Graph Rewriting Language as a Platform for Quantum Diagrammatic Calculi

PADL2026 @ Rennes   January 12, 2026

Kayo Tei, Haruto Mishina, Naoki Yamamoto, Kazunori Ueda

Waseda University, Tokyo, Japan

(Extended version available at arXiv.org)

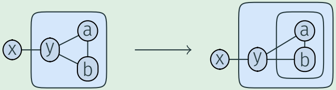
Overview

Overview

Graph Rewriting

✓ **Powerful formalism for representing & transforming structured data.**

- Applied in various domains: chemistry, math, **quantum physics**, etc.



ZX-calculus [1]

- **Diagrammatic** and **rule-based** framework for **quantum circuts**.
- **Specialized graphical tools** for optimization and proof assistance.

★ **Designing & verifying rewriting strategies is challenging.**

+

LMNtal [2]

- **Concrete declarative language** for hierarchical graph rewriting.

✓ **State space exploration** tool

✓ **Model checking** tool

✓ **Quantifiers** for expressive pattern matching **(QLMNtal)**

1 Kazunori Ueda. **"LMNtal as a hierarchical logic programming language".** In: *Theoretical Computer Science* 410.46 (2009)

2 Bob Coecke et al. ***Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning.*** 2017

Research Question and Contributions

(RQ) **How can we bridge declarative programming and quantum computing?**

Challenges in the ZX-calculus

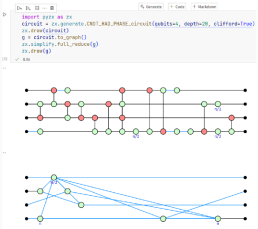
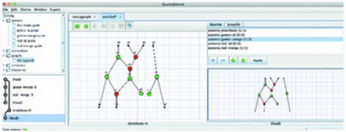
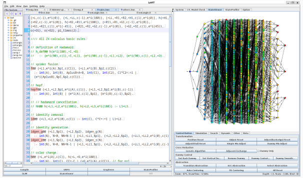
1. **Systematic exploration** of optimization strategies
2. **Analyzing properties** of strategies (confluence, termination, etc.)
3. **Gap** between **diagrammatic rules** and **implementations**

... for which QLMNtal provides a language and a comlementary platform

- **Each graphical ZX-rule can be directly encoded as a single QLMNtal rule**.
- New strategies can be **modeled and verified in our platform** to confirm their effectiveness, and **then can be implemented in existing tools**.

Existing tools vs. LMNtal

- Major **optimization/verification tools for ZX-calculus** & our **LMNtal ecosystem**:

	PyZX [3]	**Quantomatic** [4]	**LMNtal / QLMNtal**
Goal	Optimization	Verification	**General graph rewriting**
Approach	automatic simplification via **built-in heuristics**	step-by-step proofs via **user-defined rules**	**non-deterministic execution** of **user-defined rulesets**
Syntax	**graphical**	**graphical**	**textual (PL-standard)**
UI			

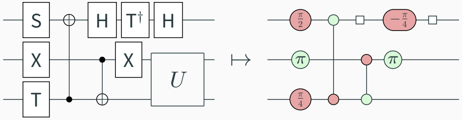
[3] Aleks Kissinger et al. **"PyZX: Large Scale Automated Diagrammatic Reasoning".** In: *16th International Conference on Quantum Physics and Logic (QPL 2019).* Vol. 318. EPTCS. 2020

[4] Aleks Kissinger et al. **"Quantomatic: A Proof Assistant for Diagrammatic Reasoning".** In: *Automated Deduction - CADE-25.* Vol. 9195. LNCS. 2015

ZX-calculus, a graphical formalism for quantum circuits

From Quantum Circuits to ZX-calculus

- Operations on quantum circuits are complex due to diverse gates and fixed wiring ...
  ⟶ **ZX-calculus** [5] born from **String Diagrams** [6]
  - All kinds of circuits can be represented by **only two types of nodes & wires!**
  - Can be bent and stretched freely.
  - Only a small number of universal rewrite rules are needed.

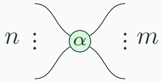
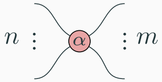
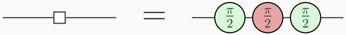


- While **String Diagrams** are more oriented to **category theory**, **ZX-calculus** has more flexibility and **affinity with graph rewriting**

5 Bob Coecke et al. **"Interacting quantum observables: categorical algebra and diagrammatics".** In: *New J. Phys.* 13.4 (2011)
6 Bob Coecke et al. ***Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning.*** 2017

ZX-diagrams

- ZX-diagrams consist of Z- and X-spiders connected by wires.
  - $n, m \in \mathbb{N}$
  - $\alpha \in \mathbb{R}$: phase

Z-spider	X-spider	Hadamard gate
		

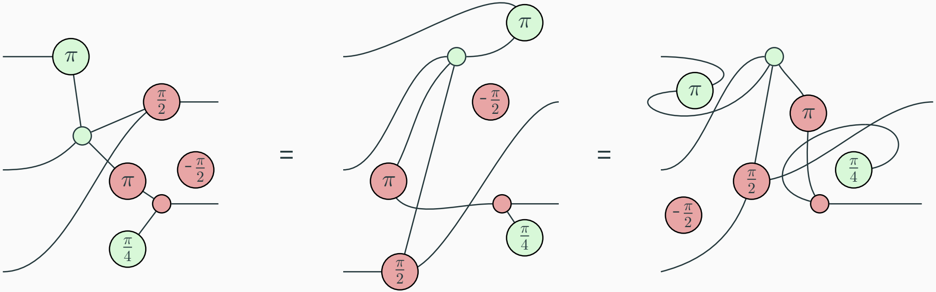
Equivalence of ZX-diagrams

- **"Only connectivity matters"**

  As long as the connectivity is preserved,

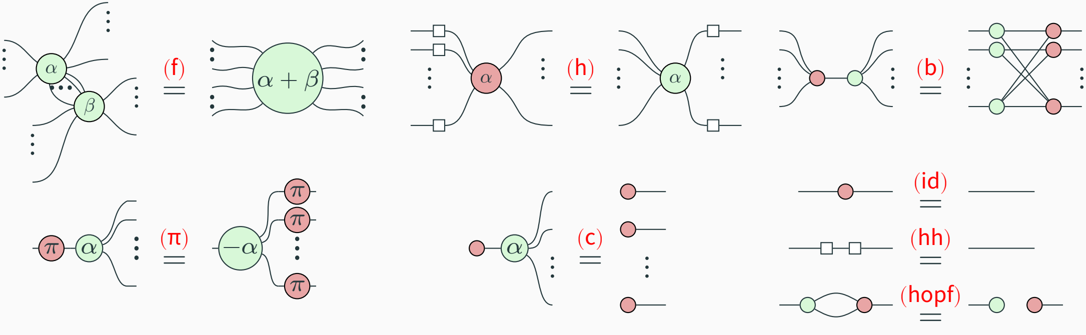
  ★ diagrams are equivalent under deformation, and

  ★ diagrams don't care about input/output order.



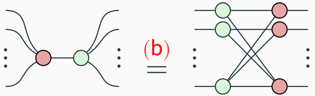
Rewrite Rules in ZX-calculus

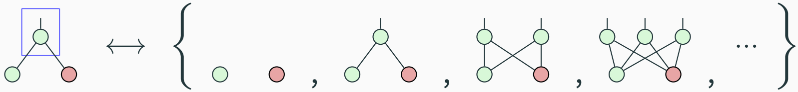
- Rewrite rules preserve the equivalence of ZX-diagrams.
- Also, rules enjoy **color symmetry**: can swap Z- (green) and X- (red) spiders



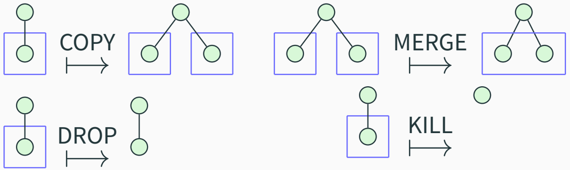
!-boxes (bang-boxes)

- In ZX-calculus, "**...** " means "any number".
  - Informal and can be ambiguous. see (b) (bialgebra rule):
  - We can use **!-boxes** [7] instead!





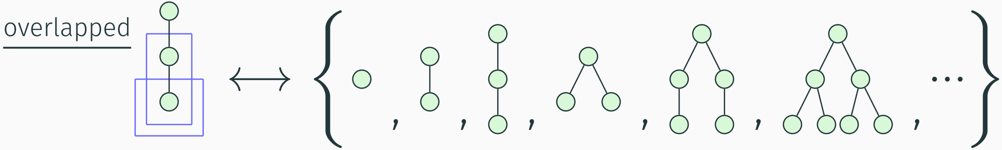
- Subgraphs surrounded by a !-box can be **instantiated** any number of times.
- Four instantiation operations: **COPY**, **KILL**, **DROP**, and **MERGE**.



7 Aleks Kissinger et al. **"Pattern graph rewrite systems".** In: *Developments in Computational Models 2012 (DCM 2012)*. Vol. 143. EPTCS. 2014
A. Kissinger et al. **"Tensors, !-graphs, and Non-commutative Quantum Structures".** In: *New Gener. Comput.* 34 (2016)
Lucas Dixon et al. **"Graphical Reasoning in Compact Closed Categories for Quantum Computation".** In: *Ann. Math. Artif. Intell.* 56 (2009)

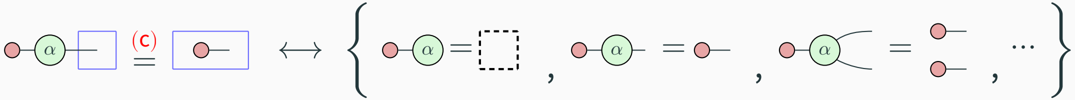
Overlapped !-boxes

- !-boxes can be **overlapped**.

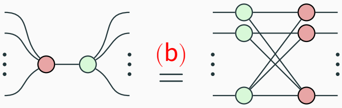


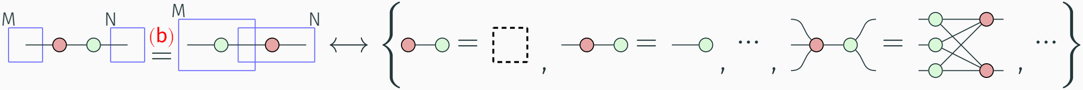
Rules written with !-boxes

- !-graphs (graphs with !-boxes) can be placed on both sides to represent rewrite rules.

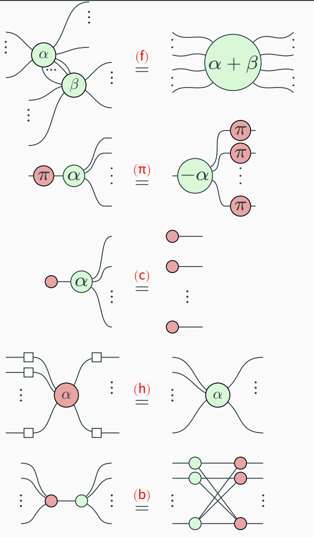
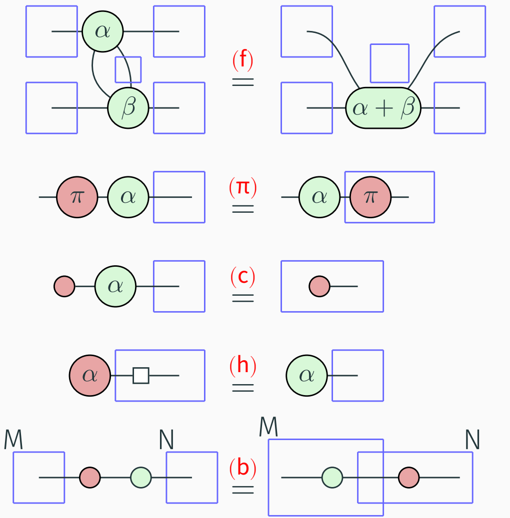


- (b)-rule with !-boxes:





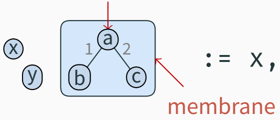
Representing ZX-rules with !-boxes

ZX-rules with "… "	ZX-rules with !-boxes
	

LMNtal and QLMNtal

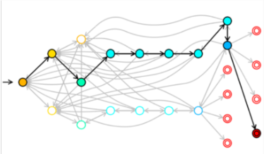
LMNtal `https://bit.ly/lmntal-portal`

- **LMNtal**: a hierarchical graph rewriting language born from **concurrent constraint programming** + **Constraint Handling Rules (CHR)** [8].
  - graph node = **atom** = formula
  - graph edge = **link** = logical variable
  - hierarchy = **membrane** = context



`:= x, y, {a(b, c)}`

- Toolkit provides various features including **visualizer**, **state space explorer**, **model checker**.
- **QLMNtal** [9] extends LMNtal with **quantifiers**: **cardinality**, **negation**, and **universal**.



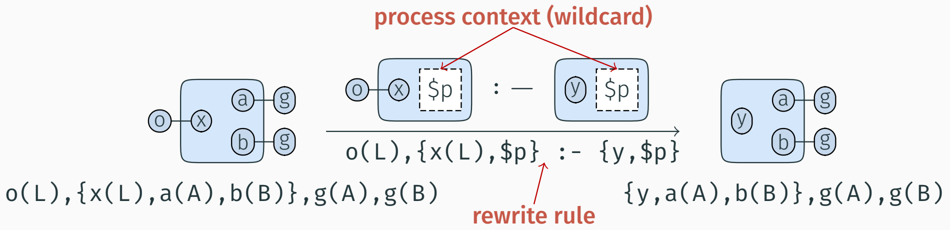
State Space visualized by **LMNtal StateViewer** (state space explorer)

[8] Thom Frühwirth. **"Theory and practice of Constraint Handling Rules".** In: *J. Logic Programming* 37.1 (1998)

[9] Haruto Mishina et al. **"Introducing Quantification into a Hierarchical Graph Rewriting Language".** In: *34th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2024)*. Vol. 14919. LNCS. Extended version available at https://arxiv.org/abs/2411.14802. 2024

Syntax of LMNtal

Process	$P ::= \mathbf{0} \mid p(X_1, \ldots, X_n) \mid P, P \mid m\{P\} \mid T :\text{-}\ T$
Process template	$T ::= \mathbf{0} \mid p(X_1, \ldots, X_n) \mid T, T \mid m\{T\} \mid T :\text{-}\ T \mid \$p$



- LMNtal graphs are **port graphs**: a(X,Y) $\neq$ a(Y,X).
- Rules are applied **repeatedly** and **non-deterministically**.
- Membranes can contain other membranes.

Introducing Quantifiers[10]

Process	$P ::= \mathbf{0} \mid p(X_1, \ldots, X_n) \mid P, P \mid m\{P\} \mid T \,\texttt{:-}\, T$
Process template	$T ::= \mathbf{0} \mid p(X_1, \ldots, X_n) \mid QT \mid T, T \mid m\{T\} \mid T \,\texttt{:-}\, T \mid \$p$
Quantifier	$Q ::= l\langle z, z\rangle \mid l\langle \hat{}\, \rangle$

- QT: Newly introduced **quantified process template**
- Q : **Quantifier**. Can represent either **cardinality** $(l\langle z, z\rangle)$ or **negation** $(l\langle \hat{}\, \rangle)$.
- **label** l is an identifier for the quantifier.

10 Haruto Mishina et al. **"Introducing Quantification into a Hierarchical Graph Rewriting Language".** In: *34th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2024)*. Vol. 14919. LNCS. Extended version available at https://arxiv.org/abs/2411.14802. 2024

Universal Quantifier in QLMNtal

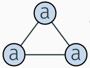
- **Universal quantifier** $l\langle * \rangle$ = **cardinality quantifier** + **negation quantifier**.
  - $l\langle 0, \infty \rangle T$: There are **arbitrarily many** instances of pattern T
  - $l\langle \hat{} \rangle T'$: **No instance** of pattern T' (a variant of T)
  - and they are **associated** by label l.

$$l\langle * \rangle T := l\langle 0, \infty \rangle T, l\langle \hat{} \rangle T'$$

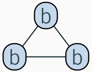
- Negation quantifier ensures that all instances of T is matched by the cardinality quantifier.

Universal Quantifiers in QLMNtal

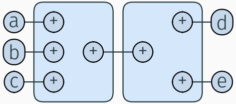
- Matching **all** a's with 2 links



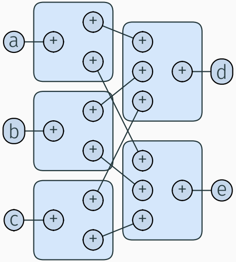

```
<*>a(X,Y) :- <*>b(X,Y)
```



- An example motivated from (b) (bialgebra) rule




```
{<*>+L1,+L2}, {+L2,N<*>+L3} :-
M<*>{+L1,N<*>+L2,}, N<*>{M<*>+L2,+L3}
```

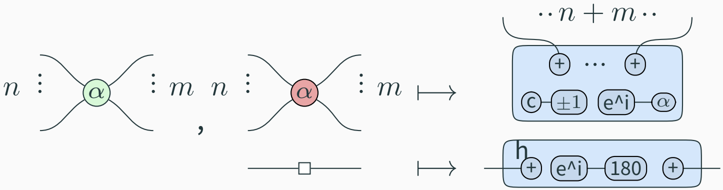


- **Quantifiers with the same label** are **associated** with each other.
- Quantifiers can be **nested**.

Encoding ZX-calculus in LMNtal

ZX-diagram in LMNtal

- **"Only connectivity matters"** corresponds nicely to **LMNtal graphs**



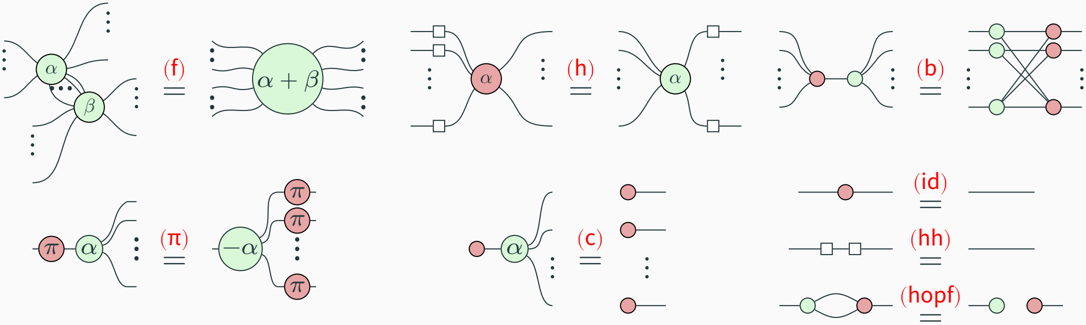
- **Can treat Z- and X-spiders uniformly**: `{e^i(alpha), c(color), $ports}`
- α (phase) is represented in degrees
- Have abbreviation for Hadamard gates: `h{e^i(180), $ports}`
  - that can be easily extended to ZH-calculus [1]

[1] Miriam Backens et al. **"ZH: A Complete Graphical Calculus for Quantum Computations Involving Classical Non-linearity".** In: *15th International Conference on Quantum Physics and Logic (QPL 2018)*. Vol. 287. EPTCS. 2019

Kayo Tei et al. ***Graph Rewriting Language as a Platform for Quantum Diagrammatic Calculi.*** 2025

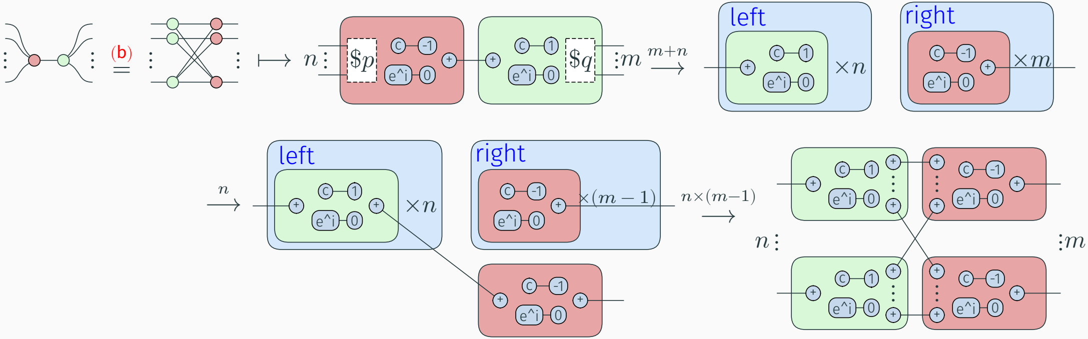
ZX-rules in LMNtal

- Some of the rules ((f), (id), (hh), (hopf) below) can be written in a single LMNtal rule
- Others need multiple LMNtal rules to handle the "..."



Implementation of the (b) rule in plain LMNtal

- The (b) (bialgebra) rule can be represented in plain LMNtal but involves multiple rewriting steps
- and only for left-to-right rewriting



Using QLMNtal for ZX-rules

- Using QLMNtal's univeral quantifiers, **all ZX-rules can be represented straightforwardly**
- Furthermore, the reverse direction is achieved **just by swapping head and body**! (with a bit of care for guard conditions)

Bidirectional (b) (bialgebra) rule in QLMNtal

Left-to-right:

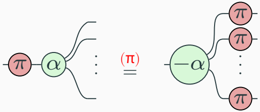
```
   {M<*>+L1,+L2,e^i(0),c(1)}, {+L2,N<*>+L3,e^i(0),c(-1)}
 :- M<*>{+L1,N<*>+L2,e^i(0),c(-1)}, N<*>{M<*>+L2,+L3,e^i(0),c(1)}.
```

Right-to-left:

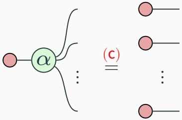
```
    M<*>{+L1,N<*>+L2,e^i(0),c(-1)}, N<*>{M<*>+L2,+L3,e^i(0),c(1)}
 :- {M<*>+L1,+L2,e^i(0),c(1)}, {+L2,N<*>+L3,e^i(0),c(-1)}.
```

ZX-rules in QLMNtal

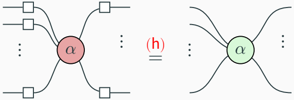
- Other ZX-rules with the "…" can also be written concisely



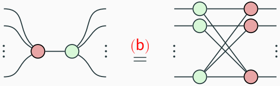

```
{+L1,+L2,e^i(180),c(C1)}, {+L2,<*>+L3,e^i(A),c(C2)}
:- int(A), AA=-A, int(C1), int(C2), C1*C2=:=-1 |
{+L1,<*>+L2,e^i(AA),c(C2)}, <*>{+L2,+L3,e^i(180),c(C1)}
```




```
{+L1,e^i(0),c(C1)}, {+L1,<*>+L2,e^i(A),c(C2)}
:- int(A), int(C1), int(C2), C1*C2=:=-1 | <*>{+L2,e^i(0),c(C1)}
```

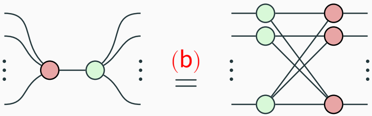


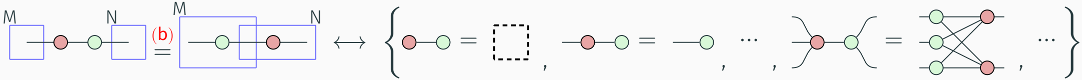

```
{<*>+L1,e^i(A),c(C)}, <*>h{+L1,+L2,e^i(180)}
:- int(A), int(C), CC=-C | {<*>+L2,e^i(A),c(CC)}
```




```
{M<*>+L1,+L2,e^i(0),c(1)}, {+L2,N<*>+L3,e^i(0),c(-1)}
:- M<*>{+L1,N<*>+L2,e^i(0),c(-1)}, N<*>{M<*>+L2,+L3,e^i(0),c(1)}
```

Expressive power of QLMNtal vs. !-boxes





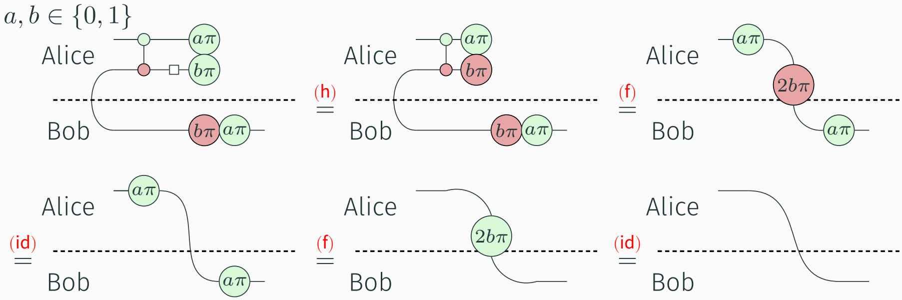

```
  {M<*>+L1,+L2,e^i(0),c(1)}, {+L2,N<*>+L3,e^i(0),c(-1)}
:- M<*>{+L1,N<*>+L2,e^i(0),c(-1)}, N<*>{M<*>+L2,+L3,e^i(0),c(1)}
```

- **!-graphs with overlapping □'s can be encoded into nested <*>'s of QLMNtal!**

Examples

Example 1: Simplification of quantum teleportation circuit [12] (1/2)

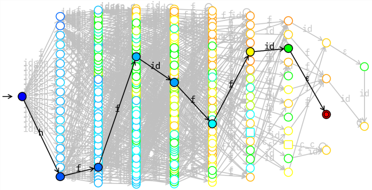
- When devising a circuit simplification method, **we can verify the procedure** using **StateViewer** and **LTL model checker**.
- An example simplication of the quantum teleportation circuit:



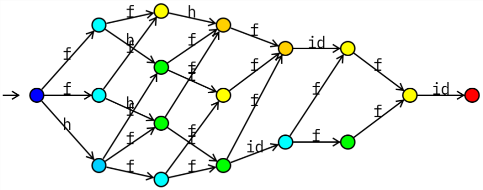
- **What if we know nothing but the final state?**
- **We can still find out the path!**

12 John van de Wetering. **"ZX-calculus for the working quantum computer scientist".** In: (2020)

Example 1: Simplification of quantum teleportation circuit (2/2)



State space with extra rules



State space without extra rules

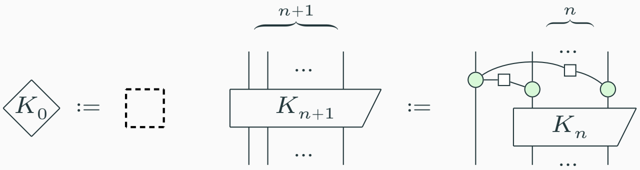
- We may trigger more rules than we need ...
  - which happens when we don't have enough knowledge of the steps.

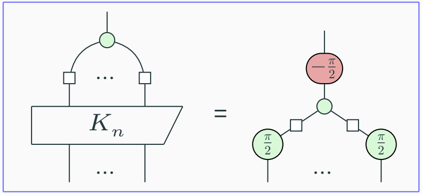
★ We can easily find out the **shortest path** using **model checker.**

★ We can easily analyze **which rules are unneeded.**

Example 2: Checking inductive proofs (1/4)

- Suppose we want to check a (manual) inductive proof we have devised.
- E.g., let's confirm that the equation below [13] holds for $n = 4$
  - ...assuming that the cases with $n \leq 3$ have already been proved

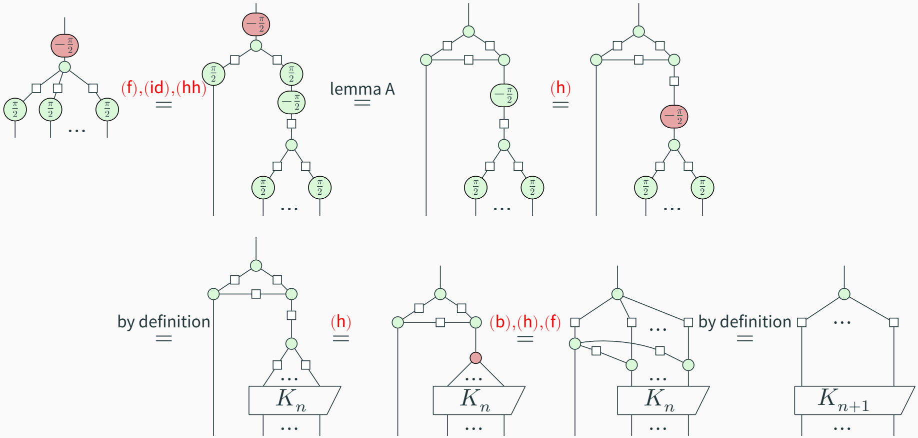




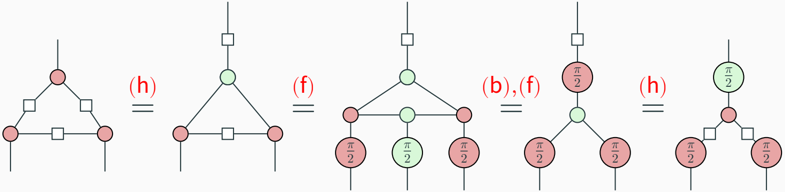
13 Bob Coecke et al. ***Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning.*** 2017

Example 2: Checking inductive proofs (2/4)

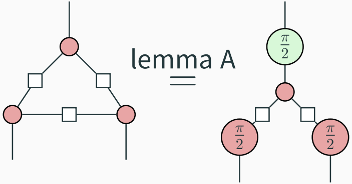
- **Assign tokens to each rule and control rule application** (as in the derivation below)
  - `use_rules=[f,id,hh,a,h,k,h,b,h,f,kn].`
    `use_rules=[f|L],` ***head*** `:- use_rules=L,` ***body***



Example 2: Checking inductive proofs (3/4)

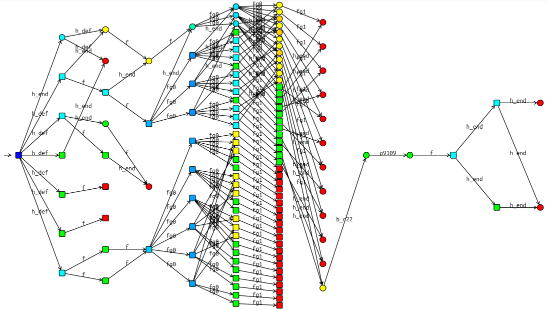


- We proved lemma A in our platform first
- ...and then introduced it as a single rule for the main proof.

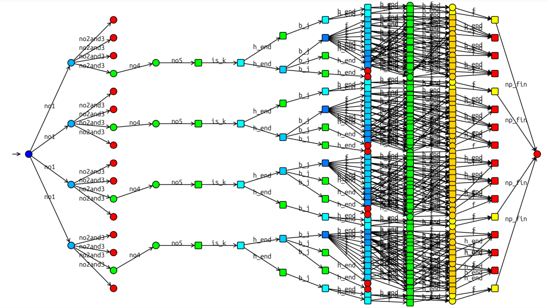


Example 2: Checking inductive proofs (4/4)

- These are the full state spaces for each proof.



state space for the lemma A proof



state space for the $n = 3 \to 4$ proof

- We can manage the rule application to **get some determinism** by using tokens.
- **Proof steps are now visually easy to understand.**

Example 3: Analyzing efficient simplification strategies (1/2)

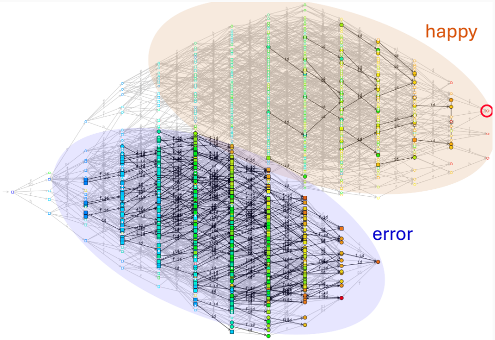
- **2-qubit QFT circuit simplification**[14]
- $(\pi)\times 2$, (f), (id), (c), (h) are used.
  - **8** different final states
    - **How to reach a better one?**
  - We can analyze which rule applications lead to better results using StateViewer.
    * "Better final state" = "fewer spiders" in this case.

Examples	# of states	# of final states
Pauli pushing	533	7
2-qubit QFT	**1186**	**8**
Detecting Entanglement	436,711	60

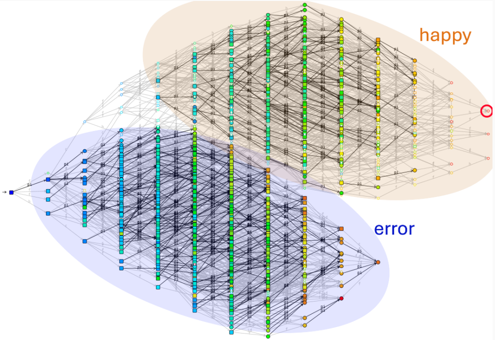
14 John van de Wetering. **"ZX-calculus for the working quantum computer scientist".** In: (2020)

Example 3: Analyzing efficient simplification strategies (2/2)

- ★ **Applying (id) early** leads to **undesired final states (dead ends) quickly.**
- ★ **Applying (π) in the middle stage** tends to lead to **better results.**



search (id) in the state space



search (π) in the state space

Temporarily increasing spiders by (π) led to more effective simplification!

Conclusion and Future Work

Conclusion

We presented a **flexible and customizable platform** based on **LMNtal with quantifiers** to design and verify rewrite strategies in ZX-calculus, a diagrammatic framework for quantum computing.

QLMNtal Approach

- **Quantified** pattern matching handling arbitrary numbers of links.
- **Model Checking** with state-space exploration.

→

Benefits for ZX-calculus

- A **concise & direct** encoding of complex rules (e.g., generalized bialgebra)
- A **Laboratory** for verifying optimization strategies and heuristics

- **Bridging declarative programming and quantum computing.**

Future Work

1. **Extending the platform:**
   - Extend Quantifier in QLMNtal to express nested !-boxes.
   - Extend the platform to other diagrammatic calculi related to ZX-calculus, such as ZW-calculus.
2. **Deeper analysis:**
   Systematically analyze existing optimization techniques using our approach to uncover new insights.
3. **Improving usability:**
   Support importing/exporting from/to widely used quantum circuit formats like OpenQASM.

Thank you for listening! Questions?

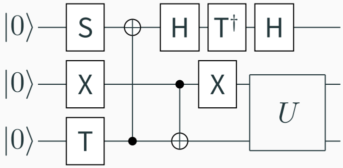
Appendix

Quantum Circuit

Quantum Gates and Circuits

- **Quantum bits (qubits)**: Unlike classical bits (0 or 1), qubits can be in **superposition**
  - State: $\alpha\,|0\rangle + \beta\,|1\rangle$ where $|\alpha|^2 + |\beta|^2 = 1$
- **Common quantum gates**:
  - **Hadamard (H)**: Creates superposition
  - **Pauli gates (X, Y, Z)**: Bit-flip and phase-flip
  - **Phase gates (S, T)**: Apply phase rotations ($\pi/2$, $\pi/4$)
  - **CNOT**: Two-qubit controlled operation

Example: 3-qubit circuit



Wires represent qubits
Gates applied left-to-right

! **Fixed wire topology**: Qubits cannot cross freely

! **Critical for quantum computers**: Shorter circuits ⇒ less decoherence & errors

! **Complex interactions**: Exponential growth of possible transformations

String Diagram

From Category Theory to Quantum Computing

- **String diagrams**: Graphical notation from category theory
  - Originated in monoidal categories
  - Objects → wires
  - Morphisms → nodes/boxes
- **Key properties**:
  - **Topological**: Only connectivity matters
  - **Compositional**: Diagrams compose naturally
  - **Equational reasoning**: Diagram equivalence = morphism equality

Evolution of notation

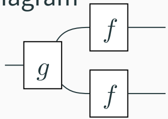
Category Theory

$f : A \to B$

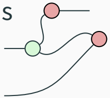
$g : B \to C$

$g \circ f : A \to C$

String Diagram



ZX-Calculus



- Why String Diagrams for Quantum Computing?
  - ! **vs. Quantum Circuits**: Not fixed topology
  - ! **ZX-calculus**: Specialized string diagrams with only 2 node types + simple rules
  - ! **Enables graphical proofs**: See E.g. 1

Overlapped and nested !-boxes

- !-box distinguishes between **overlapping and nesting.**
  - COPY performs differently: