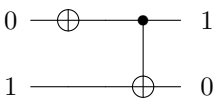
Preliminaria

Artur Ekert and Alastair Kay

I. BITS AND GATES

From a mathematical perspective, computation is an operation on abstract symbols. Any finite set of symbols is called an alphabet and any finite sequence of symbols from that alphabet is called a string. Here, without any loss of generality, we will use the binary alphabet $\{0,1\}$ and we shall denote the set of all 2^n possible binary strings of length n as $\{0,1\}^n$. For example, $\{0,1\}^2$ contains the strings $00, 01, 10$ and 11. Operations on bits are often represented as network (or circuit) diagrams. For example, the network



operates on two bits. It should be read from left to right. The horizontal line represents a wire, which inertly carries a bit from one operation to another. Various icons placed on the wires represent elementary logical operations known as logic gates. Here, the first bit (counting from the top) is negated with logical NOT, represented as $\oplus$, and then a controlled-NOT is applied to the first and the second bit. The controlled-NOT flips the second (target) bit if the first (control) bit is 1 and does nothing if the control bit is 0.

It is well known that some elementary operations on bits, for example logical NOT and logical AND, are complete, that is, any Boolean function $\{0,1\}^n \mapsto \{0,1\}$ can be expressed in terms of only these two operations. Some of these essential operations on bits, such as bit addition XOR ($\oplus$) and bit multiplication AND ($\times$), defined as

$$0 \oplus 0 = 0 \;\; 0 \oplus 1 = 1 \;\; 1 \oplus 0 = 1 \;\; 1 \oplus 1 = 0,$$
$$0 \times 0 = 0 \;\; 0 \times 1 = 0 \;\; 1 \times 0 = 0 \;\; 1 \times 1 = 1,$$

are irreversible – given the output, we usually cannot reconstruct the input. However, it is possible to implement these gates, and indeed any function $\{0,1\}^n \mapsto \{0,1\}^m$, using only reversible operations. This is important for us because we will study the physical foundations of computation and the basic laws of physics are reversible in time. This reversibility leads, in a natural way, to reversible logic gates and reversible computation.

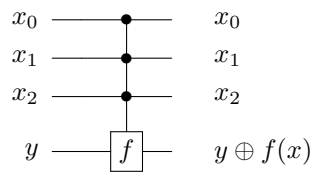
A reversible computer evaluates Boolean functions,

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

by embeding them into reversible computation

$$F : (x, y) \mapsto (x, y \oplus f(x)),$$

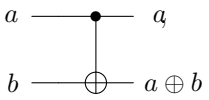
where $x \in \{0,1\}^n$, $y \in \{0,1\}$. The corresponding network diagram shows such a function evaluation for $n = 3$,



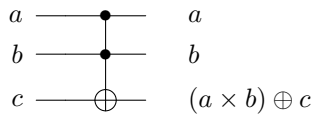
In particular for $y = 0$ we obtain $F(x, 0) = (x, f(x))$. This method can be easily generalised to include any function $\{0,1\}^n \mapsto \{0,1\}^m$. We simply embed such a function evaluation into an invertible operation on at least $n + m$ bits. Invertible functions on n bits are effectively permutations of 2^n binary strings of length n.

All elementary steps of any reversible computation permutation must be implemented in a reversible way. This is not a problem because all permutations of binary strings can be constructed using a set of simple reversible gates, such as NOT, controlled-NOT (C-NOT) and a controlled-controlled-NOT (C^2-NOT).

The controlled-NOT is a reversible gate which operates on two bits and maps $(a, b) \mapsto (a, a \oplus b)$. It flips the second (target) bit if the first (control) bit is 1 and does nothing if the control bit is 0.



The c^2-NOT gate, or the Toffoli gate, is a reversible gate which operates on three bits and maps $(a, b, c) \mapsto (a, b, (a \times b) \oplus c)$. This gate has two control bits a and b and one target bit c, and it flips the target bit if both of the the control bits are 1, and does nothing otherwise.

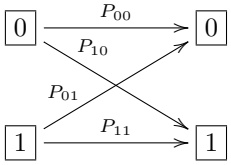


The Toffoli gate is very powerful. If we have a source of 0s and 1s and can choose to ignore some inputs or outputs then we can obtain: negation of c (set $a = b = 1$, a C-NOT gate with b as the control and c as the target (set $a = 1$) or just $b \oplus c$ if we ignore b at the output, and finally an AND gate with inputs a and b (set $c = 0$ and ignore a and b at the output). The Toffoli gate gives us all we need to construct evaluation of any Boolean function.

II. STOCHASTIC OPERATIONS

If there is any inherent randomness affecting the elementary logic operations then any fundamental descrip-

tion of computation must take it into account. Let us extend the repertoire of our gates by including stochastic operations. For example, the most general computation on one bit may be represented by the diagram,



This machine has the property that if we input the value k ($k = 0$ or 1) and then read the output, we obtain the value l ($l = 0$ or 1) with probability P_{lk}. More generally, if we input k with probability p_k, then we obtain the output l with probability $p'_l = \sum_k P_{lk} p_k$. It is convenient to tabulate the transition probabilities and express the action of the machine in a matrix form,

$$\begin{pmatrix} p'_0 \\ p'_1 \end{pmatrix} = \begin{pmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \end{pmatrix}.$$

$$\text{OUTPUT} \leftarrow \text{EVOLUTION} \leftarrow \text{INPUT}$$

The state of a physical bit, e.g. the input or output, is described by a probability vector and its evolution by a transition matrix. The transition matrix has non-negative elements P_{lk} satisfying the standard probability conditions $\sum_l P_{lk} = 1$, i.e. entries in each column add up to one, which means that each column can be viewed as a probability vector. Such matrices are called stochastic. For example, here are five stochastic matrices; the first four describe all possible deterministic limits of a one bit computation,

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix} \quad \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

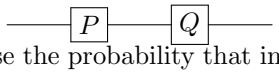
IDENTITY NOT CONST 0 CONST 1 RAND

and the fifth one describes a completely random switch. The random switch can act as a source of random bits. Algorithms that use random bits are called randomised. Despite the fact that they may generate erroneous answers they are very useful; random sampling helps to solve certain problems more efficiently.

III. COMPOSITIONS

Given two independent computations described by stochastic matrices P and Q, we can arrange them in a sequence or in parallel. The resulting computations are described by new stochastic matrices which are denoted as QP and $P \otimes Q$ respectively.

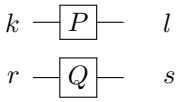
If P is followed by Q then the net result is computation described by the matrix product QP.



This is because the probability that input k evolves to output m via a specific intermediate bit value l is given by $Q_{ml} P_{lk}$ (machines are independent so the probabilities are multiplied) and we need to sum over all possible values of l (the transition can occur in several mutually exclusive ways so the probabilities are added) to obtain $\sum_l Q_{mk} P_{lk}$. For example, if P represents the RANDOM SWITCH and Q the logical NOT then the network effects the operation

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

When we bring n and m bits together we form a new system composed of $n + m$ bits. Let P acts on the first n bits and Q on the remaining m bits,



A combined input (k, r) evolves into a combined output (l, s) with the probability $P_{lk} Q_{sr}$. This is because the actions of the two machines are independent, and thus the probabilities of the two transitions multiply. If P and Q are performed in parallel then the net result is computation described by the tensor product $P \otimes Q$. The matrix elements of $P \otimes Q$ are $(P \otimes Q)_{ls,kr} = P_{lk} Q_{sr}$.

For example, when we bring together two bits, we obtain a composed system with four configuration which can be labelled as $(0, 0)$, $(0, 1)$, $(1, 0)$ and $(1, 1)$. The composite labels such as (a, b) are often written simply as strings ab, and it should be clear from the context that it is a concatenation of two symbols and not a product of two numbers. In the two bit case we write the composite labels as binary strings $00, 01, 10$ and 11.

Again, if P represents the RANDOM SWITCH and Q the logical NOT then the network above effects the operation

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} = \begin{pmatrix} 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \end{pmatrix}$$

The rows and columns in the resulting 4×4 stochastic matrix represents are labelled by the binary strings $00, 01, 10$ and 11. Some operations on two bits can be written as a tensor product of two single-bit operations and some cannot. For example, controlled-NOT does not admit such a decomposition.