Introduction to Quantum Algorithms and Quantum Programming

Benoît Valiron

Course Notes v.2024.09.10

Contents

1 Mathematical Background

1.1 Notations

1.1.1 The set of natural numbers is $\mathbb{N}$. Addition is written "+" and multiplication "·". When clear, we simply use concatenation for the multiplication. For instance, $2n$ is $2 \cdot n$.

1.1.2 The factorial $n!$ is $1 \cdot 2 \cdot 3 \cdot 4 \cdot \ldots \cdot n$. By convention $0! = 1$.

1.1.3 We assume known the notion of *real number*. The set of reals is denoted with $\mathbb{R}$. For the record, we recall that real numbers form a *field*: the set $\mathbb{R}$ is equipped with

- An addition "+" and a null element "0", also called *zero*.
- A multiplication "$\cdot$" and a neutral element "1"
- An inverse operation written $(-)^{-1}$, or $\frac{1}{-}$, defined on all non-zero real number.

1.1.4 Real intervals are written $[a,b]$ when both a and b are part of the interval, and $[a,b)$, $(a,b]$ or (a,b) when the right, or the left, or both ends of the interval are not included. Following the standard notation, we use the placeholders $-\infty$ and ∞ to represent unbounded intervals on the left or on the right.

1.1.5 We use the representation $\sqrt{x}$ for the square root of x, and x^a for x to the power a. The exponent a can be negative: in this case, we mean the power of the inverse of x. For instance, $x^{-2} = \frac{1}{x^2}$.

1.1.6 The 2-elements Boolean algebra is written $\mathbb{B}$. Its objects are 0 and 1, standing respectively for "False" and "True". The "and" operation (the conjunction) is denoted with $\wedge$, or simply as a product "$\cdot$", or by juxtaposition: $x \wedge y = x \cdot y = xy$. The "or" (the disjunction) is written $\vee$. The "not" (the negation) is written $\neg$. The exclusive or (also known as XOR) is written with $\oplus$.

1.2 Sums and series

Along this document we shall be using sums over finite and infinite sets of indices. In this section, we summarize what should be known.

1.2.1 Sum symbol. Given a function $f : \mathbb{N} \to \mathbb{R}$, and given $i \leq j \in \mathbb{N}$, we define

$$\sum_{n=i}^{j} f(n) \quad \triangleq \quad f(i) + f(i+1) + \cdots + f(j).$$

Provided that the limit is well-defined, we define

$$\sum_{n=i}^{\infty} f(n) \quad \triangleq \quad f(i) + f(i+1) + \cdots \quad \triangleq \quad \lim_{j\to\infty} \sum_{n=i}^{j} f(n)$$

Such a limit is called a *series*. If the limit is defined, we say that the series is *converging*. The series is *absolutely converging* if

$$\sum_{n=i}^{\infty} |f(n)|$$

is converging.

$$\sum_{n=0}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6} \tag{1}$$

$$\sum_{n=0}^{\infty} \frac{1}{n!} x^n = e^x \tag{2}$$

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} = \cos(x) \tag{3}$$

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = \sin(x) \tag{4}$$

$$\sum_{i=0}^{n} a^i = \frac{1-a^{n+1}}{1-a} \tag{5}$$

$$\sum_{i=0}^{\infty} a^i = \frac{1}{1-a} \tag{6}$$

Table 1: Values for a few known series. Eq. (1) is called the *Basel problem*. In Eqs. (2), (4) and (3), x is any real number. In Eq. (5), $a \neq 1$, while Eq. (6) holds whenever $0 \leq a < 1$.

1.2.2 Lists of known series. Table 1 summarizes the values of few known series. Series of the form of Eq. (6) are called *geometric series*. Maybe the most important one for this course is Eq. (2) that we recall here:

$$\sum_{n=0}^{\infty} \frac{1}{n!} x^n = e^x.$$

1.3 Complex Numbers

We write $\mathbb{C}$ for the field of *complex numbers*. A complex number is of the form $\alpha = a + b \cdot i$ with a and b reals and i the *imaginary* number: a number such that $i^2 = -1$.

1.3.1 Conjugate. The *conjugate* of a complex number $\alpha = a + b \cdot i$ is defined as $\overline{\alpha} = a - b \cdot i$. The *absolute value*, or the *norm* of α is the non-negative, real number

$$|\alpha| = \sqrt{a^2 + b^2}.$$

The conjugate and the absolute value are related through the property $|\alpha|^2 = \alpha \cdot \overline{\alpha}$. Indeed

$$\begin{aligned} \alpha \cdot \overline{\alpha} &= (a + b \cdot i)(a - b \cdot i) \\ &= a^2 + ab \cdot i - ab \cdot i + (b \cdot i)(-b \cdot i) \\ &= a^2 - i^2 \cdot b^2 \\ &= a^2 + b^2 \end{aligned}$$

1.3.2 Radial Representation. Consider the complex number $\alpha = a+b\cdot i$, and assume $\alpha \neq 0$. One can rewrite it as

$$\begin{aligned}\alpha &= \frac{|\alpha|}{|\alpha|}(a + b \cdot i) \\ &= |\alpha| \cdot \left(\frac{a}{|\alpha|} + \frac{b}{|\alpha|} \cdot i\right)\end{aligned}$$

We have

$$\begin{aligned}\left(\frac{a}{|\alpha|}\right)^2 + \left(\frac{b}{|\alpha|}\right)^2 &= \frac{a^2}{|\alpha|^2} + \frac{b^2}{|\alpha|^2} \\ &= \frac{a^2 + b^2}{|\alpha|^2} \\ &= 1\end{aligned}$$

since $|\alpha|^2 = a^2 + b^2$. So there exists an angle $\theta \in [0, 2\pi)$ such that $\cos(\theta) = \frac{a}{|\alpha|}$ and $\sin(\theta) = \frac{b}{|\alpha|}$, meaning that

$$\alpha = |\alpha| \cdot (\cos(\theta) + \sin(\theta) \cdot i)$$

If $\alpha \neq 0$, there is a unique such θ.

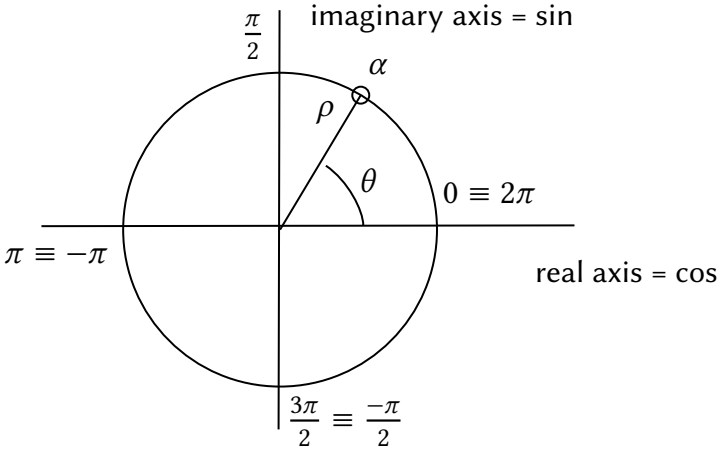
Our complex number can then be written in a canonical form

$$\alpha = \rho\,(\cos(\theta) + \sin(\theta) \cdot i),$$

with ρ non-negative real number:

- ρ is the *amplitude* of α,
- θ is the *phase* of α.

Complex numbers can then be represented in the *complex plane* using a *radial representation* as follows.



$$e^{i\frac{\pi}{2}} = i \tag{7}$$
$$e^{i\pi} = -1 \tag{8}$$
$$e^{2i\pi} = 1 \tag{9}$$
$$e^{i(\theta+2\pi)} = e^{i\theta} \tag{10}$$
$$e^{a+b} = e^a e^b \tag{11}$$
$$\overline{e^a} = e^{\bar{a}} \tag{12}$$
$$\overline{e^{i\theta}} = e^{-i\theta} \tag{13}$$
$$e^{ab} = (e^a)^b \tag{14}$$

Table 2: Equalities regarding exponentiation. θ is real, a and b are complex values.

1.3.3 Exponentiation. The number $\cos(\theta)+i\cdot\sin(\theta)$ can also be written as $e^{i\theta}$. Indeed, remember Table 1: Replacing x with $i\theta$ in Eq. (2), the definition of e^x, we get

$$\begin{aligned}
e^{i\theta} &= \sum_{n=0}^{\infty} \frac{(i\theta)^n}{n!} \\
&= \sum_{n=0}^{\infty} \frac{(i\theta)^{2n}}{(2n)!} + \sum_{n=0}^{\infty} \frac{(i\theta)^{2n+1}}{(2n+1)!} && \text{(splitting odd and even indices)} \\
&= \sum_{n=0}^{\infty} \frac{i^{2n}\theta^{2n}}{(2n)!} + \sum_{n=0}^{\infty} \frac{i^{2n+1}\theta^{2n+1}}{(2n+1)!} && \text{(developing)} \\
&= \sum_{n=0}^{\infty} \frac{(i^2)^n\theta^{2n}}{(2n)!} + \sum_{n=0}^{\infty} \frac{i(i^2)^n\theta^{2n+1}}{(2n+1)!} && \text{(developing)} \\
&= \sum_{n=0}^{\infty} \frac{(-1)^n\theta^{2n}}{(2n)!} + \sum_{n=0}^{\infty} \frac{i(-1)^n\theta^{2n+1}}{(2n+1)!} && \text{(since } i^2 = -1\text{)} \\
&= \sum_{n=0}^{\infty} \frac{(-1)^n\theta^{2n}}{(2n)!} + i\cdot\sum_{n=0}^{\infty} \frac{(-1)^n\theta^{2n+1}}{(2n+1)!} && \text{(factoring by } i\text{)} \\
&= \cos(x) + i\cdot\sin(x) && \text{(using Eqs. (3) and (4)).}
\end{aligned}$$

1.4 Vector Space

1.4.1 In this course, we only consider vector spaces in finite dimension over complex numbers.

1.4.2 Let X be a finite set $\{e_0 \dots e_{n-1}\}$. Each e_i is called a *canonical basis element*. A (complex) *vector space* $\mathscr{E}$ with basis X consists of *vectors*. The *dimension* of the vector space is the number of canonical basis elements in X. A vector v can be regarded as a

mapping from the canonical basis elements in X to complex values. From a computer science point of view, one can regard this vector as an association table, or as a "Python dictionnary":

$$v = \{e_0 : \alpha_0, \dots, e_{n-1} : \alpha_{n-1}\}.$$

We say that α_i is the *coefficient* of v at coordinate e_i, and we write $v_{e_i} = \alpha_i$.

1.4.3 A vector can be represented as an *array*. This however requires an *ordering* of the canonical basis vectors. Array-like notation are discussed later, in Sec. 1.6.8.

1.4.4 The vector space $\mathscr{E}$ is equipped with a sum and a multiplication by a scalar:

$$(+) : \mathscr{E} \times \mathscr{E} \longrightarrow \mathscr{E}$$
$$(\cdot) : \mathbb{C} \times \mathscr{E} \longrightarrow \mathscr{E}$$

Their action is pointwise: if b is a canonical basis element of $\mathscr{E}$, $(v+w)_b = v_b + w_b$ and $(\alpha \cdot v)_b = \alpha \cdot (v_b)$.

1.4.5 Ket-notation. A canonical basis element e_i yields a particular vector written $|e_i\rangle$ and called "*ket-e_i*": it is the vector of coefficient 1 at e_i and 0 everywhere else:

$$|e_i\rangle \quad \triangleq \quad e_j \mapsto \begin{cases} 1 & \text{when } j = i \\ 0 & \text{else} \end{cases}.$$

Using this convention, a vector $v \in \mathscr{E}$ can be written as the linear combination

$$\sum_{i=0}^{n-1} v_{e_i} \cdot |e_i\rangle\,,$$

or, equivalently:

$$\sum_{x \in X} v_x \cdot |x\rangle\,.$$

Such combinations behave well with sum and scalar multiplication:

$$\left(\sum_{x\in X} v_x \cdot |x\rangle\right) + \left(\sum_{x\in X} w_x \cdot |x\rangle\right) = \sum_{x\in X} (v_x + w_x) \cdot |x\rangle\,,$$
$$\alpha \cdot \left(\sum_{x\in X} v_x \cdot |x\rangle\right) = \sum_{x\in X} (\alpha \cdot v_x) \cdot |x\rangle\,.$$

1.5 Scalar Product

1.5.1 Let $\mathscr{E}$ be a complex vector space as above. We define an operation acting on two vectors

$$\langle - \mid - \rangle : \mathscr{E} \times \mathscr{E} \longmapsto \mathbb{C}$$

called a *scalar product*, defined as

$$\langle\, u \mid v \,\rangle \triangleq \sum_x \overline{u_x} v_x.$$

The operation is linear on the right:

$$\langle\, u \mid \alpha \cdot v + w \,\rangle = \alpha \cdot \langle\, u \mid v \,\rangle + \langle\, u \mid w \,\rangle$$

but anti-linear on the left:

$$\langle\, \alpha \cdot u + v \mid w \,\rangle = \overline{\alpha} \cdot \langle\, u \mid v \,\rangle + \langle\, v \mid w \,\rangle\,.$$

In particular, it is anti-symmetric as follows:

$$\langle\, u \mid v \,\rangle = \overline{\langle\, v \mid u \,\rangle}.$$

1.5.2 A (complex) *Hilbert space* is then a complex vector space equipped with such a scalar product. From this scalar product one can define two notions: a notion of norm, and a notion of orthogonality.

1.5.3 Norm. Given a Hilbert space $\mathscr{E}$ with basis X and a vector $v \in \mathscr{E}$, we define $|v|$ as $\sqrt{\langle v|v\rangle}$. Since this is $\sqrt{\sum_x \overline{v_x} v_x} = \sqrt{\sum_x |v_x|^2}$, this is always a non-negative real number. The following properties hold.

$$\begin{aligned} |\alpha \cdot v| &= |\alpha| \cdot |v|, \\ |v + w| &\leq |v| + |w|, \\ |x| &= 1 \ \text{ when } x \in X. \end{aligned}$$

1.5.4 Orthogonality. Given a Hilbert space $\mathscr{E}$ with basis X and two vectors $v, w \in \mathscr{E}$, we say that v is *orthogonal* to w, written $v \perp w$, if $\langle v|w\rangle = 0$. By definition of the scalar product, distinct canonical basis elements are pairwise orthogonal.

1.5.5 Orthonormal basis. In general, a *basis* for a vector space is a set of vectors spanning the whole space. The elements of a basis are called *basis elements*. A basis is an *orthonormal basis* if basis elements are each of norm 1, and pairwise orthogonal.

1.5.6 For the Hilbert space $\mathscr{H}$, an example of non-canonical, orthonormal basis is the set $\{\ |+\rangle, |-\rangle\ \}$, where

$$\begin{aligned} |+\rangle &\triangleq \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \\ |-\rangle &\triangleq \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \end{aligned}$$

1.6 Kets and Bras

1.6.1 In quantum computation, the canonical basis elements are representing classical values: a linear combination can then be seen as a "superposition" of classical values. As we discussed in Sec. 1.4.5, a standard notation for making canonical basis elements out of arbitrary classical values is the *ket-notation*

$$|\,\rangle\,.$$

1.6.2 The two-dimensional Hilbert space $\mathscr{H}$ that we shall be considering in these notes is built from the canonical basis set $\{0, 1\}$. A typical vector of $\mathscr{H}$ is then

$$\alpha \cdot |0\rangle + \beta \cdot |1\rangle\,.$$

The chosen "0" and "1" stands for two classical, distinguished values, such as "False" and "True" respectively. Their name is only a convention: one could have chosen any other pair of names such as $|\mathrm{T}\rangle$ and $|\mathrm{F}\rangle$, $|\mathrm{foo}\rangle$ and $|\mathrm{bar}\rangle$... They should in particular not be confused with the complex scalars 0 and 1. Similarly, $|0\rangle$ is NOT the null element of the vector space, so

$$|0\rangle \neq 0.$$

1.6.3 You might have noticed the clash of notation between

$$\langle\, u \mid v \,\rangle$$

and for instance

$$\alpha \cdot |0\rangle + \beta \cdot |1\rangle\,.$$

This is on purpose, and yields a pun: if the notation $|-\rangle$ is called a "ket", we call $\langle -|$ a "bra", which yield the standard name "braket" for $\langle -|-\rangle$ (yes, this notation is called "braket" in English), so that

$$\langle\, x \mid y \,\rangle = \langle x| \cdot |y\rangle\,,$$

the "multiplication", or the *application* of the function $\langle x|$ to the argument $|y\rangle$. If $x \in X$ is a basis element for $\mathscr{E}$, the notation $\langle x|$ stands for the linear operation

$$\langle x| \,:\, y \longmapsto \langle\, x \mid y \,\rangle \;=\; \begin{cases} 1 & \text{if } x = y \\ 0 & \text{else} \end{cases}\,.$$

By linearity, we then have $\langle x \mid u\rangle = \langle x| \cdot |u\rangle$ for a general vector u in $\mathscr{E}$.

1.6.4 Duality bra-ket. for the vector space $\mathscr{E}$ of basis elements $x \in X$, the bras $\langle x|$ are *functionals*: functions $\mathscr{E} \to \mathbb{C}$. As such, they can be equipped with a sum and a (complex) scalar multiplication. The functional $\alpha\,\langle x| + \langle y|$ is the map

$$\alpha\,\langle x| + \langle y| \,:\, u \mapsto \alpha\,\langle\, x \mid u \,\rangle + \langle\, y \mid u \,\rangle\,,$$

There is a strong duality: if $u = \sum_x \alpha_x\,|x\rangle$ and $v = \sum_x \beta_x \cdot |x\rangle$, then one can check that

$$\langle\, u \mid v \,\rangle = \left(\sum_x \overline{\alpha_x} \cdot \langle x|\right)\left(\sum_x \beta_x \cdot |x\rangle\right) = u^* \cdot v$$

where $u^* = \sum_x \overline{\alpha_x} \cdot \langle x|$ is the *dual*, or *conjugate transpose* of u.

1.6.5 Notation. Vectors of the form $\sum_{x\in X} \alpha_x \cdot |x\rangle$ will be written with greek letters inside kets, such as $|\phi\rangle$, $|\psi\rangle$, *etc.* When using lower-case latin letters $|x\rangle$, $|y\rangle$, $|b\rangle$, $|c\rangle$, we shall be referring to canonical basis elements. Functionals of the form $\sum_{x\in X} \alpha_x \cdot \langle x|$ will dually be written as $\langle\phi|$, $\langle\psi|$, *etc.* We then have the property that

$$|\phi\rangle^* = \langle\phi| \quad \text{and} \quad \langle\phi|^* = |\phi\rangle\,.$$

The application of a functional (i.e. a bra) to a vector (i.e. a ket) is always written as a multiplication.

1.6.6 Remark Bras and kets can contain more than letters and numbers: we use for instance

$$|+\rangle \triangleq \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \quad |-\rangle \triangleq \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

in 1.5.6 and

$$|+i\rangle \triangleq \frac{1}{\sqrt{2}}(|0\rangle + i\,|1\rangle), \quad |-i\rangle \triangleq \frac{1}{\sqrt{2}}(|0\rangle - i\,|1\rangle)$$

in 2.2.10.

1.6.7 Beware! Do not forget to conjugate the complex coefficient when moving from kets to bras and bras to kets.

1.6.8 Matrix-style representation. Bras, kets, and linear operations in general can be represented in a matrix-style notation. For this to make sense, we need to *order* the canonical basis elements of the vector spaces. For the Hilbert space $\mathscr{H}$, the basis $|0\rangle$,$|1\rangle$ is ordered in the *lexicographic order*. A ket is in this convention a column-vector as follow

$$|0\rangle \quad = \quad \begin{pmatrix}1\\0\end{pmatrix}, \qquad |1\rangle \quad = \quad \begin{pmatrix}0\\1\end{pmatrix},$$

so that

$$\alpha\cdot|0\rangle + \beta\cdot|1\rangle \quad = \quad \begin{pmatrix}\alpha\\\beta\end{pmatrix}.$$

A bra becomes a row-vector :

$$\langle 0| = \begin{pmatrix}1 & 0\end{pmatrix}, \quad \langle 1| = \begin{pmatrix}0 & 1\end{pmatrix}.$$

If $|\phi\rangle = \alpha\cdot|0\rangle + \beta\cdot|1\rangle$ and $|\psi\rangle = \gamma\cdot|0\rangle + \delta\cdot|1\rangle$, we can check that

$$\langle\,\phi\,|\,\psi\,\rangle = \begin{pmatrix}\overline{\alpha} & \overline{\beta}\end{pmatrix}\cdot\begin{pmatrix}\gamma\\\delta\end{pmatrix} = \overline{\alpha}\gamma + \overline{\beta}\delta.$$

The conjugate transpose of a column vector is a row vector and vis versa, as follows.

$$\begin{pmatrix}\alpha\\\beta\end{pmatrix}^* = \begin{pmatrix}\overline{\alpha} & \overline{\beta}\end{pmatrix}, \quad \begin{pmatrix}\alpha & \beta\end{pmatrix}^* = \begin{pmatrix}\overline{\alpha}\\\overline{\beta}\end{pmatrix}.$$

1.6.9 With the matrix-style representation of 1.6.8, kets and bras can be combined in arbitrary ways, as long as dimensions match. For instance,

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = a\,|0\rangle\langle 0| + c\,|1\rangle\langle 0| + b\,|0\rangle\langle 1| + d\,|1\rangle\langle 1|\,.$$

Multiplication by a ket-vector is then transparent. For instance:

$$\begin{aligned}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}(\alpha\,|0\rangle + \beta\,|1\rangle) &= (|0\rangle\langle 1| + |1\rangle\langle 0|)(\alpha\,|0\rangle + \beta\,|1\rangle)\\ &= \alpha\,|0\rangle\langle 1|\,|0\rangle + \alpha\,|1\rangle\langle 0|\,|0\rangle + \beta\,|0\rangle\langle 1|\,|1\rangle + \beta\,|1\rangle\langle 0|\,|1\rangle\\ &= \alpha\,|0\rangle\langle 1\,|\,0\rangle + \alpha\,|1\rangle\langle 0\,|\,0\rangle + \beta\,|0\rangle\langle 1\,|\,1\rangle + \beta\,|1\rangle\langle 0\,|\,1\rangle\\ &= \alpha\,\langle 1\,|\,0\rangle\,|0\rangle + \alpha\,\langle 0\,|\,0\rangle\,|1\rangle + \beta\,\langle 1\,|\,1\rangle\,|0\rangle + \beta\,\langle 0\,|\,1\rangle\,|1\rangle\\ &= \alpha\cdot 0\cdot|0\rangle + \alpha\cdot 1\cdot|1\rangle + \beta\cdot 1\cdot|0\rangle + \beta\cdot 0\cdot|1\rangle\\ &= \alpha\,|1\rangle + \beta\,|0\rangle\end{aligned}$$

1.7 Kronecker product

1.7.1 Consider two vector spaces $\mathscr{E}$ of canonical basis $B = \{e_i\}_i$ and $\mathscr{F}$ of canonical basis $C = \{f_j\}_j$. We define a new vector space $\mathscr{E}\otimes\mathscr{F}$ using the canonical basis set $B\times C$, the cartesian product of B and C. The space $\mathscr{E}\otimes\mathscr{F}$ is called the *Kronecker product* of $\mathscr{E}$ and $\mathscr{F}$, or *tensor* of $\mathscr{E}$ and $\mathscr{F}$. We could write the pair of e_i and f_j in various manner such as (e_i, f_j), e_i, f_j, $e_i f_j$ *etc.* We shall be usig simple concatenation when clear. A typical vector of $\mathscr{E}\otimes\mathscr{F}$ is then of the form

$$\sum_{i,j}\alpha_{i,j}\,|e_i f_j\rangle\,.$$

1.7.2 The tensor notation is overloaded to vectors: it is used to represent the *bilinear map* $\mathscr{E}\times\mathscr{F}\to\mathscr{E}\otimes\mathscr{F}$ defined as

$$\left(\sum_i \alpha_i\cdot|e_i\rangle\right)\otimes\left(\sum_j \beta_j\cdot|f_j\rangle\right) \quad = \quad \sum_{i,j}\alpha_i\beta_j\cdot|e_i f_j\rangle\,.$$

It is bilinear in the sense that the following properties hold.

$$(\alpha\cdot|\phi\rangle + |\psi\rangle)\otimes w = \alpha\cdot(|\phi\rangle\otimes w) + |\psi\rangle\otimes w, \tag{15}$$
$$w\otimes(\alpha\cdot|\phi\rangle + |\psi\rangle) = \alpha\cdot(w\otimes|\phi\rangle) + w\otimes|\psi\rangle\,, \tag{16}$$
$$(\alpha\cdot|\phi\rangle)\otimes|\psi\rangle = \alpha\cdot(|\phi\rangle\otimes|\psi\rangle), \tag{17}$$
$$|\phi\rangle\otimes(\alpha\cdot|\psi\rangle) = \alpha\cdot(|\phi\rangle\otimes|\psi\rangle), \tag{18}$$
$$0\otimes|\phi\rangle = 0. \tag{19}$$
$$|\phi\rangle\otimes 0 = 0. \tag{20}$$

1.7.3 As the vector space $\mathscr{E}\otimes\mathscr{F}$ is "just" a vector space, the definitions of scalar product and norm presented in Sec. 1.5 still hold. In particular, if

$$|\phi\rangle = \sum_{i,j} \alpha_{i,j} \cdot |e_i f_j\rangle\,, \qquad\qquad |\psi\rangle = \sum_{i,j} \beta_{i,j} \cdot |e_i f_j\rangle\,,$$

then

$$\langle\ \phi\ |\ \psi\ \rangle = \left(\sum_{i,j} \overline{\alpha_{i,j}} \cdot |e_i f_j\rangle\right)\left(\sum_{i,j} \beta_{i,j} \cdot |e_i f_j\rangle\right) = \sum_{i,j} \overline{\alpha_{i,j}}\beta_{i,j},$$

and

$$|\ \psi\ | = \sqrt{\sum_{i,j} |\alpha_{i,j}|^2}.$$

If instead $|\phi\rangle = \sum_i \alpha_i \cdot |e_i\rangle$ and $|\psi\rangle = \sum_j \beta_j \cdot |f_j\rangle$, one can then derive that

$$\||\phi\rangle \otimes |\psi\rangle| = \||\phi\rangle| \cdot \||\psi\rangle|,$$

and if moreover $|\phi'\rangle = \sum_i \alpha'_i \cdot |e_i\rangle$ and $|\psi'\rangle = \sum_j \beta'_j \cdot |f_j\rangle$,

$$(\langle\phi| \otimes \langle\psi|)(|\phi'\rangle \otimes |\psi'\rangle) = \langle\ \phi\ |\ \phi'\ \rangle\,\langle\ \psi\ |\ \psi'\ \rangle\,.$$

1.7.4 Notation The notation of Sec. 1.7.1 is overloaded for arbitrary kets. For instance, remember that $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$: we write $|+0\rangle$ for the vector

$$\begin{aligned} |+0\rangle &= |+\rangle \otimes |0\rangle \\ &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle \\ &= \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle). \end{aligned}$$

In general, $|\phi\psi\rangle$ is $|\phi\rangle \otimes |\psi\rangle$.

1.8 Linear Maps

1.8.1 Given two vector spaces $\mathscr{E}$ and $\mathscr{F}$ with respective bases $\{e_i\}_i$ and $\{f_j\}_j$, a *linear map* (or linear *operator*) f from $\mathscr{E}$ to $\mathscr{F}$ is a set-function from $\mathscr{E}$ to $\mathscr{F}$ such that

$$f(u+v) = f(u) + f(v), \qquad f(\alpha \cdot u) = \alpha \cdot f(u).$$

1.8.2 A linear function from $\mathscr{E}$ to $\mathscr{F}$ is uniquely characterized by its action on the basis elements. Using the ket-notation :

$$f\left(\sum_x \alpha_x \cdot |x\rangle\right) = \sum_x \alpha_x \cdot f(|x\rangle).$$

1.8.3 Provided that the canonical basis is ordered, one can use matrices to represent linear maps. In dimension 2, a matrix has the generic form

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}.$$

In the *coefficient* a_{ij}, the index i stands for the line number while j stands for the column number. Provided that we use the lexicographic ordering on canonical basis states as in Sec. 1.6.8, this matrix represents the linear map $\mathscr{H} \to \mathscr{H}$ defined by

$$\begin{aligned} |0\rangle &\mapsto a_{11} \cdot |0\rangle + a_{21} \cdot |1\rangle, \\ |1\rangle &\mapsto a_{12} \cdot |0\rangle + a_{22} \cdot |1\rangle. \end{aligned}$$

1.8.4 If A and B are two $n \times n$ matrices and if a_{ij} and b_{ij} are their respective coefficients on line i and column j, then the coefficient at position (i, j) of the matrix $A \cdot B$ is $\sum_k a_{ik} b_{kj}$. For instance, in dimension 2:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & \dots \\ \dots & \dots \end{pmatrix}.$$

1.8.5 The action of a linear operator on a vector becomes the matrix multiplication with the corresponding vector. Indeed, a column vector is "just" a matrix with only one column, so

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} a_{11}v_1 + a_{12}v_2 \\ a_{21}v_1 + a_{22}v_2 \end{pmatrix}$$

If A is the matrix of a linear operation f and B the matrix of g, then

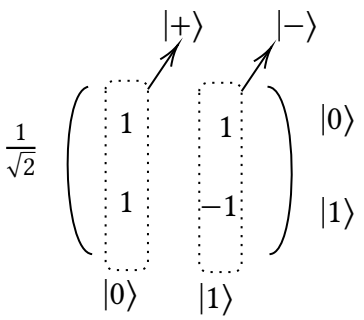
$$f(g(v)) = A \cdot (B \cdot v) = (A \cdot B) \cdot v,$$

so $f \circ g$ corresponds to $A \cdot B$.

1.8.6 For instance, the linear map $\textsc{Had} : \mathscr{H} \to \mathscr{H}$ is defined using the kets introduced in Sec. 1.5.6 as follows:

$$\textsc{Had} : \begin{cases} |0\rangle \mapsto |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \\ |1\rangle \mapsto |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \end{cases}$$

It can be represented with the matrix



(21)

The first column is $|+\rangle$, the output of the function at $|0\rangle$, and the second column is $|-\rangle$, the output of the function at $|1\rangle$.

1.8.7 The operation HAD is called *Hadamard*. It can be written in a more compact way as

$$|x\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + (-1)^x |1\rangle),$$

with the convention of Sec. 1.6.5 that x is one of the basis-set element "0" or "1". the exponent $(-1)^x$ then assimilates x to its "integer" value, and $(-1)^0 = 1$ while $(-1)^1 = -1$.

1.8.8 The application of the function to a vector corresponds to matrix-vector multiplication. for instance, HAD $|0\rangle$ is

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}1\\ 0\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\ 1\end{pmatrix}.$$

1.8.9 Similarly, the composition of functions corresponds to matrix multiplication. One can for instance check that HAD$\circ$HAD is the identity. Using the definition from Sec. 1.8.7:

$$\begin{aligned}
\text{HAD}(\text{HAD}\,|x\rangle) &= \text{HAD}(\frac{1}{\sqrt{2}}(|0\rangle + (-1)^x |1\rangle))\\
&= \frac{1}{\sqrt{2}}(\text{HAD}\,|0\rangle + (-1)^x \text{HAD}\,|1\rangle)\\
&= \frac{1}{\sqrt{2}}(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) + (-1)^x \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle))\\
&= \frac{1}{2}(|0\rangle + |1\rangle + (-1)^x |0\rangle - (-1)^x |1\rangle)\\
&= \frac{1}{2}((1 + (-1)^x)|0\rangle + (1 - (-1)^x)|1\rangle)\\
&= \begin{cases} \frac{1}{2}(2\cdot|0\rangle + 0\cdot|1\rangle) & \text{if } x = 0,\\ \frac{1}{2}(0\cdot|0\rangle + 2\cdot|1\rangle) & \text{if } x = 1\end{cases}\\
&= |x\rangle.
\end{aligned}$$

One can then check that

$$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\cdot\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}.$$

1.8.10 As for column and row vectors, we can define a notion of conjugate transpose for matrices:

$$\begin{pmatrix}\alpha_{1,1} & \cdots & \alpha_{1,n}\\ \vdots & \ddots & \vdots\\ \alpha_{m,1} & \cdots & \alpha_{m,n}\end{pmatrix}^* \triangleq \begin{pmatrix}\overline{\alpha_{1,1}} & \cdots & \overline{\alpha_{m,1}}\\ \vdots & \ddots & \vdots\\ \overline{\alpha_{1,n}} & \cdots & \overline{\alpha_{m,n}}\end{pmatrix}$$

1.8.11 We shall be using I and Id for the identity map on vector spaces. This is obviously a linear map.

1.8.12 The Kronecker product is also overloaded to functions, as follows. Given two linear maps $U : \mathscr{E} \to \mathscr{E}$ and $V : \mathscr{F} \to \mathscr{F}$, we define the linear map

$$U \otimes V : \mathscr{E} \otimes \mathscr{F} \to \mathscr{E} \to \mathscr{F}$$

by

$$(U \otimes V)\,|xy\rangle \triangleq (U\,|x\rangle) \otimes (V\,|y\rangle).$$

Because of the linearity of the operation, function application and composition permutes with the tensor as follows. Let $|\phi\rangle \in \mathscr{E}$, $|\psi\rangle \in \mathscr{F}$, $U' : \mathscr{E} \to \mathscr{E}$ and $V' : \mathscr{F} \to \mathscr{F}$, then

$$(U \otimes V)(|\phi\rangle \otimes |\psi\rangle) = (U\,|\phi\rangle) \otimes (V\,|\psi\rangle), \tag{22}$$
$$(U \otimes V) \circ (U' \otimes V') = (U \circ U') \otimes (V \circ V'). \tag{23}$$

1.8.13 The tensor symbol can be regarded as forming an impermeable, spacial separation between the left side and the right side.

> "What's on the left stays on the left;
> what's on the right stays on the right."

1.8.14 Note how the symbol "$\otimes$" has been overloaded 3 times: for vector spaces (as in "$\mathscr{E} \otimes \mathscr{F}$"), for vectors (as in "$|\phi\rangle \otimes |\psi\rangle$"), and for linear operations (as in "$U \otimes V$").

1.9 Hermitian and Unitary Maps

1.9.1 A linear map $U : \mathscr{E} \to \mathscr{E}$ is called *unitary* if

- it is invertible : there exists a linear map $\mathscr{E} \to \mathscr{E}$ written U^{-1} such that $U^{-1} \circ U$ and $U \circ U^{-1}$ are the identity,
- the inverse of U is its conjugate transpose: $U^{-1} = U^*$.

Unitary maps preserves norm and orthogonality: they can be regarded as rotations. For instance, the map Hadamard from Sec. 1.8.6 is unitary. It sends the canonical basis $\{\ |0\rangle, |1\rangle\ \}$ to the basis $\{\ |+\rangle, |-\rangle\ \}$.

1.9.2 If U is a unitary map, then

$$\langle \phi\,|\,\psi\rangle = \langle U\phi\,|\,U\psi\rangle\,.$$

1.9.3 If $U\,|\psi\rangle = \lambda\,|\psi\rangle$, we say that $|\psi\rangle$ is an *eigenvector* of U and λ an *eigenvalue* In the case where U is unitary, λ is of the form $e^{2i\pi\omega}$ with $0 \leq \omega < 1$ a real number, because U preserves the norm, and thus $|\lambda| = 1$. By abuse of language, we say that ω is the *phase* of the eigenvalue.

1.9.4 A linear map $H : \mathscr{E} \to \mathscr{E}$ is called *Hermitian* when $H = H^*$. Equivalently:

- If $(h_{i,j})_{i,j}$ is a matrix representation of H in an orthonormal basis, then $h_{i,j} = \overline{h_{j,i}}$.
- H admits a set of eigenvectors $\{|u_j\rangle\}_j$ forming an orthonormal basis, and all of its eigenvalues are real numbers.

With the convention in 1.6.9, the Hermitian operator can be written as

$$H = \sum_j \lambda_j |u_j\rangle \langle u_j| , \tag{24}$$

when λ_j is the eigenvalue corresponding to $|u_j\rangle$. One can check that

$$\begin{aligned} H|u_{j_0}\rangle &= \sum_j \lambda_j |u_j\rangle \langle u_j|u_{j_0}\rangle \\ &= \lambda_{j_0} |u_{j_0}\rangle \langle u_{j_0}|u_{j_0}\rangle + \sum_{j\neq j_0} \lambda_j |u_j\rangle \langle u_j|u_{j_0}\rangle \\ &= \lambda_{j_0} \langle u_{j_0}|u_{j_0}\rangle |u_{j_0}\rangle + \sum_{j\neq j_0} \lambda_j \langle u_j|u_{j_0}\rangle |u_j\rangle \\ &= \lambda_{j_0} \cdot 1 \cdot |u_{j_0}\rangle + \sum_{j\neq j_0} \lambda_j \cdot 0 \cdot |u_j\rangle \\ &= \lambda_{j_0} |u_{j_0}\rangle \end{aligned}$$

since the $|u_j\rangle$s were picked pairwise distinct.

1.9.5 The eigenvalues of a Hermitian matrix are all real numbers, and there is a finite number of them. One of them is them *minimal*, the other *maximal*. In general, we talk about *extremal* eigenvalues, and by abuse of notation we speak of minimal / maximal / extremal eigenvectors to refer to the corresponing eigenvectors. Note that (say) minimal eigenvector are not unique. For instance, the identity has many of them.

1.9.6 In the context of quantum computation, a Hermitian map is often called a *Hamiltonian*. Hamiltonian operators are used to model physical properties of a system such as its energy. For the purpose of this course, we can safely stay at this (low) level of understanding.

1.9.7 The set of Hermitian maps forms a real vector field: it is closed under addition and (real) scalar multiplication.

1.9.8 Unitary and Hermitian operators are related through the exponential formula in Eq (2): If H is Hermitian, then

$$e^{iH} = \sum_{k=0}^{\infty} \frac{1}{k!} H^k$$

is unitary. Conversely, every unitary U can be written as e^{iH} for a Hermitian operator H.

1.9.9 Eigenvectors are preserved through exponentiation. Suppose that $|\phi\rangle$ is an eigenvector of the Hermitian H with eigenvalue λ, so $H|\phi\rangle = \lambda|\phi\rangle$. Then

$$\begin{aligned}(e^{iH})|\phi\rangle &= \left(\sum_{k=0}^{\infty}\frac{1}{k!}H^k\right)|\phi\rangle \\ &= \sum_{k=0}^{\infty}\frac{1}{k!}H^k|\phi\rangle \\ &= \sum_{k=0}^{\infty}\frac{1}{k!}\lambda^k|\phi\rangle \\ &= \left(\sum_{k=0}^{\infty}\frac{1}{k!}\lambda^k\right)|\phi\rangle \\ &= e^{i\lambda}|\phi\rangle.\end{aligned}$$

The eigenvalue of e^{iH} corresponding to $|\phi\rangle$ is $e^{i\lambda}$.

1.9.10 If A and B are Hermitian operators (or, in general, square matrices) such that $AB = BA$, then

$$e^{A+B} = e^A e^B.$$

1.9.11 For any A and B Hermitian operators (or, in general, square matrices) without special conditions,

$$e^{A\otimes B} = e^A \otimes e^B.$$

1.10 Exercices

1.10.1 Consider the vector $|\phi\rangle = \frac{i}{3}|0\rangle + \frac{2\sqrt{2}}{3}|1\rangle$ in $\mathscr{H}$.

1. Show that $|\phi\rangle$ is of norm 1.
2. Compute $\langle\phi|$.
3. Compute $\langle\phi|+\rangle$ and $\langle-|\phi\rangle$.
4. The vector $|\phi\rangle$ has been given in the basis $\{|0\rangle, |1\rangle\}$. Write it in the basis $\{|+\rangle, |-\rangle\}$.
5. Give an vector of norm 1 orthogonal to $|\phi\rangle$

1.10.2 Assume that x is either 0 or 1. Show that

$$\frac{1}{\sqrt{2}}(|x\rangle - |1\oplus x\rangle) = (-1)^x\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle).$$

The symbol "$\oplus$" is the XOR bit operation.

1.10.3 Consider the set $\mathscr{B} \triangleq \{e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$ and a bijection σ on the set $\mathscr{B}$. Let $\mathscr{E}$ be the Hilbert space generated by the set $\mathscr{B}$, and U the linear map $\mathscr{E} \to \mathscr{E}$ defined as

$$U \ : \ |e_k\rangle \mapsto |\sigma(e_k)\rangle \qquad \text{for } k = 0, \dots, 7$$

Show that this map is unitary.

1.10.4 Consider the setting of Exo 1.10.3. Let σ be the map sending e_k to $3 \cdot k \mod 8$, that is:

$$\begin{array}{cccc} 0 \mapsto 0 & 2 \mapsto 6 & 4 \mapsto 4 & 6 \mapsto 2 \\ 1 \mapsto 3 & 3 \mapsto 1 & 5 \mapsto 7 & 7 \mapsto 5 \end{array}$$

Write the matrix U corresponding to the unitary based on σ with the basis ordering

$$e_0, e_1, e_2, e_3, e_4, e_5, e_6, e_7.$$

What does it means when $U_{m,n} = 1$?

1.10.5 Consider the ket vectors of $\mathscr{H} \otimes \mathscr{H}$

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$
$$|\Phi^-\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$$
$$|\Psi^+\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$$
$$|\Psi^-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$$

1. Show that these 4 vectors form an orthonormal basis
2. Consider the ket vector
$$|\phi\rangle = \frac{2}{\sqrt{5}}|01\rangle + \frac{i}{\sqrt{5}}|10\rangle$$
   (a) Show that this is a vector of norm 1.
   (b) Write it as a linear combination of $|\Phi^+\rangle$, $|\Phi^-\rangle$, $|\Psi^+\rangle$ and $|\Psi^-\rangle$.
   (c) Compute $\langle\, \phi \mid \Psi^+ \,\rangle$.

1.10.6 Show that for every vector $|\phi\rangle$ in $\mathscr{H}$, we have

$$|\phi\rangle = \langle 0|\phi\rangle \cdot |0\rangle + \langle 1|\phi\rangle \cdot |1\rangle$$

1.10.7 Consider the vectors $|\phi\rangle = \frac{i}{3}|0\rangle + \frac{2\sqrt{2}}{3}|1\rangle$ and $|\psi\rangle = \frac{1}{\sqrt{5}}|0\rangle + \frac{2}{\sqrt{5}}|1\rangle$ in $\mathscr{H}$.

1. Compute $|\phi\rangle \otimes |\psi\rangle$ in the basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$.
2. Compute $|\phi\rangle \otimes |\psi\rangle$ in the basis $\{|{+}{+}\rangle, |{+}{-}\rangle, |{-}{+}\rangle, |{-}{-}\rangle\}$.
3. Give an orthonormal basis of $\mathscr{H} \otimes \mathscr{H}$ for which $|\phi\rangle \otimes |\psi\rangle$ is one of the elements.

1.10.8 Consider the operation $\odot$ acting on two bitstrings:

$$(x_1 \dots x_n) \odot (y_1 \dots y_n) = (x_1 \wedge y_1) \oplus \dots \oplus (x_n \wedge y_n).$$

Show that $\textsc{Had}^{\otimes n}$ performs the action

$$|x_1 \dots x_n\rangle \longmapsto \frac{1}{\sqrt{2}^n} \sum_{y_1 \dots y_n} (-1)^{(x_1 \dots x_n) \odot (y_1 \dots y_n)} |y_1 \dots y_n\rangle .$$

1.10.9 Consider the linear maps (called *Pauli* maps)

$$X \triangleq \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Z \triangleq \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad Y \triangleq \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

in the ordered basis $\{|0\rangle, |1\rangle\}$. For each $G = X, Y, Z$:

1. Show that G is both unitary and hermitian
2. Give a map $\sqrt{G}$ such $\sqrt{G}\sqrt{G} = G$
3. If θ is a real number, compute $e^{i\theta G}$

1.10.10 A known result is that any Hermitian matrix of size $2^n \times 2^n$ can be written as a linear combination of tensors of Pauli matrices (and identity) with real coefficients. More precisely, using the notations of Exo 1.10.9, any Hermitian matrix of dimension $2^n \times 2^n$ can be written as

$$\sum_{G_1, \dots G_n \in \{X,Y,Z,I\}} h_{G_1,\dots,G_n} \cdot G_1 \otimes \dots \otimes G_n$$

with $h_{G_1,\dots,G_n} \in \mathbb{R}$. This exercise focuses on the 4×4 case.

1. Compute the 16 following 4×4 matrices

$$\begin{array}{cccccccc} X \otimes X, & X \otimes Y, & X \otimes Z, & X \otimes I, & Y \otimes X, & Y \otimes Y, & Y \otimes Z, & Y \otimes I, \\ Z \otimes X, & Z \otimes Y, & Z \otimes Z, & Z \otimes I, & I \otimes X, & I \otimes Y, & I \otimes Z, & I \otimes I. \end{array}$$

2. Consider the following Hermitian matrix:

$$\begin{pmatrix} 2 & 0 & 4-i & 0 \\ 0 & 5 & 3 & 0 \\ 4+i & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Write it as a linear decomposition of the 16 matrices in the first question, with real coefficients.

1.10.11 Consider the following matrix.

$$M = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

Questions:

1. Write M as a linear combination of Pauli matrices (and identity)
2. Find the eigenvectors and eigenvectors of M
3. Write M in the form of Eq.(24) in 1.9.4.

1.10.12 Using the notations of Exo 1.10.9, compute the following (θ is a real number).

$$\begin{array}{lll} \textsc{Had} \cdot X \cdot \textsc{Had} & \textsc{Had} \cdot Z \cdot \textsc{Had} & \textsc{Had} \cdot Y \cdot \textsc{Had} \\ \textsc{Had} \cdot e^{i\theta X} \cdot \textsc{Had} & \textsc{Had} \cdot e^{i\theta Z} \cdot \textsc{Had} & \textsc{Had} \cdot e^{i\theta Y} \cdot \textsc{Had} \end{array}$$

1.10.13 Using the notations of Exo 1.10.9, compute the following (θ is a real number).

$$e^{i\theta(Z\otimes Z)}, \qquad e^{i\theta(X\otimes X)}, \qquad e^{i\theta(X\otimes Z)}.$$

You might want to consider doing first Exos 1.10.9 and 1.10.12.

1.10.14 Consider the matrices X, Y and Z of Exo.1.10.9, together with Had defined in 1.8.6. Fill in the multiplication table starting with

$\setminus$	I	X	Y	Z	$X \cdot Z$	Had	$\cdots$
I							
X							
Y							
Z							
$X \cdot Z$							
Had		Had $\cdot X$					
$\vdots$							

You will have to add more lines and columns. For instance, Had $\cdot X$ is not in the list and should be added. But beware for redundancy: we only want to add matrices not already there (for instance, check Had $\cdot X$ and $Z \cdot$ Had). Overall, how many (distinct) matrices are obtained?

1.10.15 Consider the following two linear maps:

$$\text{SWAP} : |00\rangle \mapsto |00\rangle \qquad |01\rangle \mapsto |10\rangle \qquad |11\rangle \mapsto |11\rangle \qquad |10\rangle \mapsto |01\rangle$$

$$\text{CNOT} : |00\rangle \mapsto |00\rangle \qquad |10\rangle \mapsto |11\rangle \qquad |01\rangle \mapsto |01\rangle \qquad |11\rangle \mapsto |10\rangle$$

For each of the following linear maps acting on $\mathscr{H}\otimes\mathscr{H}$, give all its possible representative matrices depending on the orderings of the basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ (and for each matrix, give the corresponding ordering(s)).

2 Qubit-based Computation

2.1 The Quantum Co-Processor Model

2.1.1 In the standard interpretation, a quantum computer is a standard, conventional computer together with a particular kind of coprocessor. Other, conventional coprocessor includes: GPUs, FGPAs, TLS/SSL accelerators, *etc.* Inside the computer, the processor is the part that actually *runs* a program; a co-processor is governed by the processor for specific tasks (for instance, a GPU is used for fast matrix operations). A co-processor typically has its own memory, and the operations available to the co-processor will be performed on this local memory.

2.1.2 For our purpose, a typical use of a co-processor is then:

1. Allocate and initialize part of the memory local to the co-processor;
2. Perform some action;
3. Read the memory and free (de-allocate) some part of the memory.

These 3 actions are the primary requirements for a co-processor: we need to be able to set up the memory, read it, and do something on it (ideally non-trivial).

2.1.3 The part of the memory that is being manipulated is modeled in term of *register*: elementary units of data stored on the physical medium and that can be individually addressed. A classical register can for instance hold a single bit, or a byte (i.e. an 8-bit sequence), or 8 bytes (i.e. 64 bits), *etc.* By extension, an array of registers can itself be seen as a register. For instance, an array of 64-bit integers can be described as a register. In the storage of a bit, there are therefore three distinct levels of abstraction:

- Mathematical level: The Boolean algebra $\mathbb{B} = \{ 0, 1 \}$;
- Computer science level: A register holding a bit of information, with a unique identifier to programmatically address it (a *pointer* or a *reference*);
- Physics level: A bunch of transistors wired together, realizing a hardware implementation of the register.

If the word *register* can be used to refer to any of these levels, it is important to keep the distinction in mind, in particular the fact that "register" refer to both a *mathematical value* and a *location*.

2.1.4 Our *quantum* co-processor is designed with the philosophy presented in 2.1.1 and 2.1.2: it holds a *quantum memory* that can be initialized and read, while allowing one to perfom a particular set of (usually) *local* operations. The memory consists of *quantum registers*, and for the purpose of this set of notes, these holds the so-called *quantum bit*, described in Section 2.2.

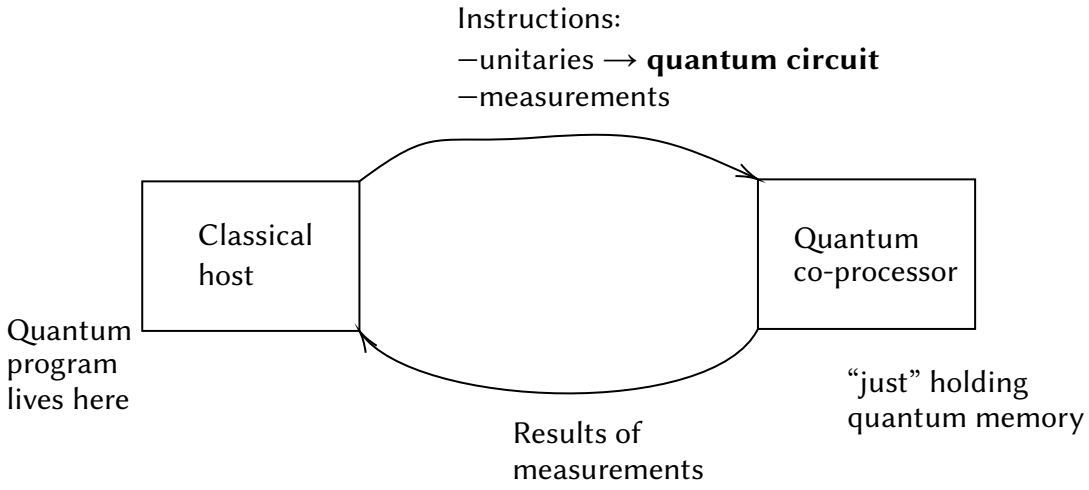


Figure 1: The coprocessor model.

2.2 One Quantum Bit

2.2.1 A *quantum bit*, or *qubit*, corresponds to a piece of quantum memory containing an elementary unit of quantum information.

2.2.2 The easiest way to understand what is going on is to refer to what happens in the classical setting. In the latter, to encode a bit of information, one chooses a medium, and two states. These states have to be *unequivocally distinguished*, and they should be easy to act upon (to encode computation). One such state then represents "True", the other "False". Examples include:

- Coin, head/tail;
- Coin, steel or copper;
- Magnet, north or south facing up;
- Piece of paper, black or white color.

The choice is completely arbitrary and is purely conventional: The knowledge of which states were chosen is mandatory to recover the information.

2.2.3 One follows the same strategy for quantum information. A quantum bit, the smallest piece of quantum information, is encoded on an object governed by the law of quantum mechanics, together with a choice of two states for this object. As for the classical case, one should be able to

- distinguish the two states
- act upon the states: initialize, read, and otherwise modify the state.

Examples include:

- Photon, vertical and horizontal polarization;

- Photon, position (wire A or wire B);
- Electron, spin up or spin down;
- Electron, orbital position;
- Atom, energy level.

2.2.4 Hilbert spaces form the framework for the mathematical formalism to represent the state of a quantum system. For our purpose, such a state is a *normalized vector* in a Hilbert space (up to global phase, see 2.2.5). A quantum bit is represented within the two-dimensional Hilbert space $\mathscr{H}$: the chosen two states are represented with $|0\rangle$ and $|1\rangle$ (called *basis state*). In this framework, distinguishability corresponds to *orthogonality*, while the actions are modeled with unitary maps.

2.2.5 Global phase. Let us formalize the mathematical representation of a quantum bit. Within the state $\mathscr{H}$, define the set of kets

$$S_1 = \left\{ \alpha |0\rangle + \beta |1\rangle \in \mathscr{H} \;\middle|\; |\alpha|^2 + |\beta|^2 = 1 \right\}.$$

We say that two elements $|\phi\rangle, |\psi\rangle \in S_1$ are *related by a global phase* if there exists an angle θ such that $|\psi\rangle = e^{i\theta} |\phi\rangle$. In this case, we write $|\psi\rangle \equiv_{phase} |\phi\rangle$.

States of quantum bits are *equivalence classes under* $\equiv_{phase}$. In other words, the state of a quantum bit (qubit) Q is a subset of S_1 such that

- Elements of Q are pariwise related by a global phase: $|\phi\rangle, |\psi\rangle \in Q$ implies that $\exists \theta$ such that $|\psi\rangle = e^{i\theta} |\phi\rangle$;
- Q is maximal: $|\phi\rangle \in Q$ implies that for every angle θ, we have $e^{i\theta} |\phi\rangle \in Q$.

2.2.6 An element $|\psi\rangle \in Q$ is called a *representative element* of Q. By abuse of notation, when talking about a qubit, the equivalence class Q is identified with its representative elements: we say that $|\psi\rangle \in Q$ is the state of a qubit instead of referring to Q.

2.2.7 Quantum register Coming back to the discussion of 2.1.3, we can spell out the 3 abstraction levels for a quantum register holding a qubit:

- At the mathematical level: a normalized vector of $\mathscr{H}$, modulo a global phase. At this level, we only have a mathematical object, there is no information.
- At the computer science level: a way to programmatically refer to it (by some pointer for instance), and the fact that $|0\rangle$ and $|1\rangle$ corresponds to the basis of information.
- At the physics level: the choice of an object and a pair of orthogonal states: photon and polarization, electron and spin, *etc.*

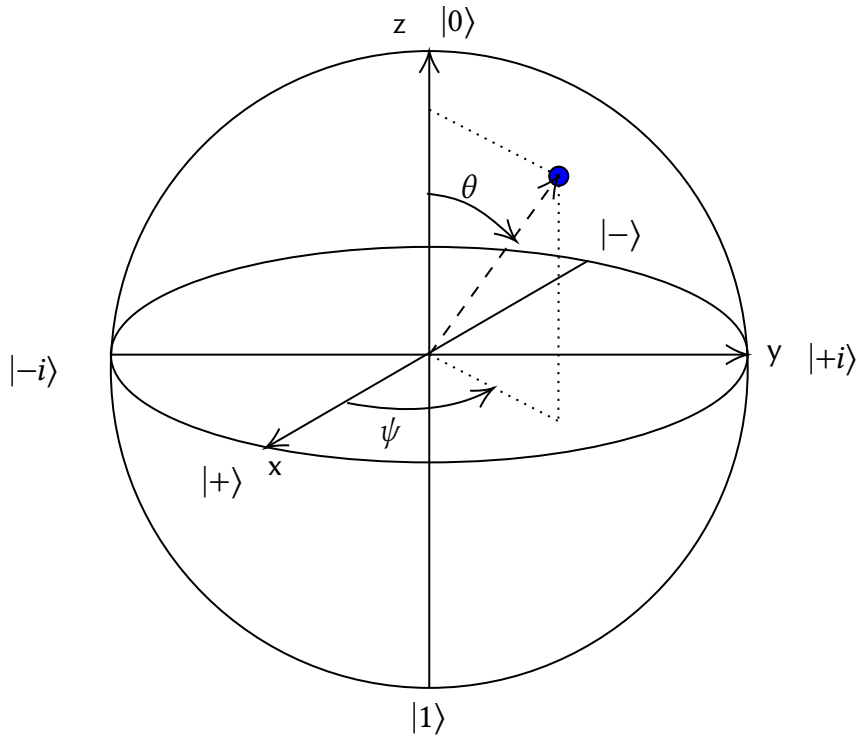


Figure 2: Bloch sphere

2.2.8 Canonical representative. Consider a qubit $\alpha|0\rangle + \beta|1\rangle$. The two complex numbers α and β can be written as $\rho_a e^{i\phi_a}$ and $\rho_b e^{i\phi_b}$ with ρ_a and ρ_b non-negative and such that $\rho_a^2 + \rho_b^2 = 1$. So there exists an angle $\theta \in [0, \pi]$ such that $\rho_a = \cos\left(\frac{\theta}{2}\right)$ and $\rho_b = \sin\left(\frac{\theta}{2}\right)$. Another representative element of the same qubit is $e^{-i\phi_\alpha}(\alpha|0\rangle + \beta|1\rangle)$ which is then $\cos\left(\frac{\theta}{2}\right)|0\rangle + \sin\left(\frac{\theta}{2}\right)e^{i(\phi_\beta - \phi_\alpha)}|1\rangle$. We call this representative element the *canonical representative element*.

2.2.9 Bloch sphere. Without loss of generality, a qubit can therefore be parameterized by two angles $\theta \in [0, \pi]$ and $\psi \in [0, 2\pi)$ as follows:

$$\cos\left(\frac{\theta}{2}\right) \cdot |0\rangle + \sin\left(\frac{\theta}{2}\right) e^{i\psi} \cdot |1\rangle .$$

The angle ψ is called the *phase* of the qubit. This gives a 3-D representation of a qubit on the so-called *Bloch sphere*, shown in Figure 2. Apart from $|0\rangle$ and $|1\rangle$, the canonical representative element is uniquely described by such a pair of angles. (θ, ψ).

2.2.10 On the Bloch sphere, two antipodal points correspond to two orthogonal kets, i.e. to an orthonormal basis. The three axes x, y and z respectively corresponds to the bases $\{|-\rangle, |+\rangle\}$, $\{|-i\rangle, |+i\rangle\}$ and $\{|1\rangle, |0\rangle\}$:

$$|+\rangle \triangleq \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \qquad |-\rangle \triangleq \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle),$$

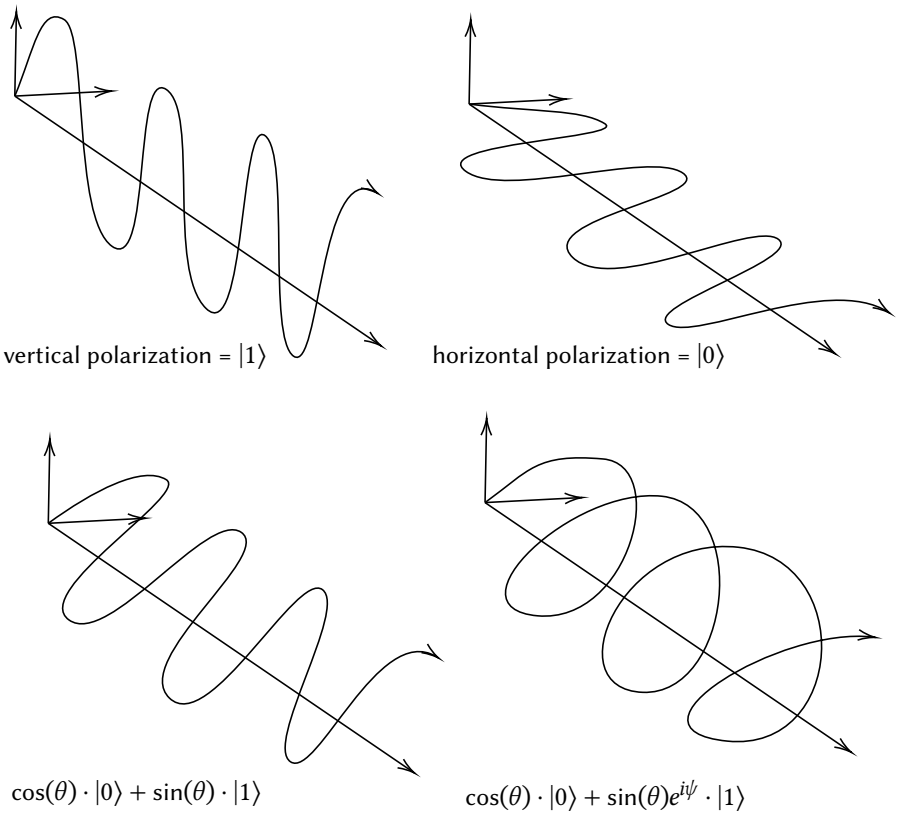


Figure 3: Encoding a qubit on the polarization of a photon.

$$|+i\rangle \triangleq \frac{1}{\sqrt{2}}(|0\rangle + i\cdot|1\rangle), \qquad |-i\rangle \triangleq \frac{1}{\sqrt{2}}(|0\rangle - i\cdot|1\rangle).$$

We find once again one of the orthonormal basis introduced in 1.5.6.

2.2.11 The canonical representation of a qubit using two angles can be visualized with the encoding of qubit as the polarization of a photon. This is presented in Fig. 3. In each case, the direction of the photon is given with the diagonal arrow. The photon is represented as a wave, and the plane of the wave stands for the polarization. The photon encodes $|0\rangle$ if the polarization is horizontal, $|1\rangle$ if it is vertical. It can then be a (real) linear combination of $|0\rangle$ and $|1\rangle$: in this case (shown on the bottom left), the plane is tilted by the corresponding angle. Finally, the photon can also encode a non-zero phase on $|1\rangle$ (bottom right in the Figure): the wave is then helical, and the phase determine how much eccentric it is.

2.3 Several Quantum Bits

2.3.1 A quantum memory usually contains more than one quantum bit: each quantum bit is encoded as a physical object, while its mathematical description relies on $\mathscr{H}$. The quantum memory therefore consists of several such objects. The mathematical model of the *joint quantum system* consisting of all of these objects has a state described by the *Kronecker product* of the individual state spaces: If A and B are two quantum systems

whose state spaces are respectively $\vec{E}$ and $\vec{F}$, the state space of the *joint system* AB is $\mathscr{E} \otimes \mathscr{F}$, defined as in 1.7.1.

2.3.2 Let us recall what happens in the classical case. When considering 2 registers, each holding a bit of information, the *state* of the joint system consisting of the 2 registers is the *product* of the individual state spaces: $\mathbb{B} \times \mathbb{B}$. If the state of the system is $(0, 1)$, we can for instance say that the second register is in state 1. The joint system is always *separable*.

2.3.3 Consider now a 2-qubit registers. This system has a state belonging to the tensor space $\mathscr{H} \otimes \mathscr{H}$. As discussed in 1.7.1, it is defined with the canonical basis elements

$$|00\rangle, |01\rangle, |10\rangle, |11\rangle.$$

The notation is scalable: the canonical basis for $\mathscr{H} \otimes \mathscr{H} \otimes \mathscr{H}$ is

$$|000\rangle, |001\rangle, |010\rangle, |011\rangle, |100\rangle, |101\rangle, |110\rangle, |111\rangle.$$

2.3.4 This notation is versatile. For instance, the following set of ket-vectors defines a basis for $\mathscr{H} \otimes \mathscr{H} \otimes \mathscr{H}$, called the *SHIFT basis*[1]:

$$|000\rangle, |111\rangle, |{+}01\rangle, |{-}01\rangle, |1{+}0\rangle, |1{-}0\rangle, |01{+}\rangle, |01{-}\rangle,$$

where for example $|01+\rangle$ stands for $|0\rangle \otimes |1\rangle \otimes |+\rangle$ as discussed in 1.7.4. The kets $|+\rangle$ and $|-\rangle$ were defined in 1.5.6.

2.3.5 The canonical basis of $\mathscr{H}^{\otimes n}$ (i.e. the tensoring of $\mathscr{H}$ n times) is the set

$$\{\ |b_1 \dots b_n\rangle\ \mid b_i \in \{0, 1\}\ \},$$

the set of all bitstrings of size n. Note how the space $\mathscr{H}^{\otimes n}$ is of dimension 2^n. These bitstrings can be seen as the binary representation of numbers between 0 and $2^n - 1$: when the dimension of the space is clear we will use

$$\sum_{i=0}^{2^n-1} \alpha_i \cdot |i\rangle$$

with $|i\rangle$ understood as the binary representation of i on the correct bitstring size. By convention, in this course the least significant bit is stored on the right: $|011\rangle$ corresponds to $|3\rangle$. Note however that this is completely arbitrary: one could have done the opposite (and QisKit indeed chooses the other—see 4.3.2 for a longer discussion).

[1] https://arxiv.org/abs/2202.00440

2.3.6 Entanglement. In general, a vector of $\mathscr{H} \otimes \mathscr{H}$ is of the form

$$\alpha \cdot |00\rangle + \beta \cdot |01\rangle + \gamma \cdot |10\rangle + \delta \cdot |11\rangle .$$

One can wonder whether all such vector can be written as $|\phi\rangle \otimes |\psi\rangle$, for $|\phi\rangle , |\psi\rangle \in \mathscr{H}$, as in the classical case discussed in 2.3.2: the answer is no.

- If it is possible, the vector is called *separable*;
- If it is not, the vector is called *entangled*.

An example of entangled state is

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle).$$

It is called a *Bell state*, or an *EPR pair*[2].

2.3.7 Bell basis. This state can be extended to a basis of entangled elements: the *Bell basis*. It is defined as

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$
$$|\Phi^-\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle)$$
$$|\Psi^+\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$$
$$|\Psi^-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$$

2.3.8 Following the convention of 1.6.8 we keep the lexicographic order for writing the basis, and this allows us to write kets as column vectors. Given two kets in $\mathscr{H}$ defined as $|\phi_1\rangle = \binom{\alpha_1}{\beta_1}$ and $|\phi_2\rangle = \binom{\alpha_2}{\beta_2}$, the tensor of $|\phi_1\rangle$ with $|\phi_2\rangle$ is the vector

$$\begin{pmatrix} \alpha_1\alpha_2 \\ \alpha_1\beta_2 \\ \beta_1\alpha_2 \\ \beta_1\beta_2 \end{pmatrix} \quad \begin{matrix} \leftarrow |00\rangle \\ \leftarrow |01\rangle \\ \leftarrow |10\rangle \\ \leftarrow |11\rangle \end{matrix}$$

It can be computed in a block-matrix manner as follows:

$$|\phi_1\rangle \otimes |\phi_2\rangle = \begin{pmatrix} \alpha_1 |\phi_2\rangle \\ \beta_1 |\phi_2\rangle \end{pmatrix} = \begin{pmatrix} \alpha_1 \begin{pmatrix} \alpha_2 \\ \beta_2 \end{pmatrix} \\ \beta_1 \begin{pmatrix} \alpha_2 \\ \beta_2 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} \alpha_1\alpha_2 \\ \alpha_1\beta_2 \\ \beta_1\alpha_2 \\ \beta_1\beta_2 \end{pmatrix}$$

[2] Named after Einstein, Podolsky and Rosen.

2.4 The Quantum Circuit Model

2.4.1 In 2.1.3 and 2.2.7, we discussed how registers should be individually addressable, and how one should be able to act on them. In the case of the quantum co-processor, all of this is constrained by what can be realized at the physical level. In general, the physics of the encoding objects allows one to perform two kinds of operations: *unitaries* on the state space: the *internal operations* discussed in 2.1.2, and *measurements*—the reading of the memory. We first focus on the internal operations: the unitaries.

2.4.2 By unitary operation, we literally mean the linear operations that have been described in 1.9.1: Internal actions of the co-processor on the quantum memory are restricted to unitary operations on the state space. Assume that the memory holds n qubits and that U is a unitary on $\mathscr{H}^{\otimes n}$. If the state of the memory is $|\phi\rangle$, applying the "action" of U on the memory has the net effect of deterministicly changing the state of the memory to $U|\phi\rangle$. There are no side-effects in the sense that no classical information is leaked to the classical computer (beside the information that the action has been performed), and in the sense that the state of the memory after the action is uniquely determined by U and by its state before the action.

2.4.3 Of course, in general the co-processor does not have access to arbitrary operations on all of the memory at once. As for a classical machine where only a finite set of instructions is available, the quantum co-processor is limited to a finite set of unitary operations, typically acting on one or two qubits at a time. Such operations are referred to as *quantum gates*: elementary unitary gates that are considered not too costly in general. The notion is flexible, and each algorithm might consider a slightly different set of gates. One of the problem is to conciliate what it can do with what we want to do. 2.5 and 2.6 present standard gate-sets; we focus here on the technique to combine gates together.

2.4.4 Following the discussion of 2.4.2, the sequential action of the unitaries U followed by V then corresponds to the action $V \circ U$, the *composition* of V and U. In this scheme, a timed sequence of actions has the mathematical correspondance of successive composition of operators. With the caveat that "U then V then W" corresponds to $W \circ V \circ U$ (note the reversal in the order).

2.4.5 The action consisting of not doing anything (such as the instruction `no-op` in assembly) is still an action: it corresponds to the *identity* map: if the state of the system is $|\phi\rangle$, it is still $|\phi\rangle$ after doing nothing. This representation plays well with the composition of action presented in 2.4.4: doing nothing followed with U is literally the same thing as doing U, and this is acknowledged with the fact that $U \circ Id = U$.

2.4.6 Consider two quantum systems A and B respectively described by the state spaces $\mathscr{E}$ and $\mathscr{F}$. We discussed in 2.3.1 that the joint system is described by $\mathscr{E} \otimes \mathscr{F}$. Suppose now that we perform a unitary U on A and V on B. The action on the $\mathscr{E} \otimes \mathscr{F}$ is the unitary map $U \otimes V$ defined in 1.8.12. If we only perform U on A (while not touching B), overall action on the global system AB is $A \otimes Id$: the sub-system B is acted upon with

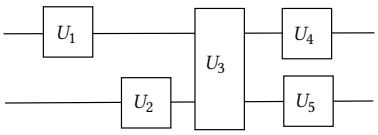


Figure 4: An example of quantum circuit

a trivial action, whose semantics is described in 2.4.5. Thanks to the properties of the tensor product, performing U on A followed by V on B is equivalent to performing first V on B then U on A. Indeed, the former is the operation $(Id \otimes B) \circ (A \otimes Id)$ while the latter is $(A \otimes Id) \circ (Id \otimes B)$. Thanks to 1.8.13, we know that both actions are equal to $A \otimes B$.

2.4.7 A quantum memory consists of separately addressable quantum bits: each quantum bit is a distinct subsystem, of state-space $\mathscr{H}$. The global state-space of the memory is the tensor product of all of these state-spaces. We then have two notion of sequentiality:

- timed sequentiality: as discussed in 2.4.4, this is represented with *composition* of operations.
- spacial separation: as discussed in 2.4.6, this is represented with *tensor* of operations.

Following what can be done for Boolean circuits, we can design a notion of *quantum circuit*.

- Wires are horizontal lines, representing the life-span of one qubit;
- Vertical juxtaposition of wires corresponds to the spacial separation of qubits. By convention, the top wire is the first qubit in the list of qubits, the last wire is the last qubit;
- Boxes on wires corresponds to actions. They can span over several wires to represent action on several qubits;
- Horizontal sequences of boxes correspond to the successive action of boxes.

2.4.8 An example of quantum circuit is shown in Fig. 4. There are two wires. The circuit corresponds to the following sequence of actions: apply U_1 on qubit 1; apply U_2 on qubit 2; apply U_3 on qubits 1 and 2; apply simultaneously U_4 and U_5 on qubit 1 and 2 respectively.

2.4.9 In the circuit of Fig. 4, we decided to place U_1 before U_2, and to place U_4 and U_5 simultaneously. Thanks 2.4.6, we know that this is completely arbitrary: boxes behave as beads on a string, and one can freely "move" boxes along wires. The only limit being that in general, one cannot make them go over one another. In particular, the two circuits presented in Fig. 5 then corresponds to the same action on the quantum memory.

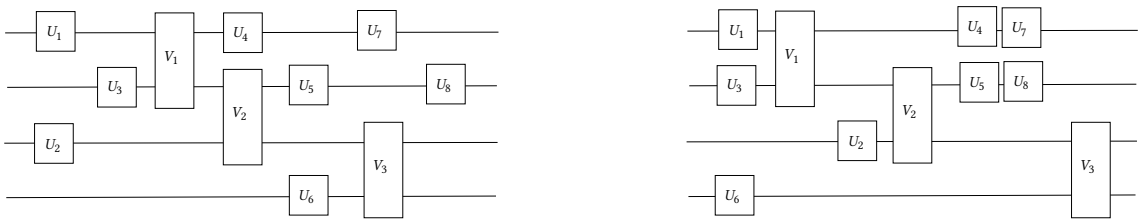


Figure 5: Two equivalent circuits

2.4.10 Suppose that A and B are two linear operations acting on $\mathscr{H}$. Using the lexicographic ordering of the basis, they can be represented with the matrices

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

Then $A \otimes B$ is defined as a block-matrix, in a same way as tensors of vectors in 2.3.8 as follows

$$\begin{aligned} A \otimes B &= \begin{pmatrix} a_{11}B & a_{12}B \\ a_{21}B & a_{22}B \end{pmatrix} \\ &= \begin{pmatrix} a_{11}\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} & a_{12}\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \\ a_{21}\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} & a_{22}\begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} \end{pmatrix} \\ &= \begin{pmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{pmatrix} \end{aligned}$$

The basis ordering is the lexicographic order used in 2.3.3: canonical basis elements are listed as $|00\rangle, |01\rangle, |10\rangle, |11\rangle$.

2.4.11 The order in the tensor product is important: performing $Id \otimes$ HAD or HAD $\otimes$ Id on a 2-qubit system is not the same thing. This is reflected by corresponding matrices, which are respectively

$$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix}, \qquad \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}.$$

2.4.12 Not all 4×4 unitary matrix corresponds to a tensor of two 2×2 unitary matrices. For instance, the operation

$$\text{SWAP} \triangleq \begin{array}{c} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{matrix} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{matrix} \\ |00\rangle\,|01\rangle\,|10\rangle\,|11\rangle \end{array}$$

is unitary, but it cannot be written as $U \otimes V$ with U and V 1-qubit operations. Written as an operation on the canonical basis, corresponds to the function

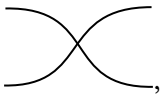
$$|x\rangle \otimes |y\rangle \longmapsto |y\rangle \otimes |x\rangle .$$

We call it the *swap* operation.

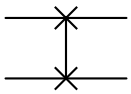
2.4.13 The swap operation has the same effect as literally swapping the position of two qubits. And indeed, one can show that

$$\text{SWAP} \circ (U \otimes Id) = Id \otimes U.$$

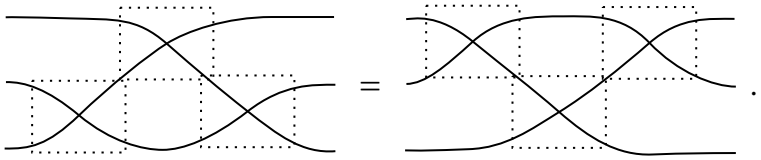
2.4.14 Thanks to 2.4.13, the graphical representation of quantum circuits can be extended with a special representation for the swap operation: the crossing of wires, as follows:



or, more compactly,



The first graphical representation is compatible with a series of equational properties satisfied by quantum circuits equipped with swaps, such as



2.4.15 The equational theory derived from the rules such as the one shown in 2.4.14 gives rise to what is called a PROP[3]. The bottom line is that

> Wires can be shuffled as much as we want, the only constraint is that they need to always stay *from left to right*.

Said otherwise, the spacial ordering of wires is irrelevant.

2.4.16 The fact that one cannot exchange two sequencial actions on the same wire means that there is a notion of *causality* in the sequence of actions. Since the spacial ordering of wires is irrelevant, this causality is somehow the only important information to keep track of. The nice, planar circuit representation contains too much information: we can instead use directed acyclic graphs (DAG) to represent circuit. The circuits of Fig. 5 can then be represented in a unique manner by the DAG shown in Fig 6. If one has to programmatically construct a circuit, a DAG might be more suitable than a sequence of gates. This is for instance one of the possible circuit representation within the Python library QisKit[4].

[3] The canonical reference is a paper from S. Lack, but it is a bit off topic

[4] https://qiskit.org/documentation/stubs/qiskit.dagcircuit.DAGCircuit.html

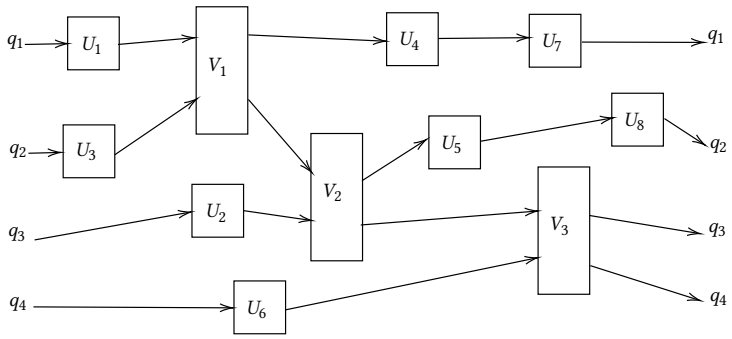


Figure 6: DAG representing a circuit

2.4.17 Consider a circuit C implementing a unitary U: the circuit is a sequence of gates $G_1, G_2, ..., G_k$. Because

$$(AB)^{-1} = B^{-1}A^{-1}, \qquad (A \otimes B)^{-1} = A^{-1} \otimes B^{-1},$$

the circuit C^{-1} defined with the reversed application of the *inverse* of the gates: $G_k^{-1}, ..., G_2^{-1}, G_1^{-1}$ implements U^{-1}. Graphically:

2.5 Quantum Gates on 1 Qubit

2.5.1 Any reasonnable set of quantum gates include the *Pauli* gates. These consists of the three 1-qubit gates

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

together with the identity. Each Pauli gate is equal to its conjugate transpose, and squaring it yields the identity. The X gate is also denoted NOT, since it flips $|0\rangle$ and $|1\rangle$. Its action on canonical basis vectors is

$$X \,:\, |x\rangle \longmapsto |\neg x\rangle\,.$$

It has a special graphical representation in circuit, as follows:

The Z-gate does not touch the canonical basis kets, but it adds a phase to $|1\rangle$:

$$Z \,:\, |x\rangle \longmapsto (-1)^x\,|x\rangle\,.$$

2.5.2 S-gate. Another standard gate is the *S gate*, defined as

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}.$$

It is the square root of Z : $S^2 = Z$.

2.5.3 Hadamard gate. The *Hadamard* presented in 1.8.6 is also amongst the usual quantum gates. It is written H, or HAD. It is its own inverse: $H^2 = Id$.

2.5.4 T-gate. By composing Pauli gates, S-gates and Hadamard gates one can only reach a finite number of unitary matrices. To get an infinite set, we need to add a refined phase-gate: the typical choice is the square root of S, the T-gate:

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}.$$

2.5.5 Theorem (Approximate Universality). Consider a unitary U acting on $\mathscr{H}$, and an error $\varepsilon > 0$. There exists a sequence C of gates H and T such that C implements U up to error ε:

$$\forall\, |\psi\rangle\,, \quad |(U - C)\,|\psi\rangle| \leq \varepsilon \cdot ||\psi\rangle|.$$

Moreover, this sequence of gates is "short" in the sense that there is a constant c such that the size of C is in $O(\log^c(1/\varepsilon))$.

2.5.6 Th. 2.5.5 is a result known since 1995 with the now standard *Solovay-Kitaev algorithm*, with $c = 3.97$, followed in 2002 by an improvement with $c = 3+\delta$, for δ arbitrarily small. Of course, as usual in this situation, the smallest δ is, the biggest the overhead. Since the Solovay-Kitaev algorithm based on a geometric interpretation, the circuit synthesis problem for 1-qubit gates has been greatly improved with algorithms relying on the resolution of Diophantine equations. In the 2010's, several competing groups developped more and more refined versions; the last paper of this academic jousting [RS16] yields an optimal strategy with $c = 1$. From a practical standpoint, it provides a very fast and efficient procedure to approximate Z-rotations, i.e. unitaries of the form

$$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$$

For instance, for $\theta = \pi/128$ and $\varepsilon = 10^{-10}$, the procedure yields in a tenth of a second the approximation

HTSHTSHTSHTHTHTHTSHTHTSHTSHTSHTHTHTSHTSHTHTHTSHTHTSHTHTH
THTHTHTHTSHTSHTSHTHTSHTHTSHTHTHTHTSHTHTHTSHTHTSHTHTHTHTS
HTSHTSHTHTHTSHTSHTSHTSHTHTSHTSHTSHTSHTHTSHTHTSHTSHTHTHTH
THTSHTHTHTHTSHTSHTSHTHTSHTSHTHTHTSHTHTHTHTHTSHTSHTHTHTHT
HTSHTHTHTHTSHTHTHTHTHTHTH.

If these optimized algorithms do not usually bring anything in the relative asymptotic complexity of quantum and classical algorithms, they are crucial when discussing practical implementations.

2.5.7 Rotations If we want to perform an exact *circuit synthesis* of 1-qubit unitaries, one strategy is to include the so-called *rotations gates* $R_X(\theta)$, $R_Y(\theta)$ and $R_Z(\theta)$, parameterized by an angle θ. These rotations respectively corresponds to rotations around the X, Y and Z axis of the Bloch sphere. They are defined as follows: ↳ 2.2.9

$$R_G(\theta) \triangleq e^{i\frac{\theta}{2}\cdot G} = \cos\left(\frac{\theta}{2}\right)\cdot Id - i\sin\left(\frac{\theta}{2}\right)\cdot G,$$

for $G \in \{X, Y, Z\}$. Matrix-wise, they can be written as

$$R_X(\theta) = \begin{pmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{pmatrix}$$
$$R_Y(\theta) = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix}$$
$$R_Z(\theta) = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}$$

2.5.8 The gate $R_Z(\theta)$ is defined with a global phase: it is customary to define an alternative version of the gate:

$$\textsc{Ph}(\theta) \triangleq R_\theta \triangleq \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}.$$

We then have $T = \textsc{Ph}(\pi/4)$, $S = \textsc{Ph}(\pi/2)$ and $Z = \textsc{Ph}(\pi)$.

2.5.9 Theorem (Parametrization of 1-qubit gates). The canonical form of a unitary map U on 1 qubit is parametrized by 3 angles *up to a global phase* as follows:

$$U = \begin{pmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i(\lambda+\phi)}\cos(\theta/2) \end{pmatrix}. \tag{25}$$

Such a map U can be written as a product of rotation gates and a global phase as follows:

$$\begin{pmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i(\lambda+\phi)}\cos(\theta/2) \end{pmatrix} = e^{i(\phi+\lambda)/2} R_Z(\phi) R_Y(\theta) R_Z(\lambda). \tag{26}$$

2.5.10 *Proof.* To prove the decomposition of U into the form of Eq. (25), one can first note that the first column is the parametrization of a qubit state given in 2.2.9. The second column is obtained by specifying the fact that it is supposed to be of norm 1, and orthogonal to the first column.

For the proof of Eq. (26), we can simply unfold the definitions of the rotations, and execute the matrix multiplications as follows.

$$\begin{aligned}
&e^{i(\phi+\lambda)/2} R_Z(\phi) R_Y(\theta) R_Z(\lambda) \\
&= e^{i(\phi+\lambda)/2} \begin{pmatrix} e^{-i\phi/2} & 0 \\ 0 & e^{i\phi/2} \end{pmatrix} \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix} \begin{pmatrix} e^{-i\lambda/2} & 0 \\ 0 & e^{i\lambda/2} \end{pmatrix} \\
&= e^{i(\phi+\lambda)/2} \begin{pmatrix} e^{-i\phi/2} & 0 \\ 0 & e^{i\phi/2} \end{pmatrix} \begin{pmatrix} e^{-i\lambda/2}\cos(\theta/2) & -e^{i\lambda/2}\sin(\theta/2) \\ e^{-i\lambda/2}\sin(\theta/2) & e^{i\lambda/2}\cos(\theta/2) \end{pmatrix} \\
&= e^{i(\phi+\lambda)/2} \begin{pmatrix} e^{-i(\lambda+\phi)/2}\cos(\theta/2) & -e^{i(\lambda-\phi)/2}\sin(\theta/2) \\ e^{i(\phi-\lambda)/2}\sin(\theta/2) & e^{i(\lambda+\phi)/2}\cos(\theta/2) \end{pmatrix} \\
&= \begin{pmatrix} \cos(\theta/2) & -e^{i\lambda}\sin(\theta/2) \\ e^{i\phi}\sin(\theta/2) & e^{i(\lambda+\phi)}\cos(\theta/2) \end{pmatrix},
\end{aligned}$$

which is the definition of U. □

2.5.11 The global phase in Th. 2.5.9 is there to make sure that "things fall in the right place". From a computational point of view, it is invisible since quantum states are considered modulo global phases. Note however that global phases in unitaries are important to keep in mind when considering controlled gates (see 2.6.11 for a discussion).

2.6 Quantum Gates on Several Qubits

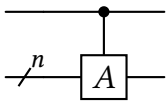
2.6.1 One-qubit operations are limited in the sense that from a basis ket they do not make it possible to realize entangled states such as the EPR state. Although the SWAP operation is a 2-qubit operations, it preserves separability: to get all possible states from a basis state we need more. A strategy to get there—and to get more expressivity in general— is to use *controlled operations*. ↳ 2.3.6 ↳ 2.4.13

2.6.2 **Notation** To represent a bundle of n wires, we shall use the notation

2.6.3 If A is a unitary on n qubits, one can build a new unitary map called *controlled operation* as follows. Pick $A : \mathscr{H}^{\otimes n} \to \mathscr{H}^{\otimes n}$, and build the unitary $C\text{-}A : \mathscr{H} \otimes \mathscr{H}^{\otimes n} \to \mathscr{H} \otimes \mathscr{H}^{\otimes n}$ by adding a *control qubit*:

$$C\text{-}A : \begin{cases} |0\rangle \otimes |x\rangle \mapsto |0\rangle \otimes |x\rangle \\ |1\rangle \otimes |x\rangle \mapsto |1\rangle \otimes (A \cdot |x\rangle)\,. \end{cases}$$

The map $C\text{-}A$ admits a graphical notation as follows:



The bullet is placed on the control qubit. The vertical line is not a wire: it is there to indicate which gate is being controlled by which wire.

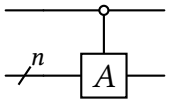
2.6.4 The matrix corresponding to $C\text{-}A$ is defined blockwise as follows:

$$\begin{array}{c} \begin{pmatrix} Id & 0 \\ 0 & A \end{pmatrix} \\ |0x\rangle\,|1x\rangle \end{array} \quad \begin{array}{c} |0x\rangle \\ |1x\rangle \end{array} .$$

It preserves the two orthogonal subspaces $|0\rangle \otimes \mathscr{H}^{\otimes n}$ and $|1\rangle \otimes \mathscr{H}^{\otimes n}$: it does nothing on the former while applying A on the latter.

2.6.5 The control operation is compositional: $C\text{-}(A \circ B)$ is $(C\text{-}A) \circ (C\text{-}B)$, and $C\text{-}I = I$.

2.6.6 It is useful to be able to perform *negative controls*: controls where the action is triggered in $|0\rangle$ and not on $|1\rangle$. The graphical notation is



The operation is

$$|0\rangle \otimes |x\rangle \mapsto |0\rangle \otimes (A \cdot |x\rangle)$$
$$|1\rangle \otimes |x\rangle \mapsto |1\rangle \otimes |x\rangle .$$

2.6.7 The gate CNOT, or C-X acts on 2 qubits as follows:

$$\text{CNOT} : \begin{cases} |0\rangle \otimes |x\rangle \mapsto |0\rangle \otimes |x\rangle \\ |1\rangle \otimes |x\rangle \mapsto |1\rangle \otimes |\neg x\rangle \end{cases} .$$

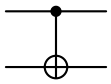
This can be written in a more compact way as

$$\text{CNOT} : \quad |x\rangle \otimes |y\rangle \mapsto |x\rangle \otimes |y \oplus x\rangle .$$

The matrix of the CNOT gate is

$$\begin{array}{c} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \\ |00\rangle\,|01\rangle\,|10\rangle\,|11\rangle \end{array} \quad \begin{array}{c} |00\rangle \\ |01\rangle \\ |10\rangle \\ |11\rangle \end{array}$$

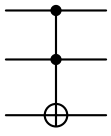
Its graphical representation in a circuit is



2.6.8 One can also build the gate C-C-X, also known as the *Toffoli* gate:

$$|xy\rangle \otimes |z\rangle \mapsto |xy\rangle \otimes |z \oplus xy\rangle .$$

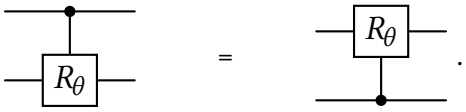
As a circuit, it is



2.6.9 The control of a gate R_θ has an internal symmetry: C-R_θ sends every canonical basis vector to itself, with a phase $e^{i\theta}$ in the case $|11\rangle$. The matrix is ↳ 2.5.8

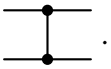
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\theta} \end{pmatrix},$$

and



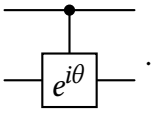
It sends $|x\rangle \otimes |y\rangle$ to $(e^{i\theta})^{xy} \cdot |x\rangle \otimes |y\rangle$.

2.6.10 A special case if the gate C-Z, a controlled rotation of angle π. It has the special diagrammatic notation



It sends $|x\rangle \otimes |y\rangle$ to $(-1)^{xy} \cdot |x\rangle \otimes |y\rangle$.

2.6.11 Control and global phases In 2.5.11, we mentionned that global phases are irrelevant for unitaries when applied on the whole state space. When the unitary correspond to a subcircuit that might be controlled, the global phase is important. Indeed, consider the circuit



where the controlled gate applies a global phase to a qubit: $|x\rangle \mapsto e^{i\theta} |x\rangle$. The circuit performs the operation:

$$|0\rangle |x\rangle \mapsto |0\rangle |x\rangle ,$$
$$|1\rangle |x\rangle \mapsto e^{i\theta} |1\rangle |x\rangle .$$

That is, the global phase is only applied when the top qubit is in state $|1\rangle$. The action of this circuit is not equivalent to the identity: it is equivalent to a phase-gate on the top qubit.

We shall come back to this point when controlling arbitrary unitaries in Th. 3.3.2 and 3.3.4.

2.7 Creating New Quantum Registers

2.7.1 In 2.1.2, we discussed how the co-processor implements three classes of actions. So far, we discussed the internal actions, mathematically represented with unitary operators. In this section, we shall discuss the two other classes: allocation and initialization, and deallocation and reading.

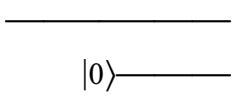
2.7.2 Initialization is the easiest to handle: if the memory has say 1 qubit in state $|\phi\rangle$, adding a new qubit in state $|0\rangle$ corresponds to changing the memory state to $|\phi\rangle \otimes |0\rangle$. Before the action, the memory state space was $\mathscr{H}$, and after the action it is $\mathscr{H} \otimes \mathscr{H}$. Note that we arbitrarily decided to add the new qubit at the end, but we could have done otherwise.

2.7.3 Initialization as in 2.7.2 can be seen as a linear operation:

$$\begin{aligned} \mathscr{H} &\to \mathscr{H} \otimes \mathscr{H} \\ |x\rangle &\mapsto |x\rangle \otimes |0\rangle \,. \end{aligned}$$

This map is clearly not unitary since it is not total: vectors of the form $|\phi\rangle \otimes |1\rangle$ are not in the image. However, it still preserves norm and orthogonality, and if we consider it as a map from $\mathscr{H}$ to the *subspace* $\mathscr{H} \otimes |0\rangle$, it is unitary.

2.7.4 We say that we initialize *auxiliary qubits*, or *ancillas*. The circuit representation for ancillas is as follows



This represents the operation of 2.7.3.

2.8 Reading Quantum Registers

2.8.1 Measurement. The last class of action is concerned with "poking the existing memory state": either by de-allocating part of it, or simply reading it. Any such operation leaves the realm of unitary maps and need the notion of *measurement*. It is moreover the *only* way to get back classical data out of quantum data.

2.8.2 The measure of a qubit state $\alpha \cdot |0\rangle + \beta \cdot |1\rangle$, we obtain

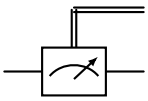
- with prob. $|\alpha|^2$: the Boolean value "0", and the qubit is now in state $|0\rangle$
- with prob. $|\beta|^2$: the Boolean value "1", and the qubit is now in state $|1\rangle$

The qubit state has been probabilistically *projected* on one of the canonical basis vector. As vectors are normalized, the sum of probabilities is indeed equal to 1: we are sure to get a result.

2.8.3 Note that measuring $|0\rangle$ returns "0" with probability 1. Therefore, measuring twice the same qubit returns twice the same result.

2.8.4 Since the state of a quantum bit is collapsed by the measurement, we can consider that the qubit "disappeared" during the process: a so-called *destructive measure*. In this case, we can consider that the qubit is turned into a bit. The circuit notation is extended with a special gate for the measurement, and a special wire type for bit, represented with a double line. Destructive measurements are represented with

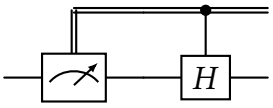
while non-destructive are



Here we consider that the gate "spits out" a bit on top. After the measurement gate, the qubit is either $|0\rangle$ or $|1\rangle$ depending on the measured Boolean value. We can control on bit wire: it means that the controlled gate is applied provided that the bit is in state 1 for black bullet and in state 0 for white bullets.

2.8.5 Beware A circuit with a measurement is in general *not* invertible, and the procedure presented in 2.4.17 makes in general no sense with measurements.

2.8.6 For instance,



implements the operation sending $\alpha|0\rangle + \beta|1\rangle$ to $|0\rangle$ with probability $|\alpha|^2$ and $|-\rangle$ with probability $|\beta|^2$. We know which state we are left with by reading the bit wire.

2.8.7 A qubit is never alone: it is part of a larger memory. The behavior in this general situation is very similar to what happens for 1 qubit: the state of the system is *projected*. ↳ 2.8.2
More precisely, suppose that the memory consists of $n+1$ qubits and that we measure the first qubit of the memory. The state space of the memory is $\mathscr{H} \otimes \mathscr{H}^{\otimes n}$. The state of the memory can then be written as

$$|\phi\rangle = \sum_{i=0}^{2^n-1} \left(\alpha_{0,i} \cdot |0\rangle \otimes |i\rangle + \beta_{1,i} \cdot |1\rangle \otimes |i\rangle\right). \tag{27}$$

Measuring the first qubit projects the state $|\phi\rangle$ to one of the two orthogonal subspaces $|0\rangle \otimes \mathscr{H}$ and $|1\rangle \otimes \mathscr{H}$. These subspaces respectively contains all of the vectors of the forms

$$\begin{aligned} |0\rangle \otimes \mathscr{H} &\triangleq \left\{ \sum_{i=0}^{2^n-1} \beta_i \cdot |0\rangle \otimes |i\rangle \right\}, \\ |1\rangle \otimes \mathscr{H} &\triangleq \left\{ \sum_{i=0}^{2^n-1} \gamma_i \cdot |1\rangle \otimes |i\rangle \right\}. \end{aligned}$$

Said otherwise, the vectors of $|0\rangle \otimes \mathscr{H}$ only have canonical basis kets starting with 0, and the vectors of $|1\rangle \otimes \mathscr{H}$ only have canonical basis kets starting with 1.

2.8.8 Measuring the first qubit of the quantum memory, the vector $|\phi\rangle$ in Eq. (27) is then collapsed into either

$$|\phi_0\rangle \quad = \quad \sum_{i=0}^{2^n-1} \frac{\alpha_{0,i}}{\rho_0} \cdot |0\rangle \otimes |i\rangle \quad = \quad |0\rangle \otimes \left(\sum_{i=0}^{2^n-1} \frac{\alpha_{0,i}}{\rho_0} \cdot |i\rangle \right)$$

or

$$|\phi_1\rangle \quad = \quad \sum_{i=0}^{2^n-1} \frac{\beta_{1,i}}{\rho_1} \cdot |1\rangle \otimes |i\rangle \quad = \quad |1\rangle \otimes \left(\sum_{i=0}^{2^n-1} \frac{\beta_{1,i}}{\rho_1} \cdot |i\rangle \right)$$

with

$$\rho_0 = \sqrt{\sum_{i=0}^{2^n-1} |\alpha_{0,i}|^2}, \qquad \rho_1 = \sqrt{\sum_{i=0}^{2^n-1} |\alpha_{1,i}|^2}.$$

The collapse happens *modulo renormalization*: the adjunction of ρ_0 and ρ_1 ensures that $|\phi_0\rangle$ and $|\phi_1\rangle$ are unit vectors. We can separate the state of the first qubit, and if we define

$$|\psi_0\rangle = \sum_{i=0}^{2^n-1} \frac{\alpha_{0,i}}{\rho_0} \cdot |i\rangle\,, \qquad |\psi_1\rangle = \sum_{i=0}^{2^n-1} \frac{\beta_{1,i}}{\rho_1} \cdot |i\rangle\,,$$

they are also normalized vectors, and

$$|\phi\rangle = \rho_0 \cdot |0\rangle \otimes |\psi_0\rangle + \rho_1 \cdot |1\rangle \otimes |\psi_1\rangle\,. \tag{28}$$

2.8.9 Having set up the framework in 2.8.8, we can now use the decomposition of Eq. 28 to describe the process of the measure of the first qubit:

- With probability ρ_0^2, the measurement returns the bit 0 and the memory state collapses to $|0\rangle \otimes |\psi_0\rangle$;
- With probability ρ_1^2, the measurement returns the bit 1 and the memory state collapses to $|1\rangle \otimes |\psi_1\rangle$.

In particular, if we perform a second measure of the same qubit, as for the case of 2.8.2 we get the same output bit with probability 1.

2.8.10 Let us consider the 2-qubit case, and measure the first qubit in

$$|\phi\rangle = \alpha \cdot |00\rangle + \beta \cdot |01\rangle + \gamma \cdot |10\rangle + \delta \cdot |11\rangle\,.$$

The projection on the subspace $|1\rangle \otimes \mathscr{H}$ is

$$\alpha \cdot |00\rangle + \beta \cdot |01\rangle\,,$$

and the one on subspace $|1\rangle \otimes \mathscr{H}$ is

$$\gamma \cdot |10\rangle + \delta \cdot |11\rangle\,.$$

If we write $\rho_0 = \sqrt{|\alpha|^2 + |\beta|^2}$ and $\rho_1 = \sqrt{|\gamma|^2 + |\delta|^2}$, the decomposition of Eq. (28) gives

$$|\phi\rangle = \rho_0 \cdot |0\rangle \otimes \left(\frac{\alpha}{\rho_0} \cdot |0\rangle + \frac{\beta}{\rho_0} \cdot |1\rangle\right) + \rho_1 \cdot |1\rangle \otimes \left(\frac{\gamma}{\rho_1} \cdot |0\rangle + \frac{\delta}{\rho_1} \cdot |1\rangle\right).$$

We can check that we are in the situation of 2.8.8: $|\rho_0|^2 + |\rho_0|^2 = 1$, and each component is normalized. According to 2.8.9, by measuring the first qubit we then obtain

- the bit 0 with probability $|\rho_0|^2 = |\alpha|^2 + |\beta|^2$, and the state of the system is now
$$|0\rangle \otimes \left(\frac{\alpha}{\rho_0} \cdot |0\rangle + \frac{\beta}{\rho_0} \cdot |1\rangle\,;\right)$$
- the bit 1 with probability $|\rho_1|^2 = |\gamma|^2 + |\delta|^2$, and the state of the system is now
$$|1\rangle \otimes \left(\frac{\gamma}{\rho_1} \cdot |0\rangle + \frac{\delta}{\rho_1} \cdot |1\rangle\right).$$

2.8.11 We can then proceed and measure the second qubit. If we had measured 0 for the first qubit, the state is

$$|0\rangle \otimes \left(\frac{\alpha}{\rho_0} \cdot |0\rangle + \frac{\beta}{\rho_0} \cdot |1\rangle\,;\right),$$

and measuring the second qubit yields

- the bit 0 with probability $\left|\frac{\alpha}{\rho_0}\right|^2$, and the state of the system is now $|00\rangle$;
- the bit 1 with probability $\left|\frac{\beta}{\rho_0}\right|^2$, and the state of the system is now $|01\rangle$.

If we had instead measured 1 for the first qubit, the state is

$$|1\rangle \otimes \left(\frac{\gamma}{\rho_1} \cdot |0\rangle + \frac{\delta}{\rho_1} \cdot |1\rangle\right).$$

and measuring the second qubit yields

- the bit 0 with probability $\left|\frac{\gamma}{\rho_1}\right|^2$, and the state of the system is now $|10\rangle$;
- the bit 1 with probability $\left|\frac{\delta}{\rho_0}\right|^2$, and the state of the system is now $|11\rangle$.

2.8.12 Combining the sequence of measurements, when a 2-qubit system is in state

$$|\phi\rangle = \alpha \cdot |00\rangle + \beta \cdot |01\rangle + \gamma \cdot |10\rangle + \delta \cdot |11\rangle\,,$$

measuring the first then the second qubit yields the values

- "00" with the state now at $|00\rangle$ with probability $|\alpha|^2$;
- "01" with the state now at $|01\rangle$ with probability $|\beta|^2$;
- "10" with the state now at $|10\rangle$ with probability $|\gamma|^2$;
- "11" with the state now at $|11\rangle$ with probability $|\delta|^2$.

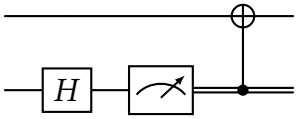
2.8.13 Note one can measure qubits in an arbitrary order, this does not change the final result.

2.8.14 In general, if we measure the whole state of a memory of n qubits:

$$\sum_{i=0}^{2^n-1} \alpha_i \cdot |i\rangle\,,$$

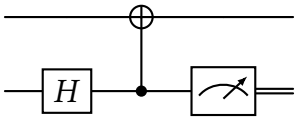
we retrieve the bitstring corresponding to i (as discussed in 2.3.5) with probability $|\alpha_i|^2$. In this case, the memory state is collapsed to the canonical basis ket $|i\rangle$.

2.8.15 Deferred measurements. The *deferred measurement* principle states that one can always "push" the meassurements at the end of a circuit without changing its overall behavior. The trick is to replace classically controlled gates with quantum controlled gates. For instance:



(29)

has the same effect as



(30)

2.8.16 Example. Applied to the state $|00\rangle$, the circuit (29) has the following effect:

$$\begin{aligned}
|00\rangle &\mapsto \frac{1}{\sqrt{2}}|0\rangle \otimes (|0\rangle + |1\rangle) && \text{applying HAD}\\
&= \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle)\\
&\mapsto \begin{cases} (|0\rangle\,, \text{bit } 0) & \text{with prob. } 1/2\\ (|0\rangle\,, \text{bit } 1) & \text{with prob. } 1/2 \end{cases} && \text{measuring}\\
&\mapsto \begin{cases} (|0\rangle\,, \text{bit } 0) & \text{with prob. } 1/2\\ (|1\rangle\,, \text{bit } 1) & \text{with prob. } 1/2 \end{cases} && \text{applying CNOT}
\end{aligned}$$

Thus, overall, the qubit in in state $|0\rangle$ or $|1\rangle$ with probability $\frac{1}{2}$. The bit wire stores the value of the qubit wire.

The circuit (30) has instead the effect:

$$\begin{aligned}
|00\rangle &\mapsto \frac{1}{\sqrt{2}}|0\rangle \otimes (|0\rangle + |1\rangle) && \text{applying HAD}\\
&= \frac{1}{\sqrt{2}}(|00\rangle + |01\rangle)\\
&\mapsto \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) && \text{applying CNOT}
\end{aligned}$$

$$\mapsto \begin{cases} (|0\rangle, \text{bit } 0) & \text{with prob. } 1/2 \\ (|1\rangle, \text{bit } 1) & \text{with prob. } 1/2 \end{cases} \qquad \text{measuring.}$$

We therefore get the same effect as the previous circuit.

2.8.17 Measuring in other bases The measurement presented in 2.8.2 "tests" a qubit against the basis states $|0\rangle$ and $|1\rangle$. It can be spelled out with the scalar product: the probability of getting $|0\rangle$ while measuring $|\phi\rangle$ is

$$|\langle 0 \,|\, \phi\rangle|^2.$$

This turns out to be arbitrary, and it can in fact be done in any basis.[5] For instance, we can measure a qubit against the basis $\{|+\rangle, |-\rangle\}$: we get $|+\rangle$ with probability

$$|\langle + \,|\, \phi\rangle|^2. \tag{31}$$

Of course, the bit returned by the measurement is not defined *a priori* and this has to be specified in advanced.

2.8.18 One can always reduce the problem of the general measurement presented in 2.8.17 to the measure in the canonical basis. Consider the probability of Eq. (31): using 1.9.2 we can reduce it as follows:

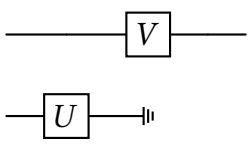
$$|\langle + \,|\, \phi\rangle|^2 = |\langle \textsc{Had}+ \,|\, \textsc{Had}\phi\rangle|^2 = |\langle 0 \,|\, \textsc{Had}\phi\rangle|^2.$$

Measuring in the basis $|+\rangle, |-\rangle$ can therefore be obtained by applying a Hadamard gate followed with a measurement in the standard basis.

2.8.19 By abuse of language we say that we measure in the basis Had, referring to the fact that the unitary can be regarded as an encoding of the basis (as in Eq (21) in 1.8.6).

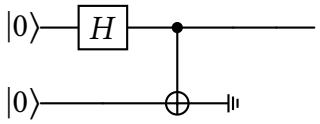
2.9 Discarding Quantum Registers

2.9.1 In regular programming languages, it makes sense to allocate and free (or *discard*) memory on the fly. With quantum registers, discarding amounts to perform a *destructive measure* and forget the result of the measurement. Discarding is represented in circuits with a small symbol reminiscent of "ground". In the following example, we discard the second qubit after the application of a gate U:



[5] As a matter of fact, it can be formalized in the more general framework of POVM but this is out of the scope of this document.

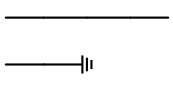
2.9.2 In general, discarding a qubit generates a probabilistic distribution of states. For instance, the circuit



returns a memory with only one qubit (the upper wire), whose state is $|0\rangle$ or $|1\rangle$ with probability $1/2$.

2.9.3 Safely discarding a wire requires to make sure that it is *not entangled* with the rest of the memory. As an example, one can make sure that it is in a canonical basis state. Indeed, if we were to discard the first qubit, the measurement process behind the discard operation would project the state of the memory on the subspaces $|0\rangle \otimes \mathscr{H}^{\otimes n}$ or $|1\rangle \otimes \mathscr{H}^{\otimes n}$. If the first qubit is separated from the rest of the memory, in state $|0\rangle$ or $|1\rangle$, the projection does not do anything beside focusing on the corresponding subspace: there is no loss of information.

2.9.4 For instance, suppose that the input of the circuit



in in state

$$\sum_x \alpha_x |x\rangle \otimes |0\rangle \quad = \quad \left(\sum_x \alpha_x |x\rangle\right) \otimes |0\rangle\,,$$

then the output is

$$\sum_x \alpha_x |x\rangle\,.$$

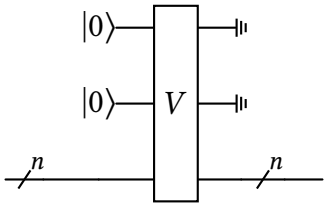
2.9.5 The canonical use-case for discard is for managing ancillas. A frequent situation is when a unitary operation U on n qubits might be cumbersome to implement with exactly n wires, but easier if we allocare more "room", that is, more wires. The operation U is then seen as a block of a larger unitary V acting on $\mathscr{H}^{\otimes k} \otimes \mathscr{H}^{\otimes n}$. Assume $k = 2$ for simplicity. Then

$$V = \begin{pmatrix} U & 0 & 0 & 0 \\ 0 & W_{01,01} & W_{01,10} & W_{01,11} \\ 0 & W_{10,01} & W_{10,10} & W_{10,11} \\ 0 & W_{11,01} & W_{11,10} & W_{11,11} \end{pmatrix} \tag{32}$$

for matrices $W_{xy,zt}$ acting on $\mathscr{H}^{\otimes n}$. In the block decomposition of V, each column corresponds to a possible value for the two auxilary qubits: $|00\rangle$ corresponds to the column with U, and the other columns are associated with $|01\rangle$, $|10\rangle$ and $|11\rangle$. Said otherwise, the action of V on a canonical basis vector is

$$V : \begin{cases} |00\rangle \otimes |x\rangle \mapsto |0\cdots 0\rangle \otimes (U|x\rangle) \\ |yz\rangle \otimes |x\rangle \mapsto |01\rangle \otimes (W_{01,yz}|x\rangle) + |10\rangle \otimes (W_{10,yz}|x\rangle) + |11\rangle \otimes (W_{11,yz}|x\rangle) \end{cases}$$

when $yz \neq 00$. The circuit for U is then



It is design so that the only column used in Eq. (32) is the most-left one, corresponding to the case where the ancillas are $|00\rangle$. The action of V then return something of the form $|00\rangle \otimes \cdots$, therefore making the ancillas qubits separated from the environment: we can discard them without perturbating the global state of the memory. Globally, we recover the action of U, as desired.

2.9.6 The circuit presented in 2.9.5 can be inverted using the technique of 2.4.17. Indeed, the inverse of V is blockwise:

$$V^{-1} = \begin{pmatrix} U^{-1} & 0 \\ 0 & W^{-1} \end{pmatrix}$$

where W is the block made out of the $W_{xy,zt}$'s. The ancillas are kept initialized at 0.

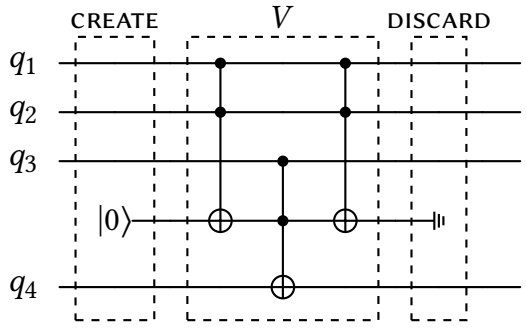
2.9.7 If V were to set the ancilla back to some canonical basis state other than $|0\rangle$, the procedure of 2.9.5 would still work: we could still measure without perturbating the global state. The inverse discussed in 2.9.6 would still be valid, modulo the fact that ancillas would have to be set to the correct, non-$ket0$ value.

2.9.8 A standard case for an operation U described using a larger matrix V as in 2.9.5 is when the matrix U performs an action on the basis vectors, made of several subcomputations. We can store the subcomputations in ancilas qubits, building the final result step by step. Once it is done, we then uncompute the ancillas by reversing the local operations.

As an example, consider the gate C-C-C-X, a flip gate controlled by 3 qubits. On canonical basis vectors, it performs the operation

$$|x\rangle \otimes |y\rangle \otimes |z\rangle \otimes |t\rangle \longmapsto |x\rangle \otimes |y\rangle \otimes |z\rangle \otimes |z \oplus xyz\rangle\,.$$

This operation can be decomposed into two conjunctions: first x with y, then xy with z. Each conjunction can be realized with a Toffoli gate, and we can store the result of the first conjunction inside an ancilla. The circuit is as follows.



It acts on 3 qubits, therefore on the space $\mathscr{H}^{\otimes 3} \otimes \mathscr{H}$, and consists of three parts:

$$\begin{aligned} \textsc{create} &: \mathscr{H}^{\otimes 3} \otimes \mathscr{H} \to \mathscr{H}^{\otimes 3} \otimes |0\rangle \otimes \mathscr{H}, \\ V &: \mathscr{H}^{\otimes 3} \otimes \mathscr{H} \otimes \mathscr{H} \to \mathscr{H}^{\otimes 3} \otimes \mathscr{H} \otimes \mathscr{H}, \\ \textsc{discard} &: \mathscr{H}^{\otimes 3} \otimes \mathscr{H} \otimes \mathscr{H} \to \mathscr{H}^{\otimes 3} \otimes \mathscr{H}. \end{aligned}$$

The operations CREATE and V composes since

$$\mathscr{H}^{\otimes 3} \otimes |0\rangle \otimes \mathscr{H} \subseteq (\mathscr{H}^{\otimes 3} \otimes |0\rangle \otimes \mathscr{H}) \oplus (\mathscr{H}^{\otimes 3} \otimes |1\rangle \otimes \mathscr{H}) = \mathscr{H}^{\otimes 3} \otimes \mathscr{H} \otimes \mathscr{H}.$$

In general, the behavior of DISCARD is *probabilistic*, as the 3rd qubit is measure. However, the state is not completely general: it is resulting from the function $V \circ$ CREATE; let us compute its action on a *basis canonical state*:

$$\begin{aligned} |x\rangle|y\rangle|z\rangle \otimes |t\rangle &\mapsto |x\rangle|y\rangle|z\rangle \otimes |0\rangle \otimes |t\rangle && \text{by CREATE} \\ &\mapsto |x\rangle|y\rangle|z\rangle \otimes |0 \oplus xy\rangle \otimes |t\rangle && \text{by 1st Toffoli} \\ &= |x\rangle|y\rangle|z\rangle \otimes |xy\rangle \otimes |t\rangle \\ &\mapsto |x\rangle|y\rangle|z\rangle \otimes |xy\rangle \otimes |z \oplus xyz\rangle && \text{by 2nd Toffoli} \\ &\mapsto |x\rangle|y\rangle|z\rangle \otimes |xy \oplus xy\rangle \otimes |z \oplus xyz\rangle && \text{by 3rd Toffoli} \\ &= |x\rangle|y\rangle|z\rangle \otimes |0\rangle \otimes |t \oplus xyz\rangle. \end{aligned}$$

For compactness, we omitted some of the tensors. This maps therefore only populates a subspace of its co-domain:

$$V \circ \textsc{create} : \mathscr{H}^{\otimes 3} \otimes \mathscr{H} \to \mathscr{H}^{\otimes 3} \otimes |0\rangle \otimes \mathscr{H}.$$

Whatever input $|\phi\rangle \in \mathscr{H}^{\otimes 3} \otimes \mathscr{H}$

$$|\phi\rangle = \sum_{x,y,z,t \in \{0,1\}} \alpha_{x,y,z,t} |x\rangle|y\rangle|z\rangle \otimes |t\rangle,$$

the resulting state lives in $\mathscr{H}^{\otimes 3} \otimes |0\rangle \otimes \mathscr{H}$:

$$(V \circ \textsc{create})|\phi\rangle = \sum_{x,y,z,t \in \{0,1\}} \alpha_{x,y,z,t} |x\rangle|y\rangle|z\rangle \otimes |0\rangle \otimes |t \oplus xyz\rangle.$$

If we perform a destructive measurement, the projection will have no effect, and we retrieve the vector

$$\sum_{x,y,z,t \in \{0,1\}} \alpha_{x,y,z,t} |x\rangle|y\rangle|z\rangle \otimes |t \oplus xyz\rangle.$$

2.9.9 Although the ancilla is not written in the same place, the circuit of 2.9.8 follows the specification given in 2.9.5: although we allocate and discard auxiliary registers, it corresponds to a unitary operation.

2.10 Cloning, Copy, Teleportation

2.10.1 Quantum information cannot be cloned: it is the so-called *no-cloning theorem*. More precisely, one cannot physically realize the map

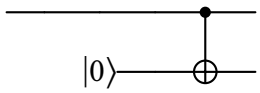
$$|\phi\rangle \otimes |0\rangle \longmapsto |\phi\rangle \otimes |\phi\rangle$$

what would work for *all* ket vectors. One way to understand it is by writing the ket $|\phi\rangle$ as a column vector: the map should therefore send

$$\begin{pmatrix} \alpha \\ 0 \\ \beta \\ 0 \end{pmatrix} \longmapsto \begin{pmatrix} \alpha^2 \\ \alpha\beta \\ \alpha\beta \\ \beta^2 \end{pmatrix}.$$

This operation is not linear: it cannot be synthetized with unitary maps...

2.10.2 If cloning is not possible, *copying* is possible. Consider the circuit



It sends a basis element $|x\rangle$ to $|xx\rangle$. This is clearly not the same map as in 2.10.1: it sends

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix} \longmapsto \begin{pmatrix} \alpha \\ 0 \\ 0 \\ \beta \end{pmatrix}.$$

It is a linear map: in ket-notation, it does

$$\alpha\,|0\rangle + \beta\,|1\rangle \longmapsto \alpha\,|00\rangle + \beta\,|11\rangle\,.$$

2.10.3 Teleportation In 2.8.2 we said that because of the projection that happens, some quantum information is lost when we perform a measurement. In particular, when we measure the qubit $\alpha\,|0\rangle + \beta\,|1\rangle$, the coefficients α and β disappear. This turns out to not be the case in general, and a non-intuitive effect can happen in the circuit presented in Fig. 7.

2.10.4 The circuit is typically read as follows. In the left dashed boxed, an entangled pair of qubits is built. The first qubit is sent to Alice, the other one to Bob. Alice has another qubit (the top wire) whose state she want to send to Bob. Unfortunately, they only share a classical communication channel! Thankfully, they can succeed: Alice performs the middle dashed box with the measurement of both of her qubits, she sends the result to Bob, and Bob uses the Boolean values he receives to execute a *correction* on his qubit: as we shall see, the qubit is now in the same state Alice's qubit was originally in. However, note that we did not clone anything: Alice's qubit has been measured, and his state is not a canonical basis element.

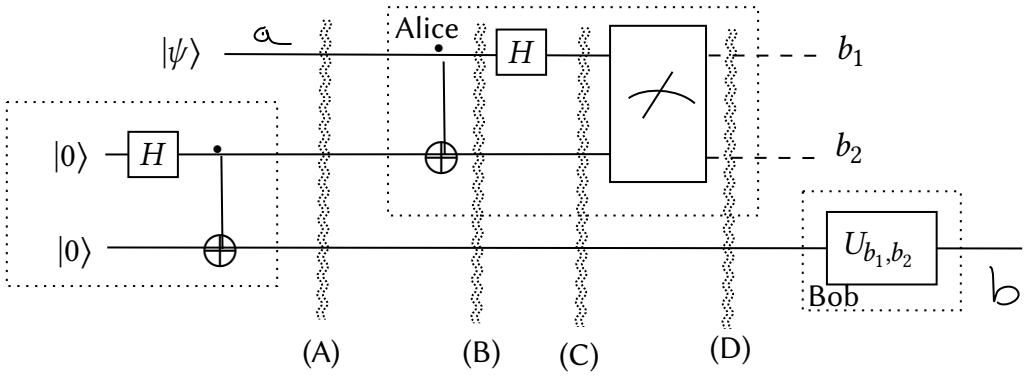


Figure 7: Scheme for Teleportation.

2.10.5 Let us develop the computation. Assume that $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$. Consider each layers in Fig. 7.
In (A).

$$\begin{aligned}
|\psi\rangle \otimes |00\rangle &\xrightarrow{H} |\psi\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle \\
&= |\psi\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle + |1\rangle \otimes |0\rangle) \\
&\xrightarrow{CNOT} |\psi\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle) \\
&= |\psi\rangle \otimes \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \\
&= (\alpha|0\rangle + \beta|1\rangle) \otimes \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \\
&= \frac{1}{\sqrt{2}}(\alpha|000\rangle + \alpha|011\rangle + \beta|100\rangle + \beta|111\rangle)
\end{aligned}$$

In (B).

$$\frac{1}{\sqrt{2}}(\alpha|000\rangle + \alpha|011\rangle + \beta|110\rangle + \beta|101\rangle)$$

In (C).

$$\begin{aligned}
&\frac{1}{\sqrt{2}}\begin{pmatrix} \alpha\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |00\rangle + \alpha\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|11\rangle \\ +\beta\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)|10\rangle + \beta\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)|01\rangle \end{pmatrix} \\
&= \frac{1}{2}(\alpha|000\rangle + \alpha|100\rangle + \alpha|011\rangle + \alpha|111\rangle + \beta|010\rangle - \beta|110\rangle + \beta|001\rangle - \beta|101\rangle) \\
&= \frac{1}{2}\begin{pmatrix} |00\rangle \otimes (\alpha|0\rangle + \beta|1\rangle) + |01\rangle \otimes (\alpha|1\rangle + \beta|0\rangle) \\ +|10\rangle \otimes (\alpha|0\rangle - \beta|1\rangle) + |11\rangle \otimes (\alpha|1\rangle - \beta|0\rangle) \end{pmatrix}.
\end{aligned}$$

In (D). The two first qubits got measured. The system lives in $\mathscr{H} \otimes \mathscr{H} \otimes \mathscr{H}$. The measurement projects the system on one of $|00\rangle \otimes \mathscr{H}$, $|01\rangle \otimes \mathscr{H}$, $|10\rangle \otimes \mathscr{H}$ and $|11\rangle \otimes \mathscr{H}$. Measuring, I then get with probability $1/4$ two bits $b_1 b_2$ as follows:

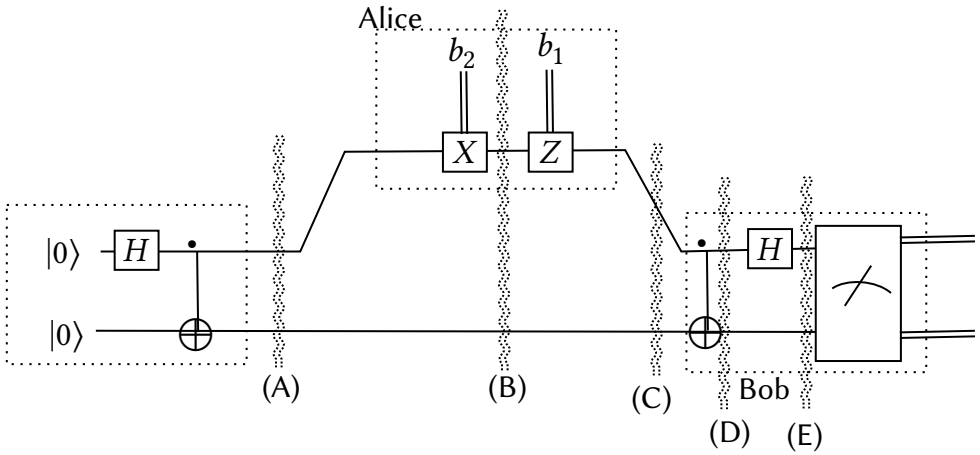


Figure 8: Scheme for Dense Coding.

- 00, and the state is now $|00\rangle \otimes (\alpha|0\rangle + \beta|1\rangle)$,
- 01, and the state is now $|01\rangle \otimes (\alpha|1\rangle + \beta|0\rangle)$,
- 10, and the state is nowt $|10\rangle \otimes (\alpha|0\rangle - \beta|1\rangle)$,
- 11, and the state is now $|11\rangle \otimes (\alpha|1\rangle - \beta|0\rangle)$.

If Bob wants to get back $|\psi\rangle$, he needs to perform

$$U_{00} = Id, \quad U_{01} = X, \quad U_{10} = Z, \quad U_{11} = ZX.$$

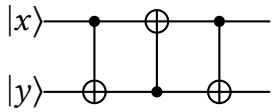
2.10.6 Discussion Note that the teleportation protocol does not refer to the localization of Alice and Bob. The only constraint is for them to share an entangled pair of qubits. But once this is done, they can go as far of each other as they want, the protocol will succeed. One could argue that the teleportation of the state is instantaneous, thus breaking the physical law stating that information cannot move faster than the speed of light. But there is no contradiction here: the state of the qubit is only modified when Bob receives the results of the measures of Alics. These bits moves inside a classical channel, subject to the law of physics, so nothing goes faster than the speed of light.

2.10.7 Recalling 2.8.18, Alice's action consists in measuring its two qubits in the *Bell basis*. ↳ 2.3.7

2.10.8 Dense Coding. The teleportation algorithm can be somehow "inverted": instead of sending a qubit through a classical channel, we can send two bits through a quantum channel, encoded on *one single qubit*. The algorithm is presented in Fig. 8. The gates X and Z are fired by Alice whenever the corresponding bit is set to 1. Bob should read the two bits at the end of the circuit in a non-probabilistic manner. See Exo. 2.11.7.

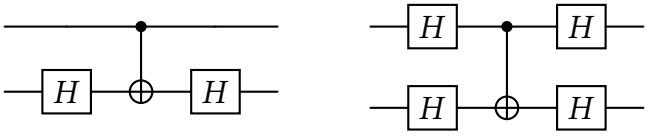
2.11 Exercices

2.11.1 Consider the circuit



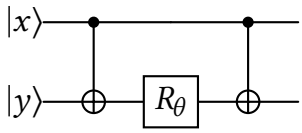
What does it compute? Give it in "function" form, and in matrix form (do not forget the basis ordering used for the representation!). Lay out the details in a convincing way.

2.11.2 Consider the following circuits.



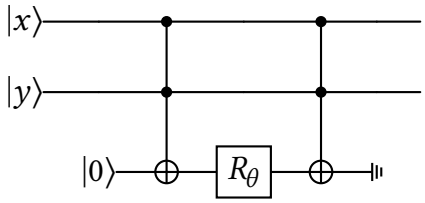
For each of them, give a simpler, equivalent circuit, and the corresponding linear map in function-style and in matrix form. Make sure to give the ordering of basis states you rely on. You might want to recall Exo 1.10.14, but it is not necessary.

2.11.3 Consider the circuit



with θ a real number. What does it compute? Give it in "function" form, and in matrix form (do not forget the basis ordering used for the representation!). Lay out the details in a convincing way.

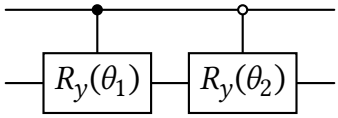
2.11.4 Consider the circuit



with θ a real number. What does it compute? Give it in "function" form, in (simpler) circuit-form, and in matrix form (do not forget the basis ordering used for the representation!). Lay out the details in a convincing way.

2.11.5 Section 2.9.8 gives a procedure to generate a multi-controlled NOT gate from Toffoli and ancillas. We want to realize a C^4-NOT gate (a NOT gate with 4 controls). Give two circuits only consisting of Toffoli gates, making use of 3 ancillas for the first circuit and only 2 ancillas for the other circuit.

2.11.6 Consider the circuit ↳ 2.5.7



1. Gives the corresponding matrices in the basis orderings $(|00\rangle, |01\rangle, |10\rangle, |11\rangle)$ and $(|00\rangle, |10\rangle, |11\rangle, |11\rangle)$.

2. Give a circuit made of multi-controlled 1-qubit gates and realizing the matrix

$$\begin{pmatrix} \cos(\theta_1) & 0 & 0 & 0 & -\sin(\theta_1) & 0 & 0 & 0 \\ 0 & \cos(\theta_2) & 0 & 0 & 0 & -\sin(\theta_2) & 0 & 0 \\ 0 & 0 & \cos(\theta_3) & 0 & 0 & 0 & -\sin(\theta_3) & 0 \\ 0 & 0 & 0 & \cos(\theta_4) & 0 & 0 & 0 & -\sin(\theta_4) \\ \sin(\theta_1) & 0 & 0 & 0 & \cos(\theta_1) & 0 & 0 & 0 \\ 0 & \sin(\theta_2) & 0 & 0 & 0 & \cos(\theta_2) & 0 & 0 \\ 0 & 0 & \sin(\theta_3) & 0 & 0 & 0 & \cos(\theta_3) & 0 \\ 0 & 0 & 0 & \sin(\theta_4) & 0 & 0 & 0 & \cos(\theta_4) \end{pmatrix}$$

when written in the canonical basis ordering

$$|000\rangle, |001\rangle, |010\rangle, |011\rangle, |100\rangle, |101\rangle, |110\rangle, |111\rangle.$$

Explain why it works.

2.11.7 Spell out the details of the computation for the dense-coding algorithm in 2.10.8, when the circuit is fed with an arbitrary pair of bits b_1, b_2.

3 Hardware Constraints and Circuit Synthesis

3.1 A bit of Complexity Theory

3.1.1 The notion of *complexity* refers to two related concepts: complexity of algorithms, and complexity of problems. In this section, we focus on the former, and we briefly recall what should be known about it in the context of quantum programming.

3.1.2 An algorithm is a mechanical procedure taking some input, and producing a result (possibly a simple "yes/no") after a certain number of well-defined operations. The complexity of the algorithm is an estimate of the amount of resources required when the size of the inputs grows to infinity. The complexity can focus on the overall memory footprint of the algorithm (*space complexity*), or on the number of operations required to run it to completion (*time complexity*).

The "size of the input" is taken in a literal sense: if we were to store the input on a regular storage device (USB key, hard-drive, etc), the input size is the number of bits it takes to store it. For instance, storing a natural number N in can be done in binary and this requires $\log_2(N)$ bits.

The notion of "number of operations" is of course a moving target, as it depends on the considered operations. In the conventional setting, one typically counts arithmetic operations and memory accesses. In the quantum setting, one considers the interaction with the quantum co-processor.

3.1.3 The complexity cares about the asymptotic behavior of the algorithm in a coarse way: If $f(x)$ is the quantity of resources required for processing an input of size x, we say that $f(x)$ is a *big-O* of $g(x)$ if there exists M and x_0 such that

$$\forall x \geq x_0, \quad f(x)leqM \cdot g(x).$$

We write $f(x) = O(g(x))$.

In general, we consider functions of several variables: the amount of resources might be based on two parameters. For instance, an adder takes an input consisting of two natural numbers M and N. The complexity of the adder depends on N and M (for an adder, typically the complexity is $O(\log_2(M) + \log_2(N))$.

3.1.4 The definition of big-O in 3.1.3 hides constants and smaller growth. This makes it possible to write $x + c = O(x)$ and $c \cdot x = O(x)$, when c is a constant. We also have $x + \log(x) = O(x)$ for instance. The definition permits overapproximation, such as $x = O(x^2)$.

Such an approach makes the notion of "input size" and "amount of resources" somehow canonical. For instance, maybe the integer N requires $\log_2(N) + 4$ bits in memory to account for its type or other meta-data. Another, more involved example is when the input is a graph with v nodes and e edges: the resources needed to store the graph will grow in correlation with e and v, but we can omit the fact that node and edge identifiers are more than 1-bit long: as long as they are "small-enough" they won't count in the complexity, and we can focus on v and e for the big-O formula.

3.1.5 The complexity of an algorithm gives some clue on how much useful it can be. For instance, one can expect an algorithm with an exponential complexity (i.e. a complexity of the form $O(e^x)$) to scale badly for large input sizes. On the contrary, an algorithm with a logarithmic, a linear or a low-degree polynomial complexity should be better-behaved: such complexities are the usual target for *reasonnable* amount of resources.

3.1.6 The complexity of an algorithm is however not the last word: the complexity is in particular hiding constant terms that might end up being very large. Consider for instance an algorithm running in logarithmic time with, for an input of size x, the number of operations being $10^{40} + \log(x)$. This is $O(\log(x))$, an arguably very good complexity. In real-life though, even for very small x's the algorithm requires a tantalizing amount of time to complete.

3.1.7 If the example of 3.1.6 might look contrived, such large, hidden overheads are unfortunately someting rather common in the context of quantum algorithms. This is why concrete resource estimation is very important to assess the effectiveness of a quantum algorithm for a given task.

3.1.8 In the discussion so far, we are omitting an important fact: because of the potential use of measurement, quantum algorithms are *probabilistic algorithms*. We say that a quantum algorithm has a complexity of $O(f(x))$ when it *sucessfully* completes with probability at least $2/3$ in $O(f(x))$.

Note that the probability $2/3$ is somewhat arbitrary: any other high probability of success can be attained in $O(f(x))$ by executing the algorithm several times. For instance, executing twice the algorithm makes a new algorithm succeeding with probability $8/9$ in $O(f(x))$. We use the fact that $2 \cdot f(x) = O(f(x))$.

3.1.9 Finally, it is always good to keep in mind that the complexity of a quantum program might be radically changed along the compilation process. Indeed, the chosen hardware might require costly gate decompositions, various error correcting schemes and complex qubit mapping, blowing up the overall cost of running the program.

3.2 Low-level gate-sets

3.2.1 As discussed in 2.1.4 and 2.4.3, on a quantum memory one cannot directly implement arbitrary unitaries: each quantum co-processor is limited to a particular *gate-set*. These constrainsts can come from the physics of the implementation, but also from the error correcting scheme, limiting what is possible. Typically, these gate-sets are chosen so that any unitary operator can be synthetized, at least in an approximate manner.

3.2.2 Formally, we say that a gate-set S is *universal* if for all n, for all unitary operator U, there exists a ciruit made out of gates from S realizing U. We say that the gate-set is *appoximately universal* if for all n, for all unitary operator U, for all error $\varepsilon > 0$, there exists a circuit C made out of gates of S such that "C approximates U up to ε", i.e.

$$\forall\, |\psi\rangle\,, |(U - C)\,|\psi\rangle| \leq \varepsilon \cdot ||\psi\rangle|.$$

These two definition generalizes the 1-qubit case discussed in Sec. 2.5.5 and 2.5.7.

3.2.3 In order to perform quantum computation, the main gate-set that is required for every set-up is a set allowing one to implement *Clifford* operators. These are the operations generated by the gates HAD, S and CNOT. It is *not universal*, and in fact it is efficiently simulable on a classical machine.

Typical examples of universal gate sets extends Clifford gates with at least one non-Clifford gates, as follows.

- CNOT + 1-qubit rotations. This is the typical gate-set that can be realized in the context of linear optics or supraconducting qubits (the quantum co-processor of IBM , Google, Rigetti, *etc*). The universality of this gate set is proved in Sec. 3.3.
- MS + R_z + R_x also form a universal gate set, with

$$MS(\theta) = \begin{pmatrix} \cos(\theta) & 0 & 0 & -i\sin(\theta) \\ 0 & \cos(\theta) & -i\sin(\theta) & 0 \\ 0 & -i\sin(\theta) & \cos(\theta) & 0 \\ -i\sin(\theta) & 0 & 0 & \cos(\theta) \end{pmatrix}$$

  This is the Mølmer-Sørensen gate, also written XX. Parameterized by an angle θ, this gate is used in the context of the ion-trap technology[6].

- Control-Z and the family of gates $J(\alpha)$, defined by

$$J(\alpha) = \begin{pmatrix} 1 & e^{i\alpha} \\ 1 & e^{-i\alpha} \end{pmatrix}.$$

  This set of gates is in strong relation with *MBQC* (*Measurement-Based Quantum Computation*), discussed in Sec. 3.6.

- Clifford+T. This is the typical gate-set for fault-tolerant quantum computation, *approximately* universal. This is discussed in Sec. 3.3.19.

Some more exotic universal gate-set exists, based on the Toffoli gate. A first result due to Kitaev [Kit97] shows that Toffoli, HAD and S are (approximately) universal. Maybe more surprisingly, Toffoli and any 1-qubit basis change such as HAD is also universal! [Shi03]. You might notice that none of these two gates contains coefficients with imaginary parts. The encoding of a generic unitary operation then requires an extraneous wire to "store" this imaginary part, but the encoding is efficient in the sense that one can turn a Clifford+T circuit into a Toffoli+HAD circuit in a polynomial manner.

3.2.4 Clifford needs "a bit of help" to get to universality. Instead of using additional unitary gates such as Toffoli or 1-qubit gates, it is also possible to consider the use of *magic states* [BK05]. These are typically 1-qubit registers, initialized in a non-canonical basis state. The use of magic states requires measurements and the possibility to perform *some* form of classical processing in the quantum co-processor. We discuss magic states in Sec. 3.5.

[6] see e.g. https://arxiv.org/abs/1603.07678

3.3 Universality of CNOT and 1-qubit rotations

3.3.1 In 2.4.3, we discussed how quantum internal operations are unitaries acting on the whole state space, while the quantum co-processor can only apply a handful of operations. In this section, we shall see how one can realize *any* unitary operation on *any* n-qubit states with only CNOT gates and 1-qubit rotations. The reader can refer to arXiv:quant-ph/9503016 for a reference.

3.3.2 Theorem For all unitary U on 1-qubit, there exists an angle α and 3 1-qubit unitaries A_1, A_2, A_3 such that $A_1A_2A_3 = I$ and $U = e^{i\alpha}A_1XA_2XA_3$.

3.3.3 *Proof.* According to Th. 2.5.9, U can be written as

$$U = e^{i\alpha}R_z(\beta)R_y(\gamma)R_z(\delta).$$

Define

$$\begin{aligned} A_1 &= R_z(\beta)R_y(\gamma/2), \\ A_2 &= R_y(-\gamma/2)R_z(-(\delta+\beta)/2), \\ A_3 &= R_z((\delta-\beta)/2), \end{aligned}$$

Then

$$\begin{aligned} A_1A_2A_3 &= R_z(\beta)R_y(\gamma/2)R_y(-\gamma/2)R_z(-(\delta+\beta)/2)R_z((\delta-\beta)/2) \\ &= R_z(\beta)R_y(\gamma/2)R_y(-\gamma/2)R_z(-\beta) \\ &= R_z(\beta)R_z(-\beta) \\ &= I. \end{aligned}$$

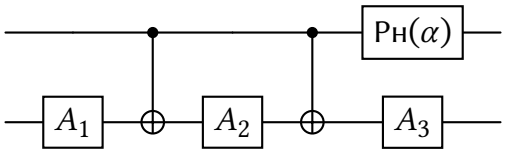
Then, we can realize that $XX = I$, and $XR_y(\alpha)X = R_y(-\alpha)$, and $XR_z(\alpha)X = R_z(-\alpha)$. The following the hold:

$$\begin{aligned} A_1XA_2XA_3 &= R_z(\beta)R_y(\gamma/2)XR_y(-\gamma/2)R_z(-(\delta+\beta)/2)XR_z((\delta-\beta)/2) \\ &= R_z(\beta)R_y(\gamma/2)XR_y(-\gamma/2)XXR_z(-(\delta+\beta)/2)XR_z((\delta-\beta)/2) \\ &= R_z(\beta)R_y(\gamma/2)R_y(\gamma/2)XR_z(-(\delta+\beta)/2)XR_z((\delta-\beta)/2) \\ &= R_z(\beta)R_y(\gamma/2)R_y(\gamma/2)R_z((\delta+\beta)/2)R_z((\delta-\beta)/2) \\ &= R_z(\beta)R_y(\gamma)R_z(\delta), \end{aligned}$$

closing the proof. □

3.3.4 Theorem Let U be a 1-qubit unitary. The operation C-U can be realized with only CNOT and 1-qubit gates.

3.3.5 *Proof.* Using Th. 3.3.2, one can decomposed U into $U = e^{i\alpha} A_1 X A_2 X A_3$. One can write the circuit:



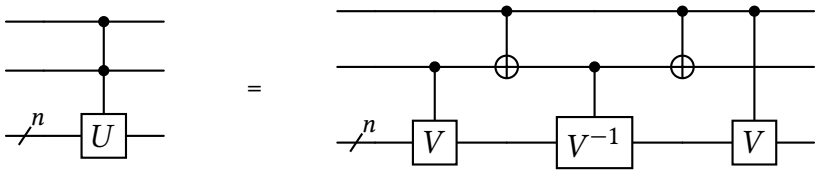
with $\text{PH}(\alpha)$ defined as in Sec. 2.5.8. Indeed, when the control wire is $|1\rangle$, we need to perform U, including the global phase. But this global phase should not be done with the control qubit is $|0\rangle$. This has already been discussed in 2.6.11. □

3.3.6 Theorem Consider an n-qubit unitary U. There exists an n-qubit unitary V such that $V^2 = U$.

3.3.7 *Proof.* The existence of V derives from the decomposition of U as e^{iH} for some hermitian matrix H (see 1.9.8). One can then use $V = e^{iH/2}$, relying on 1.9.10. □

3.3.8 Theorem Let U by an n-qubit unitary. The doubly-controlled gate C-C-U can be realized with only CNOT and simply-controlled n-qubit gates.

3.3.9 *Proof.* Let V be defined as in Th. 3.3.6. A circuit realizing C-C-U is



Indeed, the various cases for the controlling qubits are as follows:

- 00 ⟶ on ne fait rien
- 01 ⟶ on fait C-V et C-Vinv ⟶ identité
- 10 ⟶ d'abord C-Vinv puis C-V ⟶ identité
- 11 ⟶ C-V puis C-V : C-U

The operation U is then applied to the target qubit if and only if the control qubits are at 11. □

3.3.10 Theorem Let U by a 1-qubit unitary. We can realize the multi-controlled gate C-C-...-C-U with only CNOT and 1-qubit gates.

3.3.11 *Proof.* Using repeatedly Th. 3.3.8 we decompose C-⋯C-U into CNOT-gates and simply-controlled 1-qubit gates. We can conclude by using Th. 3.3.4 to decompose each simply-controlled 1-qubit gates into CNOT and 1-qubit unitaries, to reach the required form. □

3.3.12 We are now almost ready to show that a $2^n \times 2^n$ unitary matrix can be realized (we say *synthetized*) with CNOT gates and 1-qubit unitaries. For this, we need a result allowing us to relate arbitrary unitary matrices and multi-controlled gates. The result is the following theorem, yielding Th. 3.3.15.

3.3.13 Theorem (*Cosine-Sine Decomposition*)**.** Let U by any $n+1$-qubit unitary. One can decompose U as

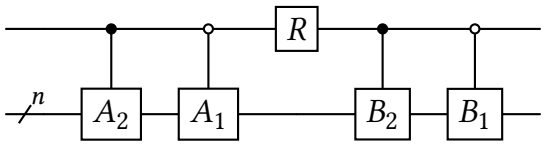
$$\begin{pmatrix} B_1 & 0 \\ 0 & B_2 \end{pmatrix}\begin{pmatrix} C & -S \\ S & C \end{pmatrix}\begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$$

with A_1, B_1, A_2 and B_2 n-qubit unitaries, and C and S n-qubit diagonals such that $S^2 + C^2 = Id$.

3.3.14 *Proof.* See Appendix C.

3.3.15 Theorem. Any $n+1$-qubit unitary can be realized with multi-controlled, 1-qubit unitaries.

3.3.16 *Proof.* The proof proceeds by induction on n. In the base case, $n = 1$ and there is nothing to fo. The inductive case makes use of the CS decomposition stated in Th. 3.3.13. This decomposition can be drawn as the circuit:



where R is a circuit realizing the matrix

$$\begin{pmatrix} C & -S \\ S & C \end{pmatrix}.$$

This circuit can be built using the technique on 2.11.6: the matrix C can be seen as a diagonal of Cosine functions, while the matrix S contains the corresponding Sine functions. This gives a *multiplexor* of multi-controlled R_y-rotations. □

3.3.17 Theorem. Given a unitary matrices on n qubit, one can synthetize it with only CNOT gates and 1-qubit rotations.

3.3.18 *Proof.* The proof consists in chaining the decompositions. First, using Th. 3.3.15 the matrices yields a circuits consisting of multi-controls of 1-qubit gates. Then, using Th. 3.3.10 each of these multi-controls can be decomposed into CNOT and 1-qubit unitaries. Finally, each of these 1-qubit unitaries can be implemented with rotations using Th. 2.5.9. □

3.3.19 Combining Th. 2.5.5 and 3.3.17, we can conclude that any n-qubit unitary can be implemented up to arbitrary error with only CNOT, Hadamard and T-gates.

3.3.20 Discussion The proof in Sec. 3.3.18 gives a construction for a circuit implementing an arbitrary unitary $2^n \times 2^n$-matrix. The size of the circuit is clearly exponential on the number n of qubits. Is there a way of constructing a non-exponentially-sized circuit? Without any constraints on U, this is bound to fail. One can count the number of degrees of liberty for the system: the matrix contains 4^n complex numbers. Even if we factor out the unitarity constraints, we still need on the order of 4^n angles to describe all of them. These angles are necessary: regardless of how we will handle them, they will end up crawling inside rotation gates in the circuit.

3.4 Tradeoffs: a Case-Study

3.4.1 Given a unitary matrix $2^n \times 2^n$, if one question is to be able to implement it with the available gate-set, the other is to be able to do it with a circuit of reasonnable size. As discussed in 3.1.5, in computer science, "reasonnable" is defined as "polynomial in n". The strategy proposed in 3.3.18 is clearly not reasonnable with this definition.

In some situation, it is possible to do better when we can capitalize on the structure of the unitary. In order to see how this can work in practice, we propose to study two techniques to implement a multi-controlled R_θ-gate (where we write R_θ in place of $\mathrm{PH}(\theta)$ for compactness).

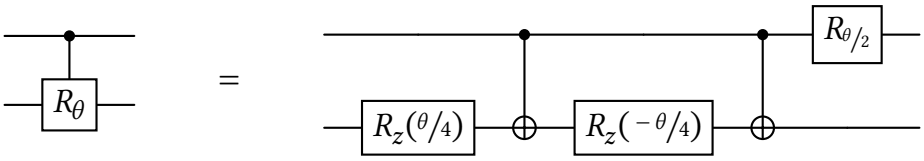
3.4.2 Exponentially large circuit Using Th. 3.3.4, note that we can write $R_\theta \triangleq \mathrm{PH}(\theta)$ as

$$e^{i\frac{\theta}{2}} R_z(\theta/4) R_y(0) R_z(\theta/4).$$

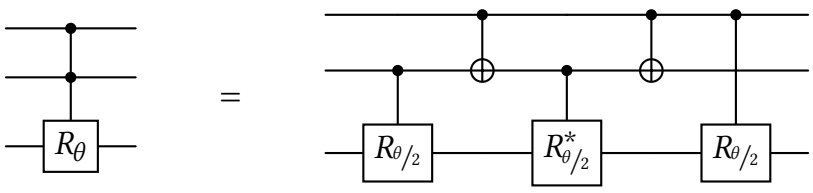
From the proof of Th. 3.3.2, we set

$$A_1 = R_z(\theta/4), A_2 = R_z(-\theta/4), A_3 = Id,$$

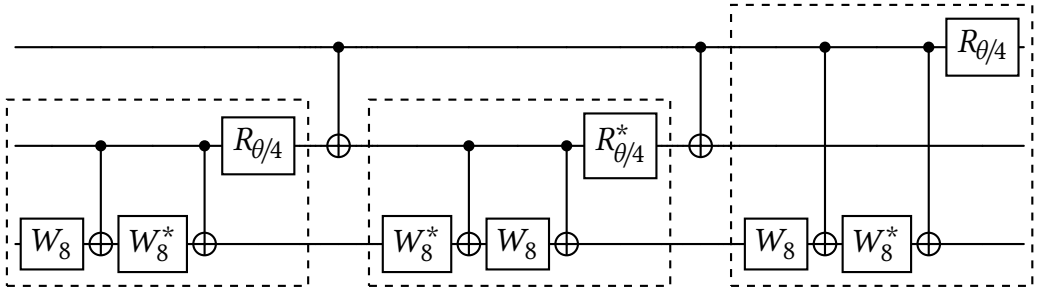
so that



3.4.3 Let us now write a circuit for C-C-R_θ: using Th. 3.3.8 with $U = R_\theta$, we get



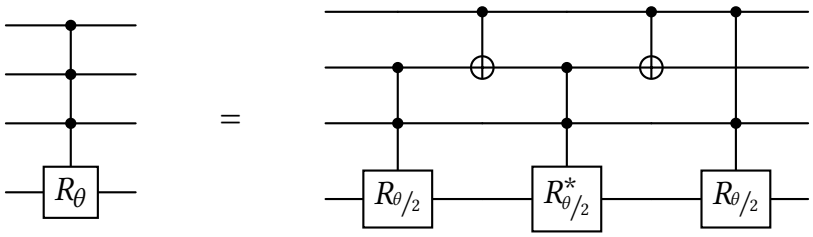
Decomposing each $C\text{-}R_{\theta/2}$ as the circuit of Sec. 3.4.2, we get:



(33)

with W_n being $R_z(\theta/n)$

3.4.4 Let us now write a circuit for $C\text{-}C\text{-}C\text{-}R_\theta$: using Th. 3.3.8 with $U = C\text{-}R_\theta$, we get



And each $C\text{-}C\text{-}R_{\theta/2}$ can itself be replaced with the circuit of Eq. (33).

3.4.5 The process can be iterated for synthetizing the multi-control gate $C^n\text{-}R_\theta$ with only CNOT-gates, R_z and R gates. Let us count how many of them we get: we write N_n^G for the number of gates G in $C^n\text{-}R_\theta$.

- $N_1^{\text{CNOT}} = 2$, and $N_{n+1}^{\text{CNOT}} = 2 + 3N_n^{\text{CNOT}}$. We deduce that
$$N_n^{\text{CNOT}} = 2\sum_{i=0}^{n-1} 3^i = 3^n - 1.$$
- $N_1^{R_z} = 2$, and $N_{n+1}^{R_z} = 3N_n^{R_z}$. We deduce that
$$N_n^{R_z} = 2 \cdot 3^{n-1}.$$
- $N_1^{R} = 1$, and $N_{n+1}^{R} = 3N_n^{R}$. We deduce that
$$N_n^{R} = \cdot 3^{n-1}.$$

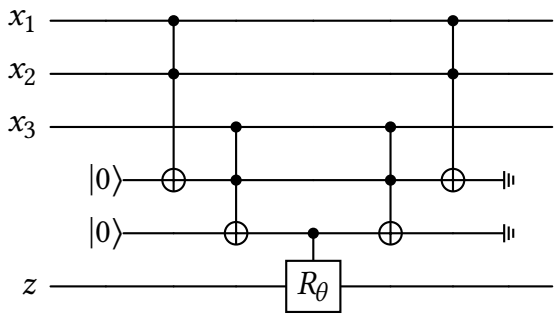
Furthermore, for n controls the angle for the R-gate is always $\pm\theta/2^n$, and the angle for the R_z-gate is always $\pm\theta/2^{n+1}$.

3.4.6 In any case, in this construction the size of the circuit is exponential on the number of controls. Furthermore, there is no possibility to "compact" the circuit by parallelizing the calls to R_z-gates, as they are all on the last line, interwined with CNOT-gates. If this circuit is arguably not the smallest one, without any ancillas we are bound to have an exponential number of rotation gates.

3.4.7 Polynomial-sized circuit If we can allocate ancillas, it is possible to do better using the trick presented in Sec. 2.9.8. Indeed, the operation C^n-R_θ performs the operation

$$|x_1\rangle |x_2\rangle \cdots |x_n\rangle \otimes |z\rangle \longmapsto (e^{i\theta})^{x_1 x_2 \ldots x_n} |x_1\rangle |x_2\rangle \cdots |x_n\rangle \otimes |z\rangle$$

We can then first compute the product of the x_i's, store it in an ancilla, perform a simple C-R_θ-gate, and undo the intermediate computations. An example for C-C-C-R_θ is as follows.



In general, the gate C^{n+1}-R_θ with $n+1$ controls requires n ancillas, $2n$ Toffoli gates and one C-R_θ gate. The former can be realized for instance with two Had-gates and the circuit of Sec. 3.4.3, while the latter can be realized with the circuit of Sec. 3.4.2. We then have a linear-sized, much more compact circuit. We can also note that the angles that are required for the single-qubit rotation gates are at worst $\theta/8$: we are not working with exponentially small angles, as discussed in Sec 3.4.5. But again, this assumes that we can afford to use auxiliary registers.

3.4.8 Discussion We presented two concrete algorithms to synthetize C^n-R_θ gates: these are of course not the only existing techniques. However, the main idea remains: there is no silver bullet, and *tradeoffs* have to be decided upon, depending both on the backend and on the targetted use-case. In particular, one can note the following three tradeoffs

1. As discussed above, the reduction of the circuit size is typically correlated with an increase in the pool of ancillas.

2. The more compact the circuit is, the more *time* it takes to synthetize it. If one needs a steady stream of circuit generation on the fly, one might prefer a technique producing slightly larger circuits but in a fast manner rather than a slower technique producing more optimized circuits.

3. Informations on the *structure* of the unitary to synthetize might be capitalized upon to produce a smaller circuit. An example of such as case is discussed in 4.2.1 with the generation of *oracles*.

3.5 Quantum Computation with Magic States

3.5.1 If the low-level target is error corrected —such as with the *surface code* [Cle22]— non-Clifford gates might not be directly implementable at the logical level. If non-Clifford such as T-gates are not available, a solution consists in initializing the memory

in a state that contains the phases that the rotations would have brought. Rotation gates are then not native, but instead built from these *magic states* [BK05], Clifford gates, and a bit of classical processing. Such a rotation gate is then way more costly than Clifford gates, requiring dedicated error correction and optimization scheme.

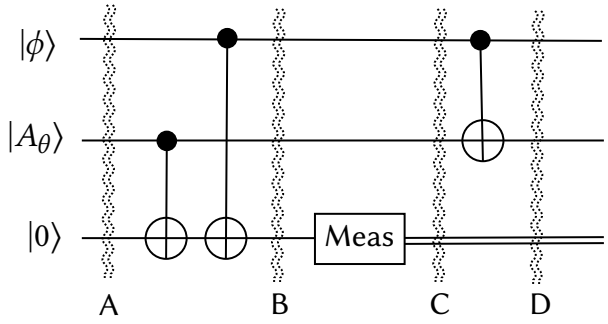
3.5.2 A typical magic state is

$$|A_\theta\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{i\theta}|1\rangle).$$

If we have access to several copies of the magic states, it is possible to realize the rotation

$$R_\theta = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}.$$

The procedure is summarized by the circuit:



Suppose that $|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$. At each step, the state of the system is:

(A) $|\phi\rangle \otimes |A_\theta\rangle \otimes |0\rangle$, that is

$$\frac{1}{\sqrt{2}}(\alpha|000\rangle + \alpha e^{i\theta}|010\rangle + \beta|100\rangle + \beta e^{i\theta}|110\rangle).$$

(B) The two CNOT-gates store in qubit 3 the parity respective of qubit 1 and 2: 0 for 00 and 11, and 1 for 01 and 10. The state becomes

$$\frac{1}{\sqrt{2}}(\alpha|000\rangle + \alpha e^{i\theta}|011\rangle + \beta|101\rangle + \beta e^{i\theta}|110\rangle).$$

(C) The measurement gives with probability $\frac{1}{2}$ the states

$$(\alpha|00\rangle + \beta e^{i\theta}|11\rangle) \otimes |0\rangle$$

when 0 is measured, and

$$(\alpha e^{i\theta}|01\rangle + \beta|10\rangle) \otimes |1\rangle$$

when 1 is measured.

(D) The last CNOT-gate will unentangle the state of qubit 1 and 2: if we had measured 0 we would get

$$(\alpha |0\rangle + \beta e^{i\theta} |1\rangle) \otimes |0\rangle \otimes |0\rangle$$

and if we had measured 1 we would get

$$(\alpha e^{i\theta} |0\rangle + \beta |1\rangle) \otimes |1\rangle \otimes |1\rangle$$

that can be rewritten (by change of global phase) into

$$(\alpha |0\rangle + \beta e^{-i\theta} |1\rangle) \otimes |1\rangle \otimes |1\rangle .$$

The two last qubits are now in a canonical basis state: they can be recycled and reused without impacting the global state of the memory.

The effect of the circuit is then

$$(\alpha |0\rangle + \beta |1\rangle) \otimes |A_\theta\rangle \otimes |0\rangle \longmapsto (\alpha |0\rangle + \beta e^{\pm i\theta} |1\rangle) \otimes |1\rangle \otimes |1\rangle ,$$

with the sign depending on the result of the measurement.

3.5.3 Because the sign measurement is probabilistic, if we repeat the process (each time with a fresh magic state $|A_\theta\rangle$), we will get roughly the same number of 0 and 1 out of the measurements —let us respectively denote these numbers with n_0 and n_1. The state is in this case:

$$(\alpha |0\rangle + \beta e^{(n_0 - n_1)i\theta} |1\rangle).$$

Because of the statistic of the process, we will eventually meet a point in time where $n_0 = 1 + n_1$: the state of qubit 1 then corresponds to the application of the gate R_θ to $|\phi\rangle$. We can stop there: we implemented the gate R_θ.

3.5.4 Of course, in general, the number of times needed to repeat the circuit is not known: we have a *repeat-until-success* scenario. In order for this scheme to work, the result of the measurement should not have to be sent to the classical host. Indeed, the delay would be way too costly. In this setting, the co-processor therefore needs to have *some* form of classical control: here, the possibility to perform some simple arithmetics, and the possibility to repeat a piece of circuit.

3.5.5 The magic state presented in Sec. 3.5.2 is a very simple one. There are many other proposals, possibly with simpler computational schemes. All of these schemes are however relying on the same underlying technique: *repeat-until-success*. See for instance [BK05].

3.6 Measurement-Based Quantum Computation

My scribbles are not yet written properly, this section is still to be filled!

3.7 Classical Computation in the Co-Processor

3.7.1 In order to implement the techniques presented in 3.5, the circuit model presented in 2.4.7 has to be extended with classical registers, and operations to act on them. In this extended model, the bit generated from a measurement gets stored in a classical register, within the quantum co-processor. If the classical host needs the information, this bit needs to be extracted from the co-processor.

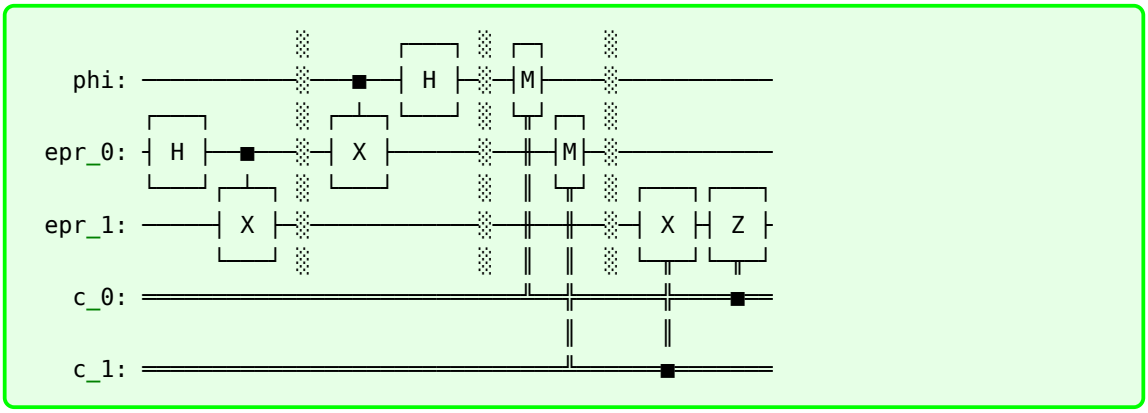
3.7.2 A quantum programming framework offering such a model is QisKit, a Python library developed by IBM[7]. An example of circuit mixing both quantum and classical control is as follows.

```
# 2 qubits for the Bell pair
epr = QuantumRegister(2, name="epr")

# The qubit to teleport
phi = QuantumRegister(1, name="phi")

# Classical register for storing the results
c = ClassicalRegister(2, name="c")

# We build a quantum circuit with both registers.
# By default, everything is initialized to 0 and to |0>
qc = QuantumCircuit(phi, epr, c)

# Generating the Bell pair
qc.h(epr[0])
qc.cnot(epr[0],epr[1])
qc.barrier()

# Action of Alice
qc.cnot(phi, epr[0])
qc.h(phi)
qc.barrier()

# Mesure of all of register q, storing results in c.
# This is still part of the circuit
qc.measure(phi, c[0])
qc.measure(epr[0], c[1])
qc.barrier()

# Now, action of Bob
qc.x(epr[1]).c_if(c[1],1)
qc.z(epr[1]).c_if(c[0],1)
```

For legibility of the printed circuit, we added "barriers" to regroup pieces of circuits together. The resulting circuit, as shown by QisKit in ASCII art is as follows. At the end

[7] https://qiskit.org/

of the circuit, the state of the qubit `phi` is in `epr_1`.



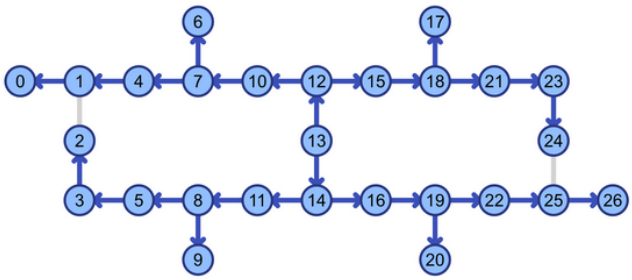
3.7.3 A question is the classical power of the quantum co-processor: what are the operations allowed on classical registers? We at least need to be able to apply quantum gates conditionally on the value of a Boolean register. To realize the scheme presented in Sec. 3.5, we also need (limited) arithmetic but also loops. For representing such a process, the circuit notation starts to show its limits, and if this were a more expressive language might be required. More generally, this question of where to place the limit between the classical and the quantum host is at the core of the discussions on hybrid computation.

3.8 A Word on Hardware

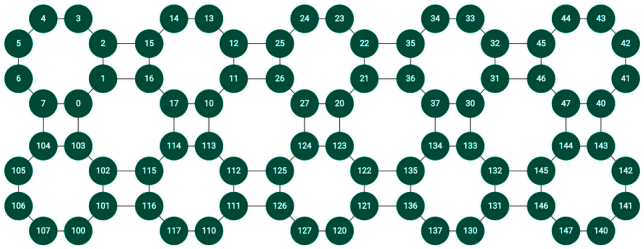
3.8.1 Noise As discussed in 2.2.3, quantum information is encoded on the state of objects governed by the laws of quantum physics. Such objects are subject to a physical phenomenon called *decoherence*: due to an interaction with the environment, the state of the system evolves in uncontrolled ways. From a computer science point of view, this amounts to *noise* and the introduction of *errors* in the stored information.

3.8.2 NISQ and LSQ Noisy hardware is the current state of quantum co-processors. As of 2024, quantum memory holds at most a few hundred of noisy qubits: we are in the realm of *NISQ*, *Noisy Intermediate Scale Quantum Computation*. If we are arguably reaching the tipping point, the error rate and the (relatively) small memory size makes it still impossible to run quantum error correcting schemes. Building (logical) stable qubits for enabling large memory (dubbed *LSQ*: *Large Scale Quantum*) is still an active subject of research, both in academic and industrial labs.

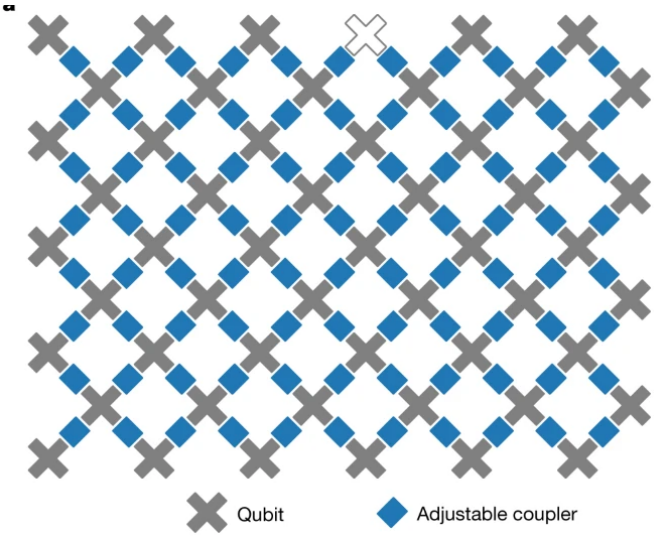
3.8.3 Topology For hardware (or some fault-tolerant scheme, such as surface code), the quantum bits have a fixed location. In these cases, two-qubits operations can only be performed on neighboring qubits: the notion of neighbors is described with a graph. We say that there is a *topology*. For instance, the IBM co-processor "Montreal" has 26 qubits arranged as follows:



The Rigetti co-processor “Aspen-M” is as follows:



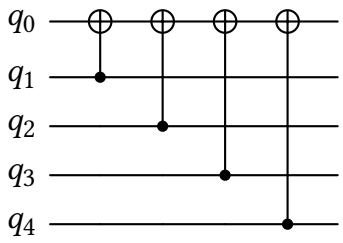
In these two cases, the circles are the qubit locations, while the edges are where CNOT-gates are allowed. Another example using the same technology is the Google coprocessor “sycamore”, with one non-functionning qubit (the white one):



Here, qubits are crosses, and the blue links are where 2-qubit operations are allowed.

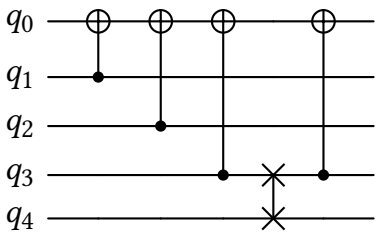
3.8.4 The problem this poses is the *mapping*, or *routing* of qubits. That is: given a (logical) circuits, to which physical qubit do we associate each wire? This question turns out to be an NP-complete problem: the *subgraph isomorphism problem*. In general not all circuits can be mapped on any given topology, and when not feasable the goal is to rewrite the circuit into an equivalent one such that a mapping exists: this turns the decision problem into an optimization problem.

3.8.5 For instance, consider the circuit



Wire q_0 needs to be mapped to a physical qubit of arity 4: if it is somewhat possible for Google Sycamore co-processor (modulo the fact that CNOT-gates are not native there), it is not possible to do it directly for IBM's Montreal and Rigetti's Aspen-M co-processor, as the largest possible arity is 3. To implement such a circuit, a solution consists in rewriting the circuit with swaps (themselves implementable with CNOT-gates) to reduce the arity of problematic gates.

3.8.6 The circuit presented in Sec. 3.8.5 can be rewritten as



with a swap between q_3 and q_4. We can now find a routing for e.g. IBM Montreal:

$$q_0 \mapsto 12, \quad q_1 \mapsto 15, \quad q_2 \mapsto 13, \quad q_3 \mapsto 10, \quad q_4 \mapsto 7.$$

At the end of the computation, the wires are found in the following locations:

$$q_0 \mapsto 12, \quad q_1 \mapsto 15, \quad q_2 \mapsto 13, \quad q_3 \mapsto 7, \quad q_4 \mapsto 10.$$

We can perform another swap if needed to place back q_3 and q_4 at their original locations if needed.

3.8.7 In general, finding an optimal *mapping* or *routing* for a given circuit is an NP-complete problem: for non-trivial circuits we have to rely on approximate solutions. Existing compilers such as T|KET⟩ from CQC or QASM from Atos implements state-of-the-art techniques for the mapping problem. Such tools can also handle additional constraints such as specific costs to minimize.

3.9 Exercises

3.9.1 Consider the following quantum algorithm for factoring a number N:

- Allocate $n = \log_2(N)$ qubits in state $|0\rangle$
- Apply Hadamard on all of them
- Measure them and retrieve the corresponding bitstring

- This bitstring is the encoding of a natural number: test whether it is a factor of N (with the usual, efficient Euler division)
- If it is not, start over.

This algorithm eventually succeeds. Questions:

1. What is the probability of success?
2. How many times should we iterate the process in order to reach a high probability?
3. Derive the complexity of this (probabilistic) algorithm.

4 Structure of Quantum Algorithms

4.1 High-Level View

4.1.1 As any algorithm, the role of a *quantum algorithm* is to solve a regular, *classical problem*. Such a problem typically consists in

- An *instance*: a list of classical information: a graph, an integer, a matrix, *etc*. These can be given as datastructure: an nsigned integer on 64 bits, an array of floating point numbers, *etc*, or as a function: for constructing the neighbors of a node in a graph, for computing the coefficient of a matrix given a pair of indices, *etc*.
- A question, whose answer might be yes or no, or an object to construct.

Given a problem, an algorithm inputs an instance and outputs an answer to the question. The critical point is that the output should indeed answer the question: this requires a mathematical analysis of the algorithm and its adequacy with the problem.

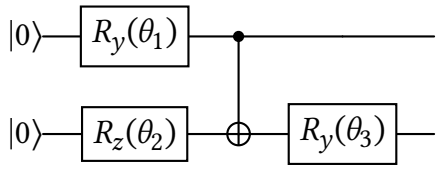
4.1.2 A quantum algorithm can be regarded as a regular algorithm which makes use of a quantum co-processor to build an answer to the question. The general structure of a quantum algorithm as found in the litterature is as follows:

1. Input a problem instance.
2. From this instance, specify a bunch of quantum registers, and build a quantum circuit made of quantum (unitary) gates. Among the registers, one of them is meant to be measured at the end.
3. Send the circuit to the quantum co-processor, and measure the specified register: This gives back a bitstring.
4. Perform some post-processing using the bitstring. There are two cases:
   (a) A solution to the problem instance was found! Exit with success.
   (b) No solution was found: Loop back at Step (3).

The measurement is probabilistic: the algorithm is designed in such a way that we eventually branch out in (4a) with high enough probability. In such an algorithm, if the circuit depends on the problem instance, once it is built it is used over and over until a solution is found. The algorithm somehow builds its very own *faulty soothsayer*. Since it is faulty, several queries might be needed to eventually get a correct solution.

4.1.3 Note how the general structure presented in 4.1.2 generates a distinct circuit for every problem instance. If your algorithm aims at finding a specific node a graph, two different graphs will give two different circuits. Similarly, if you aim at factoring 15 or 110210873687 surely you wouldn't use the same circuit either. A quantum algorihm is therefore not describing *one* quantum circuit but a *family of quantum circuits*.

4.1.4 The situation presented in 4.1.2 is very simple: given a problem instance the circuit is defined once for all. A slightly more general procedure is found in *variational algorithms*. There, instead of constructing *one* circuit, the algorithm constructs a circuit parameterized by angles: some gates, such as rotation gates, can be changed specified at run-time. For instance, the following circuit is parameterized by three angles $\theta_1, \theta_2, \theta_3$:



When the angles are not decided yet, we just have a shape, or a structure of circuit. Such a circuit-shape is called an *ansatz*.

4.1.5 The interaction with the quantum coprocessor is relatively simple for both 4.1.2 and 4.1.4: a circuit is flushed to the quantum memory, followed by a measurement, and a complete reset. The quantum memory between calls to the co-processor is not preserved. More exotic algorithms require such a preservation between calls: the stream of gates sent to the co-processor is not a circuit built once for all, but is instead based on the result of intermediary measurements. More than a fixed structure, the circuit comes as a trace of (classical) execution.

4.1.6 In the rest of this section, we shall look at typical subroutines used in the design of quantum algorithms.

4.2 Oracles

4.2.1 As discussed in 4.1.3, a quantum circuit should somehow contain a description of the problem instance. As the problem instance is typically encoded on a regular, conventional memory made of bits, such description can typically be defined using a classical, conventional function acting on bitstrings. We are therefore given a Boolean function $f : \mathbb{B}^n \to \mathbb{B}^m$, and we need to make it available to the quantum co-processor. Of course, this has to be done with the use of unitary operations.

4.2.2 If the function f is reversible, then $n = m$ and we can rely on 1.10.3 to identify f with the action of a unitary map on basis vectors. For instance, consider the operation

x	000	001	010	011	100	101	110	111
$f(x)$	001	010	011	100	101	110	111	000

You can easily convince yourself that this function is a bijection on bitstrings of size 3 as it corresponds to single digit increment modulo 8. This operation can be generalized to the unitary operation

$$|000\rangle \mapsto |001\rangle \qquad |001\rangle \mapsto |010\rangle \qquad |010\rangle \mapsto |011\rangle \qquad |011\rangle \mapsto |100\rangle$$
$$|100\rangle \mapsto |101\rangle \qquad |101\rangle \mapsto |110\rangle \qquad |110\rangle \mapsto |111\rangle \qquad |111\rangle \mapsto |000\rangle\,.$$

With the standard basis in lexicographic ordering, the corresponding matrix is

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

Referring again to 1.10.3, we can claim that this matrix is unitary. From what we discussed in Ch. 3, we know that we can implement this operation on a quantum coprocessor.

4.2.3 In general, the function f of 4.2.1 might not be reversible: we cannot identify it with a unitary.

Without loss of generality, one can assume that $m = 1$. Indeed, if $m > 1$, the function f can be regarded as a family of functions $(f_1, \dots, f_m)$, all of codomain $\mathbb{B}$. Therefore assuming $m = 1$, as the function f is usually not invertible the standard way to do this is to instead implement

$$\begin{aligned} \tilde{f} : & \mathbb{B}^n \times \mathbb{B} \to \mathbb{B}^n \times \mathbb{B} \\ & (x, y) \mapsto (x, y \oplus f(x)). \end{aligned}$$

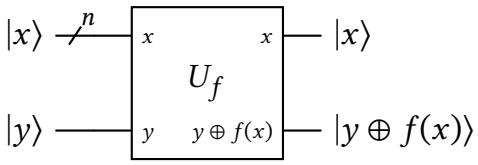
This function is invertible, as

$$\tilde{f}(\tilde{f}(x, y)) = \tilde{f}(x, y \oplus f(x)) = (x, y \oplus f(x) \oplus f(x)) = (x, y)$$

since $z \oplus z = 0$ for all z, and $z \oplus 0 = z$ for all z. This function being invertible, the unitary

$$U_f : \mathscr{H}^{\otimes n} \otimes \mathscr{H} \to \mathscr{H}^{\otimes n} \otimes \mathscr{H}$$

defined by



Note that U_f is indeed a unitary, it sends a basis vector to a basis vector,

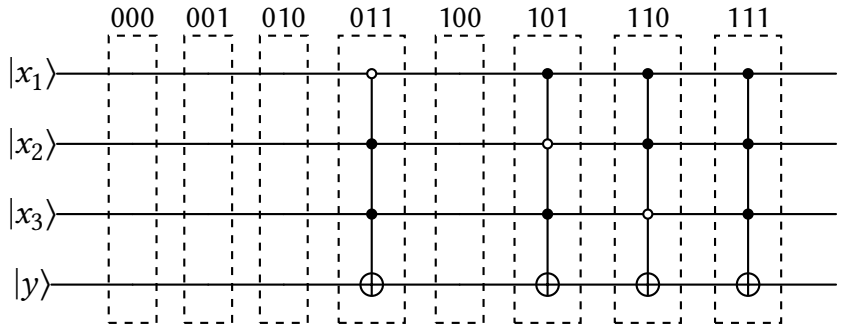
4.2.4 In general, such a box is called an *oracle*: it captures the (classical) structure of the problem instance. For instance, it can correspond to an arithmetic operation, or the neighboring relation for a graph, etc.

4.2.5 From 3.3, we know that regardless of its internal structure, the unitary U_f is realizable using CNOT and 1-qubit rotations. The procedure however requires in general to perform Cosine-Sine decompositions. In the case of U_f, we can rely on the fact that it was built from the classical, boolean function f.

4.2.6 In the most general situation, f is given as a truth table. As an example, consider the function $f : \mathbb{B}^3 \to \mathbb{B}$ defined as follows.

$x_1x_2x_3$	000	001	010	011	100	101	110	111
$f(x_1, x_2, x_3)$	0	0	0	1	0	1	1	1

Given the truth table, a simple way to build the circuit U_f is to add one multi-control for each value entry yielding 1, as follows.

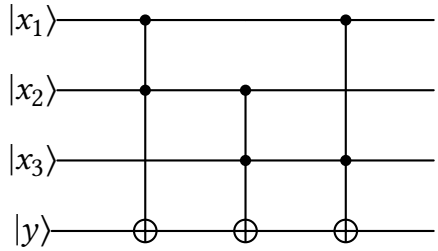


As you can easily see, this is not optimal in general: the size of the circuit is typically exponential on the number of inputs of f.

4.2.7 This function is the *majority function*: $f(x_1x_2x_3)$ is 0 if there are more 0's than 1 in $x_1x_2x_3$, and 1 otherwise. Instead of a truth table, the function can be described with the formula:

$$f(x_1, x_2, x_3) = (x_1x_2) \oplus (x_2x_3) \oplus (x_3x_1).$$

Compared to the circuit presented in 4.2.6, we can then do better with only 3 Toffoli gates:



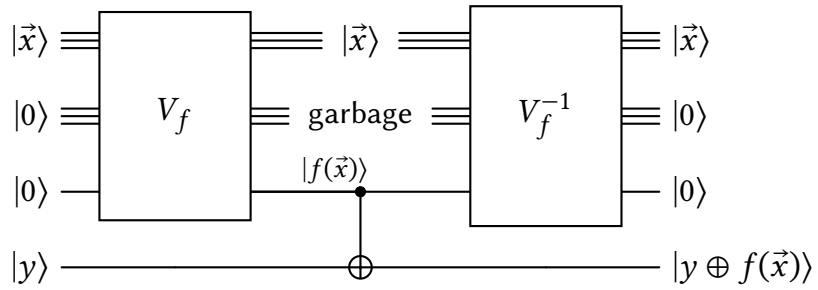
(34)

The idea behind this circuit is akin to what was described in 3.4.7: subformulas are computed and their results are stored for use later on. For the circuit of Eq. 34 we did not have to use auxiliary wires, but in general we can: this is the topic of 4.2.8.

4.2.8 If the Boolean function f is given as a boolean formula made of conjunctions and negations, using ancillas one can build a circuit of size linear to the size of the formula. The idea is that

- Conjunctions: implementable with Toffolis;
- Negations: implementable with X-gates and CNOT-gates;
- Composition: corresponds to circuit composition.

The procedure follows the description given in 3.4.7: we first *compute* the final result using a sub-circuit V_f, aggregating subcomputations in ancillas (these are called *garbage qubits*), we *copy* the result to the dedicated register, and we finally *uncompute* the ancillas:



(35)

4.2.9 The operator V_f has a strict specification: the input qubits should be left untouched (when in canonical basis state), the last wire should contain the result (when fed with $|0\rangle$), while the middle qubits (the *garbage qubits*) store the subcomputations. In particular, the circuit V_f sends canonical basis states to canonical basis states.

4.2.10 The construction presented in 4.2.8 is using the trick discussed in 2.9.8: we temporarily work inside a larger space to have more rooms, in this case to be able to store intermediary computations. If V_f is sending

$$|\vec{x}\rangle \otimes |0\rangle \otimes |0\rangle \longmapsto |\vec{x}\rangle \otimes |f_{\text{garbage}}(\vec{x})\rangle \otimes |f(vecx)\rangle$$

then the sequence of operations given in Eq. 35 is doing the following:

$$\begin{aligned}
&\sum_{\vec{x},y} \alpha_{\vec{x},y} \cdot |\vec{x}\rangle \otimes |y\rangle \\
\mapsto &\sum_{\vec{x},y} \alpha_{\vec{x},y} \cdot |\vec{x}\rangle \otimes |0\rangle \otimes |0\rangle \otimes |y\rangle && \text{(ancilla allocation)} \\
\mapsto &\sum_{\vec{x},y} \alpha_{\vec{x},y} \cdot |\vec{x}\rangle \otimes |f_{\text{garbage}}(\vec{x})\rangle \otimes |f(\vec{x})\rangle \otimes |y\rangle && \text{(applying } V_f\text{)} \\
\mapsto &\sum_{\vec{x},y} \alpha_{\vec{x},y} \cdot |\vec{x}\rangle \otimes |f_{\text{garbage}}(\vec{x})\rangle \otimes |f(\vec{x})\rangle \otimes |y \oplus f(\vec{x})\rangle && \text{(CNOT-gate)} \\
\mapsto &\sum_{\vec{x},y} \alpha_{\vec{x},y} \cdot |\vec{x}\rangle \otimes |0\rangle \otimes |0\rangle \otimes |y \oplus f(\vec{x})\rangle && \text{(applying } V_f^{-1}\text{)}
\end{aligned}$$

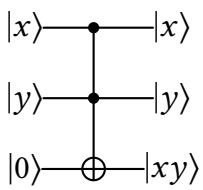
The middles qubits are back to $|0\rangle$, so there are not entangled with the rest of the system: if we discard them, following the discussion of 2.9, the result is deterministic and yield with probability 1

$$\sum_{\vec{x},y} \alpha_{\vec{x},y} \cdot |\vec{x}\rangle \otimes |y \oplus f(\vec{x})\rangle$$

which is precisely the behavior expected for U_f.

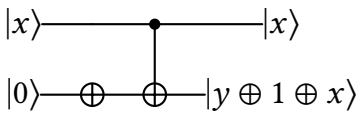
4.2.11 One can build the circuit for the operator V_f is built inductively on the structure of f, as follows:

- If f is the conjunction: $f(x,y) = x \wedge y = xy$, we build V_f with a Toffoli gate:



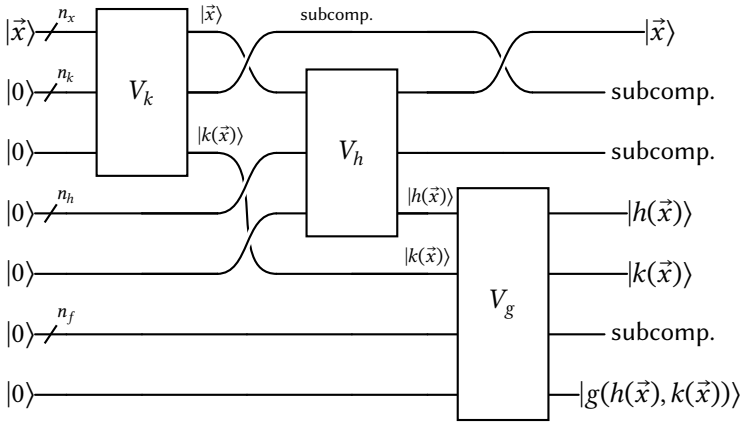
  It leaves the inputs untouched, while storing the $f(x,y)$ in the last wire if it was initialized with $|0\rangle$.

- If f is the negation: $f(x) = \neg x = 1 \oplus x$, we can builf V_f with a CNOT-gate:



  Note that we could have used a simple X-gate, but this would have modified the input qubit, therefore breaking the specification.

- Finally, if f is a composition of sub-formulas, such as $f(x) = g(h(x), k(x))$, one can build V_f out of V_g, and V_h and V_k:



(36)

  The first wires are set back to $|\vec{x}\rangle$, the last wire contains the result, while the middle wires contains all of the sub-computations, including the two values $g(\vec{x})$ and $k(\vec{x})$ computed along the composition.

4.2.12 Note how in Eq.(36) the input $|\vec{x}\rangle$ is used twice for V_h and V_k. This is why, in general, using this method one cannot use an X-gate to realize the negation: one has to keep keep the original value around. One can use another strategy to produce a more compact circuit, but this circuit construction is however efficient in the sense that the number of gates and auxiliary wires is linear on the size of the formula describing f.

4.2.13 Instead of a basis change, the operation U_f can also be regarded as a phase-flip, as follows:

$$\begin{aligned}
U_f |\vec{x}\rangle \otimes |-\rangle &= U_f \frac{1}{\sqrt{2}} |\vec{x}\rangle \otimes (|0\rangle - |1\rangle) \\
&= U_f \frac{1}{\sqrt{2}} (|\vec{x}\rangle \otimes |0\rangle - |\vec{x}\rangle \otimes |1\rangle) \\
&= \frac{1}{\sqrt{2}} (U_f(|\vec{x}\rangle \otimes |0\rangle) - U_f(|\vec{x}\rangle \otimes |1\rangle)) \\
&= \frac{1}{\sqrt{2}} (|\vec{x}\rangle \otimes |f(\vec{x})\rangle - |\vec{x}\rangle \otimes |1 \oplus f(\vec{x})\rangle) \\
&= \frac{1}{\sqrt{2}} |\vec{x}\rangle \otimes (|f(\vec{x})\rangle - |1 \oplus f(\vec{x})\rangle) \\
&= (-1)^{f(\vec{x})} \frac{1}{\sqrt{2}} |\vec{x}\rangle \otimes (|0\rangle - |1\rangle) \qquad \text{(From 1.10.2)} \\
&= (-1)^{f(\vec{x})} |\vec{x}\rangle \otimes |-\rangle .
\end{aligned}$$

4.3 Encoding Natural Numbers

4.3.1 A typical classical function to turn into a unitary consists in an arithmetic operation such as

$$\begin{array}{rccl} f_{a,N} : & \{0 \dots N-1\} & \longrightarrow & \{0 \dots N-1\} \\ & x & \longmapsto & a \cdot x \mod N \end{array}$$

for a and N natural numbers. If a and N are co-prime, the function is a bijection (this function is the oracle for Shor's algorithm, see 5.2). For instance, $f_{5,8}$ consists in the map

x	0	1	2	3	4	5	6	7
$f_{5,8}(x)$	0	5	2	7	4	1	6	3

The map $f_{5,8}$ is a bijection of $\{0, \dots, 7\}$: we are in the setting of 1.10.4 (where the map was $f_{3,8}$) and 4.2.2: the function can be regarded as a unitary on a 8-dimensional Hilbert space. However, the map acts on numbers whereas the discussion in 4.2 is based on Boolean values: we need a bit *encoding* of numbers as *bitstrings*.

4.3.2 A natural number x can be decomposed in *base 2* as

$$x = \sum_{k=0}^{N-1} b_k \cdot 2^k$$

for some number N and some Boolean values $b_0, \dots b_{N-1}$. We say that b_0 is the *least significant bit*. The Boolean values b_k's are enough to recover the number x, provided that we choose the ordering of the bits in the list. There are two canonical ones, that we can refer to as

- *big-endian*: the number x is represented with a list starting with the most significant bits:
$$b_{N-1}, \dots, b_1, b_0.$$

This is the mathematical representation (used in base 10 for instance), and used in most paper and textbooks in quantum computation.

- *little-endian*: the number x is represented with a list starting with the *least* significant bit:
$$b_0, b_1, \dots, b_{N-1}.$$
This is useful if the number is stored in an array: b_0 is then literally $b[0]$. This convention is used in QisKit for instance.

4.3.3 Beware In summary, there are at least two difficulties to look for when debugging quantum programs:

- conflicting ordering for binary representations;
- conflicting basis ordering for matrix and vector representations.

4.3.4 The domain and codomain of the map $f_{5,8}$ in 4.3.1 can be encoded on bitstrings of size 3. As discussed in 4.3.2, we can for instance decide on big-endian, so that 3 is represented by 011. The lexicographic ordering of the canonical basis on $\mathscr{H}^{\otimes 3}$ then exactly corresponds to the natural number ordering, and the matrix for $f_{5,8}$ can be constructed in the same way as in 1.10.4:

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

This matrix corresponds to the map $f_{5,8}$ acting on

$$\{|e_0\rangle, |e_1\rangle, |e_2\rangle, |e_3\rangle, |e_4\rangle, |e_5\rangle, |e_6\rangle, |e_7\rangle$$

as in 1.10.3, but also as the corresponding action on $\mathscr{H} \otimes \mathscr{H} \otimes \mathscr{H}$ with the basis ordering

$$|000\rangle, |001\rangle, |010\rangle, |011\rangle, |100\rangle, |101\rangle, |110\rangle, |111\rangle.$$

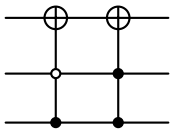
It performs the action:

$$\begin{array}{llll} |000\rangle \mapsto |000\rangle & |010\rangle \mapsto |010\rangle & |100\rangle \mapsto |100\rangle & |110\rangle \mapsto |110\rangle \\ |001\rangle \mapsto |101\rangle & |011\rangle \mapsto |111\rangle & |101\rangle \mapsto |001\rangle & |111\rangle \mapsto |011\rangle \end{array}$$

4.3.5 For the case of 4.3.4, it is possible to infer a circuit "by hand" by realizing that the map can be factored into

$$|x\rangle \otimes |01\rangle \mapsto |x'\rangle \otimes |01\rangle \quad |x\rangle \otimes |11\rangle \mapsto |x'\rangle \otimes |11\rangle$$

and identity otherwise. A circuit implementing the action is therefore



4.3.6 Example: Reversible Adder The oracle in 4.3.1 is reversible: it was therefore possible to build a circuit without ancillas as for instance shown in 4.3.5. In general however, the classical function we care about is not reversible: we therefore need a more sophisticated circuit. We discuss here how to encode an *adder*: a function inputting two numbers a and b and outputting $a+b$.

4.3.7 There are still several design choices for arithmetic on a bitstring encoding.

- We want a circuit: an *encoding* of our numbers as bitstrings. But which numbers? Integers (*signed* integers), natural numbers (*unsigned* integers)? On which range? Is it the same range for a and for b?
- How do we treat overflows? Is the range of the output expanded by one bit to take care of it, or do we consider arithmetic modulo instead?
- There are two "standard" ways to turn addition into a reversible operation. One can first turn the adder into a reversible function as in 4.2.3:

$$(\underline{a},\underline{b},z)\mapsto(\underline{a},\underline{b},z\oplus(\underline{a+b})) \tag{37}$$

  where it is understood that $\underline{n}$ is a bitstring encoding of the number n. Of course, we still need to characterize the size of each bitstring, and the actual behavior of the "+" operator (signed, unsigned, etc).

  The procedure shown in Eq. (37) does not modify the input registers a and b but requires an additional register z. If one does not need to retain the input register, for the addition one can follow a possibly more compact strategy and store the output in one of the registers:

$$(\underline{a},\underline{b})\mapsto(\underline{a},\underline{a+b}). \tag{38}$$

  Of course, this only makes sense if the register b is not used later on.

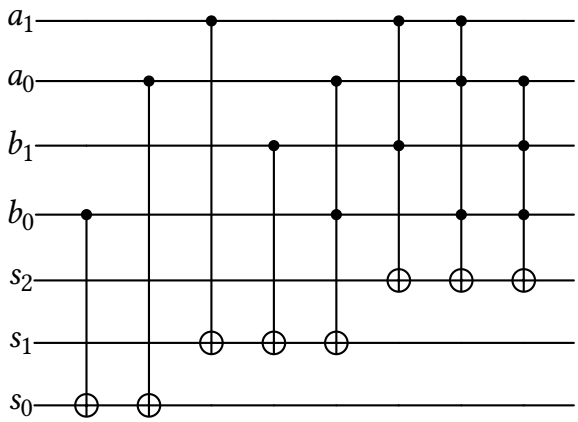
4.3.8 Consider an adder for numbers coded on bitstrings of size 2: $\underline{a}=a_1a_0$ and $\underline{b}=b_1b_0$ (with a_ks and b_ks Boolean values). We have

$$a=a_1\cdot 2+a_0,\qquad b=b_1\cdot 2+b_0.$$

With a bit of thinking one can show that

$$a+b=(a_1b_1\oplus a_0b_0a_1\oplus a_0b_0b_1)\cdot 2^2+(a_1\oplus b_1\oplus a_0b_0)\cdot 2+(a_0\oplus b_0) \tag{39}$$

The operator U_{add} as in 4.2.3 can be realized with the circuit



The answer is read in register s: we have $\underline{a+b} = s_2 s_1 s_0$.

4.3.9 The circuit presented in 4.3.8 was built "by hand". In order to go further, we have to *program* the adder. The simplest method is the *ripple-carry adder* procedure—the procedure followed when perfoming addition "by hand" on a sheet of paper. There are two cases:

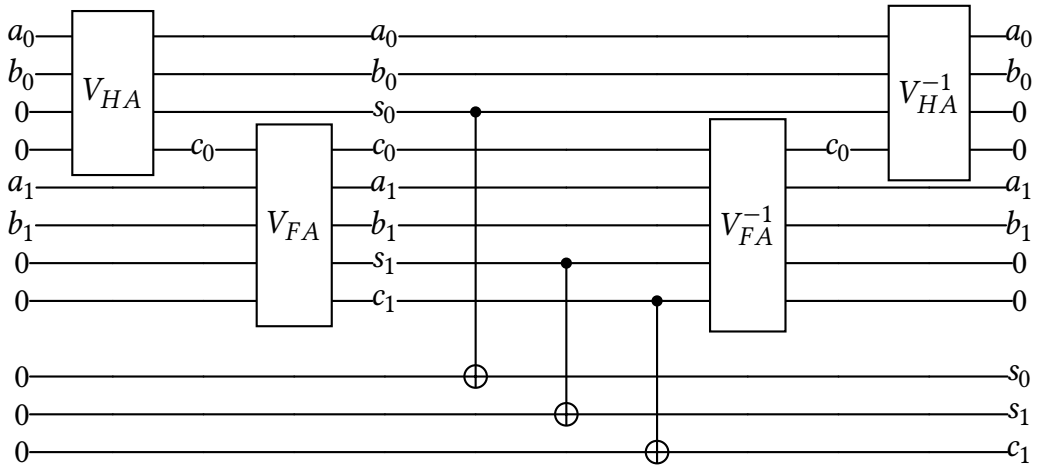
- Base case: adding a_0 and b_0, yieling a result s_0 and a *carry* c_0. This is a *half-adder*, and the formulas are
$$s_0 = a_0 \oplus b_0, \qquad c_0 = a_0 b_0. \tag{40}$$
- Inductive case: adding a_{k+1}, b_{k+1} and the carry c_k from the layer below, yielding a result s_{k+1} and a new carry c_{k+1}. This is a *full-adder*, and the formulas are
$$s_{k+1} = a_k \oplus b_k \oplus c_k, \qquad c_{k+1} = a_k b_k \oplus c_k a_k \oplus c_k b_k. \tag{41}$$

Unsurprisingly, from these formulas we can recover Eq. (39). Such a construction makes what is called a *ripple-carry adder*. The 2-bit adder of 4.3.8 can then be written in pseudo-code as

```
s0,c0 = HA(a0,b0)
s1,c1 = FA(c0,a1,b1)
return (c1,s1,s0)
```

where `HA` is a function coding the half-adder and `FA` is a function coding the full-adder. One can easily extend the procedure to any number of bits by chaining together additional full-adders.

Using the procedure described in 4.2.8, assuming V_{HA} and V_{FA} don't require any ancillas, we can write a reversible 2-bits adder as follows.



4.3.10 In term of circuit size, it is possible to show that $V_{HA} = 2 \cdot \text{CNOT} + C^2\text{-}X$ and $V_{FA} = 3 \cdot \text{CNOT} + 3 \cdot C^2\text{-}X$. For n-bits integers, the ripple-carry procedure yields a sumber of gates of

$$n \cdot V_{HA} + n \cdot V_{FA} + (n+1) \cdot \text{CNOT} = (6n+1) \cdot \text{CNOT} + (4n) \cdot C^2\text{-}X.$$

For the "naive" procedure of 4.3.8, we have for 2-bits integers: $4 \cdot \text{CNOT} + 4 \cdot C^2\text{-}X$. Compared to the ripple-carry adder, the computation $a_0 b_0$ is performed 2 times. For n-bits integers, we will also need $C^3\text{-}X$, $C^4\text{-}X$, ...$C^n\text{-}X$, and all sub-computations would be repeated several times. It in fact corresponds to unfolding Eqs (40) and (41): we can see that we would end up with a quadratic number of gates.

A last strategy to generate U_{add} would consists in using the decompositon summarized in 3.3.17. As discussed in 3.3.20, the size of the generated circuit would then be of the order of 4^N, where N is the size of the matrix. In our case, $N = 3n + 1$ (3 registers of size n and an additional wubit to store the final carry). The circuit is therefore of size $O(4 \cdot 64^n)$.

In summary, we are back to the trade-off discussion in 3.4. The ripple-carry produces a linear-sized circuit but with ancillas. It is possible to get rid of the ancillas, but at the expense of a circuit of larger size. However, in this particular case, the unitary is not arbitrary: we can rely on its internal structure to dodge an exponential asymptotic complexity.

4.4 Amplitude Amplification

4.4.1 A typical situation met in the design of quantum algorithms is the case where the quantum memory is the superposed state

$$\sum_{i=0}^{2^n-1} \alpha_i |i\rangle \otimes |f(i)\rangle$$

living in the space $\vec{H}^{\otimes n} \otimes \mathscr{H}$, where $f : \mathbb{B}^n \to \mathbb{B}$ states whether i is a solution to the problem.

4.4.2 For instance, consider the problem SquareNat:

- Input: a natural number N, with the promise that it is a square;
- Output: the square-root of N: the non-negative number a such that $a^2 = N$.

We can define f as follow: it inputs a bitstring $\vec{x}$ and reads it as the binary representation of a number j. The value $f(\vec{x})$ is then 1 if $j^2 = N$, and 0 otherwise.

4.4.3 A possible quantum algorithm to solve SquareNat can be:

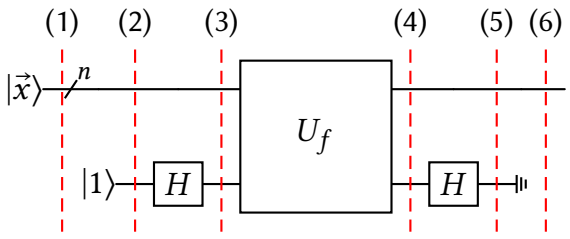
- Start with $|0 \dots 0\rangle \in \vec{H}^n \otimes \vec{H}$.
- Apply a tower of Hadamard on the nth first qubits to get
$$\frac{1}{\sqrt{2}^n} \sum_{j=0}^{2^n-1} |i\rangle \otimes |0\rangle .$$
- Apply U_f to get
$$\frac{1}{\sqrt{2}^n} \sum_{j=0}^{2^n-1} |i\rangle \otimes |f(i)\rangle .$$
- Measure: if the last bit is 1, we succeed, if not, we start again.

Provided that the promise "N is a square" is held, this algorithm eventually converge to a solution. However, because all of the coefficients are equal, the probability of success for each run is very low: $\frac{1}{2^n}$.

4.4.4 The procedure called *Amplitude Amplification*, or *Grover*'s algorithm (from the first person to have described it), aims at speeding up the process presented in 4.4.3 by augmenting the amplitudes of the "good states", i.e. the ones for which $f(i)$ is 1. For this, we need 3 operations on $\vec{H}^{\otimes n}$:

- an *oracle* $O : |x\rangle \mapsto (-1)^{f(x)} \cdot |x\rangle$;
- an operator $U_{0^\perp}$ sending $|0 \dots 0\rangle$ to $|0 \dots 0\rangle$ and every other canonical basis vector $|x\rangle$ to $-|x\rangle$;
- a tower of Hadamard $H^{\otimes n}$.

4.4.5 Note that O is not under the canonical form $U_f : |x\rangle \otimes |z\rangle \mapsto |x\rangle \otimes |z \oplus f(x)\rangle$, but one can build it from U_f using the circuit



(42)

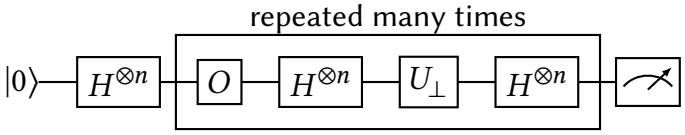
Starting from a canonical basis state $|\vec{x}\rangle$, we get:

$$
\begin{aligned}
&|\vec{x}\rangle && (1)\\
\mapsto &|\vec{x}\rangle \otimes |1\rangle && (2)\\
\mapsto &|\vec{x}\rangle \otimes |-\rangle && (3)\\
\mapsto &(-1)^{f(\vec{x})} |\vec{x}\rangle \otimes |-\rangle && (4) \text{ (from 4.2.13)}\\
\mapsto &(-1)^{f(\vec{x})} |\vec{x}\rangle \otimes |1\rangle && (5)
\end{aligned}
$$

The measurement operation on the second qubit is deterministic since the state is not entangled with the rest of the system. We then get at (6) desired state $(-1)^{f(\vec{x})} |\vec{x}\rangle$.

4.4.6 In Eq. 42, we measure the auxiliary register. If we have to repeat the circuit (and we will have to do so for Grover), we do not have to measure it: we can reuse it since it is back to the state $|1\rangle$. If we reuse it for the same circuit, we can even simplify it by not applying intermediary Hadamard gates.

4.4.7 Using the building blocks of 4.4.4, Grover's algorithm is as follows:



How many should "many times" be depends on f.

4.4.8 Let us denote $H^{\otimes n} U_{0^\perp} H^{\otimes n}$ with $U_\psi^\perp$. What is its action? Let us consider $|\psi\rangle = H^{\otimes n} |0 \dots 0\rangle$. Let $V_{|\psi\rangle}$ be the subspace generated by $|\psi\rangle$ and $V_{|\psi\rangle}^\perp$ the orthogonal subspace. We then have

$$H^{\otimes n} U_{0^\perp} H^{\otimes n} |\psi\rangle = |\psi\rangle$$

Indeed:

$$
\begin{aligned}
H^{\otimes n} U_{0^\perp} H^{\otimes n} |\psi\rangle &= H^{\otimes n} U_{0^\perp} H^{\otimes n} \left(H^{\otimes n} |0\rangle\right)\\
&= H^{\otimes n} U_{0^\perp} \left(H^{\otimes n} H^{\otimes n}\right) |0\rangle\\
&= H^{\otimes n} U_{0^\perp} Id |0\rangle\\
&= H^{\otimes n} U_{0^\perp} |0\rangle\\
&= H^{\otimes n} |0\rangle\\
&= |\psi\rangle .
\end{aligned}
$$

Now, if $|\phi\rangle \in V_{|\psi\rangle}^\perp$, the vector $H^{\otimes n} |\phi\rangle$ does not contain any instance of $|0 \dots 0\rangle$ in its canonical decomposition. Indeed, supposed that there were such an instance. We would then have $H^{\otimes n} |\phi\rangle = \alpha |0\rangle + \beta |0^\perp\rangle$ with $|0^\perp\rangle \perp |0\rangle$. So

$$
\begin{aligned}
H^{\otimes n} \left(H^{\otimes n} |\phi\rangle\right) &= H^{\otimes n} \left(\alpha |0\rangle + \beta |0^\perp\rangle\right)\\
&= \alpha H^{\otimes n} |0\rangle + \beta H^{\otimes n} |0^\perp\rangle
\end{aligned}
$$

$$= \alpha |\psi\rangle + \beta |\psi^\perp\rangle$$

with $|\psi^\perp\rangle \in V_{|\psi\rangle}^\perp$. But then $H^{\otimes n}\left(H^{\otimes n}|\phi\rangle\right) = |\phi\rangle$ is not in $V_{|\psi\rangle}^\perp$: contradiction. Therefore, $H^{\otimes n}U_{0^\perp}H^{\otimes n}|\phi\rangle = -|\phi\rangle$. Indeed, $U_{0^\perp}\left(H^{\otimes n}|\phi\rangle\right) = -H^{\otimes n}|\phi\rangle$ since $H^{\otimes n}|\phi\rangle$ does not contain any instance of $|0...0\rangle$, and is therefore orthogonal to $|0...0\rangle$. So $H^{\otimes n}U_{0^\perp}H^{\otimes n}|\phi\rangle = H^{\otimes n}\left(-H^{\otimes n}|\phi\rangle\right) = -H^{\otimes n}\left(H^{\otimes n}|\phi\rangle\right) = -|\phi\rangle$.

4.4.9 We are now ready to see how the algorithm is working. We start by buiding the state $|\psi\rangle$. It can be decomposed into

$$|\psi\rangle = |\psi_{good}\rangle + |\psi_{bad}\rangle \tag{43}$$

The former consists in the canonical basis kets for which f gives 1, the latter consists in the canonical basis kets for which f yields 0.

The state $|\psi_{good}\rangle$ can be decomposed into $\epsilon|\psi\rangle + \delta|\psi_\perp\rangle$, with $|\psi_\perp\rangle \in V_{|\psi\rangle}^\perp$. From Eq. (43) we derive that $|\psi_{bad}\rangle = |\psi\rangle - |\psi_{good}\rangle = (1-\epsilon)|\psi\rangle - \delta|\psi\rangle^\perp$. If there are not "too many" solutions for f, the amplitude ϵ is small.

Let us apply G to $|\psi\rangle$:

- we first apply O: it flips around the "bad" states and turns $|\psi\rangle = |\psi_{good}\rangle + |\psi_{bad}\rangle$ into $-|\psi_{good}\rangle + |\psi_{bad}\rangle$. , that is,

$$(1-2\epsilon)|\psi\rangle - 2\delta|\psi_\perp\rangle.$$

- We then apply $U_\psi^\perp$: it flips around the $|\psi\rangle$ axis and changes the sign of $|\psi_\perp\rangle$: we get

$$(1-2\epsilon)|\psi\rangle + 2\delta|\psi_\perp\rangle,$$

that is,

$$(3-4\epsilon)|\psi_{good}\rangle + (1-4\delta)|\psi_{bad}\rangle$$

If ϵ is small enough, the amplitude of "good" states got increased. We can draw a geometrical intuition as shown in Fig. 9: Two symmetries gives a rotation in the direction of the "good" states. If we iterate the process, we get closer and closer. If we do too much, we go too far and the amplitude starts decreasing.

4.4.10 One can compute the optimal number of iterations of G. If there are k values x such that $f(x) = 1$, the optimal is

$$r \sim \frac{\pi}{4}\sqrt{\frac{2^n}{k}}$$

and we get a quadratic speedup compared to a classical approach.

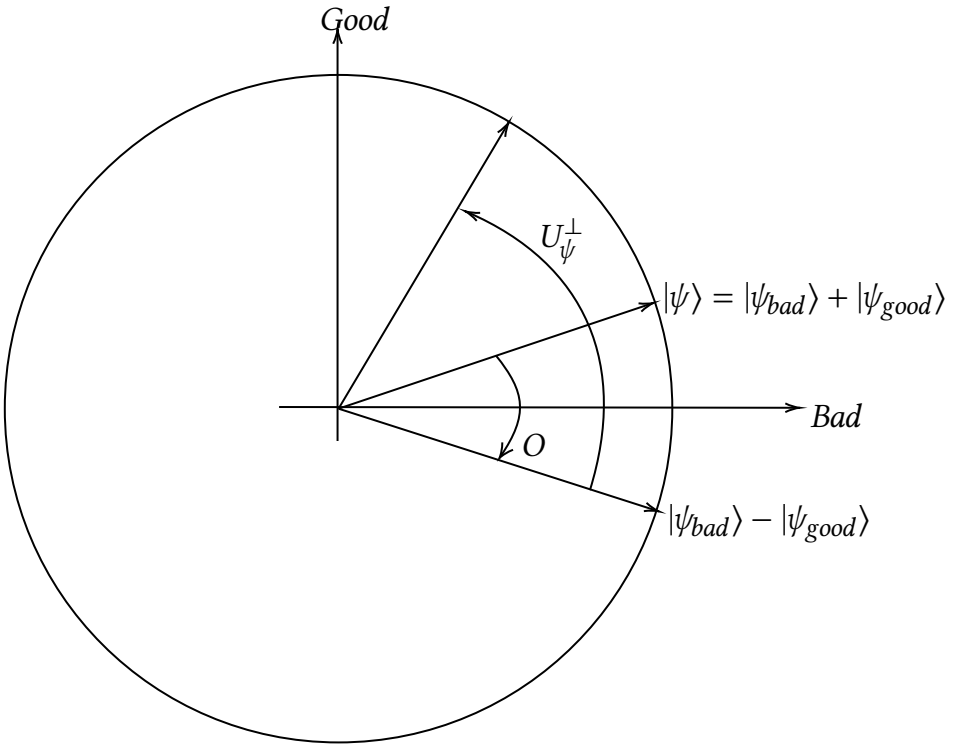


Figure 9: Geometric Intruition for Amplitude Amplification

4.5 Quantum Fourier Transform

4.5.1 The *Quantum Fourier Transform* (*QFT*) is a circuit that moves information between the phase and the canonical basis-ket. The circuit is the *QFT*, while the reversed one is the *QFT inverse*. The latter can presented as a problem to solve as follows. You are given a state on n qubits hiding a number $x \in \{0, \dots 2^n - 1\}$ as follows.

$$\frac{1}{\sqrt{2}^n} \sum_{k=0}^{2^n-1} \omega_x^k |k\rangle,$$

where

$$\omega_x = e^{i \cdot \frac{2\pi}{2^n} \cdot x}.$$

Can you (classically) retrieve the value of x?

4.5.2 The answer is yes: we can build a circuit computing the operation

$$\frac{1}{\sqrt{2}^n} \sum_{k=0}^{2^n-1} \omega_x^k |k\rangle \longmapsto |x\rangle$$

(with the least significant bit on the left in $|x\rangle$).

4.5.3 Let us try with $n = 1$: the hidden value x is then 0 ou 1. The state of the system can be rewritten as

$$\sum_{k=0}^{1} e^{\frac{2i\pi}{2}kx} |k\rangle = |0\rangle + (-1)^x |1\rangle .$$

To get back $|x\rangle$, an Hadamard gate is enough.

4.5.4 Let us try with $n = 2$. The hidden value x is then 0, 1, 2 or 3. In binary notation, $\lfloor x \rfloor_2 = x_1 x_2$, i.e. $x = 2x_1 + x_2$. The input state on two qubits is

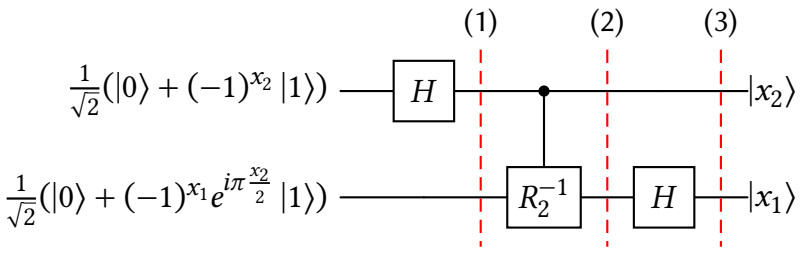
$$\begin{aligned}
&\sum_{k=0}^{2^n-1} e^{\frac{2i\pi}{2^n}kx} |k\rangle \\
=&\sum_{k=0}^{3} e^{2i\pi\left(\frac{x_1}{2}+\frac{x_2}{4}\right)k} |k\rangle \\
=&\sum_{k=0}^{3} e^{2i\pi\frac{x_1}{2}k} e^{2i\pi\frac{x_2}{4}k} |k\rangle \\
=&\sum_{k=0}^{3} e^{i\pi x_1 k} e^{i\pi\frac{x_2}{2}k} |k\rangle \\
=&\frac{1}{2}(|00\rangle + e^{i\pi x_1} e^{i\pi\frac{x_2}{2}} |01\rangle + e^{2i\pi x_1} e^{i\pi x_2} |10\rangle + e^{3i\pi x_1} e^{3i\pi\frac{x_2}{2}} |11\rangle) \\
=&\frac{1}{2}\left(|00\rangle + e^{i\pi x_1} e^{i\pi\frac{x_2}{2}} |01\rangle + e^{i\pi x_2} |10\rangle + e^{i\pi x_1} e^{3i\pi\frac{x_2}{2}} |11\rangle\right) \\
=&\frac{1}{2}\left(|0\rangle + e^{i\pi x_2} |1\rangle\right) \otimes \left(|0\rangle + e^{i\pi x_1} e^{i\pi\frac{x_2}{2}} |1\rangle\right) \\
=&\frac{1}{2}\left(|0\rangle + (-1)^{x_2} |1\rangle\right) \otimes \left(|0\rangle + (-1)^{x_1} e^{i\pi\frac{x_2}{2}} |1\rangle\right).
\end{aligned}$$

Note how the memory state is separable:

- Applying Hadamard on the first qubit, we retrieve $|x_2\rangle$.
- If it were not for the phase $e^{i\pi\frac{x_2}{2}}$, we could get $|x_1\rangle$ with an Hadamard. We first need to get rid of the phase, and this can be done with the controlled rotation $C\text{-}R_2^{-1}$, where

$$R_n = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{2\pi}{2^n}x_2} \end{pmatrix}.$$

Summarizing, to get back $|x_2 x_1\rangle$ one can use the circuit



(44)

At each step, the memory is changed towards the goal:

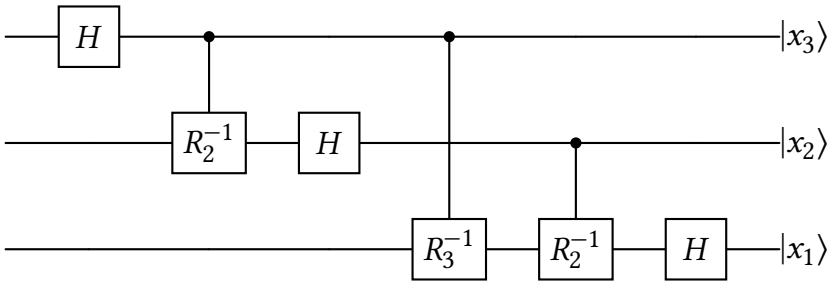
- at (1): $|x_2\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + (-1)^{x_1} e^{i\pi \frac{x_2}{2}} |1\rangle)$;
- at (2): $|x_2\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + (-1)^{x_1} |1\rangle)$;
- at 3: $|x_2\rangle \otimes |x_1\rangle$.

Note that the bits of x are read from right to left!

4.5.5 The situation carries over for larger n: the state of the system is in fact separable, and the bits of x can be recovered one by one, using more and more controlled rotations to correct for the additional phases. For instance, at $n = 3$ the state of the system is

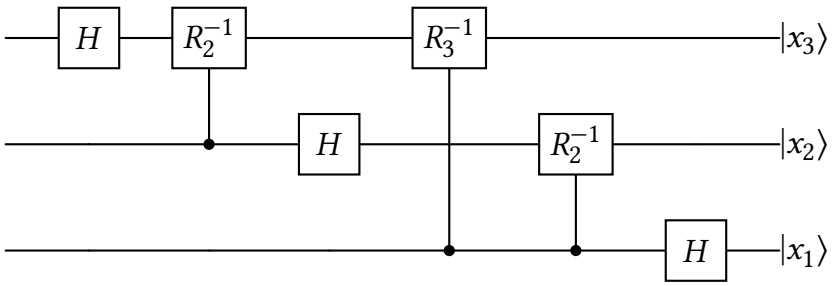
$$\frac{1}{\sqrt{2}^3}(|0\rangle + (-1)^{x_3}|1\rangle) \otimes \left(|0\rangle + (-1)^{x_2} e^{i\pi \frac{x_3}{2}} |1\rangle\right) \otimes \left(|0\rangle + (-1)^{x_1} e^{i\pi \frac{x_2}{2}} e^{i\pi \frac{x_3}{4}} |1\rangle\right).$$

One can recover $|x_3 x_2 x_1\rangle$ using the circuit



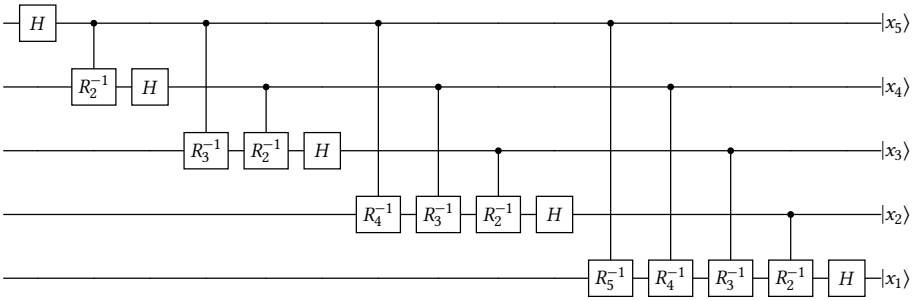
(45)

4.5.6 Remember 2.6.9: the circuit of Eq. (45) can be equivalently rewritten as



(46)

4.5.7 This structure generalizes to n qubits: the circuit called *QFT inverse* can be defined for all n in a similar manner, with blocks of increasing sizes. For instance, for $n = 5$ we get the circuit

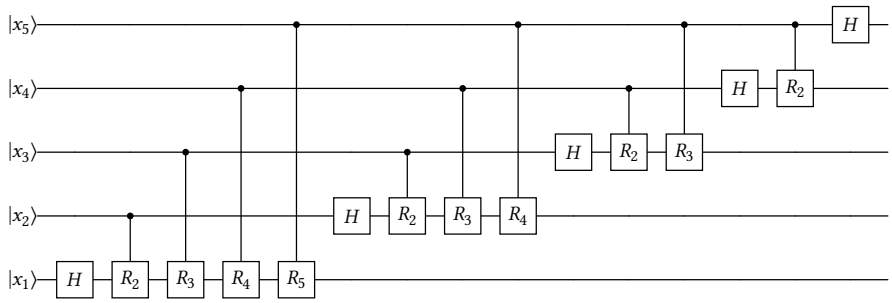


(47)

4.5.8 Reversing the circuit, we get the *Quantum Fourier Transform*, or *QFT*. It implements the map

$$|x_n \dots x_1\rangle \longmapsto \sum_{k=0}^{2^n-1} e^{\frac{2i\pi}{2^n} k \lfloor x_1 \dots x_n \rfloor_2} |k\rangle ,$$

where $\lfloor x_1 \dots x_n \rfloor_2 = \sum_{k=1}^{n} x_k 2^{n-k}$. For $n = 5$, the circuit is then the inverse of the one in Eq. (47):



(48)

4.6 Phase Estimation

4.6.1 Consider the problem PhaseEstimation, defined as follows

Input A unitary U, an eigenvector $|\psi\rangle$ and a natural number $n \in \mathbb{N}$.

Output The phase of the corresponding eigenvalue up to n bits of precision. ↳ 1.9.3

The algorithm *QPE* (for *Quantum Phase Estimation*) proposes a circuit for solving the problem.

4.6.2 The algorithm is based on the following property of $C\text{-}U$:

$$\begin{aligned} C\text{-}U\left((\alpha|0\rangle + \beta|1\rangle) \otimes |\psi\rangle\right) &= C\text{-}U\left(\alpha|0\rangle \otimes |\psi\rangle + \beta|1\rangle \otimes |\psi\rangle\right) \\ &= \alpha \cdot C\text{-}U(|0\rangle \otimes |\psi\rangle) + \beta \cdot C\text{-}U(|1\rangle \otimes |\psi\rangle) \\ &= \alpha \cdot (|0\rangle \otimes |\psi\rangle) + \beta \cdot (|1\rangle \otimes (U|\psi\rangle)) \\ &= \alpha \cdot (|0\rangle \otimes |\psi\rangle) + \beta \cdot (|1\rangle \otimes (e^{2i\pi\omega}|\psi\rangle)) \\ &= \alpha \cdot (|0\rangle \otimes |\psi\rangle) + \beta e^{2i\pi\omega} \cdot (|1\rangle \otimes |\psi\rangle) \\ &= (\alpha|0\rangle + \beta e^{2i\pi\omega}|1\rangle) \otimes |\psi\rangle . \end{aligned}$$

Note the maybe counterintuitive fact that on this particular shape of input, $C\text{-}U$ only changes the phase of the control qubit.

4.6.3 To understand how this works, let us consider that ω is $0.x_1x_2$ in binary form:

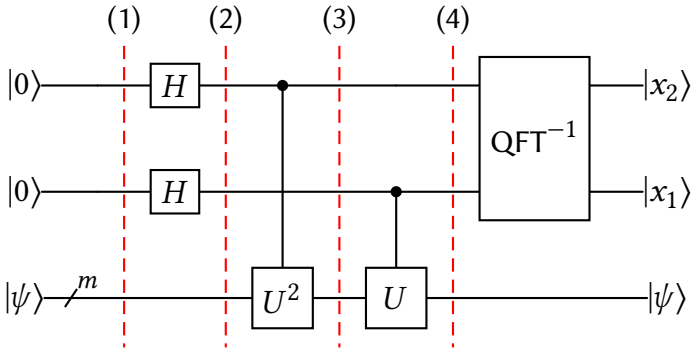
$$\omega = \frac{x_1}{2} + \frac{x_2}{4}.$$

Let $|\psi\rangle$ be the corresponding eigenvector. We then have

$$U|\psi\rangle = e^{2i\pi\omega}|\psi\rangle$$

$$= (-1)^{x_1} e^{i\pi \frac{x_2}{2}} |\psi\rangle,$$
$$U^2 |\psi\rangle = (-1)^{x_1} e^{i\pi \frac{x_2}{2}} (U |\psi\rangle)$$
$$= (-1)^{x_2} |\psi\rangle.$$

Note how the two coefficients are akin to the input of the circuit of Eq. (44). It is not exactly of the right form, but with the remark of 4.6.2, we can can derive the circuit



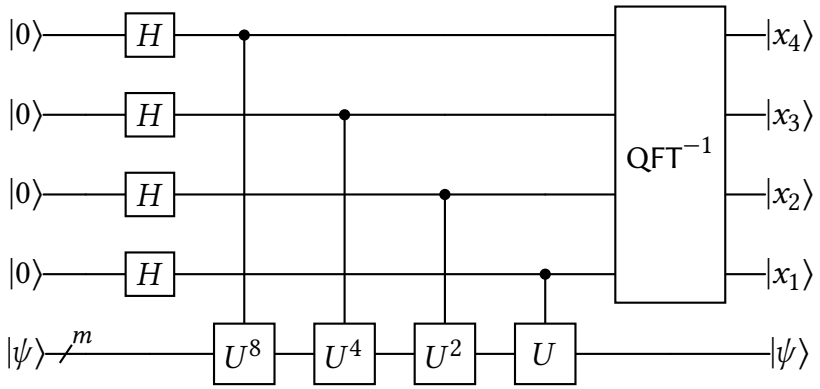
(49)

At each step, the state of the system is

1. $|0\rangle \otimes |0\rangle \otimes |\psi\rangle$.
2. $|+\rangle \otimes |+\rangle \otimes |\psi\rangle = \frac{1}{2}(|0\rangle + |1\rangle) \otimes (|0\rangle + |1\rangle) \otimes |\psi\rangle$.
3. $\frac{1}{2}(|0\rangle + (-1)^{x_2} |1\rangle) \otimes (|0\rangle + |1\rangle) \otimes |\psi\rangle$.
4. $\frac{1}{2}(|0\rangle + (-1)^{x_2} |1\rangle) \otimes \left(|0\rangle + (-1)^{x_1} e^{i\pi \frac{x_2}{2}} |1\rangle\right) \otimes |\psi\rangle$.

and at this step, the two first qubits are exactly the input of the circuit in Eq. (44): Applying the QFT inverse yields $|x_2 x_1\rangle \otimes |\psi\rangle$: we can retrieve ω with a measurement.

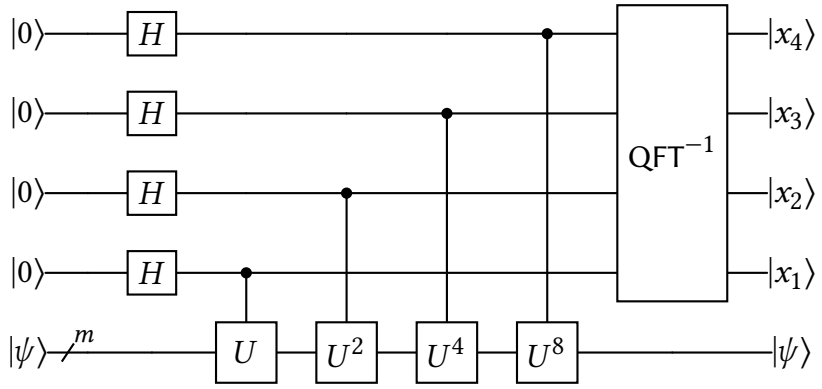
4.6.4 Once again, this generalizes. One can also show that this is also working (albeit probabilistically) if ω is not writable on precisely n bits. The circuit for 4 bits of precision is



(50)

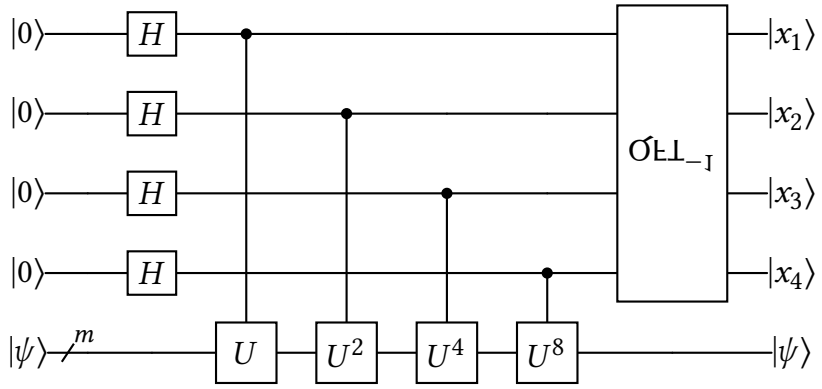
Note how the powers of U are *powers of* 2. Indeed, each power of U corresponds to one bit of ω.

4.6.5 Note how in Circ. (50) the $C\text{-}U^k$ commutes: the circuit can be equivalently written as



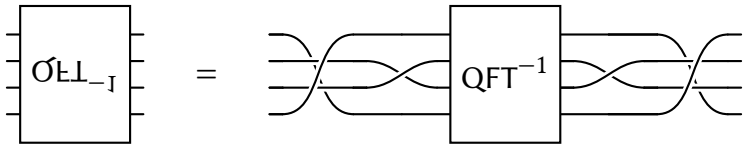
(51)

We can also reorder the top wires, *provided that the circuit QFT^{-1} is written upside down*:



(52)

with



(53)

4.6.6 Let us write QPE for the unitary realized by the circuit of 4.6.4. In particular

$$\mathrm{QPE}(|0000\rangle \otimes |\psi\rangle) = |x_4x_3x_2x_1\rangle \otimes |\psi\rangle\,.$$

Note how this operator is linear. Therefore, if $|\phi\rangle$ is another eigenvector with eivgenvalue $\lfloor 0.y_1y_2y_3y_4 \rfloor_2$, then

$$\mathrm{QPE}(|0000\rangle \otimes |\phi\rangle) = |y_4y_3y_2y_1\rangle \otimes |\phi\rangle\,,$$

and

$$\mathrm{QPE}\left(|0000\rangle \otimes \frac{1}{\sqrt{2}}(|\phi\rangle + |\phi\rangle)\right) = \frac{1}{\sqrt{2}}(|y_4y_3y_2y_1\rangle \otimes |\phi\rangle + |x_4x_3x_2x_1\rangle \otimes |\psi\rangle).$$

If we were to measure the first 4 qubits, we would get both $x_4x_3x_2x_1$ and $y_4y_3y_2y_1$ with probability $1/2$.

4.7 Trotterization

4.7.1 In QPE, and we shall again use it for VQE and QAPA in Ch. 6, given a Hermitian matrix A we need to derive a circuit for e^{itA}. An often used trick is called the *Trotter-Suzuki* decomposition. It states that when $\delta \to 0$,

$$e^{i\delta\cdot(A+B)} \simeq e^{i\delta\cdot A}e^{i\delta\cdot B} + O(\delta^2).$$

This is called the *Order 1 decomposition*. There are also higher-order decompositions with smaller errors, but they are out of scope for this course.

4.7.2 When A and B commute: $AB = BA$, the decomposition is exact and we have

$$e^{i\delta\cdot(A+B)} = e^{i\delta\cdot A}e^{i\delta\cdot B}.$$

4.7.3 If A and B do not commute, and if one cannot make δ small, one can still make use of the decomposition with the following trick:

$$\begin{aligned} & e^{i\delta\cdot(A+B)} \\ & \quad = e^{i\frac{\delta}{n}\cdot(A+B)+\cdots+i\frac{\delta}{n}\cdot(A+B)} \\ & \quad = \left(e^{i\frac{\delta}{n}\cdot(A+B)}\right)^n && (A+B \text{ commutes with itselt}) \\ & \quad \simeq \left(e^{i\frac{\delta}{n}\cdot A}e^{i\frac{\delta}{n}\cdot B}\right)^n + O(\delta^2/n) \end{aligned}$$

4.8 Exercises

4.8.1 Using controlled-rotations, and identifying the n-sized bitstring x with a natural number, implements the operation

$$|x\rangle \mapsto e^{2i\pi\frac{x}{2^n}}|x\rangle.$$

Explain how it works and your choice of encoding.

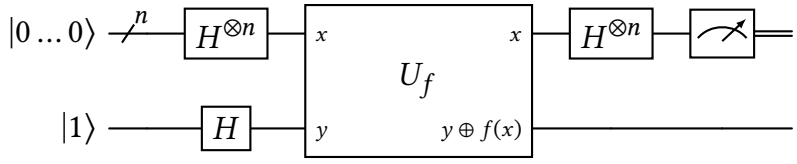
5 Algorithms for LSQ era

5.1 Simple Oracle-Based Algorithms

5.1.1 Deutsch-Josza algorithm. Suppose that we are given the set-function $f : \text{bool}^n \to \text{bool}$, with the promise that f is either constant or balanced (i.e. the sets of inputs mapping to 0 and 1 are of equal size). We are looking for an algorithm deciding on the status of f: is it constant or balanced? The catch is that f is givenas a *blackbox*: we only know how to call f, and we don't have any information on how it is built. For instance, you can consider f as a call to an external server. In such an *oracle-based* algorithm, we care about the complexity in term of calls to the oracle (the function f).

5.1.2 With a classical algorithm, the only thinf we can do is call f repeatedly on various inputs. We need $2^{n-1}+1$ calls to f: we might be very unlucky, and our classical procedure might only pick the inputs mapping to the same Boolean value for the 2^{n-1} first calls. So we really need one more to be sure that the function is indeed constant or balanced.

5.1.3 In the quantum case, we rely on the trick discussed in Section 4.2.1, and instead of f we use U_f. Deutsch-Josza algorithm is very simple: run the circuit below, and measure the n first qubits. f est constant if $|0...0\rangle$ was measured, and balanced otherwise. This requires one single run of the algorithm!



Of course, one could argue that this changes the algorithm since we somehow have to *build* the sub-circuit U_f out of f. But for the purpose of the measure of the oracle-complexity, this is irrelevant.

5.1.4 Case $n = 1$. Let us compute what happens when $n = 1$. There are 4 possible functions f: The two constant functions of value 0 and 1, the identity and the bit-flip. Originally, the state is

$$|0\rangle \otimes |1\rangle\,.$$

After applying the Hadamard gates, we get

$$\begin{aligned}&\frac{1}{2}\left(|0\rangle + |1\rangle\right) \otimes \left(|0\rangle - |1\rangle\right)\\ =&\frac{1}{2}\left(|00\rangle + |10\rangle - |01\rangle - |11\rangle\right).\end{aligned}$$

The oracle is applied:

$$\frac{1}{2}\left(\begin{array}{rl} & |0\rangle \otimes |0 \oplus f(0)\rangle + |1\rangle \otimes |0 \oplus f(1)\rangle \\ - & |0\rangle \otimes |1 \oplus f(0)\rangle - |1\rangle \otimes |1 \oplus f(1)\rangle \end{array}\right)$$

$$= \frac{1}{2}\begin{pmatrix} & |0\rangle \otimes |f(0)\rangle + |1\rangle \otimes |f(1)\rangle \\ - & |0\rangle \otimes |1 \oplus f(0)\rangle - |1\rangle \otimes |1 \oplus f(1)\rangle \end{pmatrix},$$

followed by yet another Hadamard gate on the first wire:

$$\begin{aligned} &\frac{1}{2\sqrt{2}}\begin{pmatrix} & (|0\rangle + |1\rangle) \otimes |f(0)\rangle + (|0\rangle - |1\rangle) \otimes |f(1)\rangle \\ - & (|0\rangle + |1\rangle) \otimes |1 \oplus f(0)\rangle - (|0\rangle - |1\rangle) \otimes |1 \oplus f(1)\rangle \end{pmatrix} \\ =&\frac{1}{2\sqrt{2}}\begin{pmatrix} & |0\rangle \otimes |f(0)\rangle + |0\rangle \otimes |f(1)\rangle \\ - & |0\rangle \otimes |1 \oplus f(0)\rangle - |0\rangle \otimes |1 \oplus f(1)\rangle \\ + & |1\rangle \otimes |f(0)\rangle - |1\rangle \otimes |f(1)\rangle \\ - & |1\rangle \otimes |1 \oplus f(0)\rangle + |1\rangle \otimes |1 \oplus f(1)\rangle \end{pmatrix} \\ =&\frac{1}{\sqrt{2}}\begin{pmatrix} & |0\rangle \otimes \frac{1}{2}(|f(0)\rangle + |f(1)\rangle - |1 \oplus f(0)\rangle - |1 \oplus f(1)\rangle) \\ + & |1\rangle \otimes \frac{1}{2}(|f(0)\rangle - |f(1)\rangle - |1 \oplus f(0)\rangle + |1 \oplus f(1)\rangle) \end{pmatrix}. \end{aligned} \tag{54}$$

5.1.5 Case $n = 1$, with contant f. Assume f is constant: there is some Boolean value b such that $f(x) = b$ for all x. The formula in Eq. (54) becomes

$$\begin{aligned} &\frac{1}{\sqrt{2}}\begin{pmatrix} & |0\rangle \otimes \frac{1}{2}(|b\rangle + |b\rangle - |1 \oplus b\rangle - |1 \oplus b\rangle) \\ + & |1\rangle \otimes \frac{1}{2}(|b\rangle - |b\rangle - |1 \oplus b\rangle + |1 \oplus b\rangle) \end{pmatrix} \\ =&\frac{1}{\sqrt{2}}\begin{pmatrix} & |0\rangle \otimes \frac{1}{2}(|b\rangle + |b\rangle - |1 \oplus b\rangle - |1 \oplus b\rangle) \\ + & |1\rangle \otimes \frac{1}{2}(0 - 0) \end{pmatrix} \\ =&|0\rangle \otimes \frac{1}{\sqrt{2}}(|b\rangle - |1 \oplus b\rangle). \end{aligned}$$

Measuring the first qubit yield 0 with probability 1, as claimed in Sec. 5.1.3.

5.1.6 Case $n = 1$, with non-constant f. If the function f is not constant, then it is either the identity or the bit flip. In both cases, we have $f(1) = 1 \oplus f(0)$. The formula in Eq. (54) then becomes

$$\begin{aligned} &\frac{1}{\sqrt{2}}\begin{pmatrix} |0\rangle \otimes \frac{1}{2}\begin{pmatrix} |f(0)\rangle + |1 \oplus f(0)\rangle \\ -|1 \oplus f(0)\rangle - |1 \oplus 1 \oplus f(0)\rangle \end{pmatrix} \\ + \\ |1\rangle \otimes \frac{1}{2}\begin{pmatrix} |f(0)\rangle - |1 \oplus f(0)\rangle \\ -|1 \oplus f(0)\rangle + |1 \oplus 1 \oplus f(0)\rangle \end{pmatrix} \end{pmatrix} \\ =&\frac{1}{\sqrt{2}}\begin{pmatrix} |0\rangle \otimes \frac{1}{2}\begin{pmatrix} |f(0)\rangle + |1 \oplus f(0)\rangle \\ -|1 \oplus f(0)\rangle - |f(0)\rangle \end{pmatrix} \\ + \\ |1\rangle \otimes \frac{1}{2}\begin{pmatrix} |f(0)\rangle - |1 \oplus f(0)\rangle \\ -|1 \oplus f(0)\rangle + |f(0)\rangle \end{pmatrix} \end{pmatrix} \\ =&\frac{1}{\sqrt{2}}\begin{pmatrix} |0\rangle \otimes 0 \\ + \\ |1\rangle \otimes \frac{1}{2}(2|f(0)\rangle - 2|1 \oplus f(0)\rangle) \end{pmatrix} \\ =&|1\rangle \otimes \frac{1}{\sqrt{2}}(|f(0)\rangle - |1 \oplus f(0)\rangle). \end{aligned}$$

Measuring the first qubit, we obtain 1 with probability 1.

5.1.7 Generalization to any n Through the Hadamard, the state $|0\dots0\rangle\otimes|1\rangle$ is sent to

$$\frac{1}{\sqrt{2}^{n+1}}\left(\sum_{k=0}^{2^n-1}|k\rangle\otimes|0\rangle-\sum_{k=0}^{2^n-1}|k\rangle\otimes|1\rangle\right)$$

and the action of the oracle gives

$$\frac{1}{\sqrt{2}^{n+1}}\left(\sum_{k=0}^{2^n-1}|k\rangle\otimes|f(k)\rangle-\sum_{k=0}^{2^n-1}|k\rangle\otimes|1\oplus f(k)\rangle\right). \tag{55}$$

If f is constant of Boolean value b, we get

$$\begin{aligned}&\frac{1}{\sqrt{2}^{n+1}}\left(\sum_{k=0}^{2^n-1}|k\rangle\otimes|b\rangle-\sum_{k=0}^{2^n-1}|k\rangle\otimes|1\oplus b\rangle\right)\\ =&\frac{1}{\sqrt{2}^{n}}\left(\sum_{k=0}^{2^n-1}|k\rangle\right)\otimes\frac{1}{\sqrt{2}}\left(|b\rangle-|1\oplus b\rangle\right),\end{aligned}$$

and the last tower of Hadamard yield

$$|0\dots0\rangle\otimes\frac{1}{\sqrt{2}}\left(|b\rangle-|1\oplus b\rangle\right).$$

Measuring, we indeed get 00000..000

5.1.8 Now, if f were balanced, we can partition the set of indices $\{0\dots2^n-1\}$ into $S_0\cup S_1$, with $S_0\cap S_1=\emptyset$, with $f(x)=b$ whenever $x\in S_b$. From $|0\dots0\rangle\otimes|1\rangle$, applying the Hadamard gates and the oracle, we get

$$\begin{aligned}&\frac{1}{\sqrt{2}^{n+1}}\left(\sum_{k=0}^{2^n-1}|k\rangle\otimes|f(k)\rangle-\sum_{k=0}^{2^n-1}|k\rangle\otimes|1\oplus f(k)\rangle\right)\\ =&\frac{1}{\sqrt{2}^{n+1}}\begin{pmatrix}\sum_{k\in S_0}|k\rangle\otimes|f(k)\rangle-\sum_{k\in S_0}|k\rangle\otimes|1\oplus f(k)\rangle\\ +\ \sum_{k\in S_1}|k\rangle\otimes|f(k)\rangle-\sum_{k\in S_1}|k\rangle\otimes|1\oplus f(k)\rangle\end{pmatrix}\\ =&\frac{1}{\sqrt{2}^{n+1}}\begin{pmatrix}\sum_{k\in S_0}|k\rangle\otimes|0\rangle-\sum_{k\in S_0}|k\rangle\otimes|1\rangle\\ +\ \sum_{k\in S_1}|k\rangle\otimes|1\rangle-\sum_{k\in S_1}|k\rangle\otimes|0\rangle\end{pmatrix}\\ =&\frac{1}{\sqrt{2}^{n}}\begin{pmatrix}\sum_{k\in S_0}|k\rangle\otimes\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)\\ +\ \sum_{k\in S_1}|k\rangle\otimes\frac{1}{\sqrt{2}}(|1\rangle-|0\rangle)\end{pmatrix}\\ =&\frac{1}{\sqrt{2}^{n}}\left(\sum_{k\in S_0}|k\rangle-\sum_{k\in S_1}|k\rangle\right)\otimes\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)\\ =&\frac{1}{\sqrt{2}^{n}}\left(\sum_{k\in S_0}(-1)^{f(k)}|k\rangle+\sum_{k\in S_1}(-1)^{f(k)}|k\rangle\right)\otimes\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)\end{aligned}$$

$$= \frac{1}{\sqrt{2}^n} \left(\sum_{k=0}^{2^n-1} (-1)^{f(k)} |k\rangle \right) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle)$$

Yielding, after the last Hadamard gates, to

$$|\psi\rangle \otimes \frac{1}{\sqrt{2}} (|1\rangle - |0\rangle)$$

with

$$|\psi\rangle = \frac{1}{2^n} \sum_{k=0}^{2^n-1} (-1)^{f(k)} \sum_{j=0}^{2^n-1} (-1)^{j \odot k} |j\rangle = \frac{1}{2^n} \sum_{j=0}^{2^n-1} \left(\sum_{k=0}^{2^n-1} (-1)^{f(k)+j\odot k} \right) |j\rangle$$

since $H^{\otimes n} |k\rangle = \sum_{j=0}^{2^n-1} (-1)^{j\odot k} |j\rangle$ (identifying indices with bitstrings, recall 1.10.8). We are interested in the coefficient of $|0 \dots 0\rangle$: it corresponds to $j = 0$, so it is

$$\left(\sum_{k=0}^{2^n-1} (-1)^{f(k)+0\odot k} \right) = \left(\sum_{k=0}^{2^n-1} (-1)^{f(k)} \right)$$

and since f is balanced, there are as many k for whuch $f(k) = 0$ as there are for which $f(k) = 1$: the coefficient is 0. Thus, we if were to measure $|\psi\rangle$, we cannot obtain $0 \dots 0$ since the corresponding probability is 0.

5.1.9 Bernstein-Vazirani. An algorithm following the same structure is *Bernstein-Vazirani*. The idea is the same as for Deutsch-Josza, except that this time you are told that the function f acting on a bitstring of size n is of the form $f(x_1, \dots, x_n) = (a_1 \dots a_n) \oplus (x_1 \dots x_n) = a_1 \wedge x_1 \oplus \dots \oplus a_n \wedge x_n$, for an unknown bitstring a. The question is to figure out this bitstring a hidden in the oracle. With a classical algorithm, n calls to f are needed... With Bernstein-Vazirani only one is needed!

5.1.10 The circuit is literally the same as for Deutsch-Josza, shown in Sec. 5.1.3. Right before the last tower of Hadamard, according to Eq. (55) in Sec. 5.1.7 the state is

$$\frac{1}{\sqrt{2}^n} \left(\sum_{k=0}^{2^n-1} (-1)^{f(k)} |k\rangle \right) \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) .$$

But now $f(k) = a \odot k$: when we apply the last tower of Hadamard (see Exercise 1.10.8), we get

$$|a\rangle \otimes \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) ,$$

and measuring gives back the bitstring a.

Factorization	
Input	A number N, product of two unknown primes.
Output	A divisor of N.
OrderFinding	
Input	Two integers N and a, co-primes.
Output	The period r of a, i.e. the smallest $r > 0$ such that $a^r \equiv 1 \mod N$.
PhaseEstimation	
Input	A unitary U and an eigenvector $\|\phi\rangle$.
Output	The corresponding eigenvalue.

Table 3: Each problem is reduced to the one below.

5.2 Shor

5.2.1 Maybe the first work to have placed the subject of quantum algorithms on the table is Shor's factoring algorithm [Sho97]. Although it is not known whether it is NP-complete or not, the fact is that we do not know of any classical algorithm able to factor a number $N = pq$ (p and q prime numbers) coded on n digit in a time polynomial on n. The problem is deemed hard enough that it is at the root of the main cryptographic protocol used to encrypt data over the internet: RSA.

5.2.2 Provided that we have at disposal a quantum co-processor with a large enough memory holding stable logical quantum qubits, factoring is at reach in polynomial time (at least theoretically). The algorithm is based on the concept —standard in complexity theory— of *polynomial reduction*: the "difficult" part of factorization is encoded in another problem that we know how to solve. This process in done in two steps: Factorisation is first reduced to the problem of ordre finding, and order finding is itself reduced to the problem of phase estimation: see Table 3.

5.2.3 Step 1: Reducing Factorization to OrderFinding. This really means: "If I know how to (efficiently) solve OrderFinding, I can easily factor an integer N". In fact, this part of the algorithm is purely classical, relying on mathematical properties. Suppose that I can solve OrderFinding. I am given $N = pq$ to factor. The algorithm proceeds as follows.

1. Select a number $1 < a < N$ at random. If it is not co-prime with N, we are done: we have a non-trivial factor.

2. Otherwise, it is co-prime with N. We then invoke our algorithm for OrderFinding. It outputs the smallest number r such that $a^r = 1 \mod N$.

3. Assume r is even and $a^{r/2} \neq -1 \mod N$. Because of mathematical properties (see Appendix A in Nielsen and Chuang for details), this happens with probability greater or equal to $1/2$: if it fails, start back at step 1.

4. We have $a^2 = (a^{r/2})^2 = 1 \mod N$, so $(a^{r/2} - 1)(a^{r/2} + 1) = 0 \mod N$. Thus, N divides $(a^{r/2} - 1)(a^{r/2} + 1)$.

5. N cannot divide $a^{r/2}+1$ since we assumed that $a^{r/2} \neq -1 \mod N$. It cannot divide $a^{r/2}-1$ either since that would make $a^{r/2} = 1 \mod N$, and r is was supposed to be the smallest such integer. Therefore, the only remaining possibility is that one of the factor of N divide $a^{r/2}-1$, and the other $a^{r/2}+1$.

6. These factors can be computed with $\gcd(N, a^{r/2} \pm 1)$.

5.2.4 Step 2: Solving OrderFinding using PhaseEstimation. Note that if a and N are co-primes, then $x \mapsto a \cdot x \mod N$ is a reversible function (it is a permutation of $\{0...N-1\}$, see 1.10.3). Build the unitary $U_a : |x\rangle \mapsto |a \cdot x \mod N\rangle$ (multiplucation modulo N) According to what we discussed in 4.2, we know that we can implement it efficiently. We claim that PhaseEstimation can be used on U_a to recover the order of a modulo N.

5.2.5 The claim is that a suitable eigenvector of U_a has an eigenvalue from which the order of a modulo N can be recovered. Let us consider several possibilities.

- $U_a |0 \dots 0\rangle = |0 \dots 0\rangle$: the vector $|0 \dots 0\rangle$ is an eigenvector of eigenvalue 1. We cannot derive anything from this.

- Let us compute:

$$\begin{aligned} U_a \left(\tfrac{1}{\sqrt{r}} \textstyle\sum_{k=0}^{r-1} |a^k \mod N\rangle \right) &= \tfrac{1}{\sqrt{r}} \textstyle\sum_{k=0}^{r-1} |a^{(k+1)} \mod N\rangle \\ &= \tfrac{1}{\sqrt{r}} \textstyle\sum_{k=1}^{r} |a^k \mod N\rangle \\ &= \tfrac{1}{\sqrt{r}} (|a^r \mod N\rangle + \textstyle\sum_{k=1}^{r-1} |a^k \mod N\rangle) \\ &= \tfrac{1}{\sqrt{r}} (|1 \mod N\rangle + \textstyle\sum_{k=1}^{r-1} |a^k \mod N\rangle) \\ &= \tfrac{1}{\sqrt{r}} (|a^0 \mod N\rangle + \textstyle\sum_{k=1}^{r-1} |a^k \mod N\rangle) \\ &= \tfrac{1}{\sqrt{r}} \textstyle\sum_{k=0}^{r-1} |a^k \mod N\rangle \end{aligned}$$

  We have another eigenvector of U_a, with eigenvalue 1. If this is still useless, we can nonetheless use this technique and add a phase to the various elements in the sum, in 5.2.6

5.2.6 Fix $s \in \{0, \dots, r-1\}$ and define

$$|\phi_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-2i\pi \frac{s \cdot k}{r}} |a^k \mod N\rangle$$

Let us compute:

$$U_a |\phi_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-2i\pi \frac{s \cdot k}{r}} U_a |a^k \mod N\rangle$$

$$
\begin{aligned}
&= \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-2i\pi \frac{s\cdot k}{r}} \left| a^{(k+1)} \bmod N \right\rangle \\
&= \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-2i\pi \frac{s\cdot (k-1)}{r}} \left| a^{k} \bmod N \right\rangle \\
&= \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{2i\pi \frac{s}{r}} e^{-2i\pi \frac{s\cdot k}{r}} \left| a^{k} \bmod N \right\rangle \\
&= e^{2i\pi \frac{s}{r}} \left| \phi_s \right\rangle
\end{aligned}
$$

This eigenvalue is better since it contains a phase parametrized by r. As QPE gives us the phase, with some luck we can retrieve r out. U_a have r such eigenvectors, one for each value of s between 0 and $r-1$. We would just have to use the fact that

$$
\left| 0...0 \right\rangle \otimes \left| \phi_s \right\rangle \xrightarrow{QPE(U_a)} \left| x_1...x_n \right\rangle \otimes \left| \phi_s \right\rangle
$$

where $x_1...x_n$ is a binary representation of the phase of the eigenvalue corresponding to $|\phi_s\rangle$, from which one could infer r.

5.2.7 The problem is that one cannot directly use these eigenvectors $|\phi_s\rangle$ since we need to know r to construct them. However, what we can do is use the fact that the QPE is a linear map, so we can place them in superposition:

$$
\left| 0...0 \right\rangle \otimes \left(\alpha \left| \phi_s \right\rangle + \beta \left| \phi_{s'} \right\rangle \right) \xrightarrow{QPE(U_a)} \alpha \left| x_1...x_n \right\rangle \otimes \left| \phi_s \right\rangle + \beta \left| y_1...y_n \right\rangle \otimes \left| \phi_{s'} \right\rangle
$$

(where $y_1...y_n$ is a binary representation of the phase of the eigenvalue corresponding to $|\phi_{s'}\rangle$). The trick consists in realizing that if we place them all in (equal) superposition, as follows:

$$
\begin{aligned}
\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} \left| \phi_s \right\rangle &= \frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-2i\pi \frac{s*k}{r}} \left| a^{k} \bmod N \right\rangle \\
&= \frac{1}{r} \sum_{k=0}^{r-1} \sum_{s=0}^{r-1} e^{-2i\pi \frac{s*k}{r}} \left| a^{k} \bmod N \right\rangle \\
&= \frac{1}{r} \sum_{k=0}^{r-1} \left(\sum_{s=0}^{r-1} e^{-2i\pi \frac{s*k}{r}} \right) \left| a^{k} \bmod N \right\rangle \qquad (56)
\end{aligned}
$$

then the inner (red) sum is equal to r if $k = 0$, and 0 otherwise (because it is a sum of all of the roots of unity). Therefore, in Eq (56) all the terms are nul except when $k = 0$: we get

$$
\frac{1}{\sqrt{r}} \sum_{s=0}^{r-1} \left| \phi_s \right\rangle = \frac{1}{r} \left(r \left| a^0 \bmod N \right\rangle \right) = \left| 1 \right\rangle_n
$$

(where the encoding of 1 on n qubits is $|0...01\rangle$).

If we run QPE(U_a) on this input, we "compute" all of the phases at once. Consider two registers, the first one for retrieving the ω of the eigenvalue and the second one for the eigenvector. We have

$$QPE(U_a)\left(|0...0\rangle \otimes |\phi_s\rangle\right) = |s/r\rangle \otimes |\phi_s\rangle$$

where by $|s/r\rangle$ we mean the approximation of s/r over the corresponding number of qubits. Then

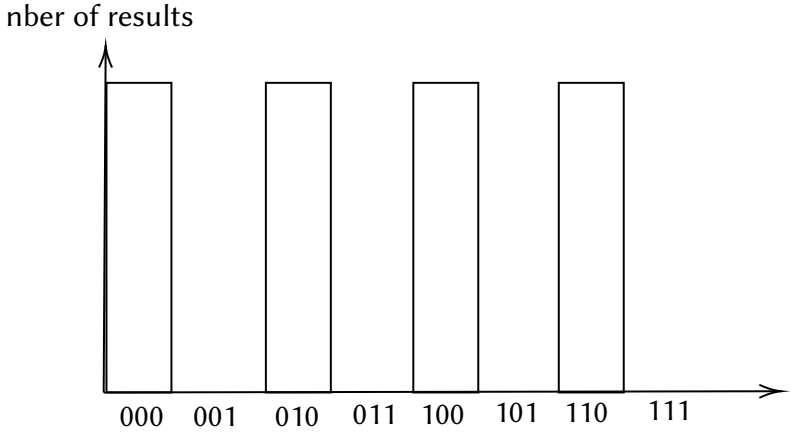
$$\begin{aligned} QPE(U_a)\left(|0...0\rangle \otimes \frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}|\phi_s\rangle\right) &= \frac{1}{\sqrt{r}}\sum_{s=0}^{r-1} QPE(U_a)\left(|0...0\rangle \otimes |\phi_s\rangle\right) \\ &= \frac{1}{\sqrt{r}}\sum_{s=0}^{r-1}|s/r\rangle \otimes |\phi_s\rangle \end{aligned}$$

Measuring the first register, we get one of the $\frac{s}{r}$ (for the sake of the discussion, assume that the decomposition on n bits is exact)

For instance, if $r = 4$, there are exactly 4 elements in the sum: measuring, we get $0/4$, $1/4$, $2/4$ and $3/4$.

- On 2 bits, this is 00, 01, 10 and 11.
- On 3 bits, this is 000, 010, 100 and 110 (since $0.x_1x_2x_3 = \frac{x_1}{2} + \frac{x_2}{4} + \frac{x_3}{8}$)

If we were to perform many measurements (for 3 bits) and collecting the results, we would get the following plot



with equiprobable results 000, 010, 100 and 110. If the decomposition were not exact (for instance when $r = 3$), we would instead get a less precise plot with 3 peaks but not as sharp. Possibly then 3 bits would not be enough to distinguish them, and we would need to get to 5 or 6 bits of precision.

In any case, when the precision is high enough, one can "read out" the period r of a mod N from the plot, if we were to run enough computation.

5.2.8 However, in a concrete use-case we cannot afford to perform enough computations to draw such a plot. Instead, Shor's algorithm is run once, and then one retrieves a possible estimate for s/r, one uses the algorithm of continued fractions.[8] to get a putative value r, and one tests whether this gives a factor of N. If not, we start over. With a high-enough probability, this succeeds.

5.3 HHL

5.3.1 A slightly more involved algorithm relying on QPE is *HHL*, initials of the author's names: *Harrow, Hassidim, Lloyd* [HHL09]. This algorithm solves a linear system of equation with a complexity arguably better than the one offered by classical algorithms.

5.3.2 The problem can be stated as follows. Consider an hermitian matrix A and a vector $\vec{b}$: we want to solve the equation

$$A \cdot \vec{x} = \vec{b}.$$

Note that if A were not hermitian, we can reduce the problem to the case where the matrix is

$$\begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix}$$

(see 5.4.1)

5.3.3 To make use of a quantum co-processor, the idea is to code the vectors $\vec{b}$ and $\vec{x}$ as the coefficients of a ket-vector. For instance assume that $\vec{b}$ is the vector

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}.$$

The vector is stored in a 2-qubit register as

$$|b\rangle = b_0 |00\rangle + b_1 |01\rangle + b_2 |10\rangle + b_3 |11\rangle$$

modulo some renormalization. In the rest, we assume that $\vec{b}$ is of norm 1.

5.3.4 Complexity-wise, the algorithm is better than classical ones in the following sense. If

- N is the size of the system
- s is the number of non-zero elements in a line of A
- κ is the condition number of A (i.e. the ration between the largest and the smallest eigenvalue of A)

[8] See e.g. `https://en.wikipedia.org/wiki/Continued_fraction#Best_rational_approximations`

- ϵ the allowed error

then the complexity are

- In the classical case: $\mathcal{O}(Ns\kappa \log(1/\epsilon))$
- In the quantum case: $\mathcal{O}\left(\log(N)s^2\kappa^2/\epsilon\right)$ for HHL.

In summary, we get an exponential gain with respect to the size of the matrix. In theory.

5.3.5 General idea. Since the matrix A is hermitian, according to 1.9.4 it can be written as

$$A = \sum_{j=0}^{N-1} \lambda_j |u_j\rangle \langle u_j|$$

with the λ_js being real numbers and $\{|u_j\rangle\}_j$ an orthonormal basis. Provided that none of the λ_j are zero (which would make the condition number infinite), we can therefore safely consider that A admits an inverse, and in fact

$$A^{-1} = \sum_{j=0}^{N-1} \lambda_j^{-1} |u_j\rangle \langle u_j|$$

Now, $|\vec{b}\rangle$ can be decompose in the basis $\left\{|u_j\rangle\right\}_j$. For instance,

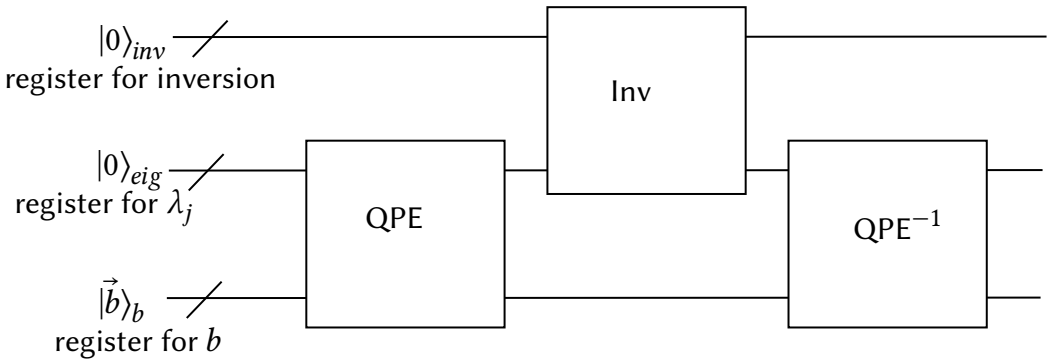
$$|\vec{b}\rangle = \sum_{j=0}^{N-1} b_j |u_j\rangle$$

with $b_j \in \mathbb{C}$. We can then write $|x\rangle$ as

$$|\vec{x}\rangle = A^{-1} |\vec{b}\rangle = \sum_{j=0}^{N-1} \lambda_j^{-1} b_j |u_j\rangle .$$

5.3.6 Can we consider that we solve the problem? If this mathematical development gives us a formal presentation of $|x\rangle$, it does not say how to *compute* the various pieces: it only say that "they exist" and that when combined they give a solution $|x\rangle$ to the problem. The objective of the algorithm HHL is to provide a computational mean to attain such a $|x\rangle$. It is however important to emphasize right away that HHL will *not* derive the $|u_j\rangle$s, the b_is and the λ_js. It will only rely on implicit mechanisms that manipulate them, without having to spell them out. At the end of the computation, a register will be set in state $|x\rangle$, solution to the problem. But everything will happen implicitely.

5.3.7 Structure of the circuit. We need two parameters, real values to calibrate the system: t and C. Their mean will be explained later on. The structure of the circuit is as follows.



The register inv contains a single qubit. The register b contains n bits, when $N = 2^n$. The size of the register eig depends on the desired precision.

The sub-circuit QPE stands for the quantum phase estimation as in 4.6.1, applied to the unitary e^{itA}. The parameter t aims at ensuring that the eigenvalues of tA are "not too large" and that QPE succeeds. The subcircuit Inv stands for any circuit implementing the operation

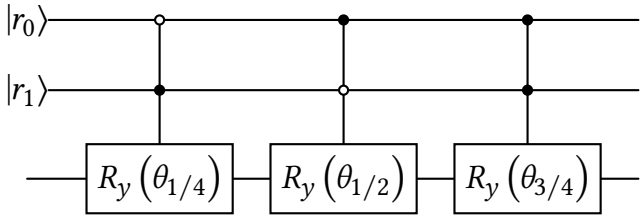
$$\text{Inv} \quad : \quad |0\rangle_{inv} \otimes |r\rangle_{eig} \longmapsto \left(\sqrt{1 - \frac{C^2}{r^2}} \, |0\rangle_{inv} + \frac{C}{r} \, |1\rangle_{inv} \right) \otimes |r\rangle_{eig}$$

where C is chosen small enough so that

- this makes sense: we need $\frac{C}{r}$ to be within 0 and 1 for the values of r we care about (see below).
- yet the amplitudes corresponding to the subspace $|1\rangle_{inv}$ are as large as possible (so that Step 5 in 5.3.10 succeeds with the highest probability).

5.3.8 The operation Inv. If we define the angle θ_r as $2 \arcsin(C/r)$, the action of Inv is "just" a rotation $R_y(\theta_r)$ parameterized by r. In 4.2, instead of a rotation R_y the action was a bit-flip: we can suggest two methods to realize a circuit for Inv replacing the bit-flip with suitable rotations. ↳ 2.5.7

Following the strategy in 4.2.6, if the eig register holds two qubits, and if we consider that $|r_0 r_1\rangle$ corresponds to the real value $\frac{r_0}{2} + \frac{r_1}{4}$, the value r can take 4 values: 0, $1/4$, $1/2$ and $3/4$, corresponding respectively to the states $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$. The circuit is then



The size of the circuit is however exponential on the size of the register eig. Of we can afford auxiliary qubits, one can rely on the structure of the function θ to build a circuit V_θ as in 4.2.8, and instead produce a circuit of size polynomial on the size of the register eig (albeit with a high overhead).

5.3.9 Beware! The problem is always the same: the encoding of natural numbers (or, for that matter, real numbers) on bitstrings relies on seemingly arbitrary conventions (see 4.3.2). When implementing an algorithm, one has to be careful about choosing the same notation for all of its subparts. Here, we have to choose the same ones for QPE and Inv.

5.3.10 Overview of the algorithm.

1. At first we have $|0\rangle_{inv} \otimes |0\rangle_{eig} \otimes |\vec{b}\rangle_b = \sum_j b_j |0\rangle_{inv} \otimes |0\rangle_{eig} \otimes |u_j\rangle_b$
2. We apply QPE with the matrix

$$U = e^{iAt} = \sum_{j=0}^{N-1} e^{i\lambda_j t} |u_j\rangle \langle u_j| = \sum_{j=0}^{N-1} e^{2i\pi \frac{\lambda_j t}{2\pi}} |u_j\rangle \langle u_j|$$

   (remember: QPE recovers the phase ω of an eigenvalue $e^{2i\pi\omega}$).

   Note how we use here the calibration parameter t. Its purpose is to adjust the values λ_js to have them fit inside the register eig and minimizing the error.

   Assuming that there are no precision error, we now have

$$\sum_j b_j |0\rangle_{inv} \otimes |\frac{\lambda_j t}{2\pi}\rangle_{eig} \otimes |u_j\rangle_b$$

3. We then use the sub-circuit Inv discussed in 5.3.7 and whose action is

   $\text{Inv}: |0\rangle_{inv} \otimes |r\rangle_{eig} \mapsto \left(\sqrt{1 - \frac{C^2}{r^2}} |0\rangle_{inv} + \frac{C}{r} |1\rangle_{inv}\right) \otimes |r\rangle_{eig}$

   This uses the second calibration parameter: the value r is potentially very small: we need to renormalize to get to a number between 0 and 1.

   In any case, after the action of Inv, we have

$$\sum_j b_j \left(\sqrt{1 - \frac{(2\pi C)^2}{(\lambda_j t)^2}} |0\rangle_{inv} + \frac{2\pi C}{\lambda_j t} |1\rangle_{inv}\right) \otimes |\frac{\lambda_j t}{2\pi}\rangle_{eig} \otimes |u_j\rangle_b$$

4. We then apply the inverse of the first QPE circuit: this uncomputes the $|\lambda_j t\rangle$s. We have

$$\sum_j b_j \left(\sqrt{1 - \frac{(2\pi C)^2}{(\lambda_j t)^2}} |0\rangle_{inv} + \frac{2\pi C}{\lambda_j t} |1\rangle_{inv}\right) \otimes |0\rangle_{eig} \otimes |u_j\rangle_b$$

   which is

$$\sum_j b_j \sqrt{1 - \frac{(2\pi C)^2}{(\lambda_j t)^2}} |0\rangle_{inv} \otimes |0\rangle_{eig} \otimes |u_j\rangle_b + \sum_j b_j \frac{2\pi C}{\lambda_j t} |1\rangle_{inv} \otimes |0\rangle_{eig} \otimes |u_j\rangle_b$$

$$= \sum_j b_j \sqrt{1 - \frac{(2\pi C)^2}{(\lambda_j t)^2}} |0\rangle_{inv} \otimes |0\rangle_{eig} \otimes |u_j\rangle_b + \frac{2\pi C}{t} \sum_j \frac{b_j}{\lambda_j} |1\rangle_{inv} \otimes |0\rangle_{eig} \otimes |u_j\rangle_b$$

$$= \sum_j b_j \sqrt{1 - \frac{(2\pi C)^2}{(\lambda_j t)^2}} |0\rangle_{inv} \otimes |0\rangle_{eig} \otimes |u_j\rangle_b + \frac{2\pi C}{t} |1\rangle_{inv} \otimes |0\rangle_{eig} \otimes \sum_j \frac{b_j}{\lambda_j} |u_j\rangle_b$$

5. We then measure the register inv in the canonical basis.

   In the case where the result is 1, accounting for the renormalization factor

   $$\eta = \frac{t}{2\pi C \sqrt{\sum_j \frac{b_j^2}{\lambda_j^2}}}$$

   we have

   $$|1\rangle_{inv} \otimes |0\rangle_{eig} \otimes \sum_j \frac{\eta b_j}{\lambda_j} |u_j\rangle_b .$$

   We therefore get the $|\vec{x}\rangle$ in a state corresponding to $\vec{x}$, modulo the renormalization factor η.

6. In case we are interested in η, this value can be recovered from the probability to measure 1 in Step 5.

5.3.11 This algorithm follows a *post-selection* strategy: we only know if we succeed after the measure of the register inv. If we get 0, we failed and we have to start over.

5.3.12 One remaining question is: what can we do with $|\vec{x}\rangle$? The coefficients are not classically available, and recovering them is costly. The HHL algorithm offers a use-case: instead of trying to recover these coefficients, one might want to compute the scalar product of $\vec{x}$ with a third vector $\vec{r}$. This can be done with the quantum co-processor without having to extract the coefficients of $\vec{x}$...See the original paper [HHL09] for details.

5.3.13 An example. A working example is

$$A = \begin{pmatrix} 1 & -1/3 \\ -1/3 & 1 \end{pmatrix}$$

and

$$\vec{b} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

The solution $\vec{x}$ is

$$\begin{pmatrix} 9/8 \\ 3/8 \end{pmatrix}.$$

To encode the problem we only need one single qubit for $\vec{b}$, and its state is $|\vec{b}\rangle_b = |0\rangle_b$.

For the parameters, we can choose $t = 2\pi\frac{3}{8}$ et $C = \frac{1}{4}$: the calculations add up. We pick a register eig with two qubits, since

$A\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{2}{3}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and then $\lambda_1 = 2/3$.

$A\begin{pmatrix} 1 \\ -1 \end{pmatrix} = \frac{4}{3}\begin{pmatrix} 1 \\ -1 \end{pmatrix}$ and then $\lambda_2 = 4/3$.

This means that $\frac{\lambda_1 t}{2\pi} = 1/4$ and $\frac{\lambda_2 t}{2\pi} = 1/2$. Therefore, with two bits one can store each of these values in binary as $0.b_1b_2 \equiv \frac{b_1}{2} + \frac{b_2}{4}$. Since $1/4$ is 0.01 in binary and $1/2$ is 0.10 in binary, the expected register states for eig are $|01\rangle$ and $|10\rangle$, fitting on two qubits.

If you follow my lecture, you might be asked to do a lab-session demonstrating how the algorithm works on this contrived example.

5.4 Exercises

5.4.1 Recall 5.3.2: When A is not Hermitian, show how one can recover a solution for the equation $A \cdot \vec{x} = \vec{b}$ using HHL, by considering instead

$$\begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix}.$$

6 Algorithms for NISQ era

6.1 Variational Algorithms

6.1.1 In Shor's algorithm, we build one single circuit, once for all. The circuit depends on the size of the input (we would not use the same circuit to factor 15 and to factor 100970708303), but once te circuit is built, it is used over and over again without change, until the algorithm succeeds. HHL or Grover are similar: a circuit is carefully crafted and then used over and over.

6.1.2 Another class of algorithms are the *variational* algorithms. In such a model, the circuit is updated each time we need to use it again. The idea is to refine the circuit to get closer and closer to the solution to the problem. They can be regarded as optimization techniques: instead of getting closer and closer to an optimal ket-vector (which we cannot manipulate directly), the algorithm optimize a circuit computing the desired ket-vector.

6.1.3 Variational algorithms are particularly well-suited for the *NISQ* regime of quantum computation (*Noisy Intermediate Scale Quantum*, see 3.8.2): they usually do not require very large quantum memories, they are very flexible in term hardware, and they are arguably resilient to noise.

6.2 VQE

6.2.1 Maybe the most typical variational algorithm is *VQE*, whose initials stand for *Variational Quantum Eigensolver*. Its objective is to find the extreman eigenvectors of an Hermitian matrix, and it perfectly embodies Feynman's intuition: using a quantum physical system to compute quantum properties [Fey82]. Indeed, Hermitian matrices (or *Hamiltonian*) are typically used to encode properties of a physical systems, and the extremal eigenvectors of such operators capture important informations. For non-trivial systems, the number of dimensions of the state space quickly becomes daunting on conventional computers. ↳ 1.9.5

6.2.2 Optimization problems can be reduced to the quest for an extremal eigenvector in the following way. An optimization problem is typically under the form

$$\text{Maximize / minimize } C(x) \text{ when } x \text{ belongs to some set } S$$

with C a *cost function* outputting real values.

In the discrete, finite case, one can always pick $S = \{0..2^n - 1\}$ (with potentially a dummy padding to match a power of two in size). The function C then simply inputs bitstrings of size n, using for instance big-endian notation (see 4.3.2).

If one builds a diagonal hermitian matrix as ↳ 1.9.4

$$H_C = \sum_x C(x) \left|x\right\rangle \left\langle x\right|,$$

we have $H_C \left|x\right\rangle = C(x) \left|x\right\rangle$. So

- Minimizing $C(x)$ consists in finding the minimal eigenvalue of H_C.
- Maximizing $C(x)$ consists in finding the maximal eigenvalue of H_C, therefore the minimal eigenvalue of $-H_C$: this is the same problem (up to a change in sign).

6.2.3 The kind of problem VQE can solve could be named QuantumMinEigen:

QuantumMinEigen	
Input	an hermitian matrix H
Output	an eigenvector $\|\psi\rangle$ with minimal eigenvalue

The algorithm is very simple: it is about solving an optimization problem in the state space. We aim at minimizing the function

$$F : \begin{cases} \mathscr{H}^{\otimes n} & \longrightarrow & \mathbb{R} \\ |\psi\rangle & \longmapsto & \langle\psi| H |\psi\rangle \end{cases} \tag{57}$$

In theory this is a regular function: one can use any procedure such as gradient descent to solve it. In practice, it is not clear how to do this efficiently.

6.2.4 Claim. Assuming that $\lambda_{\min}$ is the minimal eigenvalue of H, for all ket-vector $|\psi\rangle$ we have

$$\langle\psi| H |\psi\rangle \geq \lambda_{\min}.$$

The equality is attained when $|\psi\rangle = |\psi_{\min}\rangle$.

6.2.5 Proof of 6.2.4. According to 1.9.4, one can rewrite H as

$$H = \sum_j \lambda_j \cdot |\psi_j\rangle \langle\psi_j| ,$$

where the $|\psi_j\rangle$s form an orthonormal basis of eigenvectors. Among them, $|\psi_{\min}\rangle$ is a minimal one. For the sake of the discussion, assume that it is the only one. We then have

$$H = \lambda_{\min} \cdot |\psi_{\min}\rangle \langle\psi_{\min}| + \sum_{j \neq \min} \lambda_j \cdot |\psi_j\rangle \langle\psi_j| .$$

We can also decompose $|\psi\rangle$ as

$$|\psi\rangle = \alpha |\psi_{\min}\rangle + \beta |\psi_{\min}^{\perp}\rangle$$

where $|\psi_{\min}^{\perp}\rangle$ is of norm 1 and orthogonal to $|\psi_{\min}\rangle$ and $|\alpha|^2 + |\beta|^2 = 1$. The ket-vector $|\psi_{\min}^{\perp}\rangle$ can be written as

$$|\psi_{\min}^{\perp}\rangle = \sum_{j \neq \min} \gamma_j |\psi_j\rangle$$

with

$$\sum_{j \neq \min} |\gamma_j|^2 = 1. \tag{58}$$

Note how $H\left|\psi_{\text{min}}^{\perp}\right\rangle$ is orthogonal to $\left|\psi_{\text{min}}\right\rangle$. Let us compute:

$$\begin{aligned}\left\langle\psi\right| H \left|\psi\right\rangle &= \left(\overline{\alpha}\left\langle\psi_{\text{min}}\right| + \overline{\beta}\left\langle\psi_{\text{min}}^{\perp}\right|\right) H \left(\alpha\left|\psi_{\text{min}}\right\rangle + \beta\left|\psi_{\text{min}}^{\perp}\right\rangle\right)\\ &= |\alpha|^2 \left\langle\psi_{\text{min}}\right| H \left|\psi_{\text{min}}\right\rangle + \alpha\overline{\beta}\left\langle\psi_{\text{min}}^{\perp}\right| H \left|\psi_{\text{min}}\right\rangle + \overline{\alpha}\beta\left\langle\psi_{\text{min}}\right| H \left|\psi_{\text{min}}^{\perp}\right\rangle + |\beta|^2 \left\langle\psi_{\text{min}}^{\perp}\right| H \left|\psi_{\text{min}}^{\perp}\right\rangle\\ &= |\alpha|^2\lambda_{\text{min}} \left\langle\psi_{\text{min}}\right|\left|\psi_{\text{min}}\right\rangle + \alpha\overline{\beta}\lambda_{\text{min}}\left\langle\psi_{\text{min}}^{\perp}\right|\left|\psi_{\text{min}}\right\rangle + 0 + |\beta|^2 \left\langle\psi_{\text{min}}^{\perp}\right| H \left|\psi_{\text{min}}^{\perp}\right\rangle\\ &= |\alpha|^2\lambda_{\text{min}} + |\beta|^2 \left\langle\psi_{\text{min}}^{\perp}\right| H \left|\psi_{\text{min}}^{\perp}\right\rangle.\end{aligned}$$

Because of the minimality assumption, we have $\lambda_{\text{min}} \leq \lambda_j$ for all j. We also have that

$$\left\langle\psi_{\text{min}}^{\perp}\right| H \left|\psi_{\text{min}}^{\perp}\right\rangle = \sum\nolimits_j |\gamma_j|^2 \left\langle\psi_j\right| H \left|\psi_j\right\rangle = \sum\nolimits_j |\gamma_j|^2\lambda_j.$$

Since $\lambda_{\text{min}} \leq \lambda_j$, we derive that

$$\lambda_{\text{min}} = 1 * \lambda_{\text{min}} = \left(\sum\nolimits_j |\gamma_j|^2\right)\lambda_{\text{min}} = \sum\nolimits_j |\gamma_j|^2\lambda_{\text{min}} \leqslant \sum\nolimits_j |\gamma_j|^2\lambda_j = \left\langle\psi_{\text{min}}^{\perp}\right| H \left|\psi_{\text{min}}^{\perp}\right\rangle.$$

and thus

$$\left\langle\psi_{\text{min}}^{\perp}\right| H \left|\psi_{\text{min}}^{\perp}\right\rangle \geqslant \lambda_{\text{min}}.$$

Summarizing,

$$\left\langle\psi\right| H \left|\psi\right\rangle = |\alpha|^2\lambda_{\text{min}} + |\beta|^2 \left\langle\psi_{\text{min}}^{\perp}\right| H \left|\psi_{\text{min}}^{\perp}\right\rangle \geqslant |\alpha|^2\lambda_{\text{min}} + |\beta|^2\lambda_{\text{min}} = \left(|\alpha|^2 + |\beta|^2\right)\lambda_{\text{min}} = \lambda_{\text{min}}$$

which shows that $\left\langle\psi\right| H \left|\psi\right\rangle$ is always larger than λ_{min}. This inequality becomes an equality when $\left|\psi\right\rangle$ is the eigenvector $\left|\psi_{\text{min}}\right\rangle$ since

$$\left\langle\psi_{\text{min}}\right| H \left|\psi_{\text{min}}\right\rangle = \lambda_{\text{min}} \left\langle\psi_{\text{min}}\right|\left|\psi_{\text{min}}\right\rangle = \lambda_{\text{min}}.$$

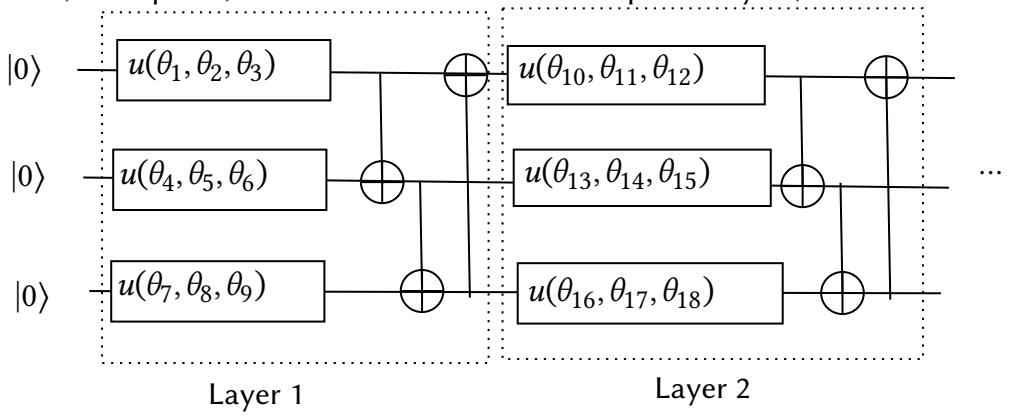
6.2.6 Sketch of the VQE algorithm. This is where we will use the quantum co-processor. Instead of directly manipulating a ket-state $\left|\psi\right\rangle$ to move it towards $\left|\psi_{\text{min}}\right\rangle$, the idea consists in manipulating a set of (real) parameters instead, used to specify a circuit. This circuit is then used to compute a candidate $\left|\psi\right\rangle$. The procedure is as follows:

$$\vec{\theta} \in \mathbb{R}^p \xmapsto{\text{build}} \text{some circuit} \xmapsto{\text{evaluate}} \left|\psi\right\rangle \xmapsto{\text{estimate}} \left\langle\psi\right| H \left|\psi\right\rangle$$

The first part generates a circuit out of p real parameters, the second part consists in evaluating the circuit to get the candidate ket-vector $\left|\psi\right\rangle$, and the third part estimates the desired scalar product. The goal of VQE is to make use of the quantum co-processor for the two last operations.

6.2.7 Structure of the circuit. For VQE, the circuit is built from an *ansatz*: a circuit typically consisting of rotations and 2-qubit gates, where the rotation are parameterized by the array of real numbers $\vec{\theta}$. There is no choice *a priori* on the structure of the circuit. We only need a circuit shape that can approximate well-enough the desired extremal eigenvector. In the context of quantum chemistry, the ansatz can for instance capitalize on the symmetry of the problem.

6.2.8 Without any knowledge on the structure of the Hermitian, we are bound tochoose a generic ansatz, entangling enough to reach enough of the state space. On 1 qubit: we only need 3 angles (see 2.5.9). On more qubits, we need at least one entangling gate, typically the CNOT gate, in order to reach states that are not necessarily separable. For instance, on 3 qubits, one can consider a circuit shaped in layers, as follows. ↳ 2.3.6



(with u a generic unitary parameterized by 3 angles.) As we increase the number of layers, the ansatz becomes more and more expressive, but also more and more expensive to compute and manipulate. Indeed, optimization techniques tend to perform poorly with a large number of parameters: this problem is known as the *Barren plateau*.

In any case, the circuit computes a candidate state parameterized by an array of θ's. We then have a purely classical set $\mathbb{R}^p$ for some p, representing angles, from which we build a circuit that, when run on the quantum co-processor, generates the $|\psi\rangle$ we care about.

6.2.9 Estimating the scalar product Running the circuit on a quantum co-processor indeed produces a ket-vector $|\psi\rangle$, but it is hidden inside the quantum memory. Estimating $\langle\psi| H |\psi\rangle$ can be done using an estimation of the amplitudes of the various basis vectors in $|\psi\rangle$.

In this course, we focus on the case of diagonal hermitian matrices. If $|\psi\rangle$ is

$$|\psi\rangle = \sum_j \alpha_j |j\rangle$$

then

$$\langle\psi| H |\psi\rangle = \sum_j |\alpha_j|^2 C(j)$$

and this can be computed offline, on the classical device, using an estimate of the probability distribution coming from the measure of $|\psi\rangle$.

6.2.10 In details, running the VQE algorithm consists in choosing an ansatz and building a function $f : \vec{\theta} \mapsto$[some real] as follows:

- Using the ansatz and the parameters $\vec{\theta}$, generate a circuit.
- Run many times (say N times) the circuit on $|00\ldots0\rangle$, followed by a measure.

- This gives an estimation of the probability distribution that associate to each $x_1, \dots, x_n$ the corresponding probability $p_{\vec{x}} \in [0,1]$.
- The output of f is built as $\sum_{\vec{x}} p_{\vec{x}} C(\vec{x})$.

The objective is to minimize the function f: it is a purely classical function that can be coded in any language (such as Python) and this can be done using any suitable optimization library.

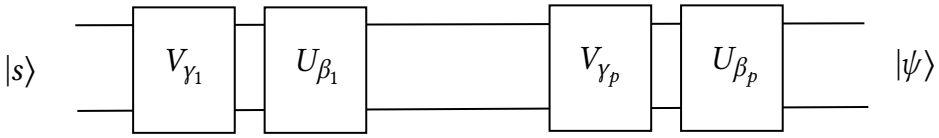
6.3 QAOA

6.3.1 Let us take a bit of a high-level view from VQE: we start from a Hermitian, but the ansatz is completely blind to the structure of this Hermitian. The algorithm only uses it at the very end to adjust the parameters. One can think that maybe making a circuit shape specific to the problem could somehow speed up the algorithm: this is the proposal of Farhi and his co-authors with *QAOA* [FGG14].

6.3.2 QAOA stands for *Quantum Approximate Optimization Algorithm*, and can be regarded as a specialized VQE for optimization problems. It essentially implements the simulation of an adiabatic evolution.

6.3.3 How it works. We make use of 2 hermitians, both acting on our n qubits. The first one if $B = \sum_{i=1}^{n} \sigma_X^i$, where σ_X^i is the action of X on qubit i, and the Hamiltonian encoding the cost function: $H_C = \sum_x C(x) |x\rangle\langle x|$. The shape of the algorithm is very close to VQE:

- Start from a state $|s\rangle = \frac{1}{\sqrt{2}^n} \sum_{x_1 \dots x_n} |x_1 \dots x_n\rangle$
  - this is just a tower of Hadamard acting on $|000\dots00\rangle$
- Generate $|\psi\rangle$ from a circuit parameterized by two arrays $\vec{\beta}$ et $\vec{\gamma}$, each of p angles
  - $|\psi\rangle = U_{\beta_p} V_{\gamma_p} U_{\beta_{p-1}} V_{\gamma_{p-1}} \dots U_{\beta_2} V_{\gamma_2} U_{\beta_1} V_{\gamma_1} |s\rangle$
  - with
    - $U_\lambda = e^{-i\lambda B}$
    - $V_\lambda = e^{-i\lambda H_C}$



- Compute $\langle\psi| H_C |\psi\rangle$ as for VQE.
- This series of operations can be seen as a function that inputs two arrays $(\vec{\beta}, \vec{\gamma})$ and that outputs a real numbers. As for VQE, we can optimize this function (here, we will want to MAXIMIZE it).

Seen like that, QAOA is just an instance of VQE, with a circuit that is a bit more funky than the one we used above. The question is: why do we have a good (better) chance to get to the right vector if we perform a maximization?

6.3.4 This raises two questions:

- Is there any **proof** for any speedup compared to classical methods?
  - No
  - Some theoretical results for p=1 et p=2, with error bounds
  - No speedup shown up to date
  - But not proof that there are none...
- Do we at least have a guarantee on the convergence towards a solution?
  - Yes, for p "large enough"
  - To understand how it works, we need a small sidestep.

6.3.5 Sidestep: adiabatic evolution. (*Disclaimer:* I am not physicists. There are a lot of caveats in what I will say, but the intuition still stands.) A physical system is subject to an Hamiltonian H which in our case is nothing more than a hermitian matrix. This Hamiltonian might evolve along time: $H(t)$. The evolution of a system $|\psi_t\rangle$ is described by the Shrödinger equation

$$\frac{d\,|\psi_t\rangle}{dt} = -i \cdot H(t)\,|\psi_t\rangle$$

Note: if $H(t)$ is constant with value H, the solution with initial condition $|\psi_0\rangle$ is...

$$|\psi_t\rangle = e^{-itH}\,|\psi_0\rangle$$

(and we find back the relationship between hermitian and unitary). In any case, the *adiabatic theorem* states that

> Adiabatic Theorem
>
> If $H(t)$ varies slow enough, and if the system is at t=0 at minimal energy level (i.e. its state is a minimal eigenvector of $H(0)$), then at each t its energy level is minimal (i.e. its state is a minimal eigenvector of $H(t)$.

The "slow enough" has to do with the *spectral gap*: the distance between the two lowest eigenvalues. Note that one can play the same game with the maximal eigenvalue by instead considering $-H(t)$ (which is also Hermitian!).

6.3.6 Link with QAOA. We start with

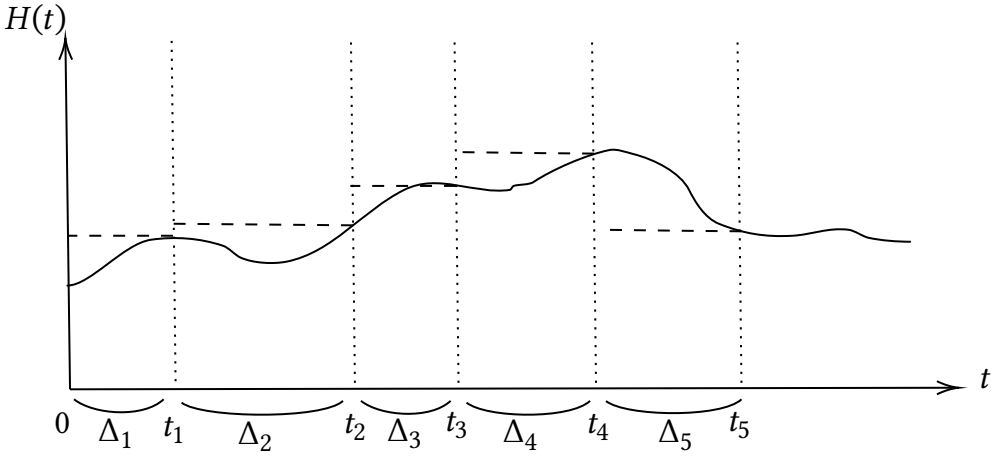
$$|\psi_0\rangle = |s\rangle = \frac{1}{\sqrt{2}^n} \sum_{x_1 \ldots x_n} |x_1 \ldots x_n\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right) \otimes \ldots \otimes \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right)$$

We want to get to the eigenvector of maximal eigenvalue for H_C.

6.3.7 Question: is $|s\rangle$ the maximal eigenvalue for somebody?
$\rightarrow B = \sum_{i=1}^{n} \sigma_X^i$ (σ_X^i is the action of X on qubit i)
And it has all of the required properties for the adiabatic theorem to hold.

6.3.8 From the adiabatic theorem we can deduce that the interpolation:
$H(t) = \frac{t}{T}H_C + \left(1 - \frac{t}{T}\right)B$
with a large enough T will slowly send $|s\rangle$ to the desired eigenvector
$\longrightarrow H(0) = B$ et $H(T) = H_C$
Remains to understand how to do this.

6.3.9 Instead of considering $H(t)$, we shall consider a piecewise constant approximation. To not move apart from the "correct" function, we need small eough pieces.



On each timeslot Δ_i, the Hamiltonian is approximated as constant. One can then say that (according to the Shrödinger equation)

$$|\psi_{t_{i+1}}\rangle \simeq e^{-i\Delta_i H(t_i)}|\psi_{t_i}\rangle$$

and then that if we consider the slicing of $[0,T]$ into p slides,

$$|\psi_T\rangle = e^{-i\Delta_p H(t_p)}...e^{-i\Delta_2 H(t_2)}e^{-i\Delta_1 H(t_1)}|s\rangle$$

is an approximation of the maximal eigenvector of H_C. We almost have the form of the QAOA circuit

6.3.10 To conclude, we need the *Trotter-Suzuki* formula of 4.7 stating that

$$e^{\delta(A+B)} = e^{\delta A}e^{\delta B} + O\left(\delta^2\right)$$

whenever δ is small. So $e^{-i\Delta_p H(t_p)} = e^{-i\Delta_p(\alpha B + \beta H_C)} \simeq e^{-i\Delta_p \alpha B}e^{-i\Delta_p \beta H_C}$ since Δ_p is supposed to be small.

6.3.11 Let us summarize: for QAOA, we build

- $|\psi\rangle = U_{\beta_p}V_{\gamma_p}U_{\beta_{p-1}}V_{\gamma_{p-1}}...U_{\beta_2}V_{\gamma_2}U_{\beta_1}V_{\gamma_1}|s\rangle$

with

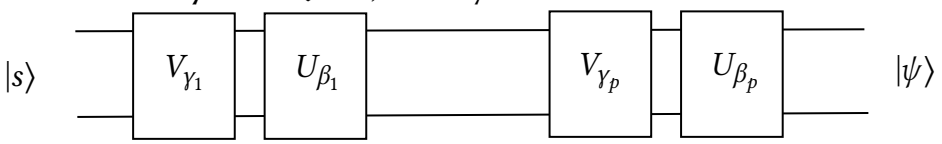
- $U_\lambda = e^{-i\lambda B}$
- $V_\lambda = e^{-i\lambda H_C}$

From the adiabatic theorem, we have

$$|\psi_T\rangle = e^{-i\Delta_p H(t_p)} \ldots e^{-i\Delta_2 H(t_2)} e^{-i\Delta_1 H(t_1)} |s\rangle$$

We can then replace each $e^{-i\Delta_p H(t_p)}$ with a product$U_\alpha V_\beta$. How to choose each of these α, β, since we do not know which Δ_i and t_i are optimal? This is why we consider this as an optimization problem: a gradient descent (for instance) will figure it out for us.

6.3.12 Summary. For QAOA, we only need to build a circuit



with

- $U_\lambda = e^{-i\lambda B} \longrightarrow$ this is just a tower of X-rotations (R_x-gates) with angles λ
- $V_\lambda = e^{-i\lambda H_C} \longrightarrow$ well, this depends on $C(x)$!

and then we minimize a function that

- Inputs $\vec{\beta}, \vec{\gamma}$
- Realize and run the corresponding circuit many times
- Get a estimation of the probability distribution
- Compute and return (an estimate of) $\langle\psi| H_C |\psi\rangle$

Remains to know how to build $e^{-i\lambda H_C}$.

6.3.13 To build $e^{-i\lambda H_C}$, one can capitalize on the decomposition presented in 1.10.10: H_C is written as a sum of (tensors of) Pauli matrices:

$$H_C \quad = \sum_{G_1,\ldots G_n \in \{X,Y,Z,I\}} h_{G_1,\ldots,G_n} \cdot G_1 \otimes \cdots \otimes G_n$$

with $h_{G_1,\ldots,G_n} \in \mathbb{R}$. Note that in the very specific case of QAOA, because H_C is a diagonal matrix, so the matrices G_i's are only among I and Z. In any case, using the Trotter-Suzuki decomposition of 4.7, we deduce that

$$e^{-i\lambda H_C} \sim \prod_{G_1,\ldots G_n \in \{X,Y,Z,I\}} e^{-i\lambda h_{G_1,\ldots,G_n} \cdot G_1 \otimes \cdots \otimes G_n}.$$

Realizing $e^{-i\lambda H_C}$ is then reduced to the implementation of an exponential of (tensors of) Pauli matrices (see for instance 1.10.13 and 2.11.3).

6.3.14 If H_C can necessary be written as a linear decomposition of (tensors of) Pauli matrices, in the context of QAOA the function C is in general given in a form that makes it easy to generate the decomposition: a (real) polynomials of the x_i's, seen as the values 0 and 1, as follows.

$$C(x_1, \dots, x_n) = \sum_k h_k \cdot x_1^{d_{1,k}} \cdots x_n^{d_{n,k}}.$$

The trick to go from C to H_C is to change the x_j's with linear functions

$$X_j : \begin{array}{ccc} \mathscr{H}^{\otimes n} & \to & \mathscr{H}^{\otimes n} \\ |x_1 \dots x_n\rangle & \to & \begin{cases} |x_1 \dots x_n\rangle & \text{if } x_j = 1 \\ 0 & \text{else} \end{cases} \end{array}$$

and the multiplications with function composition. We then have:

$$H_C = \sum_k h_k \cdot X_1^{d_{1,k}} \cdots X_n^{d_{n,k}}$$

(where X_j^0 stands for the identity function). Now, note how one can encode each X_j as

$$X_j = (I - Z_j)/2$$

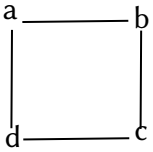
with $Z_j = I \otimes \cdots \otimes I \otimes Z \otimes I \otimes \cdots \otimes I$, the Z matrix acting on the j-th qubit. We have our decomposition:

$$H_C = \sum_k h_k \cdot ((I - Z_1)/2)^{d_{1,k}} \cdots ((I - Z_n)/2)^{d_{n,k}}$$

6.3.15 QAOA pour MAXCUT. The problem we want to solve is an *optimization problem*, defined as follows.

MAXCUT	
Input	a non-oriented graph $G = (V, E)$.
Output	A CUT: a partition of V into $V_0 \cup V_1$ (and $V_0 \cap V_1 = \emptyset$).
Constraint	Minimize the number of edges going from V_0 to V_1.

6.3.16 Consider for instance the following graph.



- The cost of $V_0 = \{a, b\}, V_1 = \{c, d\}$ is 2
- The cost of $V_0 = \{a, c\}, V_1 = \{b, d\}$ is 4
- The cost of $V_0 = \{c\}, V_1 = \{a, b, d\}$ is 2

The maximal cost one can get is 4, and one of the maxcut is $V_0 = \{a, c\}, V_1 = \{b, d\}$

6.3.17 Cuts as bitstrings In the case of MAXCUT, the cost function works over the set of all possible cuts. A cut can be coded in a bitstring as follow. Assume $V = \{0...n-1\}$: One can store in a boolean value x_i the position of the node $i \in V$:

$$i \in V_{x_i}.$$

Now, given a vector $x_0...x_{n-1}$, this vector stores where each node belongs to. For instance, the max cut of 6.3.16 can be encoded as 0011 or 1100, assuming that $a = 0$, $b = 1$, $c = 2$ and $d = 3$.

6.3.18 The Hermitian of the cost function. In general, one can write a cost function

$$C(x) = \sum_{(i,j)\in E} x_i(1-x_j) + x_j(1-x_i).$$

This function "counts" how many edges connect V_0 and V_1. The maximum cut is obtained with the maximization of this function. According to 6.3.14, the corresponding Hermitian is then

$$\begin{aligned} H_C &= \sum_{(i,j)\in E} \left(\frac{1}{4}(1-Z_i)(1+Z_j) + \frac{1}{4}(1-Z_j)(1+Z_i)\right) \\ &= \frac{1}{4}\sum_{(i,j)\in E} (1 - Z_i + Z_j - Z_iZ_j + 1 - Z_j + Z_i - Z_jZ_i) \\ &= \frac{1}{4}\sum_{(i,j)\in E} (2 - 2Z_iZ_j) \\ &= \frac{1}{2}\sum_{(i,j)\in E} (1 - Z_iZ_j) \end{aligned}$$

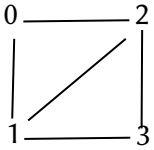
All the terms in the sum commute: the unitary $e^{i\lambda H_C}$ is then exactly

$$e^{i\lambda H_C} = \prod e^{\lambda i(1-Z_iZ_j)} = (\text{some phase}) \cdot \prod e^{-\lambda i Z_iZ_j}.$$

For the corresponding circuit, see 1.10.13 and 2.11.3...

6.4 Exercises

6.4.1 MAXCUT Consider the following graph.



What are (or "is") the maximum cuts here? What bitstrings do they correspond to? What is the cost function, and the corresponding Hermitian H_C?

A Geometric series

Assume $a \neq 1$. Then we have

$$\sum_{n=0}^{N} a^n = \frac{1-a^{N+1}}{1-a}.$$

Indeed,

$$\begin{aligned}
(1-a)\cdot\sum_{n=0}^{N} a^n &= \left(\sum_{n=0}^{N} a^n\right) - a\cdot\left(\sum_{n=0}^{N} a^n\right) \\
&= \left(\sum_{n=0}^{N} a^n\right) - \left(\sum_{n=0}^{N} a\cdot a^n\right) \\
&= \left(\sum_{n=0}^{N} a^n\right) - \left(\sum_{n=0}^{N} a^{n+1}\right) \\
&= \left(a^0 + a^1 + a^2 + \cdots + a^N\right) - \left(a^1 + a^2 + \cdots + a^N + a^{N+1}\right) \\
&= a^0 - a^{N+1} \\
&= 1 - a^{N+1}.
\end{aligned}$$

Provided that $0 \leq a < 1$, this sum converges as N tends to infinity. The corresponding *geometric series* then admits a closed form:

$$\sum_{n=0}^{\infty} a^n \quad = \quad \frac{1}{1-a}.$$

B Exponential and Trigonometric Functions

The exponential function $x \mapsto e^x$ is defined as

$$e^x \quad \triangleq \quad \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

For all real number x, the sum e^x is absolutely converging. A formal argument is as follows. For all x, there exists a natural number N such that $x < N$. When $n > N+1$, write $n' = n - N - 1$ and

$$\frac{x^n}{n!} \leq \frac{N^n}{n!}$$

$$\frac{N}{1}\cdot\ldots\cdot\frac{N}{n-1}\cdot\frac{N}{n}$$

$$cst\cdot\frac{N}{N}\cdot\frac{N}{N+1}\cdot\frac{N}{N+2}\ldots\cdot\frac{N}{N+n'+1}$$

$$\leq cst \cdot \left(\frac{N}{N+1}\right)^{n'}$$

So e^x can be bounded by

$$something + cst \cdot \sum_{n'=0}^{\infty} \left(\frac{N}{N+1}\right)^{n'}$$

Since $0 \leq N/(N+1) < 1$, this is converging.

C Cosine-Sine Decomposition

The statement of Theorem 3.3.13 is as follows:

C.1 Statement

Let U by any $n+1$-qubit unitary. One can decompose U as

$$\begin{pmatrix} B_1 & 0 \\ 0 & B_2 \end{pmatrix}\begin{pmatrix} C & -S \\ S & C \end{pmatrix}\begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$$

with A, B, C and S n-qubit unitaries, with C and S n-qubit diagonals such that $S^2 + C^2 = Id$.

C.2 Proof

C.2.1 Write U blockwise with n-qubit-sized blocks:

$$U = \begin{pmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{pmatrix}$$

Let $B_1 C A_1 = U_{11}$ be a diagonalization of U_{11}: the matrices A_1 and B_1 are unitary, and D is diagonal with real entries. Without loss of generality, we can assume the entries are non-negative and in decreasing order (with the largest on the left). Because the columns of the matrix U_{11} are pieces of columns of the unitary U, their norms are less than 1: so are the eigenvalues. The matrix C can then be represented blockwise as

$$C = \begin{pmatrix} Id & 0 & 0 \\ 0 & D & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

with D diagonal with positive entries that are neither 0 nor 1.

C.2.2 Now, using QL and QR decompositions, choose B_2' and A_2' so to make $B_2' U_{21} A_1'$ lower and $B_1' U_{12} B_2'$ upper triangular. Without loss of generality, we can assume that the former has non-negative real numbers on the diagonal, while the later has non-positive ones. Define the matrix E as

$$E = \begin{pmatrix} B_1' & 0 \\ 0 & B_2' \end{pmatrix}\begin{pmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{pmatrix}\begin{pmatrix} A_1' & 0 \\ 0 & A_2' \end{pmatrix} = \begin{pmatrix} C & B_1' U_{12} A_2' \\ B_2' U_{21} A_1' & B_2' U_{22} A_2' \end{pmatrix}. \quad (59)$$

It is unitary and can then be written as blocks as follows:

$$E = \left(\begin{array}{ccc|ccc} Id & 0 & 0 & R_1 & Q_1 & Q_2 \\ 0 & D & 0 & 0 & R_2 & Q_3 \\ 0 & 0 & 0 & 0 & 0 & R_3 \\ \hline L_1 & 0 & 0 & N_1 & N_2 & N_3 \\ P_1 & L_2 & 0 & N_4 & N_5 & N_6 \\ P_3 & P_4 & L_3 & N_7 & N_8 & N_9 \end{array}\right).$$

Unitarity imposes constraints from which we can explicit each block

C.2.3 Collapse of 1st block-row and 1st block-column . Each row in

$$\begin{array}{ccc|ccc} Id & 0 & 0 & R_1 & Q_1 & Q_2 \end{array}$$

and each column in

$$\begin{array}{c} Id \\ 0 \\ 0 \\ \hline L_1 \\ P_1 \\ P_3 \end{array}$$

is of norm 1: this forces $R_1, Q_1, Q_2, L_1, P_1, P_3$ to be zero-matrices since one of the entries is necessarily 1 (from the *Id* block).

C.2.4 Collapse of the last block-row. Consider the 3rd block of columns and the last block of rows:

$$\begin{array}{ccc|ccc} & & 0 & & & \\ & & 0 & & & \\ & & 0 & & & \\ \hline & & 0 & & & \\ & & 0 & & & \\ P_3 & P_4 & L_3 & N_7 & N_8 & N_9 \end{array}$$

The matrix L_3 is lower-triangular as follows:

$$L_3 = \left(\begin{array}{cccc} \ddots & \vdots & \vdots & \vdots \\ \cdots & l_6 & 0 & 0 \\ \cdots & l_5 & l_4 & 0 \\ \cdots & l_3 & l_2 & l_1 \end{array}\right).$$

Consider the column of E ending with l_1: all of its other entries are 0s. Since it is of norm 1, l_1 is 1. The last row of E is also of norm 1: because one of its entry is 1, all other entries are 0: the matrix L_3 is

$$L_3 = \left(\begin{array}{cccc} \ddots & \vdots & \vdots & \vdots \\ \cdots & l_6 & 0 & 0 \\ \cdots & l_5 & l_4 & 0 \\ \cdots & 0 & 0 & 1 \end{array}\right),$$

and the last lines of P_3, P_4, N_7, N_8 and N_9 are all 0s.

A similar argument can now be used on the column of E containing l_4: we derive that l_4 is 1, and that the second-to-last line of E contains zero everywhere except at the position l_4:

$$L_3 = \begin{pmatrix} \ddots & \vdots & \vdots & \vdots \\ \cdots & l_6 & 0 & 0 \\ \cdots & 0 & 1 & 0 \\ \cdots & 0 & 0 & 1 \end{pmatrix},$$

and the second-to-last lines of P_3, P_4, N_7, N_8 and N_9 are all 0s. Working through all of the rows of L_3, we end up with

$$L_3 = Id, P_3 = 0, P_4 = 0, N_7 = 0, N_8 = 0, N_9 = 0.$$

Using a symmetric argument, we can derive that

$$R_3 = -Id, Q_3 = 0, Q_2 = 0, N_3 = 0, N_6 = 0, N_9 = 0$$

(For the $-Id$, remember that the matrix has non-positive coefficients on the diagonal).

C.2.5 So far, we simplified E into

$$E = \left(\begin{array}{ccc|ccc} Id & 0 & 0 & 0 & 0 & 0 \\ 0 & D & 0 & 0 & R_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & -Id \\ \hline 0 & 0 & 0 & N_1 & N_2 & 0 \\ 0 & L_2 & 0 & N_4 & N_5 & 0 \\ 0 & 0 & Id & 0 & 0 & 0 \end{array}\right).$$

C.2.6 Instantiating R_2. The next step is to instantiate R_2 and L_2. For this, consider the second block of rows:

$$0 \quad D \quad 0 \,\big|\, 0 \quad R_2 \quad 0.$$

It is of the form

$$\left(\begin{array}{ccccccc|ccccccc} 0 & \cdots & 0 & \ddots & \vdots & \vdots & \vdots & 0 & \cdots & 0 & 0 & \cdots & 0 & \ddots & \vdots & \vdots & \vdots & 0 & \cdots & 0 \\ \vdots & & \vdots & \cdots & c_3 & 0 & 0 & \vdots & & \vdots & \vdots & & \vdots & \cdots & r_3 & z_3 & z_2 & \vdots & & \vdots \\ \vdots & & \vdots & \cdots & 0 & c_2 & 0 & \vdots & & \vdots & \vdots & & \vdots & \cdots & 0 & r_2 & z_1 & \vdots & & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & 0 & c_1 & 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & r_1 & 0 & \cdots & 0 \end{array}\right) \tag{60}$$

Because E is unitary, the last row in Eq. (60) is of norm 1:

$$c_1^2 + r_1^2 = 1.$$

Since $c_1 \neq 1$, we deduce $r_1 \neq 0$. Since the last and second-to-last are orthogonal, we have $r_1 z_1 = 0$, so $z_1 = 0$. The same argument on the last and third-to-last gives $z_2 = 0$. Proceeding similarly for all the remaining rows, we conclude that the last column of R_2 is zero, except for its last element r_1. Eq. (60) can then be rewritten

$$\left(\begin{array}{ccccccc|ccccccc} 0 & \cdots & 0 & \ddots & \vdots & \vdots & \vdots & 0 & \cdots & 0 & 0 & \cdots & 0 & \ddots & \vdots & \vdots & \vdots & 0 & \cdots & 0 \\ \vdots & & \vdots & \cdots & c_3 & 0 & 0 & \vdots & & \vdots & \vdots & & \vdots & \cdots & r_3 & z_3 & 0 & \vdots & & \vdots \\ \vdots & & \vdots & \cdots & 0 & c_2 & 0 & \vdots & & \vdots & \vdots & & \vdots & \cdots & 0 & r_2 & 0 & \vdots & & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & 0 & c_1 & 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & r_1 & 0 & \cdots & 0 \end{array}\right) \tag{61}$$

Using the same argument, focusing on the norm of the second-to-last row and the fact that $c_2 \neq 1$, we deduce that $r_2 \neq 0$, and then that all elements in R_2 above r_2 are null. We can proceed upwards, clearing all elements off diagonal: we deduce that R_2 is diagonal with non-zero entries, and that

$$D^2 + R_2^2 = Id. \tag{62}$$

C.2.7 Instantiating L_2. Using a similar reasonning but with column in place of rows, we can deduce that L_2 is diagonal with non-zero entries, and that

$$D^2 + L_2^2 = Id. \tag{63}$$

Since all of the diagonal elementsof R_2 are non-positive and ones of L_2 are non-negative, from Eq. (62) and (63) we derive that

$$R_2 = -L_2.$$

C.2.8 Let us define the diagonal matrix L_2 with T: we simplified E into

$$E = \left(\begin{array}{ccc|ccc} Id & 0 & 0 & 0 & 0 & 0 \\ 0 & D & 0 & 0 & -T & 0 \\ 0 & 0 & 0 & 0 & 0 & -Id \\ \hline 0 & 0 & 0 & N_1 & N_2 & 0 \\ 0 & T & 0 & N_4 & N_5 & 0 \\ 0 & 0 & Id & 0 & 0 & 0 \end{array}\right).$$

C.2.9 Collapsing N_2 and N_4. The next step is to show how N_2 and N_4 are zero-matrices. To show $N_2 = 0$, we consider the orthogonality of rows in

$$\left(\begin{array}{ccc|ccc} 0 & D & 0 & 0 & -T & 0 \end{array}\right)$$

with rows in

$$\left(\begin{array}{ccc|ccc} 0 & 0 & 0 & N_1 & N_2 & 0 \end{array}\right)$$

Computing the scalar product of row number i in the latter with row number j in the former yields

$$t_i n_{j,i} = 0,$$

where $n_{j,i}$ is the j,i-th element in N_2 and t_i the ith element on the diagonal of T. Since none of these diagonal elements are zero, $t_i \neq 0$, so $n_{j,i} = 0$. And this is true for all i, j. So $N_2 = 0$.

We can follow the same argument with columns instead and derive that $N_4 = 0$.

C.2.10 We then simplified E into

$$E = \left(\begin{array}{ccc|ccc} Id & 0 & 0 & 0 & 0 & 0 \\ 0 & D & 0 & 0 & -T & 0 \\ 0 & 0 & 0 & 0 & 0 & -Id \\ \hline 0 & 0 & 0 & N_1 & 0 & 0 \\ 0 & T & 0 & 0 & N_5 & 0 \\ 0 & 0 & Id & 0 & 0 & 0 \end{array}\right).$$

C.2.11 Instantiating N_5. We can now whow how N_5 is diagonal. First, consider the scalar product of one the ith row in the block of rows

$$\begin{array}{ccc|ccc} 0 & D & 0 & 0 & -T & 0 \end{array}$$

and the ith row in the block of rows

$$\begin{array}{ccc|ccc} 0 & T & 0 & 0 & N_5 & 0. \end{array} \tag{64}$$

We get $c_i t_i - t_i n_{i,i} = 0$: we deduce that $n_{i,i} = c_i$. The ith row in the block of rows of Eq. 64 has a norm of the form

$$t_i^2 + c_i^2 + \sum_{j \neq i} |n_{i,j}|^2,$$

supposed to be equal to 1. But since $t_i^2 + c_i^2 = 1$, we deduce that all of the terms $n_{i,j}$ with $i \neq j$ are zero: The matrix N_5 is diagonal, it is in fact equal to D.

C.2.12 We then simplified E into

$$E = \left(\begin{array}{ccc|ccc} Id & 0 & 0 & 0 & 0 & 0 \\ 0 & D & 0 & 0 & -T & 0 \\ 0 & 0 & 0 & 0 & 0 & -Id \\ \hline 0 & 0 & 0 & N_1 & 0 & 0 \\ 0 & T & 0 & 0 & D & 0 \\ 0 & 0 & Id & 0 & 0 & 0 \end{array}\right).$$

C.2.13 Instantiating N_1. Remains to study N_1. For this, consider the fact that $E^*E = EE^* = Id$. Blockwise, this means $N_1 N_1^* = N_1^* N_1 = Id$.

So if instead of A_2' we had chosen

$$A_2'' = A_2' \begin{pmatrix} -N_1^* & 0 & 0 \\ 0 & Id & 0 \\ 0 & 0 & Id \end{pmatrix}$$

Then we would have gotten instead some other E'

$$E' = \left(\begin{array}{ccc|ccc} Id & 0 & 0 & 0 & 0 & 0 \\ 0 & D & 0 & 0 & -T & 0 \\ 0 & 0 & 0 & 0 & 0 & -Id \\ \hline 0 & 0 & 0 & I & 0 & 0 \\ 0 & T & 0 & 0 & C & 0 \\ 0 & 0 & Id & 0 & 0 & 0 \end{array}\right),$$

of the required form.

References

[BK05] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal clifford gates and noisy ancillas. *Physical Review A*, 71:022316, Feb 2005.

[Cle22] Andrew N. Cleland. An introduction to the surface code. *SciPost Physics Lecture Notes*, 49, 2022. Part of the Quantum Information Machines Session 113 of the Les Houches School, July 2019.

[Fey82] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(7-8):467–488, 1982.

[FGG14] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. Technical Report MIT-CTP/4610, MIT, 2014.

[HHL09] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103:150502, 2009.

[Kit97] Alexei Kitaev. Quantum computations: Algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191, 1997.

[RS16] Neil J. Ross and Peter Selinger. Optimal ancilla-free Clifford+T approximation of Z-rotations. *Quantum Information and Computation*, 16(11&12):901–953, 2016.

[Shi03] Yaoyun Shi. Both Toffoli and controlled-NOT need little help to do universal quantum computing. *Quantum Information and Computation*, 3(1):84––92, 2003.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

Index