1
Making Computation Faster and Communication Secure: Quantum Solution

The main stumbling block of quantum information, computation, and to a lesser extent, communication is the lack of a definite hardware. We still do not know whether we are going to compute by ions, or by solid state systems, or by photons, or by quantum electrodynamics, or by superconducting charges . . . Yet, there are already formalisms, algorithms and theories on quantum "all that." But, didn't we have the same "problem" when we started to compute on classical computers in the forties? What was the hardware then? "Human computers," mechanical gadgets, electromechanical drums, tube-calculators, . . . And before that, we already had classical formalisms and algorithms and theories. Let us start with a classical story which will help us understand that interplay of software and hardware so that we can better apply it to qubits later on.

1.1
Turing Machine: a Real Machine or . . .

The *Turing machine* is not a computer and it cannot serve us to build a useful gadget. Yet, there are so many Turing machine applets on the web to help math students to prepare their exams. So, why can't we turn "the machine" into a realistic computing device? The answer is both simple and long.

Alan Turing graduated in mathematics from King's College, Cambridge in 1934 and was elected a fellow there the next year thanks to a paper in which he designed his famous machine. The paper gave a solution to a problem on which the famous mathematician Alonzo Church at Princeton University was also working at the time. So, in 1938, Turing went to Princeton to study under Church and received his Ph.D. from Princeton in 1938.

A few months later, Turing returned to England and started to work part-time at the *Government Code and Cypher School* on German encryption systems. A year later, he joined the wartime station of the school, now famous, *Bletchley Park*. There he soon became a main designer of electromechanical decrypting machines – named *Bombe*s, after their predecessor, a Polish *Bomba*. They helped the British to decipher many German messages and gain advantages in many actions and battles.

Companion to Quantum Computation and Communication, First Edition. M. Pavičić.

The story is a paradigm for today's university researchers who are expected to find an application for their research as soon as possible and sign as many contracts with industry as possible. Then, there were many details in Turing's approach to work and people that attract interest of the media today – also a valuable commodity for today's university researchers. For example, he induced his colleagues to see his design of the *Bombe* as follows: "[in its design] he had the idea that you could use, in effect, a theorem in logic which sounds to the untrained ear rather absurd; namely, that from a contradiction, you can deduce everything." This is our main clue and we will come back to it in Section 1.2.

In July 1942, Turing devised a new deciphering technique named *Turingery* against a new German *secret writer*, code-named *Fish* and recommended some of his coworkers for the project of building the *Colossus* computer, the world's first programmable digital electronic computer, which eventually replaced simpler prior machines and whose superior speed allowed the brute-force decryption techniques to be usefully applied to the daily-changing ciphers. Turing himself did not take part in designing the Colossus, but left for America to work on US *Bombes* ($3 \times 2.1 \times 0.61\,\text{m}^3$; 2.5 t; 120 of them were made till 1944). When he returned to England, he accepted a position as a general consultant for cryptanalysis at the Bletchley Park. At that time, he also designed a machine for a secure voice communication which has never been put into production.

Details of his work on cryptography during the war remained a secret for many years after it. Eventually, he dropped his cooperation with industry and returned to that lofty realm of science that offers a different history. But, let us first go back to his machine and examine its "simple history." We will learn that the machine is not at all a real machine, but a mathematical procedure.

1.2
… a Mathematical Procedure

In the thirties, most leading mathematicians in the field of symbolic logic and related algebras were involved in solving a problem of *decidability* – whether one can decide that a statement (formula, theorem) in a formal system is valid (holds) or not. That a system is *decidable* means that each formula in it is either provable or refutable. That a proof of a formula (*predicate* of the formula) is *effectively decidable* means that for every tree of formulae starting from the axioms of the system we can tell whether it is a proof of the formula, that is, whether it is *recursive*. For *functions* – formulae that depend on arguments – we then say that they are effectively calculable functions. If there is a system of equations that define a function recursively, then the function is *general recursive*.

One of the leading mathematicians who was engaged in these problems and who defined the notion of the general recursiveness of a function was Alonso Church (see above) who also formulated his famous *Church thesis*:

Thesis 1 Church 1936

Every effectively calculable function (effectively decidable predicate) is generally recursive.

An interpretation of the thesis is the following one. If we know a recursive procedure for obtaining a function, then, of course, the function is effectively calculable because the procedure itself is the proof that the function is valid. The converse is not obvious. That is, if we know how to decide whether there is a proof that a function is valid, we need not ever be able to find a recursive procedure for obtaining the function or even need not know whether such a procedure exists at all. The Church thesis is a conjecture that it always exists. It has not been proved so far, but there are overwhelming evidences that it is correct and, of course, it has never been disproved.

So, how do we prove that a system is decidable and that all its functions are *computable*, that is, generally recursive? Well, we have to find a procedure which would prove that every function from the system is effectively calculable or every predicate effectively decidable, and then we search for a generally recursive algorithm[1] for computing the functions. Or, better still, we first find a generally recursive algorithm and constructively prove that all functions from a systems are computable. The effective decidability and calculability then follow as consequences of the computability of the system.

The latter task is exactly what Church's general recursiveness (1933), Kleene's *λ-definability* (1935), Gödel's *reckonability* (1936), Turing's machine (1936/1937) and Post's *canonical* and *normal systems* (Emil Post; 1936; independent discovery) are about and this is why theoreticians like them so much. They, however, prefer the Turing machine over others because it is more intuitive and easier to handle.

The final "output" of any of these procedures is the same. They tell us which theory is decidable and which is not. Then, Church's thesis tells us to assume that any decidable theory is computable and that any undecidable one is not.

- Decidable theories are, for example:
  - *Presburger arithmetic* of the integers with equality and addition (Mojżesz Presburger, 1929);
  - Boolean algebras (Alfred Tarski, 1949);
  - Propositional two-valued classical logic;
- Undecidable theories are, among so many others:
  - *Peano arithmetic* with equality, addition, and multiplication (Kurt Gödel, 1932);
  - Predicate logic including metalogic of propositional calculus;
  - Every consistent formal system that contains a certain amount of finitary number theory there exists undecidable arithmetic propositions and the consistency of any such system cannot be proved in the system (many authors, including A. Turing, from the early thirties until the mid-sixties);

1) Algorithm is a computational method of getting a solution to a given problem in the sense of getting correct outputs from given inputs.

– Functions obeying *Rice's theorem*: Only trivial properties of programs are algorithmically decidable. For any nontrivial property of partial functions, the question of whether a given algorithm computes a partial function with this property is undecidable (H.G. Rice, 1953).

Thus, any of the above procedures, can be used to prove that a 0-1 Boolean algebra or equivalently, a two-valued (*true*, $\top$; *false*, $\perp$) propositional classical logic is decidable and therefore computable. Peano arithmetic and any more complicated systems using real numbers are not. Hence, our standard digital binary universal computer is actually the only one we can build without running into a contradiction sooner or later. This gives insight into Turing's colleagues' remark we cited above: "[Turing] had the idea that [in dealing with the Bombe computer] you could use, in effect, a theorem [which states that] from a contradiction, you can deduce everything." By invoking this well-known *Ex contradictione quodlibet* logical principle, they referred to Turing's checking whether particular code systems were consistent or not.

1.3 Faster Super-Turing Computation

Thus, a Boolean digital binary system is a "safe" ideal algebra for building a universal computer because it is decidable and consistent. But, does that mean that undecidable and inconsistent systems reviewed in the previous section cannot be used for computing and building computers?

When Frege, Whitehead, Russell, and Hilbert attempted to develop logic foundations of mathematics, they stumbled on paradoxes of self-reference such as the famous *Liar paradox*, on inconsistencies. Their attempts to go around such inconsistencies failed, but in the eighties, theoreticians revised such apparently inconsistent theories and saw a possibility to revive the Hilbert Program of consistently building mathematics from its logical foundations.

At the time, David Hilbert gave up his program because Gödel and others proved that the consistency of arithmetics cannot be proved within arithmetics itself. Though recently, the authors, such as R.K. Meyer and C. Mortensen, started from a widely accepted assumption that all negative results would not endanger the correctness of numerical calculations that have been carried before and since the beginning of the twentieth century, and started a new program called *Inconsistent Mathematics*. Originally, it started with a plausible claim that mathematics could be given a trouble-free interpretation if we recognized that mathematics is *not* its foundation.

We shall not elaborate on the foundational and conditional aspect of "inconsistent mathematics" any further. However, we need to discuss several nonstandard approaches in computation science and underlying formalisms, algebras, and even logic to see how it all can be applied in reaching our goal of speeding up computation – both classical and quantum.

Hilbert's program considered whether we can write down algorithms for an automated computation of any expression or carrying out any proof in any mathematical theory. Such algorithms can traditionally and rigorously be obtained only for Boolean algebras. The Church–Kleene–Gödel–Turing–Post proof of this result we can – in *2012 Year of Alan Turing* – express as follows. "A Turing machine calculating any of the 0-1 Boolean algebra problems will halt after a finite number of steps." But, what about standard arithmetics? Theory of rational or real numbers? Can there be *super-Turing* machines that are faster then the Turing ones? We shall see that there are *quantum-Turing* machines that are exponentially faster than the Turing ones and therefore a kind of super-Turing machine. Are there *classical* super-Turing machines?

To answer this question, we first have to answer several other questions.

1. Let us start with our safe "digital 0-1 algebra." Is there a logic behind it which is more general than the usual two-valued (true-false) one? Can it be implemented in a binary computer? Is it important for our purpose of devising a fast quantum computer to find a "quantum logic" behind or "under" the Hilbert space formalism we will use?
2. Can we devise computers that can handle, for example, real numbers directly, analogously to how we humans handle them on paper, that is, without any need to first digitalize them? Can analog computers be universal? Are they faster? What are their limits? Do quantum computers have a theoretical speed limit?
3. Are there other such classical or optical computers that can compute the same problems quantum would-be computers could solve? Can we achieve similar exponential speed-up of computation with such computers? Can we realistically use them to carry out super-Turing computation?
4. How much energy does the computation itself require? Can we reduce heat and energy dissipated in calculation per operation and per calculated bit? Heat is a main problem when we want to pack transistors of ever reduced size. The closer the transistors are to each other, the faster the computation. Are there processors that dissipate orders of magnitude less heat than today's standard Pentiums? What is the theoretical minimum we cannot go beyond? How do classical computers that dissipate a minimum of energy look like? Are quantum computers better?

We shall answer all of these questions in the next sections.

1.4 Digital Computers Do Not Run on Logic

It is often taken for granted that 0-1 Boolean algebra, Boolean logic, and classical propositional logic are all different names for one and the same algebra: 0-1 Boolean algebra. However, that is not the case.

If we browse through books on computation and computer organization, we shall soon notice that these books hardly ever mention *logic*. This is because the theory of classical logic contains various methods of manipulating the propositions and different possible models (semantics). The authors know that almost universally accepted valuation of the logical propositions is a 0-1 bivaluation, that the corresponding semantics is represented by truth tables, and that the only lattice model that corresponds to this bivaluation is a 0-1 Boolean algebra – a Boolean algebra is a distributive lattice. So, they all take for granted that it is OK to deal with 0-1 Bolean algebra instead. Often, Boolean algebra is called *Boolean logic*. Let us take a more detailed look.

Algebra is a mathematical structure (most often a *vector space*, for example, a lattice, a Boolean algebra, a Euclidean space, a phase space, a Hilbert space, ...) over a set of elements (most often a *field*, for example, real or complex numbers, ...). Loosely speaking, algebras describe relationships between things that might vary over time. What interests us the most are algebras that can or cannot be implemented in a computer and algebras that can serve as models of logic.

Thus, although a general Boolean algebra is a vector space over a field or a ring (e.g., set $\{0,1\}$, for which division is not defined), we shall start with its simplest 0-1 (digital, two-valued) form and define it at first as the set $\{0,1\}$ on which operations *conjunction* ($\cap$), *disjunction* ($\cup$), and *complement* ($'$) are defined as in Figure 1.1.

Operations in logics are defined equivalently, only it is taken that 1 means *true* and 0 *false*. These values are called *truth values* and are denoted as $\top$ and $\perp$, respectively. The tables from Figure 1.1 are called *truth tables*.

What is characteristic of both $\{0,1\}$ Boolean algebra rules and classical logic truth tables is that by starting from the definite initial values for all variables, we will define values of all intermediary combinations of the values until we reach a final result of our calculation as shown in Table 1.1. When the expressions become huge, evaluation of the intermediary expressions take exponentially more time. And, these intermediary expressions are exactly what quantum computers should get rid of. How?

Both classical and quantum computers require the so-called *logic gates*, which we can understand as switches or ports through which electrons, photons, ..., information flow. Schematics of some classical ones are shown in Figure 1.2 where, for example, XOR is the electrical current equivalent of the negation of the logical biconditional "$\leftrightarrow$:" It represents the following electric current behavior in a tran-

x	$'$
0	1
1	0

x'

x \ y	$\cap$ 0	1
0	0	0
1	0	1

$x \cap y$

x \ y	$\cup$ 0	1
0	0	1
1	1	1

$x \cup y$

x \ y	$\rightarrow$ 0	1
0	1	1
1	0	1

$x \rightarrow y$

x \ y	$\leftrightarrow$ 0	1
0	1	0
1	0	1

$x \leftrightarrow y$

Figure 1.1 Boolean and logical binary operations. Boolean operations $'$, $\cap$, $\cup$, $\rightarrow$ and $\leftrightarrow$ we denote in logic as $^{-}$, $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$.

Table 1.1 Logical truth table. The more complicated the expression, the more intermediary valuations we have to make to evaluate the final expression. The complexity of its evaluation grows exponentially in time with its size.[2)]

a	b	c	$a \wedge b$	$\bar{a}$	$\bar{c}$	$\bar{a} \wedge \bar{c}$	$(a \wedge b) \vee (\bar{a} \wedge \bar{c})$	$b \equiv c$	$(\bar{a} \wedge \bar{c}) \supset (b \equiv c)$
$\top$	$\top$	$\top$	$\top$	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\top$
$\bot$	$\top$	$\top$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$\top$	$\top$
$\top$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$
$\bot$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$
$\top$	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\top$	$\bot$	$\bot$
$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\bot$	$\bot$
$\top$	$\bot$	$\bot$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$

sistor: "The output is low when both inputs A and B are high and when neither A nor B is high."

That low and high voltage, 0 and 1, a binary digit, a unit of information, a *bit* for short, is what makes up every number, word, program, pixel, sound, image in a classical computer. When we want to represent a number in a binary form, we soon realize how many physical hardware elements we need to implement any input. For example, eight-bit binary forms of the first 256 nonnegative integers are 00000000, 0000001, ..., 11111111. To carry out the addition of these digits (other operations can be reduced to addition), a classical digital computer uses an eight bit binary adder. It consists of eight full adders, each full adder consists of two half adders and an OR gate, and each half adder of an XNOR (negation of XOR) and an AND gate which altogether makes 40 gates (see Figure 1.71).

Thus, a computation of problems or manipulation of images whose complexities grow exponentially require an exponential increase of the number of transistors. Today, the number of transistors (gates) in a classical processor already reached 5 billions and still a quantum processor with only several hundred quantum gates would outdo it. The reason is that quantum gates can work with an arbitrary continuous combination, we call it a *superposition*, of elementary states – *quantum*

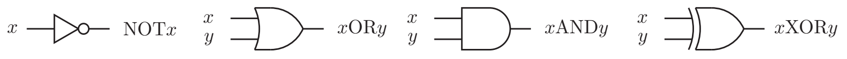

Figure 1.2 Logic gate symbols and operations. Notation used for operations: $\overline{x}$ (NOT x), $x + y$ (x OR y), xy (x AND y), $x \oplus y$ (x XOR y) (XOR is addition ($+$) modulo 2: $1 \oplus 1 = 0$; also $A \oplus B = (A + B)\overline{AB} = A\overline{B} + \overline{A}B$). See Figure 1.16.

2) To that, we can add the *satiability problem* (SAT) and the *isomorph-free generation of graphs and hypergraphs*. SAT problem consists in verifying whether Boolean expressions like that one shown in Table 1.1 are satisfied, that is, true, that is, equal to 1 in a Boolean algebra. SAT belongs to EXPTIME and that will help us understand why the factoring number problem computed in a digital computer belongs to EXPTIME too. Graphs and hypergraphs can be used to map nonlinear equations into hypergraphs and filter out equations that have solutions. They also belong to EXPTIME. We shall use them later on to generate the so-called Kochen–Specker sets.

bits, qubits. Yet, a working quantum computer still does not exist and therefore we should first explore whether we can exponentially speed-up computation in some other way.

The first option is to see whether we can implement our classical logic into some other kind of hardware instead of a binary computer. Since the models of both classical and quantum logic that we use to implement logics into computers are lattices (a distributive lattice (Boolean algebra) and a Hilbert lattice (underlying the Hilbert space)), the option boils down to a question of whether there are other lattices that can model classical logic. The answer is positive.

Let us look at the lattice shown in Figure 1.3. (A *lattice* is a partially ordered set with unique least upper and greatest lower bounds.) Here, a valuation of the proposition: "A particle is detected at position (4,3,5)" can be not only 1 (true) and 0 (false), but also a, b, a', or b'.

At the first glance, this seems acceptable as a kind of multivalued logic. One is tempted to consider a proposition to which value 1 is assigned, as an "always true" one; then, ones with values a and b as, say 66 and 33%, respectively; and a 0 one as "always false." But, we soon realize that such and actually any numerical valuation is impossible. To see this, it suffices to recognize that any numerical valuation would make a and b comparable with $\bar{a}$ and $\bar{b}$ and that is in contradiction with the main property of o6 lattice – that a and b are incomparable with $\bar{a}$ and $\bar{b}$.

That also means that one cannot construct a simple chip for o6 where we would just have different voltages for 0, a, b, and 1 as in a multivalued logic (different voltages are comparable to each other and that is precluded in o6). Actually, such a chip should have a nonclassical, nonnumerical ports and that is directly correlated with the above property that we must be able to represent propositions that are mutually incomparable, that is, nonordered.

To see how that would work for classical logic (CL), let us consider the following expression, namely,

$$\vdash_{\mathrm{CL}} (A \wedge B) \vee C \equiv (A \vee C) \wedge (B \vee C) \tag{1.1}$$

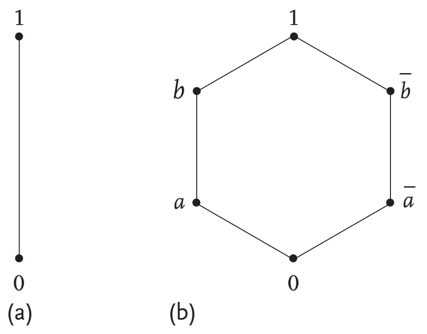

Figure 1.3 (a) Boolean lattice model of classical logic; (b) hexagon lattice model of classical logic. Also called o6 [244, 245, 277].

where $\vdash_{\text{CL}}$ denotes provability of an expression from the axioms of CL or simply that an expression is *true* in CL.

Let us consider possible interpretations of CL, that is, possible semantics or models. To map propositions and expressions formed by propositions, we use a *semantic valuation* (v): a function from the set of all formulas of CL to a set of all formulas of its model. If a model is a lattice, we will have $v(A) = a$, $v(A \wedge B) = a \cap b$, and so on.

Now, if the model is a Boolean algebra, the valuation of the valid CL formula given by expression (1.1) is a well-known property of Boolean algebra (BA) – the *distributivity*:

$$(\forall a, b, c)[(a \cap b) \cup c = (a \cup c) \cap (b \cup c)] \tag{1.2}$$

because if $\vdash A \equiv B$ is true in a BA-valuation, and then $v_{\text{BA}}(A) = a = v_{\text{BA}}(B) = b$ holds in this valuation.

However, if the model is an o6, then the following holds

$$v_{\text{o6}}(A \equiv B) = 1 \Rightarrow v_{\text{o6}}(A) = a \neq v_{\text{o6}}(B) = b\,, \tag{1.3}$$

and as a consequence, we have

$$(\exists a, b, c)[(a \cap b) \cup c \neq (a \cup c) \cap (b \cup c)]\,. \tag{1.4}$$

To prove this, let us take $a = v_{\text{o6}}(A) = b$, $c = v_{\text{o6}}(C) = a$, and $b = v_{\text{o6}}(B) = \overline{a}$, in Figure 1.3b. We obtain $a \cap b = 0$ and $(a \cap b) \cup c = 0 \cup c = c$. On the other hand, we have $a \cup c = a$, $b \cup c = 1$, and $(a \cup c) \cap (b \cup c) = a \cap 1 = a$. Since $c \neq a$, we do not have $(a \cap b) \cup c = (a \cup c) \cap (b \cup c)$.

Nevertheless, in this model, one can prove all the tautologies (theorems) and all the inference rules that are valid in the standard two-valued classical logic [244, 245, 277, pp. 272, 305].

Taken together, logic is a much wider and weaker theory than its lattice models – Boolean algebra, o6, and so on – through which logic can or cannot be implemented in a hardware. Two-valued Boolean algebra is definitely the simplest model for which such an implementation is possible and this determines the choice of the hardware. We can say that the computation is *physical*. Physical hardware determines how fast we can compute a problem, which algebra we shall use for the purpose, and how we can *translate* our problem into the chosen algebra and therefore hardware. This *physical* aspect of computation is of utmost importance for any attempt to speed-up computation, classical or quantum. In the following sections, we will discuss some of them.

1.5
Speeding up Computation: Classical Analog Computation . . .

In the previous section, we showed that the logic we use for reasoning on propositions and operations carried on them can have nonbinary models. On the other hand, although real numbers that we use for everyday calculations can be based on "binary" (two valued) logic. When we want to carry out a calculation, we first have to translate every real number to a binary one which has got more digits. Then, we have to carry out complex gate manipulations in order to apply algorithms that translate otherwise simple operations with real numbers into operations with binary digits. In the end, we have to translate the result back into real numbers. Would it not be faster to make a *real computer* that would be able to deal with real numbers directly?

In the theory of computation, real computers are hypothetical computing machines which can use infinite-precision real numbers. These hypothetical computing machines can be viewed as idealized analog and parallel computers which operate on real numbers. Realistic analog computers existed in the past, but they were abandoned for the following two reasons

1. digital (binary) computers proved to be faster;
2. analog computers have never been developed to fully universal machines.

Although, the latter reason is apparently only a consequence of the former one.

Both digital and analog computational devices are very old. For instance, Chinese *counting rods* and *abacus* digital "computers" (know in practically all ancient civilizations), are over 2000 years old. Analog *Antikythera mechanism* and *astrolabe* for calculating astronomical positions are nearly as old.

What is important for us, though, is that the analog/parallel computers in the "predigital" time were more efficient then digital for particular tasks simply because a special design of hardware enabled faster and more efficient calculation then by means of a universal digital machine. It is important, because, on the one hand, known quantum algorithms (mostly based on the Fourier transform) determine which feature quantum hardware must possess, and on the other, as opposed to current classical computers, massive parallel computation is what characterizes would-be quantum computers and is likely to make them universal. We can say that both analog classical and quantum computers perform a physical calculation.

To better understand what that means, let us have a look at Figure 1.4.

The examples show how we can calculate even irrational (π) using geometrical and physical features of our "hardware" as suitable algorithms for solving particular problems. More sophisticated examples of analog computational devices based on such algorithms are, for example, slide rule and the *Water integrator* shown in Figures 1.5 and 1.6, respectively.

Of course, the analog/parallel computers that were in use after World War II were electronic ones, but the principle stayed the same – physical calculation. With the help of the so-called *operational amplifier* (op-amp)[3] we can add, subtract, multiply, and divide number as well as obtain derivatives and integrals of a chosen function in one step by simply reading output voltages. For example, if we want to divide two numbers, we use a circuit shown in Figure 1.7.

3) The first vacuum tube op-amp was built in 1941.

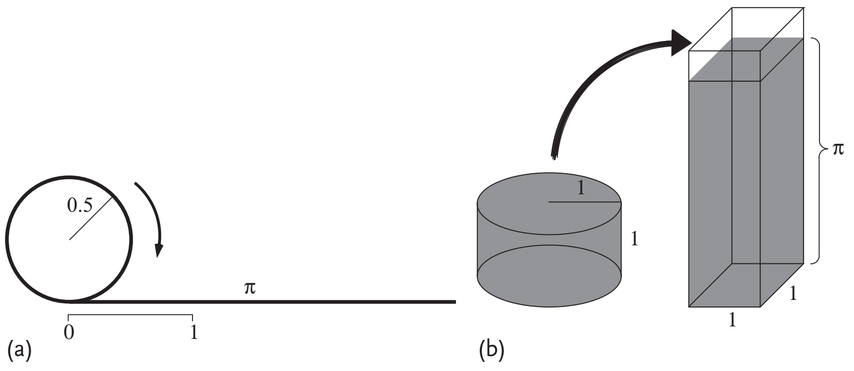

Figure 1.4 (a) "Calculating" π by measuring the length of a string originally wrapped around a cylinder with a radius equal to 0.5; (b) "calculating" π by pouring over water from a cylinder to a vessel whose base is a square 1×1 and measuring the height of the water level.

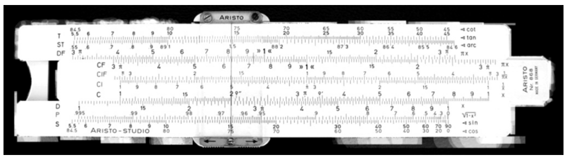

Figure 1.5 A slide rule, essentially being an analog computer, is much more efficient than its digital competitor abacus.

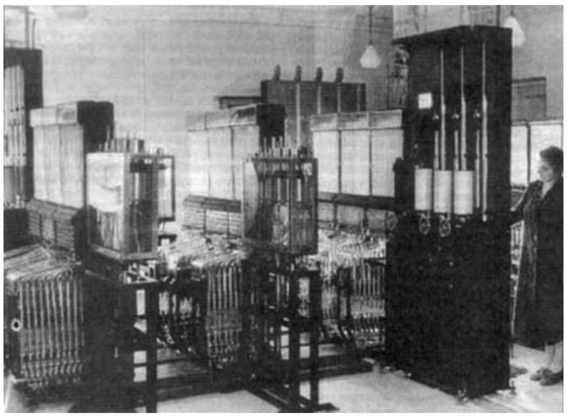

Figure 1.6 A Russian water analog computer built in 1936 by Vladimir Lukyanov. It was capable of solving nonhomogeneous differential equations. Image courtesy of the Polytechnic Museum, Moscow.

Op-amp has a resistor with a very high resistance between − and + terminal so that the current across them is practically zero. Thus, we have $I_- = I_+$ and therefore $V_g = V_+$. Since in our circuit we have $V_g = 0 = V_+$, we must also have $V_- = 0$. Also, $V_{in} - V_- = V_{in} = I_{in}R_{in}$ and $V_- - V_{out} = -V_{out} = I_f R_f$. Then, from $I_{in} = I_f + I_-$, we get $V_{out} = -R_f V_{in}/R_{in}$. By setting R_f to one, we can divide V_{in} by R_{in} in one step. We see that for each division, we have to change V_{in} and R_{in}. When we want to integrate a function, we have to use different elements and for integrations, yet other ones.

There were no such problems with universal digital computers that were advancing rapidly and the *Moore law* finally proclaimed the dead sentence to analog and parallel computers. Moore's law is an *Intel Corporation* self-imposed[4] longterm production road map. After an obviously too ambitious formulation by Gordon Moore, the "law" was recalibrated in 1975 so as to receive the following formulation as announced by David House, an Intel executive at the time [17, 271, 308, 329]:

Moore's Law. *CPU clock speed and the number of transistors on an integrated circuit double every 18 months.*

However, it was obvious from the very begging that both the CPU speedup and its miniaturization as well as miniaturization of memory units would one day hit the quantum wall. Miniaturization has to stop when the bit carriers come down to one electron, when logic gates and memory units come down to one atom and when the conductors between them come down to monolayers. Actually, in the very same year, when the Moore's law received its definitive wording – in 1975 – Robert Dennard's group at IBM predicted that the power leakage which would switch a transistor out of its "off" state should happen by 2001 – shown in the left image of Figure 1.8. They also formulated their – *Dennard's scaling law* – which specified how to simultaneously reduce gate length, gate insulator thickness, and other feature dimensions to improve switching speed, power consumption, and transistor density and ultimately postpone the leakage. However, the fast developing industry

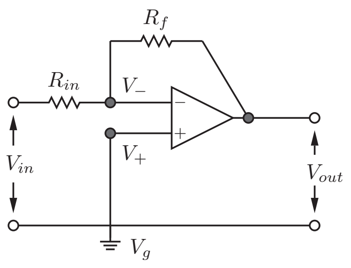

Figure 1.7 Analog computer. Dividing numbers by means of voltages with the help of an operational amplifier.

4) by Gordon Moore, a cofounder of Intel, in the early seventies of the last century

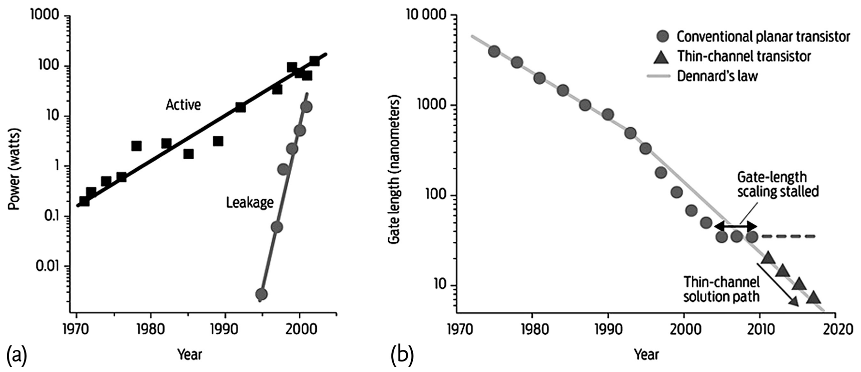

Figure 1.8 Moore's miniaturization will stop by 2020 at the latest. Figure reprinted from [4] with permission from © 2011 IEEE Spectrum magazin.

modified their law and used other technological solutions to pack that 5 billions transistors in a CPU.

However, the quantum wall is inevitable one way or another and now the sizes of gates themselves are approaching the nanometer barrier – an atom has the size of about half a nanometer and, as shown in Figure 1.8b, this will happen by 2020 if the new *thin-channel* solution of designing and connecting transistors proves to be successful [4]. If not, the miniaturization – as Figure 1.8 also shows – has already stopped.

The CPU clock exponential speed-up already hit the wall a few years ago. In 2003, the exponential speed-up turned in a linear one and in 2005 Intel gave up the speed-up completely – see Figure 1.11. In 2008, IBM took over at a pace even slower than linear and dedicated its CPUs to the mainframe usage with a price of over $ 100 000 per CPU. Therefore, after 2005, individuals cannot even dream of speeding up their computations for quite some time to come.

Thus, the researchers started to look for alternatives and turned to parallel computation. For today's market, that meant parallelizing digital computers (we shall come back to this in Section 1.7), but development research turned to quantum and analog computers (again).

For example, the special 2010 issue of *Computers* entitled *Analog Computation* introduces the renewed interest as follows. "Computer scientists worldwide are exploring analog computing under such names as amorphous computing, unconventional computing, computing with bulk matter, nonsilicon computing and other designations. Biologists and computer scientists team up to build "computers" out of neural tissue or slime molds. Physicists design new materials, such as graphenes, whose molecular properties are analogous to the atomic-level quantum behavior. Theoretical computer scientists investigate the complexity of analog computing, and speculate on new complexity classes. All this emerging work has resulted from the limits that physical laws impose on digital computers."

This may enable real computers to solve problems that are inextricable on digital computers. For instance, Hava Siegelmann's neural nets can have noncomputable real weights, making them able to compute nonrecursive languages. Also, the recent development of the massively parallel computer, the so-called *field computer*, indicates that we might be able to solve the so-called NP-problems in a polynomial time.

What that would mean was best explained by Kurt Gödel in a 1956 letter to John von Neumann: "If there actually were a machine with [a polynomial running time] this would have consequences of the greatest magnitude. That is to say, it would clearly indicate that, despite the unsolvability of the Entscheidungsproblem, the mental effort of the mathematician could be completely (apart from the postulation of axioms) replaced by machines."

However, there is a new kind of parallel computers on which we can – in a polynomial time – solve problems which require an exponential time on classical computers. These are *quantum computers* whose physical and parallel computation we are going to analyze in most of the following sections.

1.6 … vs. Quantum Physical Computation

In this section, we shall show – on a small scale – how a quantum computer works – in principle. What is important here is

- a feature of a quantum system – *photon* – called *superposition* which is a nonclassical property and which enables massive parallelism;
- a physical calculation in a polynomial time of a problem whose solving requires an exponential time on a classical computer.

Let us consider a simple experiment consisting of a photon splitting its path at a 50 : 50 beam splitter (BS; a semitransparent mirror), as shown in Figure 1.9. We denote the two possible incoming paths and also the corresponding states of the photon moving along them by $|0\rangle$ and $|1\rangle$. These are the so-called *ket* vectors belonging to Dirac's *bra-ket* notation which we will formally introduce in Sections 1.8 and 1.11. So, either the photon arrives from above and has the state described by $|0\rangle$ or from below in state $|1\rangle$.

The photon can either go through or be reflected from the beam splitter. Let us take the case of photon $|0\rangle$ coming in. If it passes through, its field vector will remain unchanged. But, because it passes through BS with only 50% probability, we multiply its ket by $1/\sqrt{2}$. On the other hand, a vector field reflected from BS undergoes a phase shift $\pi/2$ with respect to the one which passes through it. (See [78] where you have to assume that the lower incoming beam does not contain a photon.) This phase shift corresponds to multiplying the ket by $e^{i\pi/2} = i$, and therefore the reflected photon will be described by $(1/\sqrt{2})i|1\rangle$. Hence, *before* we detect which outgoing path the photon took – by registering a "click" in either D_0 or D_1 – we

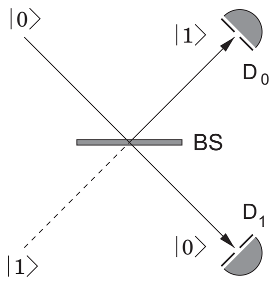

Figure 1.9 Photon at a beam splitter.

describe its state by the following *superposition* (see Section 1.11 for a formal definition) of paths:

$$|\text{out}\rangle = \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle) \,. \tag{1.5}$$

Such a superposition of states is the crucial ingredient of quantum computation.

Let us now use our photon, our beam splitter, and another beam splitter to make a quantum computer prototype. Such a two beam splitter set through which a photon passes is a device known under the name of a *Mach–Zehnder interferometer* and is shown in Figure 1.10.

The path to the second beam splitter (BS) from above is described by $(i/\sqrt{2})|1\rangle$ and from below by $(1/\sqrt{2})|0\rangle$. Here, we can simply reverse the process we have on the first beam splitter as follows. The two paths superpose at the beam splitter so that upper outgoing path is described by

$$\frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}|0\rangle + i\frac{i}{\sqrt{2}}|0\rangle\right) = 0 \,, \tag{1.6}$$

where the second $|0\rangle$ comes from $i|1\rangle$ which was reflected from BS (at the upper side of BS we denote it as $|0\rangle$). The lower path is described by

$$\frac{1}{\sqrt{2}}\left(\frac{i}{\sqrt{2}}|1\rangle + i\frac{1}{\sqrt{2}}|1\rangle\right) = i|1\rangle \,, \tag{1.7}$$

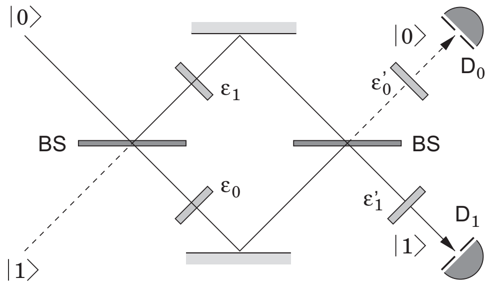

Figure 1.10 Mach–Zehnder interferometer. An incoming $|0\rangle$ ($|1\rangle$) photon will always end up in D_1 (D_0) detector.

Mladen Pavičić: Companion to Quantum Computation and Communication —
Chap. pavicic8481c01 — 2013/3/5 — page 16 — le-tex

where the second $|1\rangle$ comes from $|0\rangle$ which is reflected from BS (on the lower side of BS we denote it as $|1\rangle$) and the first one from a passage of $i|1\rangle$ from above. The phase shifters ϵ_0 and ϵ_1 are set to $\epsilon_0 = 0$ and $\epsilon_1 = 0$ (we tune them off the zero-values to obtain an arbitrary probability of photons exiting through any of the two port). For the zero-values setup, the process at the second beam splitter is just a reversed image of the process at the first one. The probability of detecting the photon by D_0 is 0, and the probability of detecting it by D_1 is $|\langle 1|(-i)i|1\rangle|^2 = 1$.

If we, however, set ϵ_0, ϵ_1, ϵ'_0, and ϵ'_1 so as to make phase shifts (with respect to the state of the incoming photon) ϕ_0, ϕ_1, ϕ'_0, and ϕ'_1, respectively, then the probability of detecting a photon by D_1 is no longer 1, but

$$p_1 = \cos^2 \frac{\phi_1 - \phi_0}{2} = \frac{1}{2}(1 + \cos\phi) \ , \tag{1.8}$$

where $\phi = \phi_1 - \phi_0$. Note that the probability would stay the same if we took out the phase shifters ϵ'_0 and ϵ'_1 which means that the result depends only on the phase difference $\phi_1 - \phi_0$.

Let us see how we can use the result to factor numbers in order to illustrate Shor's algorithm (short of entanglement and the corresponding speed-up, which we are going to address later on), following Johann Summhammer [301].

We obtain the factors of a chosen number, say N, in a "physical" way using the setup shown in Figure 1.10 of the previous section and (1.8). Let us increase the phase shift ϕ in discrete steps $2\pi/n$ so as to have $\phi_j = 2\pi k N/n$, $k = 1, \ldots, n$. If we let n photons through the device: $k = 1, \ldots, n$, the sum of all individual probabilities that the detector D_1 would register a photon – given by (1.8) – will be:

$$I_n = \sum_{k=1}^{n} p_1(k) = \frac{1}{2}\left[n + \sum_{k=1}^{n} \cos\left(\frac{2\pi k N}{n}\right)\right] . \tag{1.9}$$

If n were a factor of N, we would have $p_1(k) = 1$ and $I_n = n$. If not, the cosines would roughly cancel each other and we would get $I_n \approx n/2$. If n were a factor of N then only detector D_1 would react and if n were not a factor of N, then on average we would get half of the clicks in D_1 and half in D_0. So, if we perform n measurements and obtain n clicks in detector D_1 then n is a factor of N.

The numbers we can factor in this way are not big, but the result is very instructive for understanding the problems we face with classical computers and the way we can solve them with quantum ones. For the light with $\lambda = 500$ nm, using a continuous wave (CW) laser (for example, Nd:YAG) with which we can have the *coherence length*, Δl – the length over which the phase is fairly constant – of up to 300 km. The corresponding *coherence time* is $\Delta t = \Delta l/c$. The Heisenberg uncertainty relation for energy and time $\Delta E \Delta t \geq \hbar$ and the Planck postulate: $E = h\nu$ give $\Delta\nu\Delta t \approx 1/4\pi$, where $\Delta\nu$ is called the *bandwidth*. From $c = \nu\lambda$ by differentiation we get $\Delta\lambda = -c\Delta/\nu^2 = -\lambda^2\Delta\nu/c$, where $\Delta\lambda$ is called the *linewidth*. Dropping the minus sign which only shows that the changes of $\Delta\nu$ and $\Delta\lambda$ are opposite and using the previous relations we get: $\Delta l \approx \lambda^2/\Delta\lambda$. To keep the linewidth at $\Delta\lambda \approx 10^{-17}$ is feasible since it corresponds to the coherence length $\Delta l \approx 25$ km.

In our setup, at each phase step $\Delta\phi = 2\pi/n$ a photon is sent into the interferometer. The phase difference $\Delta\phi$ in our interferometer is proportional to $\Delta o/\lambda$, where Δo is the *optical path difference* [33]. The Δo must be smaller than the coherence length and we can estimate that $n < \lambda/\Delta\lambda$.

Hence, the biggest numbers we could factor are $N \approx 10^{10}$ and any PC can factor a number with 10 digits in a fraction of a second. However, the important property of this example of physical computing is that our "transistor," Mach–Zehnder interferometer, is faster per computing unit (quantum gate) than the standard classical transistor for the same "clock" speed.

The longest factorization test according to (1.9) will take time proportional to nN, because the maximum value of k is n. Since the largest n we have to check is $\sqrt{N}$, the maximum time would be proportional to $N^{3/2}$. The required time is therefore a polynomial function of N.

A direct and the most inefficient algorithm of factoring a number would simply be $\sqrt{N}$ trial divisions. Hence, the number of checks the most inefficient classical factoring algorithm has to carry out is smaller than $N^{3/2}$ we obtained for our "physical calculation" above. Still, given the same clock frequency, a classical computer calculation is slower per computing unit (gate). There are two reasons for this. First, we have to turn numbers into bits, and then we have to carry out binary operations that correspond to division (which is one of the most complicated basic computer operations). The number of used transistors, that is, loops needed for the operations increases exponentially with N and that means that the required time is an exponential function of N. In other words, we obtained an exponential speed-up of factorization of numbers on our optical "quantum analog device" with respect to a binary computer cracking.

As opposed to a computer search-verify procedure, the photon search-verify Mach–Zehnder factorization procedure is instantaneous for each photon. The problem is that we cannot calculate much with only one Mach–Zehnder interferometer. We could parallelize the calculation by putting another Mach–Zehnder interferometer at each output of the first one, then putting another Mach–Zehnder interferometer at each output of the previous one, and so on (see Figure 2.1 in Section 2.2). However, that would mean an exponentially growing number of elements, causing us to lose the advantage we gained. We will show how to get around this later on. But, before we dwell on the solution to this problem, we should first show why do we need to speed-up our calculation at all.

1.7 Complexity Limits: Exponential Time

We have mentioned that the Moore law already hit the clock wall (see Figure 1.11) and that it will soon hit the quantum shrinking barrier – single electron transistor (SET) and monolayer conductors.

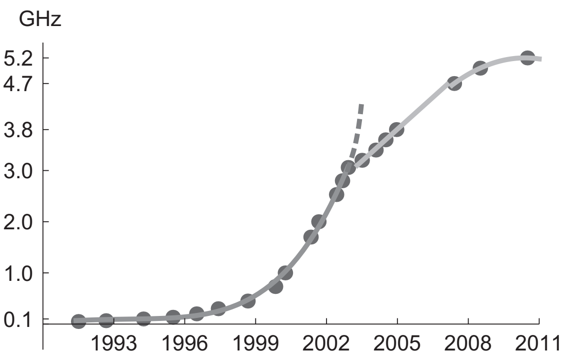

Figure 1.11 CPU's clock frequencies: Intel 486-50 MHz June 1991, DX2-66 August 1992, P(entium)-100 March 1994, P-133 June 1995, P-200 June 1996, PII-300 May 1997, PII-450 August 1998, PIII-733 October 1999, PIII-1.0 GHz March 2000, P4-1.7 April 2001, P4-2.0 August 2001, P4-2.53 May 2002, P4-2.8 August 2002, P4-3.0 April 2003, P4-3.4 April 2004, P4-3.6 June 2004, Intel P4-3.8 November 2004, IBM Risc Power-6 4.7 GHz June 2007, 5 GHz August 2008, IBM zEnterprise 196 (z196) 5.2 GHz August 2010.

Since 2004, when the clock frequency corollary of Moore's law died,[5] parallel processing has been introduced into the very processors: dual cores, quad cores, . . . 16 cores. In a way, these processors are just mini clusters. Clusters and interconnected mainframe units have been used for decades to speed-up computation using algorithms that can distribute parts of a task to many CPUs in parallel. But, is that efficient? Actually, for the hardest computing problems we have to solve in various applications, neither classical parallelization nor a speed-up of CPUs are efficient in the sense of obtaining results proportionally faster with higher speed or a higher number of CPUs. Here is why.

Computing problems are categorized according to their complexity in the so-called *complexity classes*. The problems are defined by their models of computation and before they are considered as decision problems that algorithms have to resolve to reach a decision, that is, the final outcome. There are many undecidable problems as, for example, the so-called *halting problem*: "Given a description of a program and a finite input, decide whether the program finishes running or will run forever."

It is often said in the literature that already Alan Turing proved that no Turing machine can solve the halting problem, i.e, that it is undecidable for Turing ma-

5) A widespread rendering of the law: "The number of transistors on a single integrated-circuit chip doubles every 18 months" [28] does not correspond to the historical data which show 26 months [42]. Moore himself commented: "I never said 18 months. I said one year [in 1965], and then two years [in 1975]. One of my Intel colleagues changed it from the complexity of the chips to the performance of computers and decided that not only did you get a benefit from the doubling every two years but we were able to increase the clock frequency, too, so computer performance was actually doubling every 18 months. I guess that's a corollary of Moore's Law. Moore's Law has been the name given to everything that changes exponentially in the industry. I saw, if Al Gore invented the Internet, I invented the exponential." [271, 308, 329]

chines. But, that is only to be expected because a deterministic Turing machine stops after solving a decidable problem by definition. It can say nothing about a theory for which it cannot be defined. So, although there are many other undecidable problems, for us, only those problems that have an algorithm for their solving will be of interest.

We shall be even more specific and will concentrate on the *time complexity* of algorithms, although there is also a space complexity. We shall do so because one of the main advantages of quantum computers is that they are expected to require a polynomial time for solving problems for which classical computers would require an exponential time.

Thus, the class EXPTIME is the set of decision problems that can be solved by some algorithm in an exponential time, the class NP is the set of decision problems that can be solved by a nondeterministic Turing machine in polynomial time, while the class P is the set of decision problems that can be solved by a deterministic Turing machine in a polynomial time.

We stress here that all the problems we shall consider do have some algorithms for their solution. Thus, our main problem with quantum computation will not be to find algorithms for a computation of particular programs in general, but to find algorithms which will be exponentially faster (Shor's algorithm) or at least a few polynomial orders faster (Grover's algorithm) than classical algorithms. Such a speed-up is also possible in the realm of classical computation. Since a classical speed-up can compete with and even outdo quantum ones, some details might be helpful.

Let us consider P and EXPTIME problems for which there exist algorithms described by means of functions $f(n) = a_i n^i$, $i = 1, 2, 3$, $g(n) = b2^n$ and $h(n) = c3^n$, where a_i, b, and c are constants. We shall say that the algorithm is *of order* $O(n)$, $O(n^2)$, $O(n^3)$, $O(2^n)$, and $O(3^n)$.

From Table 1.2, we see that when we take a linear problem that we solved within one day on a personal computer (PC), increase its size by a factor of 1000, and put

Table 1.2 Problems of a polynomial complexity, that is, problems from a P class, can really take advantage of a speed-up of classical computers ($100N_3^3 = x^3 \Rightarrow x = \sqrt[3]{100}N_3 = 4.64N_3$). However, for a problem of an exponential time complexity the speed-up is limited to an additive constant. For example, $100 \cdot 2^{N_4} = 2^x \Rightarrow x = N_4 + (\log 100)/(\log 2) = N_4 + 6.64$.

Time complexity	**Size of a solvable problem in a time unit**		
	In 1991	**Today (2013; on 100 times faster CPU)**	**Today on a cluster with 1000 CPUs (10^5 times faster)**
n	N_1	$100N_1$	$100\,000N_1$
n^2	N_2	$10N_2$	$316.2N_2$
n^3	N_3	$4.64N_3$	$46.4N_3$
2^n	N_4	$N_4 + 6.64$	$N_4 + 16.6$
3^n	N_5	$N_5 + 4.19$	$N_5 + 10.5$

it on a cluster with 1000 CPUs, we will obtain a result also within one day. If one does that with a problem from an EXPTIME class, the required time would exceed the age of the Universe even on whatever cluster we have today.

This was the reason why the following definitions have been proposed.

Definition 2 A polynomial algorithm of a problem is called *feasible* [73].

We often simply say that such a problem is feasible.

Definition 3 A problem which does not have a feasible algorithm is called *intractable*.

Definitions 2 and 3 are not always appropriate because

- Constant factors and lower terms in a polynom can make an "intractable" problem feasible and a "feasible" one intractable. For example, an algorithm that would take time $10^{100}n$ cannot be carried out, but is nevertheless called "feasible" because it is in P, while an algorithm that takes time $10^{-1000}2^n$ can easily be carried out for n as large as 1000, but is called "intractable" because it is in EXPTIME;
- The size of the exponent and of the input can have the same effect.

Still, we do not encounter such unfavorable cases often and therefore the definitions are widely accepted. But, we have to keep in mind that "intractable" does not mean that a problem cannot be computed or that we do not have an algorithm for it. It simply means that we have to spend more time or that we do not have enough money to solve the problem.

Let us have a look at a few problems: *Euler tour*, *Traveling salesman*, and *factoring number* ones.[6]

The first one is the Euler tour problem for a multigraph. An Euler tour is a tour which covers all the edges but none of them more than once. For example, the multigraph shown over Königsberg bridges in Figure 1.12 does not have an Euler tour. It is shown here because Euler formulated his tour problem and found a linear algorithm for it while solving the Königsberg bridges problem.

Definition 4 A *graph* $G = (V, E)$ consists of a set (V) of *vertices* (points) and a set (E) of *edges* (lines), each of which connects two vertices. A *multigraph* is a graph which has multiple edges.

6) To that, we can add the *satiability problem* (SAT) and the *isomorph-free generation of graphs and hypergraphs*. SAT problem consists in verifying whether Boolean expressions like that one shown in Table 1.1 are satisfied, that is, true, that is, equal to 1 in a Boolean algebra. SAT belongs to EXPTIME and that will help us understand why the factoring number problem computed in a digital computer belongs to EXPTIME too. Graphs and hypergraphs can be used to map nonlinear equations into hypergraphs and filter out equations that have solutions. They also belong to EXPTIME. We shall use them later on to generate the so-called Kochen–Specker sets.

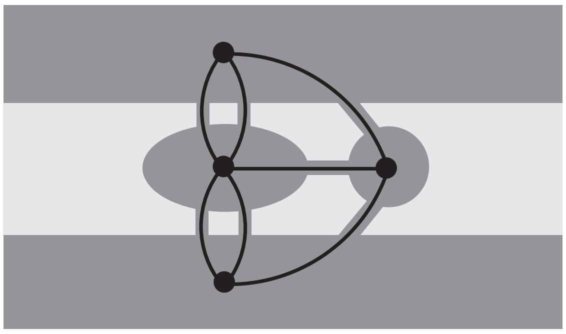

Figure 1.12 Euler tour on the example of Königsberg bridges. Is it possible to take a tour over the bridges, crossing each one only once?

A brute force approach to this problem gives us a search algorithm of complexity order $O(n!)$ (where n is the number of edges) which is even harder to solve than those of order $O(c^n)$. For instance, the number of paths we have to verify for the Königsberg bridges is $(7-1)!$. If a computer needs 1 sec to verify all of them, then the time required to verify paths for twice so many (14) bridges is $13!/6! \approx 100$ days.

A similar problem is the traveling salesman problem (TSP) which consists of finding the cheapest way of visiting all given cities and returning to your starting point. The vertices are cities and edges are routes between any two of them. Each link (edge) connecting two cities (vertices on the edge) is pondered by the cost of going from one of the two cities to the other. Since this is a realistic problem every travel agency would like to have a solution for, we will first try to estimate to what extent they can be of service to their customers if they use a brute-force algorithm which is again of the order $O(n!)$.

Let us assume the agency has a fast machine which provides the cheapest route for 10 cities in 1 s. Should they try to serve a demanding customer who would like to make the cheapest tour through 25 cities? Well, the required time is about 136 billion years or 10 ages of the Universe.

Therefore, a better algorithm for such problems are wanted. But no general approach has been found so far. For instance, it is not known whether the NP set strictly contains the P set or perhaps coincides with it. So, problems are approached individually or according to some features they share.

Euler proved that connected graphs have an Euler tour if every vertex shares an even number of edges and this immediately reduced the time complexity of the problem from EXPTIME to linear P.

S. Lin found a good approximate algorithm of order $O(n^3)$ for the traveling salesman problem. With the help of this algorithm, the agency would be able to serve its customer within 15.6 s, if only approximately.

The next problem of factoring numbers will show us how the complexity of an important application of algorithms we make use of every day depends on a platform we use to solve problems and how we can find faster algorithms on new platforms.

As shown in Section 1.6, the complexity of algorithms for factoring numbers on our photon prototype device is of order $O(n^{3/2})$. The latter algorithms could use the electric analog machine for resetting the device. But, the number of voltage steps an analog computer can tell from each other is also limited. A more sophisticated analog computer would, when compared with a digital one, have two disadvantages: a much lower speed and an inefficient error correction. Therefore, for the time being, a digital computer is the only option for the task.

However, to introduce a natural number N we want to "crack" into a digital computer we have to translate it into a binary string:

$$N_2 = \alpha_{n-1}\alpha_{n-2}\dots\alpha_1\alpha_0 \tag{1.10}$$

where α_i, $i = 0, \dots, n-1$ are determined from the following equation:

$$N_2 = \alpha_{n-1}2^{n-1} + \alpha_{n-2}2^{n-2} + \dots + \alpha_1 2^1 + \alpha_0 2^0 = \sum_{i=0}^{n-1} \alpha_i 2^i \,. \tag{1.11}$$

For instance, to obtain a binary representation of 255, we divide it by 2 until we reach 1. Reminders determine bits. So, 255/2 is 127 with the remainder $\alpha_0 = 1$, and so on, down to $\alpha_7 = 1$ and we get 11111111. In the opposite direction, we have $2^0+2^1+2^2+2^3+2^4+2^5+2^6+2^7 = 255$. For 256, we have all the remainders, but the last one equal to zero: $\alpha_0 = \alpha_1 = \ldots = 0$. The last one (of 1/2) is, of course, 1. Thus, we get 100000000 and $2^8 = 256$.

A brute force algorithm for the task consists of trial divisions using basic Boolean operations by means of logic gates shown in Figure 1.2 combined in binary adders as mentioned in Section 1.4 and explained in [239, Section 1.16]. That means that such a search would be of order $O(2^n)$ or higher, where n is the number of bits.

The majority of encryption we use today for bank and Internet transactions are based on composite numbers consisting of two huge prime numbers. They are called RSA after Ron Rivest, Adi Shamir, and Leonard Adleman who invented this encryption method in 1978 [269]. And again, many faster *subexponential*7) algorithms have been found.

RSA company provides harder and harder challenges every year to stimulate finding better algorithms. On December 12, 2009, such a number with 768 bits and 232 digits was cracked8) after 2000 CPU years of computation, meaning that it required one month on a cluster with 24 000 CPUs. Also in 2009, a group of enthusiasts factored 512-bit RSA keys for Texas Instruments calculators using software found on the Internet and a distributed computing project. Since 512-bit RSA numbers are the standard for almost all Internet keys, the aforementioned crackings stirred a debate on the RSA keys security.

7) Subexponential or superpolynomial complexity is the one between P and EXPTIME.
8) Number Field Sieve algorithm of subexponential complexity $O\{\exp[c(\log n)^{1/3}(\log\log n)^{2/3}]\}$ was used.

The response of the companies will most probably be to simply switch to a 1024-bit standard, but two issues emerge here. First, already now, all previously illegally intercepted documents encoded by older 256- and 128-bit keys are easily readable. Soon, all intercepted and stored 512-bit ones will be readable. Second, tonight a mathematician somewhere in his attic room can come with an ingenious P algorithm for factoring numbers and crash down the security of the World Internet as of tomorrow morning.

Here, quantum computation and quantum communication can provide a patch.

What the Internet needs is to make connections secure and eavesdropping impossible and that is what quantum cryptography can provide the Internet with already today.

What computation needs is a speed-up, and that is what the hardware of would-be quantum computers together with quantum software of the "Shore kind" can provide us with.

However, before we dwell on these two issues, we first want to consider another important point that will give us a bridge from classical to quantum platforms and from classical to quantum formalism. We have already mentioned that the classical technology already "went parallel" and that means a lot of CPUs, that is, a lot of heat. So, the final issue we have to elaborate on before we go completely quantum is the issue of energy.

1.8
Energy Limits . . .

According to the Environmental Protection Agency (EPA) US Congress report in 2007, the energy used by servers and data centers in the US is estimated to be about 61 billion kWh in 2006 (1.5% of total US electricity consumption) for a total electricity cost of about $ 4.5 billion [265]. This estimate includes neither office nor private PCs and it is evaluated to be higher than the electricity consumed by all color televisions in the US.[9]

EPA also estimated the energy use of servers and data centers in 2006 to be more than double the electricity that was consumed for this purpose in 2000, and that the power and cooling infrastructure that supports IT equipment in data centers also uses significant energy, accounting for 50% of the total consumption of data centers [265]. Taken together, servers and data centers together with their infrastructure in 2006 spent 2.25% of total US electricity consumption. Similar statistics are available for Europe. Intensity of computations constantly going on in Europe

9) The total energy consumption of energy related to computers (in industry, offices, and at home) and Internet (including cooling, personal, maintenance, rooms, and so on) is independently estimated to be between 3 and 9% (in the US), but no detailed study has been carried out in at least 10 years. This is partly because computers are so much a part of production, education, communication, traveling, and everyday life that it is practically impossible to determine the energy spent by them from the total amount of spent energy.

is obvious from the European Particle Physics Real Time Monitor shown in Figure 1.13.

Should we add the new *parallel law*

- *The energy spent by clusters and data centers doubles every five years*

to the dying Moore law.

Apparently, still not. Because the energy spent in the subsequent five year period (2005–2010) did not double [153]. It only increased by about 60% worldwide. There are two main reasons for that. The first is the global crisis occurring in the past few years. The second is the recent virtualization of computational tasks. Recently developed cloud computing installations have higher server utilization levels and infrastructure efficiencies than in-house data centers. But, since the latter filling of the presently existing computational "vacancies" in the existing in-house servers will soon saturate them and since most reports do predict an exponential growth in energy spent by the data centers, the parallel law will continue to hold in its exponential formulation.

On the other hand, the designers of computers and the Internet argue that their efficiency has increased several times over in the last twenty years. Previous NMOS and PMOS transistors dissipated heat through their resistors while today's CMOS gates dispense with resistors; optical fibers substitute copper wires; resistance within conductors is being lowered by reducing the number of electrons within a gate from thousands to one hundred and soon it will be reduced to one in single electron transistors (SET). Can this development outweigh the exponential increase of processed and stored petabytes (PB, 10^{15} byte)? Here, we should mention that in the face of all the mentioned improvements in the efficiency there is a part of the Moore law that has outperformed itself recently and that is about the heat dissipated by the processors since the dissipated heat doubles not every 18 months, but each year or less.

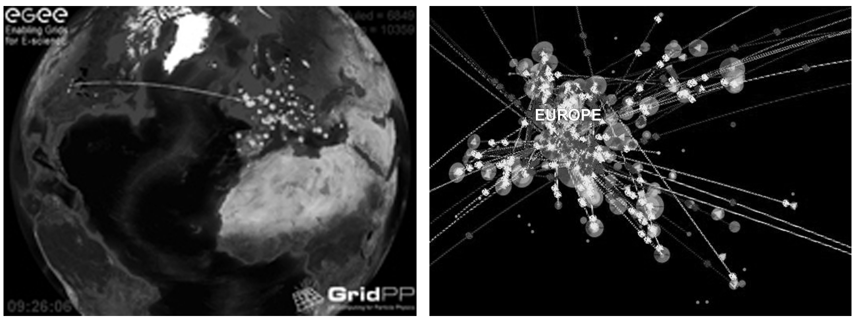

Figure 1.13 Distributed or grid computing consists of sharing computing tasks over multiple computer clusters. Elementary particle physics tasks constantly running on European EGEE and GridPP (distributed over all national European Grids and with links to American (both), Asian, and Australian computing clusters) are shown (57 226 tasks running (dots; bigger dots mean bigger clusters), 21 884 queueing). Reprinted with permission of the UK Grid Operations Support Centre; © GridPP.

The problem is that when we come down to one electron per gate, we are left with pure "information heat" and for so many bytes, it becomes considerable. The information energy that is dissipated in gates just because they compute data or erase data is then significant. For thousands of electrons per transistor erasing data can be compared with erasing data from a book. When we burn two books, one with blank pages and the other with Galileo's *Dialogue on the Two Chief World Systems* printed in it, we would not be able to detect a difference in dissipated heat. But, when we go down to just several electrons we move around or several atoms whose magnetization in a hard disk[10] we changed, then computation and communication become physical. The energy needed for creating or erasing one bit of information directly corresponds to the energy needed to move or change its carrier. Let us calculate this energy.

We shall do so by means of an ideal gas model. We put gas consisting of atoms in a cylinder with a piston as shown in Figure 1.14. Pressure which atoms exert on the piston is $p = F_x/a$ where F is the force with which atoms bounce onto the piston and the walls of the cylinder. So, the work done by the gas is

$$W = \int F_x dx = \int p\, dV\,; \tag{1.12}$$

because $dV = a dx$.

We assume that the gas is in a bath at a constant temperature T. The law of ideal gas reads $pV = NkT$, where N is the number of atoms and $k = 1.381 \times 10^{-23}$ J/(molecule K) is the Boltzmann constant. Since the temperature T is constant, the average kinetic energy of the atoms does not change and therefore there is no change of the internal energy. Hence, according to the first law of thermodynamics, work W is equal to the heat Q transfered to a heat reservoir.

Our process is reversible (there is no friction and by returning the heat by means of a reversible process attached to ours, we can restore its initial state) and therefore, using the second law of thermodynamics (the definition of the entropy change for a reversible process is $\Delta S = Q/T$), from (1.12), we obtain

$$\Delta S = \frac{Q}{T} = \frac{1}{T}\int_{V_i}^{V_f} \frac{NkT}{V} dV = Nk \ln \frac{V_f}{V_i}\ . \tag{1.13}$$

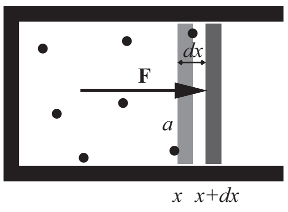

Figure 1.14 Work done by ideal gas during isothermal expansion.

10) Thin layer of magnetic material hard disks are coated with layers 10 nm thick.

Now, let us put just one atom in an empty cylinder – as shown in Figure 1.15. A Maxwell demon watches it and when it is in the left half of the cylinder, he records it ("1") and introduces a piston (adiabatically and reversibly) in the middle of the cylinder (a). In this way, he records one bit of information in cylinder's memory "for free." When he wants to erase this information from its memory, he allows the piston to move freely (without friction) and isothermally. The atom will do work against the piston and push it to the right (b). Our demon now takes out the piston (reversibly and adiabatically) and removes "1"; one bit is erased (c). The cost is given by (1.15).

The entropy increase of the environment caused by erasure of one bit of information is

$$\Delta S = \frac{Q}{T} = k \ln \frac{2V_i}{V_i} = k \ln 2\,. \tag{1.14}$$

This is known as the *Landauer principle* [165]. The dissipated heat caused by the erasure of one bit is

$$Q = kT \ln 2\,. \tag{1.15}$$

Instead of dividing the cylinder in two compartments, the demon could have divided it in 4 or 8 or any number w of possible states. Then, we arrive at the famous Boltzmann microscopic entropy

$$\Delta S = k \ln w \tag{1.16}$$

which is as epitaph engraved in Boltzmann's gravestone.

From (1.14), it follows that we cannot discard information in a computer without dissipating heat, no matter how clever we design our circuits. This is a physical law which we cannot go around because we have to assume some work on the part of the computer (atom in Figure 1.15) at least when the calculation is over and the output has to be obtained. This corresponds to "removing of 1" in Figure 1.15b; also, if the demon simply adiabatically removed the piston in (b), then the system would not be in any way connected to its environment and would not provide us with any output. But, we can carry out calculation without discarding information on each step of calculation and that can save us from unnecessary heat dissipation. Let us see how we can do that.

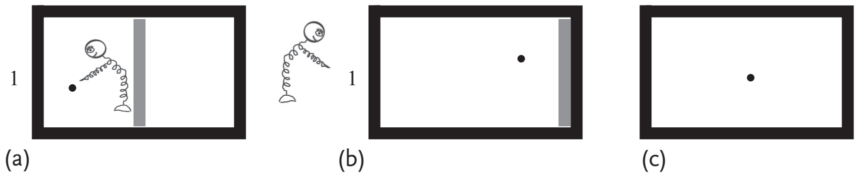

Figure 1.15 Entropy of single atom gas.

1.9
. . . and Reversible Gates

When we, in a decade or two, scale down the transistors to one electron (single electron transistors, SET) and the conductors to monolayers one atom thick, that is, when all Moore's laws die, we will have to take care of "information garbage," that is, the informational heat it produces, given by (1.16).

Since transistors in such atom-level processors will be extremely densely packed – already today their number exceeds 5 billions[11] – we have to think of a way to get rid of the huge amount of heat per volume unit the discarded bits would develop.

And, the best way to get rid of the heat the gates (transistors) would produce is to make gates that do not produce heat. That was the idea (in the early eighties) of reversible computers that would be able to calculate running both "forwards" and "backwards" – like a pendulum – without either dissipating or taking in new energy while calculating.

However, can the binary Boolean algebra and its gates support such swinging reversible circuits?

Let us have a look at Figure 1.16 (compare with Figure 1.2). By looking at the output of a NOT gate, we immediately know what the input was. So, it is reversible. If we keep track of any of the two inputs of an XOR gate, we can reconstruct the other input by looking at it outputs. However, to be able to reconstruct the inputs of an AND gate, we have to keep track of both of them because by knowing that both the output and one of the inputs were 0, we still cannot know whether the other input was zero or one. So, the answer is in the negative. Standard logic gates cannot be implemented in a reversible circuit.

But, if we collect input and output data of a gate, that would suffice to make any operation reversible. Such three-level gates are called the gates of *logic width 3*. (The standard binary logic gates have therefore logic width 1.) Bits at the first two incoming ports of reversible ports are often called the *control* bits or *source* bits, the input bits *target* or *argument* bits, the output ones *result* bits and the one that are not used in further calculations *sink* bits or *garbage*. The terminology will often depend on the kind of gate we will use. With the Fredkin gate [98] (Table 1.3), we obtain different operations at different ports of the gate. With the Toffoli gate, we obtain

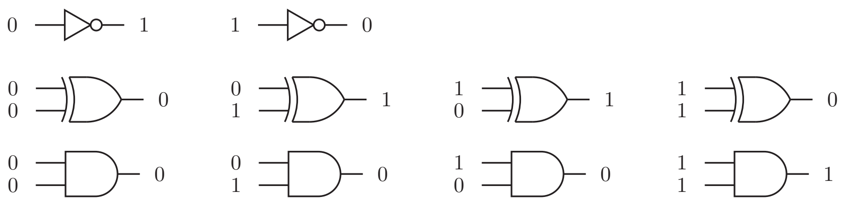

Figure 1.16 Gate NOT is reversible. For the XOR gate to be reversible, at least one of the inputs has to be kept in memory. For AND, both inputs should be kept in memory.

11) Intel 62-core Xeon Phi.

Table 1.3 Truth table of the Fredkin gate – a universal reverse gate that can be used to implement any other gate. Two examples are given. Encircled are the values of the *control* (0) and *result* bits for *controlled* AND gate. Boxed are *control* (1) and *result* values for *controlled* OR gate. Arrows show another way of implementing the latter gate.

Fredkin gate					
Input–output ports			Output–input ports		
Port 1	Port 2	Port 3	Port 1	Port 2	Port 3
0	0	$\enclose{circle}{0}$	0	$\enclose{circle}{0}$	0
0	0	$\boxed{1}$	0	1	$\boxed{0}$
0	$\rightarrow 1$	$\enclose{circle}{0}$	0	$\rightarrow\enclose{circle}{0}$	1
0	$\rightarrow 1$	$\boxed{1}$	0	$\rightarrow 1$	$\boxed{1}$
1	0	$\enclose{circle}{0}$	1	$\enclose{circle}{0}$	0
1	0	$\boxed{1}$	1	0	$\boxed{1}$
1	$\rightarrow 1$	$\enclose{circle}{0}$	1	$\rightarrow\enclose{circle}{1}$	0
1	$\rightarrow 1$	$\boxed{1}$	1	$\rightarrow 1$	$\boxed{1}$

a result mostly at the last output port. We can also use different ports as control ones. For instance, the Fredkin gate originally used input port 2 for control bits and output 2 to obtain operation OR (indicated by arrows in Table 1.3).

The truth values of the Fredkin gate show that we can run it backwards as well. That prompted Fredkin and Tofolli (in 1982 [98]) to propose another way of representing the Fredkin gate which could be integrated in a circuit and enable experimental and industrial implementation. It is shown in Figure 1.17.

In 1985, Richard Feynman [95] recognized that the ability of reversible gates to run backwards as well as forward is just the main feature of the unitary evolution of any quantum system. Thus, he proposed a concept of *quantum mechanical computers* which would essentially use the gates and circuits proposed for reversible computers only applied to *quantum bits*: photons, electrons, and atoms.

Feynman recognized that the *Toffoli gate* and circuits proposed by Tommaso Toffoli in 1980 [303] are better suited for a would-be quantum application and that the Toffoli gate is but one gate in a series of scalable gates which he called NOT, CNOT (CONTROLLED NOT), CCNOT (CONTROLLED CONTROLLED NOT), ... The Toffoli gate is a CCNOT gate. Feynman–Toffoli circuit notation enables an easy handling of gates and is widely accepted in both fields – reversible and quantum computer research.

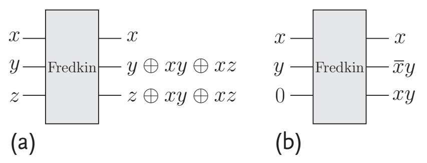

Figure 1.17 (a) General schematic of the Fredkin gate; (b) Schematic of the implementation of an AND gate by means of the Fredkin gate; We can easily check that it is a special case of (a) (see the caption of Figure 1.2). In that way, we can write down any reversible Boolean gate by means of the universal Fredkin gate.

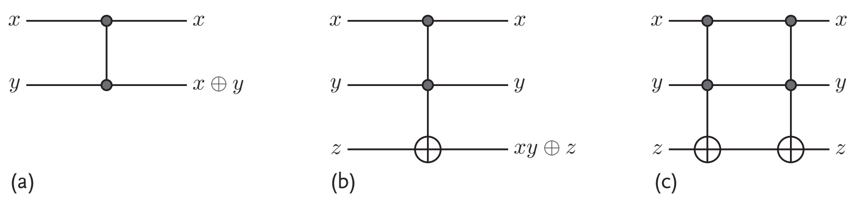

Figure 1.18 (a) General schematic of a CNOT gate; $x \oplus y = x\overline{y} + \overline{x}y$; For $x = 1$, we obtain $1 \oplus y = \overline{y}$; (b) General schematic of a CCNOT gate ($xy \oplus z = \overline{xy}z + xy\overline{z}$); For $x = y = 1$, we obtain $1 \oplus z = \overline{z}$; (c) Reversibility of the CCNOT shown by two concatenated CCNOT gates. This is equivalent to first running CCNOT forward as in (b) and then backward.

Circuit formalism for CONTROLLED-…NOT gates is shown in Figure 1.18. If we concatenate two CCNOT gates, then the 3rd port takes the output of the first gate as its input and the 3rd port of the second gate gives us

$$\begin{aligned} xy \oplus (xy \oplus z) &= \overline{xy}(\overline{xy}z + xy\overline{z}) + xy\overline{(\overline{xy}z + xy\overline{z})} \\ &= \overline{xy}z + xy(xy + \overline{z})(\overline{xy} + z) \\ &= (\overline{xy} + xy)z = z\,. \end{aligned} \tag{1.17}$$

This result is graphically presented in Figure 1.18c.

We can see that we obtain the input z we started with and therefore the gate is reversible, actually self-reversible. We can also see that in CC…NOT gates, the control bits and target-result bits are separated and that the control bit inputs are identical with control output ones and are therefore conveniently designed for scaling circuits containing them.

Since the values of the control bits stay the same and the target bit involves symmetric Boolean operation NOT, it is easy to describe the action of the gate on the input state by a matrix. The matrix representation of the three-level CCNOT gate – shown in (1.18) – consists of an "operator-matrix" that just takes care of swapping values of the target bit and "state matrices" that are just columns of the CCNOT truth table as shown in Table 1.4. Actually, this matrix representation seems to have been adopted from the quantum formalism in the classical reversible computation literature. We nevertheless write it here to point to some differences between properties of classical reversible gates and circles and their formalism, on the one side, and quantum ones, on the other.

$$\begin{bmatrix} 1 & & & & & & & \\ & 1 & & & & \mathbf{0} & & \\ & & 1 & & & & & \\ & & & 1 & & & & \\ & & & & 1 & & & \\ & & & & & 1 & & \\ & & \mathbf{0} & & & & 0 & 1 \\ & & & & & & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{bmatrix} \tag{1.18}$$

Table 1.4 Truth tables of CNOT (controlled NOT) and CCNOT (controlled controlled NOT) gates. The latter gate is also called the *Toffoli gate*. C stands for control and T for target bits.

CNOT gate			
In–out		**Out–in**	
C	**T**	**C**	**T**
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

CCNOT (Toffoli) gate					
Input–output ports			**Output–input ports**		
C1	**C2**	**T**	**C1**	**C2**	**T**
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

We see that apart from the target bit values, all the off-diagonal elements in the matrix are equal to zero. The matrix is equal to itself transposed and multiplied by itself transposed it gives a unit matrix. Since it is a real matrix, it is therefore a unitary matrix as a matrix of a quantum operator. Therefore, its action can clearly be reversed. We can obtain this result by multiplying (1.18) by the matrix from the left. The matrix multiplied by itself is equal to $\mathbb{1}$ and we obtain (1.18) with the reversed positions of "state matrices."

However, an almost diagonal form of the matrix means that the gate exerts only a limited action on the "state matrices." If we wanted to implement other operations, we would have to tamper it with the latter matrices and use both control and target bits as shown in Figure 1.19. As we can see in Figure 1.19c, we use not only the target level, but also the control levels to introduce parameters for obtaining the results. This makes building up circuits more demanding than in a standard binary computer so far as the number of gates is concerned. For example, a comparison of a reversible parallel adder with a standard binary shows that about 40% more gates is needed. This is quite acceptable, though, because both implementations have the complexity $O(n)$. The power consumption, on the other hand, is reduced to 10% of those in the standard chips [76].

On the other hand, we have some restrictions on the circuits that we do not have for the binary circuits. For example, real hardware fan-outs (copies of gate outputs) are not allowed because such copying is irreversible – number of input signals is one and there should be two or more output signals and this is not possi-

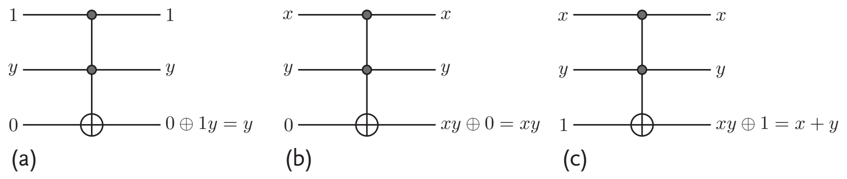

Figure 1.19 (a) Fan-out (two copies of y) simulation (it has to be used for copying gate outputs because a hardware fan-out is not allowed in a reversible circuit); (b) AND implementation; (c) OR implementation which requires four additional CCNOT gates at the control ports.

ble since we cannot generate energy from nowhere (remember that in a reversible circuit electrons/energy "swing"). We can simulate fan-out though, as shown in Figure 1.19a. Classical reversible circuits share the impossibility of having fan-outs with the quantum circuits. The reason why we cannot have a fan-out in a quantum circuit (not even simulated) is the so-called *no-cloning principle*, that is, that a quantum bit cannot be copied. (We shall come back to this principle later on in the book.) Similarly, in reversible circuit, feedbacks (loops) are also not allowed because that would disturb the regularity of "swinging."

In order to implement an OR gate, we have to use either four additional CCNOTs as indicated in Figure 1.19c or a combination of NAND (input target is 1) and NOT (both controls are 1). This is not a problem because CCNOT is universal in a reversible circuit. But, since it is universal neither in the standard binary sense nor in the sense of a quantum universal gate, we shall define the reversible universal gate here [77].

Definition 5 A reversible gate is *r*-universal if and only if any Boolean function $f(x_1, x_2, \dots, x_n)$ can be synthesized by a loop-free and fan-out-free combinatorial network built from a finite number of such gates, using each input $x_1, x_2, \dots, x_n$ at most once and using an arbitrary finite number of times the constant inputs 0 and 1.

Both Fredkin and Toffoli (CCNOT) gates are *r*-universal and are necessary and sufficient for a reversible implementation of arbitrary Boolean function of a finite number of logical variables. Now, in the standard classical circuits fan-outs are allowed and the smallest universal gates (NAND and NOR)[12] are of width 1 and are linear; In quantum circuits fan-outs are not allowed and the smallest universal gates are of width 2 and are linear. What is the smallest logic width of *r*-universal gates. Can we correlate a hardware "no fan-outs" restriction with a software condition? The answer is given by the following theorem.

Theorem 6

A reversible gate is *r*-universal if and only if it is not linear [77].

12) Not only can we express all other operations by means of NAN and NOR, but we can also compress all the conditions of the Boolean algebra in a single axiom [193] and [327, pp. 807,1174]. We can even express all the conditions with the help of a universal operation so that they keep an identical form when we substitute NAND, OR, and so on, for that universal operation [198].

A truth table of a logic gate of width w consists of 2^w lines. A gate is reversible if and only if all 2^w output values form a permutation of all 2^w input values. That makes $(2^w)!$ different reversible gates. Two reversible gates of width 1 and all 24 $[(2^2)! = 24]$ of width 2 are linear. Therefore, the smallest r-universal gate are of width 3. There are 1344 linear of all $(2^3)! = 40\,320$ reversal gates of width 3 that makes 38 976 r-universal gates of width 3. Quantum gates do not have truth tables, and so we will look for another explanation of their properties in the next sections.

This also gives us an answer as to why it is relevant to go into the details of gate algebras. They tell us a great deal about hardware; for binary, reversible, and quantum gates alike.

To sum up, the idea of reversible computers emerged from an energy consideration of the scaling down electronic elements to atomic level in the future. At the same time, the idea of quantum computers emerged from a consideration of how to speed-up computation once the CPU clock hits its quantum limits. The development of both ideas are being developed, although the quantum computers are better funded for an obvious reason – no matter how well we solve the heat dissipation, the classical computer can never have a transistor that would work on less than one electron and can never have conductors thinner than one atom while quantum superposition mimics just that.

There are properties that reversible and quantum gates share. These are the reversibility itself, gate and circuit formalism, unitarity of matrices, universality of gates at a particular level, absence of fan-outs, and functionality at the atom level.[13) There are properties that they do not share. For example, there are neither truth values nor truth tables for quantum gates, only for classical reversible ones. Then, at least for the time being, the reversible circuits are much slower than the standard binary ones, while the quantum ones are exponentially faster than the standard binary ones, at least for particular algorithms.

1.10 Ultimate Efficiency: Quantum Computers and Qubits

In Section 1.7, we have seen that the classical solution to a collapse of the expected exponential increase of the CPU speed is a massive parallelism in both individual PCs and supercomputers. Intel has put forward a new slogan: "Parallelism Full Steam Ahead!" in its new journal *The Parallel Universe* [266].

On the other hand, photons – being quantum systems – inherently possess massive parallelism based on their ability to superpose their states and, as we have

13) The main feature of reversible circuits is their power efficiency which stems from the Landauer principle given by (1.14). However, the Landauer heat given by (1.15) is significant only for single electrons (any technology that relies on many electrons supporting a single bit dissipates the heat that is altogether much higher than the sum of information heats of individual electrons). The need for reversible gates will increase as we continue with the miniaturization of transistors and eventually arrive at single electron ones. Quantum circuits, on the other hand, are bound to an atomic level from the very start since they compute by means of photons, electrons, and atoms.

seen in Section 1.6, we can use this superposition of quantum states for a "physical" quantum computation. The prototype we have described there cannot cope with massive calculations due to the requirement that the optical path difference be smaller than the coherence lengths of the laser. Yet, it *was* a quantum computer with one quantum bit only – serving as a quantum CPU with a single quantum transistor, that is, a single quantum gate – that was capable of factoring numbers with up to 10^{10} digits.

This means that 50 quantum bits, each having two states $|1\rangle$ and $|0\rangle$, would give us a superposition of $2^{50} \approx 10^{15}$ states. Any gate operation on these 50 quantum bits amounts to an interaction with all 2^{50} states in parallel since they are all in collective phonon modes. These quantum bits build a composite Hilbert space $\mathcal{H} = \mathcal{H}^2 \otimes \cdots \otimes \mathcal{H}^2$. The computational basis, that is, the basis of this space, consists of the following 2^{50} vectors: $|00\cdots 00\rangle, |00\cdots 01\rangle, \ldots, |11\cdots 11\rangle$. To compute a function f of each of these states means to let the states evolve according to the time evolution unitary operator U (Schrödinger equation):

$$|i_1 i_2 \ldots i_{50}\rangle \longmapsto U|i_1 i_2 \ldots i_{50}\rangle = |f(i_1, \ldots . i_{50})\rangle \; . \tag{1.19}$$

In a classical computer, we would carry out such a computation in a one-state-at-a-time sequence. In a quantum computer, we first put all the states on the left-hand side of (1.19) in a superposition of all 2^{50} basis states and then let them evolve together and in one step:

$$\sum_{i_1 i_2 \ldots i_{50}=0}^{1} \alpha_{i_1 i_2 \ldots i_{50}} |i_1 i_2 \ldots i_{50}\rangle \overset{f}{\longmapsto} \sum_{i_1, i_2, \ldots, i_{50}=0}^{1} \alpha_{i_1 i_2 \ldots i_{50}} |f(i_1 i_2 \ldots i_{50})\rangle \; . \tag{1.20}$$

After that, we let the obtained (evolved) superposition collapse to a particular state that we read as a result. Of course, since such a collapse of the wave packet is intrinsically statistical, we have to repeat it a number of times, but this procedure is of a polynomial complexity provided that we find a proper function f for a problem we want to calculate.

The difference between this quantum and a binary classical computer consists of the fact that 2^{50} states are formed by only 50 quantum transistors (quantum bits), while in a classical computer, we need a new transistor for each new state, that is, 2^{50} or about one million billion transistors or about half a million of today's most advanced CPUs. More realistic and detailed estimations give about 10^6 quantum bits for such computational power, though [264].

Of course, to be able to use this parallelism we must – as for classical parallel systems – find appropriate quantum hardware and software solutions. Quantum computing power would depend on how well we could correct errors and faults in computation, on how well we could interconnect qubits, and on how efficient the algorithms that we would find for them are. To arrive at each aspect of quantum bits, we have to first learn of their most basic properties and how we can handle them. We shall do that starting from their abstract definition in the Hilbert space, but in a pedestrian approach.

Definition 7 A *qubit* (*qu*antum *bit*) is a two-state quantum system. The two states form a basis in a two-dimensional Hilbert space $\mathcal{H}^2$ and are denoted $|0\rangle$ and $|1\rangle$. They are vectors in $\mathcal{H}^2$, form a basis in $\mathcal{H}^2$, and span $\mathcal{H}^2$. In the matrix representation, they read:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} , \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} . \tag{1.21}$$

In the spin and polarization representation, we have $|\uparrow\rangle = |0\rangle$, $|H\rangle = |0\rangle$ (*spin up, horiziontal polarization*) and $|\downarrow\rangle = |1\rangle$, $|V\rangle = |1\rangle$ (*spin down, vertical polarization*).

Definition 8 Any vector from $\mathcal{H}^n$ denoted as $|\Psi\rangle$ is called a *ket*.

Definition 9 Hilbert space $\mathcal{H} = \mathcal{H}^2 \otimes \ldots \otimes \mathcal{H}^2$ is called a *composite qubit space*. The basis of this space, consisting of vectors $|00\cdots 00\rangle, |00\cdots 01\rangle, \ldots, |11\cdots 11\rangle$ is called the *computational basis*.

Qubits might be linear or circular polarization states of photons, two levels of an atom or ion, spin-1/2 nuclear states in a magnetic field (nuclear magnetic resonance, NMR), electron and nuclear states in a silicon, electron states in an electron dot, then charge, flux, phase, and charge-flux states in superconducting devices, and so on.

We shall often measure a state of a qubit so as to let it pass a filter which lets a qubit in a particular state through. In that case the qubit will not be distorted by passing through the filter. Such a filter is called a *perfect filter* and an *ideal measurement*. In quantum computation, filtering (perfect and imperfect) is often used for preparing and handling states of a qubit and we can consider the qubit that exited a specified port of a filter to be measured, although it has not been destroyed by – we say, *collapsed* in – a measurement device.

When we do not deal with a perfect filtering, then the state is distorted and changed by filtering. This can be statistically described and the outcome can be statistically predicted, but an outcome of an individual measurement will in the latter case remain random and unpredictable.

This inherent randomness in triggering a measuring device or passing through a filter whenever their setups do not match the state in which the qubit was prepared is one of the main features of a qubit. For example, if a qubit is prepared in a spin-up state oriented along the $+Oz$-axis as shown in Figure 1.20, then it will always pass through an s_z filter, but it will pass through $+Ox$ and $+Oy$ only every second time. We can verify similar behaviors with polarized photons. A photon prepared by a polarizer will pass through another polarizer oriented in the same direction.

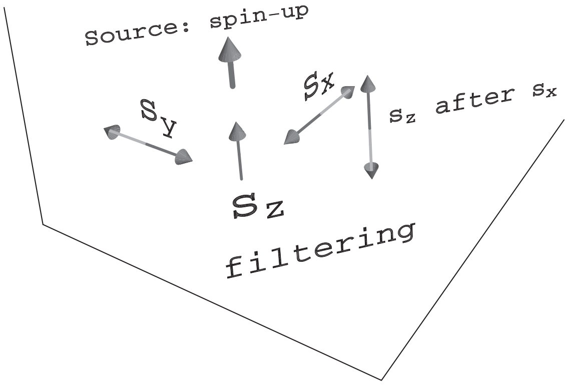

Figure 1.20 A qubit prepared in a spin-up state along $+Oz$ will always pass a $+s_z$ filter, but will pass (at random) $+s_x$ and $+s_y$ only in 50% of verifications (it will pass $-s_x$ and $-s_y$ in the other 50% of verifications). After passing s_x filters, the qubit will pass a new $+s_z$ filter only in half of the cases.

However, a polarizer rotated by 45° will only let every second photon through.

The probability of a photon passing the second polarizer oriented in the same direction as the first one is 1. The probability of a photon passing through the second polarizer rotated at an angle ϕ with respect to the first one is

$$P_\phi = \cos^2 \phi \ . \tag{1.22}$$

This is the so-called *Malus law*.[14] It gives meaning to the "statistical description" we mentioned above. For a particular photon, there is no way of predicting whether it will pass the polarizer or not, but the probability of passing it is $\cos^2 \phi$ for every one of them.

The randomness will make the algorithms for quantum computation rather demanding, but on the other hand, it will make messages coded by a quantum cryptography protocol unbreakable and eavesdropping impossible.

1.11
Combining and Measuring Qubits: Quantum Superposition – Qubit Primer

Superposition of qubits enables exponential speed-up of would-be quantum computation, as we explained in the previous section. It also determines the way in which we describe interaction of qubits, their manipulation, and their measurement. In Section 1.6, we introduced a superposition of photons in a Mach–Zehnder interferometer. In a quantum computer, we, however, expect to deal with qubits that are parts of atoms or ions and therefore we shall introduce a superposition by considering atom levels of a rubidium atom ^{87}Rb in a cavity as shown in Figure 1.21.

14) Of, say, 100 photons, $100\cos^2\varphi$ will pass the filter and $100\sin^2\varphi$ will not.