An Introduction to Logical Operations on Classical and Quantum Bits [*]

F.L. Marquezino and R.R. Mello Júnior

CBPF - Centro Brasileiro de Pesquisas Físicas
CCP - Coordenação de Campos e Partículas
Av. Dr. Xavier Sigaud, 150
22.290-180 Rio de Janeiro (RJ) Brazil
(CNPq Fellows/PIBIC)
franklin@serraon.com.br, rui.rodrigues@inf.ucp.br

April, 2004

Abstract

This article is a short review on the concept of information. We show the strong relation between Information Theory and Physics, and the differences between classical and quantum information, with emphasis in their manipulation through logical gates.This paper is intended to be read by non-specialists and undergraduate students of Computer Science, Mathematics and Physics, with knowledge in Linear Algebra and Quantum Mechanics.

Keywords: Logical Gates, Quantum Computation, Computer Science.

1 Introduction

In a previous paper [1], we expose the physical character of information, making it clear that every information must be represented by a physical system. However, another question that may arise is how to interfere on the information being represented? This subject will be studied in this article, under the viewpoints of both Classical and Quantum Computation.

Sections 2 and 3 give an introduction to two different models of computation. The former is the traditional circuit model, compatible with the idea of Turing machines, but more adequate to the purposes of this paper.

---

arXiv:physics/0404134 v1 28 Apr 2004

The latter is the corresponding concept that uses properties of Quantum Mechanics, instead of Classical Mechanics.

As soon as we learn how to perform simple operations with a certain physical system, it is very important to learn how to keep these operations reliable. In Section 4, we briefly introduce classical and quantum error detection and correction.

Here, we assume that the reader already has a certain backgroung on Quantum Information. If necessary, we recommend the reading of [1] or [2] before starting reading Section 3.

2   On classical circuits

The Turing machine is an important concept, developed by the british mathematician Alan Turing [3]. Any calculation using a classical physical system, can be carried out by a Turing machine, according to the Church-Turing thesis [4]:

> "Any process that is effective or algorithmic in nature defines a mathematical function belonging to a specific well-defined class, known variously as the recursive, the λ-definable, or the Turing computable functions."

Here, we use a different, yet equivalent formalism, which is the circuit of gates and wires. The proof for the equivalence between Turing machines and circuits can be achieved through the notion of *uniform circuit family*. The reader may found this proof in many books on Computer Science or Quantum Computation, including [5].

We shall begin our study of the classical circuits by introducing the notion of Boolean functions. A function $f : \{0,1\}^n \rightarrow \{0,1\}^m$ is called a Boolean function, due to the english mathematician George Boole.

Interesting information on this subject can be found in [6], specially in the section on Combinational Logics. Another good reference is the book by Andrew Tanenbaum [7].

We can define a Boolean function $f(x)$ which is equal to 0 if $x = 1$ and is equal to 1 if $x = 0$. This function corresponds to the logical operation **NOT**. We can write the result of $f(x)$ as $\neg x$ or $\bar{x}$.

Another interesting function should be $f(x,y)$, which is equal to 1 if $x = y = 1$, and 0 otherwise. This is the logical gate **AND**. Similarly, we could define a function $f(x,y)$, equal to 0 if $x = y = 0$ and 1 otherwise, which corresponds to the logical gate **OR**. We can also write an AND operation as a product and an OR operation as a sum. For instance, $f(x,y) = xy$, meaning "x and y", or $f(x,y) = x + y$, meaning "x or y".

These functions can be represented by a table, called the truth table. See Tables 1, 2 and 3, for the functions NOT, AND and OR, respectively. We can build any circuit with these three logical operations.

x	$\neg x$
0	1
1	0

Table 1: Logical operation NOT.

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

Table 2: Logical operation AND.

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

Table 3: Logical operation OR.

At this point we should finally introduce the circuit of gates and wires. In this model each gate has a graphical representation. Wires carry 0's and 1's, through space or time. See Fig.1, for the functions NOT, OR and AND.

The output of a gate can be the input of another gate[1]. So, let $f(x)$ be a Boolean function corresponding to the NOT logical operation, and let $g(x, y)$ be a Boolean function corresponding to the AND logical gate. We can define a function $h = f \circ g = \neg(xy)$. This function is very important in Computer Science, and is called NAND. Graphically, it can be represented by putting a small circle in the outcome of the AND gate, meaning that this result is negated (see Fig. 2). Any classical logical circuit can be designed using only the NAND logical gate, and the FANOUT[2].

Another interesting logical gate is the NOR, defined as $f(x, y) = \neg(x+y)$, that is, an OR logical gate with its outcome inverted. As well as the NAND gate, this gate can also be used to build any logical circuit.

---

[1]Normally, the output of a gate is not a input of the same gate, although it may be possible in some special cases, such as flip-flop circuits. Flip-flops are special circuits that work as memories, keeping the value of a bit even after the input signalization.

[2]A "gate" that replicates the value of one bit in two different wires. It is often represented by a bifurcation in a wire.
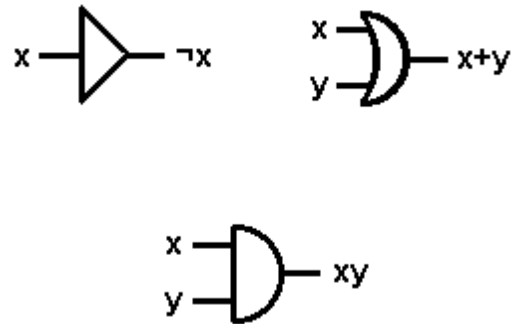
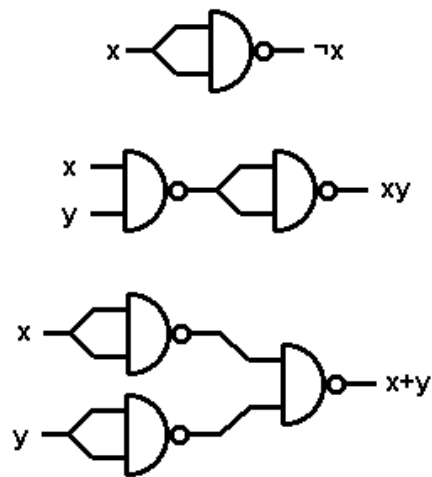Figure 1: Graphical representation for the logical operations NOT, OR and AND.

Figure 2: We can use the NAND gate to build the gates AND, OR and NOT. Consequently, we can build any classical circuit using only NAND gates.
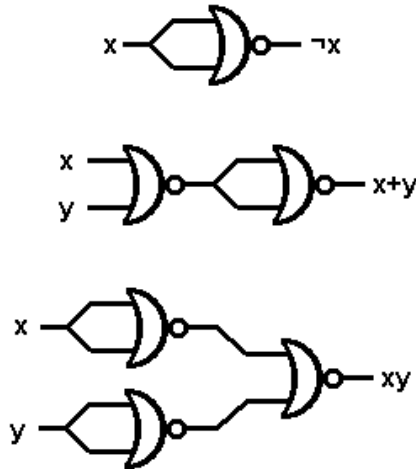
Figure 3: We can also use the NOR gate to build the gates AND, OR and NOT. Consequently, the gate NOR is also universal.

We cannot expect to create quantum circuits in a straightfoward way by simply using the same techniques of classical circuits. Firstly, because in classical circuits the universal gate is NAND, which is irreversible[3], while in quantum circuits all gates must be reversible. Not even FANOUT is possible in Quantum Mechanics, because of the non-cloning theorem [8].

3 On quantum circuits

Similarly to Classical Computation, there must be a mathematical representation for the operations performed in the quantum physical system. Computer scientists are used to represent logical operations by Boolean functions. However, in Quantum Computation we will use a slightly different formalism, which will be more suitable. The reader must be familiar with Linear Algebra, because that is what we will use from now on.

A quantum state is represented by a vector in a Hilbert space (a complex vector space with inner product). Hence, if we consider an orthonormal basis composed by the states $|0\rangle$ and $|1\rangle$, the quantum state can be any of the form $\alpha|0\rangle + \beta|1\rangle$, where α and β are complex numbers called amplitudes. Since this state must be a unit vector, we know that $\|\alpha\|^2 + \|\beta\|^2 = 1$.

We can represent our computational basis by the state vectors

$$|0\rangle \equiv \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ and } |1\rangle \equiv \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \tag{1}$$

---

[3]If we know only the output of an irreversible gate, we do not necessarily know the inputs.

for example. This representarion may help the understanding of the calculations carried out in this section.

Operations on a quantum state must preserve its norm, so they are described by unitary matrices. In order to represent a logical operation on a qubit, one should find a unitary matrix that gives the desired result when applied to the qubit.

$$U|\psi\rangle = |\psi'\rangle. \tag{2}$$

To illustrate that, let us observe the equivalent of the NOT classical logic gate. We must find a unitary operator U, such that

$$U|0\rangle = |1\rangle \tag{3}$$
$$U|1\rangle = |0\rangle, \tag{4}$$

and, since U is a linear operator,

$$U(\alpha|0\rangle + \beta|1\rangle) = \alpha U|0\rangle + \beta U|1\rangle. \tag{5}$$

The operator

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \tag{6}$$

one of the three Pauli matrices, corresponds to the unitary matrix that we are looking for.

Note that if we apply the operator σ_x twice, we keep the original value of the qubit.

$$\sigma_x\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{7}$$

which is the identity matrix.

Another interesting operator is the Hadamard gate:

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}. \tag{8}$$

This gate is able to create superpositions in a qubit. Note that $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

Another interesting type of operator that we cannot forget mentioning is the controlled operator. When we have at least two qubits (a quantum register), we can define an operator that will interfere in a particular qubit if and only if some requirement is observed. For instance, let $\sigma_x^{c,t}$ be an operator that applies the NOT logic operation in the qubit t if and only if the qubit c is $|1\rangle$.

6

$$\sigma_x^{c,t}(|0\rangle_c|0\rangle_t) = |0\rangle_c|0\rangle_t \tag{9}$$

$$\sigma_x^{c,t}(|0\rangle_c|1\rangle_t) = |0\rangle_c|1\rangle_t \tag{10}$$

$$\sigma_x^{c,t}(|1\rangle_c|0\rangle_t) = |1\rangle_c|1\rangle_t \tag{11}$$

$$\sigma_x^{c,t}(|1\rangle_c|1\rangle_t) = |1\rangle_c|0\rangle_t. \tag{12}$$

In this case, we say that qubit t is the "target qubit" and qubit c is the "control qubit". This gate is called CNOT and, similarly, it may be expressed as a unitary matrix.

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{13}$$

Similarly to Classical Computation, there are gates in Quantum Computation that can be used to build any possible circuit. It can be proven that an arbitrary unitary matrix U can be decomposed into a product of two-level unitary matrices, i.e., matrices that act non-trivially on **no more than** two vector componets. One could also prove that single qubit gates and CNOTs can be used to achieve any two-level unitary matrix. Hence, it can be proven that single qubit gates and CNOT gates can implement an arbitrary operation and, therefore, they are universal in Quantum Computation.

The proof for the last paragraph is not straightfoward and would spend a reasonable space in this paper. If the reader is interested in studying this subject deeper, the main reference is [9].

4   Error detection and correction: some brief remarks

An important area of Computer Science concerns the detection and correction of errors. If we intend to build a computer, no matter if a classical or a quantum-mechanical one, we have to deal with its errors, so that the calculations can be trusted.

In a classical digital computer, we can consider one single kind of error. Suppose we wish to send a bit through a channel. If there is noise in this channel, a bit 0 can be changed to 1, with probability p, and a bit 1 can be changed to 0, with probability q. So, the bit 0 will keep correct with probability $1-p$, and the bit 1 with probability $1-q$. Normally, we consider $p = q$. In this case, both bits 0 and 1 have the same probability p of being affected by the noise, and the same probability $1-p$ of being sent correctly. This channel is called a *binary symmetric channel*.

Classical schemes for error detection and correction use some kind of redundancy. A very simplistic example would be repeating the same bit n times. For instance

$$0_L = 000 \tag{14}$$

$$1_L = 111 \tag{15}$$

with $n = 3$.

In this case, if we wish to represent the number 5 (101 in the binary system), we must repeat each logical bit three times:

$$(101)_L = 111000111. \tag{16}$$

If some noise inverts a single bit in a triplet, we can easily repair the information. For instance, if the above number is changed to 111010110, we know that the triplets 010 and 110 are wrong. We can recover the information by performing the *majority vote*. So, the triplet 010 was originally 000 and the triplet 110 was 111. One could easily prove that the majority vote works properly whenever $p < \frac{1}{2}$.

In Quantum Mechanics, errors are not so simple. Firstly, we cannot simply copy the quantum bits, because of the non-cloning theorem. Moreover, even if we could copy qubits, we would not be able to detect errors by simply measuring them. These limitations are enough to show us that our techniques must me changed if we wish to handle the quantum-mechanical world. But, if we wish to go a little further, we must consider that different kinds of errors may occur in this case.

A qubit can suffer a phase flip, when $\alpha|0\rangle + \beta|1\rangle$ is changed to $\alpha|0\rangle - \beta|1\rangle$. Besides, we must remember that a qubit is continuous, so it can suffer a small error. Imagine a Bloch sphere where the state is accidentally rotated by a small angle. This error would not be detected if we consider only bit flips and phase flips. How can we protect our quantum computers against all these threats? Peter Shor, who created the code to correct arbitrary errors on a single qubit, answered this question [10]. Nowadays, this code is known as the *Shor code*. In order to understand it, we will analyze two "modules": the three-qubit bit flip code and the three-qubit phase flip code. The Shor code is a combination of these two "modules".

Before the explanation of these modules, it should be important to have a brief digression on the third postulate of Quantum Mechanics, which concerns to measurements. A measurement is characterized by an observable, M, such that $M = M^\dagger = (M^T)^\star$. So, the spectral decomposition of M is:

$$M = \sum_m P_m. \tag{17}$$

According to this postulate, the possible outcomes that may occur in an experiment are the eigenvalues of M, i.e., m. If the state before the measurement is $|\psi_{bef}\rangle$, then the probability that a result m occurs is

$$p(m) = \langle\psi_{bef}|P_m|\psi_{bef}\rangle \tag{18}$$

and the state after the measurement is

$$|\psi_a ft\rangle = \frac{M_m|\psi\rangle}{\sqrt{\langle\psi_{bef}|P_m|\psi_{bef}}}. \tag{19}$$

Now, we shall return to the study of the Shor code.

4.1 The three-qubit bit flip code

At first, we must encode our qubits, as follows:

$$|0\rangle \text{ becomes } |0_L\rangle \equiv |000\rangle \tag{20}$$
$$|1\rangle \text{ becomes } |1_L\rangle \equiv |111\rangle, \tag{21}$$

so that the qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ is encoded as $\alpha|000\rangle + \beta|111\rangle$.

If we wish to check if some error occurred, we use a measurement called *syndrome diagnosis*. Its result is called the *error syndrome*. We use four projection operators, each one corresponding to a bit flip in a specific qubit.

$$P_0 \equiv |000\rangle\langle000| + |111\rangle\langle111| \quad \text{no error occurred} \tag{22}$$
$$P_1 \equiv |100\rangle\langle100| + |011\rangle\langle011| \quad \text{error on 1st qubit} \tag{23}$$
$$P_2 \equiv |010\rangle\langle000| + |101\rangle\langle101| \quad \text{error on 2nd qubit} \tag{24}$$
$$P_3 \equiv |001\rangle\langle000| + |110\rangle\langle110| \quad \text{error on 3rd qubit}. \tag{25}$$

Note that, when we measure an state $|\psi\rangle$ using the operator P_i, ($0 < i \leq 3$), we obtain 1 if the i-th qubit is corrupted, and 0 otherwise. Similarly, if we use the operator P_0, we obtain 1 if the three qubits are correct, and 0 otherwise. After the measurement, the qubit is not modified! This is possible because the syndrome diagnosis does not give us any information about the qubit *per si* (i.e., the values of α and β).

4.2 The three-qubit phase flip code

The three-qubit bit flip code protects our qubit against the simplest kind of error. However, as we mentioned before, quantum bits are subject to much more errors than simple flips. The second type of noise that we will analyze is the phase flip. This error may change a qubit $\alpha|0\rangle + \beta|1\rangle$ into $\alpha|0\rangle - \beta|1\rangle$.

The idea behind the three-qubit phase code is very similar to that used in the three-qubit bit flip code. Instead of encoding the qubit according to equations 20 and 21, we must use

$$|0\rangle \text{ becomes } |0_L\rangle \equiv |+++\rangle \tag{26}$$

$$|1\rangle \text{ becomes } |1_L\rangle \equiv |---\rangle, \tag{27}$$

where $|+\rangle \equiv (|0\rangle + |1\rangle)/\sqrt{2}$ and $|-\rangle \equiv (|0\rangle - |1\rangle)/\sqrt{2}$.

The only difference is that, in this code, we use the basis $\{|+\rangle, |-\rangle\}$ to encode the qubit, instead of the basis$\{|0\rangle, |1\rangle\}$. Remember that we can use the Hadamard operator (eq. 8) to change between one basis and the other.

To perform the syndrome measurement, we only need to use the correct operators for the new basis.

$$P_i \text{ becomes } P_i' \equiv (H \otimes H \otimes H)P_i(H \otimes H \otimes H). \tag{28}$$

If the reader is interested in demonstrations for the subject exposed so far in this section, we recommend the references [5] and [11].

4.2.1 The Shor code

The Shor code is a *concatenation* of the two codes explained before. In the first step, we encode the qubit using the phase flip code (eq. 26 and eq. 27). Consequently, we will have three qubits[4] from this point on. In the second step, we encode each of the three resulting qubits using the bit flip code (eq. 20 and eq. 21). Finally, we have nine qubits representing the initial qubit.

$$|0\rangle \text{ becomes } |0_L\rangle \equiv \frac{(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)}{2\sqrt{2}} \tag{29}$$

$$|1\rangle \text{ becomes } |1_L\rangle \equiv \frac{(|000\rangle - |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle)}{2\sqrt{2}}. \tag{30}$$

As we can see, this code is very expensive, especially if we consider the present status of experimental quantum computation. The best quantum computer ever built had only seven qubits! However, the Shor code is extremely important because it protects the qubit against arbitrary errors, not only bit flips or phase flips. The main lesson of this section is that, although quantum states can present a *continuum* of errors, they can be repaired by detecting and correcting only a discrete subset of errors.

---

[4]We used three qubits in our example, but we could have used more. The same comment is valid for the bit flip code. Three is the minimum necessary in both cases.

5   Conclusions

This paper is an introduction to one of the most important topics in Computer Science. Whenever we wish to perform a computation, we must know how to store a certain initial value, and then manipulate it until we achieve a result, when we can finally measure the system and obtain an answer.

We began by showing how we can perform simple operations in physical system, with classical and quantum logic gates. Then, we showed how to keep it reliable, with error detection and correction codes. Naturally, this is a vast area of knowledge, which could not be completely described in one single article. However, our goal here is to introduce this subject in a simple but effective way, helping to students and researchers from different areas of knowledge.

The references mentioned throughout this paper can be used to improve the comprehension of the subject.

Acknowledgements

References

[1] Marquezino, F.L. and Mello Júnior, R.R., "Considerations on Classical and Quantum Bits". *Work in progress.*

[2] Lavor, C., Manssur, L.R.U and Portugal, R., "Grover's Algorithm: Quantum Database Search". (2003) Available in `www.arxiv.org/quant-ph/0301079`.

[3] Turing, A.M., "On computable numbers, with an application to the Entscheidungsproblem", Proc.Lond.Math.Soc. 2, 42:230. (1936)

[4] Shapiro, S., "Church's Thesis". Encyclopedia of Artificial Inteligence, John Wiley & Sons, New York. (1990) pp. 99-100.

[5] Nielsen, M.A. and Chuang, I.L., "Quantum Computation e Quantum Information", Cambridge University Press (2000).

[6] WWW page at University of Maryland: `http://www.cs.umd.edu/class/spring2003/cmsc311/Notes/`.

[7] Tanenbaum, A.S., "Structured Computer Organization". 4th edition, Upper Saddle River, NJ: Prentice Hall. (1999)

[8] Wooters, W.K. and Zurek, W.H., "A single quantum cannot be cloned", Nature, **2**99, 802-803. (1982)

[9] Barenco, A., Bennett, C.H., Cleve, R., DiVicenzo, D.P., Margolus, N., Shor, P., Sleator, T., Smolin, J. and Weinfurter, H., "Elementary gates for quantum computation", Phys.Rev.A, 52:3457-3467. (1995). Also available in `www.arxiv.org/quant-ph/9503016`.

[10] Shor, P.W., "Fault-tolerant quantum computation". In *Proceedings, 37th Annual Symposium on Foundations of Computer Science*, pages 56-65, IEEE Press, Los Alamitos, CA. (1996)

[11] Preskill, J., "Lecture Notes for Physics 229: Quantum Information and Computation". California Institute of Technology. (1998)

[12] Williams, C.P. and Clearwater, S.H., "Explorations in Quantum Computing", The Electronic Library of Science, California. (1997)