# PyZX: Quantum circuit optimization using the ZX-calculus

Aleks Kissinger

aleks@cs.ru.nl

John van de Wetering

john@vdwetering.name

Institute for Computing and Information Sciences
Radboud University Nijmegen

December 17, 2018

# Introduction

- In fault-tolerant quantum computing, Clifford gates are cheap.

# Introduction

- In fault-tolerant quantum computing, Clifford gates are cheap.
- But to achieve universal QC we need other gates.

# Introduction

- In fault-tolerant quantum computing, Clifford gates are cheap.
- But to achieve universal QC we need other gates.
- Most commonly $T = R_Z(\pi/4)$.

# Introduction

- In fault-tolerant quantum computing, Clifford gates are cheap.
- But to achieve universal QC we need other gates.
- Most commonly $T = R_Z(\pi/4)$.
- T gates are far more expensive than Clifford gates.

# Introduction

- In fault-tolerant quantum computing, Clifford gates are cheap.
- But to achieve universal QC we need other gates.
- Most commonly $T = R_Z(\pi/4)$.
- T gates are far more expensive than Clifford gates.
- So:
  Optimizing fault tolerant QC means optimizing T-count.

# T-count optimization

Finding optimal T-count is NP-hard,
so we need heuristics.

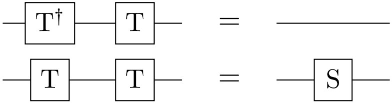
# T-count optimization

Finding optimal T-count is NP-hard,
so we need heuristics.

Existing heuristics fall basically in two categories.

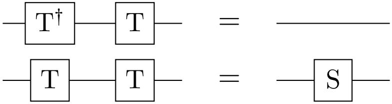
# Method 1: Commutation and Cancellation

Adjacent T gates become Clifford:



# Method 1: Commutation and Cancellation

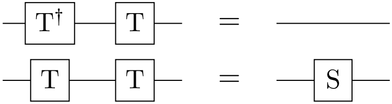
Adjacent T gates become Clifford:



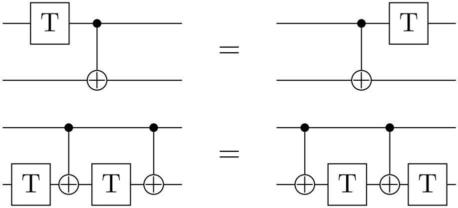

$\Rightarrow$ By making T gates adjacent, we can decrease T count.

# Method 1: Commutation and Cancellation

Adjacent T gates become Clifford:

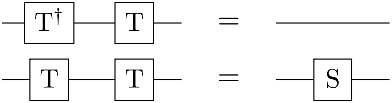



$\Rightarrow$ By making T gates adjacent, we can decrease T count.

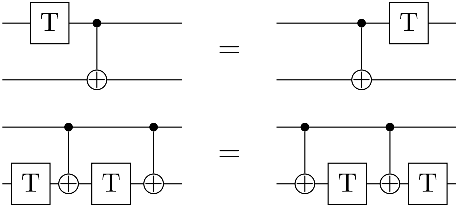


# Method 1: Commutation and Cancellation

Adjacent T gates become Clifford:




$\Rightarrow$ By making T gates adjacent, we can decrease T count.




And loads more...

# Method 2: Phase Polynomials

Circuits built out of CNOT and T gates can be written as

$$U\,|\mathbf{x}\rangle = e^{i\frac{\pi}{4}g(\mathbf{x})}\,|f_1(\mathbf{x}),\ldots,f_n(\mathbf{x})\rangle$$

where $\mathbf{x} \in \mathbb{Z}_2^n$ is a binary vector,
$g : \mathbb{Z}_2^n \to \mathbb{Z}_8$ is a polynomial and
$f_i : \mathbb{Z}_2^n \to \mathbb{Z}_2$ are $\mathbb{Z}_2$-linear functions.

# Method 2: Phase Polynomials

Circuits built out of CNOT and T gates can be written as

$$U|\mathbf{x}\rangle = e^{i\frac{\pi}{4}g(\mathbf{x})}|f_1(\mathbf{x}), \ldots, f_n(\mathbf{x})\rangle$$

where $\mathbf{x} \in \mathbb{Z}_2^n$ is a binary vector,
$g : \mathbb{Z}_2^n \to \mathbb{Z}_8$ is a polynomial and
$f_i : \mathbb{Z}_2^n \to \mathbb{Z}_2$ are $\mathbb{Z}_2$-linear functions.

$\Rightarrow$ Using this translation, we can make interesting simplifications

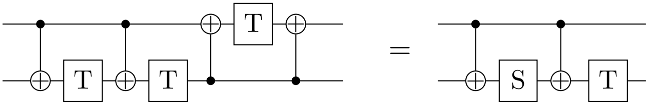
# Method 2: Phase Polynomials

Circuits built out of CNOT and T gates can be written as

$$U\,|\mathbf{x}\rangle = e^{i\frac{\pi}{4}g(\mathbf{x})}\,|f_1(\mathbf{x}),\ldots,f_n(\mathbf{x})\rangle$$

where $\mathbf{x} \in \mathbb{Z}_2^n$ is a binary vector,
$g : \mathbb{Z}_2^n \to \mathbb{Z}_8$ is a polynomial and
$f_i : \mathbb{Z}_2^n \to \mathbb{Z}_2$ are $\mathbb{Z}_2$-linear functions.

$\Rightarrow$ Using this translation, we can make interesting simplifications




(but optimal T-count finding still seems to be in NP)

# Limitations

Using commutation, cancellation and phase polynomials,
a lot of progress can be made...

# Limitations

Using commutation, cancellation and phase polynomials,
a lot of progress can be made...

...but there is an obvious limitation:

# Limitations

Using commutation, cancellation and phase polynomials,
a lot of progress can be made...

...but there is an obvious limitation:

## These methods never stray from the circuit model

# Limitations

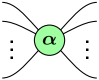
Using commutation, cancellation and phase polynomials,
a lot of progress can be made...

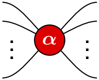
...but there is an obvious limitation:

These methods never stray from the circuit model

Enter the ZX-calculus

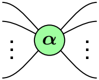
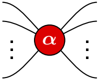
# ZX-diagrams

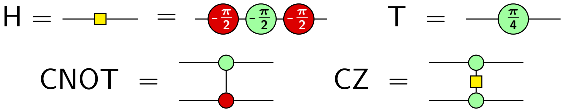
- ZX-diagrams consist of two types of maps
- Z-spiders 

$:= \; |0\cdots0\rangle\langle0\cdots0| + e^{i\alpha}\,|1\cdots1\rangle\langle1\cdots1|$
- X-spiders 

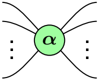
$:= \; |+\cdots+\rangle\langle+\cdots+| + e^{i\alpha}\,|-\cdots-\rangle\langle-\cdots-|$

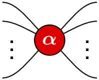
# ZX-diagrams

- ZX-diagrams consist of two types of maps
- Z-spiders  $:= \ |0\cdots0\rangle\langle0\cdots0| + e^{i\alpha}\,|1\cdots1\rangle\langle1\cdots1|$
- X-spiders  $:= \ |+\cdots+\rangle\langle+\cdots+| + e^{i\alpha}\,|-\cdots-\rangle\langle-\cdots-|$
- By wiring these together, we can make arbitrary linear maps between qubits. For instance:

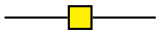
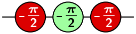


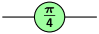
# ZX-diagrams

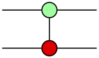
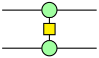
- ZX-diagrams consist of two types of maps
- Z-spiders 

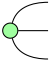
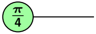
$:= |0\cdots0\rangle\langle0\cdots0| + e^{i\alpha}|1\cdots1\rangle\langle1\cdots1|$
- X-spiders 

$:= |+\cdots+\rangle\langle+\cdots+| + e^{i\alpha}|-\cdots-\rangle\langle-\cdots-|$
- By wiring these together, we can make arbitrary linear maps between qubits. For instance:

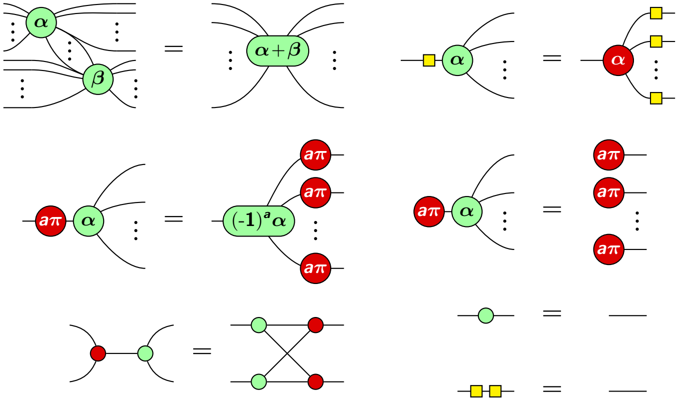
H =  = 

T = 


CNOT =  CZ = 

But also: GHZ =  T magic state = 

# ZX-calculus




$\alpha, \beta \in [0, 2\pi]$, $a \in \{0, 1\}$

# Circuit optimization with the ZX-calculus

- Write your circuit as a ZX-diagram.

# Circuit optimization with the ZX-calculus

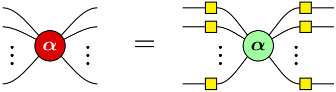
- Write your circuit as a ZX-diagram.
- Apply rewrite rules to simplify it.

# Circuit optimization with the ZX-calculus

- Write your circuit as a ZX-diagram.
- Apply rewrite rules to simplify it.
- Turn the resulting diagram back into a circuit.

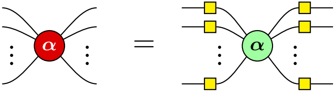
# First steps: graph-like ZX-diagrams

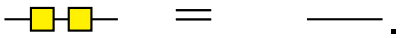
- First turn all X-spiders into Z-spiders:



# First steps: graph-like ZX-diagrams

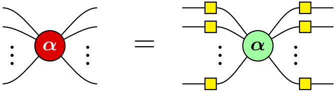
- First turn all X-spiders into Z-spiders:

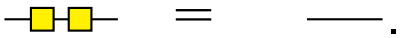



- Cancel all double Hadamards: 

# First steps: graph-like ZX-diagrams

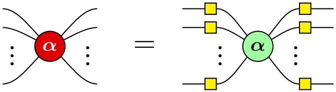
- First turn all X-spiders into Z-spiders:

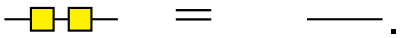


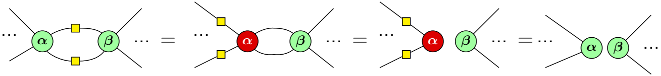

- Cancel all double Hadamards: 
- Fuse all adjacent spiders.

# First steps: graph-like ZX-diagrams

- First turn all X-spiders into Z-spiders:

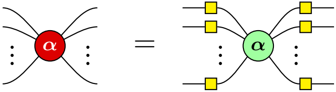


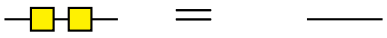

- Cancel all double Hadamards: 
- Fuse all adjacent spiders.
- Cancel parallel connections:

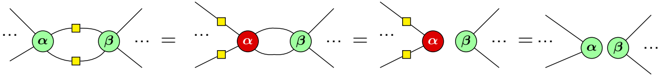


# First steps: graph-like ZX-diagrams

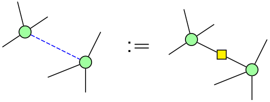
- First turn all X-spiders into Z-spiders:



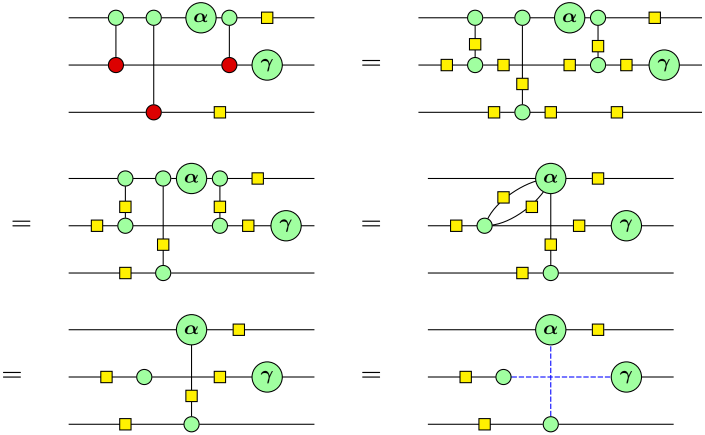

- Cancel all double Hadamards: .
- Fuse all adjacent spiders.
- Cancel parallel connections:



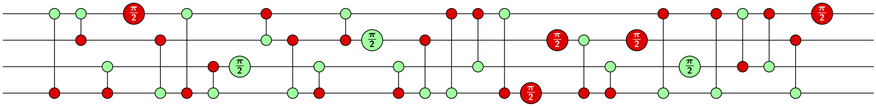

- Use new notation:



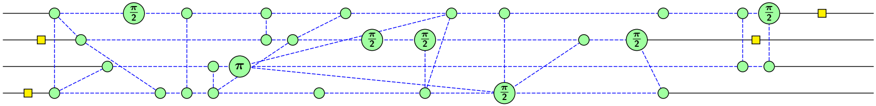
# Example



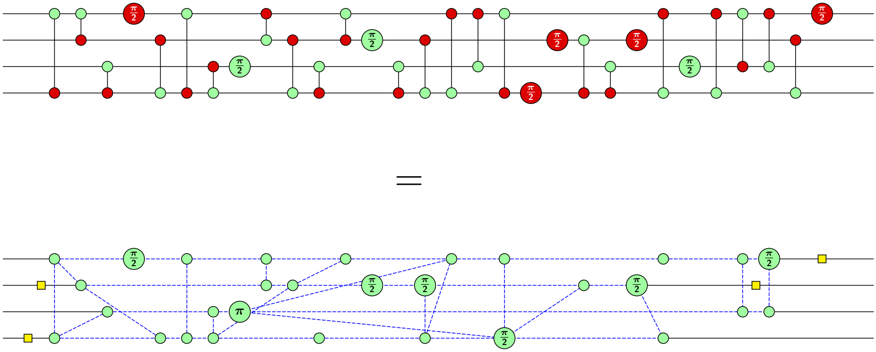
# More involved example




=

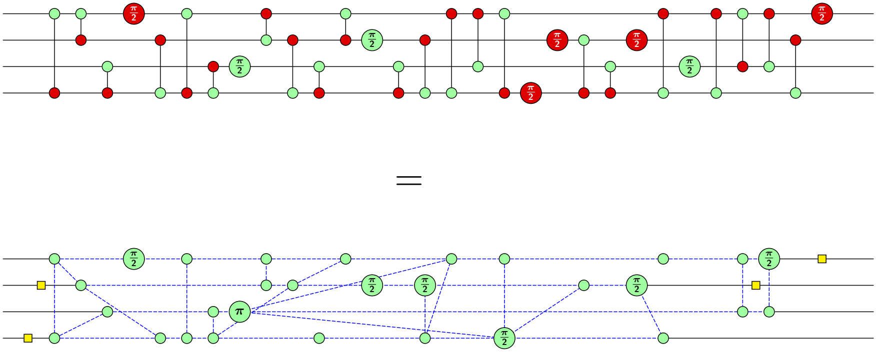


# More involved example



We call these diagrams *graph-like*.

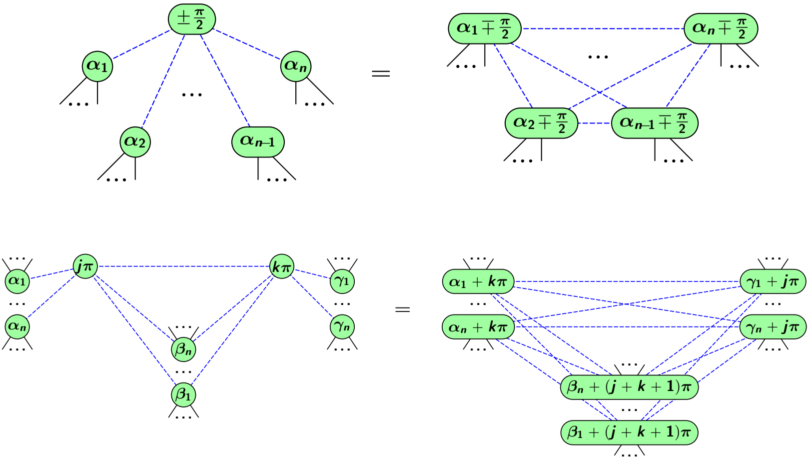
# More involved example




We call these diagrams *graph-like*.

To simplify these diagrams, we want to remove as many interior vertices as possible.

# Local complementation and pivoting



# Simplification so far

- Convert diagram into graph-like diagram.

# Simplification so far

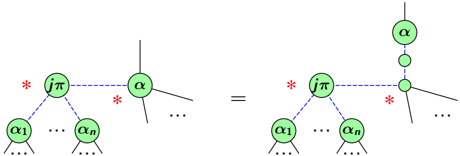
- Convert diagram into graph-like diagram.
- Remove all internal $\pm\pi/2$ spiders by local complementation.

# Simplification so far

- Convert diagram into graph-like diagram.
- Remove all internal $\pm\pi/2$ spiders by local complementation.
- Remove all connected internal $a\pi$ spiders by pivoting.

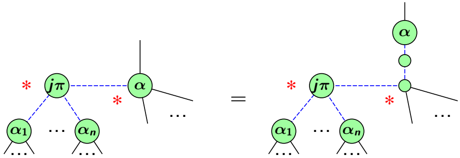
# Simplification so far

- Convert diagram into graph-like diagram.
- Remove all internal $\pm\pi/2$ spiders by local complementation.
- Remove all connected internal $a\pi$ spiders by pivoting.
- Remove internal $a\pi$ spider connected to boundary by unfusing and pivoting:



# Simplification so far

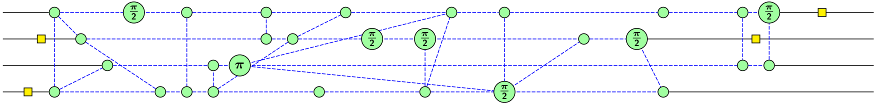
- Convert diagram into graph-like diagram.
- Remove all internal $\pm\pi/2$ spiders by local complementation.
- Remove all connected internal $a\pi$ spiders by pivoting.
- Remove internal $a\pi$ spider connected to boundary by unfusing and pivoting:




- If the original diagram was Clifford, then the simplified diagram has no internal spiders

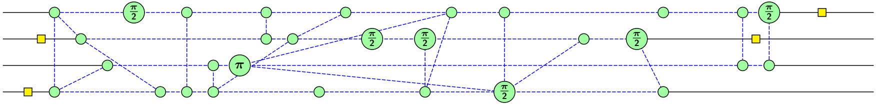
# Clifford example

Recall the example Clifford diagram:

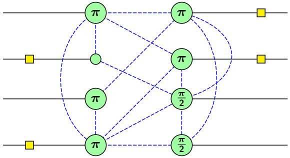


# Clifford example

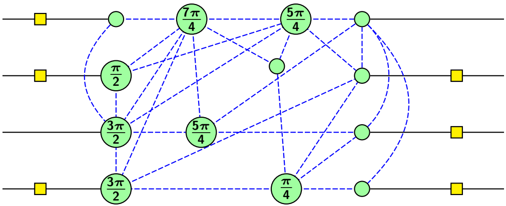
Recall the example Clifford diagram:



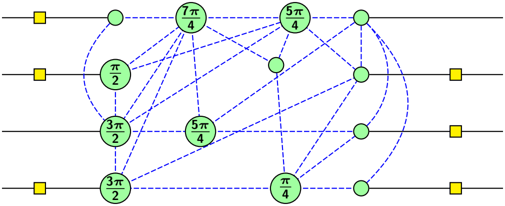

This can be reduced by the described procedure to:



# Clifford+T example

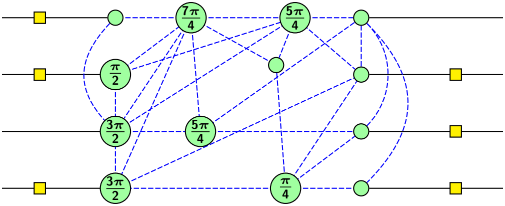


# Clifford+T example




Question: How do we turn this into a circuit.

# Clifford+T example




Question: How do we turn this into a circuit.
Answer: We use the fact that it has a *gFlow*.

# gFlow

Informally, a gFlow associates an ‘arrow of time’ with a graph.
Circuits have a gFlow.

# gFlow

Informally, a gFlow associates an ‘arrow of time’ with a graph.
Circuits have a gFlow.

### Proposition

Local complementation and pivoting preserve gFlow

# gFlow

Informally, a gFlow associates an 'arrow of time' with a graph. Circuits have a gFlow.

### Proposition

Local complementation and pivoting preserve gFlow

### Theorem

There is an efficient procedure that transforms a ZX-diagram with a gFlow into a circuit.

# The simplification procedure

- We indeed have a circuit-to-circuit simplification procedure using the ZX-calculus.
- It reduces Clifford circuits to a quasi normal-form.

# The simplification procedure

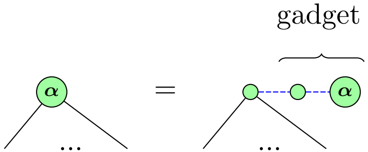
- We indeed have a circuit-to-circuit simplification procedure using the ZX-calculus.
- It reduces Clifford circuits to a quasi normal-form.
- But: T gates never get removed by lcomp and pivoting.

# The simplification procedure

- We indeed have a circuit-to-circuit simplification procedure using the ZX-calculus.
- It reduces Clifford circuits to a quasi normal-form.
- But: T gates never get removed by lcomp and pivoting.
- So:
  To do significant T-count optimization, we need to do better.

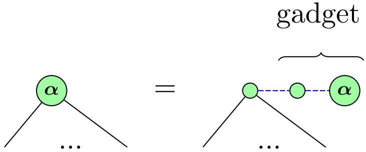
# Gadgetization

We turn all non-Clifford spiders into *phase gadgets*:



# Gadgetization

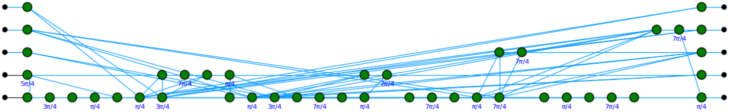
We turn all non-Clifford spiders into *phase gadgets*:




This makes the base of the gadget available for pivoting.

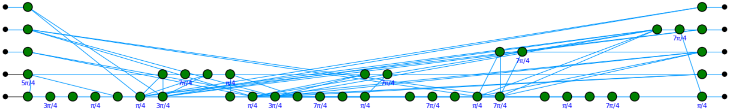
# Example: Gadgetization and pivoting

After the first round of simplifications:

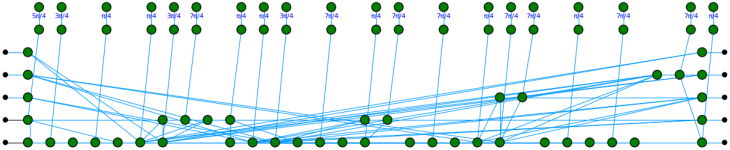


# Example: Gadgetization and pivoting

After the first round of simplifications:

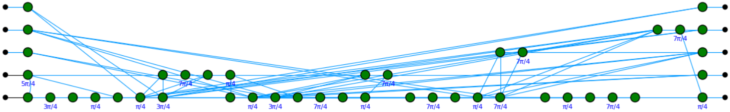


After gadgetization:

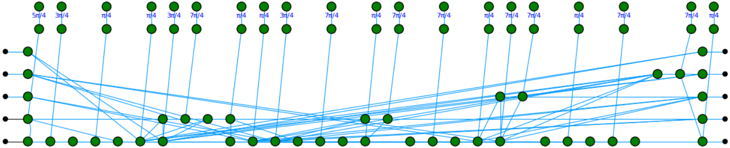


# Example: Gadgetization and pivoting

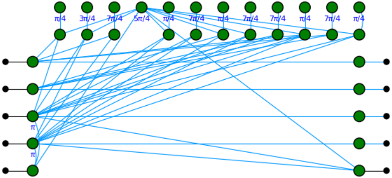
After the first round of simplifications:



After gadgetization:

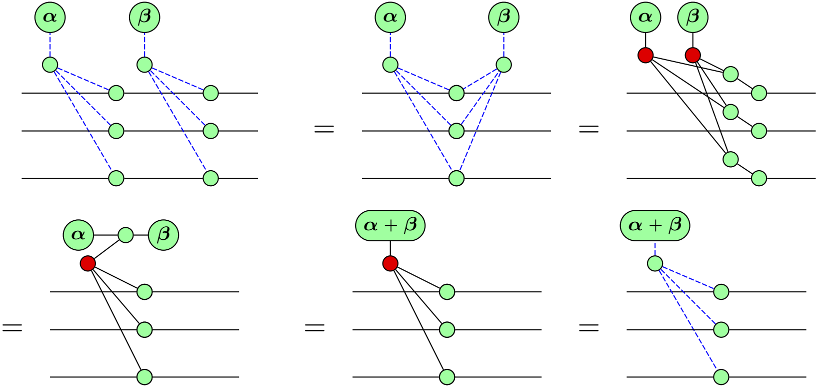


After pivots/lcomps:



# Final step: phase gadget fusion

Whenever phase gadgets have the same set of neighbours, they can fuse:



Spider fusion + Local complementation + Pivoting
+ Gadgetization + Gadget fusion
=
State-of-the-art T-count optimization

# Circuit extraction

Problem: We still need to get a circuit out of the diagram.

# Circuit extraction

Problem: We still need to get a circuit out of the diagram.

The good news: We have 'heuristics' that always seem to work.

# Circuit extraction

Problem: We still need to get a circuit out of the diagram.

The good news: We have 'heuristics' that always seem to work.
The bad news: We don't know why.

# Demonstration of PyZX

# Conclusion and future work

The Takeaway:

The ZX-calculus is very useful for
T-count optimization

# Conclusion and future work

The Takeaway:

The ZX-calculus is very useful for
T-count optimization

Open problems:

- Why does our circuit extraction work?

# Conclusion and future work

The Takeaway:

The ZX-calculus is very useful for T-count optimization

Open problems:

- Why does our circuit extraction work?
- How to use phase-polynomial methods on ZX-diagrams?

# Conclusion and future work

The Takeaway:

The ZX-calculus is very useful for T-count optimization

Open problems:

- Why does our circuit extraction work?
- How to use phase-polynomial methods on ZX-diagrams?
- Is the ZH-calculus useful?

Thank you for your attention