like VOQC [Hietala et al. 2021b] and Giallar [Tao et al. 2022], the latter of which verified components of the popular (and buggy) QISKIT compiler [Tao et al. 2022]. In addition to allowing us to write a verified PyZX-style optimizer [Kissinger and van de Wetering 2020], *Vy*ZX can serve as a platform for connecting the various Z* calculi and their applications [Backens and Kissinger 2019; Hadzihasanovic 2017; Shaikh et al. 2023],without losing confidence in the tools' correctness.

2 *VYZX*

Reasoning about a graphical process theory like ZX inside a proof assistant is difficult. As in most graphs, our main concern is connectivity, but these processes have semantics, meaning that we need a way to construct diagrams that enforces a consistent order between the graph's nodes. To find a suitable inductive definition, we turn to the category that underpins the ZX-calculus and use categorical definitions to inspire our inductive constructors. Specifically, ZX-diagrams are a symmetric monoidal category [Joyal and Street 1993; Selinger 2010], which corresponds to string diagrams, meaning that they can be described by a small set of inductive constructors, along with the *Z* and *X* spiders that are the distinguishing morphisms of the ZX-calculus. This gives rise to the inductive definition of ZX-diagrams in Figure 1.

To assign meaning to our syntactic constructs, we construct a semantic evaluation function $[\![\,.\,]\!] : \texttt{ZX n m} \longrightarrow \mathbb{C}^{2^m \times 2^n}$; a ZX-diagram with n inputs and m outputs semantically evaluates to a matrix of size 2^m by 2^n (see Figure 2). Our semantic functions are built using QuantumLib's [INQWIRE Developers 2022] matrices and complex numbers. We define the equivalence relation *proportionality*:

$$\forall(\texttt{zx}_0, \texttt{zx}_1 : \texttt{ZX n m}), \texttt{zx}_0 \propto \texttt{zx}_1 \coloneqq \exists c \in \mathbb{C}, [\![\texttt{zx}_0]\!] = c \cdot [\![\texttt{zx}_1]\!] \land c \neq 0$$

We show within Coq that this is an equivalence relation and that `Stack` and `Compose` are *parametric morphisms* [Sozeau 2023], meaning that we can safely rewrite using proportionality within ZX-diagrams. Using the definition of proportionality, we can then prove facts about the ZX-calculus.

In contrast to conventional proofs in ZX-calculus, where one reasons about ZX-calculus through the lens of adjacency, we must also reason about stacking and composition. A common challenge in dependently typed programming shows up, as we require precise equality of dimensions across proportionality and the composition constructor. In practice we often however find ourselves with non-trivial semantic equality of dependent type arguments. To bridge this gap, we define a function called `cast` with the following type:

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

3 DISTRIBUTIVITY, CAST, AND ASSOCIATIVITY IN COQ (DC⇕AC)

When reasoning automatically about equalities, we often want to avoid repeatedly computing proofs to/from the same state while obtaining a complete list of equivalent expressions. This is where E-Graphs [Nelson and Oppen 1980; Nieuwenhuis and Oliveras 2005] come in. E-Graphs

$$\llbracket \texttt{Empty} \rrbracket = I_{1\times 1} \qquad \llbracket \texttt{Wire} \rrbracket = I_{2\times 2} \qquad \llbracket \texttt{Box} \rrbracket = H \qquad \llbracket \texttt{Cup} \rrbracket = [1,0,0,1]$$
$$\llbracket \texttt{Swap} \rrbracket = |00\rangle\langle 00| + |11\rangle\langle 11| + |01\rangle\langle 10| + |10\rangle\langle 01| \qquad \llbracket \texttt{Cap} \rrbracket = [1,0,0,1]^T$$
$$\llbracket \texttt{Z n m } \alpha \rrbracket = |0\rangle^{\otimes m}\langle 0|^{\otimes n} + e^{i\alpha}|1\rangle^{\otimes m}\langle 1|^{\otimes n} \qquad \llbracket \texttt{X n m } \alpha \rrbracket = H^{\otimes m} \times \llbracket \texttt{Z n m } \alpha \rrbracket \times H^{\otimes n}$$
$$\llbracket \texttt{zx}_0 \leftrightarrow \texttt{zx}_1 \rrbracket = \llbracket \texttt{zx}_1 \rrbracket \times \llbracket \texttt{zx}_0 \rrbracket \qquad \llbracket \texttt{zx}_0 \updownarrow \texttt{zx}_1 \rrbracket = \llbracket \texttt{zx}_0 \rrbracket \otimes \llbracket \texttt{zx}_1 \rrbracket$$

Fig. 2. *Vy*ZX semantics

represent each equivalent state of an expression as an equivalence class, where each equivalence class contains a set of "equivalence nodes". For a deeper introduction intro E-Graphs as relevant to equality saturation, please refer to Willsey et al. [2021].

In 2021, Willsey et al. developed *egg*, a tool to perform equality saturation [Tate et al. 2009] on E-Graphs. Instead of having to either guess or try all rewrites concurrently, equality saturation uses the E-Graph structure to bound the search space. Then rewrites are repeated on new equality class until a fixpoint or timeout is reached. On top of the *egg* E-Graph saturation tool, *egglog* [Zhang et al. 2023] is a prolog/datalog-style DSL that abstracts the underlying *egg* representation. With this abstraction *egglog* can also perform optimization on the E-Graph terms. In *Vy*ZX, we choose to use *egglog* over *egg*, due to the representation allowing to express conditional rewrite rule in a more conducive fashion. In its current state, *egglog* does not support proof extraction; though, this is a planned feature. Hence we build proof trees within *egglog* to be able to create formal proofs in Coq.

Within *egglog* we encode *Vy*ZX diagrams along with dependent type information. Using these data structures we are able to represent *Vy*ZX rules and use *egglog* to automatically find the equivalence of two diagrams. We encode rules related to associativity, casts, and distributivity to remove proof overhead arising from these synatic constructs.

4 FUTURE WORK

In the future, we hope to abstract the work done in *Vy*ZX to talk about a broader range of categories and their corresponding diagrammatic representations. Examples of these include non-symmetric braided monoidal categories for knot theory, cartesian closed categories for simple type theory, and symmetric bimonoidal categories for linear algebra. By using typeclasses, we can generalize these automated structural rewrites to work for arbitrary instances of the appropriate typeclass. We hope to not only improve our visualizations once we can ignore associativity information, but also include a practical graph representation, with semantics-preserving translations between adjacency lists and our inductive structure. This will allow these diagrams to be viewed and manipulated both as syntax trees and as graphs themselves. We also hope to expand DC⇕AC to be able to solve more complicated *Vy*ZX equalities. For that we want to be able to dynamically supply lemmas and have DC⇕AC automatically encode them into *egglog*.

REFERENCES

Miriam Backens and Aleks Kissinger. 2019. ZH: A Complete Graphical Calculus for Quantum Computations Involving Classical Non-linearity. *Electronic Proceedings in Theoretical Computer Science* 287 (Jan 2019), 23–42. https://doi.org/10.4204/eptcs.287.2

Amar Hadzihasanovic. 2017. The algebra of entanglement and the geometry of composition. arXiv:1709.08086 [math.CT]

Kesha Hietala, Robert Rand, Shih-Han Hung, Liyi Li, and Michael Hicks. 2021a. Proving Quantum Programs Correct. In *12th International Conference on Interactive Theorem Proving (ITP 2021) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 193)*, Liron Cohen and Cezary Kaliszyk (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 21:1–21:19. https://doi.org/10.4230/LIPIcs.ITP.2021.21

Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. 2021b. A Verified Optimizer for Quantum Circuits. *Proc. ACM Program. Lang.* 5, POPL, Article 37 (jan 2021), 29 pages. https://doi.org/10.1145/3434318

INQWIRE Developers. 2022. *INQWIRE QuantumLib.* https://github.com/inQWIRE/QuantumLib

Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. 2020. Completeness of the ZX-Calculus. *Logical Methods in Computer Science* (6 2020). https://doi.org/10.23638/LMCS-16(2:11)2020

André Joyal and Ross Street. 1993. Braided tensor categories. *Advances in Mathematics* 102, 1 (1993), 20–78.

Aleks Kissinger and John van de Wetering. 2020. PyZX: Large Scale Automated Diagrammatic Reasoning. In Proceedings 16th International Conference on *Quantum Physics and Logic,* Chapman University, Orange, CA, USA., 10-14 June 2019 *(Electronic Proceedings in Theoretical Computer Science, Vol. 318)*, Bob Coecke and Matthew Leifer (Eds.). Open Publishing Association, 229–241. https://doi.org/10.4204/EPTCS.318.14

Xavier Leroy. 2009. Formal Verification of a Realistic Compiler. *Commun. ACM* 52, 7 (jul 2009), 107–115. https://doi.org/10.1145/1538788.1538814

Greg Nelson and Derek C. Oppen. 1980. Fast Decision Procedures Based on Congruence Closure. *J. ACM* 27, 2 (apr 1980), 356–364. https://doi.org/10.1145/322186.322198

Robert Nieuwenhuis and Albert Oliveras. 2005. Proof-Producing Congruence Closure. In *Term Rewriting and Applications*, Jürgen Giesl (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 453–468.

P. Selinger. 2010. A Survey of Graphical Languages for Monoidal Categories. In *New Structures for Physics*. Springer Berlin Heidelberg, 289–355. https://doi.org/10.1007/978-3-642-12821-9_4

Razin A. Shaikh, Quanlong Wang, and Richie Yeung. 2023. How to Sum and Exponentiate Hamiltonians in ZXW Calculus. In *Proceedings 19th International Conference on Quantum Physics and Logic, Wolfson College, Oxford, UK, 27 June - 1 July 2022 (Electronic Proceedings in Theoretical Computer Science, Vol. 394)*, Stefano Gogioso and Matty Hoban (Eds.). Open Publishing Association, 236–261. https://doi.org/10.4204/EPTCS.394.14

Matthieu Sozeau. 2023. Generalized rewriting. https://github.com/coq/coq/blob/58ac2d8dd403fa7a96429fc1f839225d83be9566/doc/sphinx/addendum/generalized-rewriting.rst

Runzhou Tao, Yunong Shi, Jianan Yao, Xupeng Li, Ali Javadi-Abhari, Andrew W. Cross, Frederic T. Chong, and Ronghui Gu. 2022. Giallar: Push-Button Verification for the Qiskit Quantum Compiler. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI '22)*. Association for Computing Machinery, New York, NY, USA, 641–656. https://doi.org/10.1145/3519939.3523431 arXiv:2205.00661

Ross Tate, Michael Stepp, Zachary Tatlock, and Sorin Lerner. 2009. Equality saturation: a new approach to optimization. *SIGPLAN Not.* 44, 1 (jan 2009), 264–276. https://doi.org/10.1145/1594834.1480915

Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. 2021. Egg: Fast and extensible equality saturation. *Proceedings of the ACM on Programming Languages* 5, POPL (2021), 1–29.

Yihong Zhang, Yisu Remy Wang, Oliver Flatt, David Cao, Philip Zucker, Eli Rosenthal, Zachary Tatlock, and Max Willsey. 2023. Better Together: Unifying Datalog and Equality Saturation. arXiv:2304.04332 [cs.PL]