Quantum Computing

Kim Bauters

May 10, 2008

Preface

Quantum computing is nothing like you have seen before. It has many possibilities and even more new hurdles. It is a concept that defies the things you took for granted and it fuels the imagination. Yet quantum computing is nothing new. It is simply how nature works: it *is* nature. It is nature working in the most bedazzling way you have ever seen.

Contents

1 Introduction

Quantum theory is an amazing thing. At the moment, it is the best mathematical model we have of the physical world. Without it, one would be at a loss to explain numerous things that happen at a very tiny scale. Though Newtonian laws hold up quite well in the big world, they start to produce errors as things get smaller. Quantum theory does not suffer from these problems and is currently still flawless; not a single prediction based on quantum theory has contradicted with the outcome of experiments to date.

Without quantum theory, it would be impossible to explain things of everyday life. One would not be able to understand the functioning of the laser, nor would one be able to fully explain as to how or why transistors work. Yet day after day billions of computers, each consisting of billions of transistors, perform a trillion number of computations. Clearly, quantum theory deals with something very real and this in itself makes it worth studying.

Studying quantum theory will also become a must in the near future. The never-ending endeavour to scale electronic circuits down further and further means that sooner rather than later one will be working in a true quantum world where quantum mechanics rules. Quantum mechanics is not some problem that one may tackle at the time one desires. Quantum mechanics is simply how nature works. Because classical computers are themselves a part of nature, it is only obvious that information processing is a part of nature. Quantum computers similarly process information, but are often able to do it in a more efficient way.

Furthermore, when taking a closer look at quantum *computing*, there seems to be an enormous potential. The factorisation algorithm of Shor is able to solve a problem in polynomial time on a quantum machine that takes an exponential amount of time on a classical machine, given our current knowledge. Though the algorithm of Shor is still mostly a theoretical matter – the number of required quantum bits is a lot bigger than any of the quantum computers researchers have yet realised – there are some more practical results that demonstrate the potential that lies within quantum computing. One such example is quantum cryptography, which provides a far more secure basis to exchange secret messages than any classical approach.

However, before one becomes too excited, it is important to note that quantum theory is a true challenge. At times, quantum theory is surprising. At times, quantum theory is counter-intuitive. And at all other times, quantum theory is downright frustrating. It is not too hard to see the enormous potential. Yet it takes a touch of genius to come up with a working algorithm that overcomes all the hurdles. Luckily, there is a solid mathematical background.

There are plenty of counter-intuitive experiments to demonstrate the strange and wonderful world of quantum physics. One well-known example is the Mach-Zehnder interferometer (see figure 1). It is rather simple, consisting only of a number of sources, detectors, full mirrors and beam splitters. Full mirrors will

simply reflect a particle. A beam splitter works likewise in only half the cases. In the other half of the cases, the beam splitter will simply let the particle pass. Some detectors are put into place and, to prevent two different particles from interfering with each other, it is ensured that the source sends out only one particle at a time.

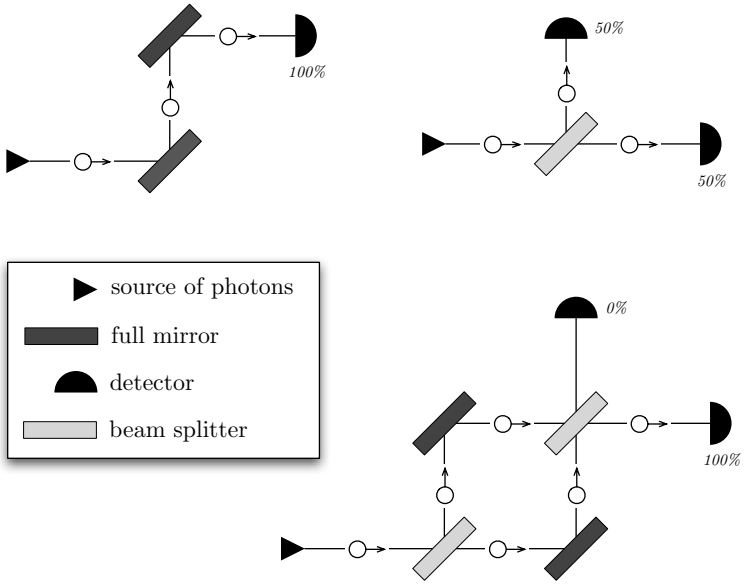


Figure 1: The Mach-Zehnder interferometer.

As long as the system is kept simple, the results are as expected. Having a number of full mirrors will simply alter the path and having a splitter will cause half the particles to go in one direction, the other half in another direction. However, a well thought out arrangement of these basic components gives a surprising result. When ensuring that the two paths overlap where the particle *might* be, the amazing find is that only *one* of the detectors perceives *all* the particles (see Figure 1).

The cause, in short, is caused by the superposition of this particle. The particle does not simply take a single path, but it takes both. And none. And only one at a single time. This results in a most remarkable fact: the particle starts to interfere with *itself* at the second splitter. It is this wonderful, yet bizarre world that will be further explored in the rest of this paper.

2 Mathematical Background

2.1 Notation

In quantum computing, the state of quantum systems is described as vectors in a Hilbert space. To be a bit more precise, the finite-dimensional vector spaces over the complex numbers are considered. In this space, a quantum bit is represented by a unit vector for which the Dirac notation is used. Though this notation may seem superfluous and at bit complex at first, it is in fact a space-saving notation that effectively highlights the information that is important for quantum computing. Besides, the Dirac notation is the de facto standard for describing quantum systems.

In Dirac notation, a vector a is written inside a so-called *"ket"* which is written as $|a\rangle$. The dual vector is written inside a *"bra"* which is written as $\langle b|$ for some vector b. The inner product of two vectors is written as *"bra-ket"* [1,2] and is denoted as $\langle c||d\rangle$ or, more commonly, as $\langle c|d\rangle$.

To get accustomed to the Dirac notation, it is useful to limit oneself to classical bits for the time being. If there is only one bit, then this bit can be in one out of two orthogonal states: the bit is either in the state 0 or the state 1. This is denoted as $|0\rangle$ and $|1\rangle$. Since these are other notations for vectors, these states can be expressed as column vectors to ease the transition:

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

If there are two bits, then these can be in the state 00, 01, 10 or 11. Likewise, these are denoted as $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$. These states can be written down more easily when regarding the zeros and ones as constituting the binary representation of an integer. This gives $|0\rangle_2$, $|1\rangle_2$, $|2\rangle_2$ and $|3\rangle_2$ where the subscript n denotes the number of bits one has or more specifically the dimensions of the basis one is working in (that is a basis of size 2^n, which is a common size to use whilst doing quantum computing with quantum bits).

Another equally valid but more elaborate way to write down the state of two bits would be to write $|0\rangle|0\rangle$, $|0\rangle|1\rangle$, $|1\rangle|0\rangle$ and $|1\rangle|1\rangle$. This notation hints at a multiplication and that is just what it is: it is a short-hand for the tensor product of two single-bit vectors. Each quantum bit can be seen as a single physical system. The state space of the combined system is the tensor product space $\mathcal{H}_1 \otimes \mathcal{H}_2$. If the state of the first system is $|\psi_1\rangle$ and the state of the second is $|\psi_2\rangle$, then the state of the combined system is $|\psi_1\rangle \otimes |\psi_2\rangle$. Remember that

[1] This makes a very handy mnemonic to keep the different names apart.

[2] This is the very definition of the dual vector. It is the vector $\langle c|$ such that the inner product of $|c\rangle$ with $|d\rangle$ is given by $\langle c|d\rangle$.

the tensor product is defined as

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

As an illustrative example, let us write out $|6\rangle_3$:

$$|6\rangle_3 = |110\rangle = |1\rangle \otimes |1\rangle \otimes |0\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}^T$$

As an easy reference, it holds that:

$$|i\rangle_b = \begin{pmatrix} \underbrace{0}_{0th} & \underbrace{0}_{1st} & \cdots & 0 & \underbrace{1}_{i-th} & 0 & \cdots & 0 & \underbrace{0}_{(2^b-1)th} \end{pmatrix}^T$$

2.2 Operations

To do some useful work with these states, operations are needed to move from one state to the other. When performing an operation, it is essential to move from one quantum state to another quantum state. Also, as stated earlier, quantum bits are unit vectors. This leads to the requirement of operations being unitary matrices; they need to be linear, norm-preserving and inproduct-preserving. The only trivial operation on classical bits that satisfies these conditions is the NOT operation, denoted as $\mathbf{X}$. The matrix representation as well as the schematic representation is given below:

$$\mathbf{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad |i\rangle — \boxed{\mathbf{X}} — |\neg i\rangle$$

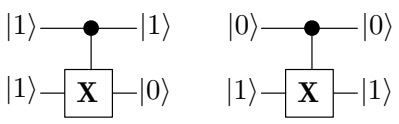
This matrix is clearly unitary since $\mathbf{X}\mathbf{X}^\dagger = \mathbf{X}^\dagger\mathbf{X} = I_2$. As a reminder, note that the dagger notation is the representation for the conjugate transpose[3] of this matrix. Applying this operation to a state effectively results in the opposite state, as desired. Another matrix is $\mathbf{Z}$, which is of little to no use in the classical world, but which becomes a useful tool in the quantum world. This matrix is such that it anticommutes with $\mathbf{X}$. This means that $\mathbf{X}\mathbf{Z} = -\mathbf{Z}\mathbf{X}$. This gives:

$$\mathbf{Z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \qquad \mathbf{Z}|i\rangle = (-1)^i \cdot |i\rangle$$

[3]This consists of two smaller steps: take the transpose of the matrix (*i.e.* switching rows and colums) and take the complex conjugated entries (*i.e.* take a complex number from $a + bi$ into $a - bi$). To summarise, for a given $\mathbf{X} = (x_{ij})$, it holds that $\mathbf{X}^\dagger = (x_{ij}^*)^T$.

Another operation is the controlled NOT or cNOT gate. This operation acts on two bits and is called a 2-Qbit gate in quantum computing. The gate can be visualised as two lines, where the first line is the control. It is only when the control line is on, that the second line will experience a NOT. Otherwise, the second line will be left untouched:

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



The last operation is the *Hadamard transformation*. Like the first two operations, it acts on only a single bit. In quantum terminology this is called a 1-Qbit gate. This Hadamard matrix has the property that $\mathbf{HXH} = \mathbf{Z}$. This is a real quantum gate that will prove to be very useful. It is defined as:

$$\mathbf{H} = \frac{1}{\sqrt{2}}(\mathbf{X} + \mathbf{Z}) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

An important thing to note is that the requirement of being unitary is quite a hefty one since, by consequence, all the operations are reversible. At first glance, this would mean that a quantum computer would be unable to simulate a classical computer because most operations performed on classical bits are not reversible. Luckily this is not the case as any non-reversible operation can be transformed into a reversible one through a very small trick. If the function one wants to apply is f and one wants to apply this on some x, then $\forall c : (x, c) \rightarrow (x, c \oplus f(x))$ is reversible. This is quite handy, since one can simply let c be $|0\rangle$, thus having $(x, |0\rangle) \rightarrow (x, f(x))$ since $|0\rangle \oplus f(x) = f(x)$.

2.3 Quantum states and measurement

A quantum bit, or in short a Qbit, is not simply limited to being one out of two orthogonal states like a classical bit. Rather, a Qbit can be in any superposition of these two states. A Qbit is defined as:

$$|\Psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix},$$

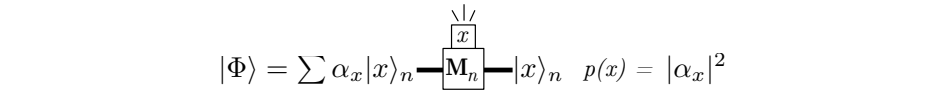
$$\text{where } \mathbf{Z}|\Psi\rangle = \alpha_0\mathbf{Z}|0\rangle + \alpha_1\mathbf{Z}|1\rangle$$

and where α_x is called an *amplitude*. A limitation originates from a Qbit being a unit vector, as this requires that $|\alpha_0|^2 + |\alpha_1|^2 = 1$.[4] In general, for n Qbits, this gives:

$$|\Psi\rangle = \sum_{0 \leq x < 2^n} \alpha_x |x\rangle_n, \text{ where } \sum_{0 \leq x < 2^n} |\alpha_x|^2 = 1$$

[4]Note that this limitation also holds for classical bits: $|1\rangle = 0 \cdot |0\rangle + 1 \cdot |1\rangle$, so $|0|^2 + |1|^2 = 1$.

This seems to indicate that n Qbits contain information on 2^n complex numbers. However, there is a *major* catch. In the classical world, measuring is a non-disruptive process. Sadly, this is far from the truth in quantum computing. When sending a Qbit through a measurement gate – which is denoted as $\mathbf{M}$ – its state will collapse according to the Born rule. This rule states that the chance of collapsing to a certain state $|x\rangle$ is given by $p(x) = |\alpha_x|^2$:



Though the amplitudes may hold a lot of information, they are only used during measurement to statistically determine the outcome. No information can be obtained on what these amplitudes are/were. Measurement in quantum computing is irreversible. This is a major drawback. To be able to write useful programs, a clever arrangement of the operations on a certain Qbit is needed so that the amplitudes (and thus the chance of collapsing to that particular state) are very high for the desirable results and negligible for the others. Another approach may be to use interference so that it does not matter what is measured, as the information is contained globally in the type of correlations. Regardless, this implies that classical computers will remain needed. They will be used to verify the results (as faulty results with small – but non-zero – amplitudes may still be selected) and to perform all those tasks that are currently already efficiently solvable on classical computers.

One may question whether these amplitudes exist altogether. Luckily, this can be proven. The Hadamard transformation that was defined earlier, is so useful in quantum computing because it will assign equal chances to all states. If one applies H to the state $|1\rangle$ and then measure, the result is the state $|0\rangle$ or the state $|1\rangle$ with an equally likely probability:

$$\mathbf{H}|1\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \qquad x \in \{0,1\} : p(x) = \left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}.$$

If one does not measure just yet and apply another Hadamard once more, one can experimentally verify that the state is once again $|1\rangle$ with a probability $p(1) = |1|^2 = 1$:

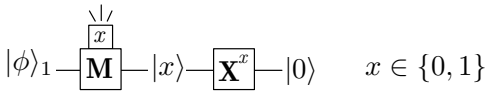
$$\mathbf{HH}|1\rangle = I_2|1\rangle = |1\rangle$$

This result agrees with the unitary nature of the Hadamard matrix and could not be possible if the states would collapse in between of two applications of the operation. If the state did in fact collapse after every operation, the chance would only be $p(1) = |-1/\sqrt{2}|^2 = 50\%$:

$$\mathbf{H}|x\rangle = \frac{|0\rangle \pm| 1\rangle}{\sqrt{2}} \qquad x \in \{0,1\} : p(x) = \left|\pm\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}.$$

Apparently it is possible to measure a single Qbit, but is it possible to measure multiple Qbits – or a composed system – in a practical way? This problem is one of a practical nature. Creating physical 1-Qbit gates is a tough job. Creating 2-Qbit gates is even more formidable a task. Any gates larger than 2-Qbit gates are most likely out of our reach for the foreseeable future. Fortunately, measuring multiple Qbits is easy thanks to the generalised Born rule. This rule states that measuring a single Qbit (and thus collapsing this Qbit to a particular state) does not influence the other unmeasured Qbits when those Qbits are not entangled. To measure a large number of Qbits, it is as easy as measuring them one by one, since measuring one Qbit will not affect the others.

Measurement gates are not all bad though. Like so often in the world of quantum computing, a seeming limitation can be transformed into an advantage. As already briefly mentioned when discussing reversible computing, it is important to be able to choose the initial state of a Qbit, very often being $|0\rangle$. Yet a Qbit found in the wild may be in any superposition. Luckily, simply measuring this Qbit will cause the state to collapse and the amplitudes to vanish. The act of measurement will reveal whether the Qbit is now in the state $|0\rangle$ or $|1\rangle$. Depending on the result, an additional application of $\mathbf{X}$ ensures that the state is the desired $|0\rangle$:



2.4 Composition and entanglement

Clearly it is possible to compose two systems into a bigger one. However, not every Qbit can be decomposed into two smaller Qbits. In such a case, the Qbits are *entangled*: the value of one Qbit depends on the value of the other. One such an example would be $\frac{1}{\sqrt{2}}(|0\rangle_2 + |3\rangle_2)$, which is one of the Bell states. No matter how hard one tries, one cannot write this as a product state of two or more Qbits. This seems to be an amazing source of power for quantum computing: determining the outcome of one Qbit will immediately influence the outcomes of the other entangled Qbits without any further interaction on our part.

A good way to see this in action is to rewrite the Bell state that was given as an example. Instead of writing $\frac{1}{\sqrt{2}}(|0\rangle_2 + |3\rangle_2)$, it is possible to write this as $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. The state will thus either collapse to $|00\rangle$ or to $|11\rangle$. When the first Qbit collapses to $|0\rangle$, it immediately follows that the second Qbit will also collapse to this state. In an unentangled system, the outcome of one Qbit would not be related to the outcome of another, as described in the previous section.

3 Quantum Model of Computation

3.1 Computing with Qbits

The idea of a quantum computer is that it should take some input x and produce the result $f(x)$ for some specified function f. If the numbers x are specified as n-bit integers and the numbers $f(x)$ as m-bit integers, then one needs $n + m$ Qbits. These are called the input and output register, respectively.[5] It may seem strange to not simply have a single register in which one transforms the input into the output. However, a lot of useful functions have the same output $f(x)$ for different values of x. In such a case, the computation is not reversible and it would be impossible to compute on a quantum computer.[6] In general, there will have a transformation $\mathbf{U}_f$ such that:

$$\begin{aligned}\mathbf{U}_f(|x\rangle_n|y\rangle_m) &= |x\rangle_n|y \oplus f(x)\rangle_m \\ &= |x\rangle_n|f(x)\rangle_m \Leftrightarrow |y\rangle_m = |0\rangle_m\end{aligned}$$

If the output register is set to a start value of $|0\rangle_m$, then it contains the computed value after the transformation $\mathbf{U}_f$. Note that this computation is reversible, since applying $\mathbf{U}_f$ once again will result in the initial state. Indeed, $|0 \oplus f(x) \oplus f(x)\rangle_m = |0\rangle_m$ since $z \oplus z = 0$.

Let us now take a closer look at the input register and see how to come to one of the most important tricks in quantum computing. If one applies the Hadamard transformation $\mathbf{H}$ to some register, this results in an equally weighted superposition of all possible states (each state is equally likely to be picked):

$$\mathbf{H}^{\otimes n}|0\rangle_n = \frac{1}{2^{n/2}} \sum_{0 \le x < 2^n} |x\rangle_n \text{ where } \mathbf{H}^{\otimes n} = \underbrace{\mathbf{H} \otimes \mathbf{H} \otimes \ldots \otimes \mathbf{H}}_{n \text{ times}}$$

It is as if the input register contains all possible inputs, all at once. Applying $\mathbf{U}_f$ on this superposition results in a state that seemingly holds the result of all 2^n evaluations of the function f. This apparent miracle of so easily evaluating 2^n possibilities is called *quantum parallelism*:

$$\mathbf{U}_f\mathbf{H}^{\otimes n}|0\rangle_n = \frac{1}{2^{n/2}} \sum_{0 \le x < 2^n} |f(x)\rangle_n$$

However, one cannot say that the result of the calculation is in fact 2^n evaluations of f. All that can be said is that those evaluations *characterise the form* of the state that describes the output of the computation, as it is impossible to learn the *state* of a Qbit without actually performing those 2^n computations. After measuring – and the resulting collapse of state – all one has learned is a

[5] This register approach is only one of many possible views.

[6] See also page 6.

single value of f and the x_0 where f has that value. Yet, the miracle is not all gone. The random selection of the x for which one can learn the value $f(x)$ is only done *after* the computation has been carried out.[7]

One may think that one need not worry about this strangeness. Since it is only the act of measuring that causes the state to collapse, it is not too hard to see that a simple trick makes it possible to obtain the 2^n results. If one copies the Qbit before measuring, then measuring this copy and repeatedly making new copies would make it possible to obtain the 2^n results. Alas, the quantum world once again proves how very different it is from the classical world; copying a Qbit is not generally possible. This is an immediate consequence of linearity. Consider that a cloning operator $\mathbf{C}$ exists:

$$\mathbf{C}(|\psi\rangle|0\rangle) = |\psi\rangle|\psi\rangle \text{ and } \mathbf{C}(|\phi\rangle|0\rangle) = |\phi\rangle|\phi\rangle$$

It then follows from linearity that:

$$\mathbf{C}(a|\psi\rangle + b|\phi\rangle)|0\rangle = a\mathbf{C}|\psi\rangle|0\rangle + b\mathbf{C}|\phi\rangle|0\rangle = a|\psi\rangle|\psi\rangle + b|\phi\rangle|\phi\rangle$$

But also that:

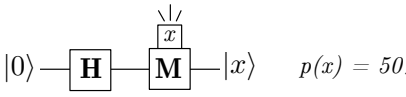
$$\begin{aligned}\mathbf{C}(a|\psi\rangle + b|\phi\rangle)|0\rangle &= (a|\psi\rangle + b|\phi\rangle)(a|\psi\rangle + b|\phi\rangle) \\ &= a^2|\psi\rangle|\psi\rangle + b^2|\phi\rangle|\phi\rangle + ab|\psi\rangle|\phi\rangle + ab|\phi\rangle|\psi\rangle \;\lightning \quad \square\end{aligned}$$

Which is clearly not the same, thus proving by contradiction that a cloning operator does not exist and as a result that Qbits cannot in general be cloned.

3.2 Basic algorithms

3.2.1 Random number generator

Before looking at some of the algorithms that offer a speedup thanks to quantum computing, let us first look at some algorithms that are surprisingly simple because of a very clever use of the quantum way of doing things. One of the simplest algorithms is a true random number generator. Whereas in classical computing random numbers are always pseudo-random, a quantum computer can generate true random numbers. The algorithm to do so is surprisingly simple and uses nothing but the most basic 1-Qbit gates. All one needs to do is apply a Hadamard transformation to an initial state of a single, random Qbit and measure the state to obtain a truly random $|0\rangle$ or $|1\rangle$:



[7] This must be one of the greatest mind-benders in quantum computing and it is appropriately called quantum *weirdness*: it is only after the computation is done that the x is selected, but since this selection is totally random the time of the selection seems to be irrelevant.

3.2.2 Crypto

Another algorithm that is – surprisingly – very simple is an algorithm for quantum cryptography. An effective way to secure messages is to use a key that is only known by the sender and the receiver. If one takes a message of size n bits and a key of size k bits, one can chop up the message in $\lceil \frac{n}{k} \rceil$ parts. If one *xor*s each part with the key and appends the different encrypted parts, one ends up with the encrypted message. Obtaining the original message is as simple as applying the process once again.

message	00100101		encrypted	10101111	
key	10001010	$\oplus$	key	10001010	$\oplus$
encrypted	10101111		message	00100101	

If the size of the key is sufficiently long, any attempt to decode the message through a brute-force approach would be futile. This approach does require it to be easy to generate new keys. Reusing the same key is not secure and most often the key is taken to be $k = n$.

One major issue with this approach is that the key needs to be known by both parties. Various techniques to exchange a key exist, but each technique has a certain degree of insecurity against eavesdropping. When using quantum computing however, the degree of insecurity can be made arbitrarily small. Here is how this is done.

If Alice wants to create a key that both she and Bob know, she starts by sending a number of Qbits to Bob. For each Qbit, Alice can choose to send one out of four possible Qbits. She either sends $|0\rangle$ or $|1\rangle$, which are called type-1 Qbits, or she sends $\mathbf{H}|0\rangle$ or $\mathbf{H}|1\rangle$, which are called type-H Qbits. Having these two types of Qbits will make it possible for Alice and Bob to work together and detect a possible eavesdropper.

Bob receives the Qbits one by one and applies at random a type-1 or type-H measurement for each Qbit he receives. A type-1 measurement will simply measure, but a type-H measurement will first apply a $\mathbf{H}$ before measuring. When Alice is done sending the prepared Qbits to Bob, Alice shares with Bob over an insecure channel which Qbits she prepared as type-1 or type-H Qbits. Note that she does not divulge whether it was a $|0\rangle$ or a $|1\rangle$ that she prepared; she only tells the *type* of how she prepared each Qbit and thus whether she applied a H before sending or not. Bob verifies this information with his choices and he informs Alice for which Qbits he made the same type of measurement (in the example below he made the same decision for Qbit number 2, 4, 6 and n). If no one was eavesdropping, they now both possess the same key since $H^2 = I_2$.

	1	2	3	4	5	6	...	n
Alice	H	H	H	1	1	1		1
Bob	1	H	1	1	H	1		1

If someone, Eve, eavesdrops by listening to the quantum channel, then she too needs to measure the Qbits she intercepted with a random type of measurement before sending them through to Bob (which must be done, because Qbits cannot be cloned). Like Bob, she only has 50% chance to pick the right type of measurement. In 50% of *these* cases, she will have influenced the results that Bob obtains because of the random application of H in both the preparation and the measurement of the Qbits (see Figure 2).

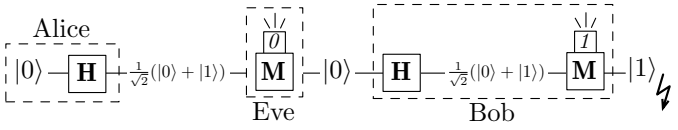


Figure 2: A possible example of how the actions of Eve disrupts the quantum.

If Alice and Bob now sacrifice a number of their commonly chosen bits to compare them over an insecure channel, they can detect the presence of Eve and they can statistically determine how secure their new key is! So the randomisation of 1-type and H-type is the source of the security of this approach.

3.2.3 Deutsch's problem

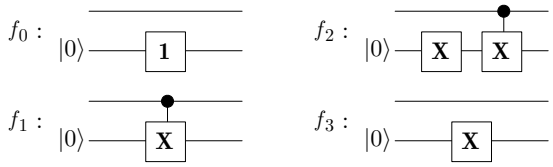
A quantum algorithm that demonstrates how quantum computing can result in a speedup over classical computing is Deutsch's Problem. Consider the most simple classical computer. Such a computer has only a single bit in the input register and a single bit in the output register. With only two possible values for the input register and two possible values for the output register, it is easy to see that there are only 4 possible functions:

$f_0 : 0 \to 0 \quad 1 \to 0$

$f_1 : 0 \to 0 \quad 1 \to 1$

$f_2 : 0 \to 1 \quad 1 \to 0$

$f_3 : 0 \to 1 \quad 1 \to 1$



The figures on the right hand side depict in a schematic way what the functions may be. The top line is always the input line whereas the bottom line is the output line. Depending on the desired result, there will be some transformation on the output line and/or the input line will interact with the output line.

When one does not know which function the computer performs, then what is there to learn when one is allowed to run the computer only once? In classical computing, all one can do is feed some input (being it either $|0\rangle$ or $|1\rangle$) and looking what the output is that is generated. For example, if one inputs a $|0\rangle$ and a $|0\rangle$ pops out, all one can say is that the function will be either f_0 or f_1.

A quantum computer can do better. Using a cunning arrangement of operators, it is possible to learn whether or not the function is stable in only a single application. That is, it is possible to learn in a single application whether or not the output is the same, regardless of the input (however, it is important to know that one will not be able to learn what the output actually *is*. All one will be able to say is if the output is the same or not when the input varies). Doing the same thing with a classical computer would require two applications.

Once again the Hadamard transformation plays a major role. This time around, the Hadamard is used in a different way and the stability of the function will be represented by a simple sign change. For this to work, the Hadamard is applied to $|1\rangle|1\rangle$ instead of $|0\rangle|0\rangle$. Afterwards applying the function $\mathbf{U}_f$ might cause a sign change in the *input* register, which indicates the stability of the function. Applying the Hadamard once again to the input register will, thanks to the sign change, gives the desired answer as follows;

$$\begin{aligned}
&(\mathbf{H}\otimes\mathbf{I})\mathbf{U}_f(\mathbf{H}\otimes\mathbf{H})(\mathbf{X}\otimes\mathbf{X})(|0\rangle|0\rangle)\\
&=(\mathbf{H}\otimes\mathbf{I})\mathbf{U}_f(\mathbf{H}\otimes\mathbf{H})(|1\rangle|1\rangle)\\
&=(\mathbf{H}\otimes\mathbf{I})\mathbf{U}_f(\mathbf{H}|1\rangle\otimes\mathbf{H}|1\rangle)\\
&=(\mathbf{H}\otimes\mathbf{I})\mathbf{U}_f\left(\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)\otimes\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)\right)\\
&=(\mathbf{H}\otimes\mathbf{I})\mathbf{U}_f\cdot\frac{1}{2}(|0\rangle|0\rangle-|1\rangle|0\rangle-|0\rangle|1\rangle+|1\rangle|1\rangle)\\
&=(\mathbf{H}\otimes\mathbf{I})\cdot\frac{1}{2}(\mathbf{U}_f(|0\rangle|0\rangle)-\mathbf{U}_f(|1\rangle|0\rangle)-\mathbf{U}_f(|0\rangle|1\rangle)+\mathbf{U}_f(|1\rangle|1\rangle))\\
&=(\mathbf{H}\otimes\mathbf{I})\cdot\frac{1}{2}(|0\rangle|f(0)\rangle-|1\rangle|f(1)\rangle-|0\rangle|\tilde{f}(0)\rangle)+|1\rangle|\tilde{f}(1)\rangle)
\end{aligned}$$

where $\tilde{f}(0)$ is $|1\oplus f(0)\rangle$ instead of $|0\oplus f(0)\rangle$. Note that, when $f(0)\neq f(1)$, then $f(0)=\tilde{f}(1)$ and $f(1)=\tilde{f}(0)$. Rewriting the terms gives:

$$\begin{aligned}
&=\begin{cases}(\mathbf{H}\otimes\mathbf{I})\cdot\frac{1}{2}(|0\rangle-|1\rangle)(|f(0)\rangle-|\tilde{f}(0)\rangle) & f(0)=f(1)\\ (\mathbf{H}\otimes\mathbf{I})\cdot\frac{1}{2}(|0\rangle+|1\rangle)(|f(0)\rangle-|\tilde{f}(0)\rangle) & f(0)\neq f(1)\end{cases}\\
&=\begin{cases}|1\rangle\frac{1}{\sqrt{2}}(|f(0)\rangle-|\tilde{f}(0)\rangle) & f(0)=f(1)\\ |0\rangle\frac{1}{\sqrt{2}}(|f(0)\rangle-|\tilde{f}(0)\rangle) & f(0)\neq f(1)\end{cases}
\end{aligned}$$

Amazingly, the input register is now in the state $|1\rangle$ when the function $\mathbf{U}_f$ is stable, but one loses all information on the actual values of x or $f(x)$. Apparently, in quantum computing, it is possible to renounce the ability to gain the actual values in order to obtain information on the relation between the values.[8] So information about the correlations of all the branches can sometimes be accessed!

[8]This problem is the same as determining whether the millionth bit of $\sqrt{2}$ is the same as the millionth bit of $\sqrt{3}$. In quantum computing, determining if these two bits are the same is no harder than to actually determine what the millionth bit of either $\sqrt{2}$ or $\sqrt{3}$ is.

4 The Algorithms of Shor and Grover

One of the most amazing, yet one of the most misunderstood algorithms in quantum computing must be the algorithm of Shor. Explaining the algorithm in detail is no small feat and it would be unwise to do so in the limited space of this paper. However, it is important to learn how this algorithm works on a conceptual level in order to discard a large number of the hyped up expectations.

Basically, the algorithm of Shor allows one to crack RSA encryption in polynomial time on a quantum computer. On a classical computer, this problem is known to be NP-hard (But it is not known to be NP-complete!). The reason why cracking RSA encryption is NP-hard on a classical computer is because it relies on the difficulties to factorise a large number into two prime numbers.[9] If one could speed up this part of the process, one may be able to leave the realm of NP and move to P.

What Shor's algorithm does not do is try out all possible divisors in parallel and magically come up with the correct one. An algorithm that actually does this is the algorithm of Grover, but this algorithm will be discussed later on in this section. However, The Deutsch's algorithm already demonstrated that a quantum computer can sometimes tell something about the relationship between the data that is stored in the superpositions. For this to work, one first needs to find some sort of relationship that can be exploited. Luckily, the factorisation problem has oodles of special properties that can be exploited in an algorithm.

One of those special properties is that the factorisation problem can be reduced to the problem of period finding. Take for example the sequences $2^x \bmod 15$ and $4^x \bmod 15$, then one finds that these are periodic:

$$\underbrace{2, 4, 8, 1}_{\text{period is 4}}, 2, 4, 8, 1, 2 \ldots$$

$$\underbrace{4, 1}_{\text{period is 2}}, 4, 1, 4 \ldots$$

If N is the product of two prime numbers p and q, then the series

$$y \bmod N, y^2 \bmod N, y^3 \bmod N, y^4 \bmod N \ldots$$

has a period that divides $(p-1)(q-1)$, given that y is not divisible by p or q. For example, if $N = 15$, then the prime factors of N are $p = 3$ and $q = 5$, so $(p-1)(q-1) = 8$. And indeed, two periods one can find are 2 and 4, which both divide 8. If this step is repeated a number of times for different values of y, it is possible to learn the desired number of divisors of the totient and put them together to learn $(p-1)(q-1)$. Some more tricks make it possible to recover p and q afterwards, but these fall outside the scope of this paper.

[9] Recall that RSA consists of both a public and a private key. Both keys contain N, which is the product of two prime numbers. Obtaining these two prime numbers allows one to repeat the steps needed to obtain the private key, thus allowing one to decode the secret message.

The catch with this approach on a classical computer is that the series

$$x \bmod N, x^2 \bmod N, x^3 \bmod N, x^4 \bmod N \ldots$$

eventually repeats itself, but the number of steps before it repeats might be as large as N itself. This new problem is no easier to solve than the factorisation of the number N. A quantum computer can do better than any known classical approach because of the superposition one can create over all the numbers in the sequence. Note that this is not like having a superposition over the possible divisors of N. In that case, there was no idea of where to look. This time around, one seeks a relation between these numbers, namely their period.

Creating a superposition $\sum x^r \bmod N : 0 < r \leq N$ is not too hard when one uses the modulo calculus. For example, to compute $3^{14} \bmod 17$:

$$\begin{aligned} 3^{14} \bmod 17 &= (3^{2^3+2^2+2^1}) \bmod 17 = (3^{2^3} \cdot 3^{2^2} \cdot 3^2) \bmod 17 \\ &= ((3^2)^2)^2 \cdot (3^2)^2 \cdot 3^2 \bmod 17 = (9^2)^2 \cdot 9^2 \cdot 9 \bmod 17 \\ &\equiv \text{(using various properties)} \ 13^2 \cdot 13 \cdot 9 \bmod 17 \\ &\equiv \text{(using various properties)} \ 2 \bmod 17 \end{aligned}$$

All that is left – since it is clearly possible to build a superposition in a reasonable amount of time – is to find the period of the sequence. This is the essence of the algorithm of Shor and the part that crucially depends on quantum mechanics. To get the period, Shor uses the quantum Fourier transform.

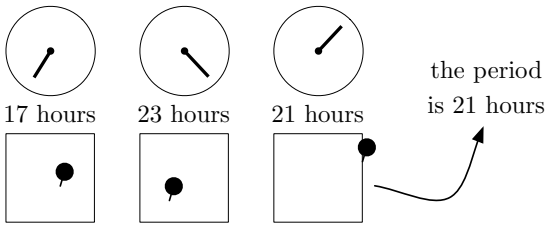
To understand how it works,[10] consider a number of students who each have their own rhythm. The first student has a normal 24-hour day. He gets up at 8 in the morning and goes to bed at midnight. Another student may have a 23-hour day; he gets up at 8 one day, goes to bed at midnight, gets up at 7, goes to bed one hour before midnight and so on. The question of finding a period is the same as finding out whether a particular student is on an x-hour schedule.

Now assume that this student can help you out. He lives in a room that is filled with a number of clocks with only a single hand to indicate the hour. Additionally, each clock has a different revolution time. One of those clocks is an ordinary clock that makes a revolution every 12 hours, but there are also other clocks, *i.e.* clocks that make a revolution every 25 hours or every 17 hours. Underneath each clock is a posterboard with a thumbtack on it. Every time the particular student wakes up, he looks at each clock and he moves the thumbtack one centimetre in the direction that the hand is pointing.

When the revolution time is not the same as the period, the thumbtack will move around the board, but will eventually end up in the same place as where it started. At times, the hand will point in one direction, but at other times, the hand will point in the opposite direction which will effectively cancel out

[10] Full credit goes to Scott Aaronson for coming up with this wonderful, down-to-earth explanation. It is only natural that the person who explained it in an understandable way deserves as much kudos as the inventor of the algorithm.

the previous movement of the thumbtack. The only posterboard where one will see something different is the one with exactly the same revolution time as the period of the student. On this board, the hand will always point in the same direction when the student gets up and the thumbtack will continue to move in one direction until it reaches the outer parts of the posterboard. This is basically what the quantum Fourier transform does. It moves thumbtacks around and by doing so it gains insight in the period of the sequence:



More specifically, the quantum Fourier transform is an operation that transforms a number of Qbits in such a way that their amplitudes cancel each other out (this is called *interference*) in the case where they are not the actual period.[11] In the case that they are the period, the amplitudes will continue to enforce each other and one will effectively obtain an amplitude close to 1 that is associated with the answer state. As so often, one starts out with a superposition of all the possible periods in such a way that they all have equal likelihood (the thumbtack is centred on all boards when we start).

The algorithm of Shor is a great example of how problems can be sped up thanks to quantum computers. The key trick is to use the quantum computer in a cunning way and letting it solve the hard part in such a way that a speedup is obtained. However, it is essential to realise that the algorithm of Shor only works because of some inherent property of the problem, *i.e.* the periodic structure. Without this property, even a quantum computer has no general approach for solving the problem in polynomial time.

Another important quantum algorithm is the algorithm of Grover. This algorithm tackles the problem of finding an item x that satisfies a given condition in a database. The unsorted database itself contains N items and x is one of them. Also, when an item is examined, it is possible to determine in a single step whether or not this item is the specific one, *i.e.* the solution to a problem.

Given a classical computer, an algorithm to solve this problem would need to look at an average of $\frac{N}{2}$ items before it finds the one that satisfies the condition. Basically, the only thing that a classical computer can do is take the item and see if it is or or is not the required item. The only thing that can help it in its search is to remember which items were *not* the ones that exhibited the property in order not to pick them again.

[11] This is possible because amplitudes can be complex, so one amplitude can cancel out the other. This is a consequence of the *particle-wave duality* of qubits. Like waves, qubits can enforce each other or cancel one another out.

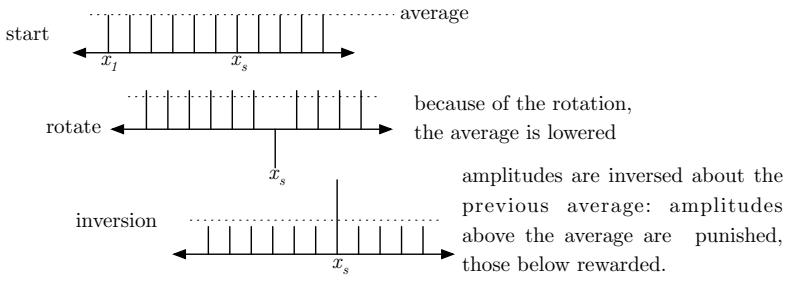
Interestingly, a quantum computer can do better; Grover's algorithm needs only $O(\sqrt{N})$ steps to find the desired item. Though the performance increase is nothing like the one found in the algorithm of Shor – taking a problem from NP to P is no small feat – it does offer an amazing speedup in practice. Consider one needs to look for a specific element in a set of 2^{32} possible elements. A classical computer, without any additional information, could do no better than examining on average two trillion items before finding the "special" one. Using the algorithm of Grover, a mere 65 thousands steps will suffice. While the problem is still in NP, nevertheless the algorithm deserves a lot of credit for obtaining a speedup for problems as basic as finding an element in a set.

The algorithm of Grover works as follows. The amplitudes of the different items are massaged in such a way that the desired solution obtains a high amplitude and thus the chance of measuring this outcome becomes high. Because there is no property to exploit, the algorithm itself is a lot simpler – and slower – than the quantum Fourier transform. It consists out of only three parts:

1. the input register is prepared so that all items have the same amplitudes, *i.e.* they are equally likely to be picked;
2. massage the amplitudes $\frac{\pi}{4}\sqrt{N}$ times to increase the amplitude of the solution to the desired level. This is done by looking at the outcome of the condition checker C where $C(y) = 1$ when $y = x$ and otherwise $C(y) = 0$;
3. measure the state and verify if the item measured is indeed the desired one. This is the case with a probability of nearly 1.

Note that one needs to iterate the 2nd step just the right number of times. Too few iterations results in the probability of picking the desired item to be less than 1, while too many iterations degenerate the amplitude once again and the probability of picking the desired item once again falls below 1.

Each time the amplitudes are massaged, two things happen: the amplitudes are flipped whenever $C(z) = 1$ and there is an inversion about average:



After the correct number of iterations, measuring will result in x_s with a high probability. To learn more about the algorithm of Grover, it is worthwhile to take a look at the paper of Grover. The paper is very concise, well written and all in all a pleasure to read [6].

5 Quantum Error Correction

Quantum Error Correction is a recent advancement in quantum computing and one of fundamental necessity in order to allow functional quantum computers. Without error correction, the operation of all but the smallest quantum computer is problematic. The main cause of problems is the scale. Since one cannot expect to operate in a perfectly isolated environment, there will always be factors that will influence the Qbits. Some examples of such influences are small temperature fluctuations and/or particles that are not part of the quantum computer that start to entangle with particles that are part of it.

Though quantum error correction is necessary, it is a very hard problem to tackle. Only in the last decade mechanisms were found to perform quantum error correction. That this is even possible is in itself a reason to rejoice. Unlike classical computing, one is not dealing with simple states like $|0\rangle$ or $|1\rangle$, but one is dealing with states that have complex amplitudes associated with them. Measuring the states to detect errors is out of the question, since this would cause the state to collapse and lose all the desired information. Even copying general Qbits to prevent errors is out of the question, as dictated by the no-cloning theorem. Still, this problem has been overcome.

Before rushing ahead, let us take a look at a very crude error correction code for classical computing, the 3-bit error correction code. Each bit is encoded in three bits to be able to detect errors. So $|0\rangle$ and $|1\rangle$ are replaced as follows:

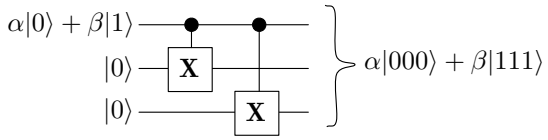
$$|\overline{0}\rangle = |0\rangle|0\rangle|0\rangle = |000\rangle \qquad |\overline{1}\rangle = |1\rangle|1\rangle|1\rangle = |111\rangle$$

By monitoring these codewords, one can check for possible bit flips and correct them in time before they become problematic, given that the probability of possible bit flips is low enough. When one receives 010, one can apply the rule of the majority and correct it to 000. Likewise 110 is corrected to 111. If one corrects frequently enough, one avoids the chance of having two bit flips that could cause erroneous corrections. Sadly enough, this entire mechanism relies on measurements and on having multiple versions of the same Qbit. If one wants this idea to work in quantum computing, one will have to come up with a more ingenious and less intrusive way.

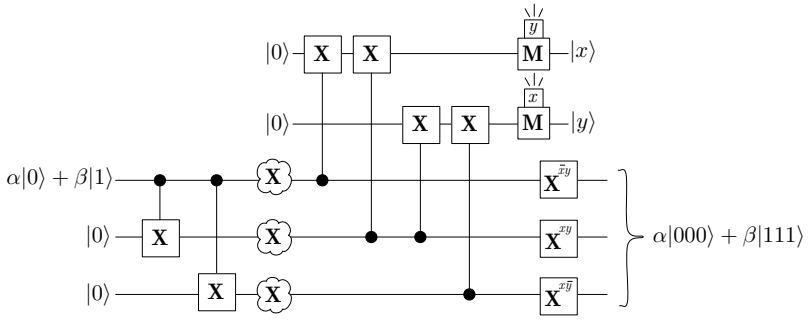
An artificial model of a quantum computer can help to develop the required notions and techniques for error correction codes in the quantum world. In this model only bit flips can occur.[12] Such a simplification is a lot like the classical case, making it worthwhile to try to create a 3-Qbit code for error correction. Remarkably, this can be made to work, though the encoding becomes a lot more important and the error correction itself is done in a far more subtle way than in classical computing.

[12]Having only bit flips is not much like the conditions in an ordinary quantum environment. For example, phase errors where a + becomes a - or vice versa are equally devastating. Qbits can even become entangled with their environment, which is clearly not something one desires! The latter phenomenon is called *decoherence* and it is rampant in any quantum computer.

The first thing that is needed is an operation $|\overline{0}\rangle = |0\rangle|0\rangle|0\rangle$. The generalisation of this for a Qbit in the state $\alpha|0\rangle + \beta|1\rangle$ is an operator such that $\alpha|\overline{0}\rangle + \beta|\overline{1}\rangle = \alpha|000\rangle + \beta|111\rangle$. This operator must work without replying on any of the amplitudes and must be universally applicable. Luckily, while it is not possible in general to find a universal operator to clone a Qbit, it is possible for every Qbit to find an operator that clones it into the same state (*e.g.* $|0\rangle$) or the orthogonal state (*e.g.* $|1\rangle$). For the given example, the case that is of importance for quantum error correction, this can be accomplished through the use of two cNOT gates:



The assumption was that there are only bit flips. This means that there may or may not be an **X** that is applied on one of the lines, depicted as an **X** operator in a cloud in the figure below. In order to detect an error, one will have to make some measurement. Fortunately, there is an indirect way to obtain the information one wants without actually measuring the current Qbits. For this, one will introduce two auxiliary Qbits that tell something about the relation between the Qbits and nothing about the state.[13] This gives enough information to determine how to correct the codeword.

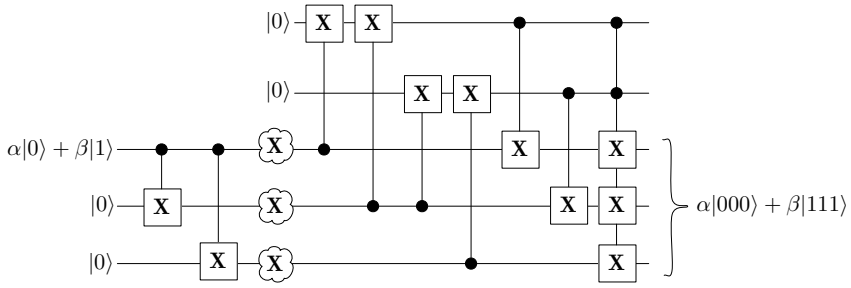


Amazingly, this simple error correction code allows us to obtain enough information to correct the Qbits to their error-free state without obtaining any information whatsoever about the *actual* state of the Qbits. For example, consider the case where the second line of the codeword experiences a bit flip. This results in the state $\alpha|010\rangle + \beta|101\rangle$. The consequence of this bit flip on the auxiliary bits is that they both will experience the effect of a single CNOT gate,

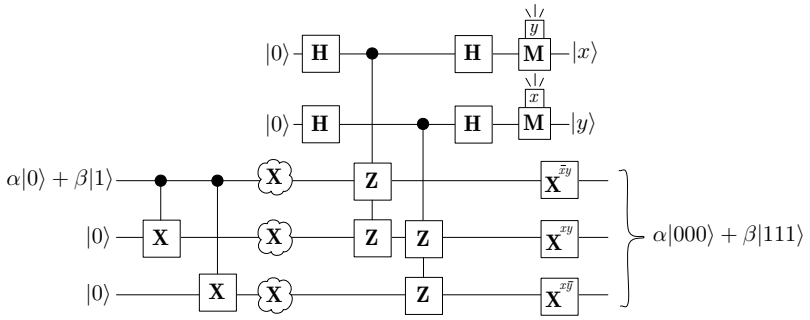
[13] One can only obtain information on the *relation* between the Qbits. If one could obtain even the slightest information on the state, it would be possible to repeat the operation in order to obtain complete information. This would be a direct violation of the no-cloning theorem.

resulting in $|x\rangle = |y\rangle = |1\rangle$. Therefore, the $\mathbf{X}$ gate on the bottom line will not be applied since $\mathbf{X}^{1\cdot 0} = I_2$. The $\mathbf{X}$ operation on the second line is executed, as $\mathbf{X}^{1\cdot 1} = \mathbf{X}$. The third line will remain unchanged. The net result is that the state $\alpha|010\rangle + \beta|101\rangle$ is corrected to the state the state $\alpha|000\rangle + \beta|111\rangle$.

This can be done regardless of possible entanglements with other Qbits. Even better is that one can do without measurement gates and automate the process at the same time. This allows to encode error correction in the quantum program itself, making it possible to correct errors as it is executing. To do this, the measurement gates and the associated applications of $\mathbf{X}$ are replaced by two cNOT gates and a double-controlled NOT or ccNOT gate. Such a ccNOT gate will only trigger an $\mathbf{X}$ operation when two Qbits are "on".



Constructing such schemes makes the working of the error correction more clear. However – and this is especially true in more elaborate error correction models – the space taken by such a scheme can become troublesome. A property that is often applied is to use the equality $\mathbf{HXH} = \mathbf{Z}$ and $\mathbf{HZH} = \mathbf{X}$. The original scheme, without automatic correction, has been adapted using this idea:



Though the scheme for the 3-Qbit error correction code is equally large, it hints at some benefits when moving to more potent codes. A quantum error correction code that is capable of efficiently coping with all possible errors in a quantum computer is the 7-Qbit error correction code. This code has 7 main Qbits and 6 auxiliary Qbits, rationalising the use of the aforementioned property to come up with more compact schemes.

6 Addendum

Though this paper may have sparked your interest in the wonderful world of quantum computing, it is actually quite a concise introduction to this world. For the interested reader, this paper is a stepping stone towards a deeper exploration of the field. A number of items that were not discussed in this paper but that may interest you are quantum dense coding, teleportation of Qbits and the physics behind errors in quantum computing.

Yet obtaining more information on the subject may prove to be an insurmountable task. Ever since the first reports in the 30s, quantum computing has made people dream of additional dimensions where these massive computations take place or made them think of computers that were able to solve any problem in an instant. Though far from the truth, these are only a few of the uncountable number of misconceptions that surround quantum theory. Ideas like "*quantum computers can perform exponential computations in polynomial time*" or "*Qbits are two bits*" are omnipresent and prevent you from obtaining valid information on the subject.

On top of that, quantum computing is rather multidisciplinary. It is not limited to computer science and mathematics as it finds it origins in physics and engineering disciplines. If valid information is obtained, it may prove to be illegible due to the complexity or the assumption of prior knowledge that is not there. Luckily, to write this paper, a lot of people have helped to point at very helpful documentation that is both legible and thorough. Though the list of resources I am about to present is far from complete, it will provide you with a broad enough basis to skim your way through other documentation you may find and separate the useless information from some of the true gems that are to be found.

The book that has been my trustworthy guide in the world of quantum computing must be the book on quantum computing written by David Mermin [1]. The book is based on years of experience in teaching quantum computing to a wide variety of students. This results in a very readable book with just the right amount of information. Even Shor considers this book to be an ideal introduction to the field of quantum computing for computer scientists and mathematicians. The lecture notes of these lessons can also be found online [2].

Another true gem is the blog of Scott Aaronson [3]. The posts and comments are of a remarkable quality and should appeal to all degrees of students. One can find lecture notes, discussions on papers and simple explanations of hard things, all on this single blog. A true recommendation that is both easy to digest and humorous to read.

Other books on quantum computing are abound, of which a book that is published by Oxford University Press stands out [4]. This book is ideal to have a more precise introduction and helps to explain when the book by Mermin becomes a little too concise.

Another book is the book by Jozef Gruska [5]. Though the book is no longer in print, it serves as one of the first books that was able to bundle all the needed knowledge to get one started in the field of quantum computing into a single book. Special thanks goes out to Gruska himself, for lending me a PostScript version of the book when I inadvertently lost the only copy of his book that was available.

Finally, I would like to thank a number of people, without whom I would not have been able to write the paper as it is today. Special thanks go out to Ellie D'Hondt [7] for thoroughly reading the draft of this paper and giving numerous suggestions on both the writing style and contents of the paper. Special thanks also go out to Dirk Vermeir [8] for introducing me to this topic. I would like to thank Ann Dooms [9] for helping me to quickly recapitulate the math needed for this topic. Last but not least I would like to thank Line Caes, Wim Van Litsenborg, Pieter Coucke and Wouter De Geest for reading through this paper and suggesting numerous small improvements, that helped to increase the quality of this paper.

References

[1] N. David Mermin, *Quantum Computer Science – An Introduction*, 2007.

[2] N. David Mermin, *http://people.ccmr.cornell.edu/~mermin/qcomp/CS483.html*, as retrieved on 18 April 2008.

[3] Scott Aaronson, *http://www.scottaaronson.com/blog/*, as retrieved on 18 April 2008.

[4] Philip Kaye, Raymond Laflamme, Michelle Mosca, *An Introduction to Quantum Computing*, 2007.

[5] Jozef Gruska, *Quantum Computing*, 1999.

[6] Lov K. Grover, *A fast quantum mechanical algorithm for database searc*, 1996.

[7] Ellie D'Hondt, *http://www.vub.ac.be/CLEA/ellie/homepage/welcome.html*, as retrieved on 18 April 2008.

[8] Dirk Vermeir, *http://tinf2.vub.ac.be/~dvermeir/*, as retrieved on 18 April 2008.

[9] Ann Dooms, *http://homepages.vub.ac.be/~andooms/*, as retrieved on 18 April 2008.