IV Quantum Computing and Quantum Algorithms

1. The circuit model

a) Classical computation

Use of classical computers (abstractly):

Solve problems $\equiv$ compute functions

$$f : \{0,1\}^n \longrightarrow \{0,1\}^m$$

$$\underline{x} = (x_1, \dots, x_n) \longmapsto f(x_1, \dots, x_n)$$

The function f depends on the problem we want to solve, $\underline{x}$ encodes the instance of the problem.

E.g.: Problem = multiplication : $(a,b) \longmapsto a \cdot b$

$$\underline{x} = (\underline{x}^1, \underline{x}^2) \longmapsto f(\underline{x}) = \underline{x}^1 \cdot \underline{x}^2$$

encoded in binary

Problem = Factorization:

$\underline{x}$: integer; $f(\underline{x})$: list of prime factors (suitably encoded)

More precisely:

Each problem is encoded by a family of functions $f \equiv f^{(n)} : \{0,1\}^n \to \{0,1\}^m$, with $m = \text{poly}(n)$, $n \in \mathbb{N}$ – one for each input size.

i.e.: m grows at most polynomially with n (technically, $\exists \alpha > 0$ s.th. $\frac{m}{n^\alpha} \to 0$).

(Technical point: It must be possible to "construct the functions $f^{(n)}$ systematically and efficiently", see later!)

Which ingredients do we need to compute a general function f?

(i) $f : \{0,1\}^n \longrightarrow \{0,1\}^m$

$$f(\underline{x}) = (f_1(\underline{x}), f_2(\underline{x}), \dots, f_m(\underline{x}))$$

where $f_k(\underline{x}) : \{0,1\}^n \longrightarrow \{0,1\}$

$\Rightarrow$ can restrict analysis to boolean functions

$$f : \{0,1\}^n \longrightarrow \{0,1\}.$$

(ii) Define $L = \{y \mid f(y) = 1\} = \{y^1, y^2, \dots, y^\ell\}$.

Define $\delta_{\underline{y}}(\underline{x}) = \begin{cases} 0 \,; & \underline{x} \neq \underline{y} \\ 1 \,; & \underline{x} = \underline{y} \end{cases}$ $\leftarrow$ Saturate equality!

Then, $f(\underline{x}) = \delta_{\underline{y}^1}(\underline{x}) \vee \delta_{\underline{y}_2}(\underline{x}) \vee \dots \vee \delta_{\underline{y}_\ell}(\underline{x})$

"$\vee$" : logical "<u>or</u>" :

$(0 \equiv$ "false",
$1 \equiv$ "true")

$0 \vee 0 = 0$
$0 \vee 1 = 1$
$1 \vee 0 = 1$
$1 \vee 1 = 1$

"$\vee$" is <u>associative</u>:

$$a \vee b \vee c := (a \vee b) \vee c = a \vee (b \vee c)$$

and <u>commutative</u>: $a \vee b = b \vee a$,

(iii) Define bitwise δ:

$$\delta_y(x) = \begin{cases} 0 : & y \neq x \\ 1 : & y = x \end{cases}$$

Then,

$$\underline{\delta}_{\underline{y}}(\underline{x}) = \delta_{y_1}(x_1) \wedge \delta_{y_2}(x_2) \wedge \dots \wedge \delta_{y_n}(x_n)$$

"$\wedge$": logical "and":

(0 ≡ "false",
1 ≡ "true")

$0 \wedge 0 = 0$
$0 \wedge 1 = 0$
$1 \wedge 0 = 0$
$1 \wedge 1 = 1$

"$\wedge$" is associative & commutative;

"$\wedge$" & "$\vee$" are distributive:

$$(a \vee b) \wedge c = (a \wedge c) \vee (b \wedge c).$$

(In essence, same rules as $\wedge \to \cdot$, $\vee \to +$)

(iv)

$$\delta_y(x) = \begin{cases} x & \text{if } y = 1 \\ \neg x & \text{if } y = 0 \end{cases}$$

Logical "not": $\neg 0 = 1$

$\neg 1 = 0$

Combine (i) – (iv):

Any $f(\underline{x})$ can be constructed from 4 ingredients:

"and", "or", "not" gates,

plus a "copy" gate $x \longmapsto (x, x)$.

This is called a universal gate set.

(Note: In fact, already either $\neg(x \wedge y)$ "nand", or $\neg(x \vee y)$ "nor" are universal, together with "copy".)

This gives rise to the

Circuit model of computation:

The functions $f \equiv f^{(n)}$ which we can compute are constructed by concatenating gates from a simple universal gate set (e.g. and/or/not/copy) sequentially in time (i.e., there are no loops allowed). This gives rise to a circuit for $f^{(n)}$.

The difficulty ("computational hardness") of a problem in the circuit model is measured by the number $K(n)$ of elementary gates needed to compute $f^{(n)}$ ($\hat{=}$ # of time steps).

We often distinguish to qualitatively different regimes:

$K(n) \sim \mathrm{poly}(n)$: efficiently solvable (class P)
easy problem

$K(n) \gg \mathrm{poly}(n)$ — e.g. $K(n) \sim \exp(n^{\alpha})$:

hard problem

(Technical note: We must suppose that the circuits used for $f^{(n)}$ are <u>uniform</u>, i.e. they can be generated efficiently – e.g. by a simple n-independent computer program. More formally, $f^{(n)}$ should be generated by a Turing machine.)

Example:

f = Multiplication:

Efficient:

$$
\begin{array}{r}
\overbrace{10110}^{\ell} \times \overbrace{10011}^{\ell'} \\
\hline
10110 \\
10110 \\
10110 \\
\hline
\underbrace{110100010}
\end{array}
\left.\vphantom{\begin{array}{c}1\\1\\1\end{array}}\right\} \ell'
$$

$\ell \times \ell'$ additions:

$O(\ell\ell') \sim O(n^2)$ gates.

f: Factorization.

E.g.: Sieve of Erathostenes:

$\{0,1\}^n \rightarrow$ try about $\sqrt{2^n} \sim 2^{n/2}$ cases

$\Longrightarrow$ hard / exp. scaling.

No efficient algorithm known!

Is a typical problem easy or hard?

$$f : \{0,1\}^n \longrightarrow \{0,1\}$$

\# of different f: $\quad 2^{\underbrace{(2^n)}_{\#\text{ of inputs}}}$

$f(\underline{x}) = \begin{cases} 0 \\ 1 \end{cases}$ for each input

But: there are only $c^{\text{poly}(n)}$ circuits of length poly(n)!

\# of elem. gates

$\Rightarrow$ As n gets large, most f cannot be computed efficiently (i.e. with poly(n) operations).

Does the computational power depend on the gate set?

NO! By definition, any universal gate set can simulate any other gate set with constant overhead!

Remark: There is a wide range of alternative models of computation, some more and some less realistic:

- CPU
- parallel computers
- "Turing machines" – tape + read/write head
- cellular automata
- ... and lots of exotic models ...

But: All known "reasonable" models of computation can simulate each other with poly(n) overhead

$\Longrightarrow$ same computational power (in the sense above).

Church-Turing-Thesis: All reasonable models of computation have the same computational power.

b) Reversible circuits

For quantum computing – coming soon – we will use the circuit model.

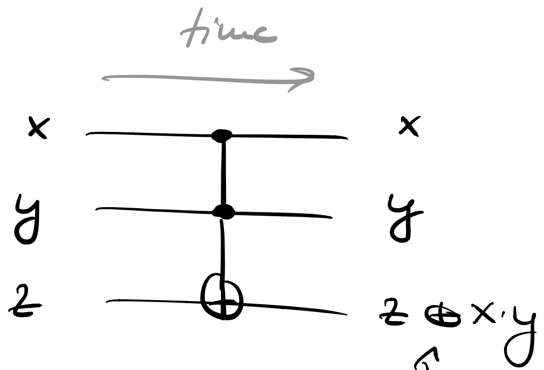
Gates will be replaced by unitaries.

But: Unitaries are reversible, while classical gates (and, or) are irreversible.

Could such a model even do classical computations – i.e., can we find a universal gate set with only reversible gates?

YES! – Classical computation can be made reversible:

Toffoli gate:



"XOR" = addition mod 2:

$0 \oplus 0 = 0$

$0 \oplus 1 = 1$

$1 \oplus 0 = 1$

$1 \oplus 1 = 0$

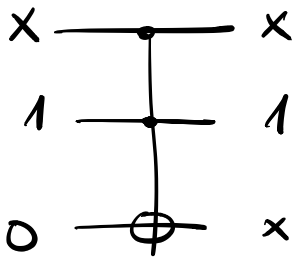
also assoc., commu., & distr. w/ $\wedge$. (like "$\vee$")

→ Toffoli gate is reversible

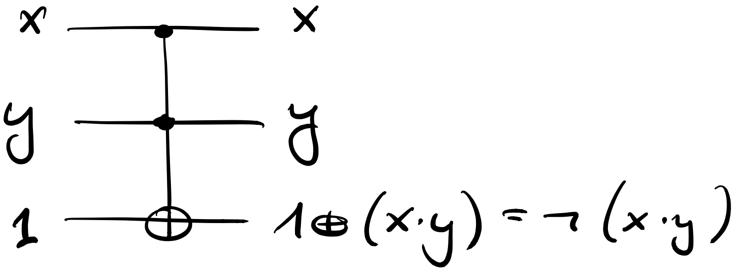
(it is its own inverse, since $(z \oplus x \cdot y) \oplus x \cdot y = z$)

→ Toffoli gate can simulate and/or/not/copy, by using ancillas in state "0" or "1":

<u>E.g.</u>:



<u>"copy"</u>



<u>"Nand"</u>

⟹ gives reversible universal gate set (but requires ancillas)

This can be used to <u>compute any $f(\underline{x})$ reversibly</u>, using ancillas, with essentially the same # of gates:

$$f^R(\underline{x}, y) \longmapsto (\underline{x}, f(\underline{x}) \oplus y)$$

bitwise XOR.

(<u>Idea</u>: Replace any gate by a reversible gate using ancillas. Then XOR the result into the y register. Finally, run the circuit backwards to "uncompute" the ancillas. Ancilla count can be optimized for → cf. Preskill's notes.)

⟹ Everything can be <u>computed reversibly</u>.

<u>BUT</u>: <u>3-bit</u> gate is <u>required</u>!

(→ Homework)

c) Quantum Circuits

Most common model for quantum computation:

<u>The circuit model:</u>

- Quantum system consisting of qudits: tensor product structure.
- <u>Universal gate set</u> $S = \{U_1, \dots, U_k\}$ of few-qudit gates (typ. 1- and 2-qudit gates) U_j. (See later for definition of "universal"!)
- Construct circuits by sequentially applying

elements of S to a subset of qubits:

$$|\psi_{out}\rangle = V_T V_{T-1} \cdots V_1 |\psi_{in}\rangle$$

U_j acting on subset of qubits

- Initial state:

$$|\psi_{in}\rangle = |x_1\rangle|x_2\rangle \dots |x_n\rangle \overbrace{|0\rangle|0\rangle \dots |0\rangle}^{\ell}$$

$$= |\underline{x}\rangle |\underline{0}\rangle$$

ancillas

encodes instance of problem

– alternatively, we can also have

$$|\psi_{in}\rangle = |\underline{0}\rangle \equiv |0\rangle^{\otimes \ell}$$

and encode the instance in the circuit.

- At the end of the computation, measure the final state $|\psi_{out}\rangle$ in the computational basis $\{|0\rangle, |1\rangle\}$

$\longrightarrow$ outcome $|y\rangle$ w/ prob. $p(y) = |\langle y|\psi_{out}\rangle|^2$

Notes: • This is a <u>probabilistic</u> scheme — it outputs y w/ some prob. $p(y)$. In principle, we should compare to <u>class. probabilistic</u> schemes — see later.

• We need not measure all qubits —

not measuring = tracing = measuring and ignoring outcome

◦ POVMs don't help — we can simulate them (→ Naimark). Similarly, CP maps don't help — we can simulate them (Stinespring + trace ancilla).

◦ Measurements at earlier times don't help: Can always postpone them (they commute). If gate at later time would depend on meas. outcome: This dependence can be realized <u>inside</u> the circuit w/ "controlled gates"

(cf. later + homework)

What gate set should we choose?

- There is a <u>continuum</u> of gates — situation much more rich.

- <u>Different notions of universality</u> exist:

  - <u>exact universality</u>: Any n-qubit gate can be realised <u>exactly</u>.

    → Requires a <u>continuous family</u> of universal gates (counting argument!)

  - <u>approximate universality</u>: Any n-qubit gate can be approximated well by gate set

    (Finite gate set sufficient;

    <u>Solovay-Kitaev-Theorem</u>: ε-approximation (in $\|\cdot\|_\infty$-Norm) of 1-qubit gate requires

    $O(\mathrm{poly}(\log(1/\varepsilon)))$ gates from a suitable finite set.)

- 1- and -2 qubit gates alone are universal! (cf. classical: 3-bit gates needed!!)

- For approximate universality, almost any single two-qubit gate will do! w/ prob. 1.

- More univ. sets: later!

d) Universal gate set

Our exact universal gate set:

(i) 1-qubit rotations about X & Z axis:

$$R_X(\phi) = e^{-iX\phi/2} \quad ; \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad X^2 = I$$

$$R_Z(\phi) = e^{-iZ\phi/2} \quad ; \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad Z^2 = I.$$

For $M^2 = I$: $\quad e^{-iM\phi/2} = \cos\phi/2 \; I - i\sin\phi/2 \; M$

$$\Longrightarrow \quad R_X(\phi) = \begin{pmatrix} \cos\phi/2 & -i\sin\phi/2 \\ -i\sin\phi/2 & \cos\phi/2 \end{pmatrix}$$

$$R_Z(\phi) = \begin{pmatrix} e^{-i\phi/2} & 0 \\ 0 & e^{i\phi/2} \end{pmatrix}$$

Can be understood as rotations on Bloch sphere about X/Z axis by angle ϕ (i.e., rotations in $SO(3) \cong SU(2)/\mathbb{Z}_2$).

Together, R_x and R_z generate all rotations in $SO(3)$ (Euler angles!), and thus in $SU(2)$ up to a phase.

<u>Lemma:</u> For any $U \in SU(2)$,

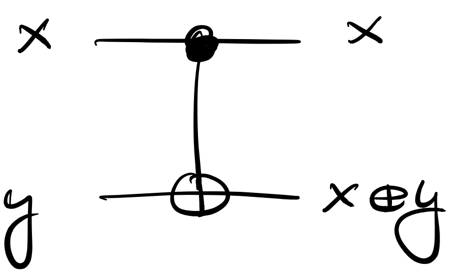
$$U = e^{i\phi} R_x(\alpha) R_z(\beta) R_x(\gamma) \quad \text{for some } \phi, \alpha, \beta, \gamma.$$

<u>Proof:</u> <u>Homework.</u>

(ii) <u>one</u> two qubit gate (almost all would do!).

Typically, we use "controlled-NOT" = "CNOT":

CNOT =



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

CNOT flips y iff $x = 1$: <u>classical</u> gate!

<u>Can prove</u>: This gate set can create any n-qubit U <u>exactly</u> (but of course not efficiently – U has $\sim (2^n)^2 = 4^n$ real parameters).

Overview of a number of important gates & identities

(Proof/check: Homework!)

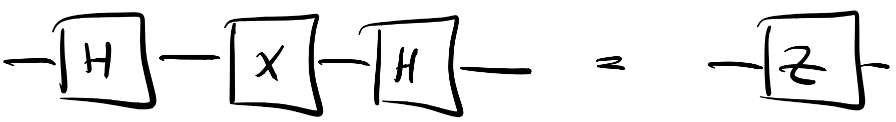
<u>Hadamard gate</u>: $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$

$$H = H^\dagger ; \quad H^2 = I .$$

$$H R_x(\phi) H = R_z(\phi)$$

$$H R_z(\phi) H = R_x(\phi)$$

<u>Graphical "circuit" notation</u>:

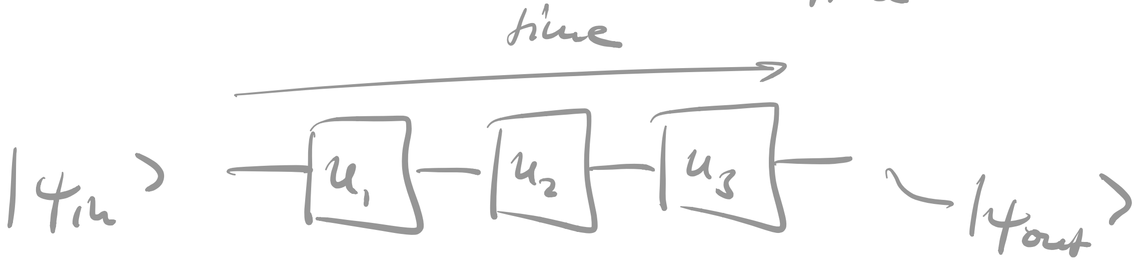


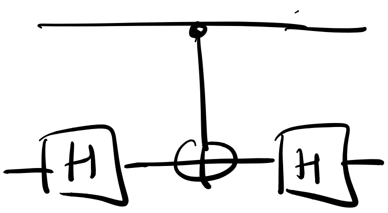
<u>Important</u>:

Matrix notation: time goes right to left

Circuit notation: time goes left to right:

i.e.: $|\psi_{in}\rangle \longmapsto |\psi_{out}\rangle = \underbrace{U_3 U_2 U_1}_{\xleftarrow{\text{time}}} |\psi_{in}\rangle$

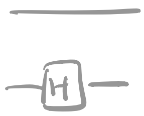




$=$

$= \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & -1 \end{pmatrix}$

only applied to 2nd qubit, i.e.:

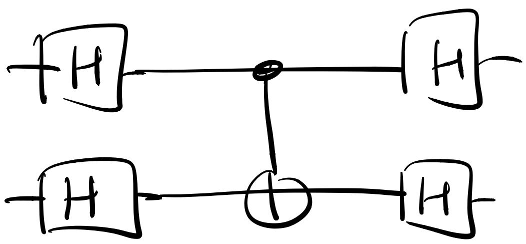


$\equiv I \otimes H.$

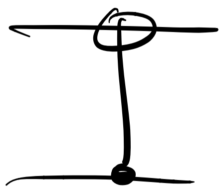
"controlled-Z"

"controlled-Phase"

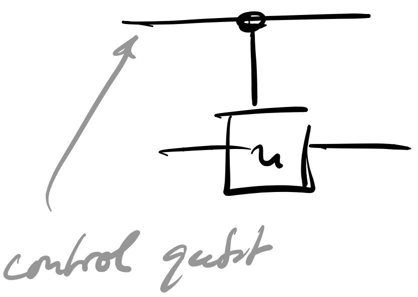
CZ, CPHASE



$=$



Generally: For a unitary $U \in SU(2)$,

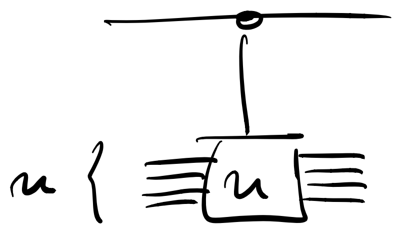


$$= \left(\begin{array}{c|c} I & 0 \\ \hline 0 & U \end{array}\right) \quad \text{"controlled-}U\text{"}$$

$$\underbrace{\qquad}_{\substack{\text{control}\\ =|0\rangle}} \qquad \underbrace{\qquad}_{\substack{\text{control}\\ =|1\rangle}}$$

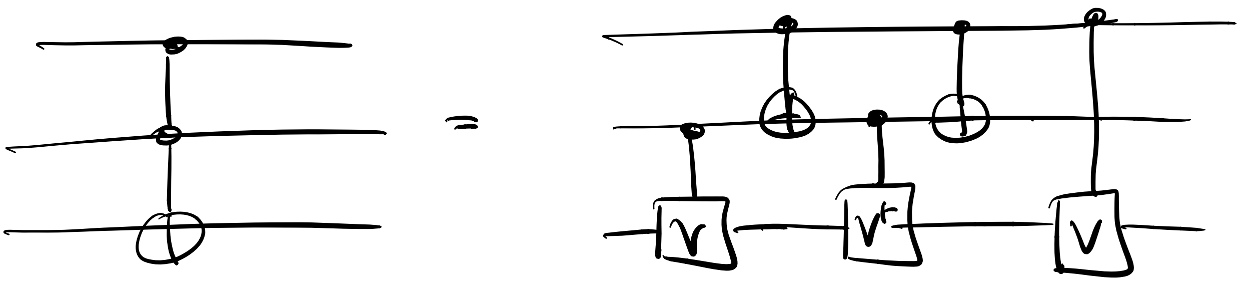
Can be implemented w/ 2 CNOT ($\to$ HW!)

Also for $U \in SU(2^n)$:



$$= \left(\begin{array}{c|c} I_{2^n} & 0 \\ \hline 0 & U \end{array}\right)$$

<u>Circuit for Toffoli:</u>

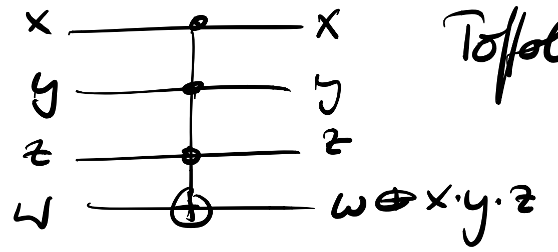


with $V = \frac{1-i}{2}(I + iX)$

<u>U to controlled-U:</u>

Given circuit for U – in particular, a classical reversible circuit – we can also build controlled-U:

Just replace every gate by its controlled-version, in particular Toffoli by



Toffoli w/ 3 controls can be built from normal Toffoli (since class. universal!)

Finally, some further approx. universal gate sets:

- CNOT + 2 random 1-qubit gates
- CNOT + H + T = $R_z(\pi/4)$ ("$\pi/8$ gate")