Quantum circuits: From Structure to Software

Aleks Kissinger

Quantum Natural Language Processing, Oxford 2020

Quantum software

1. := the code the runs on a quantum computer

factoring

search

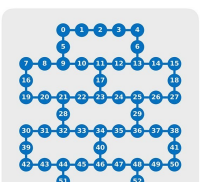
physical simulation
optimisation problems
linear systems & codes
network flows
natural language processing
...

2. := the code that makes that code (better)
   - **compilers**
   - **optimisation**
   - **verification**

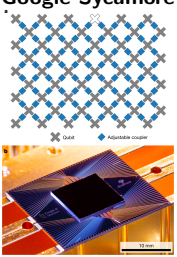
Problem: ~~no quantum computers~~

Rigetti Aspen

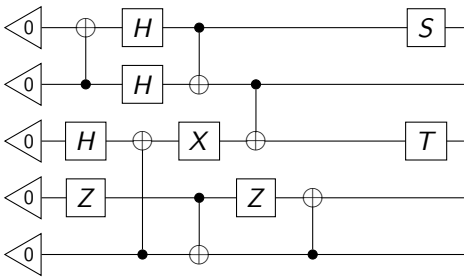
IBM Q Rochester

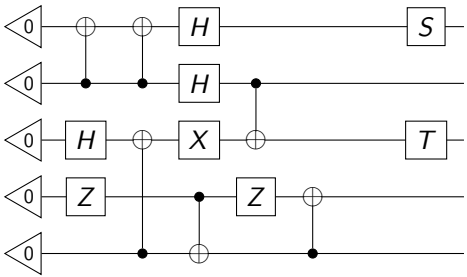


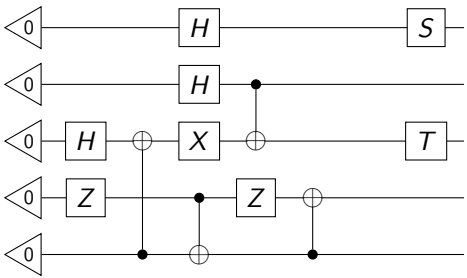
Google Sycamore

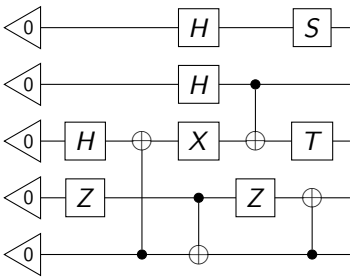


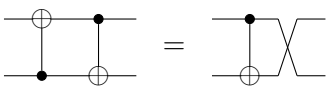
(+ Oxford, Vienna, Delft, Sussex, Grenoble...)

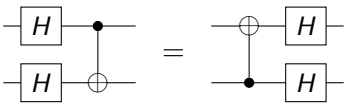

0
0
0
0
0
H
H
H
X
Z
Z
S
T

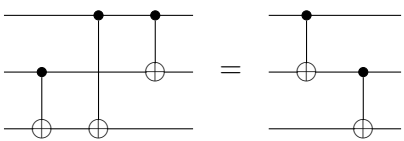

0
0
0
0
0
H
H
H
X
Z
Z
S
T

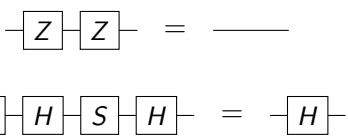

0
0
0
0
0
H
H
H
X
Z
Z
S
T


0
0
0
0
0
H
H
H
X
S
T
Z
Z

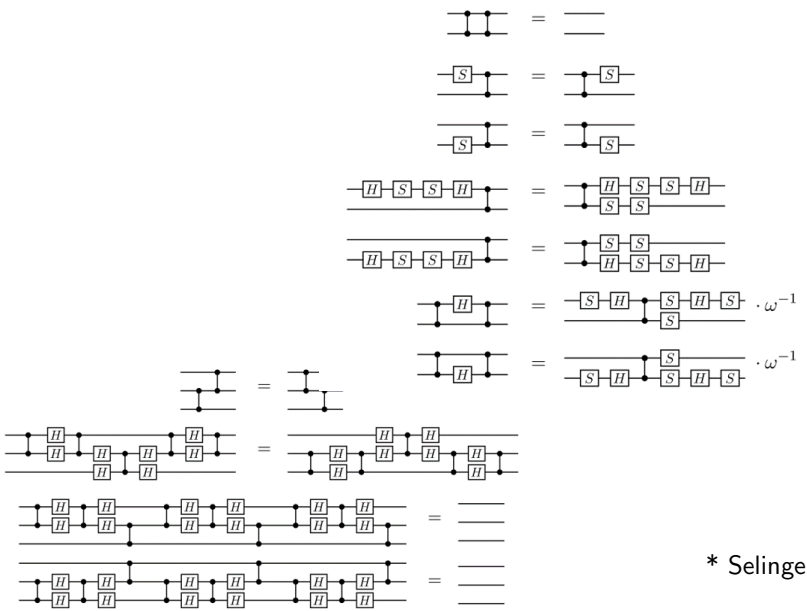






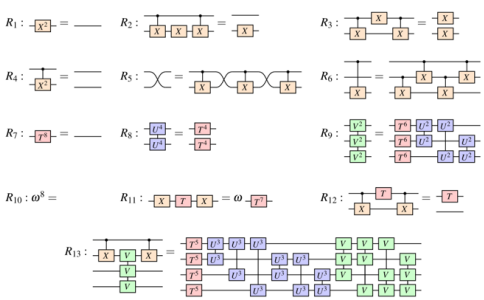


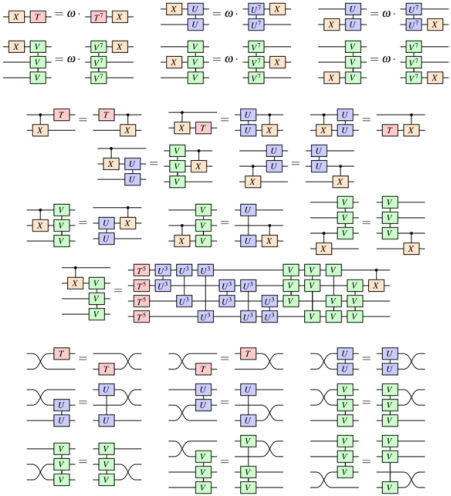
. . .



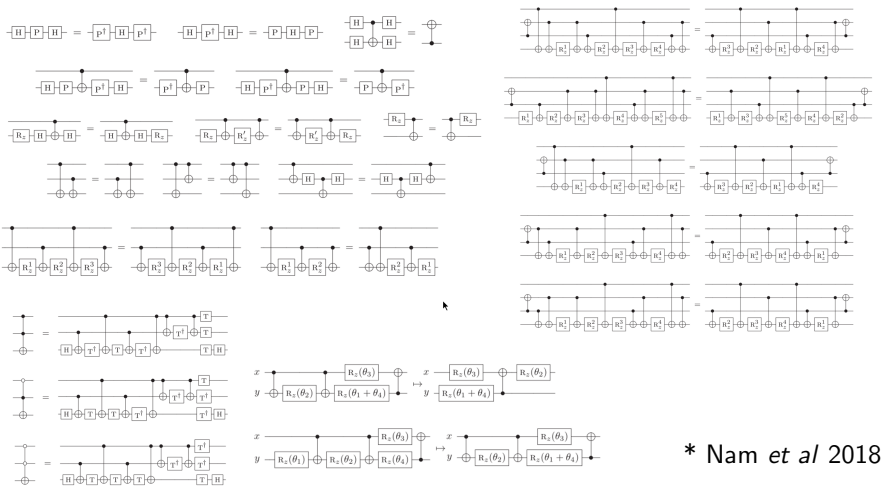
* Selinger 2015

* Selinger 2015

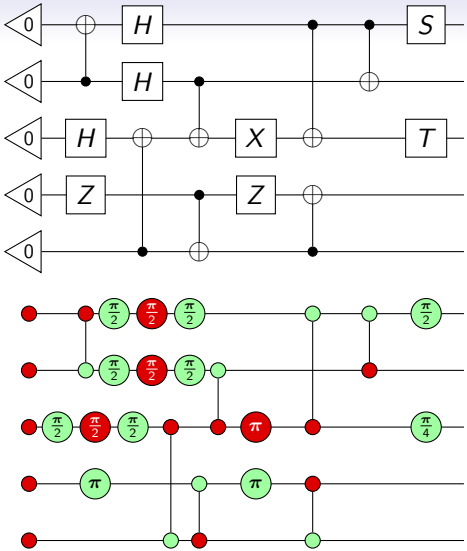


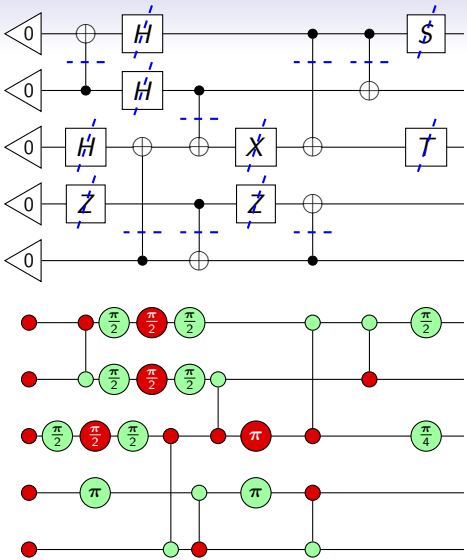


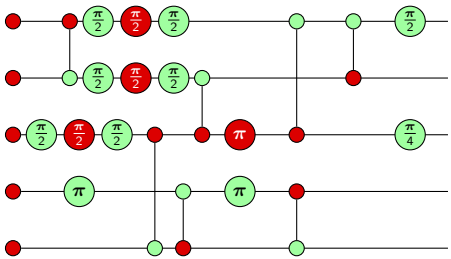
* Amy, Chen, & Ross 2018

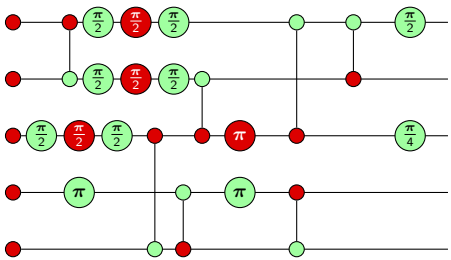


* Nam *et al* 2018


0
0
0
0
0
H
H
H
Z
X
Z
S
T
π/2
π
π/4


0
0
0
0
0
H
H
H
Z
X
Z
S
T
π/2
π
π/4

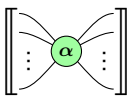

π/2
π/2
π/2
π/2
π/2
π/2
π/2
π/2
π/2
π/2
π/2
π/2
π
π/4
π
π



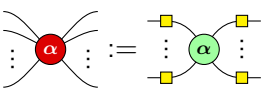
ZX-diagram

ZX-diagrams

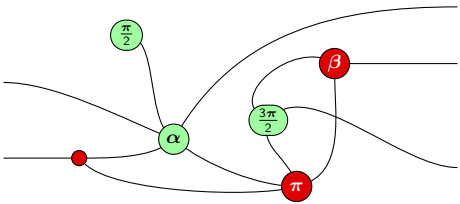
...are made of *spiders*:

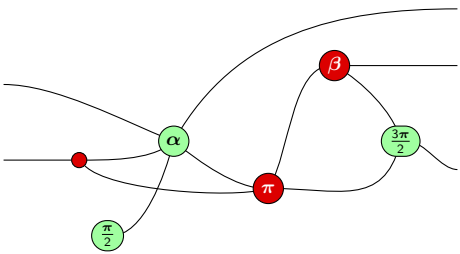


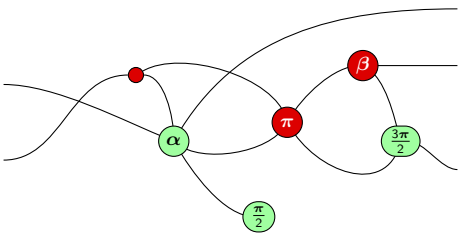
$$= \begin{pmatrix} 1 & 0 & & \\ 0 & 0 & \ddots & \\ & \ddots & 0 & 0 \\ & & 0 & e^{i\alpha} \end{pmatrix}$$

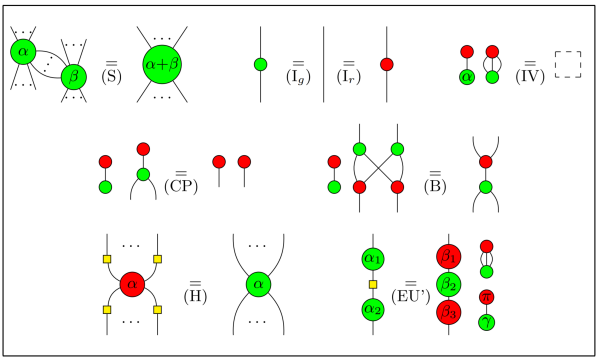


$$[\![\text{—□—}]\!] := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$


$\frac{\pi}{2}$
α
β
$\frac{3\pi}{2}$
π


β
α
3π/2
π
π/2


β
π
α
3π/2
π/2

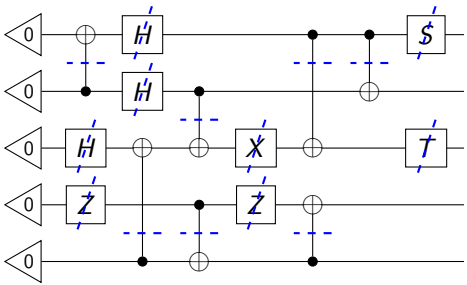


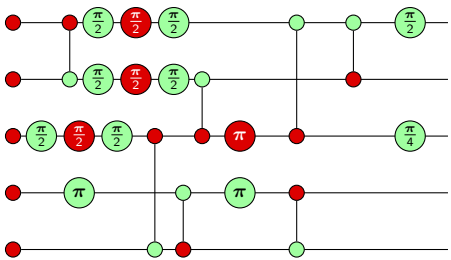
[1]

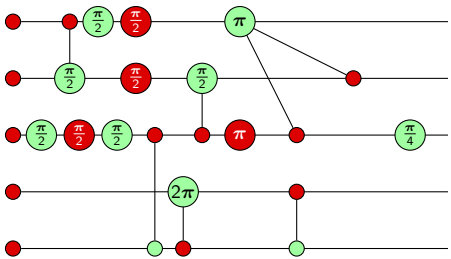
ZX calculus

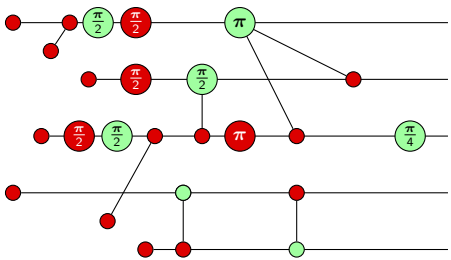
these 8 rules $\Longrightarrow$ everything before

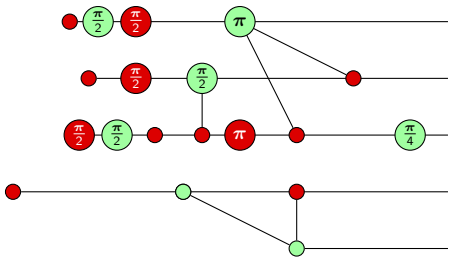
[1] Vilmart 2018. arXiv:1812.09114

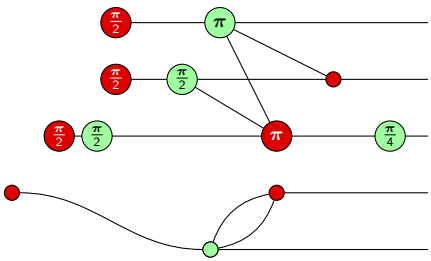

0
0
0
0
0
H
H
H
Z
X
Z
S
T

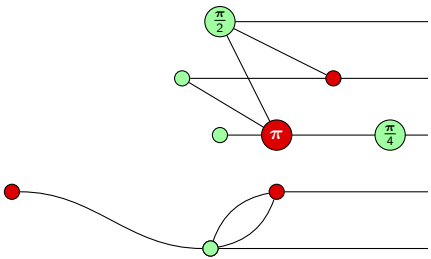

π/2
π/2
π/2
π/2
π/2
π/2
π/2
π/2
π/2
π/2
π/2
π
π/4
π
π

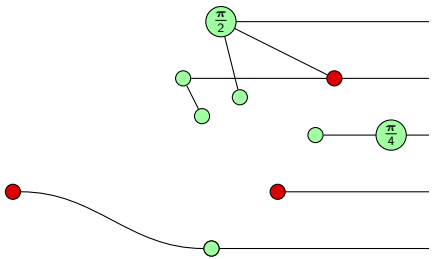

$\frac{\pi}{2}$
$\frac{\pi}{2}$
π
$\frac{\pi}{2}$
$\frac{\pi}{2}$
$\frac{\pi}{2}$
$\frac{\pi}{2}$
$\frac{\pi}{2}$
$\frac{\pi}{2}$
π
$\frac{\pi}{4}$
2π

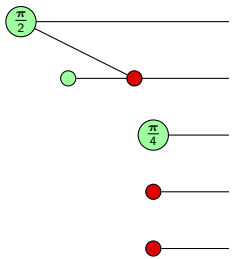

$\frac{\pi}{2}$
$\frac{\pi}{2}$
π
$\frac{\pi}{2}$
$\frac{\pi}{2}$
$\frac{\pi}{2}$
$\frac{\pi}{2}$
π
$\frac{\pi}{4}$

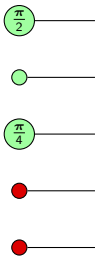

π/2
π/2
π
π/2
π/2
π/2
π/2
π
π/4


π/2
π
π/2
π/2
π/2
π/2
π
π/4

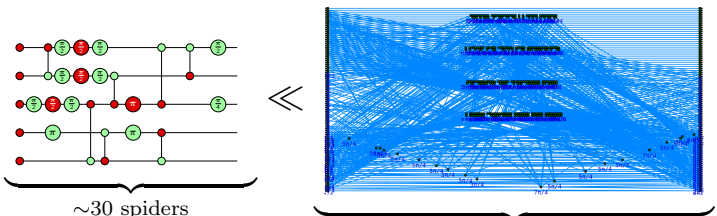

π/2
π
π/4


π/2
π/4


$\frac{\pi}{2}$
$\frac{\pi}{4}$

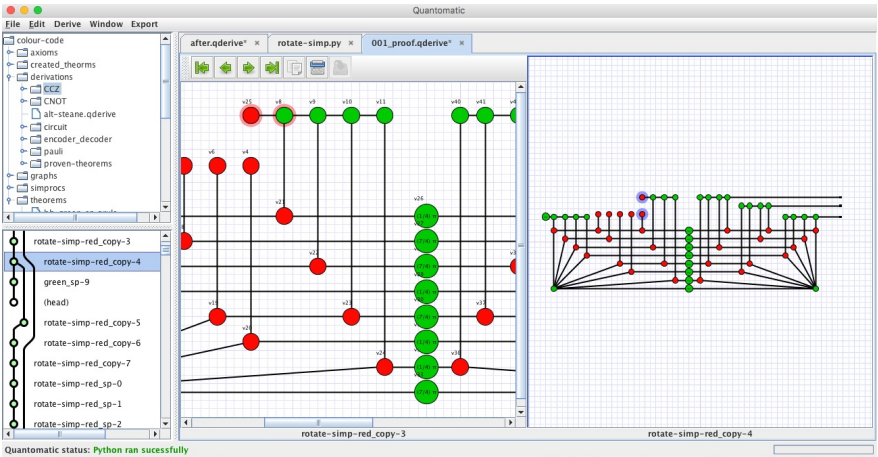

$\frac{\pi}{2}$
$\frac{\pi}{4}$

Q: How do we scale up?



A: Automation.

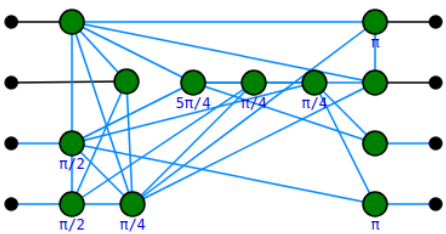
Quantomatic



- General-purpose, **medium scale** tool for teaching and experimenting with graphical calculi

PyZX

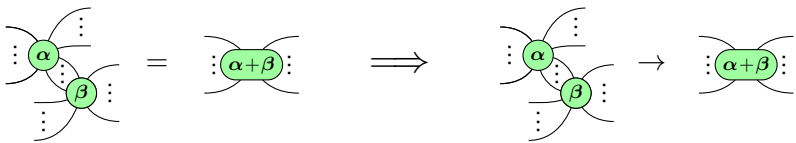
```
In [11]: zx.simplify.clifford_simp(g)
         g.normalise()
         zx.d3.draw(g)
```



- Special-purpose, **large scale** circuit optimisation tool for ZX-calculus

The idea

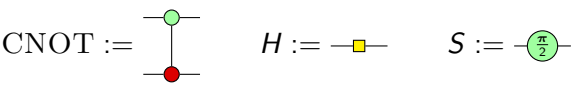
1. Turn *equations* into directed *rewrite rules*



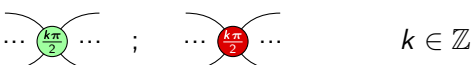
2. Use rewrite rules to *simplify* ZX-diagrams
3. *Extract* meaningful data from simplified diagram
   (e.g. optimised circuits, amplitudes/probabilities, ...).

Clifford circuits

- Built from:

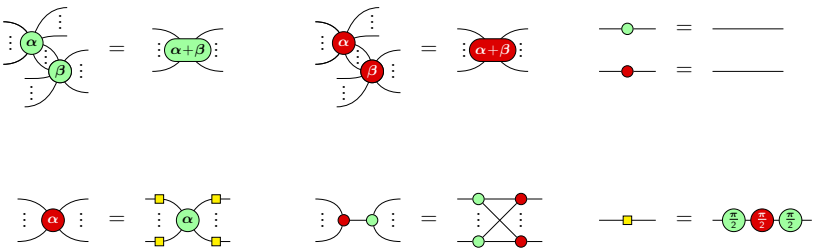


- Efficient to classically simulate (Gottesman-Knill)
- Expressible by *Clifford ZX-diagrams*



$k \in \mathbb{Z}$

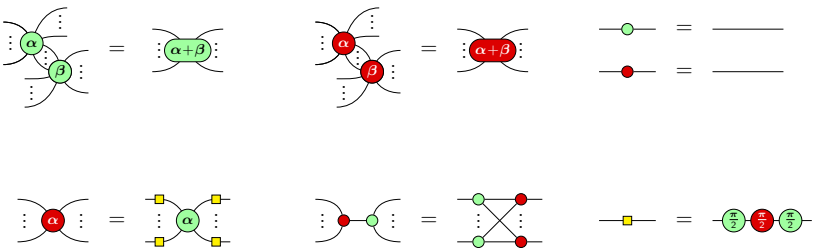
Clifford ZX-calculus



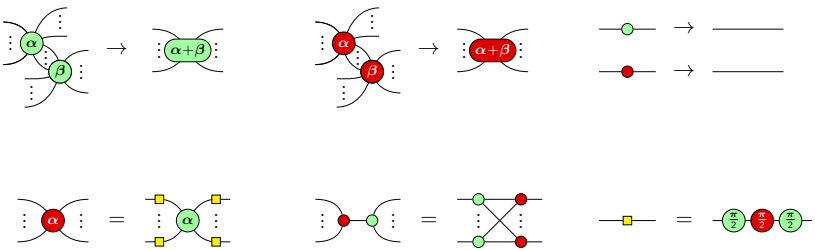
complete for Clifford ZX-diagrams

Equations $\rightsquigarrow$ Rewrite rules

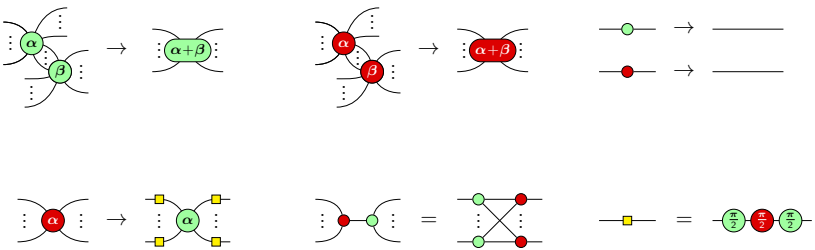
Equations $\rightsquigarrow$ Rewrite rules

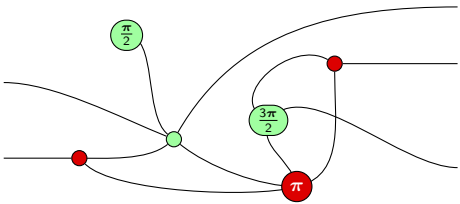


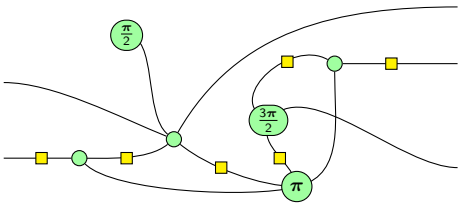
Equations $\rightsquigarrow$ Rewrite rules

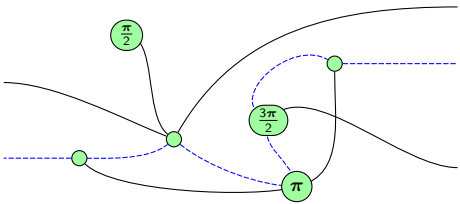


Equations $\rightsquigarrow$ Rewrite rules

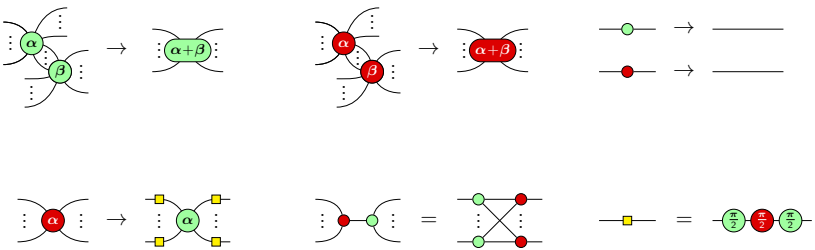



π/2
3π/2
π

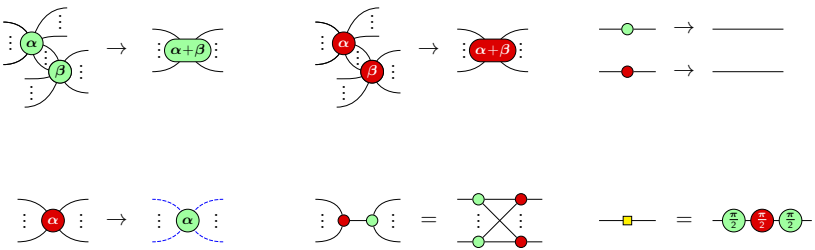

π/2
3π/2
π


$\frac{\pi}{2}$
$\frac{3\pi}{2}$
π

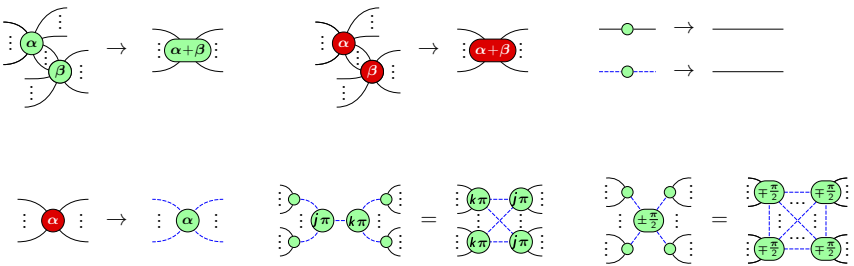
Equations $\rightsquigarrow$ Rewrite rules



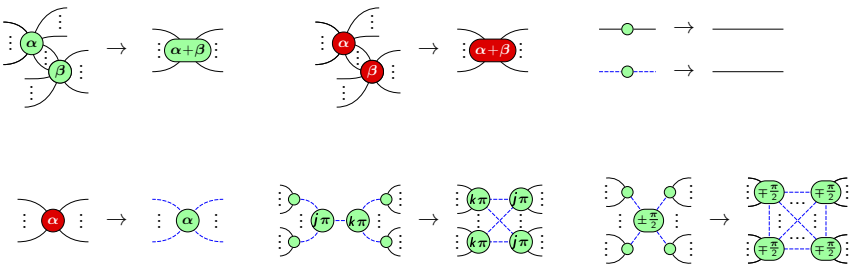
Equations $\rightsquigarrow$ Rewrite rules



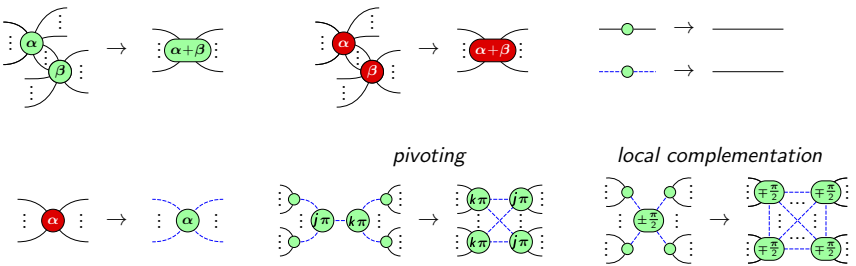
Equations $\rightsquigarrow$ Rewrite rules



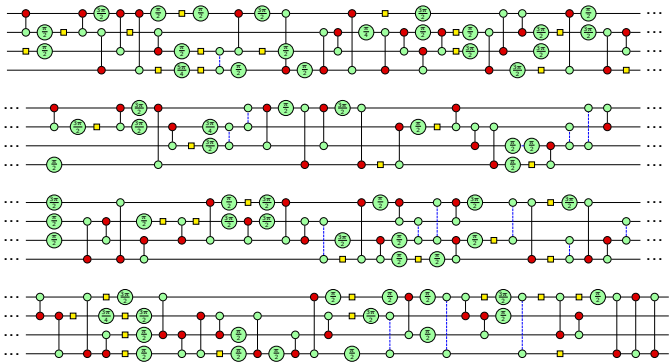
Equations $\rightsquigarrow$ Rewrite rules



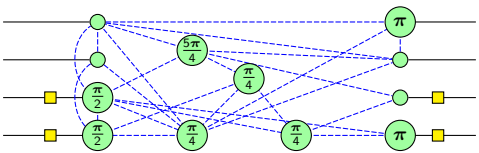
Equations $\rightsquigarrow$ Rewrite rules



Computing the skeleton



ZX-diagram $\rightsquigarrow$ circuit?



- ZX-diagrams can represent any linear map, but circuits are always unitary
- $\Longrightarrow$ not all ZX-diagrams are equal to circuits
- All *unitary* ZX-diagrams can be written as circuits, but at what cost???

Problem (Circuit extraction problem)

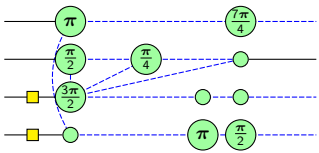
Given a ZX-diagram that represents a unitary linear map, find an equivalent quantum circuit.

Conjecture

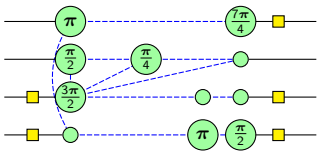
If the circuit extraction problem is solvable with polynomial overhead, then $BQP = PostBQP$.

$\Longrightarrow$ need heuristics or **extra data**

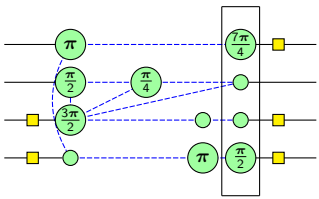
Circuit extraction strategy



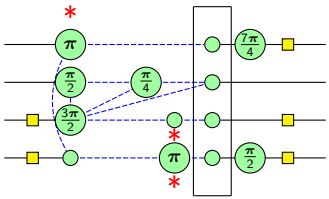
Circuit extraction strategy



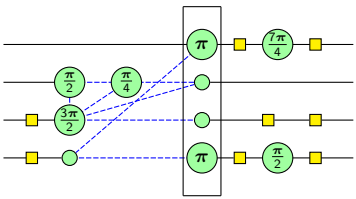
Circuit extraction strategy



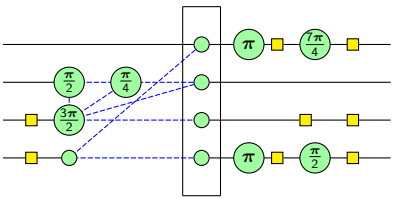
Circuit extraction strategy



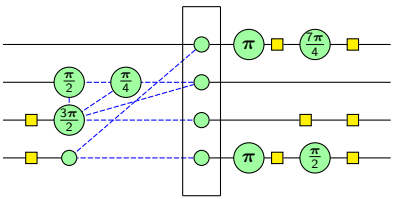
Circuit extraction strategy



Circuit extraction strategy

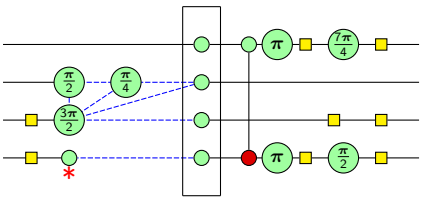


Circuit extraction strategy



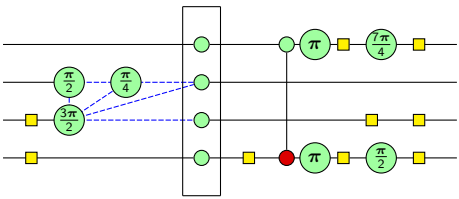
$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Circuit extraction strategy

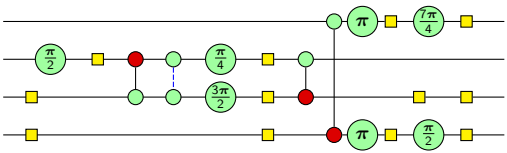


$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \xrightarrow{R_1 := R_1 + R_4} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Circuit extraction strategy



Circuit extraction strategy

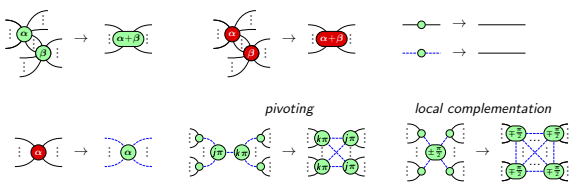


Extraction and gFlow

- Strategy always works iff the graph of a ZX-diagram has *generalised flow* (gFlow)

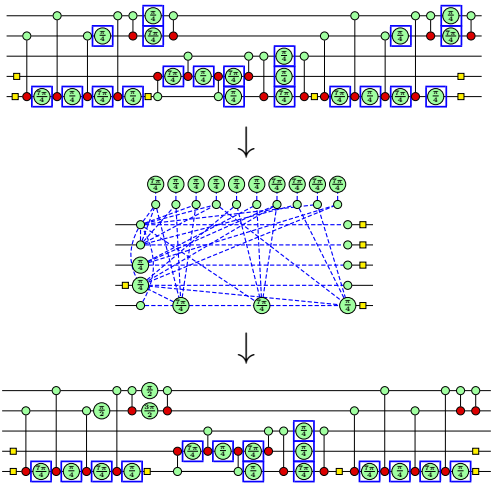
$$\exists \prec, g. \forall v.\, g(v) \subseteq v^{\succ} \;\wedge\; \mathbf{Odd}(v) \cap \bar{I} \cap v^{\prec} = \{v\}$$

- quantum circuits have gFlow
- + these rules all preserve gFlow:



- $\Longrightarrow$ skeletons can always be extracted into circuits

T-count reduction

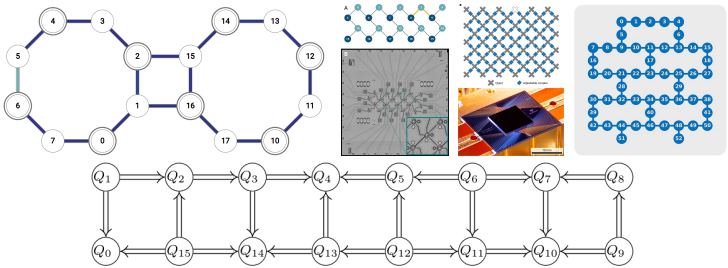


T-count reduction

Circuit	n	T	Best prev.	Method	PyZX
adder$_8$	**24**	**399**	**213**	**RM$_m$**	**173**
Adder8	23	266	56	NRSCM	56
Adder16	47	602	120	NRSCM	120
Adder32	95	1274	248	NRSCM	248
Adder64	191	2618	504	NRSCM	504
barenco-tof3	5	28	16	Tpar	16
barenco-tof4	7	56	28	Tpar	28
barenco-tof5	9	84	40	Tpar	40
barenco-tof10	19	224	100	Tpar	100
tof$_3$	5	21	15	Tpar	15
tof$_4$	7	35	23	Tpar	23
tof$_5$	9	49	31	Tpar	31
tof$_{10}$	19	119	71	Tpar	71
csla-mux$_3$	*15*	*70*	*58*	*RM$_t$*	*62*
csum-mux$_9$	*30*	*196*	*76*	*RM$_t$*	*84*
cycle17$_3$	**35**	**4739**	**1944**	**RM$_m$**	**1797**
gf(2^4)-mult	*12*	*112*	*56*	*TODD*	*68*
gf(2^5)-mult	*15*	*175*	*90*	*TODD*	*115*
gf(2^6)-mult	*18*	*252*	*132*	*TODD*	*150*
gf(2^7)-mult	*21*	*343*	*185*	*TODD*	*217*
gf(2^8)-mult	*24*	*448*	*216*	*TODD*	*264*
ham15-low	17	161	97	Tpar	97
ham15-med	**17**	**574**	**230**	**Tpar**	**212**
ham15-high	20	2457	1019	Tpar	1019
hwb$_6$	7	105	75	Tpar	75
hwb$_8$	**12**	**5887**	**3531**	**RM$_{m\&r}$**	**3517**
mod-mult-55	*9*	*49*	*28*	*TODD*	*35*
mod-red-21	11	119	73	Tpar	73
mod5$_4$	**5**	**28**	**16**	**Tpar**	**8**
nth-prime$_6$	**9**	**567**	**400**	**RM$_{m\&r}$**	**279**
nth-prime$_8$	*12*	*6671*	*4045*	*RM$_{m\&r}$*	*4047*
qcla-adder$_{10}$	36	589	162	Tpar	162
qcla-com$_7$	*24*	*203*	*94*	*RM$_m$*	*95*
qcla-mod$_7$	26	413	235	NRSCM	237
rc-adder$_6$	14	77	47	RM$_{m\&r}$	47
vbe-adder$_3$	10	70	24	Tpar	24

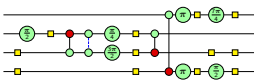
- Compared with state of the art for (ancilla-free) T-count reduction:
  - Amy, Maslov, Mosca. 2014
  - Amy, Mosca. 2016
  - Heyfron, Campbell. 2018
  - Nam *et al.* 2018
- As of March 2019:
  - $26/36 \approx 72\%$ **match** SotA
  - $6/36 \approx 17\%$ **beat** SotA
- In April 2019, Zhang & Chen matched PyZX numbers with circuit method
- In Nov 2019, de Beaudrap, Bian, & Wang beat SotA again with new ZX method + TODD

Circuit routing



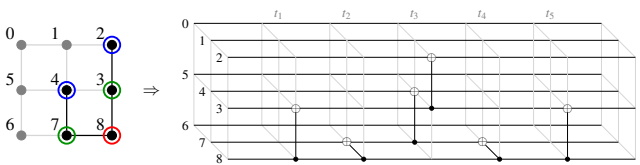
Circuit routing

PyZX uses Gaussian elimination to extract circuits:



$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \xrightarrow{R_1 := R_1 + R_4} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

By constrained GE, we produce **routed** circuits:



CNOT Circuit routing

Architecture	CNOT	QuilC	PyZX	Savings
9q-square	3	3.8	**3**	21.05%
9q-square	5	10.82	**5.2**	51.94%
9q-square	10	20.08	**11.6**	42.23%
9q-square	20	46.24	**25.85**	44.10%
9q-square	30	72.89	**35.55**	51.23%
16q-square	4	6.14	**4.44**	27.69%
16q-square	8	19.68	**12.41**	36.94%
16q-square	16	48.13	**33.08**	31.27%
16q-square	32	106.75	**82.95**	22.30%
16q-square	64	225.69	**147.38**	34.70%
16q-square	128	457.35	**168.12**	63.24%
16q-square	256	925.85	**169.28**	81.72%
rigetti-16q-aspen	4	7.05	**4.15**	41.13%
rigetti-16q-aspen	8	28.2	**11.22**	60.21%
rigetti-16q-aspen	16	69.15	**33.95**	50.90%
rigetti-16q-aspen	32	147.3	**101.75**	30.92%
rigetti-16q-aspen	64	324.6	**189.15**	41.73%
rigetti-16q-aspen	128	664.65	**220.75**	66.79%
rigetti-16q-aspen	256	1367.89	**222.15**	83.76%
ibm-qx5	4	6.75	**4**	40.74%
ibm-qx5	8	23.7	**8.95**	62.24%
ibm-qx5	16	60.5	**26.55**	56.12%
ibm-qx5	32	140.05	**84.4**	39.74%
ibm-qx5	64	301.05	**152.65**	49.29%
ibm-qx5	128	600.9	**188.25**	68.67%
ibm-qx5	256	1247.8	**193.8**	**84.47%**
ibm-q20-tokyo	4	5.5	**4**	27.27%
ibm-q20-tokyo	8	17.3	**7.69**	55.55%
ibm-q20-tokyo	16	43.83	**20.44**	53.37%
ibm-q20-tokyo	32	93.58	**66.93**	28.48%
ibm-q20-tokyo	64	215.9	**165.6**	23.30%
ibm-q20-tokyo	128	432.65	**237.64**	45.07%
ibm-q20-tokyo	256	860.74	**245.84**	71.44%

- up to 5X more efficient vs. Rigetti, CQC, IBM (as of April 2019)

Verification

- **Q:** How do we verify correctness of optimised circuits?

$$C \rightsquigarrow D \qquad \Longrightarrow \qquad \underbrace{[\![C]\!] = [\![D]\!]}_{\text{QMA-complete}}$$

- PyZX optimiser is self-checking

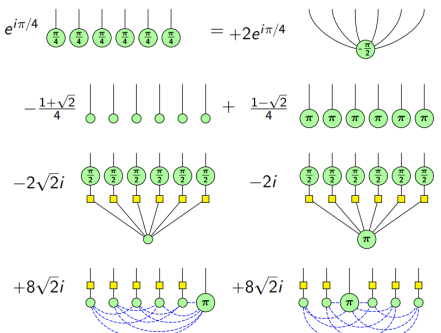
$$C \rightsquigarrow D \qquad \Longrightarrow \qquad \underbrace{C; D^{-1} \rightsquigarrow 1}_{\sim O(n^{1.5})}$$

- *Sound* approx. of circuit equality helps **find bugs**:

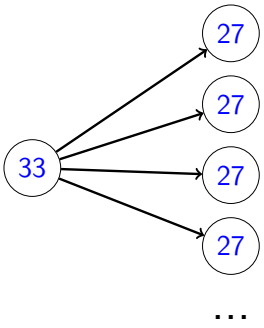
qcla-com$_7$	24	203	94	RM$_m$	95
qcla-mod$_7$	26	413	235**(X)**	NRSCM	237
rc-adder6	14	77	47	RM$_n$	47

Classical simulation of circuits

- The ZX-calculus can 'supercharge' *stabilizer rank* techniques, which decompose blocks of non-Clifford gates into sums of stabilizer circuits
- e.g. Bravyi/Smith/Smolin decomposition in ZX-language is:



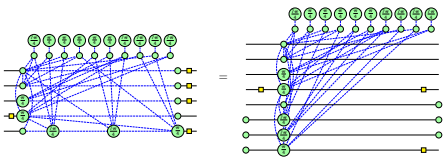
Classical simulation of circuits



- Produces $7^{n/6} \approx 2^{0.468n}$ terms for n T-gates
- Interleaving ZX-simplification can make that signficantly less

Where to?

- More aggressive optimisation $\rightsquigarrow$ more difficult circuit extraction



- Circuit routing for general circuits
- "Quantum static analysis" to find bugs in circuits
- Classical simulation techniques + data

Thanks!

- *Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus.* **Ross Duncan**, Simon Perdrix, AK, John van de Wetering. arXiv:1902.03178 [quant-ph]
- *Reducing T-count with the ZX-calculus.* AK, **John van de Wetering** arXiv:1903.10477 [quant-ph]
- *CNOT circuit extraction for topologically-constrained quantum memories.*AK, **Arianne Meijer-van de Griend** arXiv:1903.00633 [quant-ph]

Quantomatic: quantomatic.github.io

PyZX: github.com/Quantomatic/pyzx