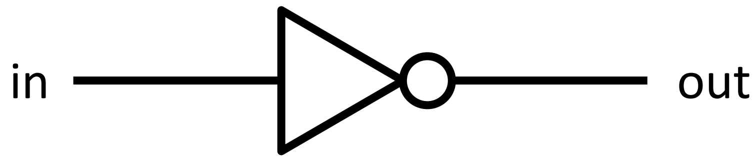
Generalising ZX-calculus for efficient parameter sampling

Matthew Sutcliffe

University of Oxford

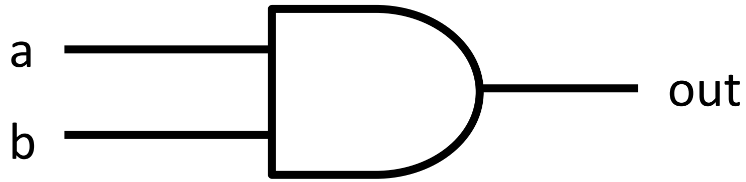
Classical Logic Gates

“NOT”



in	out
0	1
1	0

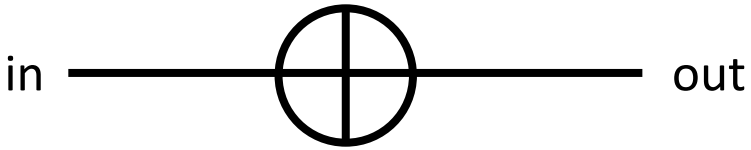
“AND”



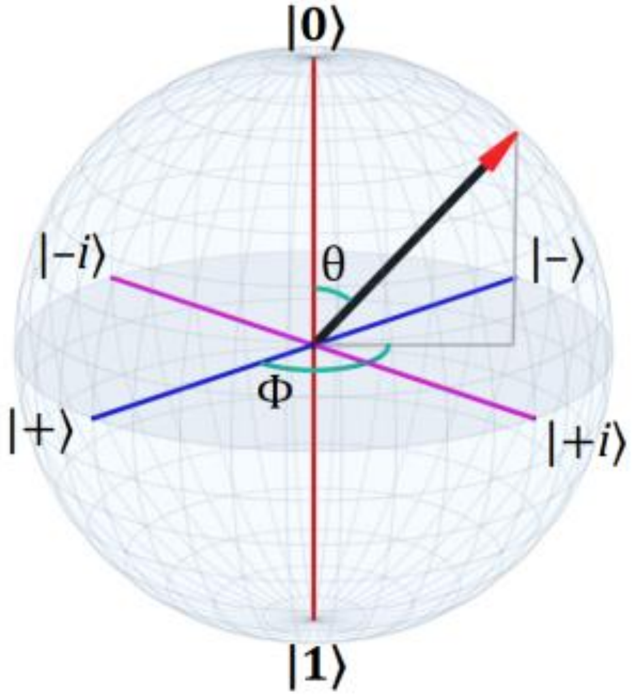
a	b	out
0	0	0
0	1	0
1	0	0
1	1	1

Quantum Gates

"Pauli-X"

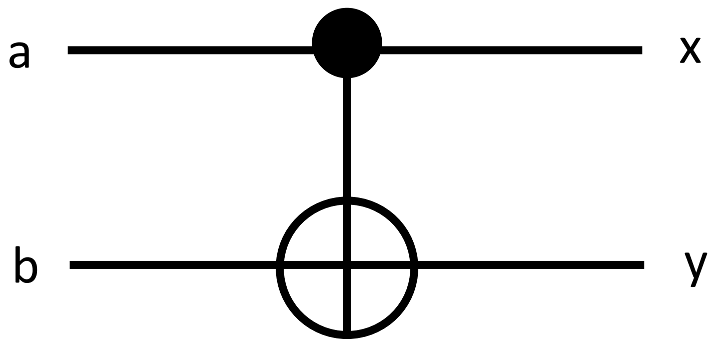


$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$



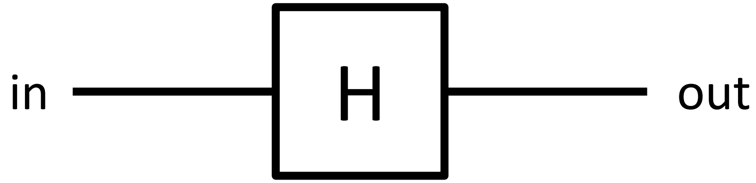
Quantum Gates

“CNOT”

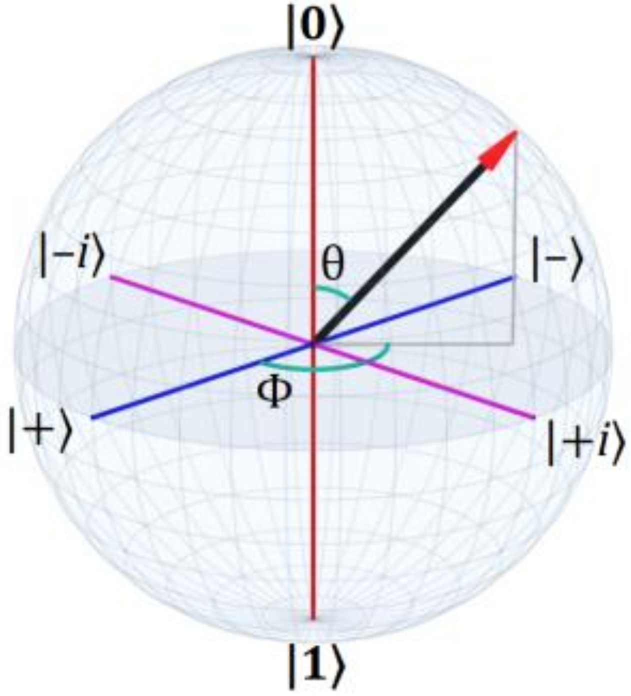


$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

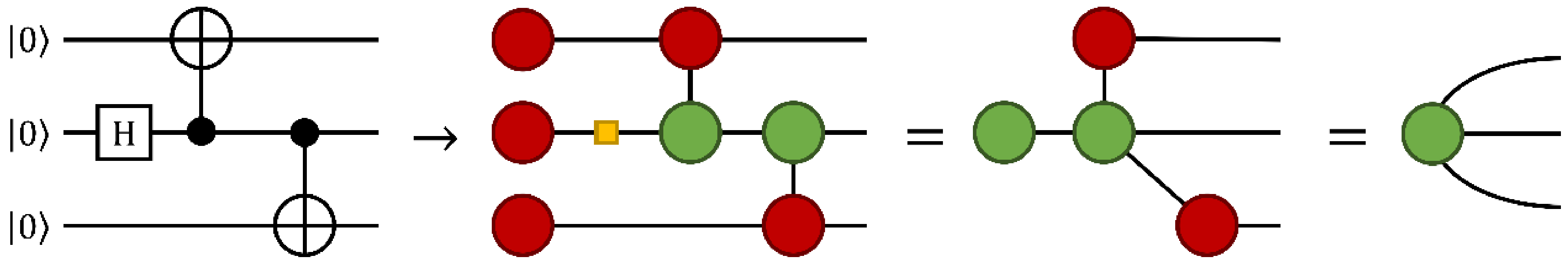
“Hadamard”



$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

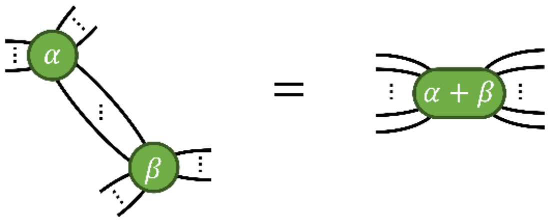


ZX-Calculus

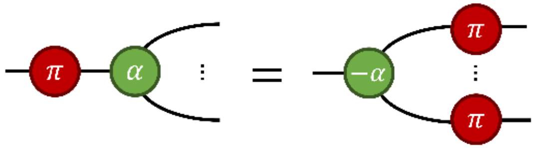


Rewrite Rules

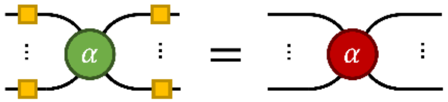
Spider fusion:



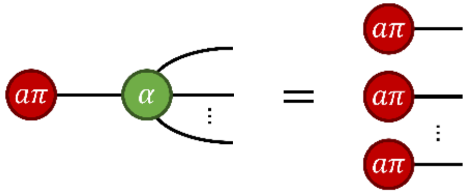
π − commutation:



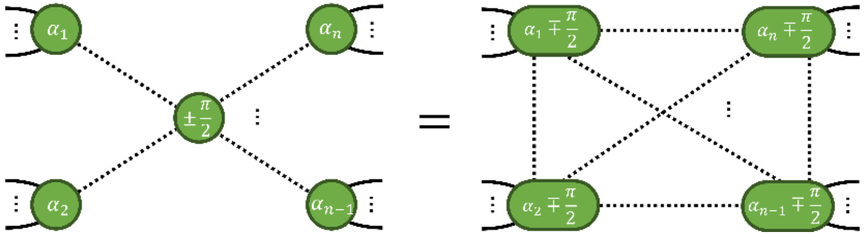
Colour change:



State copy:

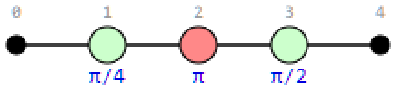


Local complementation:



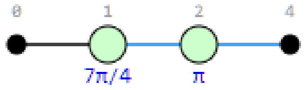
PyZX

```
zxGraph = genCirc()                          # Generate the example circuit (see appendix 1.B)
zx.draw(zxGraph, labels=True)                # Draw the circuit
```

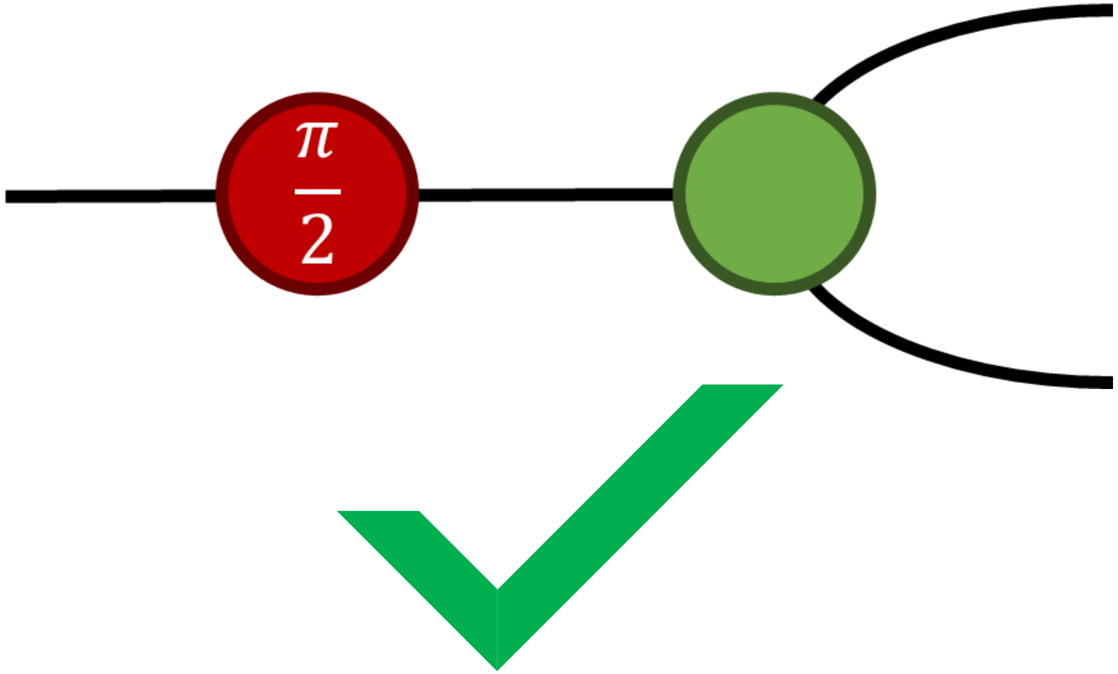


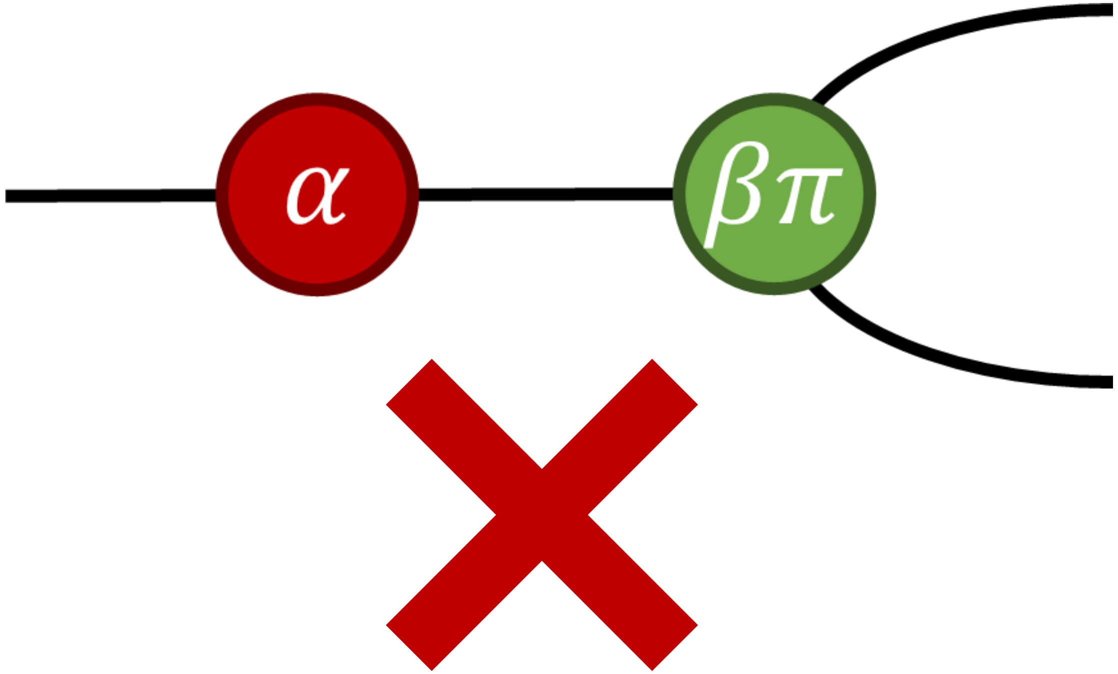

```
zx.simplify.full_reduce(zxGraph,False) # Fully simplify the circuit
zx.drawing.evenly_space(zxGraph)       # Reformat circuit with uniform spacing
zx.draw(zxGraph, labels=True)          # Draw the circuit
```

```
pivot_boundary_simp: 1.  1 iterations
lcomp_simp: 1. 1.  2 iterations
```



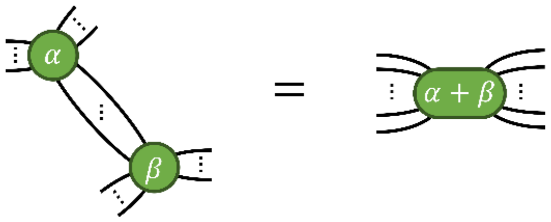
PyZX



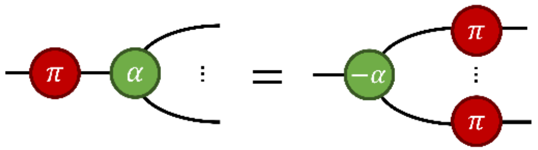


Rewrite Rules

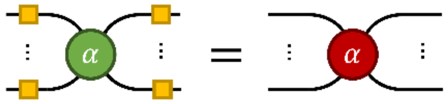
Spider fusion:



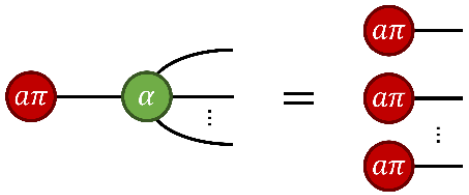
π − commutation:



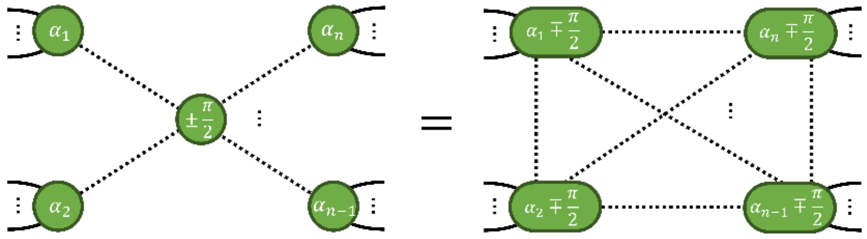
Colour change:



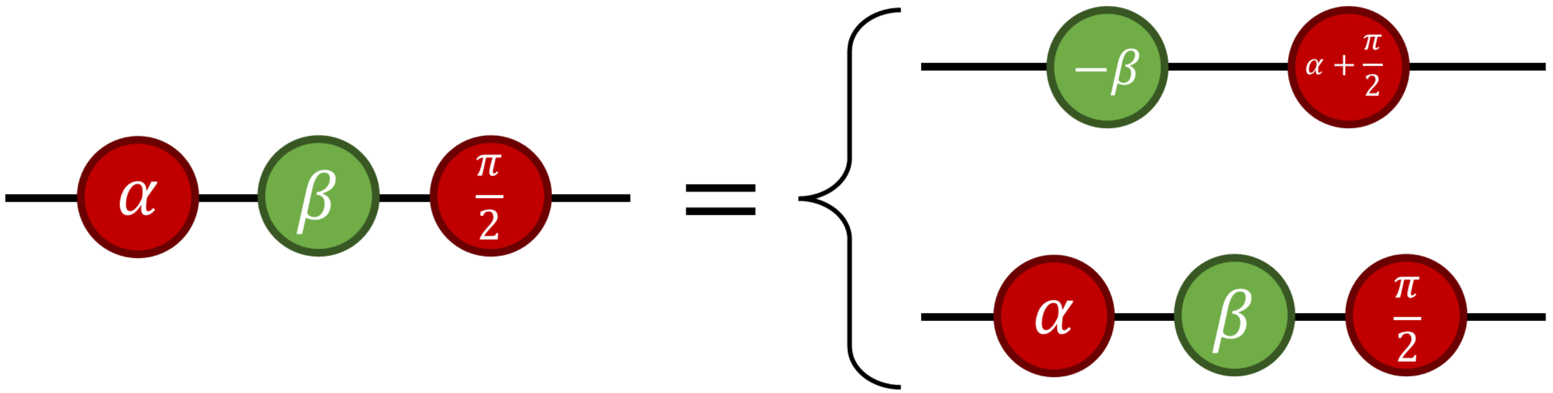
State copy:



Local complementation:

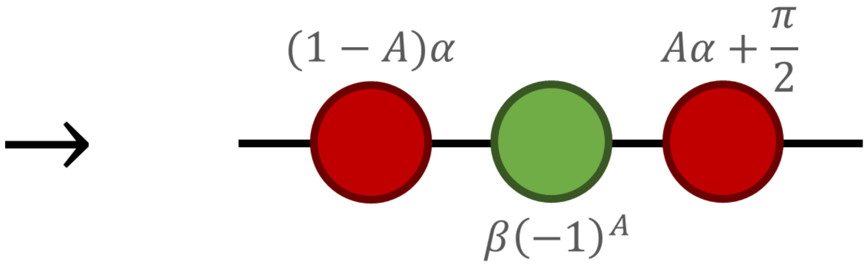


Parameterised Rewriting



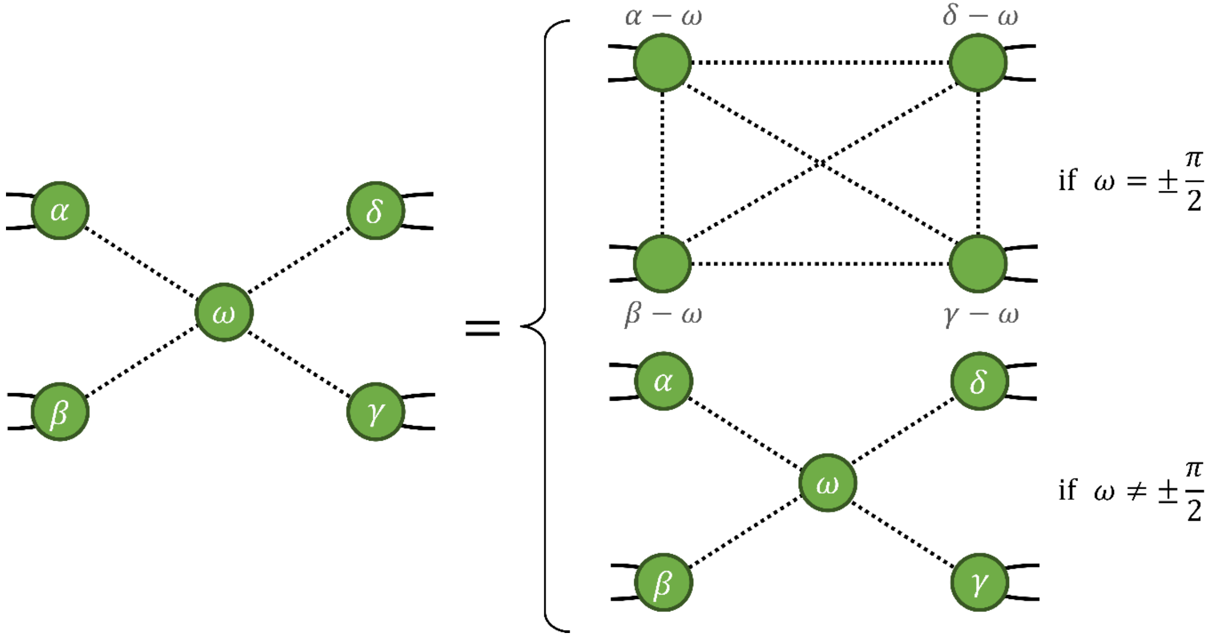
$$\text{if } \alpha = (2n+1)\pi$$

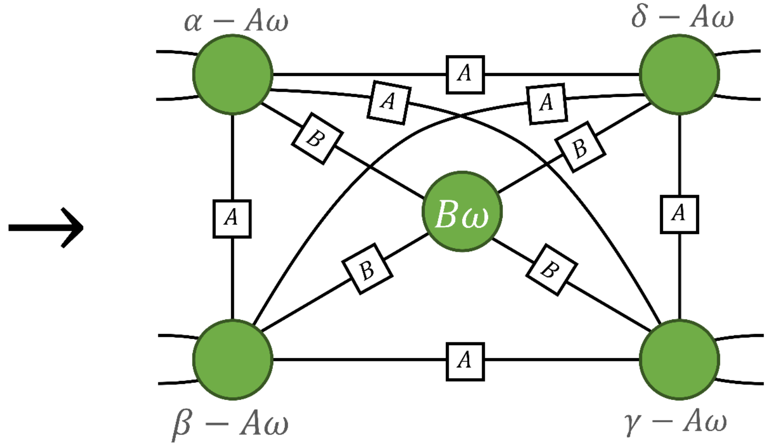
$$\text{if } \alpha \neq (2n+1)\pi$$



$$\text{where } A = 1 \text{ if } \alpha = (2n+1)\pi,$$
$$\text{and } A = 0 \text{ if } \alpha \neq (2n+1)\pi.$$

Parameterised Rewriting

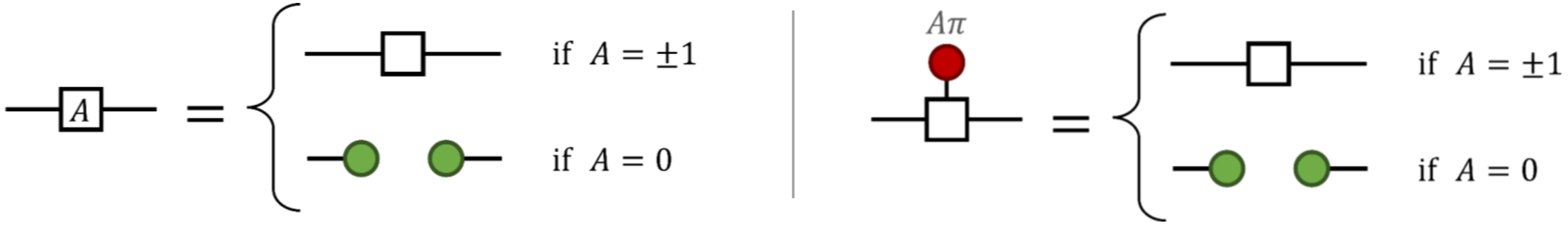




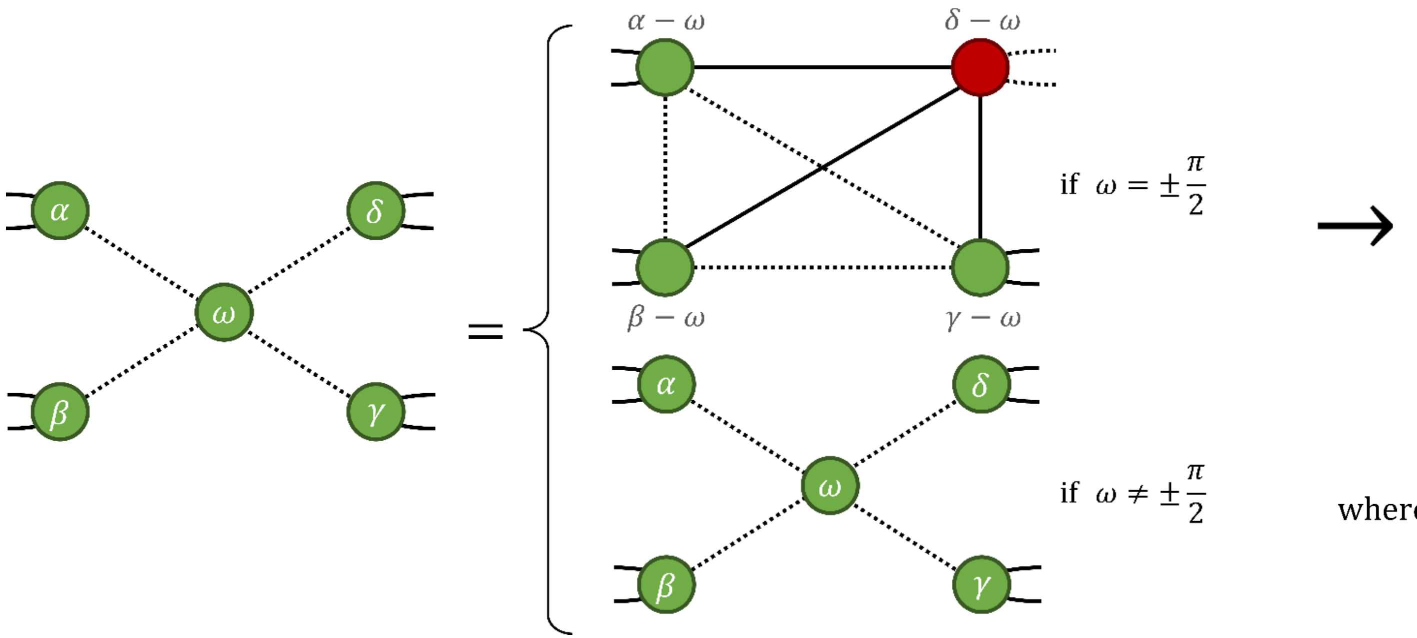
where $A = \pm 1$ if $\omega = \pm\frac{\pi}{2}$, and $A = 0$ if $\omega \neq \pm\frac{\pi}{2}$

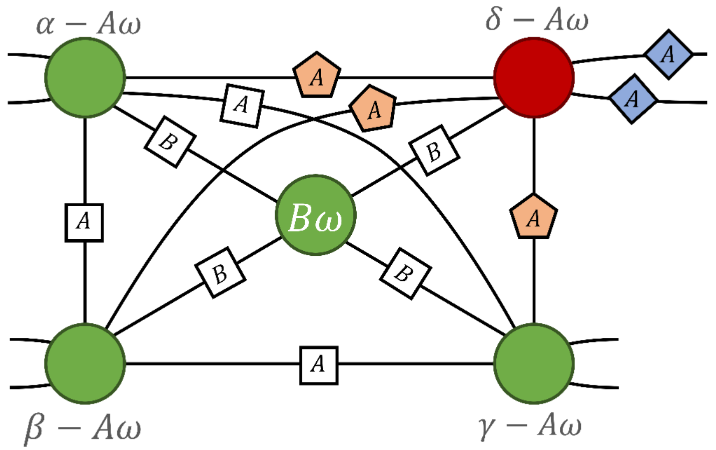
and $B = 1 - |A| = 0$ if $A = \pm 1$,

$= 1$ if $A = 0$.



Parameterised Rewriting

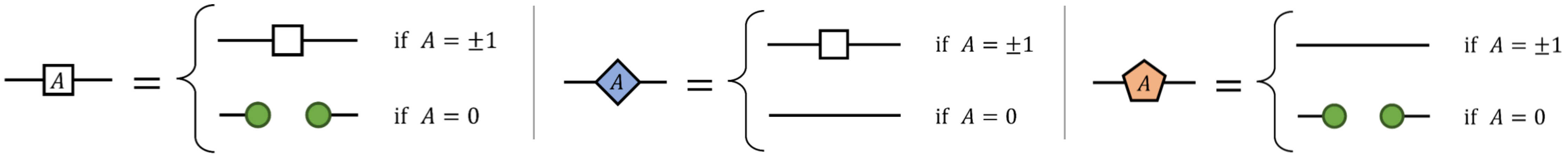




where $A = \pm 1$ if $\omega = \pm\frac{\pi}{2}$, and $A = 0$ if $\omega \neq \pm\frac{\pi}{2}$;

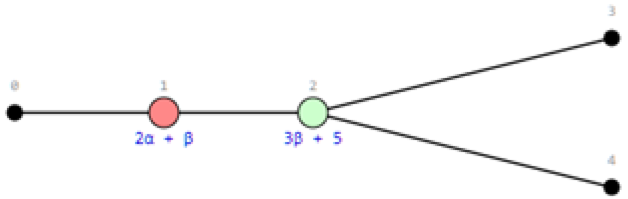
and $B = 1 - |A| = 0$ if $A = \pm 1$,

$= 1$ if $A = 0$.

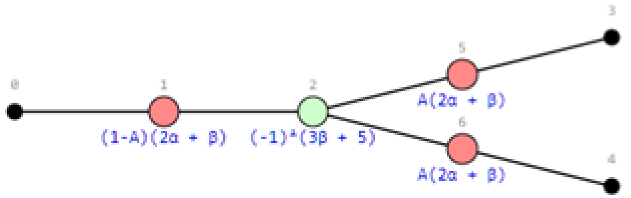


Parameterised Rewriting in PyZX

```
zxGraph = genCircSymbolic3()  # (adapted from genCircSymbolic)
zx.draw(zxGraph, labels=True) # Draw the circuit
```

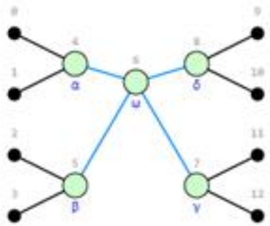



```
pi_commute_para(zxGraph, 2, 1) # apply parameterised pi_commutation rule to spiders #2 and #1
zx.draw(zxGraph, labels=True)  # Draw the circuit
zxGraph.print_cond_vars()      # Print the list of conditional/dependent (dummy) variables
```

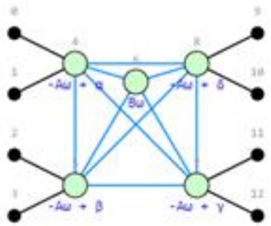


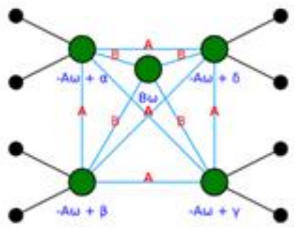

```
Where:
A = 1   if 2α + β = π
A = 1   if 2α + β = -π
A = 0   otherwise
```

```
zxGraph = genCircSymbolicLC() # (adapted from genCircSymbolic)
zx.draw(zxGraph, labels=True) # Draw the circuit
```



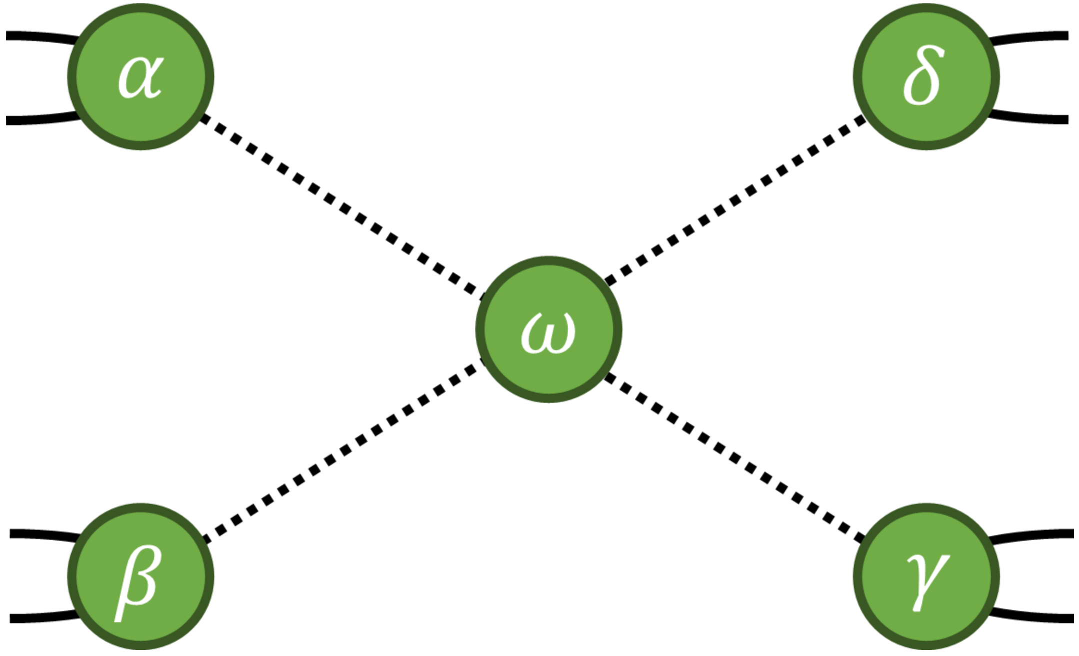

```
local_comp_para(zxGraph, 6)          # Apply parameterised local comp rule to spider #6
zx.draw(zxGraph, labels=True)        # Draw the graph using d3 [AS YET DOES NOT SUPPORT CONDITIONAL EDGE LABELS]
zxGraph.print_cond_vars()            # Print the list of conditional (dummy) variables
display(zx.draw_matplotlib(zxGraph)) # Draw the graph using matplotlib
```






```
Where:
A = 1   if ω = 0.5π
A = -1  if ω = -0.5π
A = 0   otherwise
and:
B = 0   if ω = 0.5π
B = 0   if ω = -0.5π
B = 1   otherwise
```

Parameter Sampling



$$\alpha, \beta, \gamma, \delta, \omega \in \left\{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\right\}$$

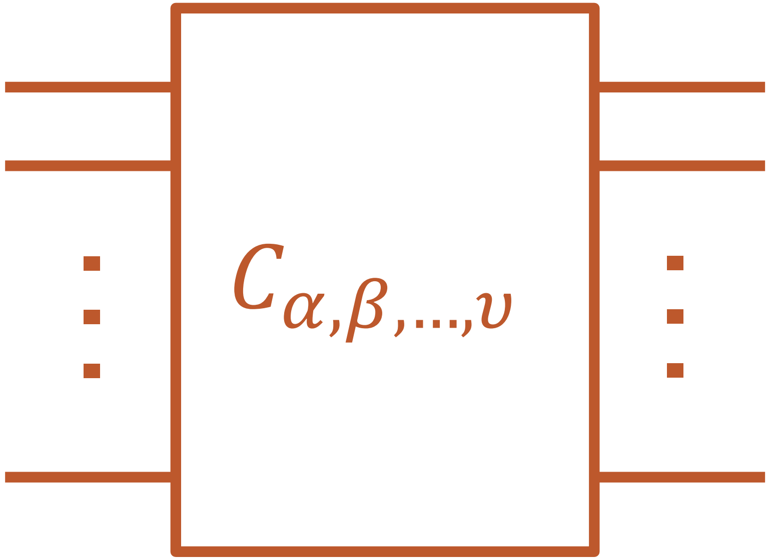
- p = no. of parameters
- x = no. of states per parameter

➔ **x^p simplifications**

e.g. $4^5 = 1024$ simplifications

~ 1 second

Parameter Sampling



$$\alpha, \beta, \ldots, \upsilon \in \left\{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\right\}$$

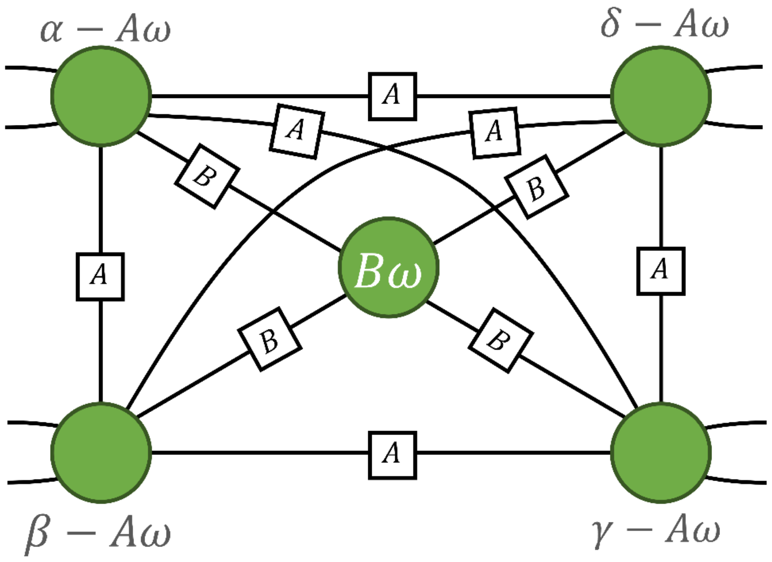
- p = no. of parameters
- x = no. of states per parameter

➔ **x^p simplifications**

e.g. $4^{20} = 1{,}099{,}511{,}627{,}776$ simps.

~ 34.84 years

Parameter Sampling



where $A = \pm 1$ if $\omega = \pm\frac{\pi}{2}$, and $A = 0$ if $\omega \neq \pm\frac{\pi}{2}$

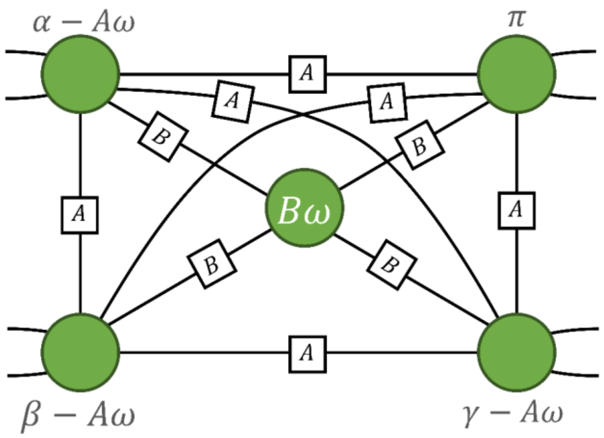
and $B = 1 - |A| = 0$ if $A = \pm 1$,
$= 1$ if $A = 0$.

- p = no. of parameters
- x = no. of states per parameter

➔ **1 "simplification"**

+ x^p ***evaluations***

Parameter Sampling



where $A = \pm 1$ if $\omega = \pm\frac{\pi}{2}$, and $A = 0$ if $\omega \neq \pm\frac{\pi}{2}$;

and $B = 1 - |A| = 0$ if $A = \pm 1$,
$= 1$ if $A = 0$.

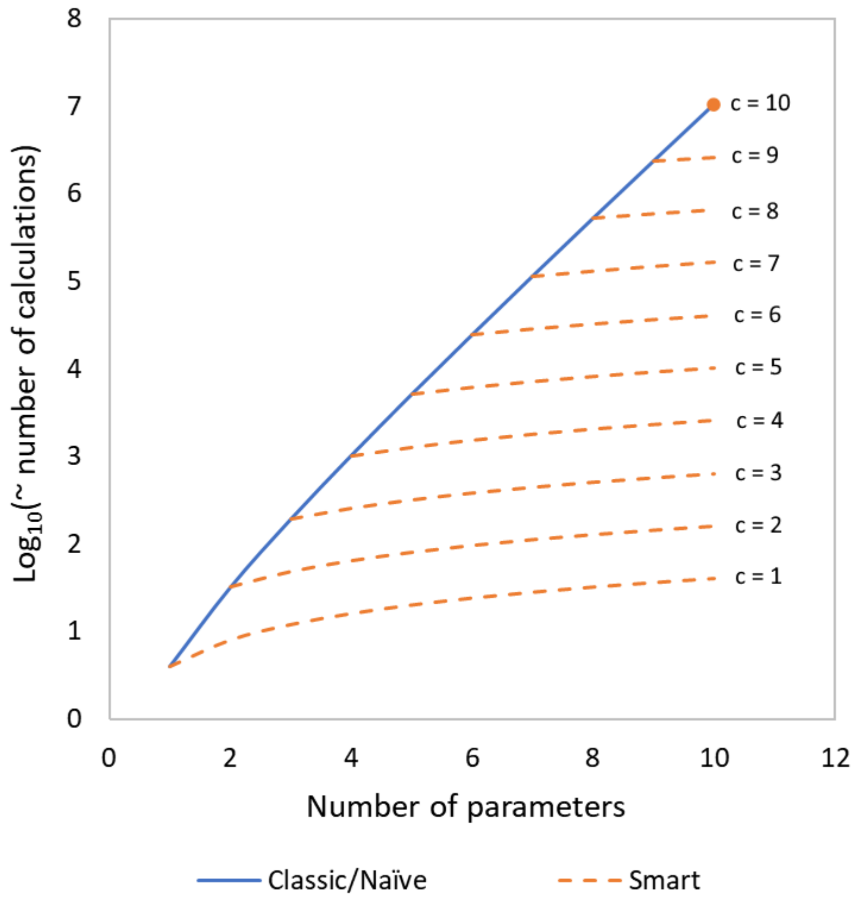
α	β	γ	ω	A	B	α-Aω	index	β-Aω	index	γ-Aω	index	Bω	index
0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	0	0.5	1	0	-0.5	1	-0.5	1	-0.5	1	0	1
0	0	0.5	0	0	1	0	0	0	0	0.5	2	0	0
0	0	0.5	0.5	1	0	-0.5	1	-0.5	1	0	3	0	1
0	0.5	0	0	0	1	0	0	0.5	2	0	0	0	0
0	0.5	0	0.5	1	0	-0.5	1	0	3	-0.5	1	0	1
0	0.5	0.5	0	0	1	0	0	0.5	2	0.5	2	0	0
0	0.5	0.5	0.5	1	0	-0.5	1	0	3	0	3	0	1
0.5	0	0	0	0	1	0.5	2	0	0	0	0	0	0
0.5	0	0	0.5	1	0	0	3	-0.5	1	-0.5	1	0	1
0.5	0	0.5	0	0	1	0.5	2	0	0	0.5	2	0	0
0.5	0	0.5	0.5	1	0	0	3	-0.5	1	0	3	0	1
0.5	0.5	0	0	0	1	0.5	2	0.5	2	0	0	0	0
0.5	0.5	0	0.5	1	0	0	3	0	3	-0.5	1	0	1
0.5	0.5	0.5	0	0	1	0.5	2	0.5	2	0.5	2	0	0
0.5	0.5	0.5	0.5	1	0	0	3	0	3	0	3	0	1

$$\alpha, \beta, \gamma, \omega \in \left\{0, \frac{\pi}{2}\right\}$$

No. of evaluations = ~~$2^4 \times 4 = 64$~~

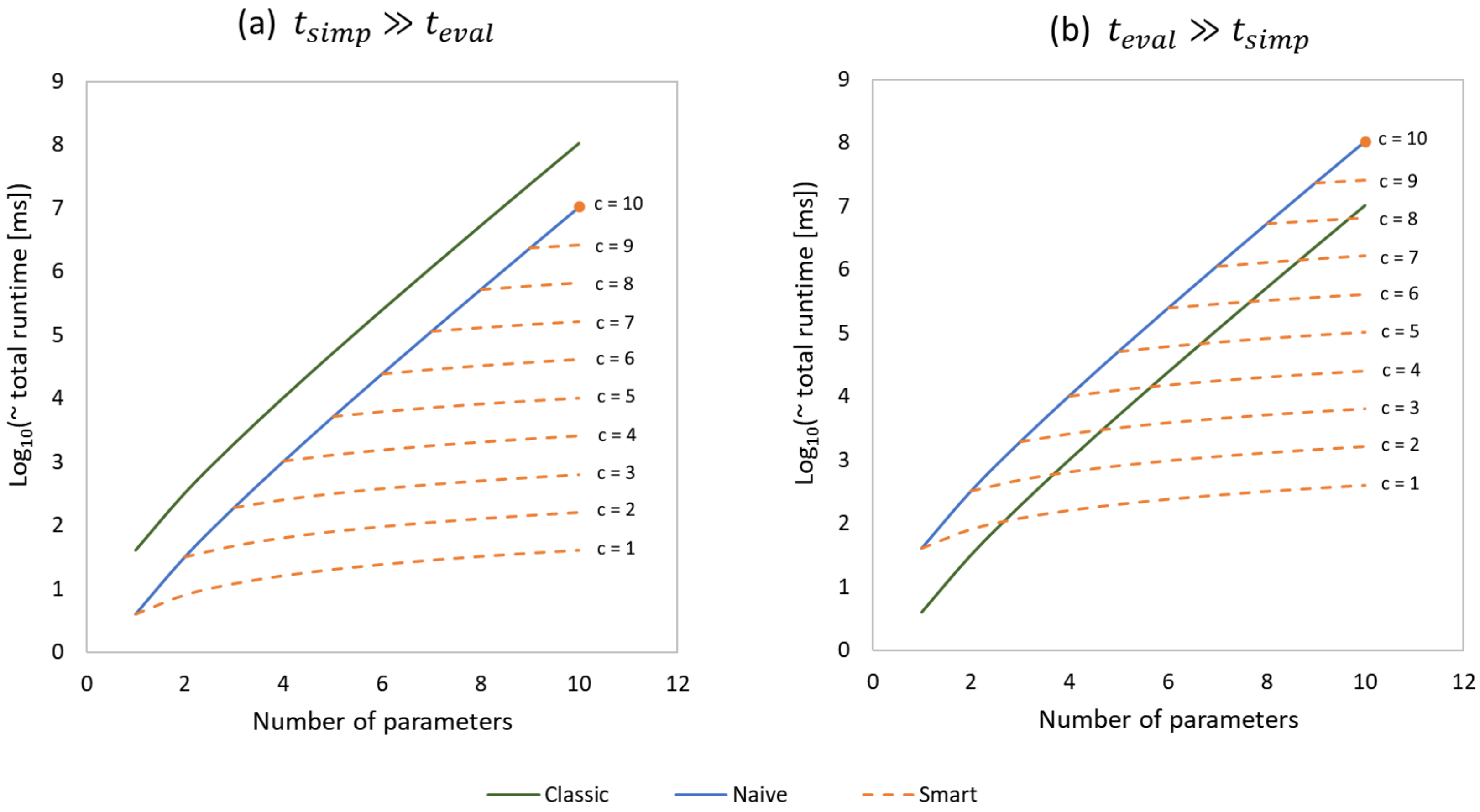
$$2^2 + 2^2 + 2^2 + 2^1 = 14$$

Parameter Sampling



- *c* = "complexity"
  = *max* no. of parameters **per spider**

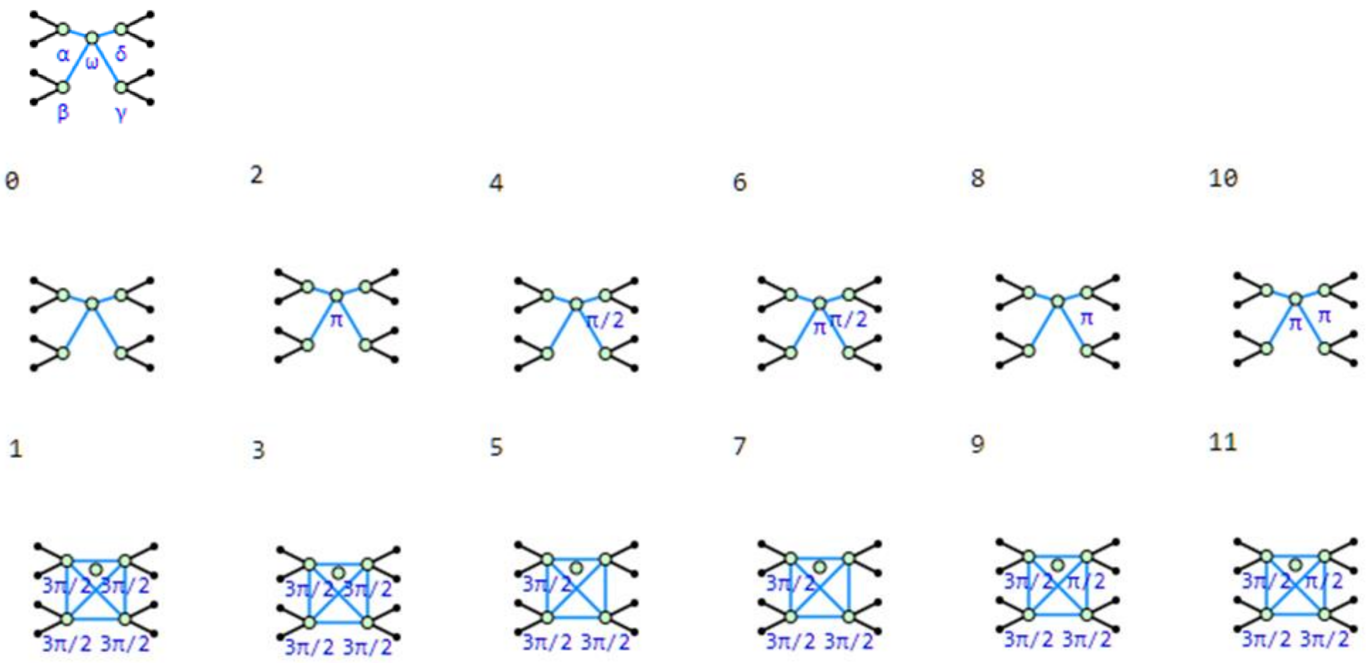
Parameter Sampling



Parameter Sampling – PyZX

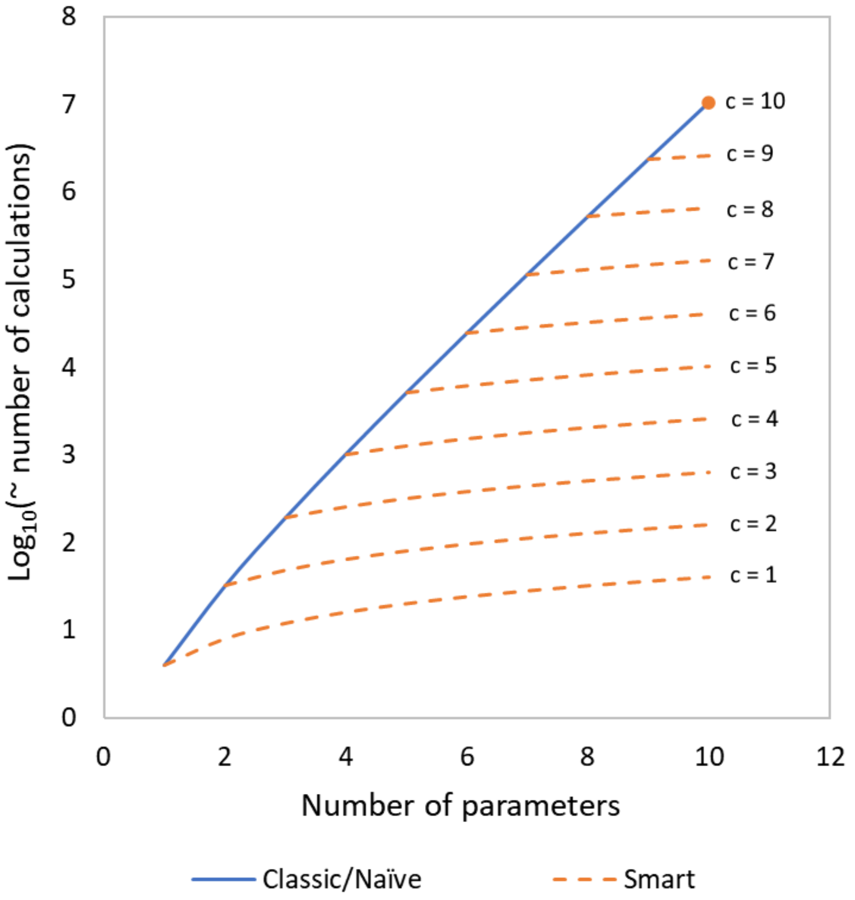
```
zxGraph = genCircSymbolicLC()
gList = zx.simplify.genParamSampledGraphs(zxGraph, 0.5, showOutput=True, scale=15)
```

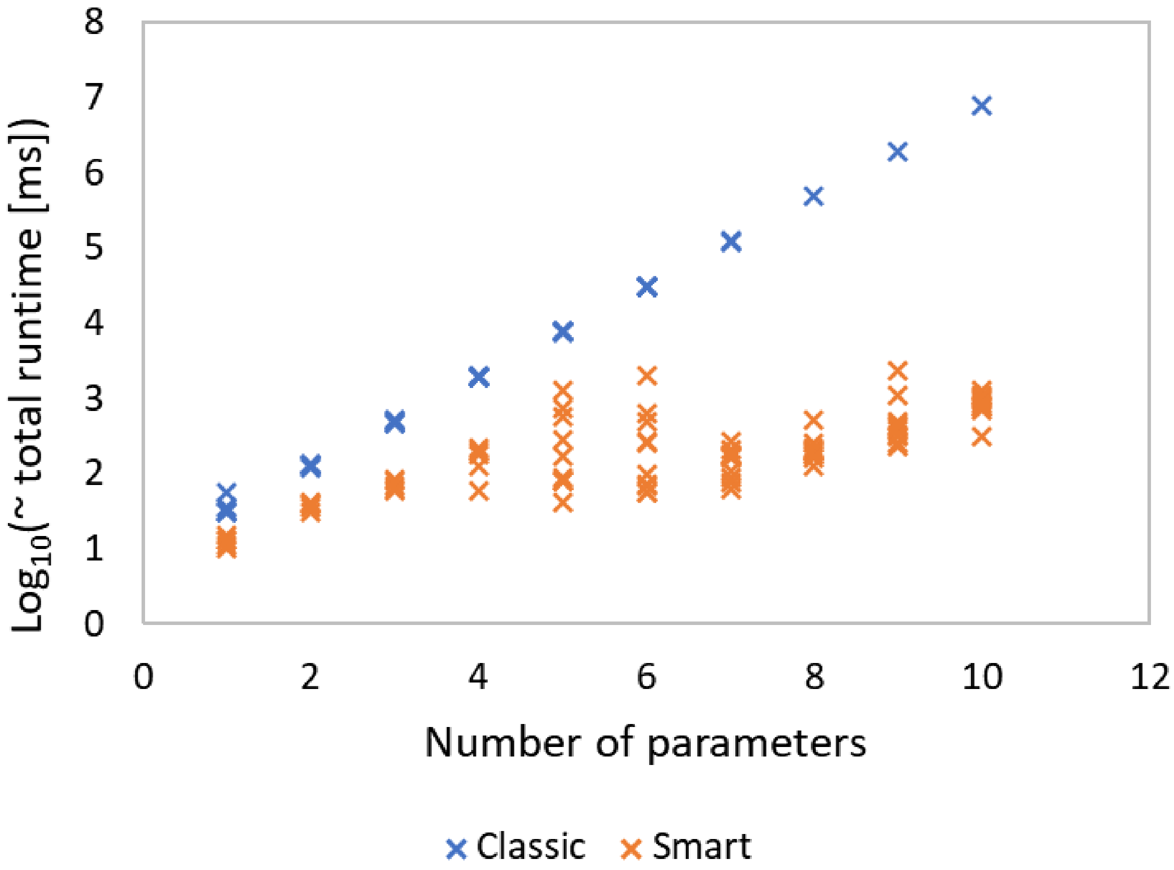
```
n_combos =  1024
```



...

Parameter Sampling – Result



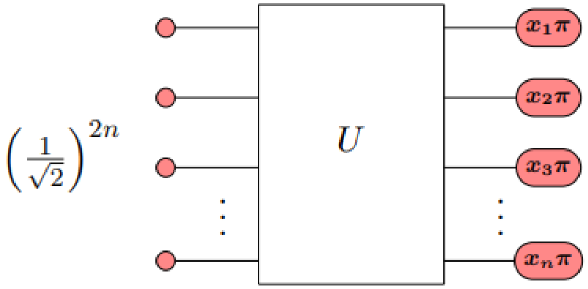


Theoretical

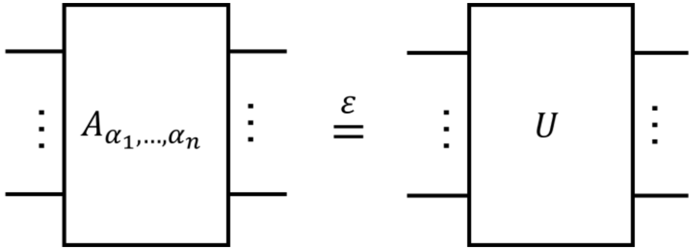
Experimental (Random circuits)

Applications?

- Classical simulation



- Ansatz fitting



Classical Simulation

Strong simulation

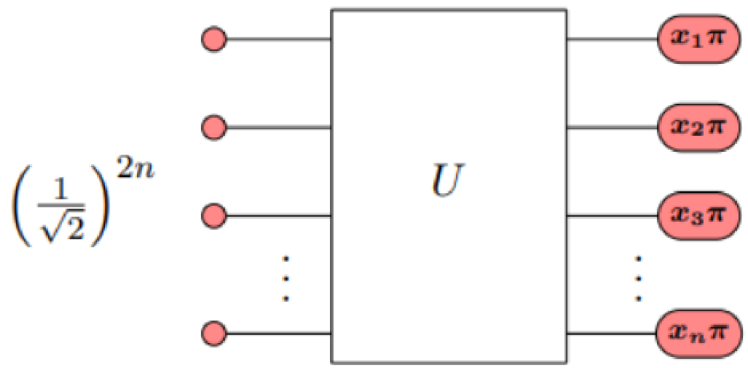
- Classically evaluate the probability of a given outcome, e.g. $P(0010)$.
- **Classically evaluating the probability distribution of all possible outcomes of a circuit: $\boldsymbol{P(x), \forall x}$.**

Weak simulation

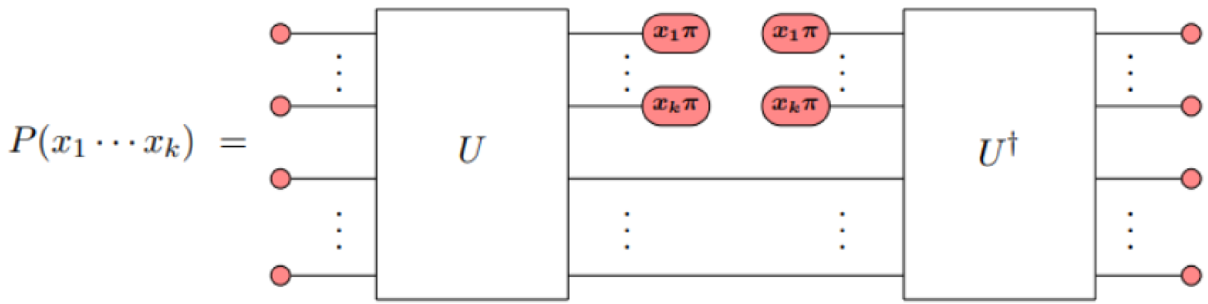
- From a known output distribution, classically simulate a circuit for a given input.

Classical Simulation: Stabiliser States Decomposition

Calculating a single amplitude:



Calculating marginal probability:



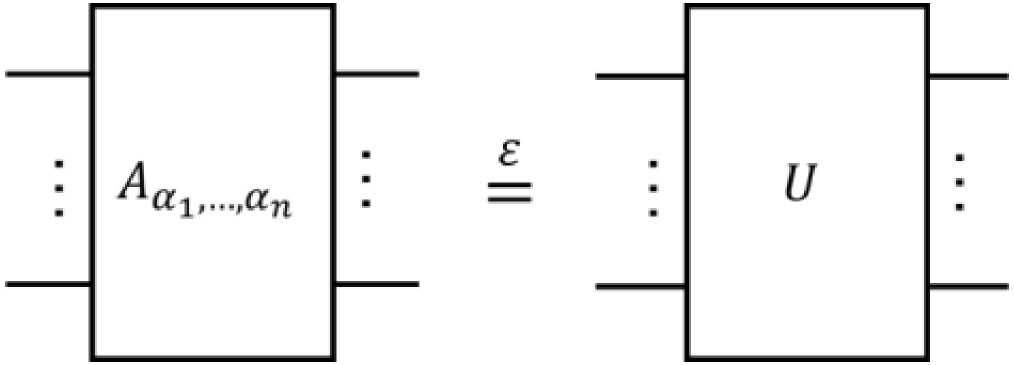
No. of calculations = $2^{\alpha t}$ $\therefore$ $t \rightarrow 2t$ = not good :(

NORM ESTIMATION → *Approximate* the marginal probabilities (to arbitrary precision)

'However, in preliminary experiments, we found this method to be prohibitively slow at obtaining enough samples to approximate the norm to high precision. Part of the problem here is that, ***for calculating many independent amplitudes, ZX-calculus simplification seems to be quite a bit slower*** *than a fast tableau simulator.'* – Kissinger and van de Wetering

Kissinger, A. and van de Wetering, J. Simulating quantum circuits with ZX-calculus reduced stabiliser decompositions. Quantum Science and Technology. 2022.

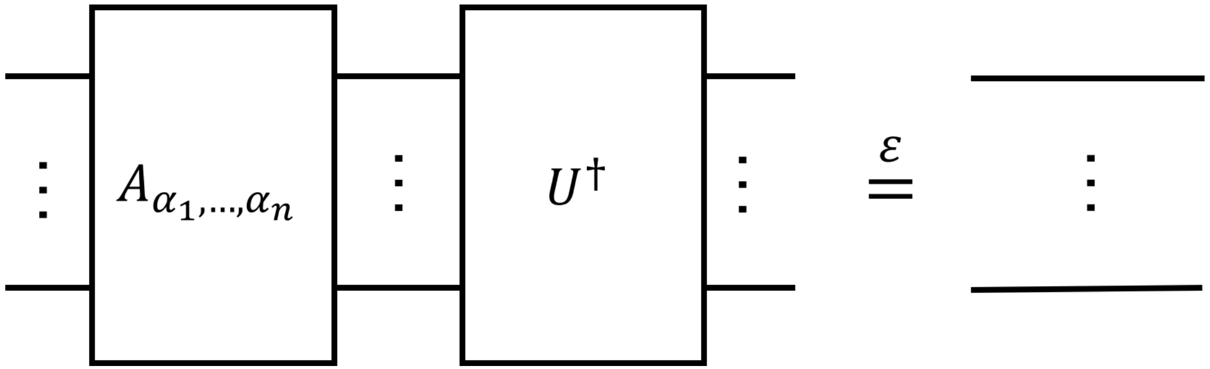
Ansatz Fitting



$$\alpha_1, \ldots, \alpha_n \in [0, 2\pi]$$

$$U \overset{\varepsilon}{=} V \quad \text{iff} \quad \max_{|\psi\rangle} \| U|\psi\rangle - V|\psi\rangle \| \leq \varepsilon$$

$U^\dagger U = I \quad \therefore \quad U^\dagger A_{\alpha_1,\ldots,\alpha_n} \overset{\varepsilon}{=} I,$ i.e....



Possible Further Improvements

- Optimising phase expression calculations
  - Identifying patterns/repetition
  - e.g. $\alpha + 2\beta$ and $\gamma + 2\delta$, where $\alpha, \beta, \gamma, \delta \in \left\{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\right\}$
- Computing in GPU
  - Parallel processing of phase evaluations